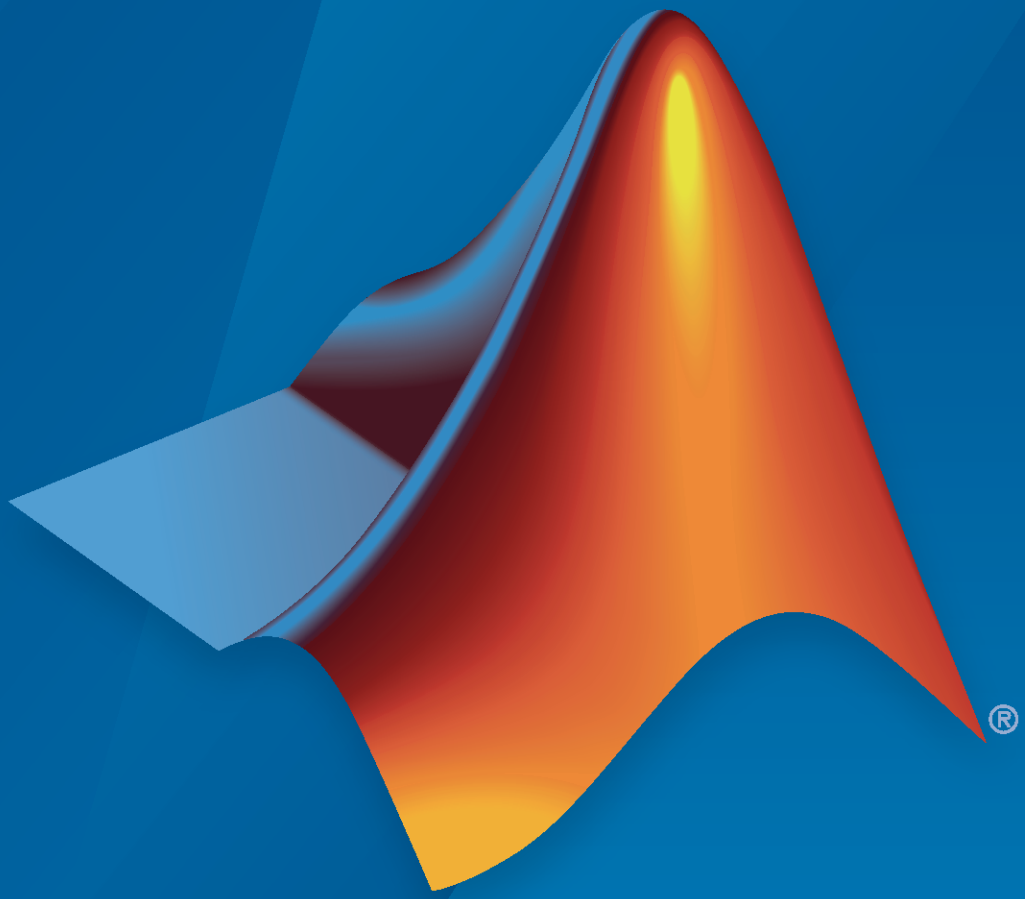
Bluetooth® Toolbox

User's Guide



MATLAB®

R2022a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Bluetooth® Toolbox User's Guide

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

March 2022	Online only	New for Version 1.0 (Release 2022a)

Contents

1 PHY Modeling

2 Coexistence Modeling

3 Localization

4 Test and Measurement

7 Code Generation and Deployment

8 Bluetooth Topic Pages

9 Tutorials

PHY Modeling

Bluetooth LE Waveform Reception Using SDR

This example shows how to implement a Bluetooth® Low Energy (LE) receiver using the Bluetooth® Toolbox. You can either use captured signals or receive signals in real time using the ADALM-PLUTO Radio. A suitable signal for reception can be generated by simulating the companion example, “Bluetooth LE Waveform Generation and Transmission Using SDR” on page 1-8, with any one of the following setup: (i) Two SDR platforms connected to the same host computer which runs two MATLAB sessions (ii) Two SDR platforms connected to two computers which run separate MATLAB sessions.

Refer to the “Guided Host-Radio Hardware Setup” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio) documentation for details on how to configure your host computer to work with the Support Package for ADALM-PLUTO Radio.

Required Hardware and Software

To receive signals in real time, you need an ADALM-PLUTO radio and the corresponding support package Add-On:

- Communications Toolbox Support Package for ADALM-PLUTO Radio

For a full list of Communications Toolbox supported SDR platforms, refer to Supported Hardware section of the Software Defined Radio (SDR) discovery page.

Background

The Bluetooth Special Interest Group (SIG) introduced LE for low power short range communications. The Bluetooth standard [3] specifies the **Link** layer which includes both **PHY** and **MAC** layers. Bluetooth LE applications include image and video file transfers between mobile phones, home automation, and the Internet of Things (IoT).

Specifications of Bluetooth LE:

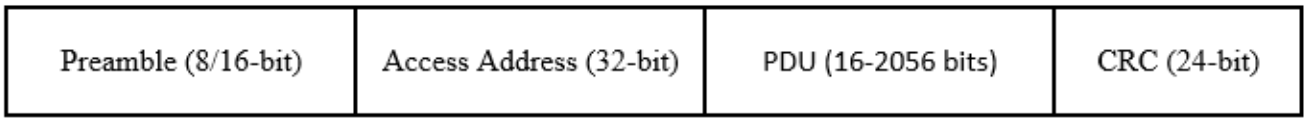
- **Transmission frequency range**: 2.4-2.4835 GHz
- **RF channels** : 40
- **Symbol rate** : 1 Msym/s, 2 Msym/s
- **Modulation** : Gaussian Minimum Shift Keying (GMSK)
- **PHY transmission modes** : (i) LE1M - Uncoded PHY with data rate of 1 Mbps (ii) LE2M - Uncoded PHY with data rate of 2 Mbps (iii) LE500K - Coded PHY with data rate of 500 Kbps (iv) LE125K - Coded PHY with data rate of 125 Kbps

The Bluetooth standard [3] specifies air interface packet formats for all the four PHY transmission modes of Bluetooth LE using the following fields:

- **Preamble**: The preamble depends on PHY transmission mode. LE1M mode uses an 8-bit sequence of alternate zeros and ones, '01010101'. LE2M uses a 16-bit sequence of alternate zeros and ones, '0101...'. LE500K and LE125K modes use an 80-bit sequence of zeros and ones obtained by repeating '00111100' ten times.
- **Access Address**: Specifies the connection address shared between two Bluetooth LE devices using a 32-bit sequence.
- **Coding Indicator**: 2-bit sequence used for differentiating coded modes (LE125K and LE500K).

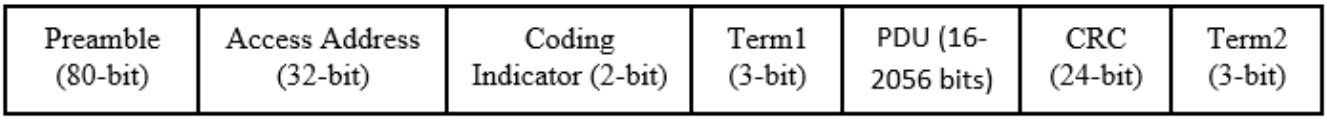
- **Payload**: Input message bits including both protocol data unit (PDU) and cyclic redundancy check (CRC). The maximum message size is 2080 bits.
- **Termination Fields**: Two 3-bit vectors of zeros, used in forward error correction encoding. The termination fields are present for coded modes (LE500K and LE125K) only.

Packet format for uncoded PHY (LE1M and LE2M) modes is shown in the figure below:



Bluetooth LE Uncoded PHY Packet Format

Packet format for coded PHY (LE500K and LE125K) modes is shown in the figure below:



Bluetooth LE Coded PHY Packet Format

Example Structure

The general structure of the Bluetooth LE receiver example is described as follows:

1. Initialize the receiver parameters
2. Signal source
3. Capture the Bluetooth LE packets
4. Receiver processing

Initialize the Receiver Parameters

The helperBLEReceiverConfig.m script initializes the receiver parameters. You can change `phyMode` parameter to decode the received Bluetooth LE waveform based on the PHY transmission mode. `phyMode` can be one from the set: {'LE1M','LE2M','LE500K','LE125K'}.

```
phyMode = 'LE1M';
bleParam = helperBLEReceiverConfig(phyMode);
```

Signal Source

Specify the signal source as 'File' or 'ADALM-PLUTO'.

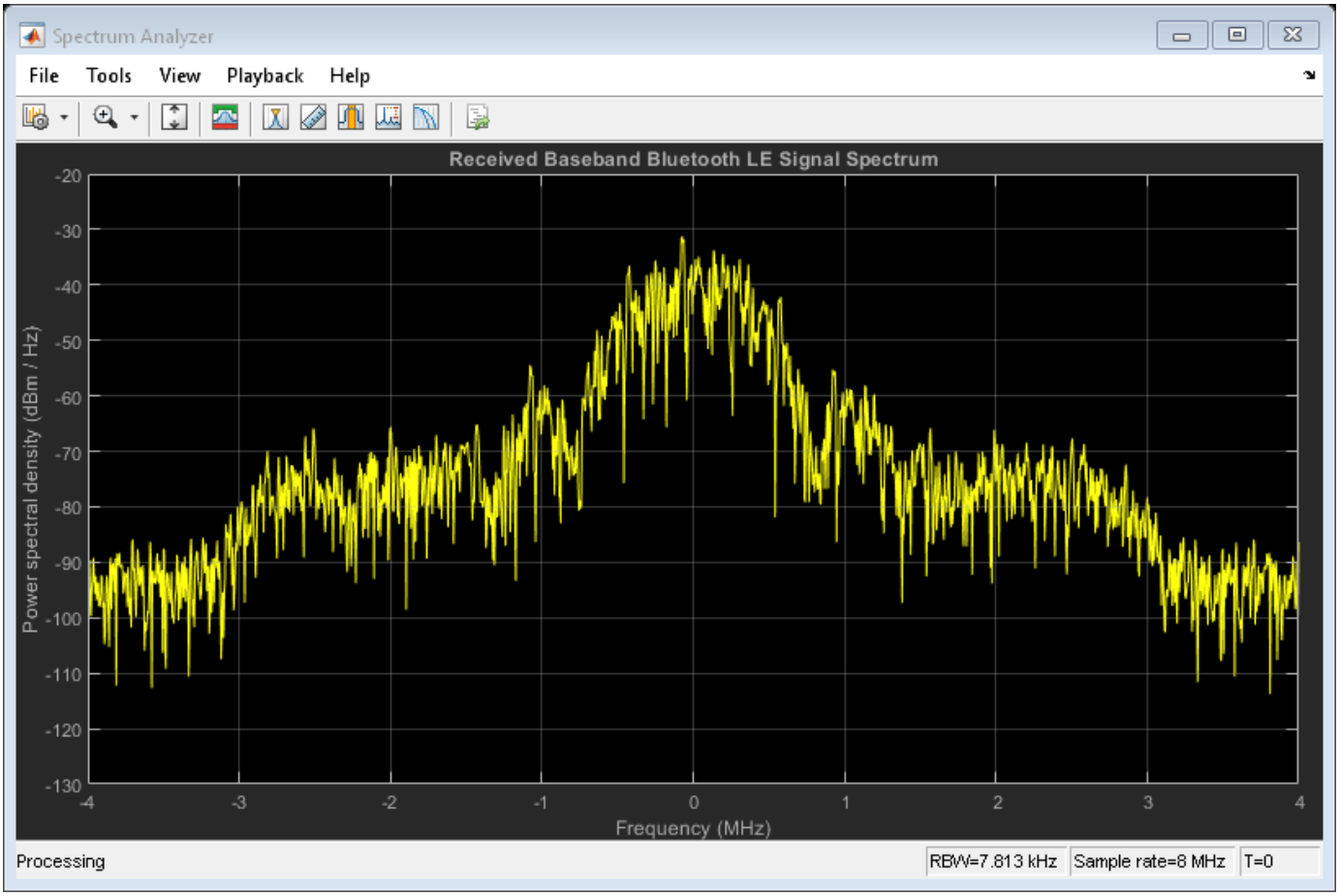
- **File**:Uses the `comm.BasebandFileReader` to read a file that contains a previously captured over-the-air signal.
- **ADALM-PLUTO**: Uses the `sdrrx` (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio) System object to receive a live signal from the SDR hardware.

If you assign ADALM-PLUTO as the signal source, the example searches your computer for the ADALM-PLUTO radio at radio address 'usb:0' and uses it as the signal source.

```
signalSource = 'File'; % The default signal source is 'File'

if strcmp(signalSource,'File')
    switch bleParam.Mode
        case 'LE1M'
            bbFileName = 'bleCapturesLE1M.bb';
        case 'LE2M'
            bbFileName = 'bleCapturesLE2M.bb';
        case 'LE500K'
            bbFileName = 'bleCapturesLE500K.bb';
        case 'LE125K'
            bbFileName = 'bleCapturesLE125K.bb';
        otherwise
            error('Invalid PHY transmission mode. Valid entries are LE1M, LE2M, LE500K and LE125K
    end
    sigSrc = comm.BasebandFileReader(bbFileName);
    sigSrcInfo = info(sigSrc);
    sigSrc.SamplesPerFrame = sigSrcInfo.NumSamplesInData;
    bbSampleRate = sigSrc.SampleRate;
    bleParam.SamplesPerSymbol = bbSampleRate/bleParam.SymbolRate;

elseif strcmp(signalSource,'ADALM-PLUTO')

    % First check if the HSP exists
    if isempty(which('plutoradio.internal.getRootDir'))
        error(message('comm_demos:common:NoSupportPackage', ...
            'Communications Toolbox Support Package for ADALM-PLUTO Radio',...
            ['<a href="https://www.mathworks.com/hardware-support/' ...
            'adalm-pluto-radio.html">ADALM-PLUTO Radio Support From Communications Toolbox</a>']))
    end

    bbSampleRate = bleParam.SymbolRate * bleParam.SamplesPerSymbol;
    sigSrc = sdrrx('Pluto',...
        'RadioID',             'usb:0',...
        'CenterFrequency',     2.402e9,...
        'BasebandSampleRate',  bbSampleRate,...
        'SamplesPerFrame',     1e7,...
        'GainSource',         'Manual',...
        'Gain',                25,...
        'OutputDataType',     'double');
else
    error('Invalid signal source. Valid entries are File and ADALM-PLUTO.');
end

% Setup spectrum viewer
spectrumScope = dsp.SpectrumAnalyzer( ...
    'SampleRate',       bbSampleRate,...
    'SpectrumType',     'Power density', ...
    'SpectralAverages', 10, ...
    'YLimits',          [-130 -20], ...
    'Title',            'Received Baseband Bluetooth LE Signal Spectrum', ...
    'YLabel',           'Power spectral density');
```

Capture the Bluetooth LE Packets

```
% The transmitted waveform is captured as a burst
dataCaptures = sigSrc();

% Show power spectral density of the received waveform
spectrumScope(dataCaptures);
```



Receiver Processing

The baseband samples received from the signal source are processed to decode the PDU header information and raw message bits. The following diagram shows the receiver processing.

1 Perform automatic gain control (AGC)

2 Remove DC offset
3 Estimate and correct for the carrier frequency offset
4 Perform matched filtering with gaussian pulse
5 Timing synchronization
6 GMSK demodulation
7 FEC decoding and pattern demapping for LECoded PHYs (LE500K and LE125K)
8 Data dewhitening
9 Perform CRC check on the decoded PDU
10 Compute packet error rate (PER)

```
% Initialize System objects for receiver processing
agc = comm.AGC('MaxPowerGain',20,'DesiredOutputPower',2);

freqCompensator = comm.CoarseFrequencyCompensator('Modulation','OQPSK', ...
    'SampleRate',bbSampleRate,...
    'SamplesPerSymbol',2*bleParam.SamplesPerSymbol,...
    'FrequencyResolution',100);

prbDet = comm.PreambleDetector(bleParam.RefSeq,'Detections','First');

% Initialize counter variables
pktCnt = 0;
crcCnt = 0;
displayFlag = false; % true if the received data is to be printed

% Loop to decode the captured Bluetooth LE samples
while length(dataCaptures) > bleParam.MinimumPacketLen

    % Consider two frames from the captured signal for each iteration
    startIndex = 1;
    endIndex = min(length(dataCaptures),2*bleParam.FrameLength);
    rcvSig = dataCaptures(startIndex:endIndex);

    rcvAGC = agc(rcvSig); % Perform AGC
    rcvDCFree = rcvAGC - mean(rcvAGC); % Remove the DC offset
    rcvFreqComp = freqCompensator(rcvDCFree); % Estimate and compensate for the carrier frequency
    rcvFilt = conv(rcvFreqComp,bleParam.h,'same'); % Perform gaussian matched filtering

    % Perform frame timing synchronization
    [~, dtMt] = prbDet(rcvFilt);
    release(prbDet)
    prbDet.Threshold = max(dtMt);
    prbIdx = prbDet(rcvFilt);

    % Extract message information
    [cfgLLAdv,pktCnt,crcCnt,remStartIdx] = helperBLEPhyBitRecover(rcvFilt,...
        prbIdx,pktCnt,crcCnt,bleParam);

    % Remaining signal in the burst captures
    dataCaptures = dataCaptures(1+remStartIdx:end);

    % Display the decoded information
    if displayFlag && ~isempty(cfgLLAdv)
        fprintf('Advertising PDU Type: %s\n',cfgLLAdv.PDUType);
```

```
        fprintf('Advertising Address: %s\n',cfgLLAdv.AdvertiserAddress);
    end

    % Release System objects
    release(freqCompensator)
    release(prbDet)
end

% Release the signal source
release(sigSrc)

% Determine the PER
if pktCnt
    per = 1-(crcCnt/pktCnt);
    fprintf('Packet error rate for %s mode is %f.\n',bleParam.Mode,per);
else
    fprintf('\n No Bluetooth LE packets were detected.\n')
end
```

```
Packet error rate for LE1M mode is 0.000000.
```

Further Exploration

The companion example “Bluetooth LE Waveform Generation and Transmission Using SDR” on page 1-8 can be used to transmit a standard-compliant LE waveform which can be decoded by this example. You can also use this example to transmit the data channel PDUs by changing channel index, access address and center frequency values in both the examples.

Troubleshooting

General tips for troubleshooting SDR hardware and the Communications Toolbox Support Package for ADALM-PLUTO Radio can be found in “Common Problems and Fixes” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio).

Appendix

This example uses these helper functions:

* helperBLEReceiverConfig.m: Configures Bluetooth LE receiver parameters

* helperBLEPhyBitRecover.m: Recovers the payload bits

Selected Bibliography

1 Bluetooth Technology Website | The Official Website of Bluetooth Technology, Accessed November 22, 2021. https://www.bluetooth.com. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". Version 5.3, Volume 6. https://www.bluetooth.com.

See Also

More About

- “Bluetooth LE Waveform Generation and Transmission Using SDR” on page 1-8
- “Bluetooth LE Waveform Generation and Visualization” on page 1-14

Bluetooth LE Waveform Generation and Transmission Using SDR

This example shows how to implement a Bluetooth® Low Energy (LE) transmitter using the Bluetooth® Library. You can either transmit Bluetooth LE signals using the ADALM-PLUTO radio or write to a baseband file (*.bb). The transmitted Bluetooth LE signal can be received by the companion example, “Bluetooth LE Waveform Reception Using SDR” on page 1-2, with any one of the following setup: (i) Two SDR platforms connected to the same host computer which runs two MATLAB sessions (ii) Two SDR platforms connected to two computers which run separate MATLAB sessions.

Refer to the “Guided Host-Radio Hardware Setup” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio) documentation for details on how to configure your host computer to work with the Support Package for ADALM-PLUTO Radio.

Required Hardware and Software

To transmit signals in real time, you need ADALM-PLUTO radio and the corresponding support package Add-On:

- Communications Toolbox Support Package for ADALM-PLUTO Radio

For a full list of Communications Toolbox supported SDR platforms, refer to Supported Hardware section of the Software Defined Radio (SDR) discovery page.

Background

The Bluetooth Special Interest Group (SIG) introduced Bluetooth LE for low power short range communications. The Bluetooth standard [3] specifies the **Link** layer which includes both **PHY** and **MAC** layers. Bluetooth LE applications include image and video file transfers between mobile phones, home automation, and the Internet of Things (IoT).

Specifications of Bluetooth LE:

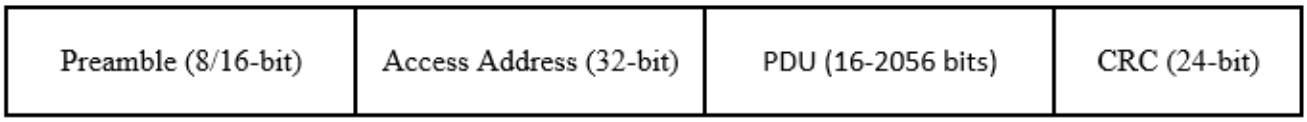
- **Transmission frequency range**: 2.4-2.4835 GHz
- **RF channels** : 40
- **Symbol rate** : 1 Msym/s, 2 Msym/s
- **Modulation** : Gaussian Minimum Shift Keying (GMSK)
- **PHY transmission modes** : (i) LE1M - Uncoded PHY with data rate of 1 Mbps (ii) LE2M - Uncoded PHY with data rate of 2 Mbps (iii) LE500K - Coded PHY with data rate of 500 Kbps (iv) LE125K - Coded PHY with data rate of 125 Kbps

The Bluetooth standard [3] specifies air interface packet formats for all the four PHY transmission modes of Bluetooth LE using the following fields:

- **Preamble**: The preamble depends on PHY transmission mode. LE1M mode uses an 8-bit sequence of alternate zeros and ones, '01010101'. LE2M uses a 16-bit sequence of alternate zeros and ones, '0101...'. LE500K and LE125K modes use an 80-bit sequence of zeros and ones obtained by repeating '00111100' ten times.
- **Access Address**: Specifies the connection address shared between two Bluetooth LE devices using a 32-bit sequence.
- **Coding Indicator**: 2-bit sequence used for differentiating coded modes (LE125K and LE500K).

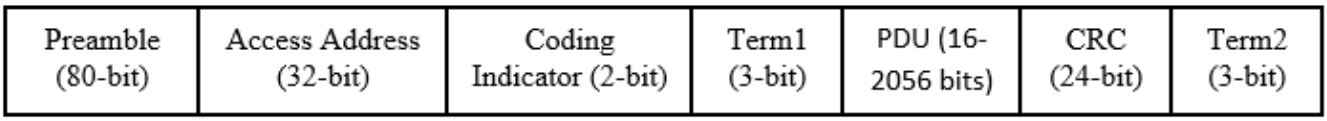
- **Payload**: Input message bits including both protocol data unit (PDU) and cyclic redundancy check (CRC). The maximum message size is 2080 bits.
- **Termination Fields**: Two 3-bit vectors of zeros, used in forward error correction encoding. The termination fields are present for coded modes (LE500K and LE125K) only.

Packet format for uncoded PHY (LE1M and LE2M) modes is shown in the figure below:



Bluetooth LE Uncoded PHY Packet Format

Packet format for coded PHY (LE500K and LE125K) modes is shown in the figure below:



Bluetooth LE Coded PHY Packet Format

Example Structure

The general structure of the Bluetooth LE transmitter example is described as follows:

1. Generate link layer PDUs
2. Generate baseband IQ waveforms
3. Transmitter processing

Generate Link Layer PDUs

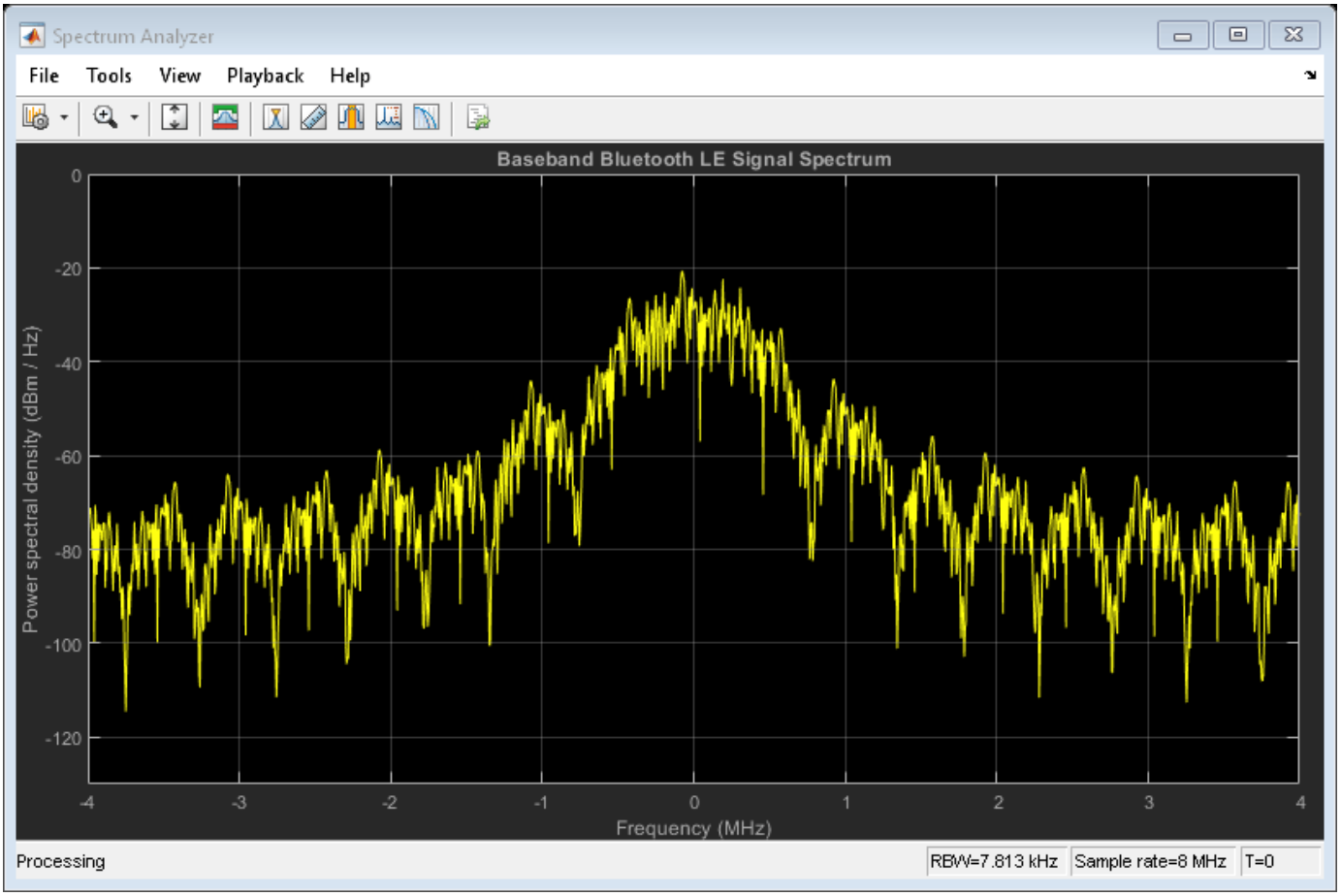
Link layer PDUs can be either advertising channel PDUs or data channel PDUs. You can configure and generate advertising channel PDUs using `bleLLAdvertisingChannelPDUConfig` and `bleLLAdvertisingChannelPDU` functions respectively. You can configure and generate data channel PDUs using `bleLLDataChannelPDUConfig` and `bleLLAdvertisingChannelPDUDecode` functions respectively.

```
% Configure an advertising channel PDU
cfgLLAdv = bleLLAdvertisingChannelPDUConfig;
cfgLLAdv.PDUType         = 'Advertising indication';
cfgLLAdv.AdvertisingData = '0123456789ABCDEF';
cfgLLAdv.AdvertiserAddress = '1234567890AB';

% Generate an advertising channel PDU
messageBits = bleLLAdvertisingChannelPDU(cfgLLAdv);
```

Generate Baseband IQ Waveforms

You can use the `bleWaveformGenerator` function to generate standard-compliant waveforms.

```
phyMode = 'LE1M'; % Select one mode from the set {'LE1M','LE2M','LE500K','LE125K'}
sps = 8;          % Samples per symbol
channelIndex = 37;  % Channel index value in the range [0,39]
accessAddressLen = 32;% Length of access address
accessAddressHex = '8E89BED6';  % Access address value in hexadecimal
accessAddressBin = int2bit(hex2dec(accessAddressHex),accessAddressLen,false); % Access address in

% Symbol rate based on |'Mode'|
symbolRate = 1e6;
if strcmp(phyMode,'LE2M')
    symbolRate = 2e6;
end

% Generate Bluetooth LE waveform
txWaveform = bleWaveformGenerator(messageBits,...
    'Mode',            phyMode,...
    'SamplesPerSymbol',sps,...
    'ChannelIndex',    channelIndex,...
    'AccessAddress',   accessAddressBin);

% Setup spectrum viewer
spectrumScope = dsp.SpectrumAnalyzer( ...
    'SampleRate',       symbolRate*sps,...
    'SpectrumType',     'Power density', ...
    'SpectralAverages', 10, ...
    'YLimits',          [-130 0], ...
    'Title',            'Baseband Bluetooth LE Signal Spectrum', ...
    'YLabel',           'Power spectral density');

% Show power spectral density of the Bluetooth LE signal
spectrumScope(txWaveform);
```



Transmitter Processing

Specify the signal sink as 'File' or 'ADALM-PLUTO'.

- **File**:Uses the `comm.BasebandFileWriter` to write a baseband file.
- **ADALM-PLUTO**: Uses the `sdrtx` (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio) System object to transmit a live signal from the SDR hardware.

```
% Initialize the parameters required for signal source
txCenterFrequency       = 2.402e9;  % Varies based on channel index value
txFrameLength           = length(txWaveform);
txNumberOfFrames        = 1e4;
txFrontEndSampleRate    = symbolRate*sps;

% The default signal source is 'File'
signalSink = 'File';

if strcmp(signalSink,'File')

    sigSink = comm.BasebandFileWriter('CenterFrequency',txCenterFrequency,...
        'Filename','bleCaptures.bb',...
        'SampleRate',txFrontEndSampleRate);
    sigSink(txWaveform); % Writing to a baseband file 'bleCaptures.bb'
```

```
elseif strcmp(signalSink,'ADALM-PLUTO')

    % First check if the HSP exists
    if isempty(which('plutoradio.internal.getRootDir'))
        error(message('comm_demos:common:NoSupportPackage', ...
                      'Communications Toolbox Support Package for ADALM-PLUTO Radio',...
                      ['<a href="https://www.mathworks.com/hardware-support/' ...
                      'adalm-pluto-radio.html">ADALM-PLUTO Radio Support From Communications Too
    end
    connectedRadios = findPlutoRadio; % Discover ADALM-PLUTO radio(s) connected to your computer
    radioID = connectedRadios(1).RadioID;
    sigSink = sdrtx( 'Pluto',...
        'RadioID',           radioID,...
        'CenterFrequency',   txCenterFrequency,...
        'Gain',              0,...
        'SamplesPerFrame',   txFrameLength,...
        'BasebandSampleRate',txFrontEndSampleRate);
    % The transfer of baseband data to the SDR hardware is enclosed in a
    % try/catch block. This means that if an error occurs during the
    % transmission, the hardware resources used by the SDR System
    % object(TM) are released.
    currentFrame = 1;
    try
        while currentFrame <= txNumberOfFrames
            % Data transmission
            sigSink(txWaveform);
            % Update the counter
            currentFrame = currentFrame + 1;
        end
    catch ME
        release(sigSink);
        rethrow(ME)
    end
else
    error('Invalid signal sink. Valid entries are File and ADALM-PLUTO.');
end

% Release the signal sink
release(sigSink)
```

Further Exploration

The companion example “Bluetooth LE Waveform Reception Using SDR” on page 1-2 can be used to decode the waveform transmitted by this example. You can also use this example to transmit the data channel PDUs by changing channel index, access address and center frequency values in both the examples.

Troubleshooting

General tips for troubleshooting SDR hardware and the Communications Toolbox Support Package for ADALM-PLUTO Radio can be found in “Common Problems and Fixes” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio).

Selected Bibliography

1 Bluetooth Technology Website | The Official Website of Bluetooth Technology, Accessed November 22, 2021. https://www.bluetooth.com.

2 Volume 6 of the Bluetooth Core Specification, Version 5.3 Core System Package [Low Energy Controller Volume].

See Also

More About

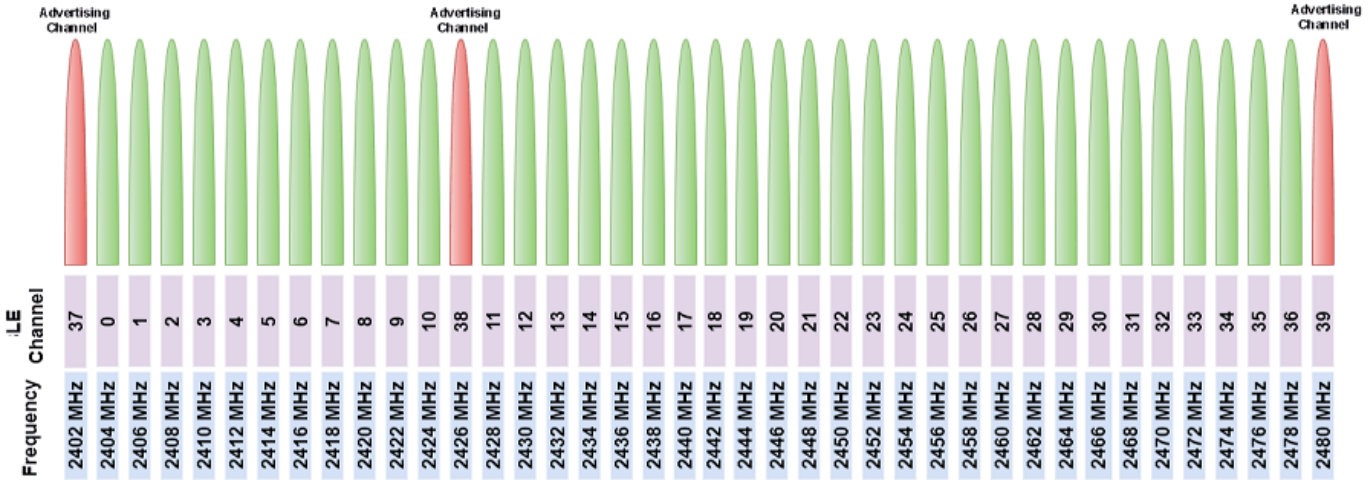
- “Bluetooth LE Waveform Reception Using SDR” on page 1-2
- “Bluetooth LE Waveform Generation and Visualization” on page 1-14

Bluetooth LE Waveform Generation and Visualization

This example shows you how to generate and visualize Bluetooth low energy (LE) waveforms for different physical layer (PHY) modes by using Bluetooth® Toolbox [2].

Background

Bluetooth Special Interest Group (SIG) introduced Bluetooth LE for low power short range communications. Bluetooth LE devices operate in the globally unlicensed industrial, scientific and medical (ISM) band in the frequency range 2.4 GHz to 2.485 GHz. Bluetooth LE specifies a channel spacing of 2 MHz, which results in 40 RF channels as shown in the figure below. The Bluetooth LE standard [2] specifies the **Link** layer which includes both **PHY** and **MAC** layers. Bluetooth LE finds applications in transfer of files such as images and MP3 between mobile phones, home automation and internet of things (IoT) trend.



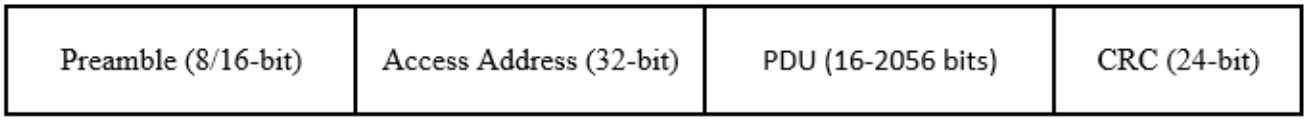
The Bluetooth standard [2] specifies the following physical layer modes:

- **LE1M** - Uncoded PHY with data rate of 1 Mbps
- **LE2M** - Uncoded PHY with data rate of 2 Mbps
- **LE500K** - Coded PHY with data rate of 500 Kbps
- **LE125K** - Coded PHY with data rate of 125 Kbps

The air interface packet formats for these modes include the following fields:

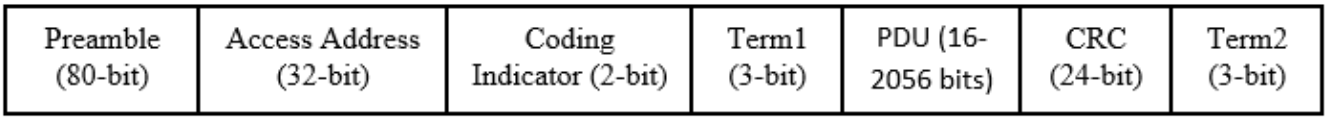
- **Preamble**: The preamble depends on which PHY mode is used. LE1M mode uses an 8-bit sequence of alternate zeros and ones, '01010101'. LE2M uses a 16-bit sequence of alternate zeros and ones, '0101...'. LE500K and LE125K modes use an 80-bit sequence of zeros and ones obtained by repeating '00111100' ten times.
- **Access Address**: Specifies the connection address shared between two Bluetooth LE devices using a 32-bit sequence.
- **Coding Indicator**: 2-bit sequence used for differentiating two coded modes (LE125K, LE500K).
- **Payload**: Input message bits including both PDU and CRC. The maximum message size is 2080 bits.
- **Termination Fields**: Two 3-bit vectors of zeros, used in Forward Error Correction encoding. The termination fields are present for coded modes (LE500K and LE125K) only.

Packet format for uncoded PHY (LE1M and LE2M) modes is shown in the figure below:



Bluetooth LE Uncoded PHY Packet Format

Packet format for coded PHY (LE500K and LE125K) modes is shown in the figure below:



Bluetooth LE Coded PHY Packet Format

Introduction

This example shows how to generate Bluetooth LE waveforms for all the physical layer modes as per the Bluetooth specification [2]. The generated Bluetooth LE waveforms are visualized in both time-domain and frequency-domain using time scope and spectrum analyzer respectively.

Initialize Parameters for Waveform Generation

```
% Specify the input parameters for generating Bluetooth LE waveform
numPackets = 10;    % Number of packets to generate
sps = 16;           % Samples per symbol
messageLen = 2000;  % Length of message in bits
phyMode = 'LE1M';   % Select one mode from the set {'LE1M','LE2M','LE500K','LE125K'};
channelBW = 2e6;    % Channel spacing (Hz) as per standard
% Define symbol rate based on the PHY mode
if any(strcmp(phyMode,{'LE1M','LE500K','LE125K'}))
    symbolRate = 1e6;
else
    symbolRate = 2e6;
end
```

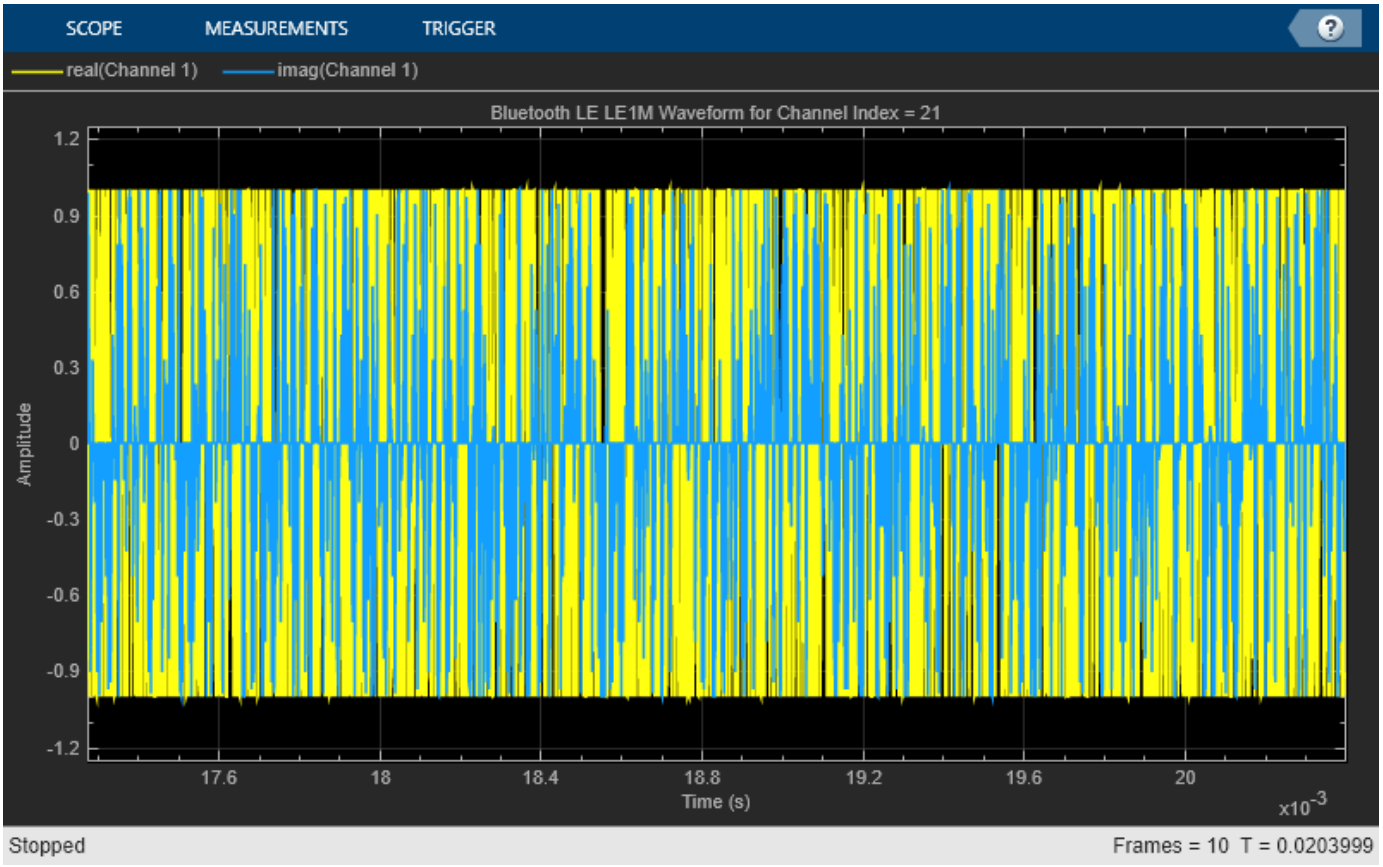
Create Objects for Visualization

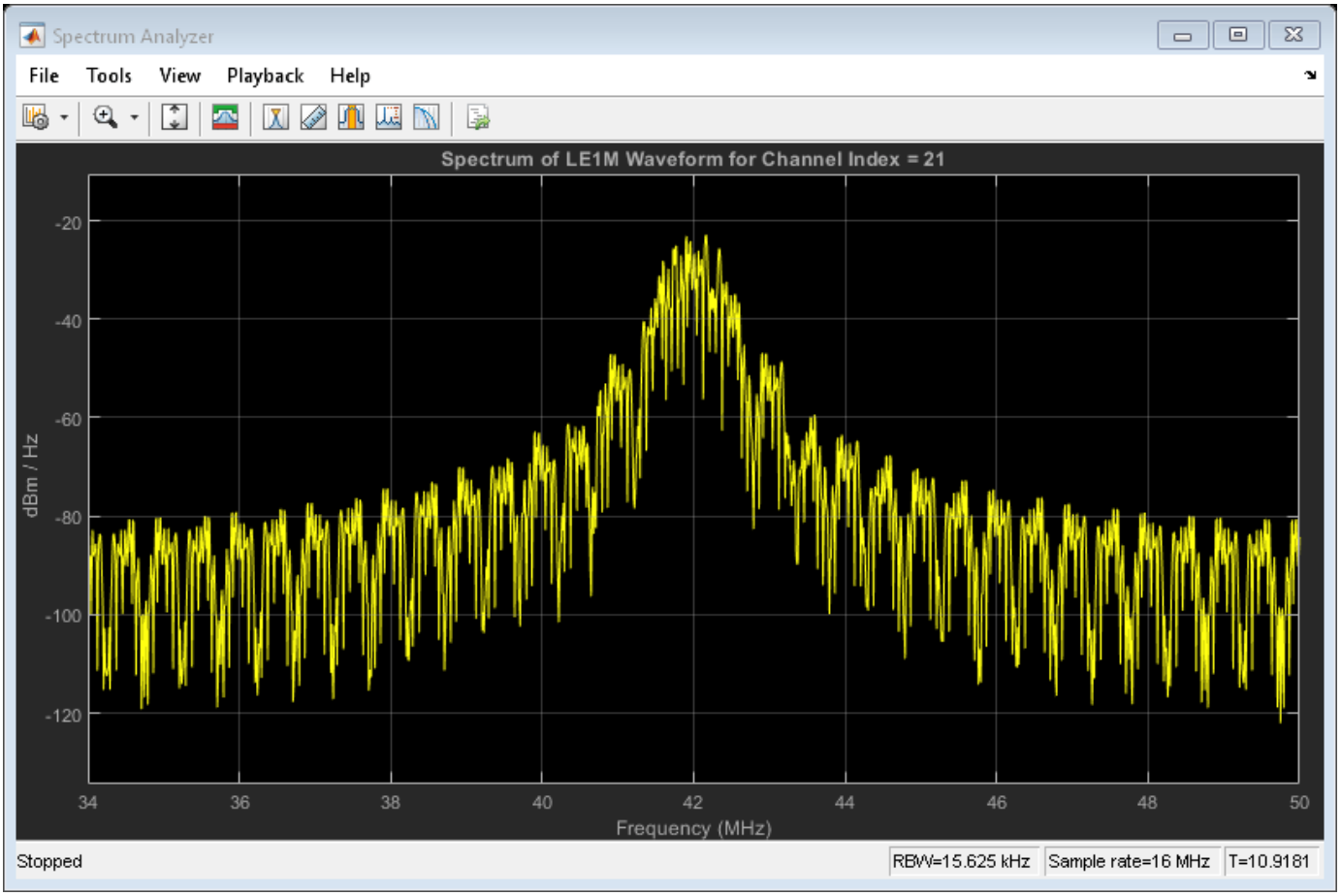
```
% Create a spectrum analyzer object
specAn = dsp.SpectrumAnalyzer('SpectrumType','Power density');
specAn.SampleRate = symbolRate*sps;

% Create a time scope object
timeScope = timescope('SampleRate', symbolRate*sps,'TimeSpanSource','Auto',...
     'ShowLegend',true);
```

Waveform Generation and Visualization

```
% Loop over the number of packets, generating a Bluetooth LE waveform and
% plotting the waveform spectrum
```

```
rng default;
for packetIdx = 1:numPackets
    message = randi([0 1],messageLen,1);    % Message bits generation
    channelIndex = randi([0 39],1,1);          % Channel index decimal value

    if(channelIndex >=37)
        % Default access address for periodic advertising channels
        accessAddress = [0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 ...
                         1 0 0 0 1 0 1 1 1 0 0 0 1]';
    else
        % Random access address for data channels
        % Ideally, this access address value should meet the requirements
        % specified in Section 2.1.2 of volume 6 of the Bluetooth Core
        % Specification.
        accessAddress = [0 0 0 0 0 0 0 1 0 0 1 0 0 ...
            0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1]';
    end

    waveform = bleWaveformGenerator(message,...
                                    'Mode',phyMode,...
                                    'SamplesPerSymbol',sps,...
                                    'ChannelIndex',channelIndex,...
                                    'AccessAddress',accessAddress);

    specAn.FrequencyOffset = channelBW*channelIndex;
    specAn.Title = ['Spectrum of ',phyMode,' Waveform for Channel Index = ', num2str(channelIndex

    tic
    while toc < 0.5 % To hold the spectrum for 0.5 seconds
        specAn(waveform);
    end

    % Plot the generated waveform
    timeScope.Title = ['Bluetooth LE ',phyMode,' Waveform for Channel Index = ', num2str(channel
    timeScope(waveform);
end

% Release objects
release(specAn);
release(timeScope);
```


SCOPE
MEASUREMENTS
TRIGGER
real(Channel 1)
imag(Channel 1)
Bluetooth LE LE1M Waveform for Channel Index = 21
1.2
0.9
0.6
0.3
0
-0.3
-0.6
-0.9
-1.2
Amplitude
17.6
18
18.4
18.8
19.2
19.6
20
Time (s)
x10^-3
Stopped
Frames = 10 T = 0.0203999



Selected Bibliography

1 Bluetooth Technology Website | The Official Website of Bluetooth Technology, Accessed November 22, 2021. https://www.bluetooth.com.

2 Volume 6 of the Bluetooth Core Specification, Version 5.3 Core System Package [Low Energy Controller Volume].

See Also

More About

- “Bluetooth LE Waveform Generation and Transmission Using SDR” on page 1-8
- “Bluetooth LE Waveform Reception Using SDR” on page 1-2

Bluetooth LE Bit Error Rate Simulation with AWGN

This example shows how to measure the bit error rate (BER) for different modes of Bluetooth Low Energy (LE) [3] using an end-to-end physical layer simulation by using the Bluetooth® Toolbox.

Introduction

In this example, an end-to-end simulation is used to determine the BER performance of Bluetooth LE [3] under an additive white gaussian noise (AWGN) channel for a range of bit energy to noise density ratio (Eb/No) values. At each Eb/No point, multiple Bluetooth LE packets are transmitted through a noisy channel with no other radio front-end (RF) impairments. Assuming perfect synchronization, an ideal receiver is used to recover the data bits. These recovered data bits are compared with the transmitted data bits to determine the BER. BER curves are generated for the four PHY transmission throughput modes supported in Bluetooth LE specification [3] as follows:

- Uncoded PHY with data rate of 1 Mbps (LE1M)
- Uncoded PHY with data rate of 2 Mbps (LE2M)
- Coded PHY with data rate of 500 Kbps (LE500K)
- Coded PHY with data rate of 125 Kbps (LE125K)

The following diagram summarizes the simulation for each packet.

Initialize the Simulation Parameters

```
EbNo = -2:2:8;                          % Eb/No range in dB
sps = 4;                                % Samples per symbol
dataLen = 2080;                         % Data length in bits
simMode = {'LE1M','LE2M','LE500K','LE125K'};
```

The number of packets tested at each Eb/No point is controlled by two parameters:

1. `maxNumErrors` is the maximum number of bit errors simulated at each Eb/No point. When the number of bit errors reaches this limit, the simulation at this Eb/No point is complete.
2. `maxNumPackets` is the maximum number of packets simulated at each Eb/No point and limits the length of the simulation if the bit error limit is not reached.

The numbers chosen for `maxNumErrors` and `maxNumPackets` in this example will lead to a very short simulation. For statistically meaningful results we recommend increasing these numbers.

```
maxNumErrors = 100; % Maximum number of bit errors at an Eb/No point
maxNumPackets = 10; % Maximum number of packets at an Eb/No point
```

Simulating for Each Eb/No Point

This example also demonstrates how a `parfor` loop can be used instead of the `for` loop when simulating each Eb/No point to speed up a simulation. `parfor`, as part of the “Parallel Computing Toolbox”, executes processing for each Eb/No point in parallel to reduce the total simulation time. To

enable the use of parallel computing for increased speed, comment out the 'for' statement and uncomment the 'parfor' statement below. If Parallel Computing Toolbox™ is not installed, 'parfor' will default to the normal 'for' statement.

```
numMode = numel(simMode);          % Number of modes
ber = zeros(numMode,length(EbNo)); % Pre-allocate to store BER results

for iMode = 1:numMode

    phyMode = simMode{iMode};
    % Set signal to noise ratio (SNR) points based on mode
    % For Coded PHY's (LE500K and LE125K), the code rate factor is included
    % in SNR calculation as 1/2 rate FEC encoder is used.
    if any(strcmp(phyMode,{'LE1M','LE2M'}))
        snrVec = EbNo - 10*log10(sps);
    else
        codeRate = 1/2;
        snrVec = EbNo + 10*log10(codeRate) - 10*log10(sps);
    end

%     parfor iSnr = 1:length(snrVec)  % Use 'parfor' to speed up the simulation
    for iSnr = 1:length(snrVec)       % Use 'for' to debug the simulation

        % Set random substream index per iteration to ensure that each
        % iteration uses a repeatable set of random numbers
        stream = RandStream('combRecursive','Seed',0);
        stream.Substream = iSnr;
        RandStream.setGlobalStream(stream);

        % Create an instance of error rate
        errorRate = comm.ErrorRate('Samples','Custom','CustomSamples',1:(dataLen-1));

        % Loop to simulate multiple packets
        numErrs = 0;
        numPkt = 1; % Index of packet transmitted
        while numErrs < maxNumErrors && numPkt < maxNumPackets

            % Generate Bluetooth LE waveform
            txBits = randi([0 1],dataLen,1,'int8'); % Data bits generation
            channelIndex = randi([0 39],1,1); % Random channel index value for each packet
            if channelIndex <=36
                % Random access address for data channels
                % Ideally, this access address value should meet the requirements specified in
                % Section 2.1.2, Part-B, Vol-6 of Bluetooth specification.
                accessAddress = [1 0 0 0 1 1 1 0 1 1 0 0 1 ...
                          0 0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 0]';
            else
                % Default access address for periodic advertising channels
                accessAddress = [0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 ...
                            1 0 0 0 1 0 1 1 1 0 0 0 1]';
            end
            txWaveform = bleWaveformGenerator(txBits,'Mode',phyMode,...
                                              'SamplesPerSymbol',sps,...
                                              'ChannelIndex',channelIndex,...
                                              'AccessAddress',accessAddress);

            % Pass the transmitted waveform through AWGN channel
            rxWaveform = awgn(txWaveform,snrVec(iSnr));
```

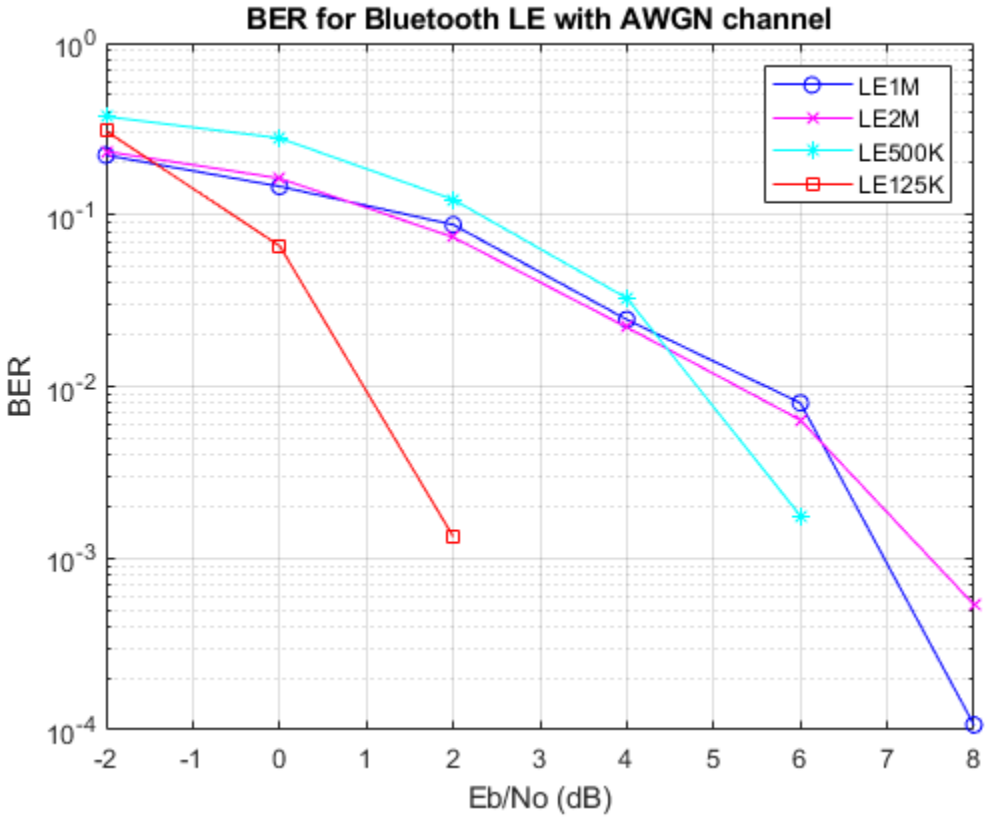
```
            % Recover data bits using ideal receiver
            rxBits = bleIdealReceiver(rxWaveform,'Mode',phyMode,...
                                      'SamplesPerSymbol',sps,...
                                      'ChannelIndex',channelIndex);

            % Determine the BER
            errors = errorRate(txBits,rxBits);
            ber(iMode,iSnr) = errors(1);
            numErrs = errors(2);
            numPkt = numPkt + 1;
        end
    disp(['Mode ' phyMode ', '...
        'Simulating for Eb/No = ', num2str(EbNo(iSnr)), ' dB' ', '...
        'BER:',num2str(ber(iMode,iSnr))])
    end
end
```

```
Mode LE1M, Simulating for Eb/No = -2 dB, BER:0.22222
Mode LE1M, Simulating for Eb/No = 0 dB, BER:0.14622
Mode LE1M, Simulating for Eb/No = 2 dB, BER:0.087542
Mode LE1M, Simulating for Eb/No = 4 dB, BER:0.024531
Mode LE1M, Simulating for Eb/No = 6 dB, BER:0.0080167
Mode LE1M, Simulating for Eb/No = 8 dB, BER:0.00010689
Mode LE2M, Simulating for Eb/No = -2 dB, BER:0.23377
Mode LE2M, Simulating for Eb/No = 0 dB, BER:0.16306
Mode LE2M, Simulating for Eb/No = 2 dB, BER:0.074074
Mode LE2M, Simulating for Eb/No = 4 dB, BER:0.022126
Mode LE2M, Simulating for Eb/No = 6 dB, BER:0.0063733
Mode LE2M, Simulating for Eb/No = 8 dB, BER:0.00053444
Mode LE500K, Simulating for Eb/No = -2 dB, BER:0.37326
Mode LE500K, Simulating for Eb/No = 0 dB, BER:0.27946
Mode LE500K, Simulating for Eb/No = 2 dB, BER:0.12266
Mode LE500K, Simulating for Eb/No = 4 dB, BER:0.032708
Mode LE500K, Simulating for Eb/No = 6 dB, BER:0.0017637
Mode LE500K, Simulating for Eb/No = 8 dB, BER:0
Mode LE125K, Simulating for Eb/No = -2 dB, BER:0.30736
Mode LE125K, Simulating for Eb/No = 0 dB, BER:0.065897
Mode LE125K, Simulating for Eb/No = 2 dB, BER:0.0013361
Mode LE125K, Simulating for Eb/No = 4 dB, BER:0
Mode LE125K, Simulating for Eb/No = 6 dB, BER:0
Mode LE125K, Simulating for Eb/No = 8 dB, BER:0
```

Plot BER vs Eb/No Results

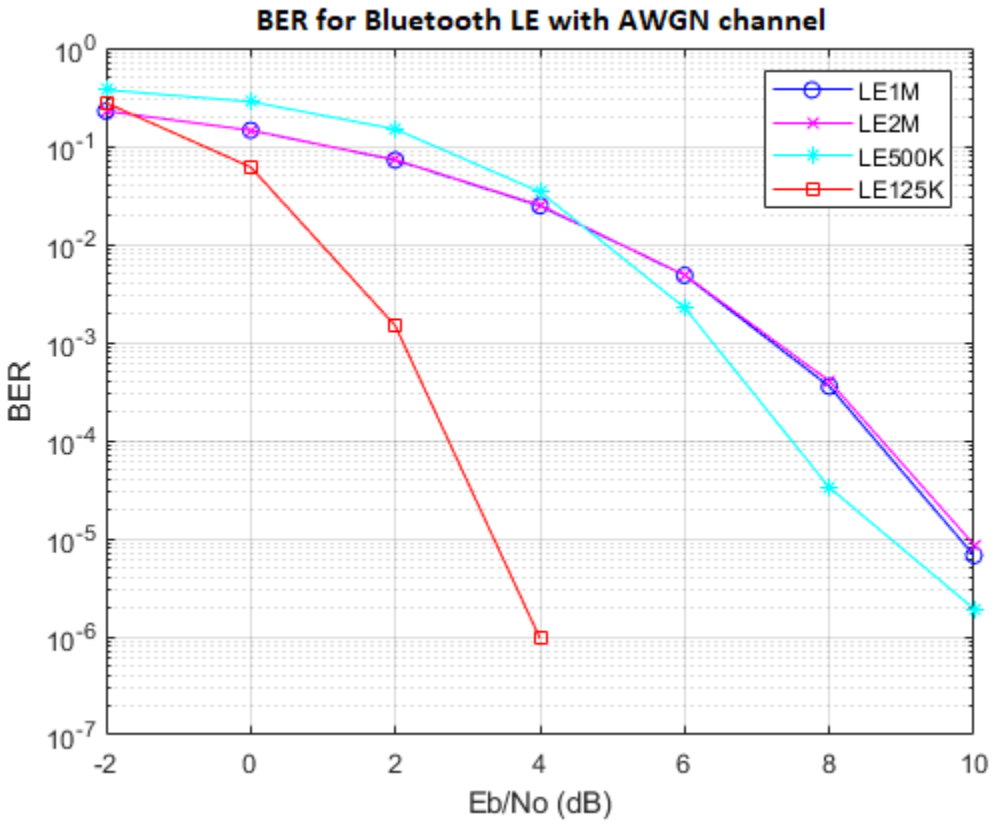
```
markers = 'ox*s';
color = 'bmcr';
dataStr = {zeros(numMode,1)};
figure;
for iMode = 1:numMode
    semilogy(EbNo,ber(iMode,:).',['-' markers(iMode) color(iMode)]);
    hold on;
    dataStr(iMode) = simMode(iMode);
end
grid on;
xlabel('Eb/No (dB)');
ylabel('BER');
```

```
legend(dataStr);
title('BER for Bluetooth LE with AWGN channel');
```



Further Exploration

The number of packets tested at each Eb/No point is controlled by `maxNumErrors` and `maxNumPackets` parameters. For statistically meaningful results these values should be larger than those presented in this example. The figure below was created by running the example for longer with `maxNumErrors = 1e3`, `maxNumPackets = 1e4`, for all the four modes.



Summary

This example simulates a Bluetooth LE physical layer link over an AWGN channel. It shows how to generate Bluetooth LE waveforms, demodulate and decode bits using an ideal receiver and compute the BER.

Selected Bibliography

1 Bluetooth Technology Website | The Official Website of Bluetooth Technology, Accessed November 22, 2021. https://www.bluetooth.com.

2 Volume 6 of the Bluetooth Core Specification, Version 5.3 Core System Package [Low Energy Controller Volume].

See Also

Functions

`bleWaveformGenerator` | `bleIdealReceiver`

Bluetooth Full Duplex Data and Voice Transmission in MATLAB

This example shows you how to model a full duplex communication in a Bluetooth® piconet having WLAN interference and supporting the adaptive frequency hopping (AFH) functionality using the Bluetooth® Toolbox. Using this example, you can:

- Create and configure a Bluetooth piconet with basic rate (BR) physical layer (PHY) mode.
- Configure asynchronous connection-oriented (ACL) logical transport and synchronous connection-oriented (SCO) logical transport to simultaneously transmit data and voice packets, respectively.
- Add WLAN interference and additive white Gaussian noise (AWGN) to the Bluetooth BR waveform.
- View packet error rate (PER) of each Bluetooth node in the presence of WLAN interference and AFH.
- Visualize the power spectral density of Bluetooth BR waveforms in the presence of WLAN interference and AFH.
- Add your own channel classification algorithm to analyze the simulation results.

Bluetooth Specifications

Bluetooth technology operates in 2.4 GHz industrial, scientific, and medical (ISM) band and shares it with other wireless technologies like ZigBee and WLAN. The Bluetooth Core Specification [2 on page 1-0] defined by the Special Interest Group (SIG) specifies two PHY modes: the mandatory BR and the optional enhanced data rate (EDR). The Bluetooth BR/EDR radio implements a 1600 hops/s frequency hopping spread spectrum (FHSS) technique. The radio hops in a pseudo-random way on 79 designated Bluetooth channels. Each Bluetooth channel has a bandwidth of 1 MHz. Each channel is centered at (2402 + k) MHz, where k = 0, 1, ... 78. The modulation technique on the payload for BR and EDR mode is Gaussian frequency shift-keying (GFSK) and differential phase shift-keying (DPSK), respectively. The baud rate is 1 MSymbols/s. The Bluetooth BR/EDR radio uses a time-division duplex (TDD) scheme in which data transmission occurs in one direction at one time. The transmission alternates in two directions, one after the other.

Bluetooth and WLAN radios often operate in the same physical scenario and in the same device. Therefore, Bluetooth and WLAN transmissions can interfere with each other, thus impacting the performance and reliability of both the networks. To mitigate this interference, the IEEE 802.15.2 Task Group [2 on page 1-0] recommends using the AFH technique. To study AFH and the coexistence of Bluetooth with WLAN, see “Bluetooth-WLAN Coexistence” on page 8-53.

For more information about the Bluetooth BR/EDR radio and the protocol stack, see “Bluetooth Protocol Stack” on page 8-9. For more information about Bluetooth BR/EDR packet structures, see “Bluetooth Packet Structure” on page 8-27.

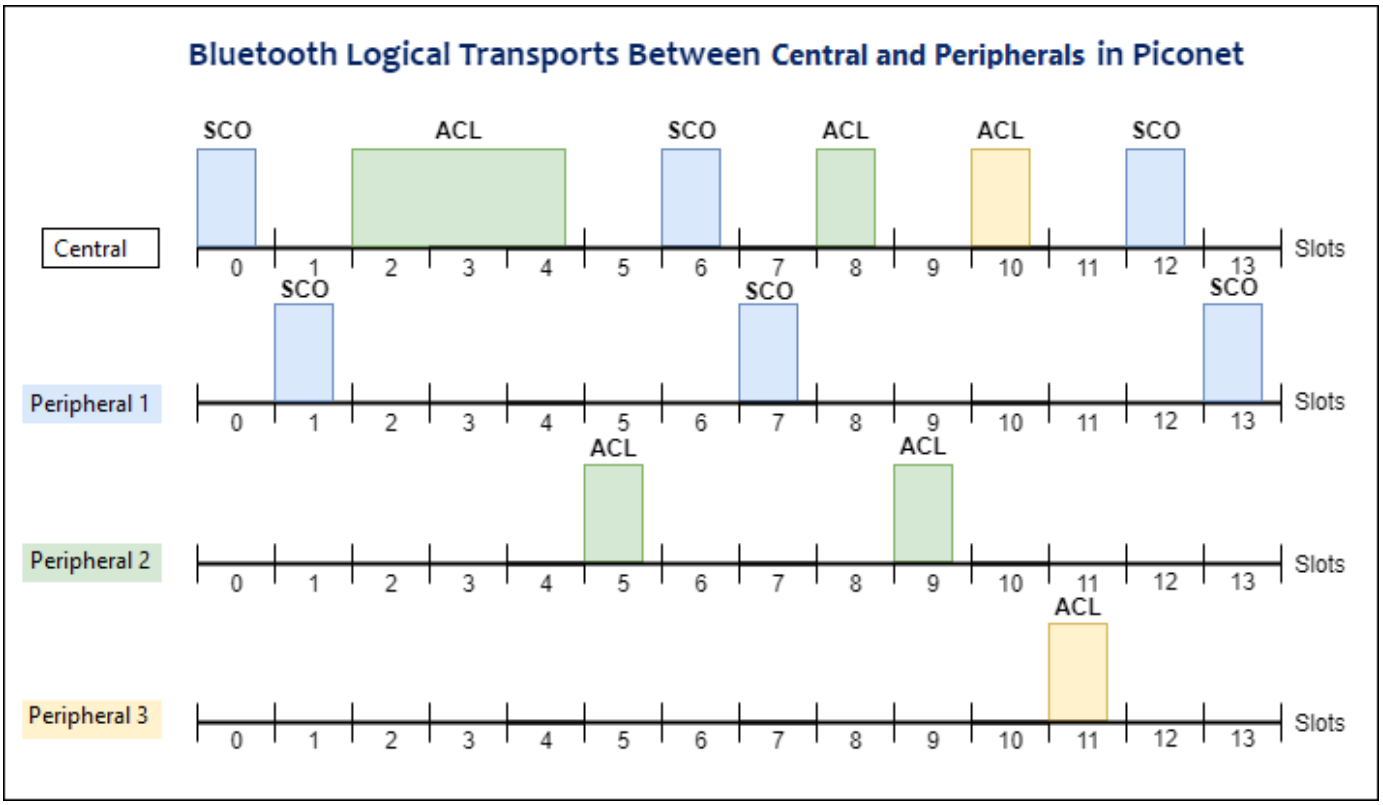
Logical Transports

The Bluetooth system supports point-to-point or point-to-multipoint connections called as *piconets*. Each piconet consists of a node in the role of Central, with other nodes in the Peripheral role. The Central and Peripheral exchange data over multiple logical transports. These logical transports are:

- *SCO*: The Central and Peripheral exchange SCO packets at regular intervals in the reserved slots. The Bluetooth nodes use SCO logical transport to exchange periodic data such as audio streaming. This logical transport does not support retransmissions.
- *Extended synchronous connection-oriented (eSCO)*: The Bluetooth nodes use eSCO to exchange periodic data such as audio streaming. This logical transport supports retransmissions.

- *ACL*: The Bluetooth nodes use ACL to exchange asynchronous data such as a file transfer protocol (FTP). During each poll interval, the Central polls the ACL logical transport of a Peripheral at least once.
- *Active peripheral broadcast (APB):* The Bluetooth nodes use ASB logical transport to send messages from the Central to all of the Peripherals in a piconet. This logical transport supports unidirectional traffic with no acknowledgments.
- *Connectionless peripheral broadcast (CPB):* The Central node uses a CSB logical transport to send profile broadcast data to multiple Peripherals. This logical transport supports unidirectional traffic with no acknowledgments.

This example supports ACL and SCO logical transports between a Central and Peripherals by considering data as random bits of 0s and 1s. This figure shows the communication between a Central and three Peripherals in a piconet over ACL and SCO logical transports.



Bluetooth uses reserved time slots for communication between the nodes. The duration of each slot is 625 microseconds. The Central node initiates the transmission in even slots and extends the transmission to odd slots when transmitting a multislot packet. The Peripheral node initiates the transmission in odd slots and extends the transmission to even slots when transmitting a multislot packet.

Configure Simulation Parameters

This section shows how to configure the simulation parameters for the Bluetooth piconet, the wireless channel, and WLAN interference.

Bluetooth Piconet

The `NumPeripherals` parameter specifies the number of Peripherals in the Bluetooth piconet. The `LinkTraffic` parameter specifies the type of traffic over Bluetooth logical transports between a Central and the respective Peripheral. This table maps `LinkTraffic` to different logical transports.

linkTraffic Value	Logical Transport
1	ACL
2	SCO
3	ACL and SCO

If the Central communicates with multiple Peripherals, `LinkTraffic` must be a vector.

The `SequenceType` parameter specifies the type of frequency hopping algorithm that the Bluetooth node uses. When you set `SequenceType` to `'Connection adaptive'`, the Bluetooth channels are classified as *good* or *bad* periodically based on the PER of each Bluetooth channel. To classify the Bluetooth channels, you can use the classifyChannels object function.

```
simulationParameters = struct;

% Set the simulation time in microseconds
simulationTime = 3*1e6;

% Enable or disable the visualization in the example
enableVisualization = true;

% Configure the number of Peripherals in the piconet
simulationParameters.NumPeripherals = 1 ;

% Specify the positions of Bluetooth nodes in the form of n-by-3 array.
% where n represents the number of nodes in the piconet. Each row specifies
% the cartesian coordinates of a nodes starting from Central and followed
% by Peripherals.
simulationParameters.NodePositions = [10 0 0; 20 0 0];

% Configure the logical links between the Central and Peripherals

% Each element represents the logical link between the Central and the
% respective Peripheral. If the Central is connected to multiple
% Peripherals, this value must be a row vector.
simulationParameters.LinkTraffic = 1;

% Configure the frequency hopping sequence as 'Connection basic' or
% 'Connection adaptive'
simulationParameters.SequenceType = Connection adaptive ;

% To enable an ACL logical transport, set linkTraffic to 1 or 3. Specify
% the ACL packet type as 'DM1', 'DH1', 'DM3', 'DH3', 'DM5', or 'DH5'.
```

```
simulationParameters.ACLPacketType = DM1;

% To enable a SCO logical transport, set linkTraffic to 2 or 3. Specify the
% SCO packet type as 'HV1', 'HV2', or 'HV3' for the respective Peripheral that
% has SCO link traffic. Here, 1 represents the Peripheral number, and 'HV3'
% represents the corresponding SCO packet type used by Peripheral 1.
simulationParameters.SCOPacketType = {1, 'HV3'};
```

Wireless Channel and WLAN Interference

Configure the wireless channel by using the helperBluetoothChannel helper object. You can set the `EbNo` value for the AWGN channel. To generate the WLAN signal interference, use the helperBluetoothGenerateWLANWaveform helper function. Specify the sources of WLAN interference by using the `WLANInterference` parameter. Use one of these options to specify the source of the WLAN interference.

- `'Generated'`: To add a WLAN Toolbox™ signal, select this option. Perform the steps shown in further exploration on page 1-0 to add the signal from the WLAN Toolbox™.
- `'BasebandFile'`: To add a WLAN signal from a baseband file (`.bb`), select this option. You can specify the file name using the `WLANBBFilename` input argument. If you do not specify the `.bb` file, the example uses the default `.bb` file, `'WLANNonHTDSSS.bb'`, to add the WLAN signal.

The `'None'` option implies that no WLAN signal is added. AWGN is present throughout the simulation.

```
% Configure wireless channel parameters
simulationParameters.EbNo = 22; % Ratio of energy per bit (Eb) to spectral noise density (No) in

% Configure the WLAN interference

% Specify the WLAN interference as 'Generated', 'BasebandFile', or 'None'.
% To use the 'wlanBBFilename' option, set wlanInterference to 'BasebandFile'.

simulationParameters.WLANInterference = BasebandFile;
simulationParameters.WLANBBFilename =  'WLANNonHTDSSS.bb';

% Signal to interference ratio in dB
simulationParameters.SIR = [-15 -16];
```

Channel Classification Parameters

Classify the Bluetooth channels as good or bad by using the helperBluetoothChannelClassification helper object only when the `SequenceType` is `'Connection adaptive'`. The example classifies the Bluetooth channels by using these parameters.

- `PERThreshold`: PER threshold
- `ClassificationInterval`: Periodicity (in slots) of channel classification
- `RxStatusCount`: Maximum number of received packets status maintained for each channel
- `MinRxCountToClassify`: Minimum number of received packets status for each channel to classify a channel as good or bad
- `PreferredMinimumGoodChannels`: Preferred number of good channels required to communicate between the Central and Peripherals

You can add your own classification algorithm by customizing the classifyChannels object function of the helperBluetoothChannelClassification helper object.

```
simulationParameters.PERThreshold = 50;                    % Packet error
simulationParameters.ClassificationInterval = 3000;        % In slots
simulationParameters.RxStatusCount = 10;                   % Maximum Rx pa
simulationParameters.MinRxCountToClassify = 4;             % Minimum packe
simulationParameters.PreferredMinimumGoodChannels = 20;    % Preferred num
```

Create Bluetooth Piconet

Specify the total number of Bluetooth nodes in the piconet. Set the role of the nodes as Central or Peripheral. To create a Bluetooth piconet from the configured parameters, use the helperBluetoothCreatePiconet helper function.

```
% Set the default random number generator ('twister' type with seed value 0).
% The seed value controls the pattern of random number generation. For high
% fidelity simulation results, change the seed value and average the
% results over multiple simulations.
rng('default');

% Specify Tx power, in dBm
simulationParameters.TxPower = 20;

% Specify the Bluetooth node receiver range (in meters)
simulationParameters.ReceiverRange = 40;

% Set the total number of nodes in the piconet (one Central and multiple
% Peripherals)
numNodes = simulationParameters.NumPeripherals + 1;

% Create a Bluetooth piconet
bluetoothNodes = helperBluetoothCreatePiconet(simulationParameters);
```

To visualize the Bluetooth waveforms, create the `dsp.SpectrumAnalyzer` System object™.

```
% View the Bluetooth waveforms using the spectrum analyzer
spectrumAnalyzer = dsp.SpectrumAnalyzer(...
    'ViewType','Spectrum and spectrogram', ...
    'TimeResolutionSource','Property', ...
    'TimeResolution',0.0005, ...                          % In seconds
    'SampleRate',bluetoothNodes{1}.PHY.SamplesPerSymbol*1e6, ... % In Hz
    'TimeSpanSource','Property', ...
    'TimeSpan',0.05, ...                                  % In seconds
    'FrequencyResolutionMethod','WindowLength', ...
    'WindowLength',512, ...                               % In samples
    'AxesLayout','Horizontal', ...
    'FrequencyOffset',2441*1e6, ...
    'ColorLimits',[-20 15]);
```

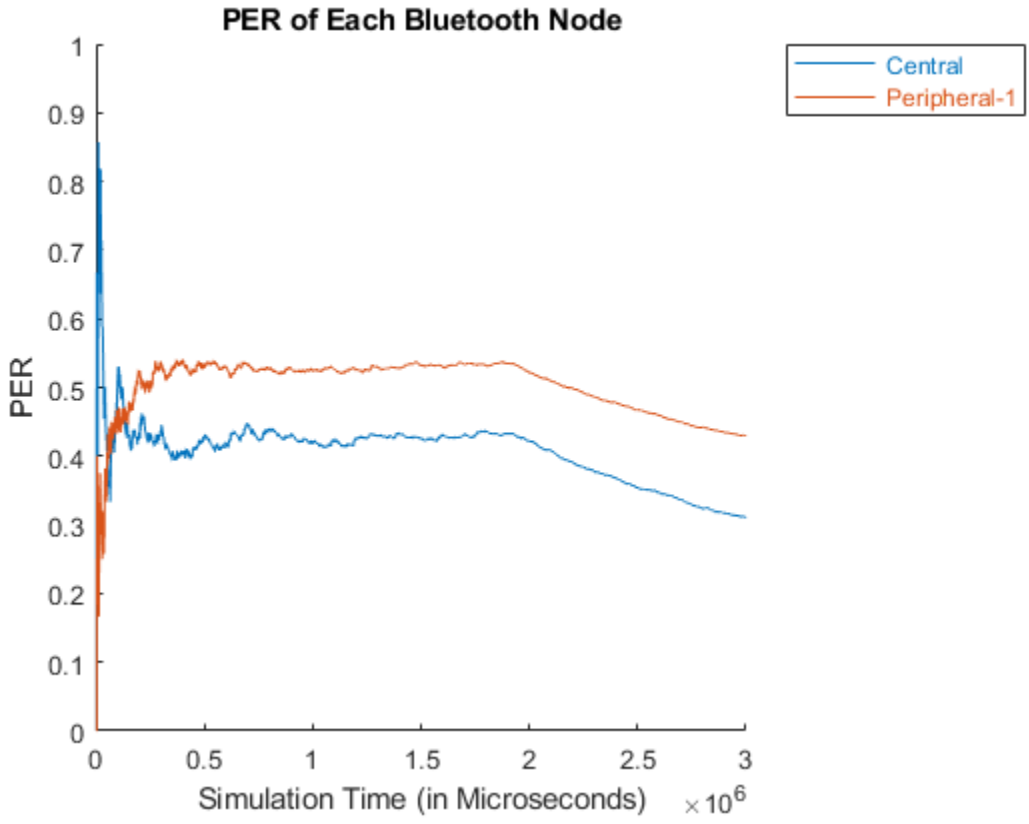
Simulation

Simulate the Bluetooth piconet using the configured parameters. Visualize the plot of the PER of each Bluetooth node in the piconet. Visualize the power spectral density of the Bluetooth waveforms by

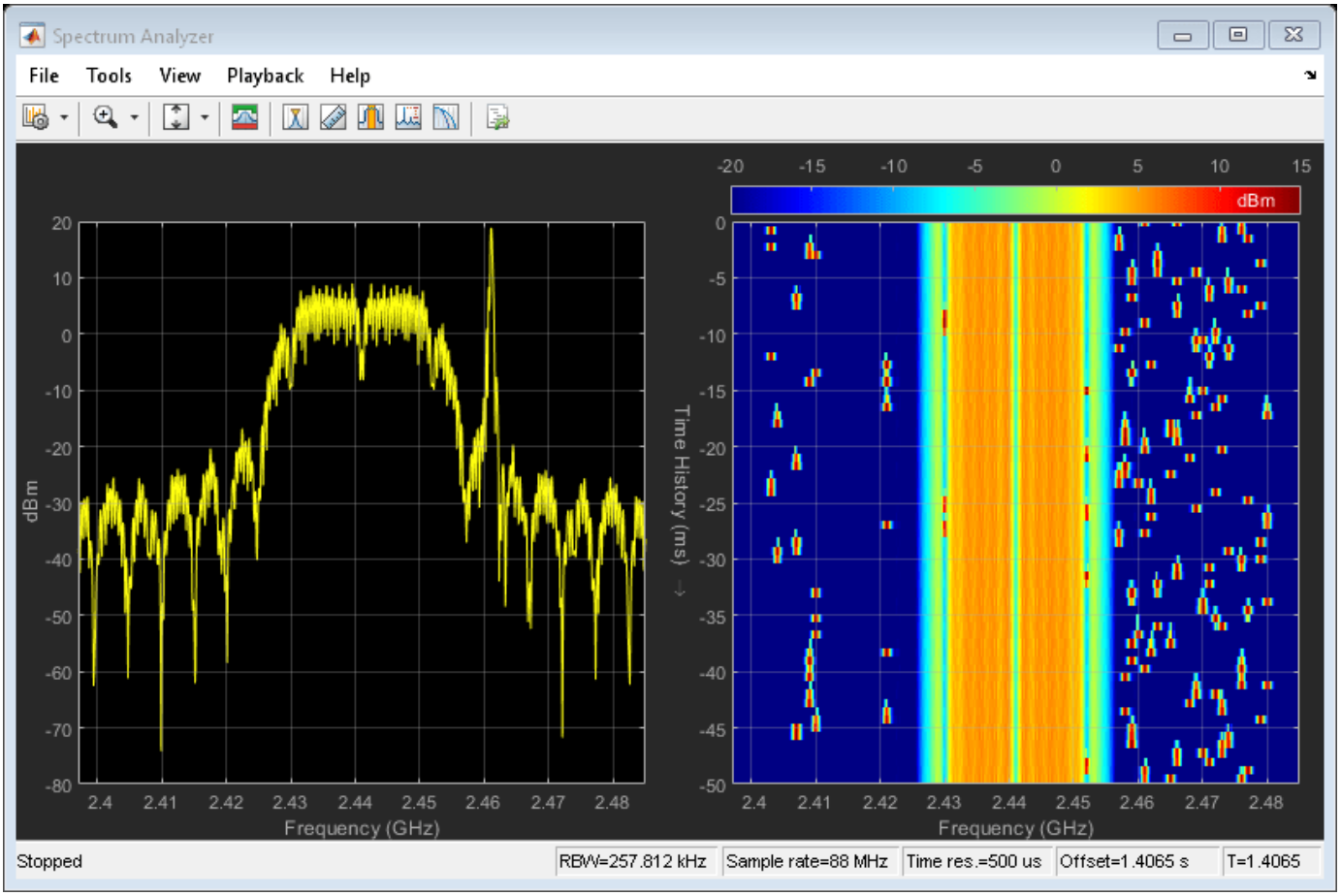
using the `dsp.SpectrumAnalyzer` System object. You can also calculate the baseband layer statistics (total transmitted packets, total received packets, and total dropped packets) and channel classification statistics at each Bluetooth node. When the sequence type is set to `'Connection adaptive'`, the Bluetooth node updates the channel classification statistics.

```
% Current simulation time in microseconds
curTime = 0;

% Elapsed time in microseconds
elapsedTime = 0;

% Next invoke times of all of the nodes in microseconds
nextInvokeTimes = zeros(1, numNodes);

if enableVisualization
    % Plot the PER
    perFigure = figure("Name","PER of Each Bluetooth Node",'Tag','BluetoothPERPlot');
    perAxes = axes(perFigure);

    % Add annotations to the figure
    ylim(perAxes, [0 1]);
    xlabel(perAxes, 'Simulation Time (in Microseconds)');
    ylabel(perAxes, 'PER');
    title(perAxes, 'PER of Each Bluetooth Node');

    % Plot the PER line for each Bluetooth node
    [perPlots, legendStr] = deal(cell(1, numNodes));
    for plotIdx = 1:numNodes
        hold on
        perPlots{plotIdx} = plot(perAxes, curTime, 0);
        if plotIdx == 1
            legendStr{1} = ['\color[rgb]{' num2str(perPlots{plotIdx}.Color) '} Central'];
        else
            legendStr{plotIdx} = ['\color[rgb]{' num2str(perPlots{plotIdx}.Color) '} Peripheral-
        end
    end

    % Add a legend to the figure
    legend(perAxes, legendStr,'Location','northeastoutside','Box','on');
    if ~strcmpi(simulationParameters.WLANInterference, 'None')
        % Generate the WLAN waveform for visualization
        wlanWaveform = helperBluetoothGenerateWLANWaveform(...
            simulationParameters.WLANInterference, simulationParameters.WLANBBFilename);
    end
end

% Run the simulation
while(curTime < simulationTime)
    % Simulate the Bluetooth nodes
    for nodeIdx = 1:numNodes
        % Push the data into the node
        pushData(bluetoothNodes{nodeIdx}, ...
            simulationParameters.ACLPacketType, simulationParameters.SCOPacketType);

        % Run the Bluetooth node instance
        nextInvokeTimes(nodeIdx) = runNode(bluetoothNodes{nodeIdx}, elapsedTime);
```

```
            if enableVisualization
                if ~isempty(findobj('Tag', 'BluetoothPERPlot'))
                    % Update and plot the PER
                    perPlots{nodeIdx}.XData = [perPlots{nodeIdx}.XData curTime];
                    if isempty(bluetoothNodes{nodeIdx}.PHY.PER)
                        per = 0;
                    else
                        per = bluetoothNodes{nodeIdx}.PHY.PER;
                    end
                    perPlots{nodeIdx}.YData = [perPlots{nodeIdx}.YData per];
                end
            end
        end

        % Update the current simulation time
        curTime = curTime + elapsedTime;

        % Distribute any transmitted packets from each node into the receiving
        % buffers of the other nodes
        [isPacketDistributed, txBuffer] = helperBluetoothDistributePackets(bluetoothNodes);

        % Update visualization
        if enableVisualization
            for txIdx = 1:numel(txBuffer)
                channelWaveform = txBuffer{txIdx}.Waveform(1:txBuffer{txIdx}.NumSamples);
                if ~strcmpi(simulationParameters.WLANInterference, 'None')
                    % Add WLAN interference to the channel waveform
                    channelWaveform = channelWaveform + wlanWaveform(1:txBuffer{txIdx}.NumSamples);
                end

                % Plot the Bluetooth waveform
                spectrumAnalyzer(channelWaveform);
            end
        end

        % If packets are distributed to the receiver nodes, run the nodes to
        % check the packet reception buffer.
        if isPacketDistributed
            elapsedTime = 0;

        % Advance the simulation time to the next event at a node
        else
            elapsedTime = min(nextInvokeTimes(nextInvokeTimes ~= -1));
        end
    end

ans = 1x79 logical array

   0   1   1   0   0   1   0   1   1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
```




```
release(spectrumAnalyzer);
```



The preceding spectrum analyzer plot shows the spectrum of the Bluetooth waveform distorted with WLAN interference (in the frequency domain) and passed through the AWGN channel. The right-side plot shows the overlapping of Bluetooth packets with the interfering WLAN signal.

The plot "PER of Each Bluetooth Node" shows the PER of each node in the Bluetooth piconet with respect to the simulation time.

To see the baseband layer statistics for each Bluetooth node, explore the `statisticsAtEachNode` variable. To see the channel classification statistics for each Central-Peripheral pair, inspect the `classificationStats` variable. The channel classification statistics are valid when `sequenceType` is set to `'Connection adaptive'`. Get the baseband layer and channel classification statistics of each Bluetooth node in the piconet.

```
% Get the baseband layer and channel classification statistics of each Bluetooth node in the pico
[statisticsAtEachNode, classificationStats] = helperBluetoothFullDuplexStatistics(bluetoothNodes)
```

```
statisticsAtEachNode=2×19 table
                 TotalRxPackets    TotalTxPackets    TxACLPackets    TxACLOneSlotPackets    TxA
                 ______________    ______________    ____________    ___________________    ___

    Central            1402              2401        {7x2 double}       {7x2 double}
    Peripheral1        2291              1444        {7x2 double}       {7x2 double}
```

```
classificationStats=1×3 table
    LTAddress    ClassificationCount                                    ChannelMap
    _________    ___________________    ____________________________________________________________

        1                1              {[0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
```

This example enables you to configure and simulate a full duplex data and voice communication in a Bluetooth piconet with WLAN interference. The example uses AFH to mitigate WLAN interference by classifying channels as *good* or *bad* based on the PER value. The performance of full duplex communication is analyzed by visualizing the impact of WLAN interference on each Bluetooth node.

Further Exploration

To add the WLAN signal using the features of the WLAN Toolbox™, set the value of wlanInterference to 'Generated'. Use this code to add the WLAN signal.

Add the WLAN signal generated using the features of the WLAN Toolbox™ as static signal interference to Bluetooth. To enable this, set `wlanInterference` value as 'Generated', add your custom signal generation code in function. Use this sample code snippet in WLAN signal generation using features of WLAN Toolbox™.

```
% % WLAN waveform to interfere with Bluetooth waveforms can be
% % modified by using the features of the WLAN Toolbox(TM).
% psduLength = 1000;
%
% % Create configuration object for WLAN waveform (802.11b)
% cfgNHT = wlanNonHTConfig('Modulation','DSSS', ...
%     'PSDULength', psduLength);
%
% % Create random PSDU
% psdu = randi([0 1], cfgNHT.PSDULength*8, 1);
%
% % Generate WLAN waveform
% wlanWaveform = wlanWaveformGenerator(psdu, cfgNHT);
```

You can add your custom signal generation code in the helperBluetoothGenerateWLANWaveform helper function. You can also write the respective signal WLAN spectrum masks and register to the `WLANSpectrum` property of the helperBluetoothChannel helper object as a function pointer.

Appendix

The example uses these helpers:

- helperBluetoothFullDuplexNode: Configure and simulate Bluetooth node
- helperBluetoothBaseband: Configure and simulate Bluetooth baseband layer
- helperBluetoothLogicalTransports: Configure logical transports between Bluetooth nodes
- helperBluetoothSlotTimer: Manage Bluetooth clock and the timing of slots
- helperBluetoothPHY: Configure and simulate Bluetooth PHY layer
- helperBluetoothChannel: Configure and simulate Bluetooth wireless channel
- helperBluetoothGenerateWLANWaveform: Generate WLAN waveform to be added as an interference to Bluetooth waveforms
- helperBluetoothCreatePiconet: Create Bluetooth piconet using the Bluetooth nodes

- helperBluetoothFullDuplexStatistics: Return statistics of each Bluetooth node in the Bluetooth piconet
- helperBluetoothQueue: Create an object for Bluetooth queue functionality

Selected Bibliography

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 29, 2021. https://www.bluetooth.com/.
2. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com.
3. IEEE® Standard 802.15.2™. "Coexistence of Wireless Personal Area Networks with Other Wireless Devices Operating in Unlicensed Frequency Bands." *IEEE Recommended Practice for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements; IEEE Computer Society.*

See Also

Objects

`bluetoothFrequencyHop`

More About

- “Create, Configure, and Simulate Bluetooth LE Network” on page 9-38
- “Create Bluetooth Piconet by Enabling ACL Traffic, SCO Traffic, and AFH” on page 9-23
- “Packet Distribution in Bluetooth Piconet” on page 9-28

Coexistence Modeling

PHY Simulation of Bluetooth BR/EDR, LE, and WLAN Coexistence

This example shows you how to model homogenous and heterogeneous coexistence between Bluetooth® basic rate/enhanced data rate (BR/EDR), low energy (LE) , and wireless local area waveforms (WLAN) by using Bluetooth® Toolbox.

Using this example, you can:

- Perform Bluetooth BR/EDR and LE end-to-end simulation in the presence of Bluetooth BR/EDR, LE, or WLAN interference.
- Perform adaptive frequency hopping (AFH) by classifying the channels as "good" or "bad" based on the packet error rate (PER).
- Compute the bit error rate (BER) and signal-to-interference-plus noise ratio (SINR).
- Visualize the spectrum and spectrogram of the Bluetooth BR/EDR or LE waveform in the presence of interference.

Bluetooth and WLAN Coexistence

Bluetooth operates in the unlicensed 2.4 GHz industrial, scientific, and medical (ISM) band from 2.4 to 2.4835 GHz, which is also used by other technologies such as Zigbee and WLAN. Multiple homogenous and heterogeneous networks operating in this band are likely to coexist in a physical scenario. To mitigate interference, Bluetooth and WLAN implement AFH and carrier-sense multiple access with collision avoidance (CSMA/CA), respectively. AFH enables Bluetooth devices to improve their robustness to interference and avoid interfering with other devices in the 2.4 GHz ISM band. The basic principle is to classify interference channels as bad channels and discard them from the list of available channels. This classification mechanism of AFH enables a Bluetooth device to use 79 channels or fewer in BR/EDR mode and 40 channels or fewer in LE mode. The Bluetooth Core Specification [2 on page 2-0] allows a minimum of 20 channels in BR/EDR mode and 2 channels in LE mode. For more information about coexistence between Bluetooth and WLAN, see “Bluetooth-WLAN Coexistence” on page 8-53.

Coexistence mechanisms can be classified into two categories: collaborative or noncollaborative, depending on whether the involved networks operate independently of one another or coordinate their use of the spectrum. In noncollaborative coexistence, each network treats other networks as interference and performs interference mitigation techniques. In collaborative coexistence, all the networks collaborate and coordinate their use of the spectrum. This example illustrates a noncollaborative coexistence mechanism between homogenous and heterogeneous networks.

This example uses these terminologies:

- AWN - Affected wireless node, can be one of these:

- Bluetooth: BR, EDR 2Mbps, EDR 3Mbps, LE 1Mbps, LE 2Mbps, LE 500Kbps, and LE 125Kbps

- IWN - Interfering wireless node, can be one of these:

- Bluetooth:

BR, EDR 2Mbps, EDR 3Mbps, LE 1Mbps, LE 2Mbps, LE 500Kbps, and LE 125Kbps

- WLAN:

802.11b with 22 MHz bandwidth

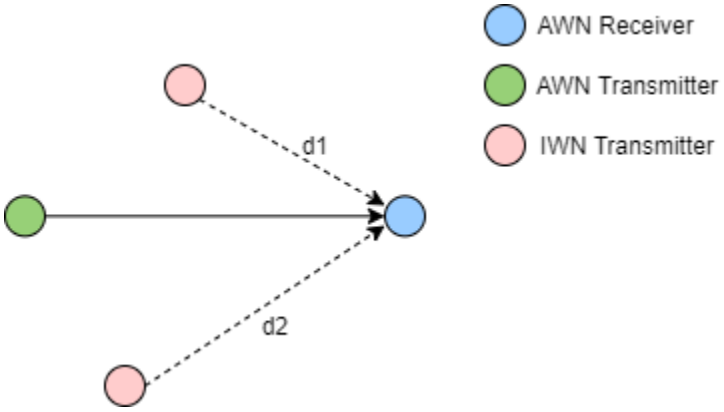
802.11g with 20 MHz bandwidth

802.11n with 20 MHz and 40 MHz bandwidths

802.11ax with 20 MHz and 40 MHz bandwidths

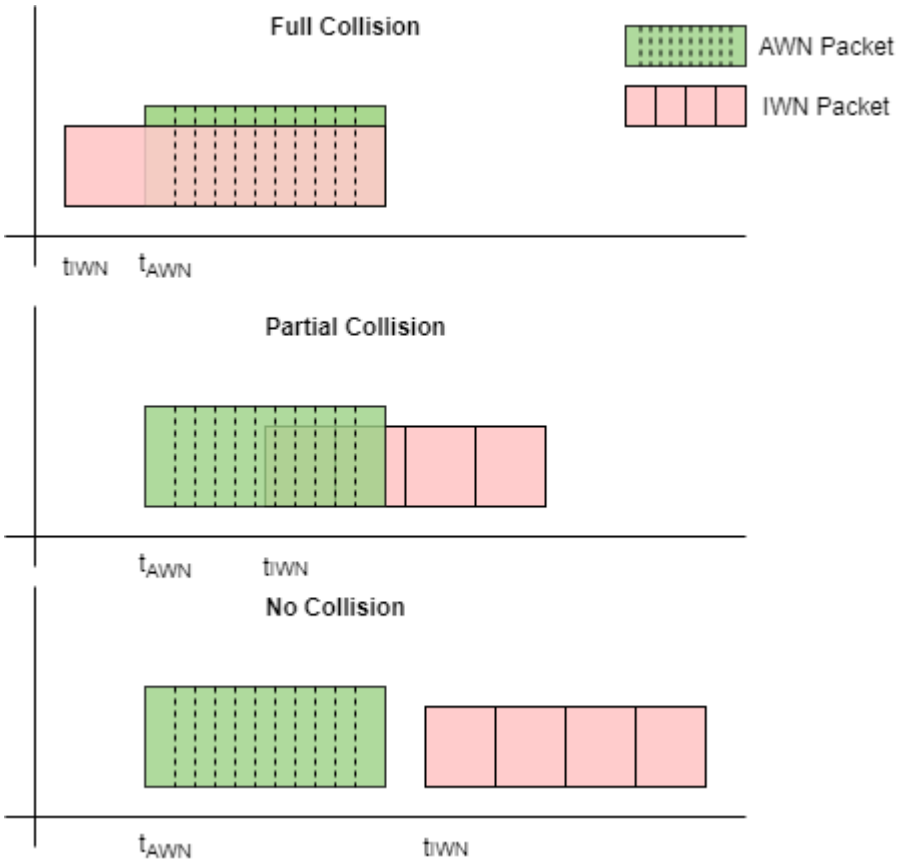
Impact of Interference in Space, Time, and Frequency Domains

Space: As the distance between AWN and IWN nodes increases, the impact of interference in space domain decreases. In this figure, if d1 and d2 increase, the impact of the IWN transmitter interference on the AWN receiver decreases.

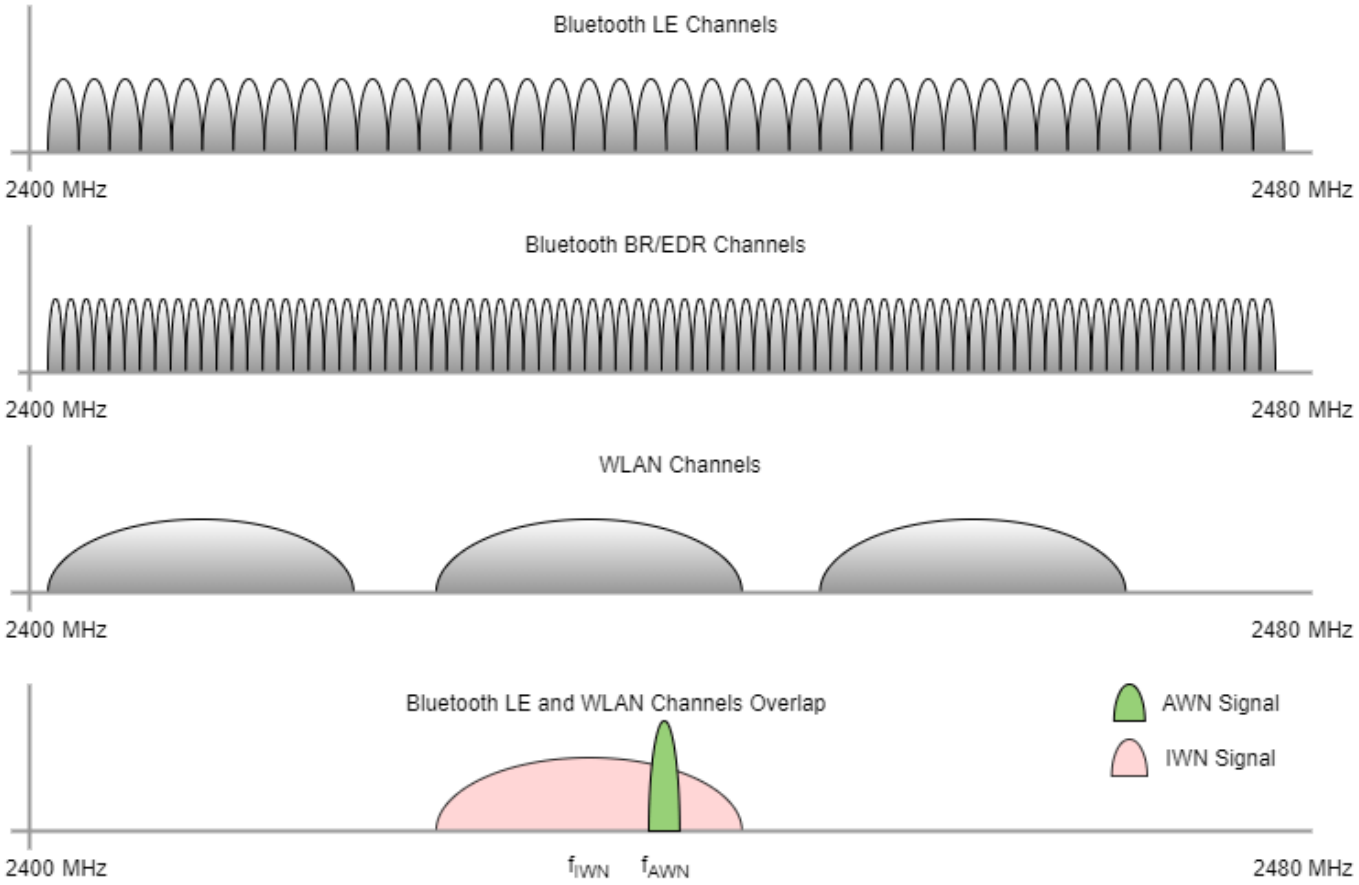


Time: Depending on the packet transmission timings, three possible collision probabilities arise in the time domain: full collision, partial collision, or no collision.

- Full - IWN packet completely interferes with the AWN packet.
- Partial - IWN packet partially interferes with the AWN packet with the given probability.
- No - IWN packet does not interfere with the AWN packet.



Frequency: As the channel separation between the AWN and IWN nodes increases, the impact of interference in frequency domain decreases. In this figure, if difference between f_{IWN} and f_{AWN} increase, the impact of the IWN transmitter interference on the AWN receiver decreases.



Simulation Parameters

Specify the AWN parameters such as the signal type, transmitter position, receiver position, transmitter power, and packet type.

Specify frequency hopping as one of these values.

- Off (default) - To run simulation at a fixed frequency, use this value. If you use this value, the example configures `awnFrequency`.
- On - To run simulation with AFH, use this value. If you use this value, the example does not configure `awnFrequency`.

```
awnSignalType = Bluetooth LE 1M;
awnTxPosition = [0,0,0];     % In meters
awnRxPosition = [10,0,0];    % In meters
awnTxPower = 30;             % In dBm
awnPacket = Disabled;
awnFrequencyHopping = Off;
awnFrequency = 2440*1e6; % In Hz
```

Configure a single or multiple IWNs and their respective parameters such as the signal type, transmitter position, fixed frequency of operation, and transmitter power. Create and configure multiple IWN nodes by using the IWN structure with different indices.

Specify the collision probability in the range [0,1]. Any value between 0 and 1 simulates partial collision. To simulate full collision, set this value to 1. To disable interference and simulate with no collisions, set this value to 0.

Add different types of WLAN signals as interference using WLAN Toolbox™. If you do not have WLAN Toolbox™, use `WLANBasebandFile` to add 802.11ax signal.

```
iwn(1).SignalType = WLANBasebandFile;
iwn(1).TxPosition = [20,0,0];      % In meters
iwn(1).Frequency = 2437e6;         % In Hz
iwn(1).TxPower = 30;               % In dBm
iwn(1).CollisionProbability = 1;   % Probability of collision in time, must be between [0,1]

iwn(2).SignalType = Bluetooth LE 1M;
iwn(2).TxPosition = [25,0,0];      % In meters
iwn(2).Frequency = 2420e6;         % In Hz
iwn(2).TxPower = 30;               % In dBm
iwn(2).CollisionProbability = 0.2; % Probability of collision in time, must be between [0,1]
```

Specify the environment, bit energy to noise density ratio (Eb/No), sample rate, and number of packets.

```
environment = Outdoor;
EbNo = 10;          % In dB
sampleRate = 80e6; % In Hz
numPackets = 500;
```

Set the seed for the random number generator.

```
rng default;
```

Configure the waveform transmission and reception parameters of the AWN.

```
phyFactor = 1+strcmp(awnSignalType,"LE2M");
sps = sampleRate/(1e6*phyFactor);                       % Samples per symbol
if sps > 8                                              % Decimation factor for the receiver filt
    decimationFactor = gcd(sps,8);
else
    decimationFactor = 1;
end

if any(strcmp(awnSignalType,["LE1M","LE2M","LE500K","LE125K"]))
    payloadLength = 100;                                % Length of the payload in bytes
    accessAddress = "01234567";                         % Access address
    accessAddBits = int2bit(hex2dec(accessAddress),32,false);

    % Derive channel index based on the AWN frequency
    channelIndexArray = [37 0:10 38 11:36 39];
    awnBandwidth = 2e6;
    channelIndex = channelIndexArray((awnFrequency-2402e6)/awnBandwidth+1);
```

```
    % Configure the receiver parameters in a structure
    rxCfg = struct(Mode=awnSignalType,SamplesPerSymbol=sps/decimationFactor,ChannelIndex=channel
        DFPacketType=awnPacket,AccessAddress=accessAddBits);
    rxCfg.CoarseFreqCompensator = comm.CoarseFrequencyCompensator(Modulation="OQPSK", ...
                                            SampleRate=sampleRate/decimationFactor, ...
                                            SamplesPerSymbol=2*rxCfg.SamplesPerSymbol, ...
                                            FrequencyResolution=100);
    rxCfg.PreambleDetector = comm.PreambleDetector(Detections="First");
else
    % Create and configure Bluetooth waveform generation parameters
    awnWaveformConfig = bluetoothWaveformConfig(Mode=awnSignalType,PacketType=awnPacket, ...
                            SamplesPerSymbol=sps);
    if strcmp(awnPacket,"DM1")
        awnWaveformConfig.PayloadLength = 17;            % Maximum length of DM1 packets in bytes
    end
    payloadLength = getPayloadLength(awnWaveformConfig); % Length of the payload

    % Get the receiver configuration parameters
    rxCfg = getPhyConfigProperties(awnWaveformConfig);
    rxCfg.SamplesPerSymbol = sps/decimationFactor;
end
```

Estimate the AWN path loss.

```
% Estimate distance between AWN transmitter and AWN receiver
distanceAWNTxRx = sqrt(sum((awnTxPosition-awnRxPosition).^2));
[awnPathloss,pathlossdB] = helperBluetoothEstimatePathLoss(environment,distanceAWNTxRx);
```

Create and configure the IWN by using the `helperIWNConfig` object. Generate IWN waveforms by using the `generateIWNWaveform` method. Add the path loss based on the environment and node positions by using the `applyPathloss` method.

```
iwnConfig = helperIWNConfig(IWN=iwn,SampleRate=sampleRate,Environment=environment);
iwnWaveform = generateIWNWaveform(iwnConfig);
[iwnWaveformPL,iwnPathloss] = applyPathloss(iwnConfig,iwnWaveform,awnRxPosition);
```

Use the `bluetoothFrequencyHop` and `bleChannelSelection` objects to select a channel index for the transmission and reception of Bluetooth BR/EDR and LE waveforms, respectively.

```
if strcmp(awnFrequencyHopping,"On")
    if any(strcmp(awnSignalType,["LE1M","LE2M","LE500K","LE125K"]))
        frequencyHop = bleChannelSelection;   % Bluetooth LE channel index System object™
        numBTChannels = 37;                   % Number of Bluetooth LE channels
        minChannels = 2;                      % Minimum number of channels to classify
    else
        frequencyHop = bluetoothFrequencyHop; % Bluetooth BR/EDR channel index object
        frequencyHop.SequenceType = "Connection Adaptive";
        numBTChannels = 79;                   % Number of Bluetooth BR/EDR channels
        minChannels = 20;                     % Minimum number of channels to classify
        inputClock = 0;
        numSlots = 1*(any(strcmp(awnPacket,["ID","NULL","POLL","FHS","HV1","HV2", ...
                    "HV3","DV","EV3","DM1","DH1","AUX1","2-DH1","3-DH1","2-EV3","3-EV3"])))...
                    +(3*any(strcmp(awnPacket,["EV4","EV5","DM3","DH3","2-EV5","3-EV5","2-DH3", ...
                    "3-DH3"])))+ (5*any(strcmp(awnPacket,["DM5","DH5","2-DH5","3-DH5"])));
        slotValue = numSlots*2;
        clockTicks = slotValue*2;             % Clock ticks (one slot is two clock ticks)
    end
end
```

Design a receiver filter to capture the AWN waveform.

```
if any(strcmp(awnSignalType,["EDR2M","EDR3M"]))
    rolloff = 0.4;
    span = 8;
    filterCoeff = rcosdesign(rolloff,span,sps,"sqrt");
else
    N = 200;                                    % Order
    Fc = 1.5e6/(1+strcmp(awnSignalType,"BR")); % Cutoff frequency
    flag = "scale";                             % Sampling flag
    alpha = 3;                                  % Window parameter

    % Create the window vector for the design algorithm
    win = gausswin(N+1,alpha);

    % Calculate the coefficients using the FIR1 function
    filterCoeff = fir1(N,Fc/(sampleRate/2),"low",win,flag);
end
firdec = dsp.FIRDecimator(decimationFactor,filterCoeff);
```

Compute the signal-to-noise ratio (SNR).

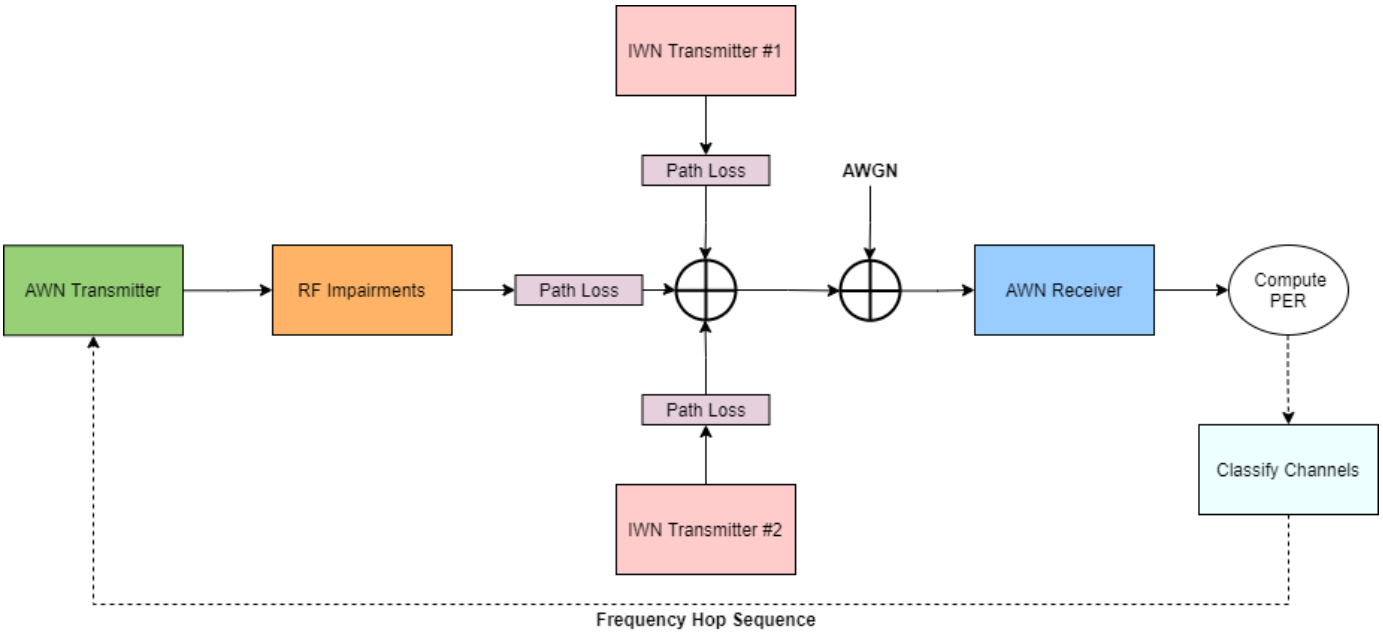
```
codeRate = 1*any(strcmp(awnSignalType,["LE1M","LE2M"]))+1/2*strcmp(awnSignalType,"LE500K")+1/8*s
           any(strcmp(awnSignalType,["BR","EDR2M","EDR3M"]))*(1-2/3*strcmp(awnPacket,"HV1")-...
           1/3*any(strcmp(awnPacket,["FHS","DM1","DM3","DM5","HV2","DV","EV4"]))); % Code rate
bitsPerSymbol = 1+ strcmp(awnSignalType,"EDR2M") + 2*(strcmp(awnSignalType,"EDR3M")); % Number o
snr = EbNo + 10*log10(codeRate) + 10*log10(bitsPerSymbol) - 10*log10(sps);
```

Create and configure the spectrum analyzer to visualize the spectrum and spectrogram of the Bluetooth BR/EDR or LE waveform in the presence of interference.

```
spectrumAnalyzer = dsp.SpectrumAnalyzer(...
    Name="Bluetooth Coexistence Modeling", ...
    ViewType="Spectrum and spectrogram", ...
    TimeResolutionSource="Property", ...
    TimeResolution=0.0005, ...
    SampleRate=sampleRate, ...
    TimeSpanSource="Property", ...
    TimeSpan=0.05, ...
    FrequencyResolutionMethod="WindowLength", ...
    WindowLength=512, ...
    AxesLayout="Horizontal", ...
    YLimits=[-100 20], ...
    ColorLimits=[-100 20]);
```

Coexistence Simulation

This diagram summarizes the example workflow.



Perform these steps to simulate the coexistence scenario.

1. Generate AWN (Bluetooth BR/EDR or LE) waveforms.
2. Distort each AWN waveform with these RF impairments: timing offset, carrier frequency offset, and DC offset.
3. Hop the waveform (frequency shift based on a center frequency of 2440 MHz) using the channel index derived from AFH.
4. Scale the hopped waveform with the transmitter power and path loss.
5. Generate and add IWN waveforms (Bluetooth BR/EDR, LE, or WLAN) based on the collision probabilities.
6. Add additive white Gaussian noise (AWGN).
7. Filter the noisy waveform.
8. Recover the bits from the filtered waveform by performing timing synchronization, carrier frequency offset correction, and DC offset correction.
9. Compute the PER, BER, and SINR.
10. If frequency hopping is on, classify the channels.

```
% Classify the channels for every |numPacketsToClassify| packets. If the PER of the
% channel is greater than |thresholdPER|, then map the corresponding channel
% as bad.
numPacketsToClassify = 50;
thresholdPER = 0.2;

% Create an instance of the error rate
errorRate = comm.ErrorRate;

% Initialize variables to perform the simulation
numErrors = 0;
numPktLost = 0;
countPER = 0;
```

```
countPreviousPER = 0;
midFrequency = 2440e6;
if strcmp(awnFrequencyHopping,"On")
    errorsBasic = deal(zeros(numBTChannels,3));
    errorsBasic(:,1) = (0:numBTChannels-1);
end

% Number of interfering nodes that collide with AWN
collisionCount = 0;
for index = 1:iwnConfig.NumIWNNodes
    collisionCount = collisionCount + (iwn(index).CollisionProbability > 0);
end

if strcmp(awnFrequencyHopping,"On") && collisionCount ~= 0
    sinr = zeros(numPackets,1);
end

% Loop to simulate multiple packets
for inum = 1:numPackets

    % Generate AWN waveform
    if any(strcmp(awnSignalType,["LE1M","LE2M","LE500K","LE125K"]))
        if strcmp(awnFrequencyHopping,"On")
            channelIndex = frequencyHop();
            channelFrequencies = [2404:2:2424 2428:2:2478 2402 2426 2480]*1e6;
            awnFrequency = channelFrequencies(channelIndex+1);
        end
        txBits = randi([0 1],payloadLength*8,1,"int8");
        awnWaveform = bleWaveformGenerator(int8(txBits),Mode=awnSignalType,ChannelIndex=channelI
            SamplesPerSymbol=sps,AccessAddress=accessAddBits,DFPacketType=awnPacket);
    else
        if strcmp(awnFrequencyHopping,"On")
            inputClock = inputClock + clockTicks;

            % Frequency hopping
            channelIndex = nextHop(frequencyHop,inputClock)
            awnFrequency = (2402+channelIndex)*1e6;

            % Generate whiten initialization vector from clock
            clockBinary = int2bit(inputClock,28,false).';
            awnWaveformConfig.WhitenInitialization = [clockBinary(2:7)'; 1];
        end
        txBits = randi([0 1],payloadLength*8,1);
        awnWaveform = bluetoothWaveformGenerator(txBits,awnWaveformConfig);
    end

    % Add timing offset
    timingOffset = randsrc(1,1,1:0.1:100);
    timingOffsetWaveform = helperBLEDelaySignal(awnWaveform,timingOffset);

    % Add frequency offset
    freqOffsetImp = randsrc(1,1,-10e3:100:10e3);
    freqOffsetWaveform = helperBLEFrequencyOffset(timingOffsetWaveform,sampleRate,freqOffsetImp)

    % Add DC offset
    dcValue = (5/100)*max(freqOffsetWaveform);
    dcWaveform = freqOffsetWaveform + dcValue;
```

```
% Shift the waveform by making 2440 MHz as the mid frequency
freqOffset = awnFrequency-midFrequency;
hopWaveform = helperBLEFrequencyOffset(dcWaveform,sampleRate,freqOffset);

% Scale the waveform as per the transmitter power and path loss
soiAmplitudeLinear = 10^((awnTxPower-30)/20)/awnPathloss;
attenAWNWaveform = soiAmplitudeLinear*hopWaveform;

% Add IWN waveforms to AWN waveform
addIWN2AWN = addInterference(iwnConfig,attenAWNWaveform,iwnWaveformPL,timingOffset);

% Frequency shift the waveform by |-freqOffset|
freqShiftWaveform = helperBLEFrequencyOffset(addIWN2AWN,sampleRate,-freqOffset);

% Add AWGN
soiPower = 20*log10(soiAmplitudeLinear);
noisePower = soiPower - snr;
splusibyn = 10*log10(var(freqShiftWaveform))-noisePower;
noisyWaveform = awgn(freqShiftWaveform,splusibyn,"measured");

% Apply filter
if rem(length(noisyWaveform),sps)
    remainder = sps-rem(length(noisyWaveform),sps);
    noisyWaveform = [noisyWaveform;zeros(remainder,1)]; %#ok<AGROW>
end
delay = floor(length(firdec.Numerator)/(2*decimationFactor));
noisyWaveformPadded = [noisyWaveform;zeros(delay*decimationFactor,1)];
filteredWaveform = firdec(noisyWaveformPadded);
release(firdec)
filteredWaveform = filteredWaveform(1+delay:end)*sqrt(decimationFactor);

% Recover the data bits
if any(strcmp(awnSignalType,["LE1M","LE2M","LE500K","LE125K"]))
    rxCfg.ChannelIndex = channelIndex;
    [rxBits,accAddress] = helperBLEPracticalReceiver(filteredWaveform,rxCfg);
    if isempty(rxBits) || ~isequal(accessAddBits,accAddress)
        pktStatus = [];
    end
else
    % Get PHY properties
    rxCfg.WhitenInitialization = awnWaveformConfig.WhitenInitialization;
    [rxBits,~,pktStatus]...
                        = helperBluetoothPracticalReceiver(filteredWaveform,rxCfg);
end
```

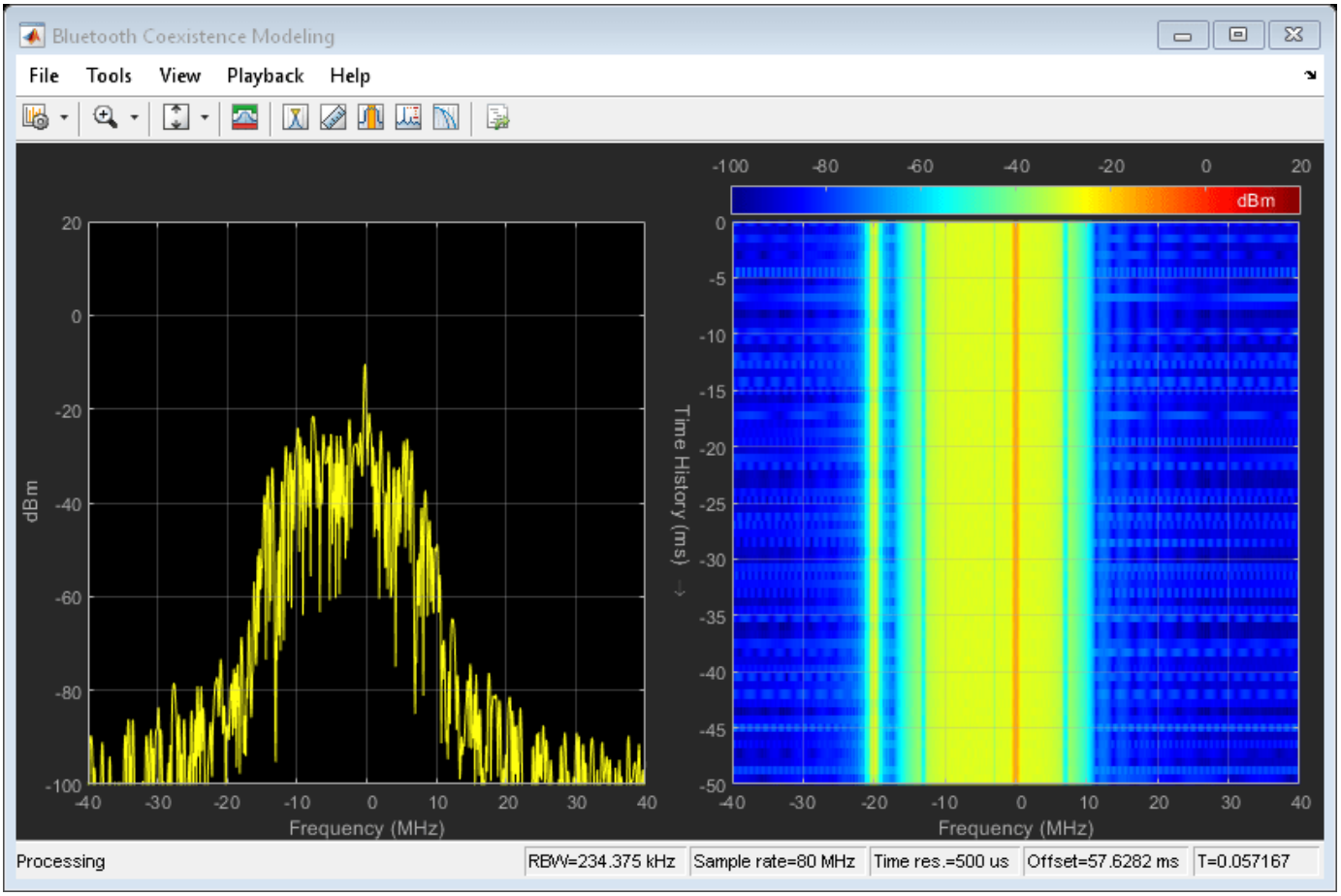
Simulation Results

Compute the BER and PER for each packet. If frequency hopping is on,

- Perform channel classification for every `numPacketsToClassify` based on the PER.
- Compute the SINR for each packet.
- Visualize the Bluetooth BR/EDR or LE waveform with the interference.

```
% Compute BER and PER
lengthTx = length(txBits);
lengthRx = length(rxBits);
lengthMinimum = min(lengthTx,lengthRx)-1;
```

```
        countPreviousPER = countPER;
        if lengthTx && lengthRx
            vectorBER = errorRate(txBits(1:lengthMinimum),rxBits(1:lengthMinimum));
            currentErrors = vectorBER(2)-numErrors;    % Number of errors in current packet
            if currentErrors || (lengthTx ~= lengthRx) % Check if current packet is in error or not
                countPER  = countPER+1;                % Increment the PER count
            end
            numErrors = vectorBER(2);
        elseif ~isempty(pktStatus)
            countPER  = countPER+~pktStatus;           % Increment the PER count
        else
            numPktLost = numPktLost+1;
        end

        % Perform frequency hopping
        if strcmp(awnFrequencyHopping,"On")
            chIdx = channelIndex+1;
            if countPreviousPER ~= countPER
                errorsBasic(chIdx,3) = errorsBasic(chIdx,3)+1;
            end

            % Classify the channels
            if any(inum == (1:floor(numPackets/numPacketsToClassify))*numPacketsToClassify)
                channelMap = errorsBasic(:,3)/numPacketsToClassify > thresholdPER;
                if nnz(channelMap) == 0
                    continue;
                end
                badChannels = find(channelMap)-1;
                if length(frequencyHop.UsedChannels)-length(badChannels) < minChannels
                    errorsBasic(badChannels+1,3) = 0;
                    usedChannels = 0:36;
                else
                    errorsBasic(badChannels+1,3) = 0;
                    usedChannels = setdiff(frequencyHop.UsedChannels,badChannels);
                end
                frequencyHop.UsedChannels = usedChannels;
            end
        end

        % Visualize the spectrum and spectrogram. Compute SINR.
        if strcmp(awnFrequencyHopping,"On") && collisionCount ~= 0
            sinr(inum) = helperBluetoothSINREstimate(snr,awnTxPower,awnFrequency,pathlossdB,iwnConfig
            spectrumAnalyzer(addIWN2AWN)
        elseif (strcmp(awnFrequencyHopping,"Off") && inum < 70) || (strcmp(awnFrequencyHopping,"On")
            if inum == 1
                sinr = helperBluetoothSINREstimate(snr,awnTxPower,awnFrequency,pathlossdB,iwnConfig,
            end
            spectrumAnalyzer(addIWN2AWN)
        end
    end
```




```
% Compute BER and PER
if ~any(strcmp(awnPacket,["ID","NULL","POLL"]))
    if numPackets ~= numPktLost
        per = countPER/(numPackets-numPktLost);
        ber = vectorBER(1);
        fprintf('Mode %s, Simulated for Eb/No = %d dB, Obtained BER: %d, Obtained PER: %d\n',awnS
    else
        fprintf('No Bluetooth packets were detected.\n');
    end
else
    if numPackets ~= numPktLost
        per = countPER/(numPackets-numPktLost);
        fprintf('Mode %s, Simulated for Eb/No = %d dB, Obtained PER: %d\n',awnSignalType,EbNo,per
    else
        fprintf('No Bluetooth packets were detected.\n');
    end
end
```

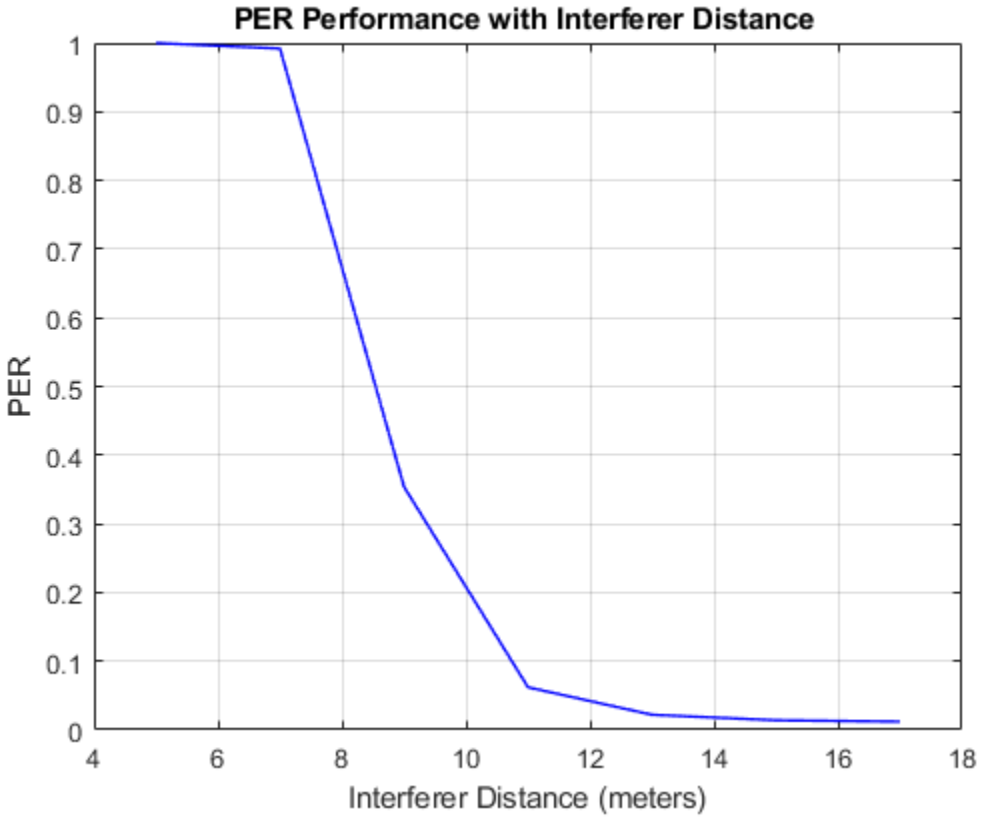
```
Mode LE1M, Simulated for Eb/No = 10 dB, Obtained BER: 4.205257e-04, Obtained PER: 8.980000e-01
```

This example simulates an end-to-end link for Bluetooth BR/EDR and LE with Bluetooth BR/EDR, LE, or WLAN waveforms as interference. You can implement AFH to mitigate interference by classifying channels as good or bad based on the PER value.

Further Exploration

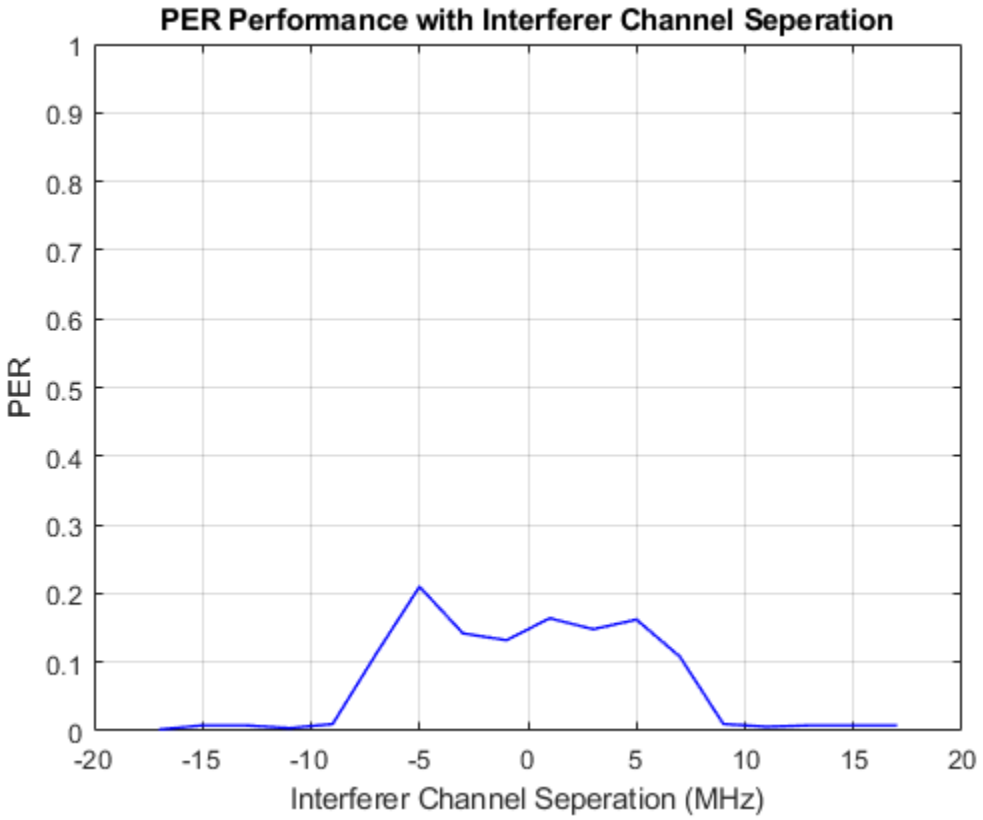
To observe the PER performance with interferer distance, you can run the simulation for different IWN transmitter positions. This plot shows the impact of interferer distance on PER given:

- Bluetooth LE1M AWN, 10 meters distance between the AWN transmitter and receiver, an AWN frequency of 2440 MHz, and an Eb/No value of 10 dB.
- 802.11 g with 20 MHz bandwidth IWN, an IWN frequency of 2437 MHz (co-channel interference), and a collision probability of 1.



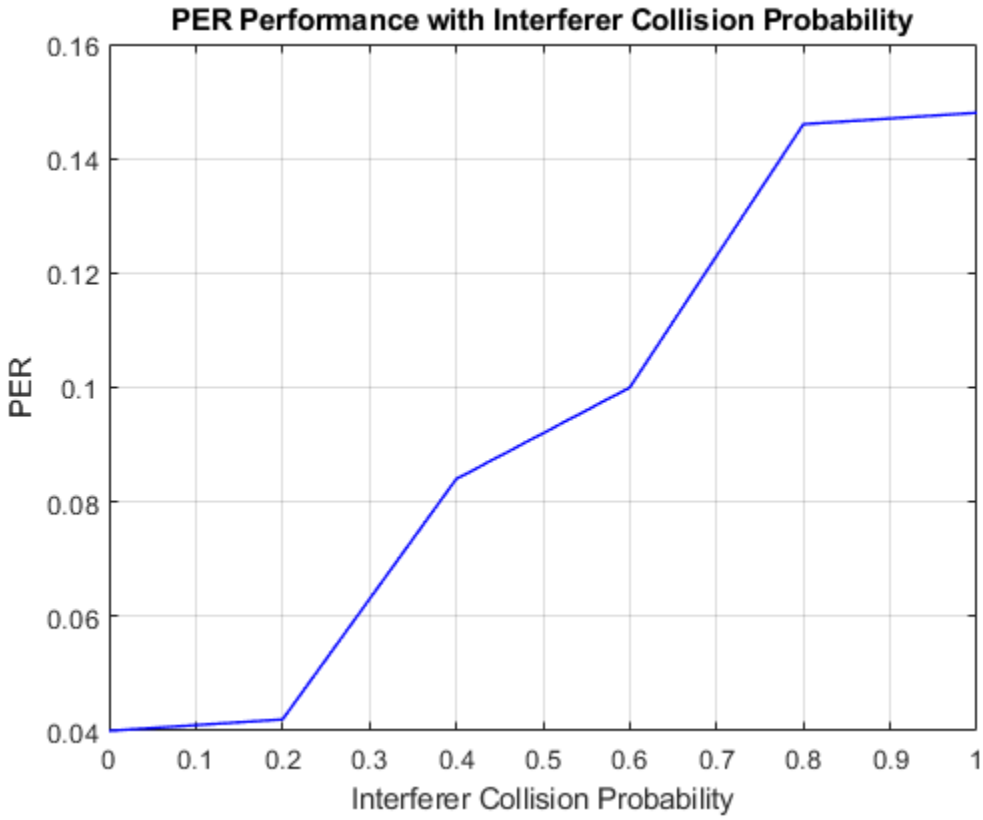
To observe the PER performance with interferer channel separation, you can run the simulation for different IWN frequencies (when frequency hopping is off). This plot shows the impact of interferer channel separation on PER given:

- Bluetooth LE1M AWN, 10 m distance between the AWN transmitter and receiver, and an Eb/No value of 10 dB.
- 802.11g with 20 MHz bandwidth IWN, an IWN frequency of 2437 MHz, a collision probability of 1, and 10 m distance between the AWN and IWN transmitters.



To observe the PER performance with collision probability, you can run the simulation for different collision probabilities. This plot shows the impact of collision probability on PER by considering:

- Bluetooth LE1M AWN, an AWN frequency of 2440 MHz, 10 m between the AWN transmitter and receiver, and an Eb/No value of 10 dB.
- 802.11g with 20 MHz bandwidth IWN, an IWN frequency of 2437 MHz, and 10 m between the AWN and IWN transmitters.



Appendix

The example uses these helper functions:

- helperIWNConfig: Interference wireless node configuration parameters
- helperBLEDelaySignal: Introduce time delay in the signal
- helperBLEFrequencyOffset: Apply frequency offset to the input signal
- helperBLEPracticalReceiver: Demodulate and decode the received signal
- helperBluetoothPracticalReceiver: Detect, synchronize, and decode the received Bluetooth BR/EDR waveform
- helperBluetoothEstimatePathLoss: Estimate the path loss between the node and locator
- helperBluetoothSINREstimate: Estimate SINR

Selected Bibliography

1 Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 15, 2021. https://www.bluetooth.com.

2 Bluetooth Special Interest Group (SIG). "Core System Package [Low Energy Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume https://www.bluetooth.com.

See Also

More About

- “Noncollaborative Bluetooth LE Coexistence with WLAN Signal Interference” on page 2-18
- “Evaluate the Performance of Bluetooth QoS Traffic Scheduling with WLAN Signal Interference” on page 6-54
- “Bluetooth-WLAN Coexistence” on page 8-53
- “Configure Bluetooth BR/EDR Channel with WLAN Interference and Pass the Waveform Through Channel” on page 9-20
- “Bluetooth LE Node Statistics” on page 8-78

Noncollaborative Bluetooth LE Coexistence with WLAN Signal Interference

This example shows how to simulate Bluetooth low energy (LE) noncollaborative coexistence with WLAN interference by using Bluetooth® Toolbox.

Using this example, you can:

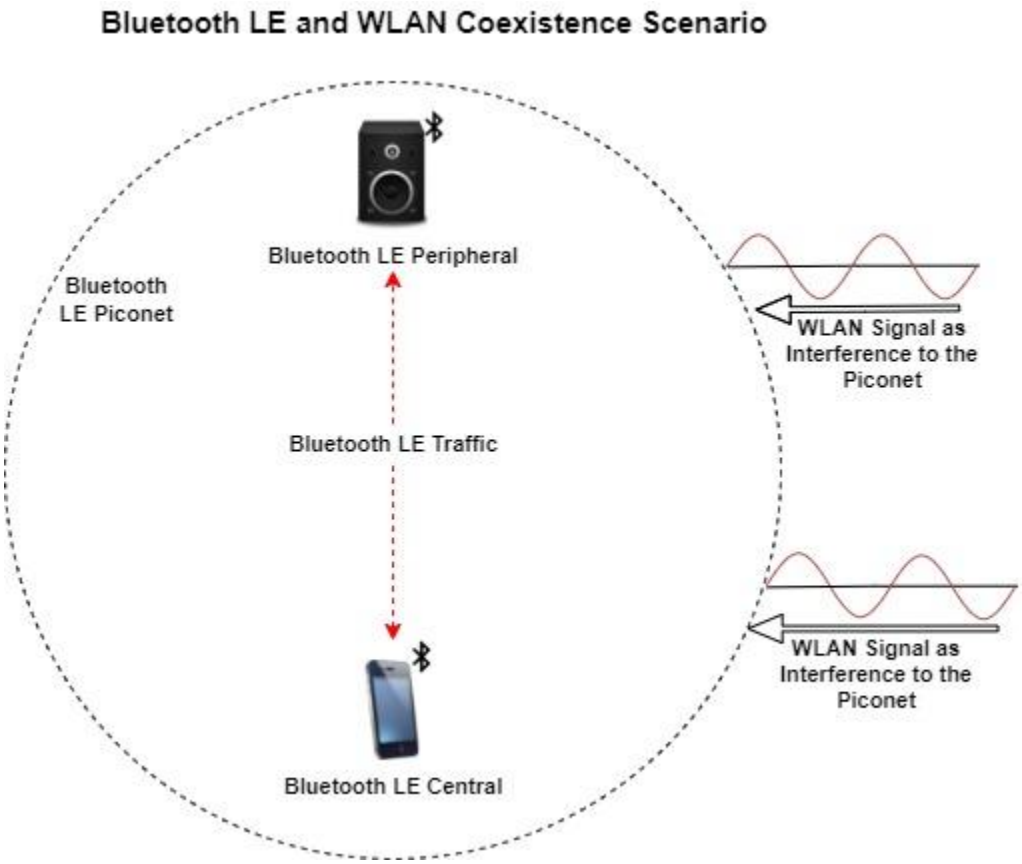
- Create and configure a Bluetooth LE piconet with Central and Peripheral nodes.
- Analyze the performance of the Bluetooth LE network with and without WLAN interference.
- Visualize Bluetooth LE coexistence with WLAN interference for each Peripheral node by implementing adaptive frequency hopping (AFH).
- Visualize the status (good or bad) and success rate (recent and cumulative) of each channel.

The example also enables you to add a custom channel selection algorithm.

Noncollaborative Bluetooth LE-WLAN Coexistence Scenario

Interference between Bluetooth and WLAN can be mitigated by two types of coexistence mechanisms: collaborative and noncollaborative. Noncollaborative coexistence mechanisms do not exchange information between two wireless networks. Collaborative coexistence mechanisms collaborate and exchange network-related information between two wireless networks. These coexistence mechanisms are applicable only after a WLAN or Bluetooth piconet is established and the data is to be transmitted. This example demonstrates a noncollaborative AFH technique deployed between Bluetooth LE and WLAN nodes to mitigate interference. AFH enables a Bluetooth node to adapt to its environment by identifying fixed sources of WLAN interference and excluding them from the list of available channels. For more information about coexistence between Bluetooth LE and WLAN, see “Bluetooth-WLAN Coexistence” on page 8-53 and “Configure Bluetooth BR/EDR Channel with WLAN Interference and Pass the Waveform Through Channel” on page 9-20.

The Bluetooth LE piconet consists of one Bluetooth LE Central node and one Peripheral node. The scenario consists of two WLAN nodes, which introduce interference in the Bluetooth LE signal. The example simulates this coexistence scenario between Bluetooth LE and WLAN.



Configure the Coexistence Scenario

For reproducibility, set the default seed for the random number generator. The seed value controls the pattern of random number generation. Initializing the random number generator using the same seed, assures the same result. For high fidelity simulation results, change the seed value and average the results over multiple simulations.

```
rng("default");
```

Create a Bluetooth LE node and set the role to `"central"` by using the `bluetoothLENode` object. Set the properties of the Central node.

```
centralNode = bluetoothLENode("central", ...
    Name="Central Node", ...
    Position=[5 0 0], ...                    % x-, y-, and z-coordinates in meters
    TransmitterPower=0);                     % In dBm
```

Create a Bluetooth LE node and set the role to `"peripheral"`. Set the properties of the Peripheral node.

```
peripheralNode = bluetoothLENode("peripheral", ...
    Name="Peripheral 1", ...
    Position=[10 0 0], ...                           % x-, y-, and z-coordinates in meters
    TransmitterPower=0);                             % In dBm
```

Create a Bluetooth LE configuration object. Set the connection interval, active period, connection offset, and access address for each connection. Connection events are established for every

connection interval duration throughout the simulation. The connection offset specifies the offset from the beginning of the connection interval. The active period specifies the active communication period for a connection after which connection event is ended. Assign the configuration to the Central and Peripheral nodes.

```
connectionConfig = bluetoothLEConnectionConfig;
connectionConfig.ConnectionInterval = 0.01;                          % In seconds
connectionConfig.ActivePeriod = 0.01;                                % In seconds
connectionConfig.ConnectionOffset = 0;                               % In seconds
connectionConfig.AccessAddress = "12345678";                         % In hexadecimal
configureConnection(connectionConfig,centralNode,peripheralNode);
```

Application Traffic

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Configure the On-Off application traffic pattern at the Central and Peripheral nodes by specifying the application data rate, packet size, and on state duration. Attach application traffic from the Central to the Peripheral nodes.

```
central2PeripheralTrafficSource = networkTrafficOnOff(...
    OnTime=inf, ...                                                  % In seconds
    DataRate=150, ...                                                % In Kbps
    PacketSize=20, ...                                               % In bytes
    GeneratePacket=true);
addTrafficSource(centralNode,central2PeripheralTrafficSource, ...
    DestinationNode=peripheralNode.Name);
```

Attach application traffic from the Peripheral to the Central nodes.

```
peripheral2CentralTrafficSource = networkTrafficOnOff(...
    OnTime=inf, ...
    DataRate=150, ...
    PacketSize=20, ...
    GeneratePacket=true);
addTrafficSource(peripheralNode,peripheral2CentralTrafficSource, ...
    DestinationNode=centralNode.Name);
```

WLAN Signal Interference

To add WLAN signal interference, enable the `enableWLANInterference` parameter.

`enableWLANInterference =` true ;

Specify the number of WLAN nodes and their positions in the network. The WLAN nodes introduce interference in the network and do not model the PHY and MAC behavior.

Set the properties of the WLAN nodes. Specify the source of WLAN interference by using the `wlanInterferenceSource` parameter. Use one of these options to specify the source of the WLAN interference.

- `"Generated"` - To add a WLAN toolbox™ signal to interfere with the communication between Bluetooth LE nodes, select this option. For steps showing how to add this signal, see Add WLAN Signal Using WLAN Toolbox Features on page 2-0 .
- `"BasebandFile"` - To add a WLAN signal from a baseband file (`.bb`) to interfere with the communication between Bluetooth nodes, select this option. You can specify the file name using

the `BasebandFile` input argument. If you do not specify the `.bb` file, the example uses the default `.bb` file, `"WLANHESUBandwidth20.bb"`, to add the WLAN signal.

To determine the pathloss of the channel during the transmission, the example uses the distance between the nodes. Create WLAN nodes to introduce interference in the network by using the helperInterferingWLANNode helper object.

```
if enableWLANInterference
    wlanInterferenceSource = BasebandFile;
    numWLANNodes = 2;
    wlanNodePositions = [0 7 5; 0 3 0];                          % x-, y-, and z-coordinates in me
    wlanCenterFrequency =  [2.412e9; 2.442e9];                   % Center frequency (in Hz) based
    wlanNodes = cell(1,numWLANNodes);
    for wlanIdx=1:numWLANNodes
        wlanNode = helperInterferingWLANNode(...
            WaveformSource=wlanInterferenceSource, ...
            Position=wlanNodePositions(wlanIdx,:), ...
            Name="WLAN node", ...
            TransmitterPower=20, ...                             % In dBm
            CenterFrequency=wlanCenterFrequency(wlanIdx), ...
            Bandwidth = 20e6, ...                                % In Hz
            SignalPeriodicity=0.0029);                           % In seconds
        wlanNodes{wlanIdx} = wlanNode;
    end
end
```

Create Bluetooth LE Network

Create a Bluetooth LE network consisting of the Bluetooth LE nodes and WLAN interfering nodes (if any).

```
nodes = {centralNode peripheralNode};
if enableWLANInterference
    nodes = [nodes wlanNodes];
end
```

Configure Visualization and Channel Classification

Specify the simulation time by using the `simulationTime` variable. Enable the option to visualize the Bluetooth LE coexistence with WLAN and the channel hopping sequence.

```
simulationTime = 0.6;                          % In seconds
enableVisualization = true;
```

To implement channel classification, enable the `enableChannelClassification` parameter.

```
enableChannelClassification = true;
```

Initialize coexistence visualization by using the helperVisualizeCoexistence helper function.

```
if enableVisualization
    coexistenceVisualization = ...
        helperVisualizeCoexistence(simulationTime,nodes,enableChannelClassification);
end
```

To simulate the scenario, initialize the wireless network by using the helperWirelessNetwork helper object.

```
networkSimulator = helperWirelessNetwork(nodes);
```

Schedule Channel Classification

The Bluetooth LE signal transmitted in a particular channel suffers interference from the WLAN signals. A new channel is selected from the channel map pseudo-randomly based on adaptive channel hopping. This example classifies the channel only when channel classification is enabled. For each Peripheral node, the Central node periodically classifies the channels as "good channels" or "bad channels" based on the total packets received and failed in that channel. If the current number of good channels is less than the preferred number of good channels, the example reclassifies all the bad channels as good channels. The Central node maintains a different channel map for each Peripheral node.

This example implements channel classification based on the statistics of each channel. For each Peripheral node, create a channel classification object by using the helperBluetoothLEChannelClassification helper object and schedule the action for individual destinations. The network simulator provides the flexibility to schedule a custom action in the simulation by using the scheduleAction object function of the helperWirelessNetwork helper object. For example, each time you call the simulator, you can schedule an action to plot the state transitions. Specify the function handle, input argument, absolute simulation time, and periodicity of the callback.

Create a function handle to classify the channel by using the classifyChannels object function of the helperBluetoothLEChannelClassification helper object. Schedule the channel classification for the periodicity of the callback by using the scheduleAction object function of the helperWirelessNetwork helper object. To perform a channel classification for the Peripheral node, create and schedule the action for individual destinations. You can modify or rewrite the helperBluetoothLEChannelClassification helper object or the functions of the class to implement a custom channel classification algorithm as discussed in the Add Custom Channel Classification on page 2-0 section.

```
if enableChannelClassification
    classifierObj = helperBluetoothLEChannelClassification(...
        connectionConfig.UsedChannels, ...
        DestinationNode=peripheralNode.Name, ...
        DestinationID=peripheralNode.ID, ...
        PERThreshold=50);
    classifyFcn = @() classifierObj.classifyChannels(centralNode);
    userData = [];                                                          % User data needed
    callAt = 0;                                                             % Absolute simulati
    periodicity = 125e-3;                                                   % In seconds

    scheduleAction(networkSimulator,classifyFcn,userData,callAt,periodicity); % Schedule channel
end
```

Create Listeners

Add a listener for an event by using the `addlistener` function. To update the channel map and status of the channel for each channel map update, create a listener that listens for when the `ChannelMapUpdated` and `PacketReceptionEnded` events are triggered on the Central node object. The source object for the listener is set to the Central node because the channel classification for all the Peripheral nodes happens at the Central node. Update the visualization by using the updateBluetoothLEVisualization object function of the helperVisualizeCoexistence helper object for each Peripheral node in the visualization.

```
if enableVisualization && enableChannelClassification
    addlistener(centralNode,"ChannelMapUpdated", ...
        @(nodeobj,eventdata) updateBluetoothLEVisualization(coexistenceVisualization,nodeobj,ever
    addlistener(centralNode,"PacketReceptionEnded", ...
        @(nodeobj,eventdata) updateBluetoothLEVisualization(coexistenceVisualization,nodeobj,ever
elseif enableVisualization && ~enableChannelClassification
    addlistener(centralNode,"ChannelMapUpdated", ...
        @(nodeobj,eventdata) updateBluetoothLEVisualization(coexistenceVisualization,nodeobj,ever
    addlistener(centralNode,"PacketReceptionEnded", ...
        @(nodeobj,eventdata) updateBluetoothLEVisualization(coexistenceVisualization,nodeobj,ever
end
```

For each packet reception, update the channel information and success rates of the channel by performing these steps.

1. Create a listener that listens for the `ChannelMapUpdated` and `PacketReceptionEnded` events are triggered on the central node object.

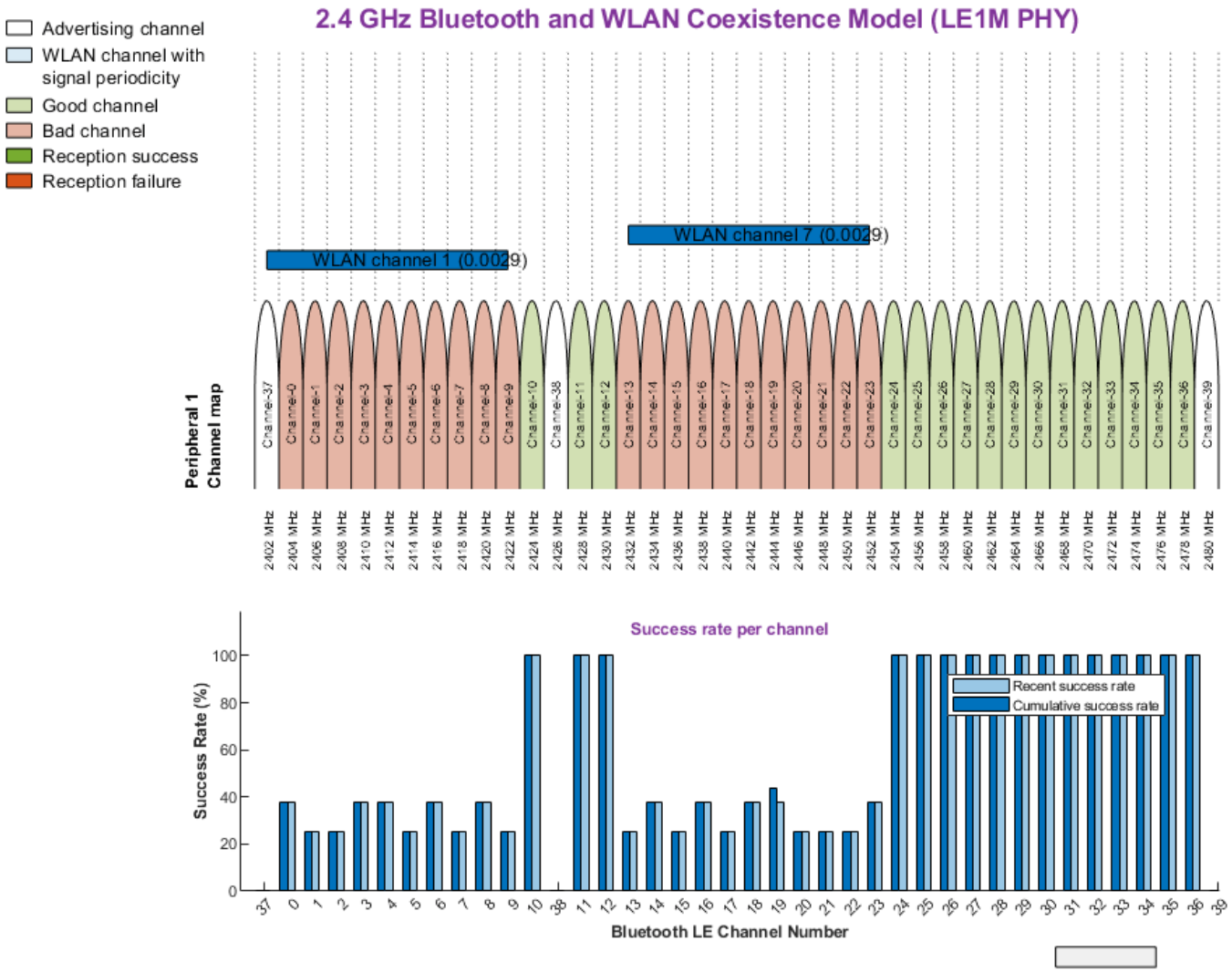
2. Create a callback for the updateRxStatus object function and updateChannelStatus object function of the helperBluetoothLEChannelClassification helper object.

```
if enableChannelClassification
    addlistener(centralNode,"ChannelMapUpdated", ...
        @(nodeobj,eventdata) updateChannelStatus(classifierObj,nodeobj,eventdata));
    addlistener(centralNode,"PacketReceptionEnded", ...
        @(nodeobj,eventdata) updateRxStatus(classifierObj,nodeobj,eventdata));
end
```

Simulation Results

The example runs the simulation for the specified time and displays the channel hopping sequence in the Bluetooth LE channels and the interference caused by the WLAN channels. Visualize the state transitions, status (good or bad), and success rate (recent and cumulative) of each channel. The recent success rate represents the cumulative success rates between each channel classification interval. The overall success rate represents the cumulative success rate throughout the simulation time.

```
run(networkSimulator,simulationTime);
```



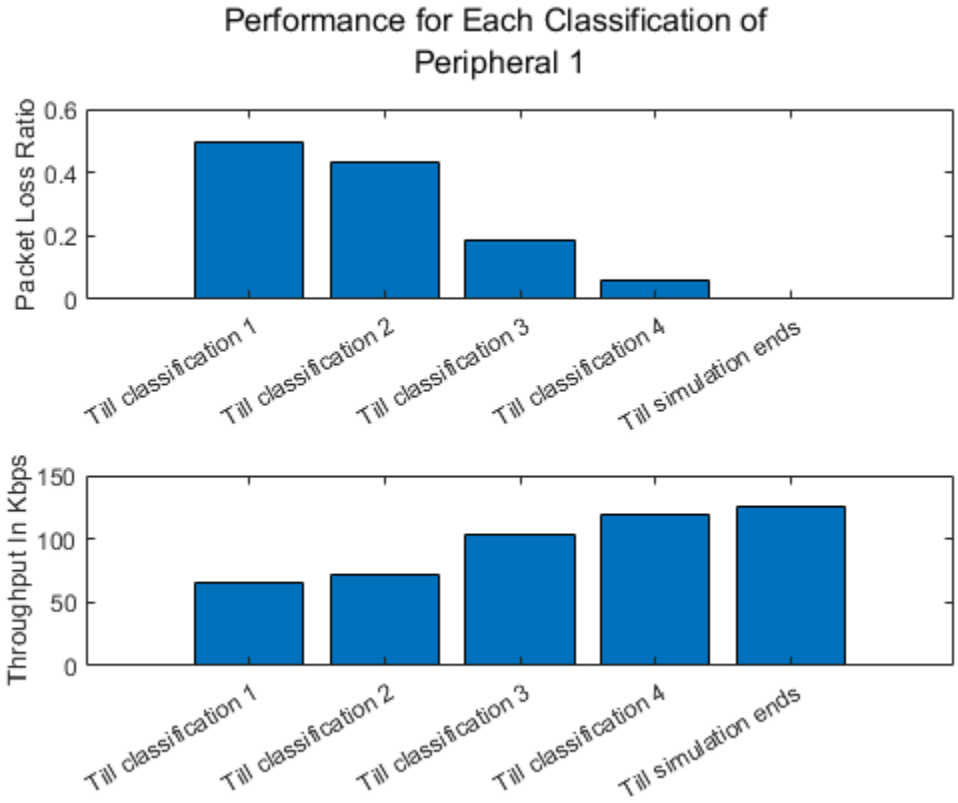
Retrieve Statistics

Retrieve the channel classification statistics by using the classificationStatistics object function of the helperVisualizeCoexistence helper object. Use this object function to visualize the packet loss ratio and throughput between each channel map update for every Peripheral node.

```
if enableChannelClassification
    bluetoothLEChannelStats = classificationStatistics(classifierObj, ...
        centralNode,peripheralNode);
end
```

```
Channel classification statistics of Peripheral 1
                                  Channel 0    Channel 1    Channel 2    Channel 3    Cha
                                  _________    _________    _________    _________    ___

    RxPacketsTillClassification1          0            8            0            8
    RxPacketsFailedTillClassification1    0            6            0            5
    ChannelStatusTillClassification1      1            1            1            1
```

```
RxPacketsTillClassification2            0          0          8          8
RxPacketsFailedTillClassification2      0          0          6          5
ChannelStatusTillClassification2        1          1          1          0
RxPacketsTillClassification3            8          0          0          8
RxPacketsFailedTillClassification3      5          0          0          5
ChannelStatusTillClassification3        1          0          1          0
RxPacketsTillClassification4            8          0          0          8
RxPacketsFailedTillClassification4      5          0          0          5
ChannelStatusTillClassification4        0          0          0          0
RxPacketsTillSimulationEnds             8          0          0          8
RxPacketsFailedTillSimulationEnds       5          0          0          5
ChannelStatusTillSimulationEnds         0          0          0          0
TotalRxPackets                          8          8          8          8
TotalRxPacketsFailed                    5          6          6          5
```



Get the Central and Peripheral node statistics by using the `statistics` object function.

```
centralStats = statistics(centralNode);
peripheralStats = statistics(peripheralNode);
```

The example simulation generates these results.

1. A runtime plot for each Central-Peripheral connection pair showing the status (good or bad) and success rate (recent and cumulative) of each channel.
2. Channel classification statistics, such as the total number of packets received and corrupted and the status (good or bad) for all the channels for each classification interval, are displayed.

3. A bar plot for each peripheral showing the packet loss ratio and throughput between each channel map update is displayed.
4. The statistics at the application layer, link layer, and PHY layer are captured for the Central and Peripheral nodes.

The Bluetooth LE Central and Peripheral nodes avoid the interfered channels through channel classification and communicate with each other to avoid packet loss. The success rate is calculated at each Bluetooth LE channel. This example concludes that for high transmit power of a WLAN channel, the achieved success rate of the respective Bluetooth LE channel is low. Therefore, these channels are not used for communication between Bluetooth LE Central and Peripheral nodes.

Further Exploration

You can use this example to further explore these options:

- Add WLAN signal interference generated using WLAN Toolbox.
- Add a custom channel classification algorithm.
- Add multiple Peripheral nodes in a piconet.

Add WLAN Signal Using WLAN Toolbox Features

To add a WLAN signal using WLAN Toolbox features, follow these steps:

Set the value of `WaveformSource` parameter of the helperInterferingWLANNode to "`Generated`".

```
% wlanNode = helperInterferingWLANNode;
% wlanNode.WaveformSource = "Generated";
```

Create a WLAN packet format configuration object and assign it to the node.

```
% cfgHT = wlanHTConfig("ChannelBandwidth","CBW40");
% wlanNode.FormatConfig = cfgHT;
```

Set the bandwidth of the signal.

```
% wlanNode.Bandwidth = 40e6;
```

Add Custom Channel Classification

You can add a custom channel classification algorithm by creating a custom channel classification object. Classify the channels by passing the classification function at an absolute simulation time or at a particular periodicity by using the scheduleAction object function. The scheduleActionAfter object function can also be used to schedule an action to process at a specified time after the current simulation time or at a particular periodicity. Instead of scheduling or calling the classification at certain simulation time instances, you can implement a custom channel classification by classifying the channels based on the status of the received packets. Analyze the status of the received packets by using the updateRxStatus object function. Classify the channels based on the status of the received packets by using the classifyChannels object function.

Add Multiple Peripheral Nodes in Piconet

To add multiple Peripheral nodes to a piconet, perform these steps:

1. Set the number of Bluetooth LE Peripheral nodes.
2. Create Peripheral nodes by using the `bluetoothLENode` object with role set to "`peripheral`".

3. Assign the configuration to the Central node and each Peripheral node.
4. Generate and add application traffic at the Central and Peripheral nodes.
5. Enable channel classification at the Central node for each of the peripherals by creating an array of classifier objects.
6. Schedule the action for individual destinations. Retrieve the statistics for all Peripheral nodes.

Appendix

The example uses these helper functions:

- helperInterferingWLANNode - Configure and simulate interfering WLAN node
- helperVisualizeCoexistence - Visualize the coexistence model
- helperWirelessNetwork - Create an object to simulate wireless network
- helperBluetoothLEChannelClassification - Create an object to classify the Bluetooth LE channels

References

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 10, 2021. https://www.bluetooth.com/.
2. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". Version 5.3. https://www.bluetooth.com/
3. IEEE® Standard 802.15.2™. "Coexistence of Wireless Personal Area Networks with Other Wireless Devices Operating in Unlicensed Frequency Bands". *IEEE Recommended Practice for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements; IEEE Computer Society*
4. IEEE P802.11ax™/D3.1. "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 6: Enhancements for High Efficiency WLAN". *Draft Standard for Information technology - Telecommunications and information exchange between systems Local and metropolitan area networks - Specific requirements; LAN/MAN Standards Committee of the IEEE Computer Society*
5. IEEE Std 802.11™. "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications". *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements; LAN/MAN Standards Committee of the IEEE Computer Society*

See Also

Functions

`addTrafficSource` | `statistics` | `configureConnection`

Objects

`bluetoothLENode` | `bluetoothLEConnectionConfig`

More About

- “PHY Simulation of Bluetooth BR/EDR, LE, and WLAN Coexistence” on page 2-2
- “Evaluate the Performance of Bluetooth QoS Traffic Scheduling with WLAN Signal Interference” on page 6-54
- “Bluetooth-WLAN Coexistence” on page 8-53

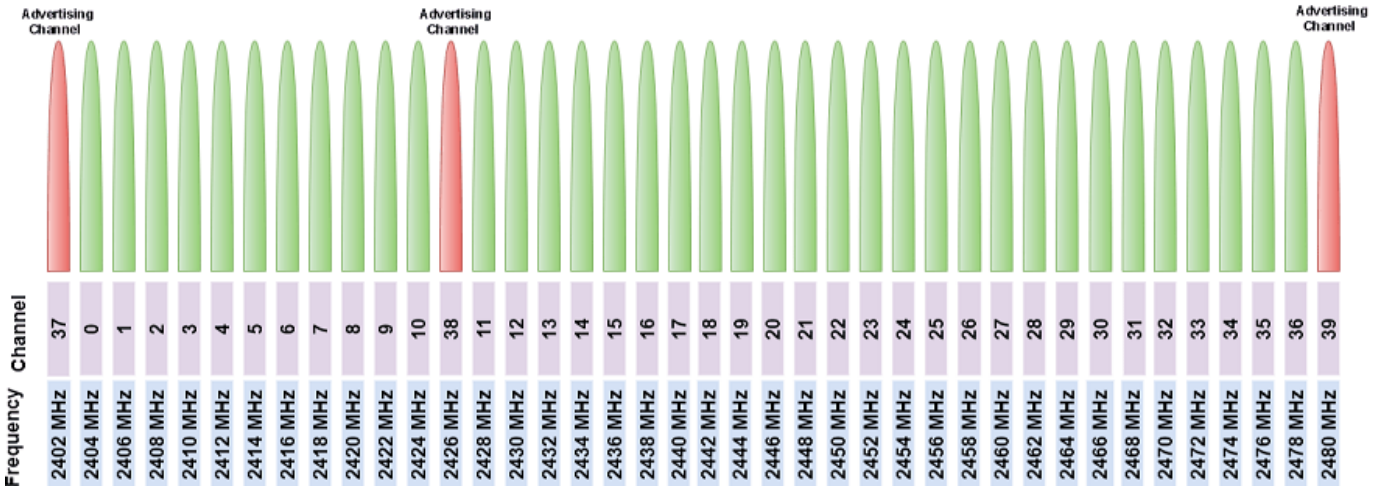
- “Configure Bluetooth BR/EDR Channel with WLAN Interference and Pass the Waveform Through Channel” on page 9-20
- “Bluetooth LE Node Statistics” on page 8-78

Bluetooth LE Channel Selection Algorithms

This example shows you how to select a channel index using the channel selection algorithms specified in the Bluetooth Core Specification [2 on page 2-0] using Bluetooth® Toolbox.

Bluetooth LE Channels

The Bluetooth LE system operates in the 2.4 GHz ISM band at 2400 - 2483.5 MHz. It uses forty RF channels (each channel is 2 MHz wide). The figure below shows the mapping between the frequencies and Bluetooth LE channels. Each of these RF channels is allocated a unique channel index (labeled as "Channel" in the figure).



Bluetooth LE categorizes these forty RF channels into three advertising channels (channel indices: 37, 38, 39) and thirty-seven data channels (channel indices: 0 to 36). Note that the advertising channels are interspersed across the 2.4 GHz spectrum. The purpose of this wide spacing is to avoid interference from other devices operating in the same spectrum, such as WLAN. Advertising channels are mainly used for transmitting advertising packets, scan request/response packets and connection indication packets. Data channels are mainly used for exchanging data packets.

Channel Hopping

Channel hopping is used in Bluetooth to reduce interference and improve throughput. The Bluetooth standard defines rules for switching between channels and algorithms used when performing channel hopping.

Use of the unlicensed 2.4GHz ISM band by several wireless technologies causes increased interference and results in retransmissions to correct errors in received packets. Since Bluetooth LE is a low energy oriented protocol, it is more susceptible to interference. Bluetooth LE uses channel hopping to combat the impact of interference. When one channel is completely blocked due to interference, devices can still continue to communicate with each other on other channels.

In Bluetooth BR/EDR, channel hopping is restricted to 1600 frequency hops/sec. For Bluetooth LE, the channel hopping specification has been revised. Different rules apply for advertising and connected devices, and two channel selection algorithms are defined.

An advertising device transmits advertising packets on the three advertising channels in a cyclic manner (starting from channel index 37). The same procedure is used by the scanning/initiating device, listening on the three advertising channels in a cyclic manner.

A connected device changes to a new data channel for every connection event. A connection event is a sequence of data packet exchanges between two connected devices. The connection events occur periodically with an interval called connection interval. All the packets within a connection event are transmitted on the same data channel. A new connection event uses a new data channel.

Two alternative channel selection algorithms are specified by the Bluetooth Core Specification (see Section 4.5.8, Part-B, Vol-6 of [2 on page 2-0]) can be used to select data channels for each connection event:

1. Algorithm #1
2. Algorithm #2

The two channel selection algorithms avoid channels that are prone to transmission errors. A channel map is exchanged between the central and peripheral devices. This map indicates the good and bad data channels. The classification of good and bad data channels is implementation dependent and can be done based on various parameters like SNR (Signal-To-Noise Ratio), PER (Packet Error Rate), etc. Only the good data channels are used for communication between devices. The channel map will be updated by the central device if it recognizes any bad data channels. The two channel selection algorithms use the channel map to determine whether the selected data channel is good to use. If the selected data channel turns out to be bad, a new data channel is selected using channel remapping procedure (see Section 4.5.8, Part-B, Vol-6 of [2 on page 2-0]), which remaps the bad data channel to one of the good data channels. Each algorithm has a remapping procedure of its own.

Simulating Algorithm #1

You can use `bleChannelSelection` System object to select a new channel index. This System object configures the fields required for selecting a channel index.

Create a System object for 'Algorithm #1'

To select a channel index, create a `bleChannelSelection` System object with `Algorithm` set to 1.

```
csa = bleChannelSelection('Algorithm', 1);
```

Configure the fields.

- The `HopIncrement` property defines the hop increment count to be used. The default value is 5. This property is applicable for 'Algorithm #1'.
- The `UsedChannels` property defines the list of used (good) data channels.

```
csa.HopIncrement = 8;
csa.UsedChannels = [0, 5, 13, 9, 24, 36]
```

```
csa =
  bleChannelSelection with properties:

       Algorithm: 1
    HopIncrement: 8
    UsedChannels: [0 5 9 13 24 36]
    ChannelIndex: 0
    EventCounter: 0
```

- `ChannelIndex` is a read-only property that indicates the current channel being used.
- `EventCounter` is a read-only property that indicates the number of connection events occurred until now. It is incremented for every new selected channel.

Select a channel index for next hop

Call the object `csa` as a function to determine the next channel hop and to select a new channel for each new connection event.

```
nextChannel = csa();
fprintf('Selected channel for connection event %d using ''Algorithm #1'' is: %d\n', csa.EventCou
```

```
Selected channel for connection event 0 using 'Algorithm #1' is: 9
```

Simulating Algorithm #2

You can use `bleChannelSelection` System object to select a new channel index. This System object configures the fields required for selecting a channel index.

Create a System object for 'Algorithm #2'

To select a channel index, create a `bleChannelSelection` System object with `Algorithm` set to 2.

```
csa = bleChannelSelection('Algorithm', 2);
```

Configure the fields.

- The `AccessAddress` property defines the 32-bit unique connection address between two devices. The default value is '8E89BED6'. This property is applicable for 'Algorithm #2'.
- The `UsedChannels` property defines the list of used (good) data channels.

```
csa.AccessAddress = 'E89BED68';
csa.UsedChannels = [9, 10, 21, 22, 23, 33, 34, 35, 36]
```

```
csa =
  bleChannelSelection with properties:

                  Algorithm: 2
              AccessAddress: 'E89BED68'
    SubeventChannelSelection: false
                UsedChannels: [9 10 21 22 23 33 34 35 36]
                ChannelIndex: 0
                EventCounter: 0
```

Select a channel index for next hop

Call the object `csa` as a function to determine the next channel hop and to select a new channel for each new connection event.

```
nextChannel = csa();
fprintf('Selected channel for connection event %d using ''Algorithm #2'' is: %d\n', csa.EventCou
```

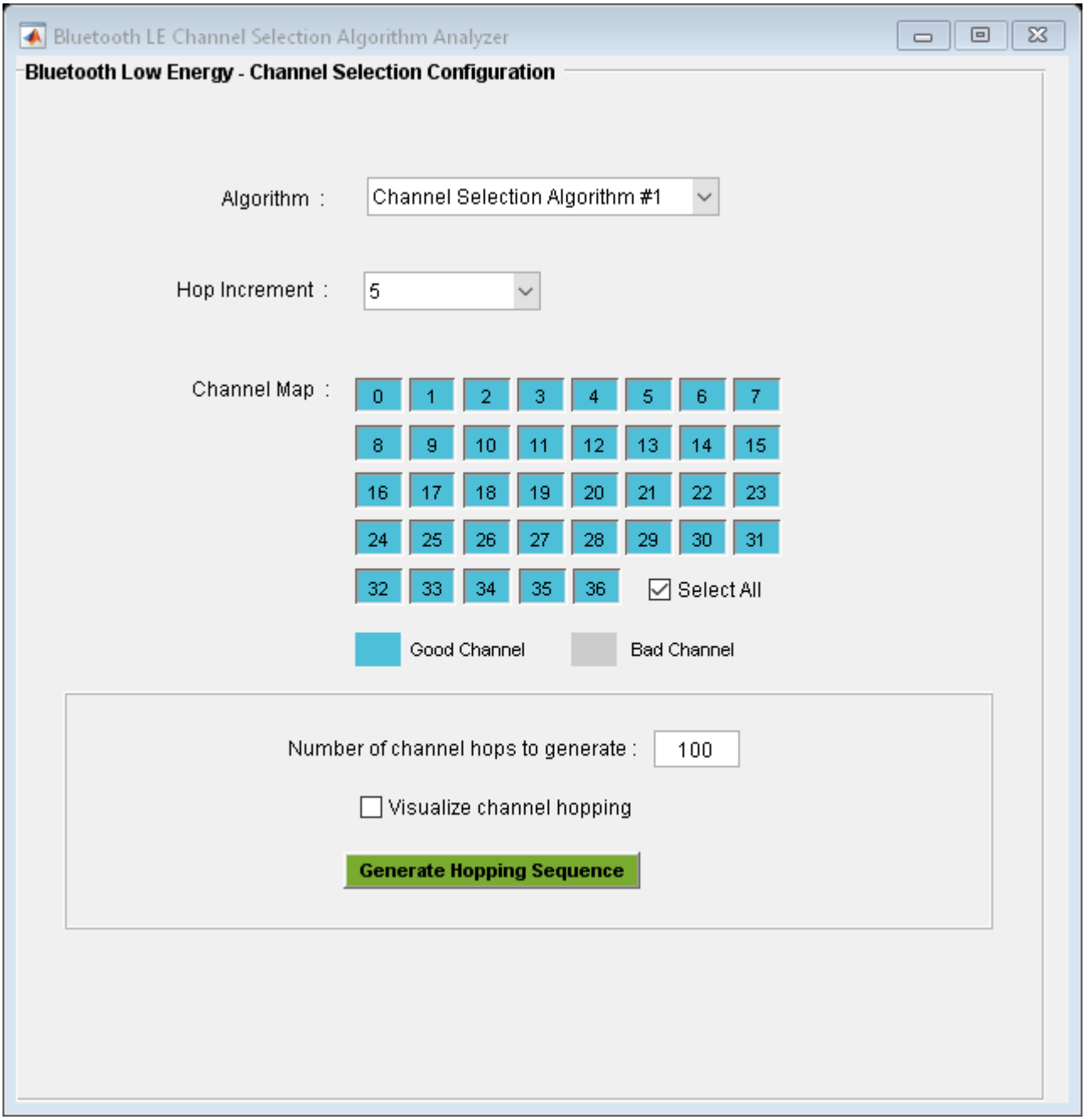
```
Selected channel for connection event 0 using 'Algorithm #2' is: 22
```

GUI for analyzing Channel Selection Algorithms

The function helperBLEChannelHopSelectionUI provides a graphical user interface to generate desired number of channel hops for analyzing the algorithm. Both channel selection algorithms can

be analyzed using this GUI. It can be used to plot the channel hopping pattern of an algorithm and also plots the corresponding histogram.

```
helperBLEChannelHopSelectionUI()
```



Algorithm verification with sample data

Sample data is provided to verify Algorithm #2 (see Section 3, Vol 6, Part C in [2 on page 2-0]). However, there is no sample data available for verifying Algorithm #1.

Sample data 1 (thirty-seven good data channels)

1. Access Address = 8E89BED6
2. Used Channels = [0:36]

When the above inputs are used, the Algorithm #2 is expected to select the following channels according to Section 3.1, Part-B, Vol-6 of [2 on page 2-0]

EventCounter	Channel
1	20
2	6
3	21

The following code selects three channels for the first three connection events.

```
% Create a System object for 'Algorithm #2'
csa = bleChannelSelection('Algorithm', 2);
```

Configure the fields with sample data #1.

```
% Connection access address
csa.AccessAddress = '8E89BED6';

% Use 37 good data channels as used channels according to the sample data
csa.UsedChannels = (0:36);
```

Select channel indices for first 4 connection events. Verify the generated outputs with the table mentioned above.

```
numConnectionEvents = 4;
for i = 1:numConnectionEvents
    channel = csa();
    fprintf('Event Counter: %d, selected Channel: %d\n', csa.EventCounter, channel);
end
```

```
Event Counter: 0, selected Channel: 25
Event Counter: 1, selected Channel: 20
Event Counter: 2, selected Channel: 6
Event Counter: 3, selected Channel: 21
```

Sample data 2 (nine good data channels)

1. Access Address = 8E89BED6
2. Used Channels = [9, 10, 21, 22, 23, 33, 34, 35, 36]

When the above inputs are used, the Algorithm #2 is expected to select the following channels according to Section 3.2, Part-B, Vol-6 of [2 on page 2-0]. Since the channel map contains bad channels, the channel remapping procedure used in the algorithm is also verified.

EventCounter	Channel
6	23
7	9
8	34

The following code selects eight channels for the first eight connection events.

```
% Create a System object for 'Algorithm #2'
csa = bleChannelSelection('Algorithm', 2);
```

Configure the fields with sample data #2.

```
% Connection access address
csa.AccessAddress = '8E89BED6';

% Use 9 good data channels as used channels according to the sample data
csa.UsedChannels = [9, 10, 21, 22, 23, 33, 34, 35, 36];
```

Select channel indices for first 9 connection events. Verify the generated outputs with the table mentioned above.

```
numConnectionEvents = 9;
for i = 1:numConnectionEvents
    channel = csa();
    fprintf('Event Counter: %d, selected Channel: %d\n', csa.EventCounter, channel);
end
```

```
Event Counter: 0, selected Channel: 35
Event Counter: 1, selected Channel: 9
Event Counter: 2, selected Channel: 33
Event Counter: 3, selected Channel: 21
Event Counter: 4, selected Channel: 34
Event Counter: 5, selected Channel: 36
Event Counter: 6, selected Channel: 23
Event Counter: 7, selected Channel: 9
Event Counter: 8, selected Channel: 34
```

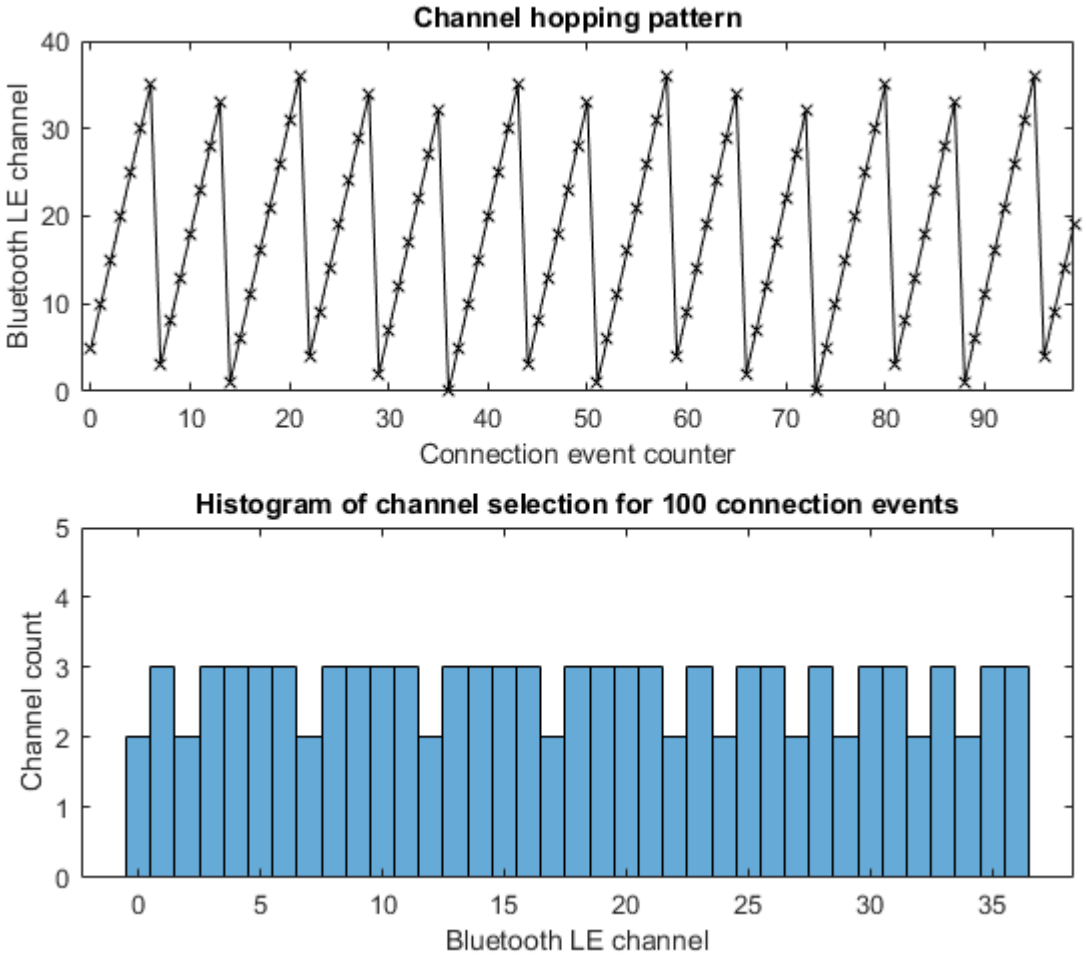
Plot and analyze the hopping pattern - Algorithm #1 and Algorithm #2

The following code selects channel indices for the first hundred connection events using 'Algorithm #1'. The selected channels are plotted and compared with those of 'Algorithm #2'.

```
% Channel selection algorithm System object for 'Algorithm #1'
csa = bleChannelSelection;

% Generate channel hop sequence for 100 connection events
numConnectionEvents = 100;
hopSequence = zeros(1, numConnectionEvents);
for i = 1:numConnectionEvents
    hopSequence(i) = csa();
end
```

The helperBLEPlotChannelHopSequence function plots the hopping pattern and also outputs a histogram of the selected channels.

```
helperBLEPlotChannelHopSequence(csa, hopSequence);
```

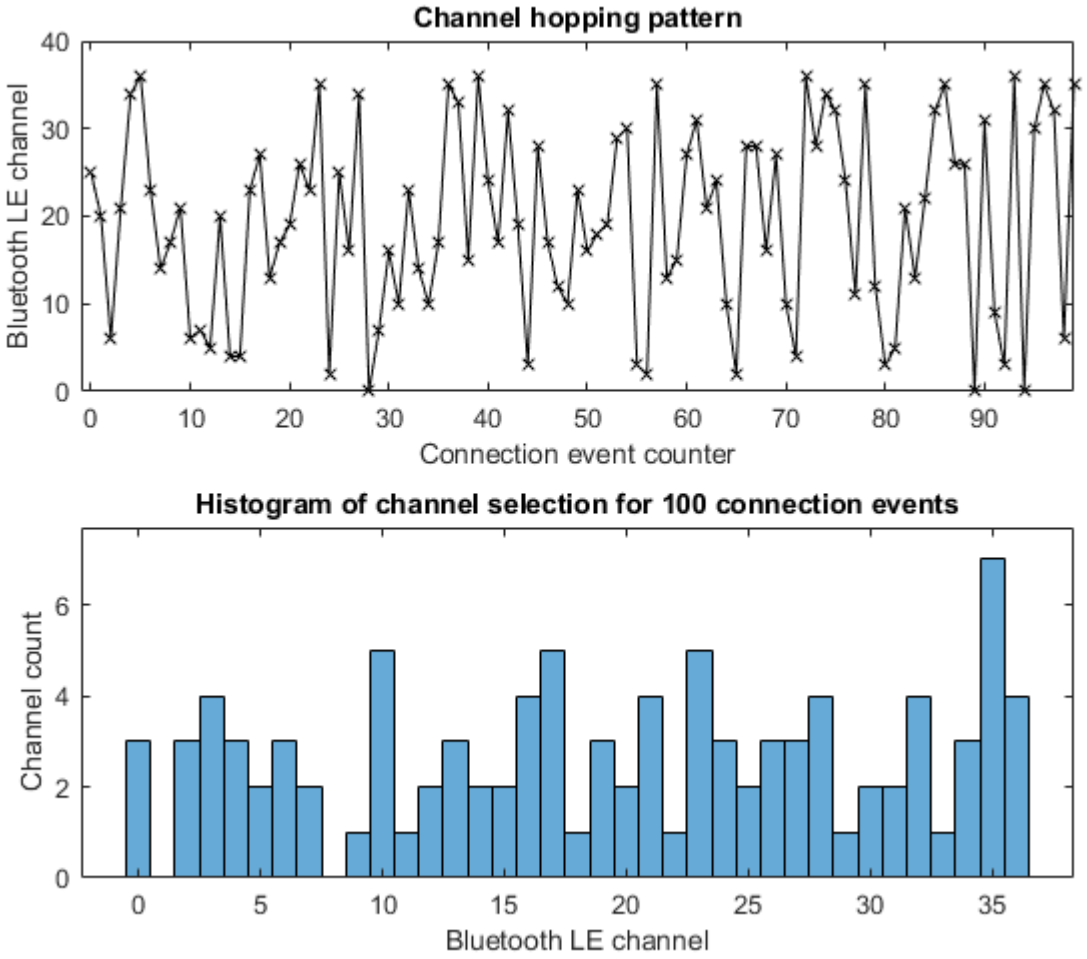


The following code generates channel indices for the first hundred connection events using 'Algorithm #2'. The selected channels are plotted and compared with those of 'Algorithm #1'.

```
% Channel selection algorithm System object for 'Algorithm #2'
csa = bleChannelSelection('Algorithm', 2);

% Generate channel hop sequence for 100 connection events
numConnectionEvents = 100;
hopSequence = zeros(1, numConnectionEvents);
for i = 1:numConnectionEvents
    hopSequence(i) = csa();
end
```

The helperBLEPlotChannelHopSequence function plots the hopping pattern and also outputs a histogram of the selected channels.

```
helperBLEPlotChannelHopSequence(csa, hopSequence);
```



Algorithm #1 vs Algorithm #2

The above plots show the difference between the two algorithms.

- Algorithm #1 is a simple incremental algorithm that produces a uniform sequence of channels. There is no randomization involved in the process of selecting a new channel.
- Algorithm #2 was introduced in version 5.0 of the Bluetooth Core Specification. Compared to Algorithm #1, this is more complex and produces a randomized sequence of channels.

Conclusion

This example demonstrated the behavior of the channel selection algorithms specified in the Bluetooth Core Specification [2 on page 2-0].

Appendix

The example uses these helpers:

- helperBLEChannelHopSelectionUI: Script for helperBLEChannelHopSelectionUI figure

- helperBLEPlotChannelHopSequence: Plot the channel hopping sequence for a given algorithm

Selected Bibliography

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 25, 2021. https://www.bluetooth.com.
2. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Objects

`bleChannelSelection`

Localization

Bluetooth LE Positioning by Using Direction Finding

This example shows you how to calculate the 2-D or 3-D position of a Bluetooth® low energy (LE) node by implementing Bluetooth direction finding features and the triangulation-based location estimation technique by using Bluetooth® Toolbox™. The Bluetooth Core Specification 5.1 [2 on page 3-0] introduced angle of arrival (AoA) and angle of departure (AoD) direction finding features to support centimeter-level accuracy in Bluetooth LE location finding.

Using this example, you can:

- Simulate the direction finding packet exchange between the Bluetooth LE node and each locator to estimate the angles between them.
- Estimate the location of Bluetooth LE node by using triangulation-based location estimation technique in an additive white Gaussian noise (AWGN) channel.
- Measure the positioning accuracy of the Bluetooth LE node related to the bit energy-to-noise density ratio (Eb/No).

Bluetooth LE Localization

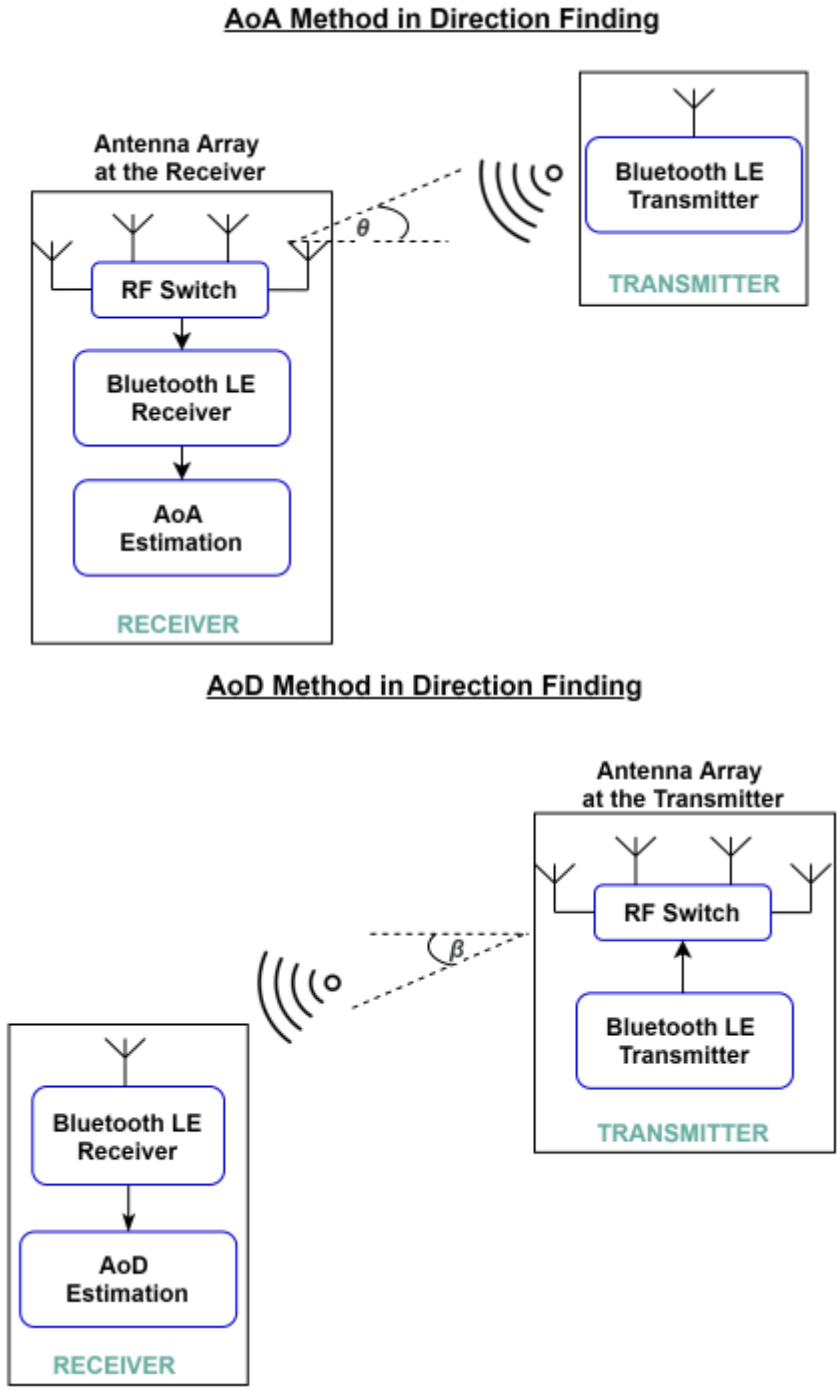
Bluetooth technology provides different types of location based services [1 on page 3-0]. On a high level, these services can be split into two categories.

- Proximity Solutions: To estimate the distance between two devices, the Bluetooth proximity solutions previously used received signal strength indication (RSSI) measurements.
- Positioning Systems: To estimate the position of device, the Bluetooth positioning systems use trilateration based on several RSSI measurements to estimate the position of the device.

Previous versions of Bluetooth provide only meter-level accuracy in estimating the device location. The Bluetooth Core Specification 5.1 [2 on page 3-0] introduced new direction finding features that support centimeter-level accuracy in estimating the location of a device. For more information about direction finding services in Bluetooth LE, see “Bluetooth Location and Direction Finding” on page 8-18.

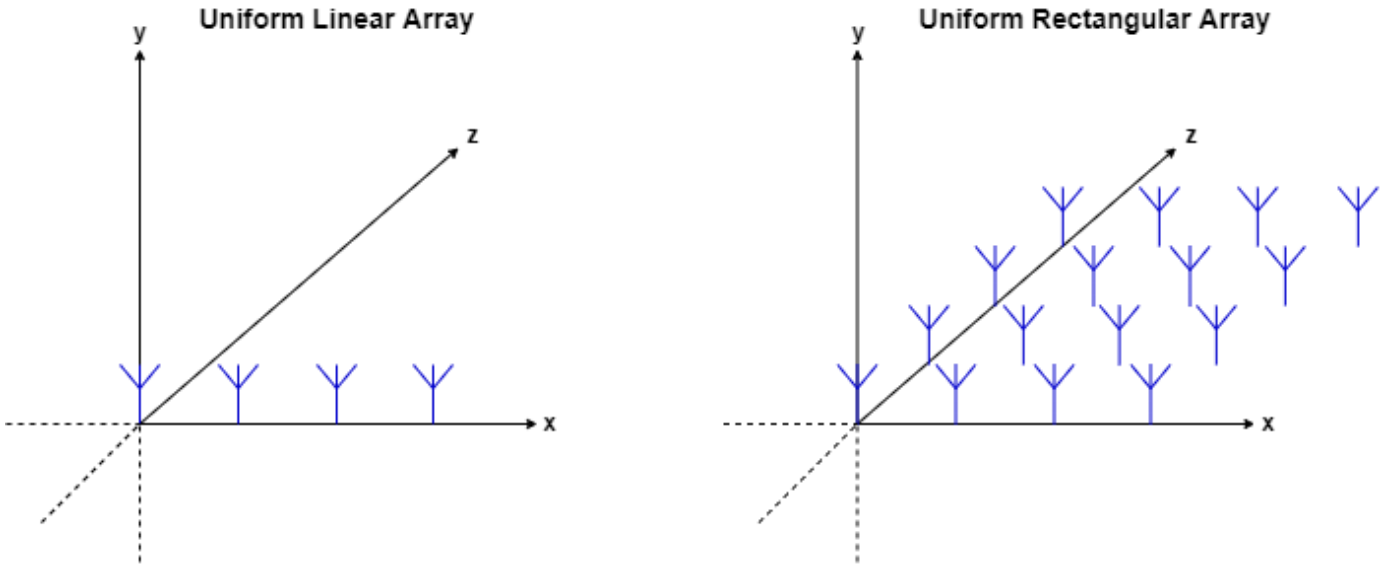
Direction Finding Methods

Bluetooth direction finding provides two distinct methods each of which exploits the same underlying basis. These direction finding methods are AoA and AoD. Each of these techniques require one of the two communicating devices to have an array of multiple antennas. In the AoA and AoD techniques, the antenna array is present at the receiver and transmitter, respectively.

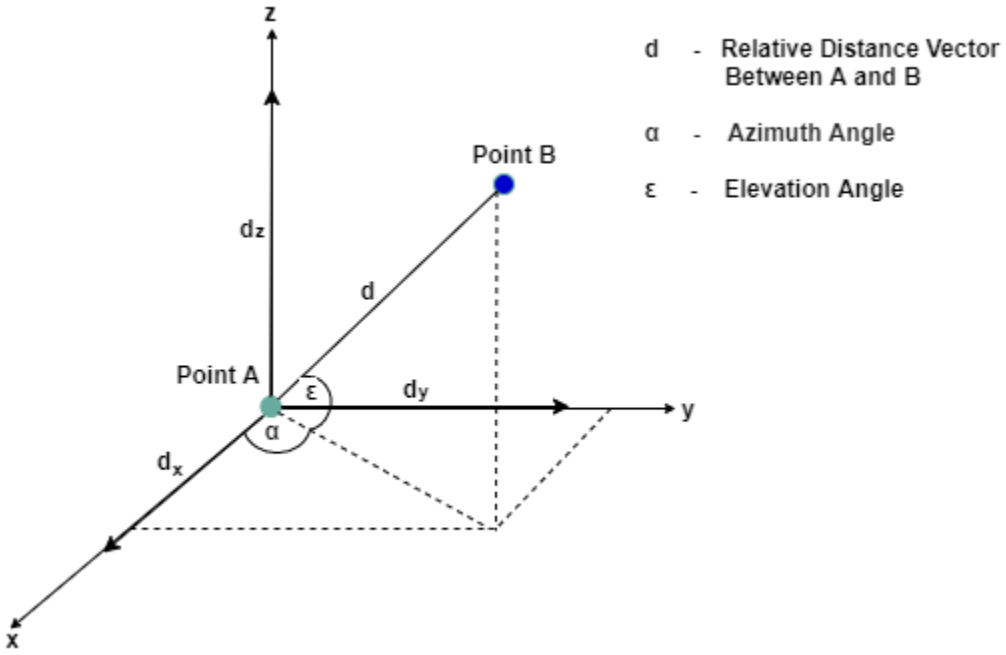


Antenna Arrays

Use a uniform linear array (ULA) or uniform rectangular array (URA) to calculate the direction of a signal. Simple linear designs like ULAs enable you to calculate only azimuth angle from a signal. Two dimensional arrays like URAs enable you to calculate both the azimuth and elevation angles in the 3-D half space. For more information about antenna arrays, see “Bluetooth Location and Direction Finding” on page 8-18.



Calculating the elevation and azimuth angles of the signal relative to a reference plane is common in these antenna arrays. This figure shows the concept of azimuth and elevation angles.



- **Azimuth angle**: This angle is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy*-plane. The angle is positive in going from the *x*-axis toward the *y*-axis.
- **Elevation angle**: This angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy*-plane.

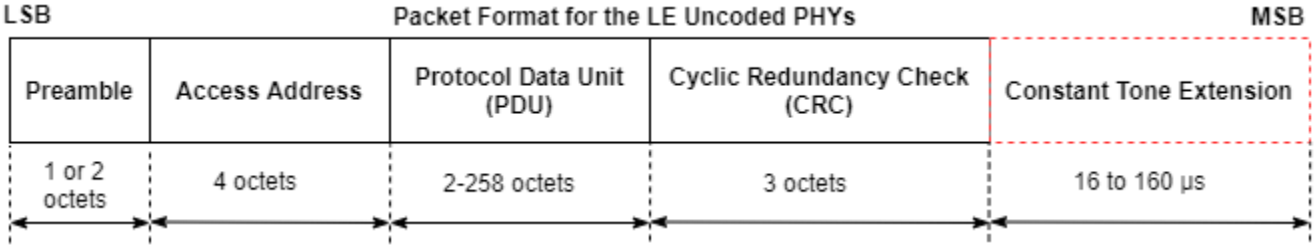
Direction Finding Signals

The communication in Bluetooth LE is realized using one of these two distinct physical layers (PHYs).

- **LE Uncoded**: This PHY is further segregated into the LE1M PHY and LE2M PHY. LE1M is the default PHY and provides a symbol rate of 1 Msym/s. LE2M provides a symbol rate of 2 Msym/s.
- **LE Coded**: This PHY is equipped for long range communication and provides a symbol rate of 1 Msym/s. It has the potential to quadruple the range that can be achieved whilst reducing the data rate.

Bluetooth direction finding can use either the LE1M or LE2M PHY, but not the LECoded PHY.

The Bluetooth Core Specification 5.1 [2 on page 3-0] specifies additional data in the protocol data unit (PDU) packet structure, known as the constant tone extension (CTE) for direction finding. This figure shows the CTE appended at the end of LE uncoded PHY packet.



Use the CTE in any of these communication types.

- **Connection-oriented communication**: It specifies the CTE using the new LL_CTE_RSP PDUs that are sent over the connection in response to the LL_CTE_REQ PDUs.
- **Connectionless communication**: It appends the CTE to the existing periodic advertising PDUs, AUX_SYNC_IND, for direction finding.

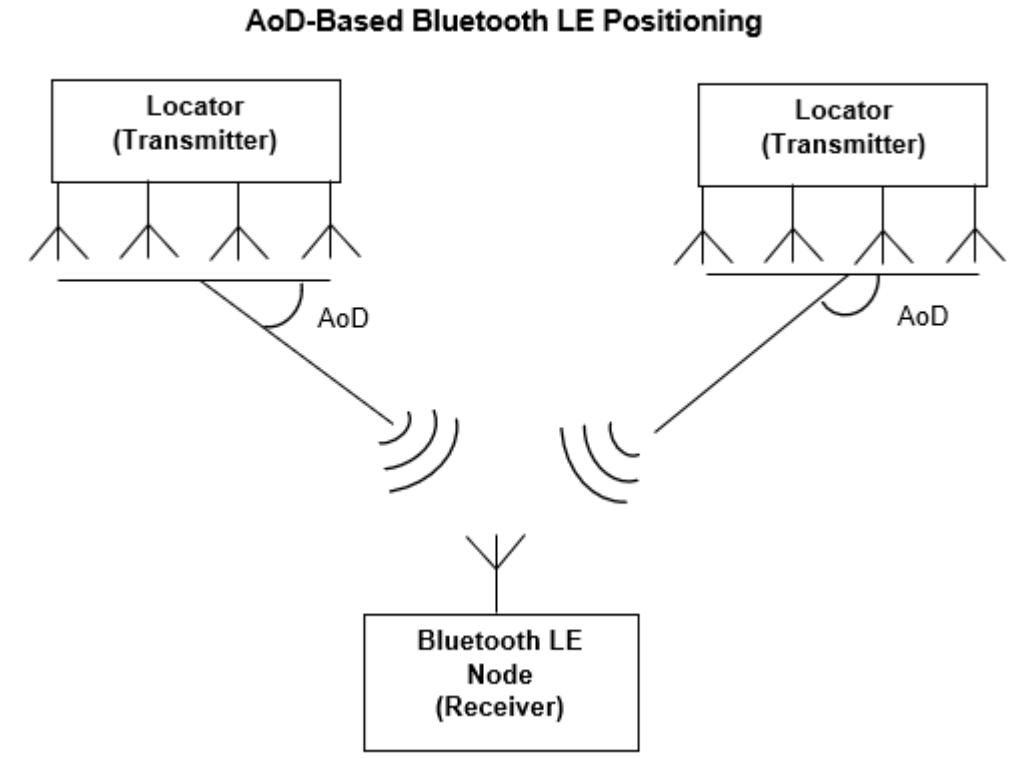
In connection-oriented and connectionless communication, the CTE is appended at the end of the PDU. For information about Bluetooth packet structures, see “Bluetooth Packet Structure”. For more information about the CTE, see volume 6, Part B, Section 2.5.1 of the Bluetooth Core Specification 5.1 [2 on page 3-0].

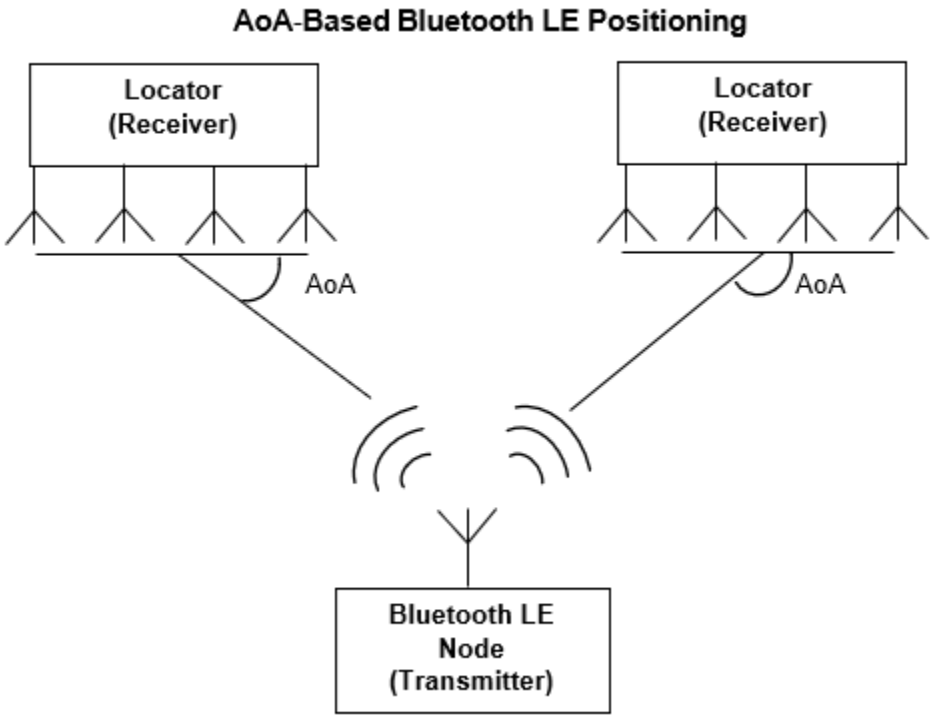
AoA and AoD Based Bluetooth LE Positioning

This example uses these terms:

- *Bluetooth LE node* - Specifies the device whose location is to be determined.
- *Locator* - Specifies the receiving device (in the AoA calculation) and transmitting device (in the AoD calculation).

This figure shows how to estimate the position of a Bluetooth LE node using the AoA and AoD methods.


AoD-Based Bluetooth LE Positioning
Locator
(Transmitter)
Locator
(Transmitter)
AoD
AoD
Bluetooth LE
Node
(Receiver)


AoA-Based Bluetooth LE Positioning
Locator
(Receiver)
Locator
(Receiver)
AoA
AoA
Bluetooth LE
Node
(Transmitter)

In the AoA method, the transmitter (Bluetooth LE node) transmits a direction finding signal using single antenna. The receiving device (locator), equipped with an antenna array, takes the IQ samples while switching between the antennas present in the array. The locator uses the IQ samples to calculate the AoA.

In the AoD method, the transmitting device (locator) is equipped with an antenna array. The transmitting device transmits the signals while switching between the antennas in the array. The receiving device, consisting of a single antenna, collects the IQ samples and calculates the AoD.

To estimate the position of a Bluetooth LE node in 2-D or 3-D, the device requires at least two locators in the network, respectively. Based on the estimated angles and the known Bluetooth LE locator positions, estimate the position of a Bluetooth LE node using the triangulation technique.

Simulation Parameters

Specify the dimension in which the Bluetooth LE node position needs to be determined and the number of Bluetooth LE locators. Specify at least two locators, to estimate the 2-D or 3-D position of a Bluetooth LE node.

```
numDimensions = 2;          % Dimension of Bluetooth LE devices position in a n
numLocators = 3;            % Number of locators
```

Specify the Eb/No range and the number of iterations to simulate each Eb/No point.

```
EbNo = 6:2:16;              % Eb/No in dB
numIterations = 200;        % Number of iterations to average the position erro
```

Specify the direction finding method, the direction finding packet type, and the PHY transmission mode.

```
dfMethod = AoA;             % Direction finding method
dfPacketType = ConnectionCTE;  % Direction finding packet type
phyMode = LE1M;             % PHY transmission mode, must be LE1M or LE2M (for
```

Specify the antenna array parameters.

```
arraySize = 16;             % Antenna array size, must be a scalar (represents
elementSpacing = 0.5;       % Normalized spacing between the antenna elements w
switchingPattern = 1:prod(arraySize);  % Antenna switching pattern, must be a 1xM row vect
```

Specify the waveform generation parameters.

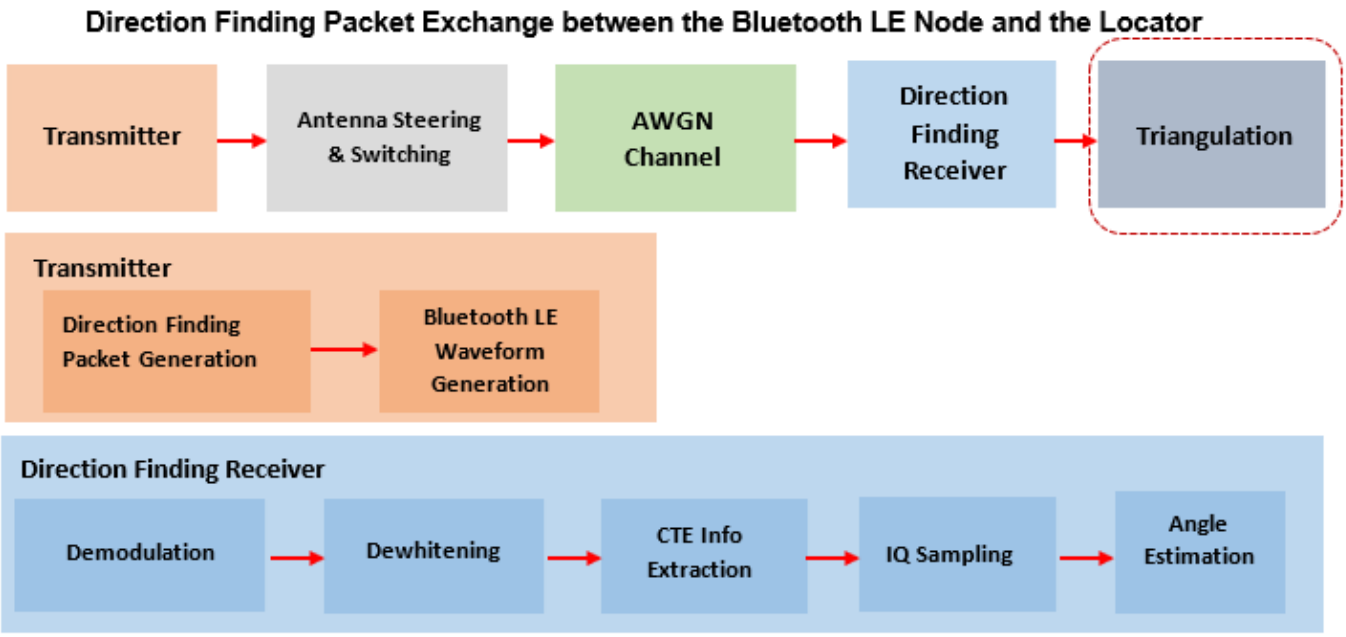
```
slotDuration = 2;           % Slot duration in microseconds
cteLength = 160;            % Length of CTE in microseconds, must be in the ran
sps = 8;                    % Samples per symbol
```

```
channelIndex = 17;    % Channel index
crcInit = '555551';   % CRC initialization
accessAddress = '01234567';   % Access address
payloadLength = 1;    % Payload length in bytes
```

Direction Finding and Position Estimation Procedure

Follow these steps to estimate the Bluetooth LE node position.

1. Position the Bluetooth LE node at the origin. Place the locators randomly in the 2-D or 3-D space.
2. Model the direction finding packet exchange between the Bluetooth LE node and each locator to estimate the angles between them.



a. Generate a direction finding packet for connection or connectionless communication.

b. Generate Bluetooth LE waveform.

c. Perform waveform steering and antenna switching.

d. Add AWGN to the waveform.

e. Perform demodulation, decoding, and IQ sampling on the noisy waveform.

f. Estimate the angle(s) between the Bluetooth LE node and each locator using the IQ samples.

3. Estimate the Bluetooth LE node position by performing triangulation using the known locator positions and the estimated angles.

For each iteration, assign random positions to the locators over a range of Eb/No points, and then repeat the preceding steps.

If you install "Parallel Computing Toolbox"™ to increase the speed of simulation, use a `parfor` loop by commenting-out the `for` statement and uncommenting the `parfor` statement in this code. The `parfor` loop processes each Eb/No point in parallel to reduce the total simulation time. If "Parallel Computing Toolbox"™ is not installed, by default the example switches to the `for` statement.

Validate the simulation parameters.

```
minLocators = 2; % Minimum number of locators
if numDimensions == 2 && size(arraySize,2) ~= 1
    error('The arraySize must be a scalar for 2-D position estimation');
end
if numDimensions == 3 && size(arraySize,2) ~= 2
    error('The arraySize must be a 1-by-2 vector for 3-D position estimation');
end
if numLocators < minLocators
    error(['The numLocators must be greater than or equal to ' num2str(minLocators)]);
end
if strcmp(dfPacketType,'ConnectionCTE') && payloadLength ~= 1
    error('The payloadLength must be 1 byte for direction finding packet type of ConnectionCTE')
elseif strcmp(dfPacketType,'ConnectionlessCTE') && payloadLength < 3
    error('The payloadLength must be greater than or equal to 3 bytes for direction finding packe
end
```

Create and configure Bluetooth LE angle estimation configuration object. Derive the type of CTE based on the slot duration and the direction finding method.

```
if numDimensions == 3 && isscalar(elementSpacing)
    elementSpacing = [elementSpacing elementSpacing]; % Element spacing must be a vector to perfo
end
cfg = bleAngleEstimateConfig('ArraySize',arraySize,'SlotDuration',slotDuration,'SwitchingPattern
                                    switchingPattern,'ElementSpacing',elementSpacing);
validateConfig(cfg);
if strcmp(dfMethod,'AoA')
    cteType = [0;0];
else
    cteType = [0;1];
    if slotDuration == 1
        cteType = [1;0];
    end
end

% Convert access address in hexadecimal to binary
accessAddBits = int2bit(hex2dec(accessAddress),32,false);

% Initialize the variables to be used outside the parfor loop
numEbNo = numel(EbNo);                                  % Number of Eb/No points
posNode = zeros(numDimensions,numEbNo);
posLocator = zeros(numDimensions,numLocators,numEbNo);
angleEst = zeros(numLocators,numDimensions-1,numEbNo);
posNodeEst = zeros(numDimensions,numEbNo);
validResult = zeros(1,numEbNo);
avgPositionError = zeros(1,numEbNo);
```

Model the direction finding packet exchange between the Bluetooth LE node and each locator to estimate the angles between them.

```
% parfor iEbNo = 1:numEbNo % Use 'parfor' to speed up the simulation
for iEbNo = 1:numEbNo % Use 'for' to debug the simulation

    % Set the random substream index to ensure that each iteration uses a
    % repeatable set of random numbers
    stream = RandStream('combRecursive','Seed',12345);
    stream.Substream = iEbNo;
    RandStream.setGlobalStream(stream);

    % Loop over the number of iterations to average the positioning error.
    % If the successful links between the locators and node are less than
    % the minimum number of locators required to perform triangulation, then
    % the iteration fails.
    posErr = zeros(1,numIterations);
    iterationFailCount = 0;
    for iterCount = 1:numIterations

        % For each iteration, generate random positions for the locators
        [tempPosNode,tempPosLocator,ang] = helperBLEGeneratePositions(numLocators,numDimensions)
        posNode(:,iEbNo) = tempPosNode;
        posLocator(:,:,iEbNo) = tempPosLocator;

        % Loop over the number of locators
        tempAngleEst = zeros(numLocators,numDimensions-1);
        idx = [];
        linkFailFlag = zeros(numLocators,1);
        for i=1:numLocators

            % Generate direction finding packet
            data = helperBLEGenerateDFPDU(dfPacketType,cteLength,cteType,payloadLength,crcInit);

            % Generate Bluetooth LE waveform
            bleWaveform = bleWaveformGenerator(data,'Mode',phyMode,'SamplesPerSymbol',sps,...
                'ChannelIndex',channelIndex,'DFPacketType',dfPacketType,'AccessAddress',accessAdd

            % Perform steering and switching between the antennas
            dfWaveform = helperBLESteerSwitchAntenna(bleWaveform,ang(i,:),...
                                        phyMode,sps,dfPacketType,payloadLength,cfg);

            % Pass the waveform through AWGN channel
            snr = EbNo(iEbNo) - 10*log10(sps); % Signal to noise ratio (SNR)
            noiseWaveform = awgn(dfWaveform,snr,'measured');

            % Pass the noisy waveform to bleIdealReceiver which returns
            % the IQ samples
            [~, ~, iqSamples] = bleIdealReceiver(noiseWaveform,'Mode',phyMode,...
                'SamplesPerSymbol',sps,'ChannelIndex',channelIndex,'DFPacketType',...
                dfPacketType,'SlotDuration',slotDuration);

            % Estimate the angle(s) using the IQ samples. A packet is
            % detected successfully when the minimum number of non-zero IQ
            % samples are present.
            refSampleLength = 8; % Reference samples length
            % IQ samples must contain at least eight samples from reference period and
            % one sample from each antenna
            minIQSamples = refSampleLength+getNumElements(cfg)-1;
            if length(nonzeros(iqSamples)) >= minIQSamples % If packet detection is successful
                tempAngleEst(i,:) = bleAngleEstimate(iqSamples,cfg);
```

```
            else
                linkFailFlag(i) = 1; % If packet detection fails, enable the link fail flag
                idx = [idx i]; %#ok<AGROW> % Store the indices corresponding to the locator-node
            end
        end

        % If the successful node-locator links are greater than the minimum
        % number of locators and all the locator angles are not same, then
        % estimate the node position using triangulation and compute the
        % position error
        if (numLocators-nnz(linkFailFlag)) >= minLocators && ~isequal(tempAngleEst(:,1),repmat(te
            % If any link fails, then assign NaN to the corresponding angle
            % estimates
            posLocatorEbNo = posLocator(:,:,iEbNo);
            tempAngEstTri = tempAngleEst;
            if any(linkFailFlag == 1)
                posLocatorEbNo(:,idx) = [];
                tempAngleEst(idx,:) = NaN(numel(idx),numDimensions-1);
                tempAngEstTri(idx,:) = [];
            end

            % Estimate the node position using triangulation
            posNodeEst(:,iEbNo) = blePositionEstimate(posLocatorEbNo,'angulation',tempAngEstTri.
            angleEst(:,:,iEbNo) = tempAngleEst;

            % Compute the position error
            posErr(iterCount) = sqrt(sum((posNodeEst(:,iEbNo)-posNode(:,iEbNo)).^2));
        else
            iterationFailCount = iterationFailCount + 1; % Count the number of failed links used
        end
    end

    if(iterationFailCount == numIterations) % If all the links fail at a given Eb/No value
        disp(['At Eb/No = ',num2str(EbNo(iEbNo)),' dB, all direction finding packet transmissions
        validResult(iEbNo) = 0; % Disable plot flag for failed links
    else
        avgPositionError(iEbNo) = sum(posErr)/(numIterations-iterationFailCount);
        disp(['At Eb/No = ',num2str(EbNo(iEbNo)),' dB, positioning error in meters = ', num2str(a
        validResult(iEbNo) = 1; % Enable plot flag for successful links
    end
end
```

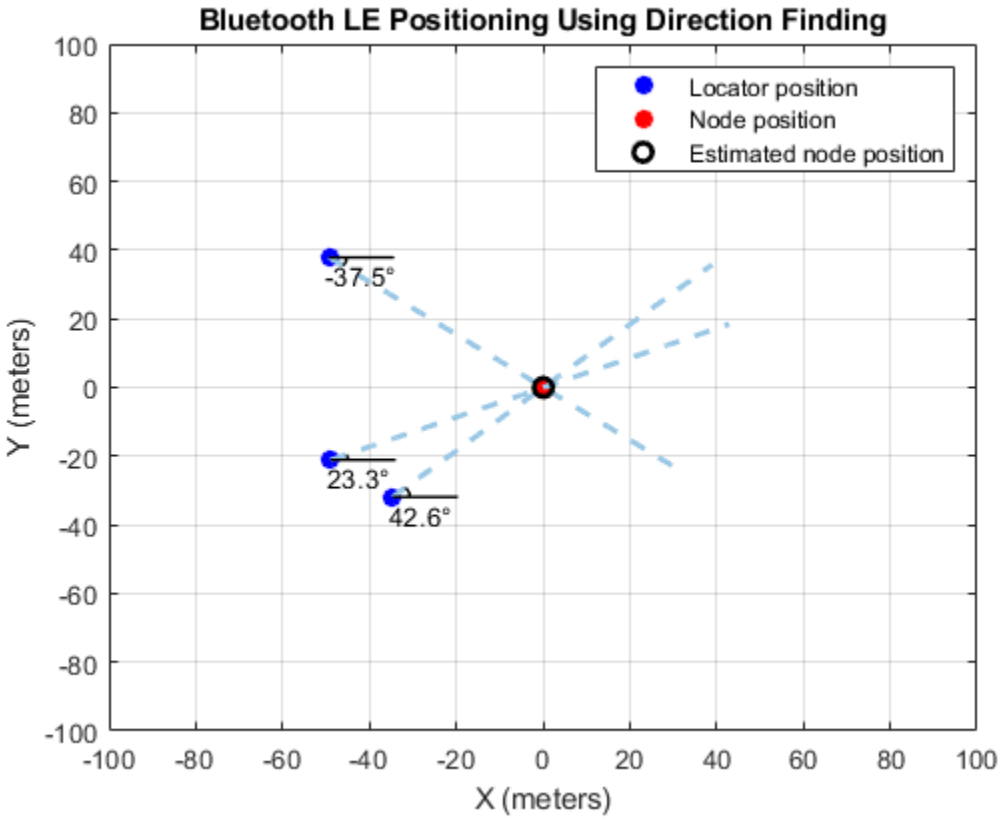
```
At Eb/No = 6 dB, positioning error in meters = 0.5594
At Eb/No = 8 dB, positioning error in meters = 0.43461
At Eb/No = 10 dB, positioning error in meters = 0.31708
At Eb/No = 12 dB, positioning error in meters = 0.27091
At Eb/No = 14 dB, positioning error in meters = 0.21464
At Eb/No = 16 dB, positioning error in meters = 0.17115
```

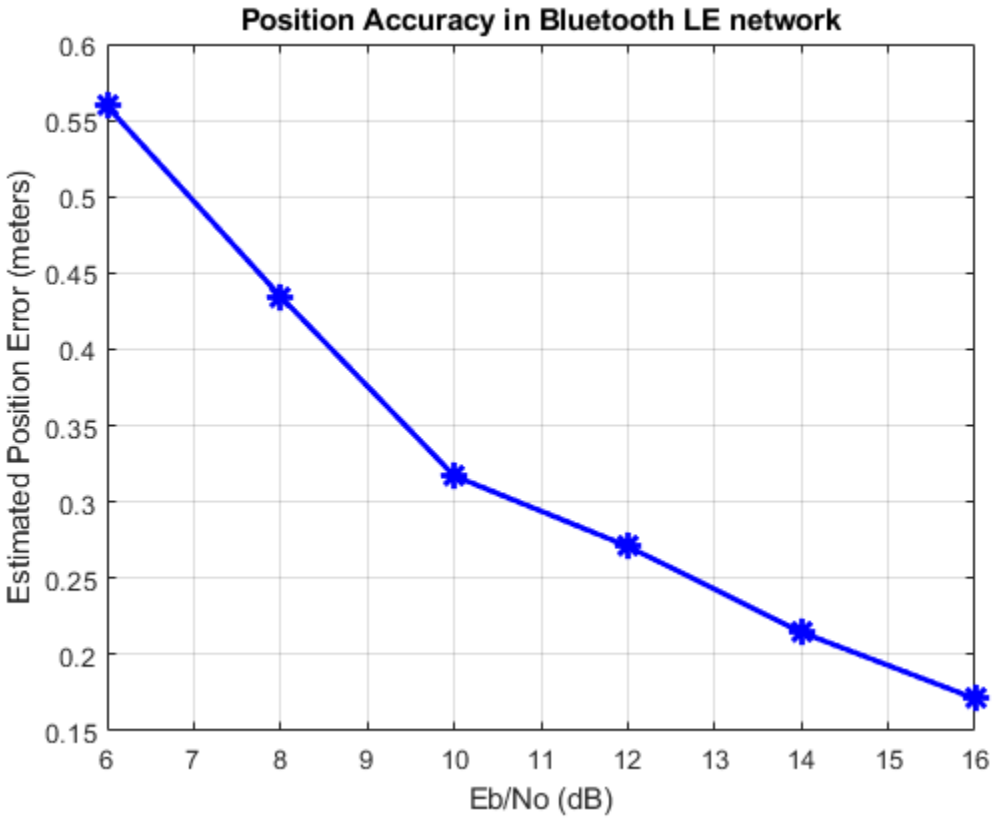
Simulation Results

If the direction finding packet transmission is successful, the example displays these plots.

- Bluetooth LE node position, locators positions, and the estimated node positions in 2-D or 3-D. The plot also shows the triangulation lines for the maximum Eb/No value and the last iteration.
- Average position error (in meters) versus Eb/No (in dB).

```
[~,validIdx] = find(validResult==1);
if ~isempty(validIdx)
    EbNoValid = EbNo(validIdx);
    [~,EbNoIdx] = max(EbNoValid);
    EbNoValidIdx = validIdx(EbNoIdx);
    helperBLEVisualizePosition(posLocator(:,:,EbNoValidIdx),posNode(:,EbNoValidIdx),...
                        angleEst(:,:,EbNoValidIdx),posNodeEst(:,EbNoValidIdx));
    figure
    plot(EbNoValid,avgPositionError(validIdx),'-b*','LineWidth',2, ...
            'MarkerEdgeColor','b','MarkerSize',10)
    grid on
    xlabel('Eb/No (dB)')
    ylabel('Estimated Position Error (meters)')
    title('Position Accuracy in Bluetooth LE network')
end
```





This example enables you to estimate the 2-D or 3-D position of a Bluetooth LE node by using the Bluetooth LE direction finding functionality. The example shows how to implement the triangulation method to calculate the angles between the locators and the Bluetooth LE node. The example also shows how to measure the positioning accuracy related to the Eb/No value by computing the distance between the estimated and actual node positions in an AWGN channel.

Appendix

The example uses these helpers:

- helperBLEGeneratePositions: Generate locators and node positions
- helperBLEGenerateDFPDU: Generate direction finding packet PDU
- helperBLESteerSwitchAntenna: Perform antenna steering and switching
- helperBLEVisualizePosition: Generate Bluetooth LE position visualization

Selected Bibliography

1 Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com.

2 Bluetooth Special Interest Group (SIG). "Core System Package [Low Energy Controller Volume]". Bluetooth Core Specification. Version 5.1, Volume https://www.bluetooth.com.

3 Bluetooth Special Interest Group (SIG). "Core System Package [Low Energy Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume https://www.bluetooth.com.

See Also

Functions
`bleWaveformGenerator` | `bleAngleEstimate` | `blePositionEstimate`

Objects
`bleAngleEstimateConfig`

More About

- “Parameterize Bluetooth LE Direction Finding Features” on page 9-10
- “Estimate Bluetooth LE Node Position” on page 9-35
- “Bluetooth Location and Direction Finding” on page 8-18
- “Bluetooth LE Direction Finding for Tracking Node Position” on page 3-15

Bluetooth LE Direction Finding for Tracking Node Position

This example shows you how to track the 2-D or 3-D position of a Bluetooth® low energy (LE) node by using Bluetooth® Toolbox.

Using this example, you can:

- Simulate direction-finding packet exchange in the presence of radio frequency (RF) front end impairments, path loss, and additive white Gaussian noise (AWGN).
- Track the node position by using Bluetooth direction-finding features and position estimation techniques.
- Improve the location accuracy by using a Kalman filter from Sensor Fusion and Tracking Toolbox™.

Bluetooth LE Direction Finding

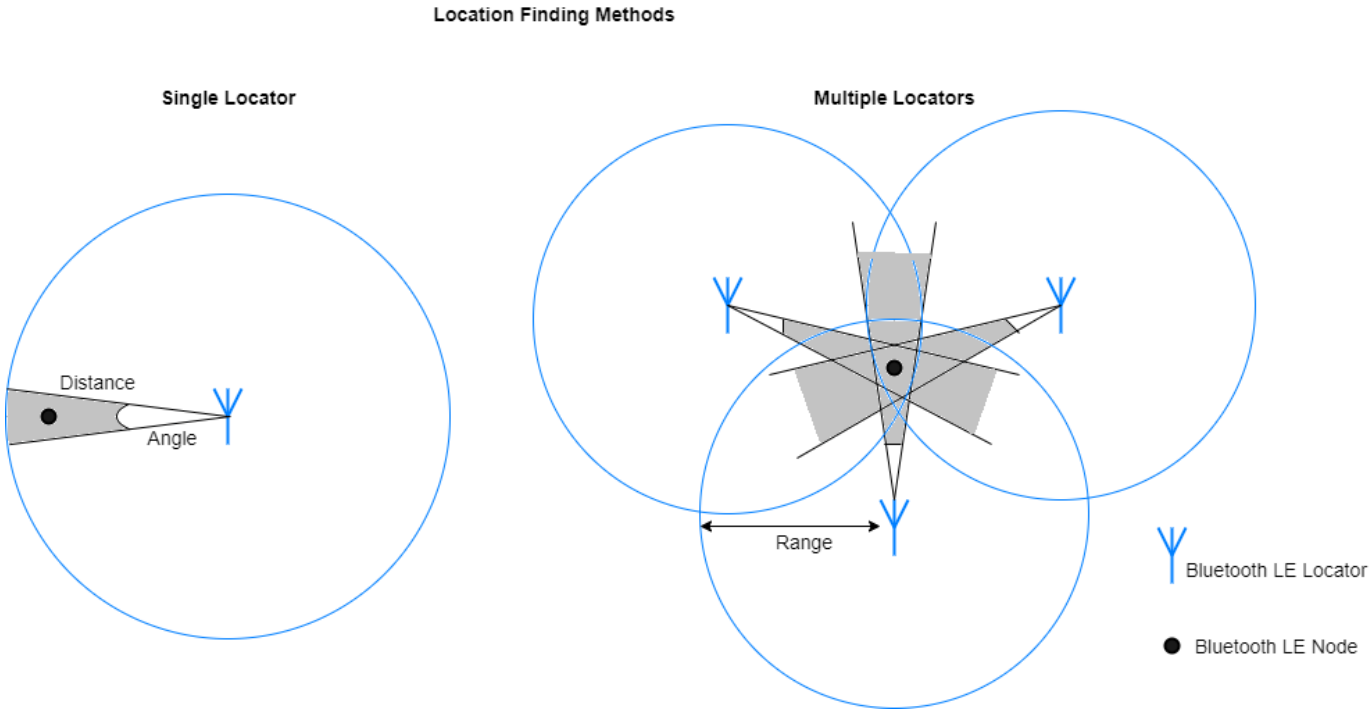
The Bluetooth Core Specification v5.1 [2 on page 3-0] introduced new features that support high-accuracy direction finding. Bluetooth direction finding presents two distinct methods to estimate the 2-D or 3-D location of the Bluetooth LE node: angle of arrival (AoA) and angle of departure (AoD). In AoA and AoD techniques, the antenna array is present at the receiver and transmitter, respectively. Some commonly used antenna arrays are uniform linear array (ULA) and uniform rectangular array (URA). For more information on Bluetooth direction finding, see “Bluetooth Location and Direction Finding” on page 8-18.

Location Finding Methods

The example uses these terminologies:

- Node - The device whose location is to be determined.
- Locator - The receiving device (in AoA calculation) or transmitting device (in AoD calculation).

This figure shows two different ways to calculate the location of the Bluetooth LE node. Each circle indicates the range of a Bluetooth LE locator.



- **Distance plus angle:** If one locator is present in the range of the Bluetooth LE node, use this method to estimate the node position. The location can be estimated by calculating the AoA or AoD of a direction-finding signal and estimating the distance between the locator and node by performing a path loss calculation on the received signal strength indicator (RSSI).
- **Angulation:** If two or more locators are present in the range of the Bluetooth LE node, use this method to estimate the node position. The location can be estimated by triangulating the AoA or AoD between each locator and node.

Simulation Parameters

Specify the linear motion model. Generate dynamic node positions by using one of these motion models:

- 2-D constant velocity
- 3-D constant velocity
- 2-D constant acceleration
- 3-D constant acceleration

```
motionModel = '2-D Constant Velocity';      % Linear motion model
```

Specify the direction-finding method, the direction-finding packet type, and the physical layer (PHY) transmission mode.

```
dfMethod = 'AoA';                           % direction-finding method
dfPacketType = 'ConnectionCTE';             % direction-finding packet type
% PHY transmission mode, must be LE1M or LE2M (for ConnectionCTE) and LE1M
```

```
% (for ConnectionlessCTE)
phyMode = "LE1M";
```

Specify the antenna array parameters.

```
% Antenna array size, must be a scalar (represents ULA) or a vector
% (represents URA) for 2-D or 3-D positioning, respectively
arraySize = 16;

% Normalized spacing between the antenna elements with respect to
% wavelength
elementSpacing = 0.5;

% Antenna switching pattern, must be a 1xM row vector and M must be in the
% range [2, 74/slotDuration+1]
switchingPattern = 1:prod(arraySize);
```

Specify the waveform generation and reception parameters.

```
slotDuration = 2;       % Slot duration in microseconds
sps = 4;                % Samples per symbol
channelIndex = 17;      % Channel index
crcInit = '555551';     % Cyclic redundancy check (CRC) initialization
accessAddress = '01234567';  % Access address
payloadLength = 1;      % Payload length in bytes

% Length of constant tone extension (CTE) in microseconds, must be in the
% range [16, 160] with 8 microseconds step size
cteLength = 160;

% Sample offset, must be a positive integer in the range [sps/8,7*sps/8]
% for LE1M and [sps/4,7*sps/4] for LE2M
sampleOffset = 2;
```

Specify the bit energy to noise density ratio (EbNo), environment and transmitter power.

```
EbNo = 25;              % Eb/No in dB
environment = "Outdoor";% Environment
txPower = 0;            % Transmit power in dBm
```

Validate the simulation parameters.

```
% Configure the bluetoothRangeConfig object
rangeConfig = bluetoothRangeConfig;
rangeConfig.SignalPowerType = "ReceivedSignalPower";
```

```
rangeConfig.Environment = environment;
rangeConfig.TransmitterPower = txPower;
rangeConfig.TransmitterCableLoss = 0;                                   % Transmitter cable loss
rangeConfig.ReceiverCableLoss = 0                                       % Receiver cable loss

rangeConfig =
  bluetoothRangeConfig with properties:

                Environment: 'Outdoor'
           SignalPowerType: 'ReceivedSignalPower'
       ReceivedSignalPower: -79
          TransmitterPower: 0
     TransmitterAntennaGain: 0
        ReceiverAntennaGain: 0
      TransmitterCableLoss: 0
         ReceiverCableLoss: 0
   TransmitterAntennaHeight: 1
      ReceiverAntennaHeight: 1

   Read-only properties:
             FSPLDistance: 87.1650
            PathLossModel: 'TwoRayGroundReflection'
```

```
numDimensions = 2+(strcmp(motionModel,"3-D Constant Velocity") || strcmp(motionModel,"3-D Constar
if numDimensions == 2 && size(arraySize,2) ~= 1
    error("The arraySize must be a scalar for 2-D position estimation");
elseif numDimensions == 3 && size(arraySize,2) ~= 2
    error("The arraySize must be a 1-by-2 vector for 3-D position estimation");
end
if strcmp(dfPacketType,"ConnectionCTE") && payloadLength ~= 1
    error("The payloadLength must be 1 byte for direction-finding packet type of ConnectionCTE")
elseif strcmp(dfPacketType,"ConnectionlessCTE") && payloadLength < 3
    error("The payloadLength must be greater than or equal to 3 bytes for direction-finding packe
end
```

Create and configure a Bluetooth LE angle estimation configuration object.

```
if numDimensions == 3 && isscalar(elementSpacing)
    elementSpacing = [elementSpacing elementSpacing];
end
cfg = bleAngleEstimateConfig("ArraySize",arraySize,"SlotDuration",slotDuration,"SwitchingPattern"
    switchingPattern,"ElementSpacing",elementSpacing);
validateConfig(cfg); % Validate the configuration object
pos = getElementPosition(cfg); % Element positions of an array

% Derive type of CTE based on slot duration and direction-finding method
if strcmp(dfMethod,"AoA")
    cteType = [0;0];
else
    cteType = [0;1];
    if slotDuration == 1
        cteType = [1;0];
    end
end
```

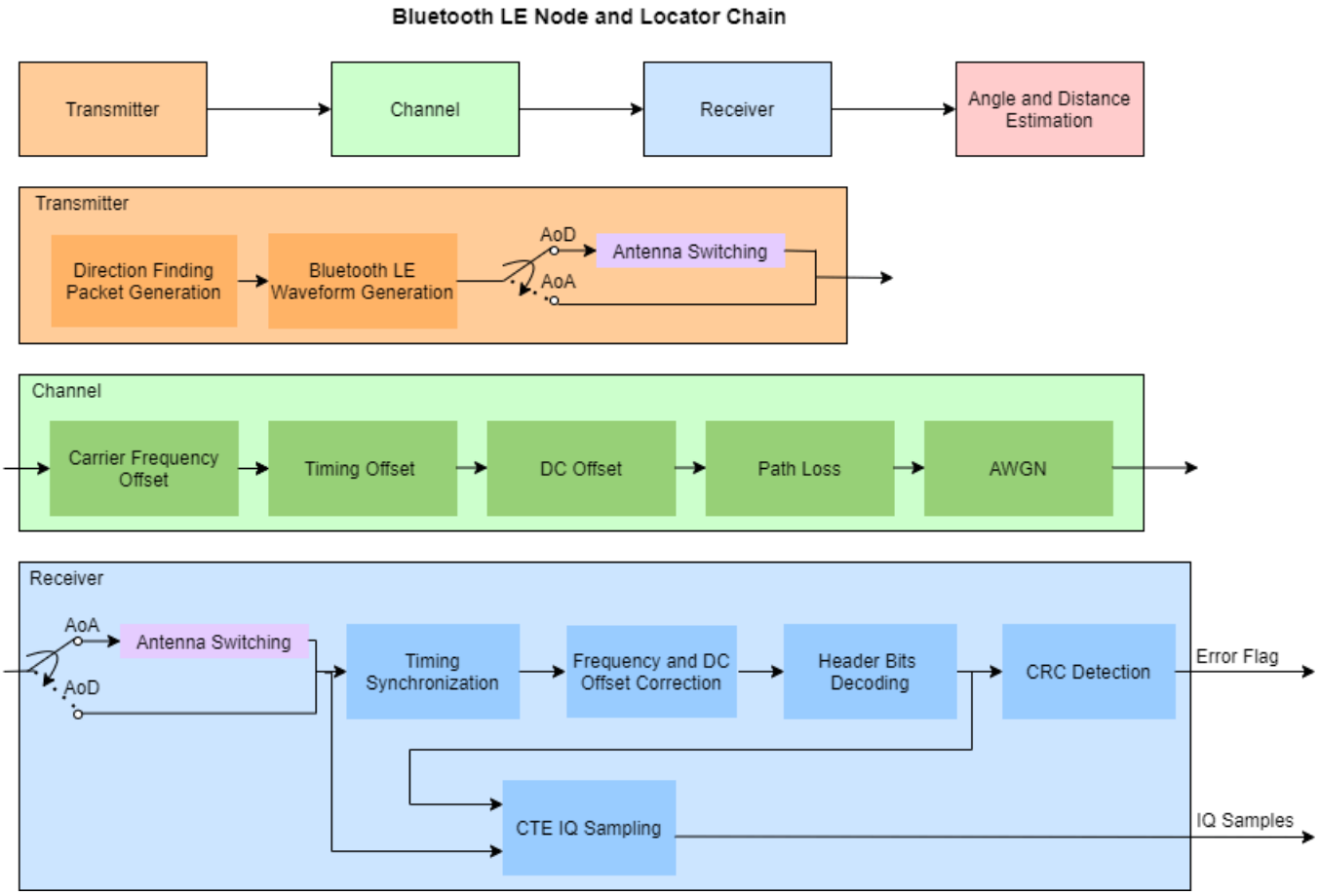
Create and configure System objects™ for receiver processing.

```
% Create and configure preamble detector System object
accessAddBitsLen = 32;
accessAddBits = int2bit(hex2dec(accessAddress),accessAddBitsLen,false);
refSamples = helperBLEReferenceWaveform(phyMode,accessAddBits,sps);
prbDet = comm.PreambleDetector(refSamples,"Detections","First");

% Create and configure coarse frequency compensator System object
phyFactor = 1+strcmp(phyMode,"LE2M");
sampleRate = 1e6*phyFactor*sps;
coarsesync = comm.CoarseFrequencyCompensator("Modulation","OQPSK",...
    "SampleRate",sampleRate,...
    "SamplesPerSymbol",2*sps,...
    "FrequencyResolution",100);

% Create and configure CRC detector System object
crcLen = 24; % CRC length
crcDet = comm.CRCDetector("x^24 + x^10 + x^9 + x^6 + x^4 + x^3 + x + 1", "DirectMethod",true,...
    "InitialConditions", int2bit(hex2dec(crcInit),crcLen).');
```

Packet Transmission and Reception

Perform these steps to track a Bluetooth LE node.

1. Consider 15 locators and a node in a network. Generate 30 node positions based on the motion of the node.
2. At each node position, consider active locators in 80 meters range.
3. Model the direction-finding packet exchange between the node and each active locator. This figure shows the processing chain that the example uses to estimate the angle(s) and distances between the node and active locator.



4. Repeat steps 2 and 3 for each node position.

5. Estimate the node position by using the known active locator positions, distances, and angles.

```
rng('default'); % Initialize the random number generator stream
snr = EbNo - 10*log10(sps); % Signal to noise ratio (SNR) in dB
headerLen = 16+8*strcmp(dfPacketType,"ConnectionCTE"); % Header length
preambleLen = 8*phyFactor; % Preamble length

% Derive initial state for whitening based on channel index
dewhitenStateLen = 6;
chanIdxBin = int2bit(channelIndex,dewhitenStateLen).';
initState = [1 chanIdxBin];

% Consider one node and 15 locators in a network. Generate random locator
% positions and generate node positions based on the motion model.
numLocators = 15; % Number of locators in a network
numNodePositions = 30; % Number of node positions in a network
[posNode,posLocators] = helperBLEPositions(motionModel,numNodePositions,numLocators);

% Simulate motion of the node by looping over the generated node positions.
% For each node position, find active locators.
posNodeEst = zeros(numDimensions,numNodePositions);
for inumNode = 1:numNodePositions
```

```
% Consider active locators that are within 80 meters of range to the node
[posActiveLocators,angleActive,distanceActive] = helperBLEActiveLocators(posNode(:,inumNode)
posLocatorBuffer{:,inumNode} = posActiveLocators; %#ok<SAGROW>
numActiveLocators = size(posActiveLocators,2); % Number of active locators

% Estimate the pathloss between the transmitter and the receiver based on the distance and t
plLin = helperBluetoothEstimatePathLoss(rangeConfig.Environment,distanceActive);
angleEst = zeros(numActiveLocators,numDimensions-1); % Estimated angles
[pathLossdB,linkFailFlag] = deal(zeros(numActiveLocators,1)); % Estimated path loss and flag

% Loop over number of active locators
for i = 1:numActiveLocators
```

Transmitter

```
    % Generate direction-finding packet
    data = helperBLEGenerateDFPDU(dfPacketType,cteLength,cteType,payloadLength,crcInit);

    % Generate Bluetooth LE waveform
    bleWaveform = bleWaveformGenerator(data,"Mode",phyMode,"SamplesPerSymbol",sps,...
        "ChannelIndex",channelIndex,"DFPacketType",dfPacketType,"AccessAddress",accessAddBits

    % If the direction-finding method is AoD, perform antenna switching
    if strcmp(dfMethod,"AoD")
        bleWaveform = helperBLESteerSwitchAntenna(bleWaveform,angleActive(i,:),...
            phyMode,sps,dfPacketType,payloadLength,cfg);
    end
```

Channel

Add RF impairments to the waveform. Attenuate the impaired waveform according to the transmitter power and pathloss.

```
    % Add frequency offset
    freqOffset = randsrc(1,1,-10e3:100:10e3);
    freqWaveform = helperBLEFrequencyOffset(bleWaveform,sampleRate,freqOffset);

    % Add timing offset
    timingoff = randsrc(1,1,1:0.2:20);
    timingOffWaveform = helperBLEDelaySignal(freqWaveform,timingoff);

    % Add DC offset
    dcValue = (5/100)*max(timingOffWaveform);
    dcWaveform = timingOffWaveform + dcValue;

    % Apply transmitter power and path loss
    dBmConverter = 30;
    attenuatedWaveform  = 10^((rangeConfig.TransmitterPower-dBmConverter)/20)*dcWaveform/plLi
```

Receiver

Perform these steps to recover IQ samples for angle estimation.

1. Perform antenna switching (for AoA)
2. Perform packet detection
3. Recover the header bits
4. Perform CRC detection

5 Perform CTE IQ sampling

```
% Perform antenna steering if the direction-finding method is AoA
if strcmp(dfMethod,"AoA")
    steerVec = helperBLESteeringVector(angleActive(i,:),pos);
    steeredWaveform = attenuatedWaveform .* steerVec.';
else
    steeredWaveform = attenuatedWaveform;
end

% Initialize the variables used for receiver processing
packetDetect = 0; % Packet detect flag
headerFlag = 1; % Header flag to perform header decoding
pduCRCFlag = 0; % PDU and CRC flag to perform CRC detection
samplingFlag = 0; % CTE IQ sampling flag
crcError = 1; % Checksum flag
samplesPerFrame = 8*sps; % Samples considered for packet detection
samplesPerModule = samplesPerFrame; % Samples considered for header, PDU and CRC, CTE IQ
numTimes = ceil(length(timingOffWaveform)/samplesPerFrame)+1; % Divide the waveform as pe
countpacketDetect = 0; % Counter for packet detection
moduleStartInd = 0; % Start index for each module
% Append zeros to the waveform
rxWaveformBuffer = [steeredWaveform;zeros(samplesPerFrame*numTimes-length(steeredWaveform
for j = 1:numTimes
    % Retrieve the waveform corresponding to each module
    if (j-1)*samplesPerFrame + moduleStartInd + samplesPerModule <= length(rxWaveformBuf
        rxChunkWaveform = rxWaveformBuffer((j-1)*samplesPerFrame+moduleStartInd+(1:sample
    else
        rxChunkWaveform = rxWaveformBuffer((j-1)*samplesPerFrame+moduleStartInd+1:end,:)
    end
```

A sampling flag is used to handle antenna switching for AoA.

- If the sampling flag is disabled, use the waveform from the first antenna. Add AWGN to the waveform.
- If the sampling flag is enabled, perform antenna switching (for AoA). Add AWGN to the waveform. Perform IQ sampling on the CTE samples.

```
    if ~samplingFlag
        rxNoisyWaveform = awgn(rxChunkWaveform(:,1),snr,"measured"); % Add AWGN
    else
        if strcmp(dfMethod,"AoA") % Perform antenna switching
            rxSwitchWaveform = helperBLESwitchAntenna(rxChunkWaveform,phyMode,sps,slotDu
        else
            rxSwitchWaveform = rxChunkWaveform;
        end
        rxNoisyWaveform = awgn(rxSwitchWaveform,snr,"measured"); % Add AWGN

        % Perform IQ sampling on the CTE
        cteSamples = rxNoisyWaveform(1:cteTime*8*sps*phyFactor);
        iqSamples = bleCTEIQSample(cteSamples,"Mode",phyMode,...
            "SamplesPerSymbol",sps,"SlotDuration",slotDuration,"SampleOffset",sampleOffse
        samplingFlag = 0;
        break;
    end
```

Perform these steps for packet detection:

1. Load the waveform into a buffer to perform preamble detection.
2. Estimate and compensate DC and frequency offsets.
3. Perform Gaussian minimum shift keying (GMSK) demodulation.
4. Compare the decoded access address with the known access address. If these addresses match, the packet is detected.

```
if packetDetect == 0
    countpacketDetect = countpacketDetect+1;
    rcvSigBuffer((countpacketDetect-1)*samplesPerFrame+1:(countpacketDetect)*samplesP
    if countpacketDetect >= (preambleLen+accessAddBitsLen+headerLen)*sps/samplesPerFr

        % Perform timing synchronization
        [~, dtMt] = prbDet(rcvSigBuffer.');
        release(prbDet)
        prbDet.Threshold = max(dtMt);
        prbIdx = prbDet(rcvSigBuffer.');
        release(prbDet)
        if prbIdx >=length(refSamples)
            rcvTrim = rcvSigBuffer(1+prbIdx-length(refSamples):end).';
        else
            rcvTrim = rcvSigBuffer.';
        end

        % Estimate and compensate DC offset
        estimatedDCOffset = mean(rcvTrim(1:length(refSamples)))-mean(refSamples)*sqrt
        rcvDCFree = rcvTrim-estimatedDCOffset;

        % Estimate and compensate frequency offset
        [~,freqoff] = coarsesync(rcvDCFree);
        release(coarsesync)
        [rcvFreqOffsetFree,iniFreqState] = helperBLEFrequencyOffset(rcvDCFree,sampleR

        % Perform GMSK demodulation
        x = rem(length(rcvFreqOffsetFree),sps);
        if x
            remsad = sps - x;
        else
            remsad = x;
        end
        remNumSamples = x;
        remSamples = rcvFreqOffsetFree(end-remNumSamples+1:end);
        [demodSoftBits,demodInitPhase] = helperBLEGMSKDemod(rcvFreqOffsetFree(1:end-r
        demodBits = demodSoftBits>0;

        % Check for access address match
        if length(demodBits) >= preambleLen+accessAddBitsLen
            accessAddress = int8(demodBits(preambleLen+(1:accessAddBitsLen))>0);
            decodeData = int8(demodBits(accessAddBitsLen+preambleLen+1:end)>0);
            if isequal(accessAddBits,accessAddress)
                packetDetect = 1;
                samplesPerModule = headerLen*sps-length(decodeData)*sps+remsad;
                rcvSigBuffer = [];
            end
        end
    end
end
```

Recover the header bits if the packet is detected.

```
            if packetDetect
                if headerFlag && (length(rxNoisyWaveform) ~= samplesPerFrame || samplesPerModule

                    % Remove DC offset
                    rxDCFreeWaveform = rxNoisyWaveform-estimatedDCOffset;

                    % Frequency offset correction
                    [rxNoisyWaveform,iniFreqState] = helperBLEFrequencyOffset(rxDCFreeWaveform,sa

                    % Perform GMSK demodulation on the header bits
                    if (length(rxNoisyWaveform) ~= samplesPerFrame)
                        rcvSigHeaderBuffer = [remSamples;rxNoisyWaveform];
                        [headerSoftBits,demodInitPhase] = helperBLEGMSKDemod(rcvSigHeaderBuffer,s
                        decodeDataHeader = [decodeData;headerSoftBits>0];
                    elseif samplesPerModule <= 0
                        decodeDataHeader = decodeData;
                    end

                    % Perform data dewhitening
                    [dewhitenedBits,initState1] = helperBLEWhiten(decodeDataHeader,initState);

                    % Extract PDU length
                    pduLenField = double(dewhitenedBits(9:16)); % Second byte of PDU header
                    pduLenInBits = bi2de(pduLenField')*8;

                    % Retrieve CTE time and CTE type information from bits
                    if strcmp(dfPacketType,"ConnectionCTE")
                        [cteTime,cteTypeDec] = helperBLECTEInfoExtract(dewhitenedBits,dfPacketTyp
                        % Slot duration based on CTE type
                        if cteTypeDec == 1 || cteTypeDec == 2
                            slotDuration = cteTypeDec;
                        else
                            slotDuration = cfg.SlotDuration;
                        end
                    end

                    % Update the samples per frame, start index to
                    % perform CRC detection
                    if samplesPerModule
                        moduleStartInd = samplesPerModule-samplesPerFrame;
                    else
                        moduleStartInd = 0;
                    end
                    samplesPerModule = (pduLenInBits+crcLen-length(dewhitenedBits)+headerLen)*sps
                    pduCRCFlag = 1; % Enable the flag for CRC detection
                    headerFlag = 0; % Disable the header recovery flag
                    rcvSigHeaderVar = rxNoisyWaveform; % Load the waveform for pathloss estimatio
                    rxNoisyWaveform = [];
                end
```

Perform CRC detection based on the PDU length recovered from header bits.

```
                if pduCRCFlag && ~isempty(rxNoisyWaveform)

                    % Remove DC offset
                    rxDCFreeWaveform = rxNoisyWaveform-estimatedDCOffset;
```

```
                % Frequency offset correction
                [rxNoisyWaveform,iniFreqState] = helperBLEFrequencyOffset(rxDCFreeWaveform,sa

                % Perform GMSK demodulation on the PDU and CRC bits
                x = rem(length(rxNoisyWaveform),sps);
                if x
                    remNumSamples = sps - x;
                else
                    remNumSamples = x;
                end
                demodPDUCRC = helperBLEGMSKDemod([rxNoisyWaveform;zeros(remNumSamples,1)],sps

                % Perform data dewhitening
                dewhitenedPDUCRC = helperBLEWhiten(demodPDUCRC,initState1);

                % Retrieve CTE time and CTE type information from bits
                headerPDUCRC = [double(dewhitenedBits);dewhitenedPDUCRC];
                if strcmp(dfPacketType,"ConnectionlessCTE")
                    [cteTime,cteTypeDec] = helperBLECTEInfoExtract(headerPDUCRC,dfPacketType)
                    % Slot duration based on CTE type
                    if cteTypeDec == 1 || cteTypeDec == 2
                        slotDuration = cteTypeDec;
                    else
                        slotDuration = cfg.SlotDuration;
                    end
                end

                % Perform CRC detection
                [dfPDU,crcError] = crcDet(headerPDUCRC);
                if crcError
                    break;
                end

                % Update the samples per frame, start index to
                % perform IQ sampling
                moduleStartInd = samplesPerModule-samplesPerFrame+moduleStartInd;
                samplesPerModule = cteTime*8*sps*phyFactor;
                samplingFlag = 1; % Enable the flag to perform IQ sampling
                pduCRCFlag = 0; % Disable the PDU and CRC flag
                rcvSigPDUVar = rxNoisyWaveform; % Load the waveform for pathloss estimation
            end
        end
    end
```

Estimate Angle and Pathloss

Use the IQ samples from the receiver to estimate the angles and path loss.

Perform angle estimation if both of these conditions meet:

- CRC is detected.
- A minimum number of non-zero IQ samples are present in `iqSamples`.

```
% IQ samples must contain at least eight samples from reference period and
% one sample from each antenna
refSampleLength = 8; % Reference samples length
minIQSamples = refSampleLength+getNumElements(cfg)-1;
```

```
        if ~crcError && length(nonzeros(iqSamples)) >= minIQSamples  % If packet detection is su
            angleEst(i,:) = bleAngleEstimate(iqSamples,cfg);
            rangeConfig.ReceivedSignalPower = 10*log10(var([rcvFreqOffsetFree;rcvSigHeaderVar;rc
            pathLossdB(i) = pathLoss(rangeConfig);
        else
            linkFailFlag(i) = 1; % If packet detection fails, enable the link fail flag
        end
    end
```

Estimate Node Position

Estimate the position of the node in the network by using one of these two methods.

- Angulation: If two or more number of active locators are successfully decoded.
- Distance-angle: If one active locator is successfully decoded.

```
    if any(linkFailFlag == 1)
        idx = find(linkFailFlag == 1);
        posActiveLocators(:,idx) = [];
        angleEst(idx,:) = [];
        pathLossdB(idx) = [];
    end
    if (numActiveLocators-nnz(linkFailFlag)) >= 2 && ~all(angleEst(:,1) == angleEst(1,1))

        % Estimate the node position by using the angulation method
        posNodeEst(:,inumNode) = blePositionEstimate(posActiveLocators,"angulation",angleEst.');
    elseif (numActiveLocators-nnz(linkFailFlag)) == 1 || (~all(linkFailFlag == 1) && all(angleEs

        % Estimate the distance between the transmitter and the receiver based on the environmen
        range = bluetoothRange(rangeConfig);
        distanceValues = min(range):max(range);
        idx = randi(length(distanceValues),1);
        distanceEst = distanceValues(idx)

        % Estimate the node position by using the distance-angle method
        posNodeEst(:,inumNode) = blePositionEstimate(posActiveLocators,"distance-angle",distanceE
    else
        posNodeEst(:,inumNode) = NaN(numDimensions,1);
    end
end
```

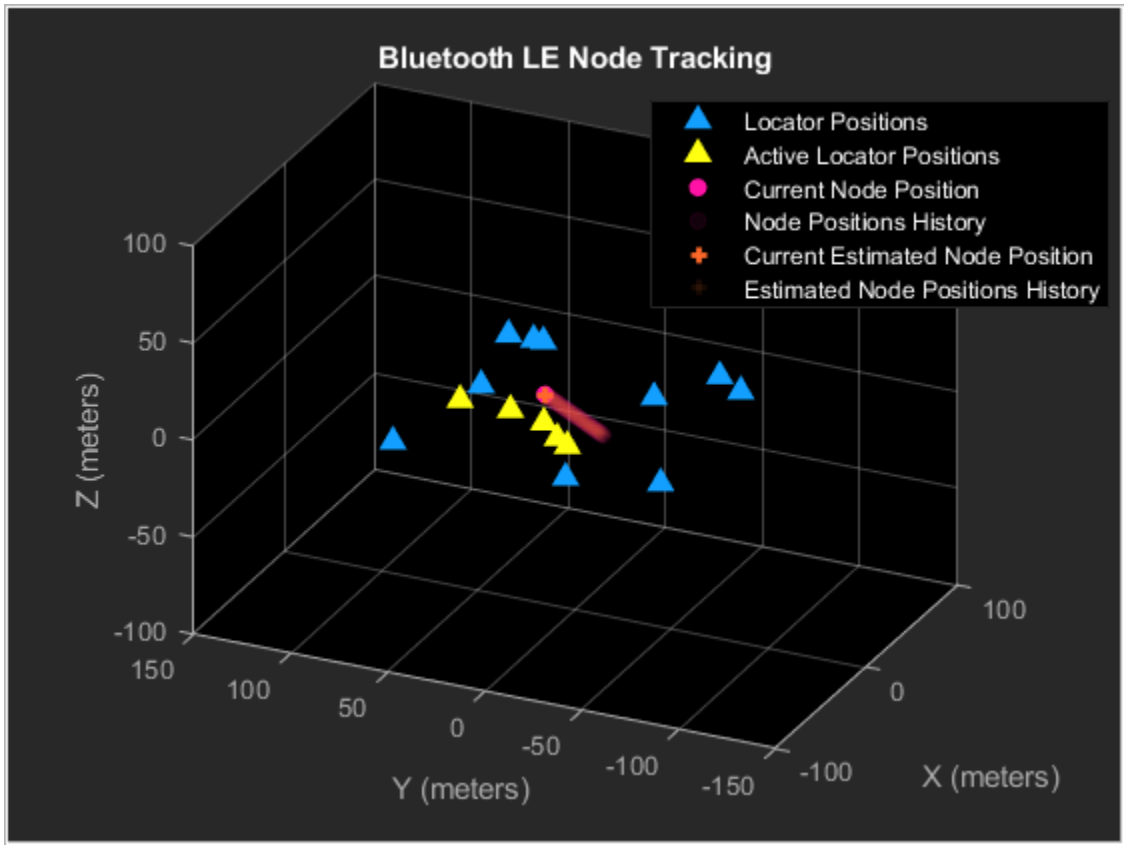
Simulation Results

Visualize the Bluetooth LE node tracking.

```
posErr = sqrt(sum((posNodeEst-posNode).^2)); % Compute the position error
disp(["Positioning error in meters = ", num2str(posErr)])
```

```
    "Positioning error in meters = "    "0.13076     0.27889    0.07999..."
```

```
if ~all(isnan(posNodeEst(1,:)))
    helperBLEVisualizeNodeTracking(posLocators,posNode,posLocatorBuffer,posNodeEst)
end
```



This example enables you to estimate the 2-D or 3-D position of a Bluetooth LE node using the direction-finding functionality and location estimation techniques. The example also measures positioning accuracy at each node position.

Further Exploration

Improve the position accuracy of the Bluetooth LE node by using the Kalman filter from Sensor Fusion and Tracking Toolbox™. The Kalman filter is a recursive algorithm that estimates the track of an object by updating the current state using the previous state. When the evolution of the state follows a linear motion model and the measurements are linear functions of the state, the Kalman filter is linear. For more information about the linear Kalman filter, see “Linear Kalman Filters” (Sensor Fusion and Tracking Toolbox).

The two important characteristics of Kalman filter are:

- Predicting the node's future location
- Reducing the noise introduced by inaccurate detections

If node estimation measurement is available, the Kalman filter reduces the noise by predicting and correcting together. If node estimation measurement is not available (all the node-locators links fail), the Kalman filter estimates the location based on the previous measurements. As there is no prior knowledge on initial estimates of the velocity or acceleration of node, the position estimation can be transient during the initial phase. Use this sample code snippet after estimating node positions.

```
% if all(isnan(posNodeEst(1,:)))
%     disp("All direction-finding packet transmissions failed")
```

```
% else
%     posNodeTrackEst = helperBLEKalmanFilter(motionModel,posNodeEst);
%     posTrackErr = sqrt(sum((posNodeTrackEst-posNode).^2)); % Compute the position error with Ka
%     disp(["Positioning error with Kalman filter in meters = ", num2str(posTrackErr)])
% end
```

Appendix

The example uses these helpers:

- helperBLEReferenceWaveform: Generate Bluetooth LE reference samples
- helperBLEPositions: Generate locators and node positions
- helperBLEActiveLocators: Generate active locators parameters
- helperBluetoothEstimatePathLoss: Estimate the path loss between the node and locator
- helperBLEGenerateDFPDU: Generate direction-finding packet PDU
- helperBLESteerSwitchAntenna: Perform antenna steering and switching
- helperBLEFrequencyOffset: Apply frequency offset to the input signal
- helperBLEDelaySignal: Introduce time delay in the signal
- helperBLESteeringVector: Generate steering vector
- helperBLESwitchAntenna: Perform antenna switching
- helperBLEGMSKDemod: Perform GMSK demodulation
- helperBLEWhiten: Perform whitening/dewhitening
- helperBLECTEInfoExtract: Extract the CTE information
- helperBLEKalmanFilter: Track the Bluetooth LE node using linear Kalman filter
- helperBLEVisualizeNodeTracking: Generate Bluetooth LE node tracking visualization

Selected Bibliography

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com.
2. Bluetooth Special Interest Group (SIG). "Core System Package [Low Energy Controller Volume]". Bluetooth Core Specification. Version 5.1, Volume https://www.bluetooth.com.
3. Bluetooth Special Interest Group (SIG). "Core System Package [Low Energy Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume https://www.bluetooth.com.

See Also

Functions

`bleWaveformGenerator` | `bleAngleEstimate` | `blePositionEstimate` | `bleCTEIQSample` | `bluetoothRange`

Objects

`bleAngleEstimateConfig` | `bluetoothRangeConfig`

More About

- “Parameterize Bluetooth LE Direction Finding Features” on page 9-10
- “Estimate Bluetooth LE Node Position” on page 9-35

- “Bluetooth Location and Direction Finding” on page 8-18
- “Bluetooth LE Positioning by Using Direction Finding” on page 3-2

Test and Measurement

Bluetooth BR/EDR Waveform Reception Using SDR

This example shows how to capture and decode Bluetooth® BR/EDR waveforms by using Bluetooth® Toolbox. You can either capture the Bluetooth BR/EDR waveforms by using the ADALM-PLUTO radio or load IQ samples corresponding to the Bluetooth BR/EDR waveforms from a baseband file (*.bb). To generate and transmit the Bluetooth BR/EDR waveforms, see “Bluetooth BR/EDR Waveform Generation and Transmission Using SDR” on page 4-10 and configure your test environment with:

- Two SDR platforms connected to the same host computer, and run two MATLAB® sessions on the single host computer.
- Two SDR platforms connected to two host computers, and run one MATLAB session on each host computer.

To configure your host computer to work with the Support Package for ADALM-PLUTO Radio, see “Guided Host-Radio Hardware Setup” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio).

Required Hardware

To capture signals in real time, you need ADALM-PLUTO radio and the corresponding support package add-on:

- Communications Toolbox Support Package for ADALM-PLUTO Radio

For a full list of communications toolboxes supported by SDR platforms, refer to the Supported Hardware section of the Software Defined Radio (SDR) discovery page.

Bluetooth BR/EDR Radio Specifications

Bluetooth is a short-range Wireless Personal Area Network (WPAN) technology, operating in the globally unlicensed industrial, scientific, and medical (ISM) band in the frequency range of 2.4 GHz to 2.485 GHz. In Bluetooth technology, data is divided into packets and each packet is transmitted on one of the 79 designated Bluetooth channels. Each channel has a bandwidth of 1 MHz. As there are different types of wireless networks operating in the same unlicensed frequency band, it is possible for two different networks to interfere with each other. To mitigate the interference, Bluetooth implements the frequency-hopping spread spectrum (FHSS) scheme to switch a carrier between multiple frequency channels by using a pseudorandom sequence known to both transmitter and receiver.

The Bluetooth standard specifies these physical layer (PHY) modes:

Basic rate (BR) - Mandatory mode, uses gaussian frequency shift keying (GFSK) modulation with a data rate of 1 Mbps

Enhanced data rate (EDR) - Optional mode, uses phase shift keying (PSK) modulation with these two variants:

- EDR2M: Uses pi/4-DQPSK with a data rate of 2 Mbps
- EDR3M: Uses 8-DPSK with a data rate of 3 Mbps

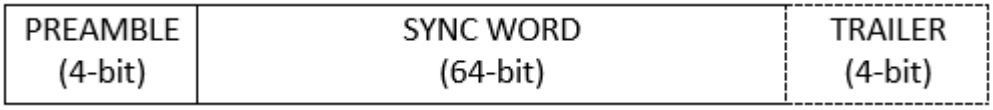
Bluetooth BR/EDR Packet Formats

The air interface packet formats for PHY modes include these fields:

Access Code: Each packet starts with an access code. If a packet header follows, the access code is 72 bits long. Otherwise, the length of the access code is 68 bits and referred to as a shortened access code. The access code consists of these fields:

- Preamble: The preamble is a fixed zero-one pattern of four symbols.
- Sync Word: The sync word is a 64-bit code word derived from the 24-bit lower address part (LAP) of the Bluetooth device address.
- Trailer: The trailer is a fixed zero-one pattern of four symbols.

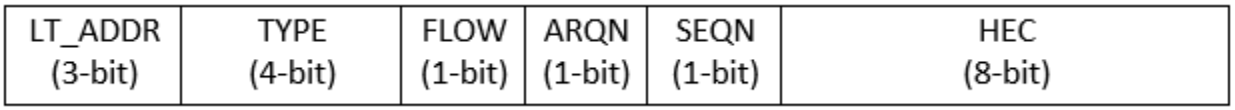
Access Code Format



Packet Header: The header includes link control information and consists of these fields:

- LT_ADDR: 3-bit logical transport address
- TYPE: 4-bit type code, which specifies the packet type used for transmission. The value of this field can be ID, NULL, POLL, FHS, HV1, HV2, HV3, DV, EV3, EV4, EV5, 2-EV3, 2-EV5, 3-EV3, 3-EV5, DM1, DH1, DM3, DH3, DM5, DH5, AUX1, 2-DH1, 2-DH3, 2-DH5, 3-DH1, 3-DH3 and 3-DH5. This field determines the number of slots the current packet occupies.
- FLOW: 1-bit flow control over the asynchronous connection-oriented logical (ACL) transport
- ARQN: 1-bit acknowledgement indication
- SEQN: 1-bit sequence number
- HEC: 8-bit header error check

Header Format



Payload: Payload includes an optional payload header, a payload body, and an optional CRC.

Payload Format

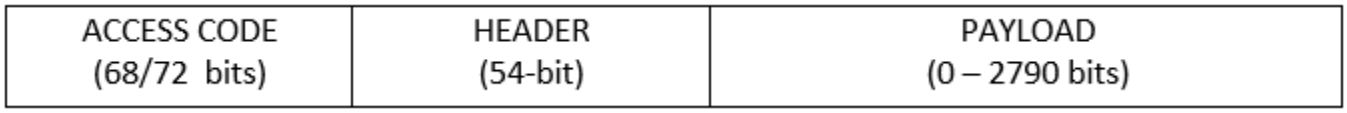
Guard: For EDR packets, guard time allows the Bluetooth BR/EDR radio to prepare for the change in modulation from GFSK to DPSK. The guard time must be between 4.75 to 5.25 microseconds.

Sync: For EDR packets, the synchronization sequence contains one reference symbol and ten DPSK symbols.

Trailer: For EDR packets, the trailer bits must be all zero pattern of two symbols, {00,00} for pi/4-DQPSK and {000,000} for 8DPSK.

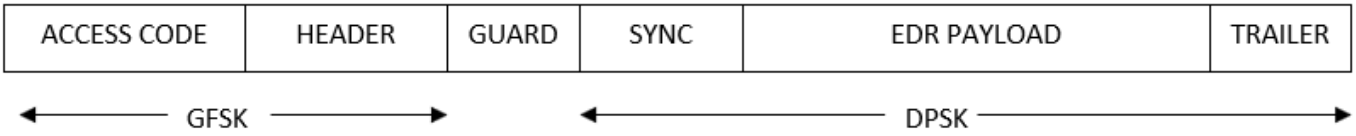
Packet format for BR mode is shown in this figure.

Basic Rate Packet Format



Packet format for EDR mode is shown in this figure.

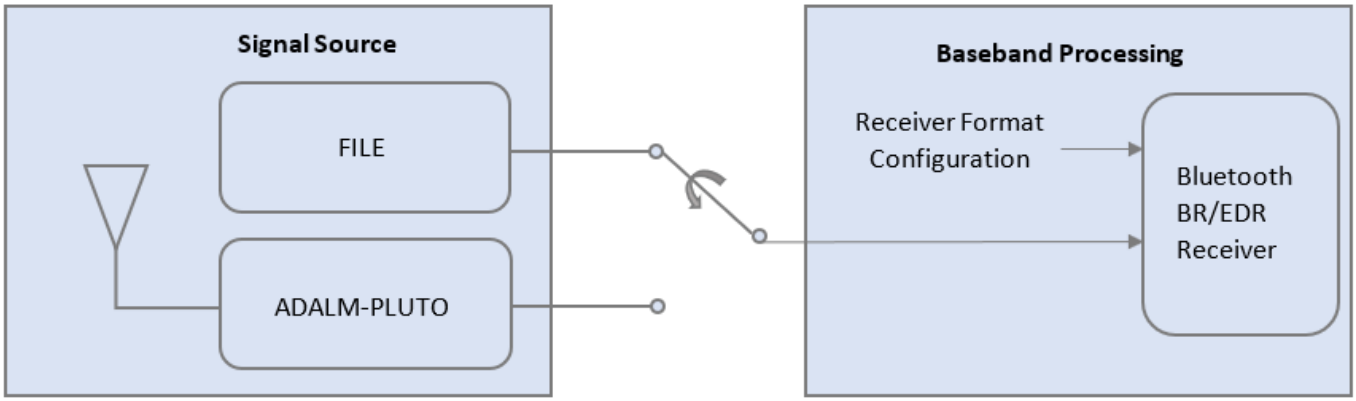
Enhanced Data Rate Packet Format



Decode Bluetooth BR/EDR Waveforms

This example shows how to decode Bluetooth BR/EDR waveforms either captured by using ADALM-PLUTO or by reading IQ samples from a baseband file.

Bluetooth BR/EDR Receiver



The general structure of the Bluetooth receiver example is:

1. Initialize the receiver parameters.
2. Specify the signal source.
3. Capture the Bluetooth BR/EDR waveforms.
4. Process Bluetooth BR/EDR waveforms at receiver.

Initialize Receiver Parameters

To configure Bluetooth BR/EDR parameters, use `bluetoothPhyConfig` object.

```
cfg = bluetoothPhyConfig;
cfg.Mode = BR; % Mode of transmission as one of BR, EDR2M and EDR3M
cfg.WhitenInitialization = [0;0;0;0;0;1;1]; % Whiten initialization
```

Specify Signal Source

Specify the signal source as File or ADALM-PLUTO.

- **File**: Uses the `comm.BasebandFileReader` to read a file that contains a previously captured over-the-air signal.
- **ADALM-PLUTO**: Uses the`sdrrx` (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio) System object to receive a live signal from the SDR hardware.

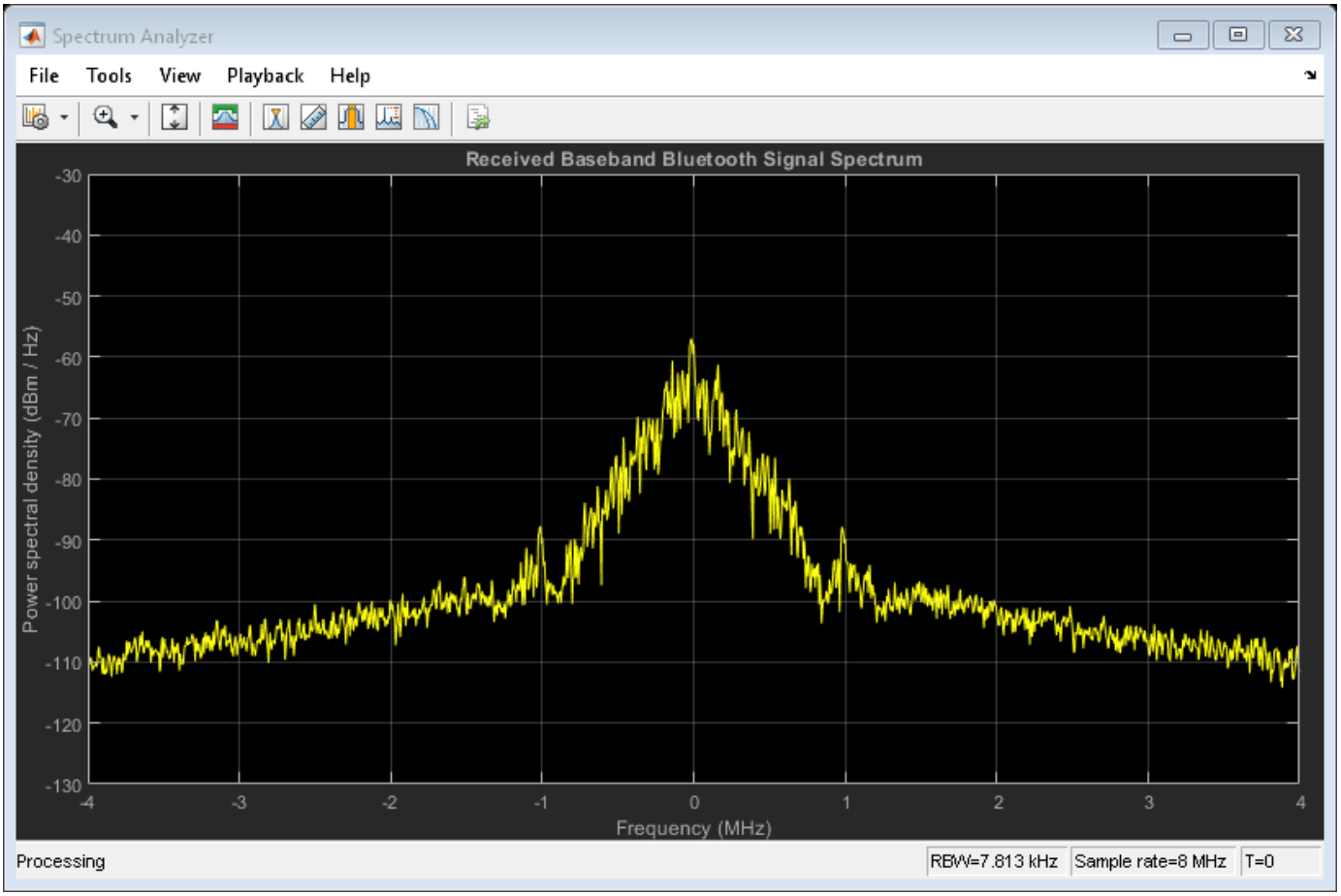
If you assign **ADALM-PLUTO** as the signal source, the example searches your computer for the **ADALM-PLUTO** radio at radio address 'usb:0' and uses it as the signal source.

```
% The default signal source is 'File'
signalSource = File;

bbSymbolRate = 1e6; % 1 MSps
if strcmp(signalSource,'File')
    switch cfg.Mode
        case 'BR'
            bbFileName = 'bluetoothCapturesBR.bb';
        case 'EDR2M'
            bbFileName = 'bluetoothCapturesEDR2M.bb';
        case 'EDR3M'
            bbFileName = 'bluetoothCapturesEDR3M.bb';
    end
    sigSrc = comm.BasebandFileReader(bbFileName);
    sigSrcInfo = info(sigSrc);
    bbSampleRate = sigSrc.SampleRate;
    sigSrc.SamplesPerFrame = sigSrcInfo.NumSamplesInData;
    cfg.SamplesPerSymbol = bbSampleRate/bbSymbolRate;

else

    % Check if the ADALM-PLUTO Hardware Support Package (HSP) is installed
    if isempty(which('plutoradio.internal.getRootDir'))
        error(message('comm_demos:common:NoSupportPackage', ...
                      'Communications Toolbox Support Package for ADALM-PLUTO Radio',...
                      ['<a href="https://www.mathworks.com/hardware-support/' ...
                      'adalm-pluto-radio.html">ADALM-PLUTO Radio Support From Communications Too
    end
    connectedRadios = findPlutoRadio; % Discover ADALM-PLUTO radio(s) connected to your computer
    radioID = connectedRadios(1).RadioID;
    rxCenterFrequency = 2445000000;  % In Hz, choose between 2.402e9 to
    bbSampleRate = bbSymbolRate * cfg.SamplesPerSymbol;
    sigSrc = sdrrx('Pluto',...
        'RadioID',             radioID,...
        'CenterFrequency',     rxCenterFrequency,...
        'BasebandSampleRate',  bbSampleRate,...
        'SamplesPerFrame',     1e7,...
        'GainSource',          'Manual',...
        'Gain',                25,...
        'OutputDataType',      'double');
end
```

Capture Bluetooth BR/EDR Waveforms

Capture the IQ samples corresponding to Bluetooth BR/EDR waveforms either by using ADALM-PLUTO or baseband file as signal source. Visualize the spectrum of the received Bluetooth waveforms by using a spectrum analyzer.

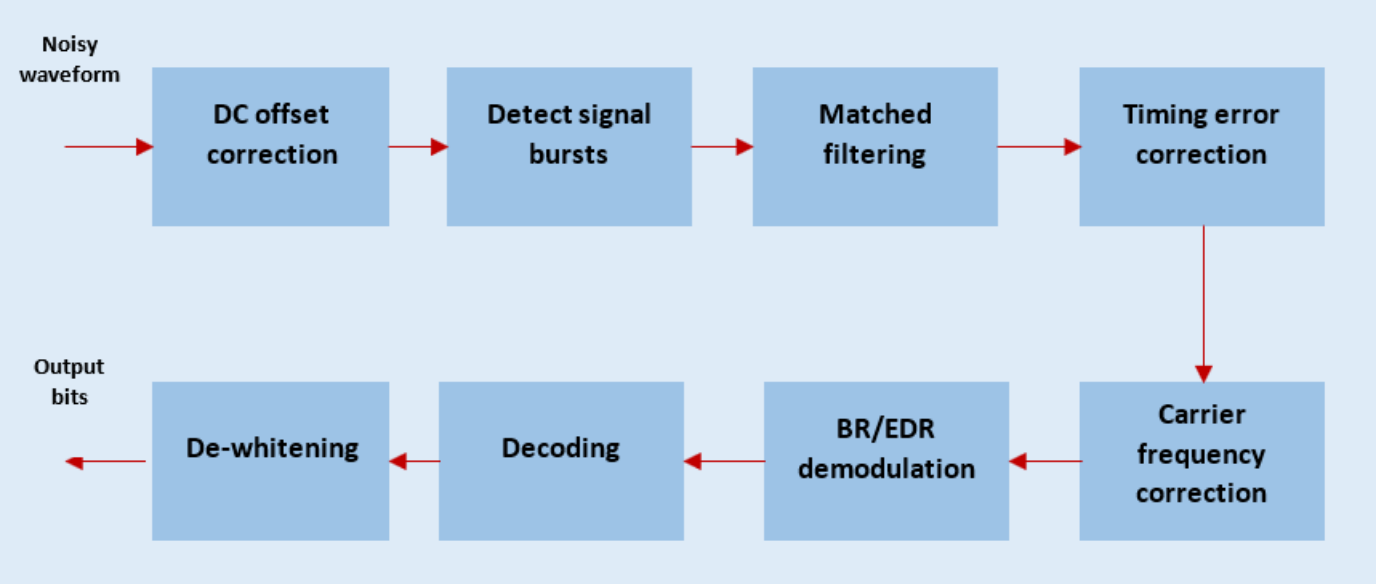
```
% The transmitted waveforms are captured as a burst
dataCaptures = sigSrc();

% Setup spectrum viewer
spectrumScope = dsp.SpectrumAnalyzer( ...
    'SampleRate',       bbSampleRate,...
    'SpectrumType',     'Power density', ...
    'SpectralAverages', 10, ...
    'YLimits',          [-130 -30], ...
    'Title',            'Received Baseband Bluetooth Signal Spectrum', ...
    'YLabel',           'Power spectral density');

% Show power spectral density of the received waveform
spectrumScope(dataCaptures);
```



Process Bluetooth BR/EDR Waveforms at Receiver

To decode the packet header, payload header information, and raw message bits, the receiver processess the baseband samples received from the signal source. This figure shows the receiver processing.

Bluetooth Practical Receiver



The Bluetooth practical receiver performs these functions:

1. Remove DC offset
2. Detect the signal bursts
3. Perform matched filtering
4. Estimate and correct the timing offset
5. Estimate and correct the carrier frequency offset
6. Demodulate BR/EDR waveform
7. Perform forward error correction (FEC) decoding
8. Perform data dewhitening
9. Perform header error check (HEC) and cyclic redundancy check (CRC)
10. Outputs decoded bits and decoded packet statistics based on decoded lower address part (LAP), HEC and CRC

```
% Bluetooth practical receiver
[decBits,decodedInfo,pktStatus] = helperBluetoothPracticalReceiver(dataCaptures,cfg);

% Get the number of detected packets
pktCount = length(pktStatus);
disp(['Number of Bluetooth packets detected: ' num2str(pktCount)])
```

```
Number of Bluetooth packets detected: 2
```

```
% Get the decoded packet statistics
```

```
displayFlag = true; % set true, to display the decoded packet statistics
if(displayFlag && (pktCount~=0))
    decodedInfoPrint = decodedInfo;
    for ii = 1:pktCount
        if(pktStatus(ii))
            decodedInfoPrint(ii).PacketStatus = 'Success';
        else
            decodedInfoPrint(ii).PacketStatus = 'Fail';
        end
    end
    packetInfo = struct2table(decodedInfoPrint,'AsArray',1);
    fprintf('Decoded Bluetooth packet(s) information: \n \n')
    disp(packetInfo);
end
```

```
Decoded Bluetooth packet(s) information:

        LAP          PacketType    LogicalTransportAddress    HeaderControlBits    PayloadLength
    _____________    __________    _______________________    _________________    _____________

    {24x1 double}     {'FHS'}           {3x1 double}            {3x1 double}            18
    {24x1 double}     {'FHS'}           {3x1 double}            {3x1 double}            18
```

```
% Get the packet error rate performance metrics
if(pktCount)
    pktErrCount = sum(~pktStatus);
    pktErrRate = pktErrCount/pktCount;
    disp(['Simulated Mode: ' cfg.Mode ', '...
        'Packet error rate: ',num2str(pktErrRate)])
end
```

```
Simulated Mode: BR, Packet error rate: 0
```

```
% Release the signal source
release(sigSrc);
```

This example enables you to decode Bluetooth BR/EDR waveforms either captured by using ADALM-PLUTO or by reading IQ samples from a baseband file. Visualize the spectrum of the received Bluetooth waveforms by using a spectrum analyzer. The packet error rate is computed based on the decoded packet information.

Further Exploration

You can use this example to receive EDR packets by changing the PHY transmission mode. To generate the Bluetooth waveforms in this example, see “Bluetooth BR/EDR Waveform Generation and Transmission Using SDR” on page 4-10.

Troubleshooting

General tips for troubleshooting SDR hardware and the Communications Toolbox Support Package for ADALM-PLUTO Radio can be found in “Common Problems and Fixes” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio).

Appendix

This example uses this helper function:

- helperBluetoothPracticalReceiver.m: Practical receiver for Bluetooth physical layer

Selected Bibliography

1 Bluetooth Special Interest Group (SIG). "Core System Package [BR/EDR Controller Volume]". *Bluetooth Core Specification*. Version 5.3, Volume 2. www.bluetooth.com

See Also

Objects

`bluetoothPhyConfig`

More About

- “Bluetooth BR/EDR Waveform Generation and Transmission Using SDR” on page 4-10

Bluetooth BR/EDR Waveform Generation and Transmission Using SDR

This example shows how to generate and transmit Bluetooth® BR/EDR waveforms using Bluetooth® Toolbox. You can either transmit the Bluetooth BR/EDR waveforms by using the ADALM-PLUTO radio or write to a baseband file (*.bb).

To receive the transmitted Bluetooth BR/EDR waveform, see “Bluetooth BR/EDR Waveform Reception Using SDR” on page 4-2 and implement any one of these setups:

- Two SDR platforms connected to the same host computer which runs two MATLAB sessions.
- Two SDR platforms connected to two host computers which runs two separate MATLAB sessions.

To configure your host computer to work with the Support Package for ADALM-PLUTO Radio, refer “Guided Host-Radio Hardware Setup” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio).

Required Hardware

To transmit signals in real time, you need ADALM-PLUTO radio and the corresponding support package:

- Communications Toolbox Support Package for ADALM-PLUTO Radio

For a full list of Communications Toolbox supported SDR platforms, refer to Supported Hardware section of the Software Defined Radio (SDR) discovery page.

Bluetooth BR/EDR Radio Specifications

Bluetooth [1 on page 4-0] is a short-range Wireless Personal Area Network (WPAN) technology, operating in the globally unlicensed industrial, scientific, and medical (ISM) band in the frequency range 2.4 GHz to 2.485 GHz. In Bluetooth technology, data is divided into packets. Each packet is transmitted on one of the 79 designated Bluetooth channels. Each channel has a bandwidth of 1 MHz. As there are different types of wireless networks operating in the same unlicensed frequency band, it is possible for two different networks to interfere with each other. To mitigate the interference, Bluetooth implements the frequency-hopping spread spectrum (FHSS) scheme to switch a carrier between multiple frequency channels by using a pseudorandom sequence known to both the transmitter and receiver.

The Bluetooth standard specifies these physical layer (PHY) modes:

Basic rate (BR) - Mandatory mode, uses Gaussian frequency shift keying (GFSK) modulation with a data rate of 1 Mbps.

Enhanced data rate (EDR) - Optional mode, uses phase shift keying (PSK) modulation with these two variants:

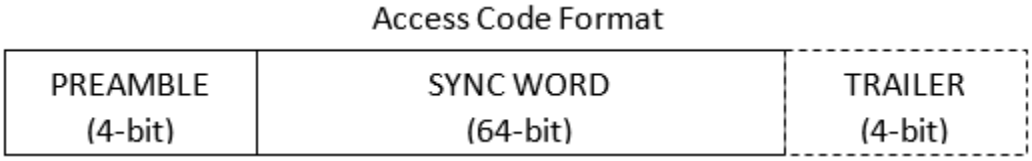
- EDR2M: Uses pi/4-DQPSK with a data rate of 2 Mbps.
- EDR3M: Uses 8-DPSK with a data rate of 3 Mbps.

Packet Formats

The air interface packet formats for PHY modes include these fields:

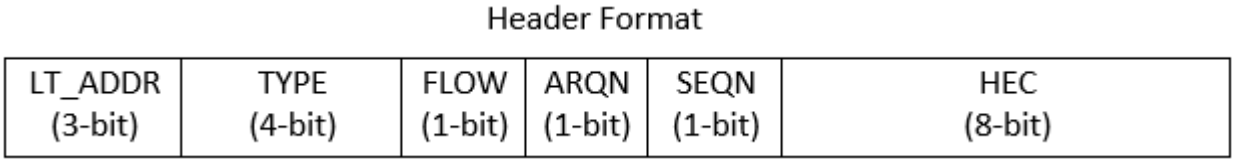
Access Code: Each packet starts with an access code. If a packet header follows, the access code is 72 bits long, otherwise the access code is 68 bits long. The access code consists of these fields:

- Preamble: The preamble is a fixed zero-one pattern of four symbols.
- Sync Word: The sync word is a 64-bit code word derived from 24-bit lower address part (LAP) of the Bluetooth device address.
- Trailer: The trailer is a fixed zero-one pattern of four symbols.

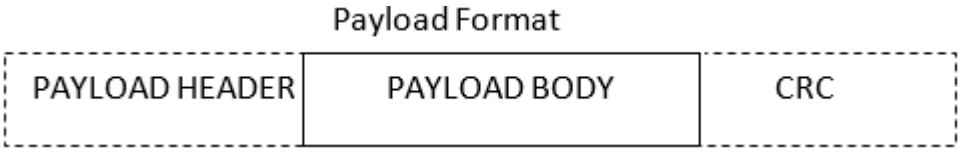


Packet Header: The header includes link control information and consists of these fields:

- LT_ADDR: 3-bit logical transport address.
- TYPE: 4-bit type code, which specifies the packet type used for transmission. It can be one of {ID, NULL, POLL, FHS, HV1, HV2, HV3, DV, EV3, EV4, EV5, 2-EV3, 2-EV5, 3-EV3, 3-EV5, DM1, DH1, DM3, DH3, DM5, DH5, AUX1, 2-DH1, 2-DH3, 2-DH5, 3-DH1, 3-DH3, 3-DH5}.
- FLOW: 1-bit flow control.
- ARQN: 1-bit acknowledgement indication.
- SEQN: 1-bit sequence number.
- HEC: 8-bit header error check.



Payload: Payload includes an optional payload header, a payload body, and an optional CRC.

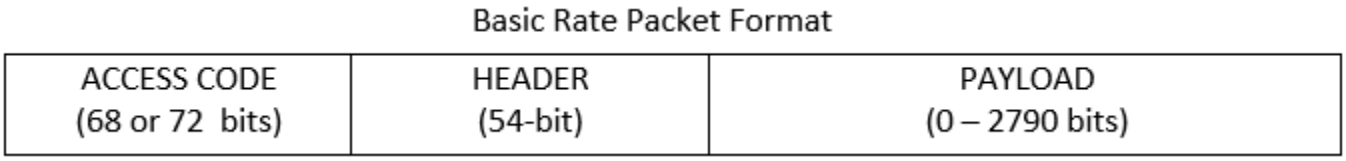


Guard: For EDR packets, guard time allows the Bluetooth radio to prepare for the change in modulation from GFSK to DPSK. The guard time must be between 4.75 to 5.25 microseconds.

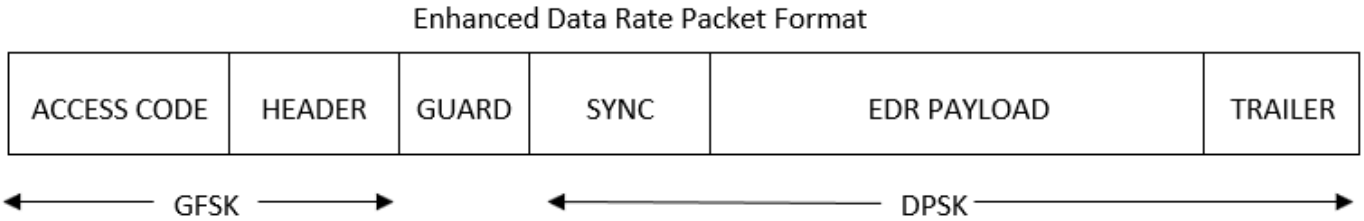
Sync: For EDR packets, the synchronization sequence contains one reference symbol and ten DPSK symbols.

Trailer: For EDR packets, the trailer bits must be all zero pattern of two symbols, {00,00} for pi/4-DQPSK and {000,000} for 8DPSK.

This figure shows the packet format for BR mode

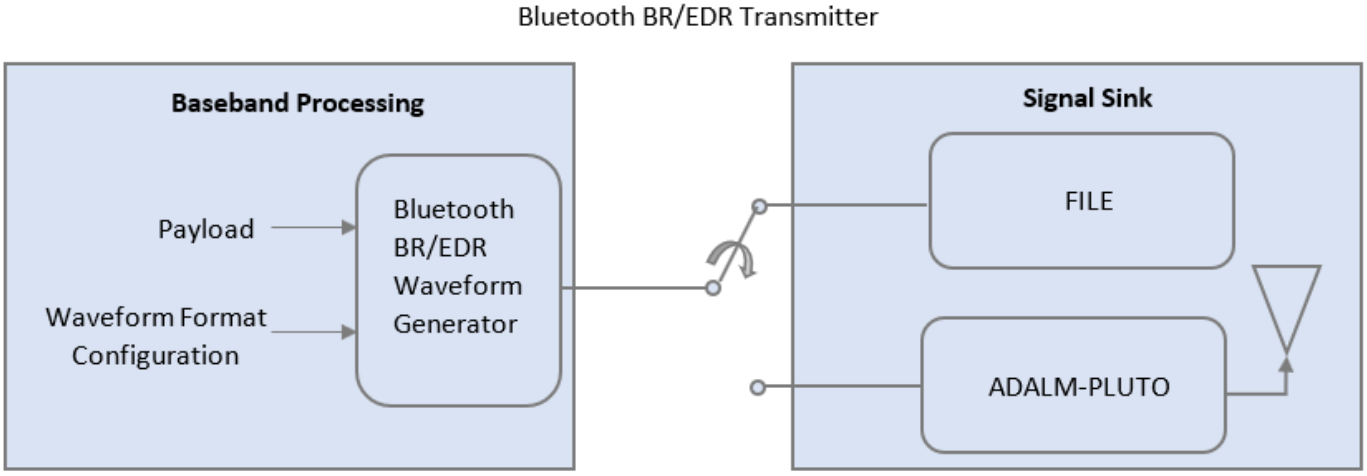


This figure shows the packet format for EDR mode



Bluetooth BR/EDR Waveform Generation and Transmission

This example shows how to generate Bluetooth BR/EDR waveforms according to the Bluetooth specification. The spectrum and spectrogram of the generated Bluetooth BR/EDR waveforms are visualized by using the spectrum analyzer. You can transmit the generated waveforms by using the ADALM-PLUTO radio or by writing them to a baseband file (*.bb).



Bluetooth BR/EDR Baseband Waveform Generation and Visualization

To configure the Bluetooth waveform generator for basic rate transmission, use the `bluetoothWaveformConfig` object.

```
cfg = bluetoothWaveformConfig;
cfg.Mode = BR; % Mode of transmission as one of BR, EDR2M and EDR3M
cfg.PacketType = 'FHS';     % Packet type
cfg.SamplesPerSymbol = 60; % Samples per symbol
cfg.WhitenInitialization = [0;0;0;0;0;1;1]; % Whiten initialization
```

To generate the Bluetooth BR/EDR waveforms, use the `bluetoothWaveformGenerator` function. Use `getPayloadLength` to determine the required payload length for the given configuration. Then use the payload length to create a random payload for transmission.

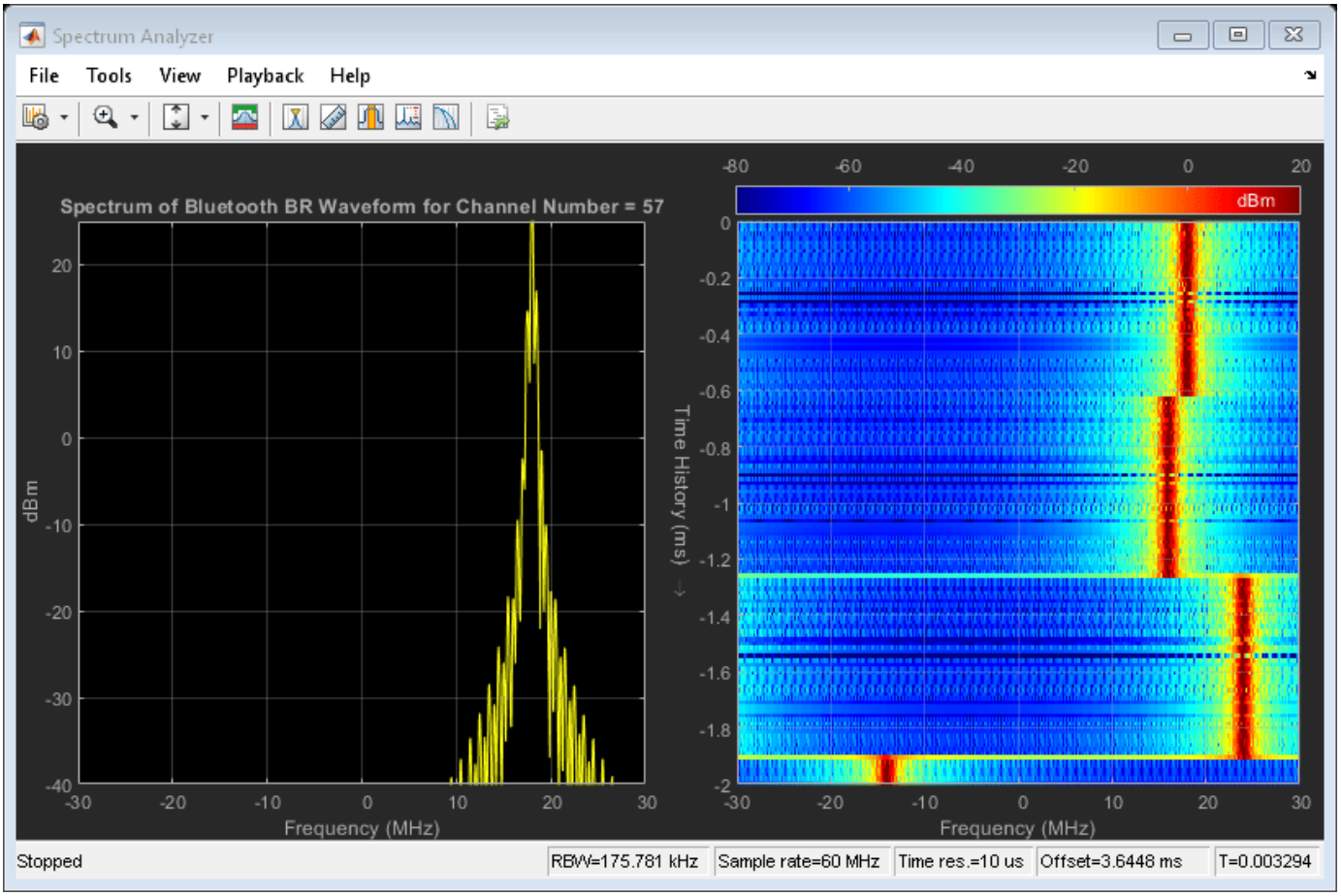
```
payloadLength = getPayloadLength(cfg); % Payload length in bytes
octetLength = 8;
dataBits = randi([0 1],payloadLength*octetLength,1); % Generate random payload bits
txWaveform = bluetoothWaveformGenerator(dataBits,cfg); % Create Bluetooth waveform
```

Compute the Bluetooth packet duration corresponding to the generated Bluetooth symbols by using the `bluetoothPacketDuration` function.

```
packetDuration = bluetoothPacketDuration(cfg.Mode,cfg.PacketType,payloadLength);
```

The `comm.PhaseFrequencyOffset` System(TM) object is used to perform a frequency shift for Bluetooth BR/EDR waveforms based on the channel number. In this example, the waveform is visualized by using the `dsp.SpectrumAnalyzer` (DSP System Toolbox) System object that selects a random channel number from the range 0 to 60 as sample rate used in this example is 60 MHz.

```
symbolRate = 1e6; % Symbol rate
sampleRate = symbolRate * cfg.SamplesPerSymbol;
numChannels = 10; % Number of channels
filterSpan = 8*any(strcmp(cfg.Mode,{'EDR2M','EDR3M'})); % To meet the spectral mask requirements

% Create and configure frequency offset System object
pfo = comm.PhaseFrequencyOffset('SampleRate',sampleRate);
% Create and configure spectrum analyzer System object
scope = dsp.SpectrumAnalyzer('ViewType','Spectrum and spectrogram',...
                'TimeResolutionSource','Property','TimeResolution',1e-5,...
                'SampleRate',sampleRate,'TimeSpanSource','Property',...
                'TimeSpan',2e-3,'FrequencyResolutionMethod','WindowLength',...
                'WindowLength',512,'AxesLayout','Horizontal','YLimits',[-40 25]);
% Loop over the number of channels to visaulize the frequency shift
for packetIdx = 1:numChannels
    channelNum = randsrc(1,1,0:60); % Generate random channel number
    freqIndex = channelNum - 39; % To visualize as a two sided spectrum
    pfo.FrequencyOffset = freqIndex*symbolRate; % Frequency shift
    hoppedWaveform = pfo(txWaveform(1:(packetDuration+filterSpan)*cfg.SamplesPerSymbol));
    scope.Title = ['Spectrum of Bluetooth ',cfg.Mode,...
                   ' Waveform for Channel Number = ', num2str(channelNum)];
    scope(hoppedWaveform);
end
% Release the System objects
release(scope);
```




```
release(pfo);
```

Transmitter Processing

Specify the signal sink as 'File' or 'ADALM-PLUTO'.

- **File**: Uses the `comm.BasebandFileWriter` System object to write a baseband file.
- **ADALM-PLUTO**: Uses the `sdrtx` (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio) function to create a `comm.SDRTxPluto` (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio) System object to transmit a live signal from the SDR hardware.

```
% Initialize the parameters required for signal sink
txCenterFrequency = 2445000000 ;  % In Hz, varies between 2.402e9 to 2.4
txFrameLength     = length(txWaveform);
txNumberOfFrames  = 1e4;
bbFileName        = 'bluetoothBRCaptures.bb';

% The default signal sink is 'File'
signalSink = File;

if strcmp(signalSink,'File')
    sigSink = comm.BasebandFileWriter('CenterFrequency',txCenterFrequency,...
```

```
        'Filename',bbFileName,...
        'SampleRate',sampleRate);
    sigSink(txWaveform); % Writing to a baseband file 'bluetoothBRCaptures.bb'
else % For 'ADALM-PLUTO'
    % Check if the pluto Hardware Support Package (HSP) is installed
    if isempty(which('plutoradio.internal.getRootDir'))
        error(message('comm_demos:common:NoSupportPackage', ...
                      'Communications Toolbox Support Package for ADALM-PLUTO Radio',...
                      ['<a href="https://www.mathworks.com/hardware-support/' ...
                      'adalm-pluto-radio.html">ADALM-PLUTO Radio Support From Communications Too
    end
    connectedRadios = findPlutoRadio; % Discover ADALM-PLUTO radio(s) connected to your computer
    radioID = connectedRadios(1).RadioID;
    sigSink = sdrtx('Pluto',...
        'RadioID',           radioID,...
        'CenterFrequency',   txCenterFrequency,...
        'Gain',              0,...
        'SamplesPerFrame',   txFrameLength,...
        'BasebandSampleRate',sampleRate);
    % The transfer of baseband data to the SDR hardware is enclosed in a
    % try/catch block. This implies that if an error occurs during the
    % transmission, the hardware resources used by the SDR System
    % object are released.
    currentFrame = 1;
    try
        while currentFrame <= txNumberOfFrames
            % Data transmission
            sigSink(txWaveform);
            % Update the counter
            currentFrame = currentFrame + 1;
        end
    catch ME
        release(sigSink);
        rethrow(ME);
    end
end

% Release the signal sink
release(sigSink);
```

In this example, you can generate and transmit Bluetooth BR/EDR waveforms by using ADALM-PLUTO or by writing the waveforms to a baseband file. The spectrum and spectogram of the generated Bluetooth BR/EDR waveforms is visualized by using a spectrum analyzer.

Further Exploration

You can use this example to transmit EDR packets by changing the mode of transmission. To decode the Bluetooth BR/EDR waveforms generated in this example, see “Bluetooth BR/EDR Waveform Reception Using SDR” on page 4-2.

Troubleshooting

General tips for troubleshooting SDR hardware and the Communications Toolbox Support Package for ADALM-PLUTO Radio can be found in “Common Problems and Fixes” (Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio).

Selected Bibliography

1 Bluetooth Special Interest Group (SIG). "Core System Package [BR/EDR Controller Volume]". *Bluetooth Core Specification*. Version 5.3, Volume 2. www.bluetooth.com

See Also

Functions

`bluetoothWaveformGenerator` | `bluetoothIdealReceiver`

Objects

`bluetoothWaveformConfig` | `bluetoothPhyConfig`

More About

- “Bluetooth BR/EDR Waveform Reception Using SDR” on page 4-2

Bluetooth LE Output Power and In-Band Emissions Tests

This example shows how to perform radio frequency (RF) physical layer (PHY) transmitter tests specific to output power and in-band emissions on Bluetooth® low energy (LE) transmitted waveforms by using Bluetooth® Toolbox features. This example also verifies whether these test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [3 on page 4-0].

Objectives of Bluetooth LE RF-PHY tests

The Bluetooth RF-PHY Test Specifications [3 on page 4-0] defined by Bluetooth Special Interest Group (SIG) includes RF-PHY tests for both transmitter and receiver. The objectives of these RF-PHY tests are to:

- Ensure interoperability between all Bluetooth devices.
- Ensure a basic level of system performance for all Bluetooth products.

Each test case has a specified test procedure and an expected outcome, which must be met by the implementation under test (IUT).

RF-PHY Transmitter Tests

This example performs output power and in-band emissions test measurements according to the Bluetooth RF-PHY Test Specifications [3 on page 4-0]. The output power measurement is designed to ensure that power levels are high enough to maintain interoperability with other Bluetooth devices and low enough to minimize interference within the ISM band. The in-band emission test is to verify that the level of unwanted signals within the frequency range from the transmitter do not exceed the specified limits. The test case IDs corresponding to the tests considered in this example are as follows:

Output Power:

- *RF-PHY/TRM/BV-01-C*: This test verifies the maximum peak and average power emitted from the IUT are within limits.
- *RF-PHY/TRM/BV-15-C*: This test verifies the maximum peak and average power emitted from the IUT are within limits, when transmitting with a constant tone extension (CTE).

In-band Emissions:

- *RF-PHY/TRM/BV-03-C*: This test verifies that the in-band emissions are within limits when the transmitter is operating with uncoded data at 1 Ms/s.
- *RF-PHY/TRM/BV-08-C*: This test verifies that the in-band emissions are within limits when the transmitter is operating with uncoded data at 2 Ms/s.

Configure the Test Parameters

You can change `phyMode`, `packetType`, `Fc`, `outputPower` and `numDominantFreq` parameters based on the PHY transmission mode, packet type, frequency of operation, output power and number of dominant frequencies, respectively.

```
% Select PHY transmission mode {'LE1M','LE2M'} according to the Bluetooth RF-PHY Test
% Specifications
phyMode = LE1M;
```

```
% Select packet type {'Disabled','ConnectionCTE'} for transmitting the
% packet with or without CTE. The CTE information field position is same
% for LE test packet and data packet. For CTE-based RF-PHY tests, set the
% packet type to 'ConnectionCTE'.
packetType = Disabled;

% Select frequency of operation for IUT based on the generic access profile
% (GAP) role(s) as shown in the table below.
%  ------------------------------------------------------------------------------
% | Operating |  Peripheral & Central Devices    |  Broadcaster & Observer Devices |
% | Frequency |                                  |                                 |
% | (MHz)     |----------------------------------|---------------------------------|
% |           | Output Power | In-band Emissions | Output Power | In-band Emissions|
% |           |     Test     |        Test       |     Test     |        Test      |
% |-----------|--------------|-------------------|--------------|------------------|
% | Lowest    |    2402      |       2406        |     2402     |       2402       |
% | Middle    |    2440      |       2440        |     2426     |       2440       |
% | Highest   |    2480      |       2476        |     2480     |       2480       |
%  ------------------------------------------------------------------------------
Fc = 2440e6; % Frequency of operation in Hz

payloadLength = 37; % Payload length in bytes, must be in the range [37
sps = 32; % Number of samples per symbol, minimum of 32 sps as per the test specifications

outputPower = 20; % Output power in dBm, must be in the range [-20,20]
```

Select number of dominant frequencies for in-band emissions test, must be in the range [1,78] and [1,74] for LE1M and LE2M modes, respectively.

```
numDominantFreq = 6;
```

The number of dominant frequencies represents the number of test frequencies near the operating frequency at which the in-band emissions test is to be performed. The number chosen in this example leads to a short simulation. For performing complete in-band emissions test, change the *numDominantFreq* parameter *to* maximum number of dominant frequencies as specified in the Section 4.4.2 of the Bluetooth RF-PHY Test Specifications [3 on page 4-0].

Generate Bluetooth LE Test Waveforms

Generate Bluetooth LE test waveform using `bluetoothTestWaveform` function and `bluetoothTestWaveformConfig` object.

```
testConfig = bluetoothTestWaveformConfig;
testConfig.PayloadType = 0; % Payload type for PRBS9 sequence
testConfig.PayloadLength = payloadLength;
testConfig.SamplesPerSymbol = sps;
testConfig.Mode = phyMode;
testConfig.PacketType = packetType;
if strcmp(packetType,'ConnectionCTE')
    % Length of CTE in 8 microseconds units
    testConfig.CTELength = 2; % Must be in the range [2, 20]
    % Type of CTE
    testConfig.CTEType = [0;0]; % [0;0] for Angle of Arrival CTE
end
waveform = bluetoothTestWaveform(testConfig);
```

```
% Calculate sampling rate in Hz based on PHY transmission mode
Rsym = 1e6;
if strcmp(phyMode,'LE2M')
    Rsym = 2e6;
end
Fs = Rsym*sps;

% Apply frequency upconversion to obtain a passband signal for the
% specified frequency of operation.
maxFreq = 2485e6; % in Hz
interpFactor = ceil(2*maxFreq/Fs); % Interpolation factor for upconversion to
                                   % cover Bluetooth LE RF frequency band (2400e6 to 2485e6)

% Change the stopband frequency in Hz based on the PHY transmission mode
stopbandFreq = 2e6;
if strcmp(phyMode,'LE2M')
    stopbandFreq = 4e6;
end

% Create a digital upconverter System object
upConv = dsp.DigitalUpConverter(...
              'InterpolationFactor', interpFactor,...
              'SampleRate', Fs,...
              'Bandwidth', 2e6,...
              'StopbandAttenuation', 44,...
              'PassbandRipple',0.5,...
              'CenterFrequency',Fc,...
              'StopbandFrequencySource','Property',...
              'StopbandFrequency', stopbandFreq);

% Upconvert the baseband waveform to passband
dBdBmConvFactor = 30;
scalingFactor = 10^((outputPower-dBdBmConvFactor)/20);
upConvWaveform = scalingFactor*upConv(waveform);
```

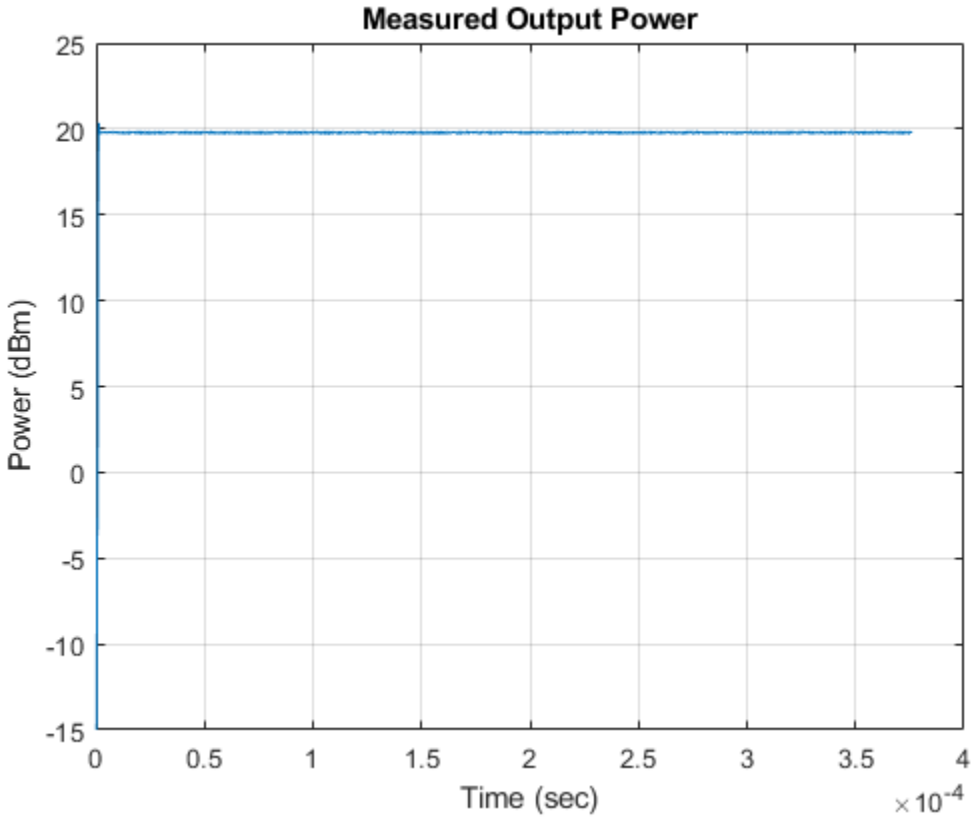
Perform Output Power Test

```
rbwOutputPower = 3e6; % Resolution bandwidth, in Hz
```

To perform power measurement in the time domain, frequency span must be set to 0. Span 0 can be replicated by taking power values from the spectrogram at frequency of operation (Fc). Frequency limits are considered starting from frequency of operation (Fc) up to maximum frequency in the frequency band.

```
[P,F,T] = pspectrum(upConvWaveform,interpFactor*Fs,'spectrogram',...
                                'TimeResolution',1/rbwOutputPower,...
                                'FrequencyLimits',[Fc,maxFreq]);
powerAtFc = P(1,:); % Extract power values at Fc (F(1) = Fc)

% Calculate average power, AVGPOWER over at least 20% to 80% of the
% duration of the burst as specified in Section 4.4.1 of the Bluetooth
% RF-PHY Test Specifications [3].
powerAvgStartIdx = floor(0.2*length(powerAtFc));
powerAvgStopIdx = floor(0.8*length(powerAtFc));
avgPower = 10*log10(mean(powerAtFc(powerAvgStartIdx:powerAvgStopIdx)))+dBdBmConvFactor;

% Calculate peak power, PEAKPOWER
```

```
peakPower = 10*log10(max(powerAtFc))+dBdBmConvFactor;

% Plot power vs time
powerAtFcdBm = 10*log10(powerAtFc) + dBdBmConvFactor;
figure,plot(T,powerAtFcdBm)
grid on;
xlabel('Time (sec)');
ylabel('Power (dBm)');
title('Measured Output Power');
```




```
% Pass verdict - All measured values shall fulfill the following conditions:
%
% * Peak power <= (Average power + 3 dB)
% * -20dBm <= Average power <= 20dBm
%
fprintf('Measured average power and peak power are %f dBm and %f dBm, respectively.\n',avgPower,p
```

```
Measured average power and peak power are 19.792338 dBm and 20.362792 dBm, respectively.
```

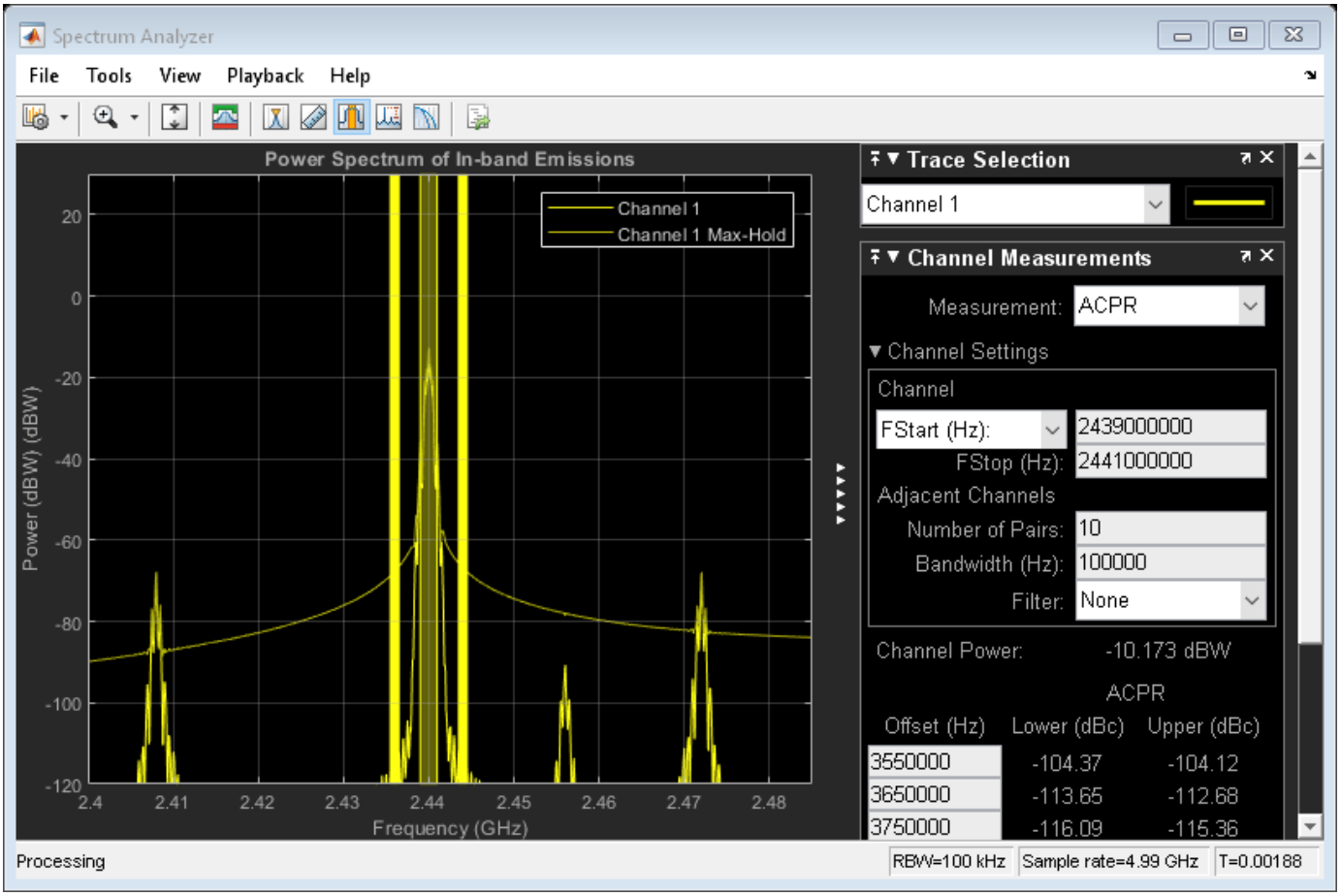
```
if (-20 <= avgPower <= 20) && (peakPower <= (avgPower+3)) %#ok<CHAIN>
    fprintf('Output power test passed.\n');
else
    fprintf('Output power test failed.\n');
end
```

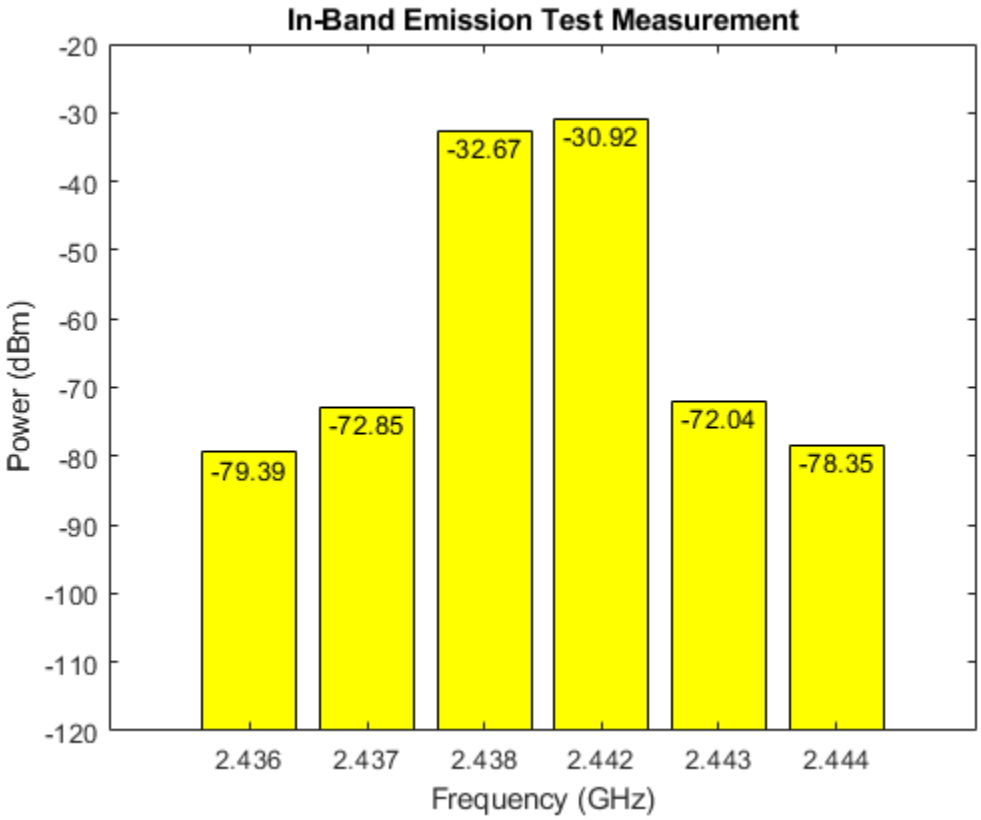
```
Output power test passed.
```

Perform In-band Emissions Test

```
if strcmp(packetType,'Disabled')
    % The function, <matlab:edit('helperBLEInbandEmissionsParams.m')
    % helperBLEInbandEmissionsParams.m>, is configured to generate dominant
    % test frequency parameters.
    [testFreq,idx1,idx2] = helperBLEInbandEmissionsParams(Fc,numDominantFreq,phyMode);

    % For each test frequency measure the power levels at the following 10
    % frequencies.
    numOffsets = 10;
    freqOffset = -450e3+(0:numOffsets-1)*100e3;
    adjChannelFreqOffsets = (freqOffset+testFreq-Fc).';

    % Create and configure a spectrum analyzer for the waveform sampling rate
    % and a resolution bandwidth of 100 kHz as specified in Section 4.4.2 of
    % the Bluetooth RF-PHY Test Specifications.
    rbw = 100e3; % Resolution bandwidth, in Hz
    spectrumScope = dsp.SpectrumAnalyzer( ...
        'SampleRate',            Fs*interpFactor,...
        'SpectralAverages',      10, ...
        'YLimits',               [-120 30], ...
        'Title',                 'Power Spectrum of In-band Emissions',...
        'YLabel',                'Power (dBW)',...
        'SpectrumUnits',         'dBW',...
        'ShowLegend',            true,...
        'FrequencySpan',         'Start and stop frequencies',...
        'StartFrequency',         2400e6,...
        'StopFrequency',          maxFreq,...
        'RBWSource',             'Property',...
        'RBW',                    rbw,...
        'PlotMaxHoldTrace',       true,...
        'PlotAsTwoSidedSpectrum', false);

    spectrumScope.ChannelMeasurements.Enable = true;
    spectrumScope.ChannelMeasurements.Algorithm = 'ACPR';
    spectrumScope.ChannelMeasurements.CenterFrequency = Fc;
    spectrumScope.ChannelMeasurements.Span = 2e6; % Main channel bandwidth
    spectrumScope.ChannelMeasurements.AdjacentBW = 1e5; % Adjacent channel bandwidth
    spectrumScope.ChannelMeasurements.NumOffsets = numOffsets;

    % Compute adjacent channel power ratio (ACPR) for the transmitted waveform
    acpr = zeros(numOffsets,numDominantFreq);
    for i = 1:numDominantFreq
        % Assign the 10 frequency offsets at each test frequency to ACPR Offsets
        spectrumScope.ChannelMeasurements.ACPROffsets = adjChannelFreqOffsets(:,i);

        % Estimate the power spectrum of the transmitted waveform using the spectrum analyzer
        spectrumScope(upConvWaveform);

        % Compute ACPR
        data = getMeasurementsData(spectrumScope); % Get the measurements data
        mainChannelPower = data.ChannelMeasurements.ChannelPower; % Main channel power at Fc
        acpr(:,i) = data.ChannelMeasurements.ACPRUpper; % Extract the ACPR values
    end

    % Power levels at 10 frequency offsets at each test frequency are
    % calculated by adding main channel power to ACPR.
```

```
        adjChannelPower = acpr(:,1:numDominantFreq) + mainChannelPower;

        % Compute the power at each test frequency by adding all the powers
        % measured at 10 frequency offsets.
        adjPowerAtTestFreq = 10*log10(sum(10.^(adjChannelPower(:,1:numDominantFreq)/10))) + dBdBmConv

        % Plot the adjacent channel powers
        tick = 1:numel(adjPowerAtTestFreq);
        ticklabel = testFreq/1e9;
        figure;
        bar(adjPowerAtTestFreq, 'BaseValue', -120, 'FaceColor', 'yellow');
        set(gca, 'XTick', tick, 'XTickLabel', ticklabel, 'YLim', [-120 -20]);
        for i = tick
            text(i, adjPowerAtTestFreq(i), sprintf('%0.2f',adjPowerAtTestFreq(i)), ...
                'HorizontalAlignment', 'Center', 'VerticalAlignment', 'Top');
        end
        title('In-Band Emission Test Measurement');
        xlabel('Frequency (GHz)');
        ylabel('Power (dBm)');

        % Pass verdict- All measured values shall fulfill the following conditions:
        % For LE1M PHY transmission mode
        %
        % * powerAtTestFreq <= -20 dBm for testFreq = Fc ± 2 MHz
        % * powerAtTestFreq <= -30 dBm for testFreq = Fc ± [3+n] MHz; where
        %   n=0,1,2,...
        %
        % For LE2M PHY transmission mode
        %
        % * powerAtTestFreq <= -20 dBm for testFreq = Fc ± 4 MHz AND testFreq = Fc
        % ± 5 MHz
        % * powerAtTestFreq <= -30 dBm for testFreq = Fc ± [6+n] MHz; where
        %   n=0,1,2,...
        %
        for i = 1:numDominantFreq
            fprintf('Measured power at test frequency (Fc%+de6) is %.3f dBm.\n',(Fc-testFreq(i))*1e-6
        end
        if (all(adjPowerAtTestFreq(idx1) <= -20)||isempty(idx1)) && (all(adjPowerAtTestFreq(idx2) <=
            fprintf('In-band emissions test passed.\n');
        else
            fprintf('In-band emissions test failed.\n');
        end
    end
```


Spectrum Analyzer
File
Tools
View
Playback
Help
Power Spectrum of In-band Emissions
Channel 1
Channel 1 Max-Hold
Power (dBW) (dBW)
Frequency (GHz)
Trace Selection
Channel 1
Channel Measurements
Measurement:
ACPR
Channel Settings
Channel
FStart (Hz):
2439000000
FStop (Hz):
2441000000
Adjacent Channels
Number of Pairs:
10
Bandwidth (Hz):
100000
Filter:
None
Channel Power:
-10.173 dBW
ACPR
Offset (Hz)
Lower (dBc)
Upper (dBc)
3550000
-104.37
-104.12
3650000
-113.65
-112.68
3750000
-116.09
-115.36
Processing
RBW=100 kHz
Sample rate=4.99 GHz
T=0.00188




```
Measured power at test frequency (Fc+4e6) is -79.393 dBm.
Measured power at test frequency (Fc+3e6) is -72.852 dBm.
Measured power at test frequency (Fc+2e6) is -32.670 dBm.
Measured power at test frequency (Fc-2e6) is -30.924 dBm.
Measured power at test frequency (Fc-3e6) is -72.038 dBm.
Measured power at test frequency (Fc-4e6) is -78.351 dBm.

In-band emissions test passed.
```

This example demonstrated the transmitter test measurements specific to output power and in-band emissions on Bluetooth LE transmitted waveforms as per the Bluetooth RF-PHY Test Specifications [3 on page 4-0].

Appendix

This example uses this helper function:

- helperBLEInbandEmissionsParams.m: Generate test frequency(s) for in-band emissions test

Selected Bibliography

1 Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com.

2 Bluetooth Special Interest Group (SIG). "Core System Package [Low Energy Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume 6. https://www.bluetooth.com.

3 Bluetooth RF-PHY Test Specification, Section 4.4.1.

See Also

Functions
`bluetoothTestWaveform`

Objects
`bluetoothTestWaveformConfig`

More About

- “Bluetooth LE Blocking, Intermodulation and Carrier-to-Interference Performance Tests” on page 4-26
- “Bluetooth LE Modulation Characteristics, Carrier Frequency Offset and Drift Tests” on page 4-33
- “Bluetooth LE IQ samples Coherency and Dynamic Range Tests” on page 4-53

Bluetooth LE Blocking, Intermodulation and Carrier-to-Interference Performance Tests

This example shows how to perform radio frequency (RF) physical layer (PHY) receiver tests specific to blocking, intermodulation and carrier-to-interference (C/I) according to Bluetooth RF-PHY Test Specifications [3 on page 4-0]. This example also verifies whether these test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [3 on page 4-0].

Objectives of Bluetooth LE RF-PHY tests

The Bluetooth RF-PHY Test Specifications [3 on page 4-0] defined by Bluetooth Special Interest Group (SIG) includes RF-PHY tests for both transmitter and receiver. The objectives of these RF-PHY tests are to:

- Ensure interoperability between all Bluetooth devices
- Ensure a basic level of system performance for all Bluetooth products.

Each test case has a specified test procedure and an expected outcome, which must be met by the implementation under test (IUT).

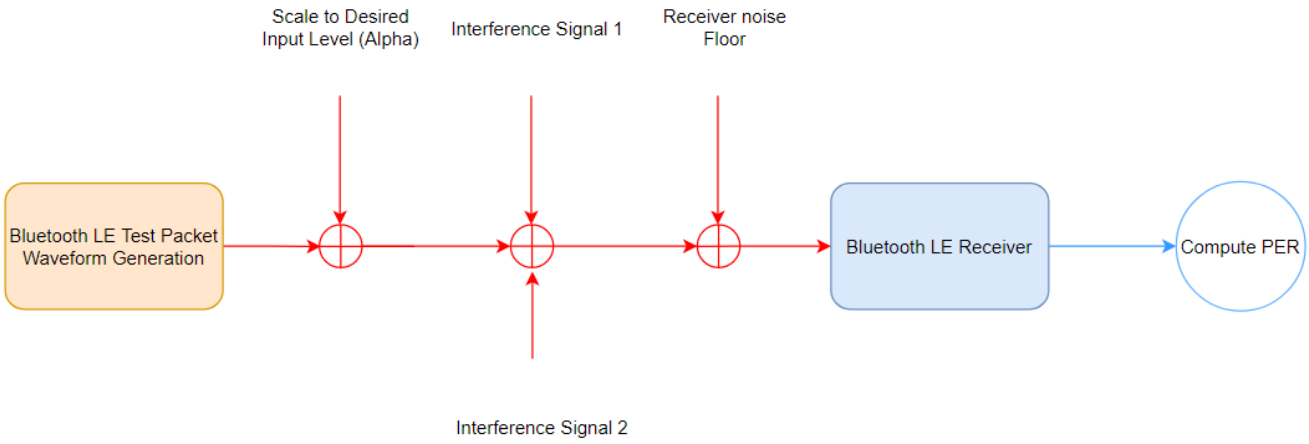
RF-PHY Receiver Tests

The Bluetooth receiver tests are designed to ensure that the IUT can receive data over a range of conditions where the transmitted signal has high power, and in presence of both in-band and out-of-band interference with a defined packet error rate (PER). This example covers three Bluetooth LE RF-PHY receiver tests for blocking, intermodulation and C/I performance as per the Bluetooth RF-PHY Test Specifications [3 on page 4-0].

- **Blocking Performance**: The blocking performance test verifies the receiver performance in the presence of out-of-band interfering signals i.e. operating outside the 2400 MHz - 2483.5 MHz band.
- **Intermodulation Performance**: The intermodulation performance test verifies the receiver performance in presence of unwanted signals nearby in frequency.
- **C/I Performance**: The C/I performance test verifies the receiver performance in presence of adjacent and co-channel interfering signals.

All the above RF-PHY tests are necessary because the wanted signal often will not be the only signal transmitting in the given frequency range.

The following block diagram summarizes the example flow.



1. Generate test packets and pass through bleWaveformGenerator to generate Bluetooth LE test waveform.
2. Perform frequency upconversion to obtain a passband signal.
3. Scale the transmitted signal to a desired input level.
4. Add the interference signal(s) depending on the performance test.
5. Add white gaussian noise based on receiver noise floor.
6. At the receiver, down convert the signal and then demodulate, decode and perform CRC check.
7. Measure the PER based on CRC check and then compare it with the reference PER.

Initialize the Simulation Parameters

You can change `rxPerformanceTest`, `phyMode` and `Fc` parameters based on the receiver performance test, PHY transmission mode and frequency of operation, respectively.

```
rxPerformanceTest = Intermodulation; % Select one from the set {'C/I', 'Blocking', 'Interm
% Select PHY transmission mode as per Bluetooth RF-PHY Test Specifications
phyMode = LE1M; % {LE1M, LE2M, LE500K, LE125K} for C/I
                % {LE1M, LE2M} for blocking and intermodulation
% Select frequency of operation for IUT based on the performance test and
% generic access profile (GAP) role(s) as shown in the table below.
% ------------------------------------------------------------------------------
% Operating | Peripheral & Central Devices    | Broadcaster & Observer Devices   |
% Frequency |                                 |                                  |
% (MHz)     |---------------------------------|----------------------------------|
%           | C/I  | Blocking |Intermodulation| C/I  | Blocking | Intermodulation|
% ----------|------|----------|---------------|------|----------|----------------|
%   Lowest  | 2406 |    -     |    2402       | 2402 |    -     |    2402        |
%   Middle  | 2440 |    2426  |    2440       | 2426 |    2426  |    2426        |
%   Highest | 2476 |    -     |    2480       | 2480 |    -     |    2480        |
% ------------------------------------------------------------------------------
Fc = 2426e6; % Frequency of operation in Hz
payloadLength = 37; % Payload length in bytes, must be in the range [37
```

```
sps = 40; % Number of samples per symbol

% Calculate sampling rate in Hz based on PHY transmission mode
Rsym = 1e6;
if strcmp(phyMode,'LE2M')
    Rsym = 2e6;
end
Fs = Rsym*sps;
```

Generate Baseband Waveforms

Generate Bluetooth LE test waveforms using `bluetoothTestWaveform` function and `bluetoothTestWaveformConfig` object.

```
% Generate a wanted signal which is always a modulated carrier with a PRBS9
% payload
wantedTestWaveformConfig = bluetoothTestWaveformConfig;
wantedTestWaveformConfig.PayloadType = 0; % Payload type for PRBS9 sequence
wantedTestWaveformConfig.PayloadLength = payloadLength;
wantedTestWaveformConfig.SamplesPerSymbol = sps;
wantedTestWaveformConfig.Mode = phyMode;
wantedWaveform = bluetoothTestWaveform(wantedTestWaveformConfig);

% Generate an interference signal #1 which is a modulated carrier with a
% PRBS15 payload
interferenceTestWaveformConfig = bluetoothTestWaveformConfig;
interferenceTestWaveformConfig.PayloadType = 3; % Payload type for PRBS15 sequence
interferenceTestWaveformConfig.PayloadLength = payloadLength;
interferenceTestWaveformConfig.SamplesPerSymbol = sps;
interferenceTestWaveformConfig.Mode = phyMode;
interferenceWaveform = bluetoothTestWaveform(wantedTestWaveformConfig);
```

Frequency Upconversion

Apply frequency upconversion to obtain a passband signal for the specified frequency of operation.

```
% Interpolation factor for upconversion to cover Bluetooth LE RF frequency band
% (2400e6 to 2485e6)
interpFactor = ceil(2*2485e6/Fs);

% Create a digital upconverter System object
upConv = dsp.DigitalUpConverter(...
              'InterpolationFactor',interpFactor,...
              'SampleRate',Fs,...
              'Bandwidth',2e6,...
              'StopbandAttenuation',44,...
              'PassbandRipple',0.5,...
              'CenterFrequency',Fc);

% Upconvert the baseband waveform to passband
wantedWaveformUp = upConv([wantedWaveform;zeros(8*sps,1)]);
```

Generate Test Parameters

Test parameters are generated based on performance test, frequency of operation and PHY transmission mode. The function, helperBLETestParamGenerate.m, is used to generate all the interference frequencies and corresponding scaling factors (alpha, beta, gamma) for selected receiver performance test.

```
[alpha,beta,gamma,interferenceFreq1,interferenceFreq2] = ...
                helperBLETestParamGenerate(rxPerformanceTest,Fc,phyMode);
```

Repeat test parameters based on the number of packets used for simulation.

```
pktCnt = 10; % Number of packets
maxInterferenceParams = min(length(interferenceFreq1),pktCnt); % Maximum number of interference p

% Repeat all the interference parameters such that PER can be averaged over
% the entire range of interference frequencies for selected receiver
% performance test.
repFact = ceil(pktCnt/maxInterferenceParams); % Repetition factor
betaRep = repmat(beta,repFact,1);
gammaRep = repmat(gamma,repFact,1);
interferenceFreq1Rep = repmat(interferenceFreq1,repFact,1);
interferenceFreq2Rep = repmat(interferenceFreq2,repFact,1);
```

Test Simulation

In this example, all the three Bluetooth LE RF-PHY performance tests are simulated as follows:

- For Blocking performance, there will be only one interference signal i.e. interference signal #2. So, the scaling factor (beta) for interference signal #1 is zero.
- For Intermodulation performance, there will be two interference signals.
- For C/I performance, there will be only one interference signal i.e. interference signal #1. So, the scaling factor (gamma) for interference signal #2 is zero.

```
% Upconvert and store the interference waveform #1 based on buffer
% size, so that the stored interference waveforms can be reused if
% the packet count exceeds the buffer size.
interferenceWaveform1Up = zeros(length(wantedWaveformUp),maxInterferenceParams);
if any(strcmp(rxPerformanceTest,{'C/I','Intermodulation'}))
    for i=1:maxInterferenceParams
        release(upConv)
        upConv.CenterFrequency = interferenceFreq1Rep(i);
        interferenceWaveform1Up(:,i) = upConv([interferenceWaveform;zeros(8*sps,1)]);
    end
end

% Initialize a variable for reusing the interference waveform #1
j = rem(1:pktCnt,maxInterferenceParams);
j(j == 0) = maxInterferenceParams;

% Create a digital down converter System object
downConv = dsp.DigitalDownConverter(...
              'DecimationFactor',interpFactor,...
              'SampleRate',Fs*interpFactor,...
              'Bandwidth',2e6,...
              'StopbandAttenuation',44,...
              'PassbandRipple',0.5,...
              'CenterFrequency',Fc);

% Create automatic gain control System object
agc = comm.AGC('DesiredOutputPower',1);

% Create a thermal noise System object
NF = 12; % Noise figure (dB)
```

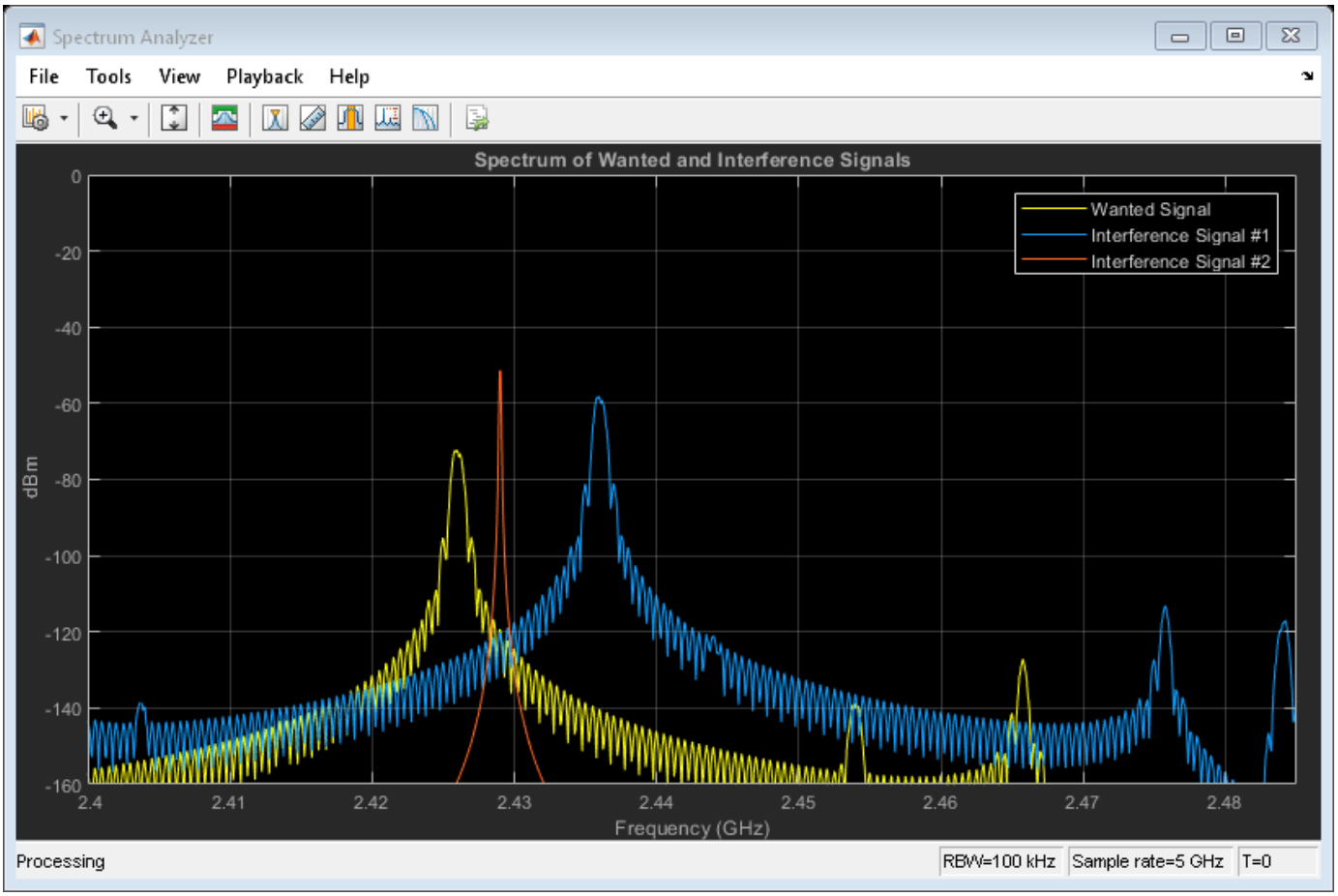
```
thNoise = comm.ThermalNoise('NoiseMethod','Noise figure',...
                            'SampleRate',interpFactor*Fs,...
                            'NoiseFigure',NF);

% Time vector to generate sinusoidal unmodulated interference signal i.e.
% interference signal #2.
t = (0:(length(wantedWaveformUp)-1)).'/(interpFactor*Fs);
pktLost = 0; % Initialize counter
for i=1:pktCnt

    % Generate an interference waveform #2 which is a sinusoidal
    % unmodulated signal. The sqrt(2) factor ensures that the power of the
    % sinusoidal signal is normalized.
    interferenceWaveform2 = sqrt(2)*sin(2*pi*interferenceFreq2Rep(i)*t);

    % Add the interference signals to wanted signal
    rxWaveform = alpha*wantedWaveformUp + betaRep(i)*interferenceWaveform1Up(:,j(i)) + gammaRep(i
    chanOut = thNoise(complex(rxWaveform)); % Add thermal noise to the signal
    downConvOut = downConv(real(chanOut)); % Perform frequency down conversion
    agcOut = agc(downConvOut); % Apply AGC
    [payload,accessAddr] = bleIdealReceiver(agcOut,'Mode',phyMode,...
                                'SamplesPerSymbol',sps,'WhitenStatus','Off'); % Extract message i
    [crcFail,pdu] = helperBLETestPacketValidate(payload,accessAddr); % Validate the BLE test pack
    pktLost = pktLost + crcFail;
end

% Determine the PER
per = pktLost/pktCnt;
```

Spectrum Visualization

Create and configure a spectrum analyzer and show the spectrum of last transmitted wanted signal and interference signal(s) based on the receiver performance test.

```
% Setup spectrum viewer
spectrumScope = dsp.SpectrumAnalyzer( ...
                    'SampleRate',           interpFactor*Fs,...
                    'SpectralAverages',     10,...
                    'YLimits',              [-160 0], ...
                    'Title',                'Spectrum of Wanted and Interference Signals',...
                    'SpectrumUnits',        'dBm',...
                    'NumInputPorts' ,       2,...
                    'ChannelNames',         {'Wanted Signal','Interference Signal'},...
                    'ShowLegend',           true,...
                    'FrequencySpan',        'Start and stop frequencies',...
                    'StartFrequency',       2400e6,...
                    'StopFrequency',        2485e6,...
                    'RBWSource',            'Property',...
                    'RBW',                  1e5,...
                    'PlotAsTwoSidedSpectrum',false);
if strcmp(rxPerformanceTest,'C/I')
    spectrumScope(alpha*wantedWaveformUp,betaRep(end)*interferenceWaveform1Up(:,end))
elseif strcmp(rxPerformanceTest,'Blocking')
    spectrumScope.StartFrequency = 30e6;
    spectrumScope(alpha*wantedWaveformUp,gammaRep(end)*interferenceWaveform2)
else
    spectrumScope.NumInputPorts = 3;
    spectrumScope.ChannelNames = {'Wanted Signal','Interference Signal #1','Interference Signal #
```

```
    spectrumScope(alpha*wantedWaveformUp,betaRep(end)*interferenceWaveform1Up(:,end),gammaRep(end
end
```



Reference Results

This section generates the reference PER values for each PHY transmission mode based on the payload length as specified in section 6.4 of the Bluetooth RF-PHY Test Specifications [3 on page 4-0].

```
berTable = [0.1 0.064 0.034 0.017]*0.01;
if(payloadLength <= 37)
    refBER = berTable(1);
elseif(payloadLength <= 63)
    refBER = berTable(2);
elseif(payloadLength <= 127)
    refBER = berTable(3);
else
    refBER = berTable(4);
end
accessAddLen = 4; % Access address length in bytes
crcLengthBytes = 3; % CRC length in bytes
pduHeaderLen = 2; % Header length in bytes
refPER = 1-(1-refBER)^((payloadLength+accessAddLen+pduHeaderLen+crcLengthBytes)*8);
fprintf('Measured PER and reference PER for payload length of %d bytes are %f, %f respectively.\n
```

```
Measured PER and reference PER for payload length of 37 bytes are 0.000000, 0.308010 respectively
```

```
if per <= refPER
    fprintf('%s performance test passed.\n',rxPerformanceTest);
else
    fprintf('%s performance test failed.\n',rxPerformanceTest);
end
```

```
Intermodulation performance test passed.
```

Appendix

This example uses the following helper functions:

- helperBLETestParamGenerate.m: Generate Bluetooth LE test parameters specific to blocking, intermodulation, and C/I
- helperBLETestPacketValidate.m: Validate Bluetooth LE test packets

Selected Bibliography

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com.
2. Bluetooth Special Interest Group (SIG). "Core System Package [Low Energy Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume 6. https://www.bluetooth.com.
3. Bluetooth RF-PHY Test Specification, Section 6.4.

See Also

Functions
`bluetoothTestWaveform`

Objects
`bluetoothTestWaveformConfig`

More About

- “Bluetooth LE Output Power and In-Band Emissions Tests” on page 4-17
- “Bluetooth LE Modulation Characteristics, Carrier Frequency Offset and Drift Tests” on page 4-33
- “Bluetooth LE IQ samples Coherency and Dynamic Range Tests” on page 4-53

Bluetooth LE Modulation Characteristics, Carrier Frequency Offset and Drift Tests

This example shows how to perform Bluetooth® low energy (LE) radio frequency (RF) physical layer (PHY) transmitter tests specific to modulation characteristics, carrier frequency offset, and drift using Bluetooth® Toolbox. The test measurements compute frequency deviation, carrier frequency offset, and drift values. This example also verifies whether these test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [1 on page 4-0].

Objectives of LE RF-PHY Tests

The Bluetooth RF-PHY Test Specifications [1 on page 4-0] defined by Bluetooth Special Interest Group (SIG) includes RF-PHY tests for both transmitter and receiver. The objectives of these RF-PHY tests are to:

- Ensure interoperability between all Bluetooth devices.
- Ensure a basic level of system performance for all Bluetooth products.

Each test case has a specified test procedure and an expected outcome, which must be met by the implementation under test (IUT).

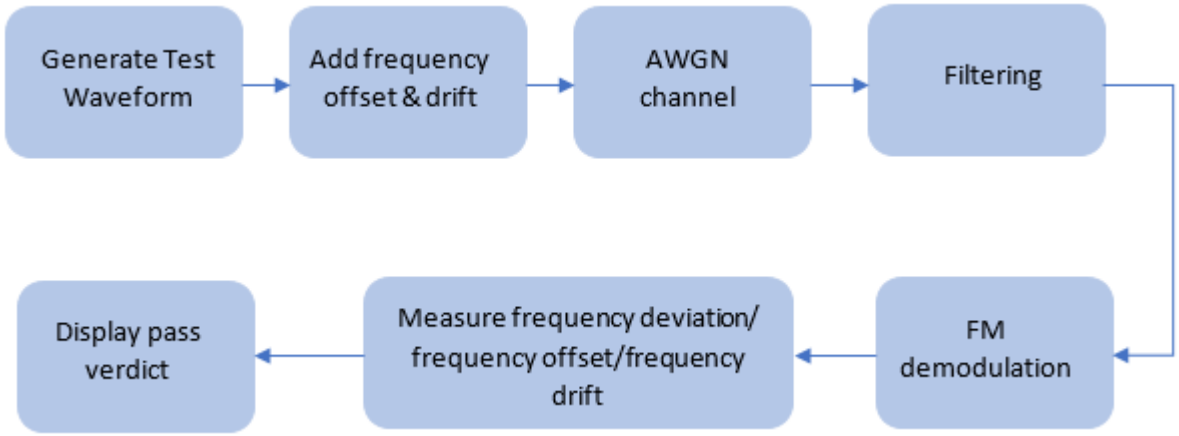
RF-PHY Transmitter Tests

The main aim of transmitter test measurements is to ensure that the transmitter characteristics are within the specified limits as specified in the test specifications [1 on page 4-0]. This example includes transmitter tests relevant to modulation characteristics, carrier frequency offset, and drift. This table shows various RF-PHY transmitter tests performed in this example.

Conformance Test	Test Case ID	Transmission Mode
Modulation Characteristics	RF-PHY/TRM/BV-09-C	LE 1M, uncoded data at 1 Mbps
	RF-PHY/TRM/BV-11-C	LE 2M, uncoded data at 2 Mbps
	RF-PHY/TRM/BV-13-C	LE 125K, coded data at 125 Kbps
Carrier Frequency Offset and Drift	RF-PHY/TRM/BV-06-C	LE 1M, uncoded data at 1 Mbps
	RF-PHY/TRM/BV-12-C	LE 2M, uncoded data at 2 Mbps
	RF-PHY/TRM/BV-14-C	LE 125K, coded data at 125 Kbps
	RF-PHY/TRM/BV-16-C	LE 1M with Constant Tone Extension, uncoded data at 1 Mbps
	RF-PHY/TRM/BV-17-C	LE 2M with Constant Tone Extension, uncoded data at 2 Mbps

Transmitter Test Procedure

This block diagram summarizes the test procedure for transmitter tests relevant to modulation characteristics, carrier frequency offset, and drift.



Generate Bluetooth LE test packets by using the `bluetoothTestWaveform` function. The test waveforms required for different test IDs are:

Test Case ID	Test Waveforms
RF-PHY/TRM/BV-09-C & RF-PHY/TRM/BV-11-C	Generate two Bluetooth LE test packets with repetitive sequence of 11110000b and 10101010b in transmission order.
RF-PHY/TRM/BV-13-C & RF-PHY/TRM/BV-14-C	Generate one Bluetooth LE test packet with repetitive sequence of 11111111b.
RF-PHY/TRM/BV-06-C & RF-PHY/TRM/BV-12-C	Generate one Bluetooth LE test packet with repetitive sequence of 10101010b.
RF-PHY/TRM/BV-16-C & RF-PHY/TRM/BV-17-C	Generate one Bluetooth LE test packet with repetitive sequence of 11110000b and CTE.

Configure the Test Parameters

Specify the RF-PHY test configuration parameters by using the `bluetoothRFPHYTestConfig` object. The function,helperBLEModulationTestConfig.m, can be configured to generate test parameters.

```
testConfig = bluetoothRFPHYTestConfig;
testConfig.Test = Modulation charact...;
testConfig.Mode = LE1M;
testConfig.PayloadLength = 240;  % Payload length in bytes, must be in th

% The CTE information field position is same for LE test packet and data packet.
% For CTE-based RF-PHY tests, set the packet type to 'ConnectionCTE'.
testConfig.PacketType = Disabled;
if strcmp(testConfig.PacketType,'ConnectionCTE')
    testConfig.CTELength = 13;  % CTE length in 8 microseconds units,
end                             % must be in the range [13,20]
testConfig.SamplesPerSymbol = 32;   % Number of samples per symbol, minimum of 32 samples
```

```
                                % per symbol as per the test specifications
testConfig.InitialFrequencyOffset =  23000                ;  % In Hz, must be in the ra
testConfig.CarrierDrift =            0                    ;  % In Hz, must be in the range [-50e3,5

% Generate test parameters
[testParams,wgParams] = helperBLEModulationTestConfig(testConfig);
```

Simulate Transmitter Tests

To simulate the transmitter tests, perform these steps:

1. Generate Bluetooth LE test packet waveform by using the `bluetoothTestWaveform` function.
2. Add frequency offset, which includes initial frequency offset, and drift to the waveform by using the `comm.PhaseFrequencyOffset` System object.
3. Add thermal noise by using the `comm.ThermalNoise` System object.
4. Perform filtering on the noisy waveform by using helperModulationTestFilterDesign helper function.
5. Perform FM demodulation on the filtered waveform.
6. Perform test measurement and display the pass verdict.

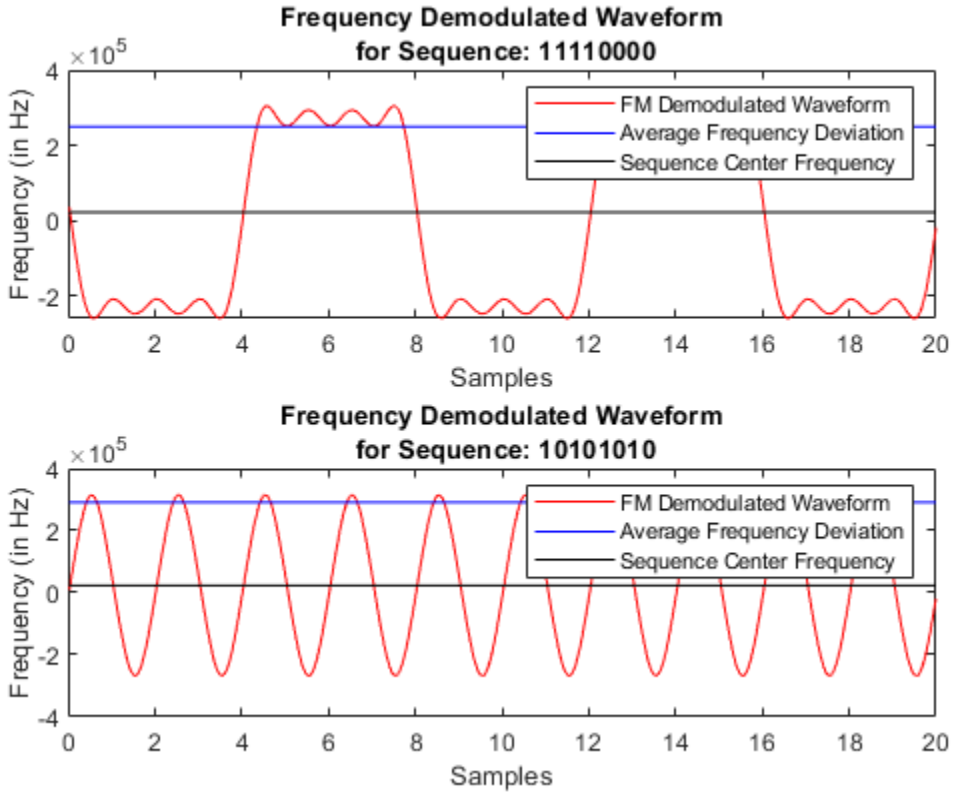
```
if strcmp(testConfig.PacketType,'ConnectionCTE')
  [~,testWfmSymLen] = bluetoothPacketDuration(testConfig.Mode,testConfig.PacketType,...
      testConfig.PayloadLength,testConfig.CTELength);
else
  [~,testWfmSymLen] = bluetoothPacketDuration(testConfig.Mode,testConfig.PacketType,...
      testConfig.PayloadLength);
end
testWfmLen = testWfmSymLen*testConfig.SamplesPerSymbol;
driftRate = testConfig.CarrierDrift/testWfmLen;% Drift rate
freqDrift = driftRate*(0:1:(testWfmLen-1))';% Frequency drift

% Frequency offset and frequency drift
freqOffset = testConfig.InitialFrequencyOffset + freqDrift;

% Create a phase frequency offset System object
pfo = comm.PhaseFrequencyOffset('FrequencyOffset',freqOffset,...
    'SampleRate',testParams.sampleRate);
% Create a thermal noise System object

NF = 12; % Noise figure in dB
thNoise = comm.ThermalNoise('NoiseMethod','Noise figure',...
                            'SampleRate',testParams.sampleRate,...
                            'NoiseFigure',NF);
filtDesign = helperModulationTestFilterDesign(testParams.phyMode,testParams.sps);
filtTestWfm = zeros(testWfmLen,testParams.numOfTestSeqs);
for wfmIdx = 1:testParams.numOfTestSeqs
    % Generate Bluetooth LE test waveforms
    wgParams.PayloadType = testParams.testSeqIds(wfmIdx);
    testWfm = bluetoothTestWaveform(wgParams);
    wfmFreqOffset = pfo(testWfm);
    wfmChannel = thNoise(wfmFreqOffset);
    % Perform filtering
    filtTestWfm(:,wfmIdx) =  conv(wfmChannel,filtDesign.Coefficients.','same');
end
```

Perform frequency modulation by using the helperBLEModulationTestMeasurements.m helper function. Based on the test, the helperBLEModulationTestMeasurements.m helper function computes either frequency deviation, or frequency drift and initial frequency offset.

```
[waveformDiffFreq,fOut1,fOut2,fOut3] = helperBLEModulationTestMeasurements(filtTestWfm,...
    testConfig.TestID,testParams);
```

The helperBLEModulationTestVerdict.m helper function verifies whether the measurements are within the specified limits, and displays the verdict on the command window.

```
helperBLEModulationTestVerdict(waveformDiffFreq,testConfig.TestID,testParams,fOut1,...
    fOut2,fOut3)
```




```
Test sequence: 11110000
    Measured average frequency deviation = 250 kHz
    Expected average frequency deviation = 247.5 kHz to 252.5 kHz
    Result: Pass
Test sequence: 10101010
    Expected 99.9% of all maximum frequency deviation > 185000 kHz
    Result: Pass
Ratio of frequency deviations between two test sequences = 1.163
Expected Ratio > 0.8
    Result: Pass
```

This example demonstrated the Bluetooth LE transmitter test measurements specific to modulation characteristics, carrier frequency offset and, drift. The simulation results verify that these computed test measurement values are within the limits specified by Bluetooth RF-PHY Test Specifications [1 on page 4-0].

Appendix

The helpers used in this example are:

- helperBLEModulationTestConfig.m: Configure Bluetooth LE transmitter test parameters
- helperBLEModulationTestMeasurements.m: Measure frequency deviation, carrier frequency offset and drift
- helperBLEModulationTestVerdict.m: Validate test measurement values and displays the result
- helperModulationCharacteristicsTest.m: Perform modulation characteristics test
- helperModulationTestFilterDesign.m: Design channel filter

Selected Bibliography

1 Bluetooth Special Interest Group (SIG). “Bluetooth RF-PHY Test Specification”, Revision: RF-PHY.TS.5.1.0, Section 4.4. 2018. https://www.bluetooth.com.

2 Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". Version 5.3. https://www.bluetooth.com.

See Also

Functions

`bluetoothTestWaveform`

Objects

`bluetoothTestWaveformConfig` | `bluetoothRFPHYTestConfig`

More About

- “Bluetooth LE IQ samples Coherency and Dynamic Range Tests” on page 4-53
- “Bluetooth LE Output Power and In-Band Emissions Tests” on page 4-17
- “Bluetooth LE Blocking, Intermodulation and Carrier-to-Interference Performance Tests” on page 4-26

Bluetooth EDR RF-PHY Transmitter Tests for Modulation Accuracy and Carrier Frequency Stability

This example shows you how to perform Bluetooth® enhanced data rate (EDR) radio frequency (RF) physical layer (PHY) transmitter tests specific to modulation accuracy and carrier frequency stability using the Bluetooth® Toolbox. The test measurements compute the initial frequency offset, root mean square (RMS) differential error vector magnitude (DEVM), and peak DEVM values. This example also verifies whether these test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [1 on page 4-0].

Objectives of Bluetooth RF-PHY Tests

The Bluetooth RF-PHY Test Specifications [1 on page 4-0] defined by the Bluetooth Special Interest Group (SIG) includes RF-PHY tests for the transmitter and receiver. The objectives of these RF-PHY tests are to:

- Ensure interoperability between all of the Bluetooth devices.
- Ensure a basic level of system performance for all of the Bluetooth products.

Each test case has a specific test procedure and an expected outcome, that the implementation under test (IUT) must achieve.

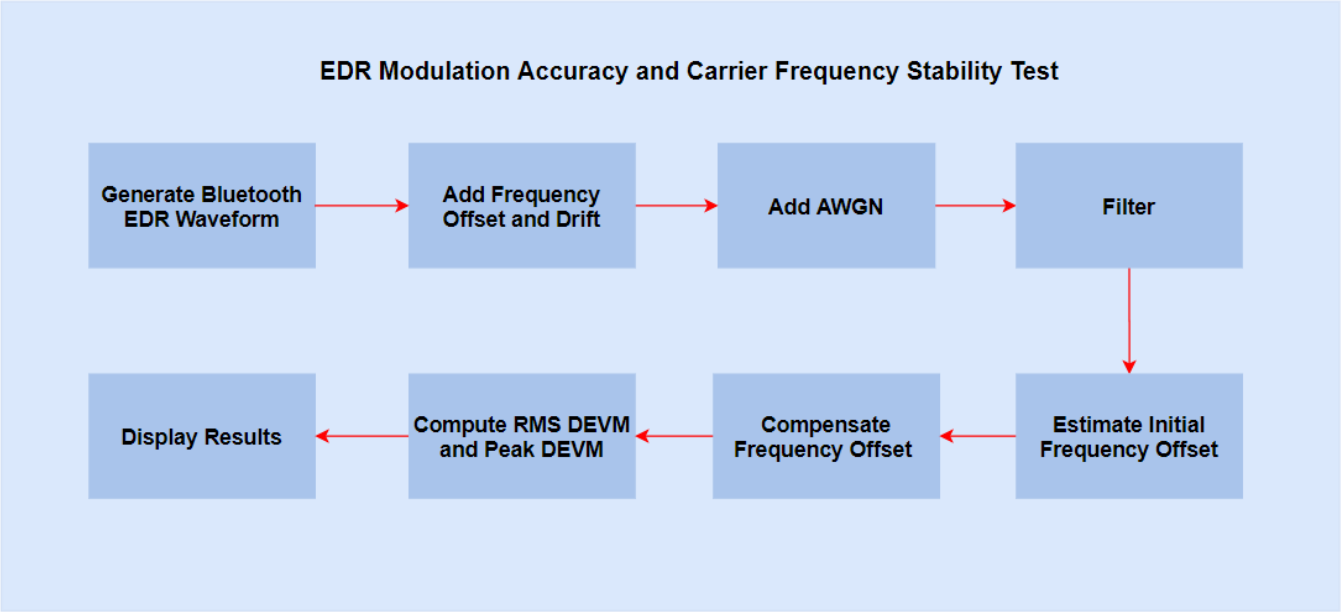
RF-PHY Transmitter Tests

The main goal of the transmitter test measurements is to ensure that the transmitter characteristics are within the limits specified by the Bluetooth RF-PHY Test Specifications [1 on page 4-0]. This example includes transmitter tests relevant to EDR modulation accuracy and carrier frequency stability. This table shows various RF-PHY transmitter tests that this example performs.

Conformance Test	Test Case ID	Test Purpose
Modulation accuracy	RF/TRM/CA/BV-11-C	This test verifies the modulation accuracy of Bluetooth EDR waveforms.
Carrier frequency stability	RF/TRM/CA/BV-11-C	This test verifies the carrier frequency stability of Bluetooth EDR waveforms.

RF-PHY Transmitter Test Procedure

This block diagram summarizes the test procedure for transmitter tests relevant to EDR modulation accuracy and carrier frequency stability of Bluetooth EDR waveforms.



This table shows the packet types and payload lengths required for different PHY modes:

PHY Mode	Packet Type	Maximum Payload Length (Bytes)
$\pi/4-$ DQPSK	2-DH1	31
	2-DH3	356
	2-DH5	656
	2-EV3	58
	2-EV5	358
8-DPSK	3-DH1	11
	3-DH3	88
	3-DH5	986
	3-EV3	88
	3-EV5	538

1. Generate Bluetooth EDR test waveforms by using the `bluetoothTestWaveform` function.
2. Add a carrier frequency offset and drift.
3. Add additive white Gaussian noise (AWGN).
4. Estimate the initial frequency offset using the basic rate (BR) portion of the waveform.
5. Compensate the EDR portion with the estimated initial frequency offset.
6. Perform square root raised cosine filtering using the filter whose coefficients are generated based on the Bluetooth RF-PHY Test Specifications [1 on page 4-0].

7 Divide the EDR portion into blocks of length 50 microseconds each.

8 For each block, delay the compensated sequence by 1 microsecond and differentiate the delay with actual compensated sequence to get the error sequence.

9 Compute the RMS DEVM and peak DEVM based on the error sequence and compensated sequence.

10 Get the test verdict and display the results.

Configure Simulation Parameters

To specify PHY transmission mode, packet type, initial frequency offset, maximum frequency drift, and samples per symbol, set the `phyMode`, `packetType`, `initialFreqOffset`, `maxFreqDrift`, and `sps` respectively.

```
phyMode = EDR2M;                    % PHY transmission mode
packetType = 2-DH1;                 % EDR packet type
initFreqOffset = 40000 ;            % Initial frequency offset in Hz
maxFreqDrift = 0 ;                  % Maximum frequency drift in Hz, must be in
sps = 8;                            % Samples per symbol
```

Generate Test Parameters

Use the preceding configured parameters to generate the test parameters. To get all of the test parameters, use the helperEDRModulationTestConfig.m helper function. To add frequency offset and thermal noise, create and configure `comm.PhaseFrequencyOffset` and `comm.ThermalNoise` System objects™, respectively.

```
[edrTestParams,waveformConfig,filtCoeff] = helperEDRModulationTestConfig(phyMode,packetType,sps);

% Create frequency offset System object
frequencyDelay = comm.PhaseFrequencyOffset('SampleRate',edrTestParams.sampleRate);

% Create thermal noise System object
NF = 12; % Noise figure in dB
thNoise = comm.ThermalNoise('NoiseMethod','Noise figure', ...
                            'SampleRate',edrTestParams.sampleRate, ...
                            'NoiseFigure',NF);
```

Simulate Transmitter Tests

Using the preceding RF-PHY transmitter test procedure, simulate the transmitter tests.

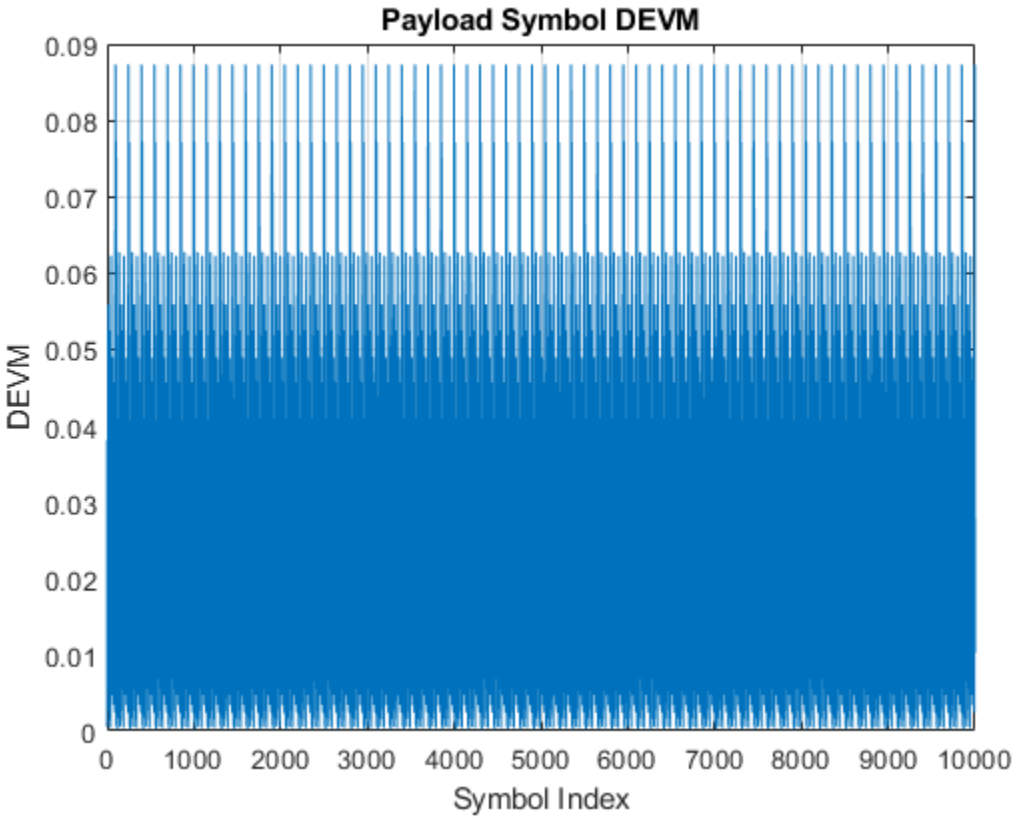
```
% Initialize variables
symDEVM = zeros(1,edrTestParams.requiredBlocks*edrTestParams.blockLength);
[blockRMSDEVM,estimatedBlockFreqDrifts] = deal(zeros(1,edrTestParams.requiredBlocks));
estimatedInitFreqOff = zeros(1,edrTestParams.NumPackets);
blockCount = 0;

% Generate 200 blocks of data as specified in Bluetooth RF-PHY Test Specifications
for packetCount = 1:edrTestParams.NumPackets

    % Generate Bluetooth EDR test waveform
    txWaveform = bluetoothTestWaveform(waveformConfig);
```

```
    % Generate ideal EDR symbols from waveform
    idealTxEDRWaveform = txWaveform(edrTestParams.startIndex*sps+1:end);

    % Perform matched filtering
    rxFilt = upfirdn(idealTxEDRWaveform,filtCoeff,1,sps);

    % Remove delay and normalize filtered signal
    idealEDRSymbols = rxFilt(edrTestParams.span+1:end,1)/sqrt(sps);

    % Add frequency offset
    driftRate = maxFreqDrift/length(txWaveform); % Drift rate
    freqDrift = driftRate*(0:1:(length(txWaveform)-1))'; % Frequency drift for the packet
    frequencyDelay.FrequencyOffset = freqDrift + initFreqOffset; % Frequency offset, includes ini
    transWaveformCFO = frequencyDelay(txWaveform(1:length(txWaveform)));

    % Add thermal noise
    noisyWaveform = thNoise(transWaveformCFO);

    % Compute initial frequency offset specified in Bluetooth RF-PHY Test Specifications
    estimatedInitFreqOff(packetCount) = helperEstimateInitialFreqOffset(noisyWaveform,sps);

    % Compensate initial frequency offset in the received waveform
    pfOffset = comm.PhaseFrequencyOffset('SampleRate',edrTestParams.sampleRate,'FrequencyOffset'
    freqTimeSyncRcv = pfOffset(noisyWaveform);

    % Remove access code, packet header, and guard time from packet
    rxEDRWaveform = freqTimeSyncRcv((edrTestParams.startIndex)*sps+1:end);

    % Perform matched filtering
    rxFilt = upfirdn(rxEDRWaveform,filtCoeff,1,sps);
    receivedEDRSymbols = rxFilt(edrTestParams.span+1:end,1)/sqrt(sps);

    % Compute DEVM values
    [rmsDEVM,rmsDEVMSymbol,samplingFreq] = ...
        helperEDRModulationTestMeasurements(receivedEDRSymbols,idealEDRSymbols,edrTestParams);

    % Accumulate measured values for 200 blocks as specified in Bluetooth RF-PHY Test Specificati
    blockCount = blockCount + edrTestParams.numDEVMBlocks;
    symDEVM(((packetCount-1)*edrTestParams.numDEVMBlocks*edrTestParams.blockLength)+1:(packetCou
        *edrTestParams.blockLength) = rmsDEVMSymbol(1:edrTestParams.numDEVMBlocks*edrTestParams.
    blockRMSDEVM(((packetCount-1)*edrTestParams.numDEVMBlocks)+1:((packetCount)*edrTestParams.nu
        rmsDEVM(1:edrTestParams.numDEVMBlocks);
    estimatedBlockFreqDrifts(((packetCount-1)*edrTestParams.numDEVMBlocks)+1:((packetCount)*edrTe
        samplingFreq(1:edrTestParams.numDEVMBlocks);
end
```

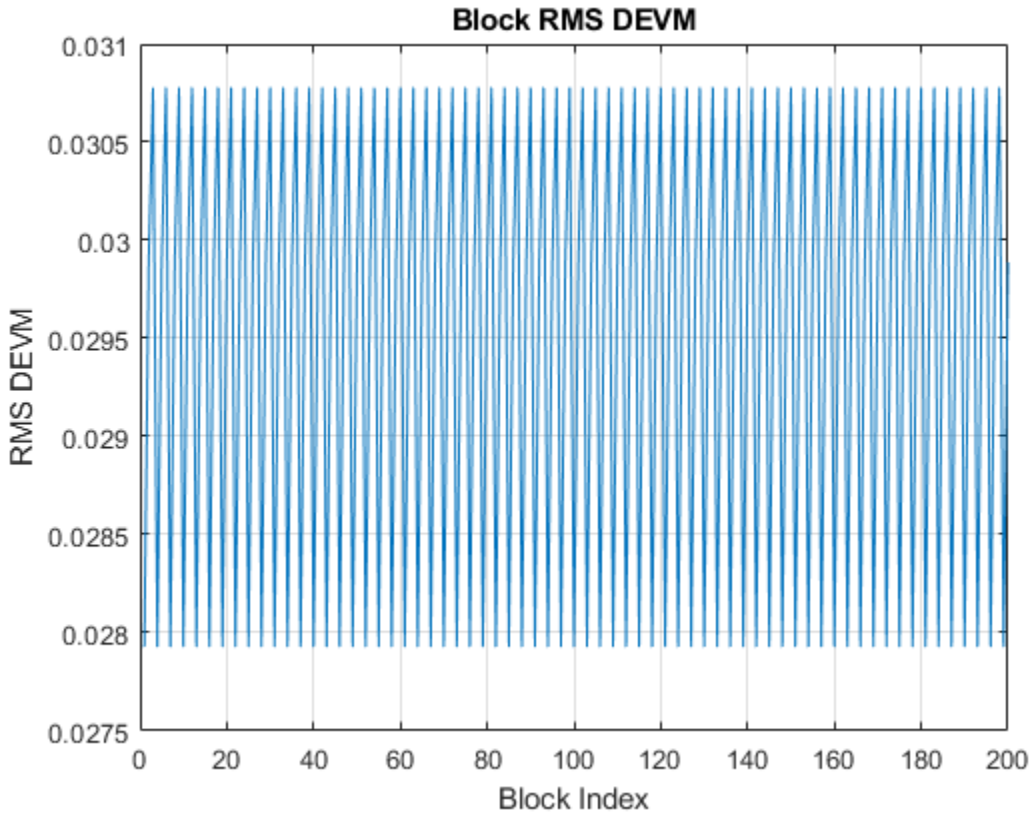
Use the helperEDRModulationTestVerdict.m helper function to verify whether the measurements are within the specified limits and display the verdict.

```
helperEDRModulationTestVerdict(phyMode, ...
    edrTestParams,estimatedInitFreqOff,symDEVM,blockRMSDEVM,estimatedBlockFreqDrifts)
```

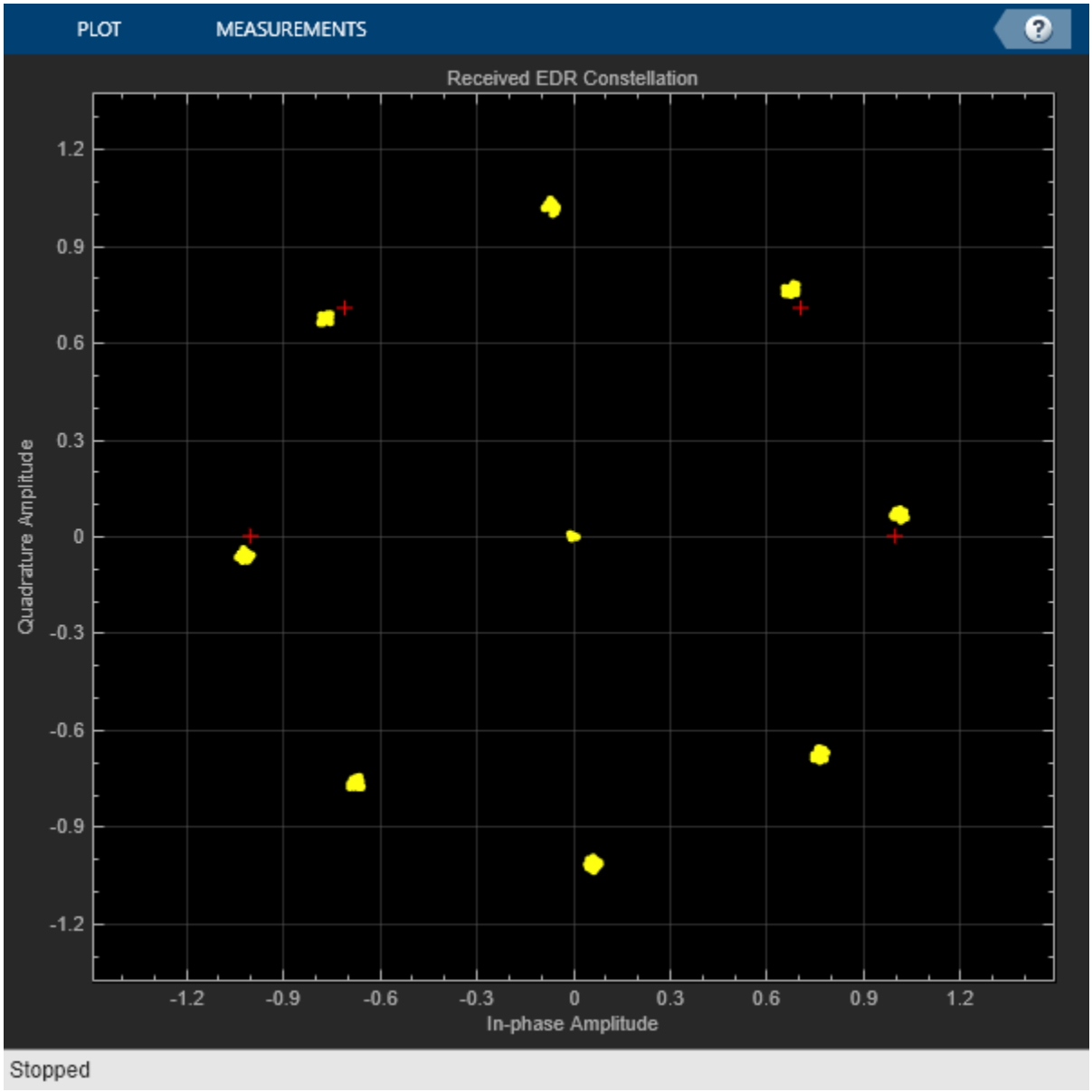
```
Modulation Accuracy Test Results:
```




```
Expected peak DEVM for all pi/4-DQPSK symbols is less than or equal to 0.35
Result: Pass
Percentage of pi/4-DQPSK symbols with DEVM less than or equal to 0.3 is 100
Expected percentage of pi/4-DQPSK symbols with DEVM less than or equal to 0.3 is 99 %
Result: Pass
```




```
       Expected RMS DEVM for all pi/4-DQPSK blocks is less than or equal to 0.2
       Result: Pass
Carrier Frequency Stability Test Results:
       Expected initial frequency offset range: [-75 kHz, 75 kHz]
       Do estimated initial frequency offsets for all the packets fall under expected values?
       Result: Yes
       Expected sampling frequencies range: [-10 kHz, 10 kHz]
       Do estimated sampling frequencies for all the blocks fall under expected values?
       Result: Yes

% Plot the constellation diagram
if strcmp(phyMode,'EDR2M')
    refSymbols = dpskmod(0:edrTestParams.M-1,edrTestParams.M,pi/4,'gray'); % Perform pi/4-DQPSK m
else
    refSymbols = dpskmod(0:edrTestParams.M-1,edrTestParams.M,0,'gray'); % Perform 8-DPSK modulati
end
constDiag = comm.ConstellationDiagram('ReferenceConstellation',refSymbols, ...
    'Title','Received EDR Constellation');
constDiag(receivedEDRSymbols);
release(constDiag);
```



This example demonstrates the Bluetooth EDR transmitter test measurements specific to modulation accuracy and carrier frequency stability. The simulation results verify that these computed test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [1 on page 4-0].

Appendix

The example uses these helpers:

- helperEDRModulationTestConfig.m: Configure Bluetooth test parameters
- helperEstimateInitialFreqOffset.m: Estimate initial frequency offset
- helperEDRModulationTestMeasurements.m: Compute all DEVM measurements required for test
- helperEDRModulationTestVerdict.m: Validate test measurements and display result

Selected Bibliography

1 - Bluetooth Special Interest Group (SIG). “Bluetooth RF-PHY Test Specification”, v1.2/2.0/2.0, EDR/2.1/2.1, EDR/3.0/3.0, HS (), RF.TS/3.0.H.1, Section 4.5. 2009. https://www.bluetooth.com

2 - Bluetooth Special Interest Group (SIG). "Core System Package [BR/EDR Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume 2. https://www.bluetooth.com

See Also

Functions

`bluetoothTestWaveform`

Objects

`bluetoothTestWaveformConfig`

More About

- “Bluetooth BR RF-PHY Transmitter Tests for Modulation Characteristics, Carrier Frequency Offset, and Drift” on page 4-48
- “Bluetooth BR/EDR Power and Spectrum Tests” on page 4-60

Bluetooth BR RF-PHY Transmitter Tests for Modulation Characteristics, Carrier Frequency Offset, and Drift

This example shows you how to perform Bluetooth® basic rate (BR) radio frequency (RF) physical layer (PHY) transmitter tests specific to modulation characteristics, carrier frequency offset, and drift using the Bluetooth® Toolbox. The test measurements compute frequency deviation, carrier frequency offset, and drift values. This example also verifies whether these test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [1 on page 4-0].

Objectives of Bluetooth RF-PHY Tests

The Bluetooth RF-PHY Test Specifications [1 on page 4-0] defined by the Bluetooth Special Interest Group (SIG) include RF-PHY tests for transmitters and receivers. The objectives of these RF-PHY tests are to:

- Ensure interoperability between all Bluetooth devices.
- Ensure a basic level of system performance for all Bluetooth products.

Each test case has a specified test procedure. The expected outcome must be met by the implementation under test (IUT).

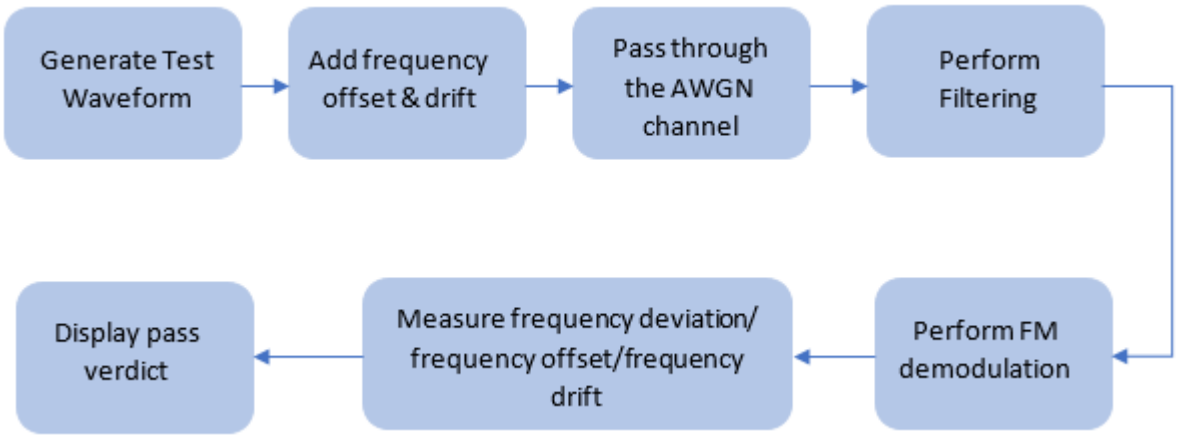
RF-PHY Transmitter Tests

The main goal of the transmitter test measurements is to ensure that the transmitter characteristics are within the limits specified by the Bluetooth RF-PHY Test Specifications [1 on page 4-0]. This example includes transmitter tests relevant to modulation characteristics, carrier frequency offset, and drift. This table shows various RF-PHY transmitter tests performed in this example.

Conformance Test	Test Case ID	Test Purpose
Modulation characteristics	RF/TRM/CA/BV-07-C	Verification of the modulation index
Initial carrier frequency tolerance	RF/TRM/CA/BV-08-C	Verification of the transmitter carrier frequency
Carrier frequency drift	RF/TRM/CA/BV-09-C	Verification of the transmitter center frequency drift within a packet

RF-PHY Transmitter Test Procedure

This block diagram summarizes the test procedure for transmitter tests relevant to modulation characteristics, carrier frequency offset, and drift.



This table shows the test waveforms and packet type(s) required for different test IDs:

Test Case ID	Test Waveforms	Packet Type
RF/TRM/CA/BV-07-C	Generate two Bluetooth DM or DH packets with the repetitive sequences of 11110000b and 10101010b in transmission order	'DH1', 'DH3', 'DH5', 'DM1', DM3', 'DM5'
RF/TRM/CA/BV-08-C	Generate one Bluetooth DH1 packet with a repetitive sequence of PRBS9	'DH1'
RF/TRM/CA/BV-09-C	Generate one Bluetooth DH1 packet with a repetitive sequence of 10101010b	'DH1', 'DH3', 'DH5'

Generate Bluetooth BR test waveforms by passing DH or DM packets through `bluetoothTestWaveform` function.

Configure Simulation Parameters

You can change the `txTestID`, `packetType`, `initFreqOffset`, `maxFreqDrift`, and `sps` parameters based on the transmitter test, packet type, initial frequency offset, maximum frequency drift, and samples per symbol, respectively.

```
txTestID = RF/TRM/CA/BV-07-C;
packetType = DH1;                 % Select packet type according to the transmitte
initFreqOffset = 0;               % Initial frequency offset in Hz
maxFreqDrift = 0;                 % Maximum frequency drift in Hz, [-25e3, 25e3] f
                                  % [-40e3, 40e3] for three and five slot packets
sps = 32;                         % Minimum of 4 samples per symbol according to the
```

Generate Test Parameters

The example generates test parameters based on transmitter test, packet type, initial frequency offset, maximum frequency drift and samples per symbol. Create and configure Bluetooth test waveform by using the `bluetoothTestWaveformConfig` object. To design channel filter based on the sample rate, use helperModulationTestFilterDesign.m helper function. To add frequency offset

and thermal noise, create and configure `comm.PhaseFrequencyOffset` and `comm.ThermalNoise` System objects™, respectively.

```
% Create a Bluetooth test waveform configuration object
cfg = bluetoothTestWaveformConfig('Mode','BR','PacketType',packetType,'SamplesPerSymbol',sps);
if strcmp(txTestID,'RF/TRM/CA/BV-07-C')
    payloadType = [1 2];
elseif strcmp(txTestID,'RF/TRM/CA/BV-08-C')
    payloadType = 0;
else
    payloadType = 2;
end

% Design channel filter
filtDesign = helperModulationTestFilterDesign('BR',sps);

% Create a phase frequency offset System object
sampleRate = sps*1e6; % Sample rate in Hz
pfo = comm.PhaseFrequencyOffset('SampleRate',sampleRate);

% Create a thermal noise System object
NF = 12;              % Noise figure (dB)
thNoise = comm.ThermalNoise('NoiseMethod','Noise figure', ...
                            'SampleRate',sampleRate, ...
                            'NoiseFigure',NF);
```

Simulate Transmitter Tests

To simulate the transmitter tests, perform these steps:

1. Generate a Bluetooth BR test waveform.
2. Add frequency offset, which includes initial frequency offset and drift to the waveform.
3. Add noise to the waveform.
4. Perform filtering on the noisy waveform.
5. Perform frequency modulation (FM) demodulation on the filtered waveform.
6. Perform test measurements and display the pass verdict.

```
filtWaveform = cell(1,size(payloadType,2));              % Initialization
for i = 1:size(payloadType,2)
    cfg.PayloadType = payloadType(i);
    txWaveform = bluetoothTestWaveform(cfg);
    driftRate = maxFreqDrift/length(txWaveform);         % Drift rate
    freqDrift = driftRate*(0:1:(length(txWaveform)-1))'; % Frequency drift for the packet
    pfo.FrequencyOffset =  freqDrift + initFreqOffset;   % Frequency offset, includes initial fre
    wfmFreqOffset = pfo(txWaveform);
    wfmChannel = thNoise(wfmFreqOffset);
    filtWaveform{i} = conv(wfmChannel,filtDesign.Coefficients.','same');
end
```

Based on the transmitter test, the helperBRModulationTestMeasurements.m helper function performs FM demodulation and returns these values:

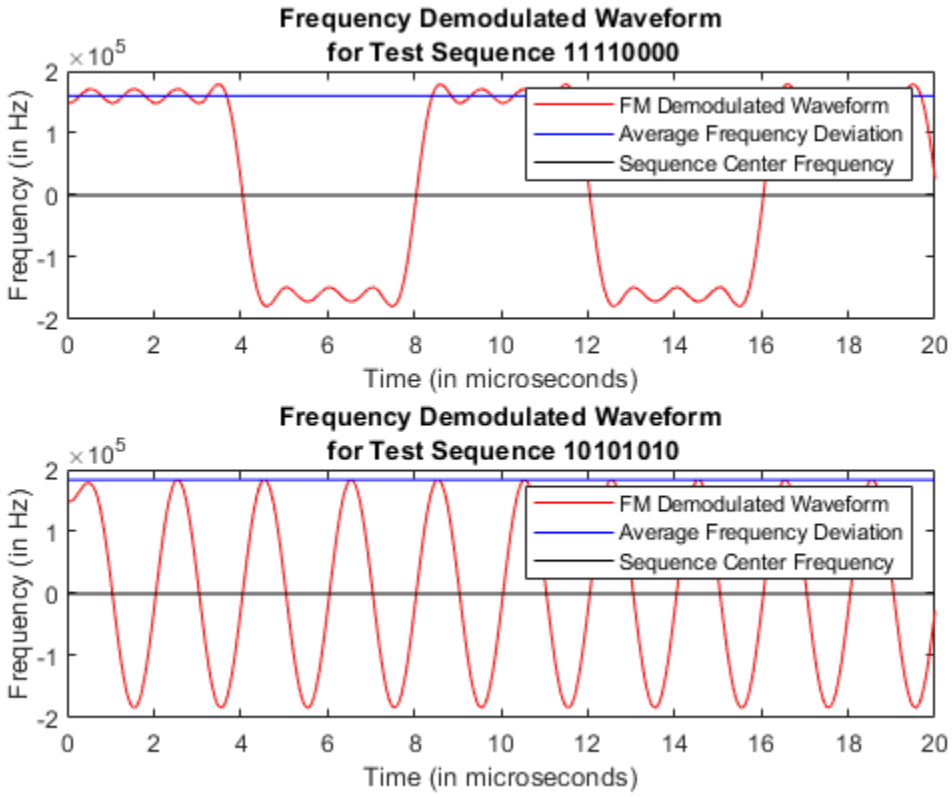
- RF/TRM/CA/BV-07-C: Returns the frequency deviations and center frequencies for the two test sequences, `freq1` and `freq2`, respectively and maximum frequency deviation for the second test sequence, `freq3`.

- RF/TRM/CA/BV-08-C: Returns initial frequency offset, `freq1`.
- RF/TRM/CA/BV-09-C: Returns initial frequency offset, `freq1`, and frequency drift, `freq2`.

```
[waveform,freq1,freq2,freq3] = helperBRModulationTestMeasurements(filtWaveform,txTestID,sps,packe
```

The helperBRModulationTestVerdict.m helper function verifies whether the test measurements are within the specified limits and displays the verdict.

```
helperBRModulationTestVerdict(waveform,txTestID,sps,freq1,freq2,freq3)
```




```
Test sequence: 11110000
    Measured average frequency deviation: 160 kHz
    Expected average frequency deviation range: [140 kHz, 175 kHz]
    Result: Pass
Test sequence: 10101010
    Expected 99.9% of all maximum frequency deviation greater than 115 kHz
    Result: Pass
Ratio of frequency deviations in the two test sequences: 1.1462
Expected Ratio greater than 0.8
    Result: Pass
```

This example demonstrates the Bluetooth BR transmitter test measurements specific to modulation characteristics, carrier frequency offset, and drift. The simulation results verify that the computed test measurement values are within the limits specified by Bluetooth RF-PHY Test Specifications [1 on page 4-0].

Appendix

The example uses these helpers:

- helperModulationTestFilterDesign.m: Design channel filter
- helperBRModulationTestMeasurements.m: Measure frequency deviation, carrier frequency offset and drift
- helperModulationCharacteristicsTest.m: Perform modulation characteristics test
- helperBRModulationTestVerdict.m: Validate test measurement values and displays the result

Selected Bibliography

1 Bluetooth Special Interest Group (SIG). “Bluetooth RF-PHY Test Specification”, v1.2/2.0/2.0, EDR/2.1/2.1, EDR/3.0/3.0, HS (), RF.TS/3.0.H.1, Section 4.5. 2009. https://www.bluetooth.com.

2 Bluetooth Special Interest Group (SIG). "Core System Package [BR/EDR Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume 2. https://www.bluetooth.com.

See Also

Functions
`bluetoothTestWaveform`

Objects
`bluetoothTestWaveformConfig`

More About

- “Bluetooth BR/EDR Power and Spectrum Tests” on page 4-60
- “Bluetooth EDR RF-PHY Transmitter Tests for Modulation Accuracy and Carrier Frequency Stability” on page 4-39

Bluetooth LE IQ samples Coherency and Dynamic Range Tests

This example shows you how to perform Bluetooth® low energy (LE) radio frequency (RF) physical layer (PHY) receiver tests specific to in-phase quadrature samples coherency (IQC) and IQ samples dynamic range (IQDR) by using Bluetooth® Toolbox. The tests compute relative phase, reference phase deviation, and amplitudes of IQ samples at each antenna in an antenna array. This example also verifies whether these test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specification [3 on page 4-0].

Objectives of Bluetooth RF-PHY Tests

The Bluetooth RF-PHY Test Specification [3 on page 4-0] defined by the Bluetooth Special Interest Group (SIG) includes RF-PHY tests for transmitter and receiver. The objectives of these RF-PHY tests are to:

- Ensure interoperability between all Bluetooth devices
- Ensure a basic level of system performance for all Bluetooth products

Each test case has a specific test procedure and an expected outcome, which must be met by the implementation under test (IUT).

IQC and IQDR Tests

The Bluetooth Core Specification 5.1 [2 on page 4-0] introduced angle of arrival (AoA) and angle of departure (AoD) direction finding features. For more information about direction finding services in Bluetooth LE, see “Bluetooth LE Positioning by Using Direction Finding” on page 3-2 and “Bluetooth Location and Direction Finding” on page 8-18. The Bluetooth RF-PHY Test Specification [3 on page 4-0] specifies the tests for direction finding transmitted waveforms with constant tone extension (CTE). This example includes AoA and AoD receiver tests specific to IQC and IQDR.

- **IQ sample coherency**: This test verifies the relative phase and reference phase deviation values derived from the I and Q values sampled on AoA or AoD receiver.
- **IQ sample dynamic range**: This test verifies the I and Q values sampled on AoA or AoD receiver by varying the dynamic range of the CTE.

This table shows various RF-PHY AoA and AoD receiver tests performed in this example.

Conformance Test	Test Case ID	PHY Mode	CTE Type	Number of Antenna Elements
IQ Samples Coherency	RF-PHY/RCV/IQC/BV-01-C	LE1M	[0;1] (2 µs)	4
	RF-PHY/RCV/IQC/BV-02-C	LE1M	[1;0] (1 µs)	
	RF-PHY/RCV/IQC/BV-03-C	LE2M	[0;1] (2 µs)	
	RF-PHY/RCV/IQC/BV-04-C	LE2M	[1;0] (1 µs)	
	RF-PHY/RCV/IQC/BV-05-C	LE1M	[0;0] (2 µs)	2 to 4
	RF-PHY/RCV/IQC/BV-06-C	LE2M	[0;0] (2 µs)	
IQ Samples Dynamic Range	RF-PHY/RCV/IQDR/BV-07-C	LE1M	[0;1] (2 µs)	4
	RF-PHY/RCV/IQDR/BV-08-C	LE1M	[1;0] (1 µs)	
	RF-PHY/RCV/IQDR/BV-09-C	LE2M	[0;1] (2 µs)	
	RF-PHY/RCV/IQDR/BV-10-C	LE2M	[1;0] (1 µs)	
	RF-PHY/RCV/IQDR/BV-11-C	LE1M	[0;0] (2 µs)	2 to 4
	RF-PHY/RCV/IQDR/BV-12-C	LE2M	[0;0] (2 µs)	

Configure Simulation Parameters

Specify the receiver test ID, array size, samples per symbol, and element spacing between the antenna elements.

```
rxTestID = RF-PHY/RCV/IQC/...;  % Receiver test case ID
arraySize = 2; % Array size, must be 4 or [2 2] for AoD receiver tests and 2
               % for AoA receiver tests
sps = 8;                        % Samples per symbol
elementSpacing = 0.5;           % Normalized spacing between the antenna elements with respect
```

Generate RF-PHY Test Parameters

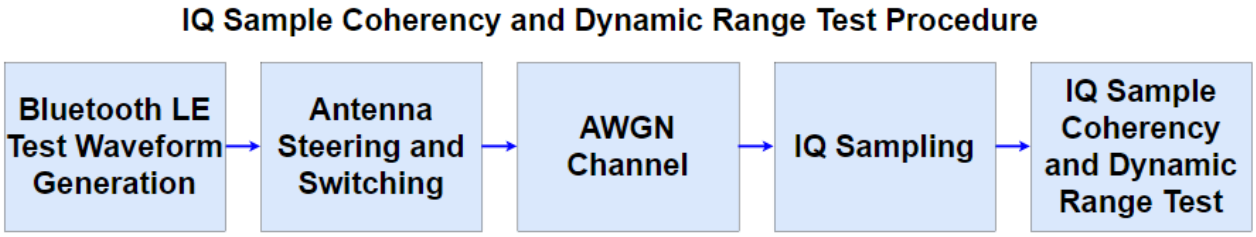
Generate test parameters based on the receiver test ID, array size, and samples per symbol. To generate the PHY mode, CTE type, slot duration, test switching pattern, number of packets to transmit and input power to the receiver, use the helperBLEIQCIQDRTestConfig function. Create and configure `comm.ThermalNoise` System object™ to add thermal noise.

```
[phyMode,cteType,slotDuration,switchingPattern,numPackets,rxPower] = ...
    helperBLEIQCIQDRTestConfig(rxTestID,arraySize,sps);
```

```
% The CTEInfo field position is same for the LE test packet and data
% packet, so consider dfPacketType as ConnectionCTE for CTE based RF-PHY
% tests
dfPacketType = 'ConnectionCTE';

% Create and configure Bluetooth LE angle estimation configuration object
cfg = bleAngleEstimateConfig('ArraySize',arraySize,'SlotDuration',slotDuration, ...
                    'SwitchingPattern',switchingPattern,'ElementSpacing',elementSpacing);
numElements = getNumElements(cfg); % Number of elements in the array

% Create a thermal noise System object
NF = 12; % Noise figure (dB)
symRate = 1e6 + 1e6*(strcmp(phyMode,'LE2M')); % Symbol rate in Hz based on PHY transmission mode
sampleRate = symRate*sps; % Sampling rate in Hz
thNoise = comm.ThermalNoise('NoiseMethod','Noise figure', ...
                            'SampleRate',sampleRate, ...
                            'NoiseFigure',NF);
```

Simulate IQ Coherency or Dynamic Range Tests

To simulate the IQC and IQDR tests, perform these steps.



1. Generate Bluetooth LE test packet waveform.
2. Perform waveform steering and antenna switching.
3. Add thermal noise.
4. Perform demodulation, decoding, and IQ sampling on the noisy waveform.
5. Perform IQC and IQDR test measurements.

Based on the receiver test, the helperBLEIQCIQDRTest function returns these values.

Tests	Test Results
• RF-PHY/RCV/IQC/BV-01-C • RF-PHY/RCV/IQC/BV-02-C • RF-PHY/RCV/IQC/BV-03-C • RF-PHY/RCV/IQC/BV-04-C • RF-PHY/RCV/IQC/BV-05-C • RF-PHY/RCV/IQC/BV-06-C	Relative phase (RP), mean relative phase (MRP), summation in the formula for mean relative phase, reference phase deviation (RPD), mean reference phase deviation (MRPD), summation in the formula for mean reference phase deviation
• RF-PHY/RCV/IQDR/BV-07-C • RF-PHY/RCV/IQDR/BV-08-C • RF-PHY/RCV/IQDR/BV-09-C • RF-PHY/RCV/IQDR/BV-10-C • RF-PHY/RCV/IQDR/BV-11-C • RF-PHY/RCV/IQDR/BV-12-C	Amplitudes of IQ samples for each antenna

```
% Initialize the number of outputs based on the receiver test ID
numOutputs = 2 + 4*any(strcmp(rxTestID,{'RF-PHY/RCV/IQC/BV-01-C','RF-PHY/RCV/IQC/BV-02-C', ...
                'RF-PHY/RCV/IQC/BV-03-C','RF-PHY/RCV/IQC/BV-04-C','RF-PHY/RCV/IQC/BV-05-C', ...
                'RF-PHY/RCV/IQC/BV-06-C'}));
[iqcIQDROutputs,iqcIQDROutputsConc] = deal(cell(1,numOutputs));

% Generate Bluetooth LE test waveform
bleTestWaveformConfig = bluetoothTestWaveformConfig;
bleTestWaveformConfig.Mode = phyMode;
bleTestWaveformConfig.PacketType = dfPacketType;
bleTestWaveformConfig.PayloadType = randsrc(1,1,1:7); % Payload type can be any value as the pay
bleTestWaveformConfig.PayloadLength = 0; % Empty payload for the considered receiver test IDs
bleTestWaveformConfig.CTELength = 20;
bleTestWaveformConfig.SamplesPerSymbol = sps;
bleWaveform = bluetoothTestWaveform(bleTestWaveformConfig);

% Loop over the number of packets
for i = 1:numPackets
    % Generate random angle(s) between -90 to 90 degrees
    angles = randsrc(2,1,-90:90);

    % Perform steering and switching between the antennas
    dfWaveform = helperBLESteerSwitchAntenna(bleWaveform,angles,...
                            phyMode,sps,dfPacketType,bleTestWaveformConfig.PayloadLength,cfg);

    % Attenuate the waveform according to the given received power
    dfWaveformAtt = dfWaveform.*10.^(rxPower/20);

    % Add thermal noise to the waveform
    noisyWaveform = thNoise(dfWaveformAtt);

    % Pass the noisy waveform to the Bluetooth LE ideal receiver and get the IQ
    % samples
    [~,~,iqSamples] = bleIdealReceiver(noisyWaveform,'Mode',phyMode, ...
                'SamplesPerSymbol',sps,'DFPacketType',dfPacketType, ...
                'SlotDuration',slotDuration,'WhitenStatus','Off');
```

```
    % Perform IQC and IQDR test measurements based on the receiver test ID
    [iqcIQDROutputs{:}] = helperBLEIQCIQDRTest(rxTestID,iqSamples,numElements);

    % Concatenate the outputs over the number of packets
    for j = 1:numOutputs
        iqcIQDROutputsConc{j} = [iqcIQDROutputsConc{j}; iqcIQDROutputs{j}];
    end
end
```

Test Verdict

Verify whether the IQC and IQDR test measurements are within the specified limits and display the test verdict.

```
if any(strcmp(rxTestID,{'RF-PHY/RCV/IQC/BV-01-C','RF-PHY/RCV/IQC/BV-02-C', ...
        'RF-PHY/RCV/IQC/BV-03-C','RF-PHY/RCV/IQC/BV-04-C', ...
        'RF-PHY/RCV/IQC/BV-05-C','RF-PHY/RCV/IQC/BV-06-C'})) % IQC test

    % For each nonreference antenna, Am, where m is in the range [0, number
    % of antenna elements-1], used in the switching pattern, the results of
    % the summations in the formulae for MRP(m) and MRPD must be nonzero
    disp ('Expected summations in the formulae for MRP(m) and MRPD must be non-zero.');
    if all(all(iqcIQDROutputsConc{2}~=0)) && all(iqcIQDROutputsConc{5}~=0)
        disp('Result: Pass');
    else
        disp('Result: Fail');
    end

    % For each nonreference antenna, Am, used in the switching pattern, 95%
    % of the values, v, in the set must be -0.52<=principal(v-MRP(m))<=0.52
    mrpRep = kron(iqcIQDROutputsConc{3},ones(length(iqcIQDROutputsConc{1})/length(iqcIQDROutputs
    subMRP = iqcIQDROutputsConc{1} - mrpRep;
    if size(subMRP,2) == 3 && any(strcmp(rxTestID,{'RF-PHY/RCV/IQC/BV-01-C',...
        'RF-PHY/RCV/IQC/BV-03-C','RF-PHY/RCV/IQC/BV-05-C','RF-PHY/RCV/IQC/BV-06-C'}))
        subMRP(3:3:end,3) = 0;
    end
    subMRPPrincipal = helperBLEPrincipalAngle(subMRP);
    subMRPRange = sum(subMRPPrincipal<=0.52 & subMRPPrincipal>=-0.52);
    disp ('Expected 95% of the values v in the set RP(m) must meet -0.52<=principal(v-MRP(m))<=0
    if all(subMRPRange>0.95*length(subMRPPrincipal))
        disp('Result: Pass');
    else
        disp('Result: Fail');
    end

    % MRPD must be in the range -1.125 to 1.125
    disp ('Expected MRPD in the range [-1.125, 1.125] radians.');
    if all(iqcIQDROutputsConc{6}<=1.125) && all(iqcIQDROutputsConc{6}>=-1.125)
        disp('Result: Pass');
    else
        disp('Result: Fail');
    end
else  % IQDR test

    % The mean of amplitudes of IQ samples measured at each antenna follows
    % the equation mean(ANT3)<mean(ANT2)<mean(ANT0)<mean(ANT1)
    meanA1 = mean(iqcIQDROutputsConc{1});
```

```
    meanA = mean(iqcIQDROutputsConc{2});
    if length(meanA) == 1
        disp('The mean of amplitudes must follow mean(ANT0)<mean(ANT1).');
        conditionCheck = meanA1<meanA(1);
    elseif length(meanA) == 2
        disp('The mean of amplitudes must follow mean(ANT2)<mean(ANT0)<mean(ANT1).');
        conditionCheck = meanA1<meanA(1) && meanA1>meanA(2);
    else
        disp('The mean of amplitudes must follow mean(ANT3)<mean(ANT2)<mean(ANT0)<mean(ANT1).');
        conditionCheck = meanA1<meanA(1) && meanA1>meanA(2) && meanA(3)<meanA(2);
    end
    if conditionCheck
       disp('Result: Pass');
    else
        disp('Result: Fail');
    end
end
```

```
Expected summations in the formulae for MRP(m) and MRPD must be non-zero.

Result: Pass

Expected 95% of the values v in the set RP(m) must meet -0.52<=principal(v-MRP(m))<=0.52.

Result: Pass

Expected MRPD in the range [-1.125, 1.125] radians.

Result: Pass
```

This example demonstrates the Bluetooth LE receiver test measurements specific to IQC and IQDR test measurements. The simulation results verify that the computed test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [3 on page 4-0].

Appendix

The example uses these helpers:

- helperBLEIQCIQDRTestConfig: Configure test parameters specific to IQC and IQDR test measurements
- helperBLESteerSwitchAntenna: Perform antenna steering and switching
- helperBLESwitchAntenna: Perform antenna switching
- helperBLEPrincipalAngle: Compute principal angle in radians
- helperBLEIQCIQDRTest: Perform IQC and IQDR test measurements

Selected Bibliography

1 Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com.

2 Bluetooth Special Interest Group (SIG). "Core System Package [Low Energy Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume 6, https://www.bluetooth.com.

3 Bluetooth Special Interest Group (SIG). “Bluetooth RF-PHY Test Specification”, RF-PHY.TS.5.1.0, Section 4.5. https://www.bluetooth.com.

See Also

Functions
`bluetoothTestWaveform`

Objects
`bluetoothTestWaveformConfig`

More About

- “Bluetooth LE Output Power and In-Band Emissions Tests” on page 4-17
- “Bluetooth LE Modulation Characteristics, Carrier Frequency Offset and Drift Tests” on page 4-33
- “Bluetooth LE Blocking, Intermodulation and Carrier-to-Interference Performance Tests” on page 4-26

Bluetooth BR/EDR Power and Spectrum Tests

This example shows how to perform radio frequency (RF) physical layer (PHY) transmitter tests specific to power and spectrum on Bluetooth® basic rate (BR) and enhanced data rate (EDR) transmitted waveforms by using Bluetooth® Toolbox features. The example also verifies whether these test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [1 on page 4-0].

Objectives of Bluetooth RF-PHY tests

The Bluetooth RF-PHY Test Specifications [1 on page 4-0] defined by the Bluetooth Special Interest Group (SIG) includes RF-PHY tests for transmitters and receivers. The objectives of these RF-PHY tests are to:

- Ensure interoperability between all Bluetooth devices.
- Verify that a basic level of system performance is guaranteed for all Bluetooth products.

Each test case has a specific test procedure and an expected outcome, which must be met by the implementation under test (IUT).

Power and Spectrum Tests

This example shows how to perform power and spectrum test measurements on Bluetooth BR/EDR transmitted waveforms according to the Bluetooth RF-PHY Test Specifications [1 on page 4-0].

- **Power Tests:** These tests verify whether the peak power, average power, power density, and power control of the transmitted Bluetooth signals are within the limits specified in the Bluetooth RF-PHY Test Specifications [1 on page 4-0]. For more information about these tests, see sections 4.5.1, 4.5.2, 4.5.3, 4.5.10, and 4.5.14 of the Bluetooth RF-PHY Test Specifications.
- **Spectrum Tests:** These tests verify whether the signal emissions are within the operating frequency range specified in the Bluetooth RF-PHY Test Specifications. For more information about these tests, see sections 4.5.4, 4.5.5, 4.5.6, and 4.5.13 of the Bluetooth RF-PHY Test Specifications.

This table shows various RF-PHY transmitter tests this example implements.

Conformance Tests	Test Case ID	Test Purpose	Required Packet Types	Center Frequency (MHz)
Power Tests	RF/TRM/CA/BV-01-C	Verifies the maximum peak and average RF-output power	'DH1', 'DH3', 'DH5', 'DM1', 'DM3', 'DM5'	2402, 2441, 2480
	RF/TRM/CA/BV-02-C	Verifies the maximum RF-power density	'DH1', 'DH3', 'DH5', 'DM1', 'DM3', 'DM5'	2441
	RF/TRM/CA/BV-03-C	Verifies the transmitter power control	'DH1'	2402, 2441, 2480
	RF/TRM/CA/BV-10-C	Verifies whether the difference in average transmit power during Gaussian frequency shift keying (GFSK) modulated and differential phase shift keying (DPSK) modulated portions of a packet is within an acceptable range	For pi/4-DQPSK, '2-DHx', '2-EVx' For 8DPSK, '3-DHx', '3-EVx' Where x = 1,3,5	2402, 2441, 2480
	RF/TRM/CA/BV-14-C	Verifies the transmitter power control	'2-DH1', '3-DH1'	2402, 2441, 2480
Spectrum Tests	RF/TRM/CA/BV-04-C	Verifies whether the emissions are within the limits of the operating frequency range	'DH1', 'DH3', 'DH5', 'DM1', 'DM3', 'DM5'	2402, 2480
	RF/TRM/CA/BV-05-C	Verifies whether the emissions for 20 dB bandwidth are within the limits	'DH1', 'DH3', 'DH5', 'DM1', 'DM3', 'DM5'	2402, 2441, 2480
	RF/TRM/CA/BV-06-C	Verifies whether the adjacent channel power emissions are within the limits specified	'DH-1'	2402, 2441, 2480
	RF/TRM/CA/BV-13-C	Verifies whether the level of unwanted signals from the DPSK transmitter are within the range	'2-DH1', '2-DH3', '2-DH5', '2-EV3', '2-EV5'	2402, 2441, 2480

Configure Simulation Parameters

Specify the test case ID, center frequency, packet type, samples per symbol, and output power.

```
% Select test case ID to perform power and spectrum tests
testCaseID = RF/TRM/CA/BV-01-C ;
```

```
% Select the frequency of operation for the IUT as shown in this table
%  ---------------------------
%  | Operating | Frequency in |
%  | Frequency |    MHz       |
%  ---------------------------
%  | Low       |    2402      |
%  | Mid       |    2440      |
%  | High      |    2480      |
%  ---------------------------
% Specify the type of center frequency required to perform the test case
%  ------------------------------------------------------
%  |    Test Case ID    |   Type of Center Frequency    |
%  ------------------------------------------------------
%  |                    |                               |
%  | RF/TRM/CA/BV-01-C  |      'Low','Mid','High'       |
%  |                    |                               |
%  |                    |                               |
%  | RF/TRM/CA/BV-02-C  |            'Mid'              |
%  |                    |                               |
%  |                    |                               |
%  | RF/TRM/CA/BV-03-C  |      'Low','Mid','High'       |
%  |                    |                               |
%  |                    |                               |
%  | RF/TRM/CA/BV-04-C  |         'Low','High'          |
%  |                    |                               |
%  |                    |                               |
%  | RF/TRM/CA/BV-05-C  |      'Low','Mid','High'       |
%  |                    |                               |
%  |                    |                               |
%  | RF/TRM/CA/BV-06-C  |      'Low','Mid','High'       |
%  |                    |                               |
%  |                    |                               |
%  | RF/TRM/CA/BV-10-C  |      'Low','Mid','High'       |
%  |                    |                               |
%  |                    |                               |
%  | RF/TRM/CA/BV-13-C  |      'Low','Mid','High'       |
%  |                    |                               |
%  |                    |                               |
%  | RF/TRM/CA/BV-14-C  |      'Low','Mid','High'       |
%  |                    |                               |
%  |                    |                               |
%  ------------------------------------------------------
centerFrequency = Low;

% Specify the type of packet required to perform the test case
packetType = 'DH1';
sps = 8;                                    % Number of samples per symbol

outputPower = 20;                           % Output power in dBm (must be in the range [-20,20])

stepSize = 2;                               % Step size in dB (for power control tests
                                            % {'RF/TRM/CA/BV-03-C',
                                            % 'RF/TRM/CA/BV-14-C'})
```

Configure Test Parameters and Generate Bluetooth Test Waveform

Configure the test parameters and generate the Bluetooth test waveform by using the helperBluetoothPowerTestConfig.m helper function.

```
% Get test parameters and test waveform
[configParams,txWaveformBaseBand] = helperBluetoothPowerTestConfig(testCaseID,centerFrequency,pac

% Interpolation factor for upconversion
interpFactor = ceil(2*configParams.StopFreq/configParams.SampleRate);

% Create a digital upconverter System object
upConv = dsp.DigitalUpConverter( ...
    'InterpolationFactor',interpFactor, ...
    'SampleRate',configParams.SampleRate, ...
    'Bandwidth',6e6, ...
    'CenterFrequency',configParams.CenterFreq);

% Upconvert the baseband waveform to passband and scale the waveform to
% required power
dBmConvFactor = 30;
scalingFactor = 10^((outputPower-dBmConvFactor)/20);
txWaveform = scalingFactor*upConv(txWaveformBaseBand);
```

Perform Spectrum Analysis on the Waveform

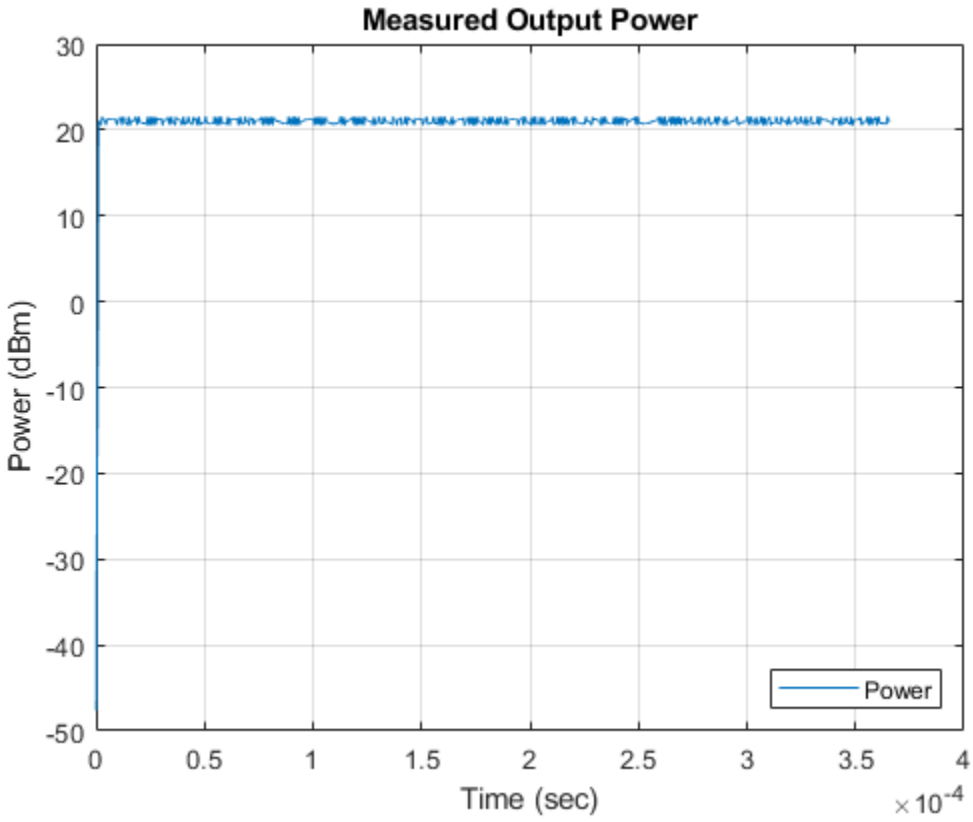
Perform the spectrum analysis by using the helperBluetoothPowerTestAnalysis.m helper function. The function returns the output power along with the frequency, time, or step size data from the spectrum.

```
[outPower,testMeas] = helperBluetoothPowerTestAnalysis(testCaseID,configParams,txWaveform,interpF
```

Calculate Test Measurements

Calculate the test measurements for each test based on the spectrum analysis outputs by using the helperBluetoothPowerTestMeasurements.m helper function.

```
output = helperBluetoothPowerTestMeasurements(testCaseID,centerFrequency,outPower,testMeas);
```



Simulation Results

Validate the test results and display the verdict using the helperBluetoothPowerTestVerdict.m helper function.

```
helperBluetoothPowerTestVerdict(testCaseID,output);
```

```
Measured average power - 21.032359 dBm
Measured peak power - 21.524863 dBm
Verdict - Output power test passed
```

This example demonstrates the Bluetooth BR/EDR RF-PHY transmitter test measurements specific to power and spectrum. The simulation results verify that the computed test measurement values are within the limits specified by the Bluetooth RF-PHY Test Specifications [1 on page 4-0].

Appendix

The example uses these helpers:

- helperBluetoothPowerTestConfig.m: Configure test parameters and generate Bluetooth test waveform
- helperBluetoothPowerTestAnalysis.m: Perform analysis of Bluetooth test waveforms in time and frequency domain
- helperBluetoothPowerTestMeasurements.m: Calculate test measurements
- helperBluetoothPowerTestVerdict.m: Validate test results and displays the verdict

Selected Bibliography

1. Bluetooth Special Interest Group (SIG). “Bluetooth RF-PHY Test Specifications”, v1.2/2.0/2.0, EDR/2.1/2.1, EDR/3.0/3.0, HS (), RF.TS/3.0.H.1, Section 4.5. 2009. https://www.bluetooth.com.
2. Bluetooth Special Interest Group (SIG). "Core System Package [BR/EDR Controller Volume]". Bluetooth Core Specification. Version 5.3, Volume 2. https://www.bluetooth.com.

See Also

Functions

`bluetoothTestWaveform`

Objects

`bluetoothTestWaveformConfig`

More About

- “Bluetooth BR RF-PHY Transmitter Tests for Modulation Characteristics, Carrier Frequency Offset, and Drift” on page 4-48
- “Bluetooth EDR RF-PHY Transmitter Tests for Modulation Accuracy and Carrier Frequency Stability” on page 4-39

End-To-End Simulation

End-to-End Bluetooth LE PHY Simulation Using Path Loss Model, RF Impairments, and AWGN

This example uses Bluetooth® Toolbox to perform end-to-end Bluetooth low energy (LE) simulation for different Bluetooth LE physical layer (PHY) transmission modes in the presence of the path loss model, radio front-end (RF) impairments, and additive white Gaussian noise (AWGN). The simulation results show the estimated value of the bit error rate (BER), path loss, and distance between the transmitter and receiver.

Path Loss Modeling in Bluetooth LE Network

The Bluetooth Core Specifications [1] on page 5-0 defined by the Bluetooth Special Interest Group (SIG) introduced Bluetooth LE to enable low-power short-range communication. Bluetooth LE devices operate in the globally unlicensed industrial, scientific, and medical (ISM) band in a frequency range from 2.4 GHz to 2.485 GHz. Bluetooth LE specifies a channel spacing of 2 MHz, resulting in 40 RF channels. The prominent applications of Bluetooth LE include direction finding services and building intelligent internet of things (IoT) solutions to facilitate home, commercial, and industrial automation. For more information about direction finding services in Bluetooth LE, see the “Bluetooth Location and Direction Finding” on page 8-18.

In past few years, there has been a significant increase in designing Bluetooth LE networks for a plethora of use case scenarios. To achieve high performance and quality in the Bluetooth LE network, studying the propagation of the Bluetooth LE signal along the link between the transmitter and the receiver is recommended. This example shows an end-to-end Bluetooth LE simulation considering these factors that impact the propagation of Bluetooth LE signals along the communication link between the transmitter and receiver.

- Receiver sensitivity
- Environment
- Transmit power
- Antenna gain

Receiver Sensitivity

Receiver sensitivity is the measure of minimum signal strength at which the receiver can detect, demodulate, and decode the waveform. The reference sensitivity level specified in the Bluetooth Core Specifications [1] on page 5-0 is -70 dBm. However, the actual sensitivity level for the receiver as per the Bluetooth Core Specifications [1] on page 5-0 is defined as the receiver input level for which the BER specified in this table is achieved.

Maximum Supported Payload Length (bytes)	BER (%)
1 to 37	0.1
38 to 63	0.064
64 to 127	0.034
128 to 255	0.017

This table shows the actual sensitivity level of the receiver for a given PHY transmission mode.

PHY Transmission Modes	Receiver Sensitivity (dBm)
LE Uncoded PHYs	≤ -70
LE Coded PHY with S = 2 coding	≤ -75
LE Coded with S = 8 coding	≤ -82

Environment

Bluetooth LE networks are operated in different environments such as home, office, industrial, and outdoor. A specific path loss model is used for each environment.

Environment	Path loss model
Outdoor	Two-Ray Ground Reflection
Industrial	Log-Normal Shadowing
Home	NIST PAP02-Task 6
Office	NIST PAP02-Task 6

Path Loss Model

Path loss or path attenuation is the decline in the power density of a given signal as it propagates from the transmitter to receiver through space. This reduction in power density occurs naturally over the distance and is impacted by the obstacles present in the environment in which the signal is being transmitted. The path loss is generally expressed in decibels (dB) and is calculated as:

$$\mathrm{PL}_{\mathrm{dB}} = P_t - P_r.$$

In this equation,

- $\mathrm{PL}_{\mathrm{dB}}$ is the path loss in dB.
- P_t is the transmitted signal power in dB.
- P_r is the received signal power in dB.

Path loss models describe the signal attenuation between the transmitter and receiver based on the propagation distance and other parameters such as frequency, wavelength, path loss exponent, and antenna gains.

Free-Space Path Loss Model

Free-space path loss is the attenuation of signal strength between the transmitter and receiver along the line of sight (LoS) path through free space (usually air), excluding the effect of the obstacles in the path. The free-space path loss is calculated as:

$$\mathrm{PL}_{\mathrm{dB}} = 20\log\left(\frac{4\pi d}{\lambda}\right).$$

In this equation,

- d is the distance between the transmitter and receiver.
- λ is the signal wavelength.

Log-Normal Shadowing Path Loss Model

A log-distance path loss model reflects the path loss that a signal encounters in an indoor environment such as a building. The log-normal shadowing model[3] on page 5-0 is an extension of log-distance path loss model. Unlike the log-distance model, the log-normal shadowing model considers the fact that the surrounding environment clutter can be vastly different at two different locations having the same transmitter-receiver separation. Measurements show that at any transmitter-receiver distance, d, the path loss at a particular location is random and distributed log normally (in dB) about the mean distance dependent value. The path loss is calculated as:

$$\mathrm{PL}_{\mathrm{dB}}(d) = \mathrm{PL}_{\mathrm{dB}}(d_0) + 10\gamma\log\left(\frac{d}{d_0}\right) + X_\sigma.$$

In this equation,

- $\mathrm{PL}_{\mathrm{dB}}(d_0)$ is the path loss at the reference distance d_0.
- d is the distance between the transmitter and receiver.
- d_0 is the reference distance.
- γ is the path loss exponent.

- X_σ is the normal or Gaussian random variable with zero mean, reflecting the attenuation caused by the flat fading.

Two-Ray Ground Reflection Model

The two-ray ground reflection model [3] on page 5-0 is a radio propagation model that estimates the path loss between the transmitter and receiver by considering these two signal components: LoS and the component reflected from the ground. When the transmitter and receiver antenna heights are approximately similar and the distance between the antennas is very large relative to the height of the antennas, then the path loss is calculated as:

$$\mathrm{PL}_{\text{linear scale}} = \frac{G\,h_t^2 h_r^2}{d^4}.$$

The path loss in logarithmic scale is calculated as:

$$\mathrm{PL}_{\mathrm{dB}} = 40\log10(d) - 10\log10\left(G\,h_t^2 h_r^2\right).$$

In this equation,

- d is the distance between the transmitter and receiver.
- G is the product of antenna gains.
- h_t is the height of the transmitter.
- h_r is the height of the receiver.

NIST PAP02-Task 6 Model

The National Institute of Standards and Technology (NIST) conducted studies for indoor to indoor, outdoor to outdoor, and outdoor to indoor propagation paths and derived these equations for calculating the path loss[4] on page 5-0 :

$$\mathrm{PL}_d = \mathrm{PL}_0 + 10(n_0)\log10\left(\frac{d}{d_0}\right). \qquad \text{for } d \le d_1$$

$$\mathrm{PL}_d = \mathrm{PL}_0 + 10(n_0)\log10\left(\frac{d}{d_0}\right) + 10(n_1)\log10\left(\frac{d}{d_1}\right). \qquad \text{for } d > d_1$$

In these equations,

- PL_0 is the path loss at the reference distance d_0.
- n_0,n_1 are the path loss exponents.
- d is the distance between the transmitter and receiver.
- d_0 is the reference distance, assumed to be 1 meter in simulations.
- d_1 is the breakpoint where the path loss exponent adjusts from n_0 to n_1.

The example considers these values for different environments.

Environment	PL_0 (dB)	n_0	d_1 (m)	n_1	σ
Home	12.5	4.2	11	7.6	3
Office	26.8	4.2	10	8.7	3.7

Most of these measurements for the NIST PAP02 Task 6 channel model were taken with transmitters and receivers located in hallways with distances ranging from 5 m to 45 m.

Transmit Power

Transmit power is the power of the radio frequency signal generated by the transmitter. Increasing the transmit power increases the likelihood that the signal can be transmitted over longer distances. Bluetooth supports transmit power from -20 dBm (0.01 mW) to 20 dBm (100 mW).

Antenna Gain

Antenna gain is the factor by which the antenna improves the total radiated power. Bluetooth designers can choose to implement a variety of antenna options. Bluetooth devices typically achieve an antenna gain in the range from -10 dBi to 10 dBi.

End-to-End Bluetooth LE Simulation Procedure

The end-to-end Bluetooth LE PHY simulations estimate the BER and the distance between the transmitter and receiver by considering a specific environment with RF impairments and AWGN added to the transmission packets.

For a given set of simulation parameters, obtain the signal-to-noise ratio (SNR) at the receiver by assuming a fixed noise figure. For the obtained value of SNR including the path loss, generate the Bluetooth LE waveform using `bleWaveformGenerator` function. Distort the generated waveform with RF impairments and AWGN. Each packet is distorted by these RF impairments:

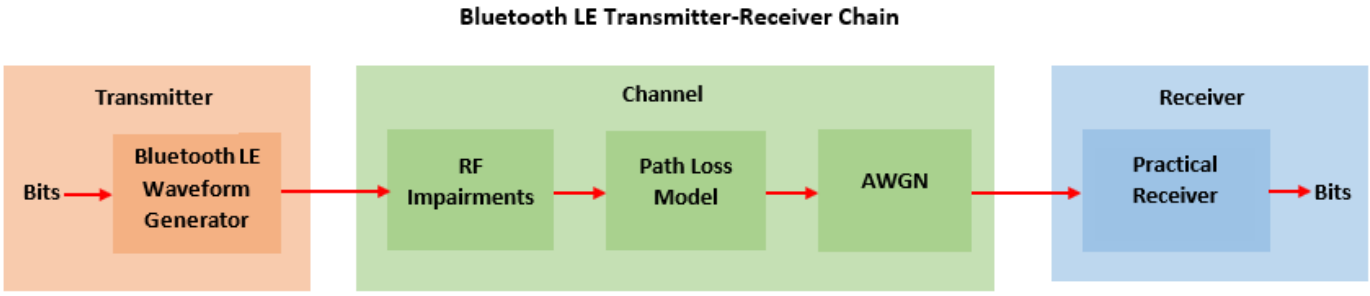
- DC offset
- Carrier frequency offset
- Carrier phase offset
- Timing drift

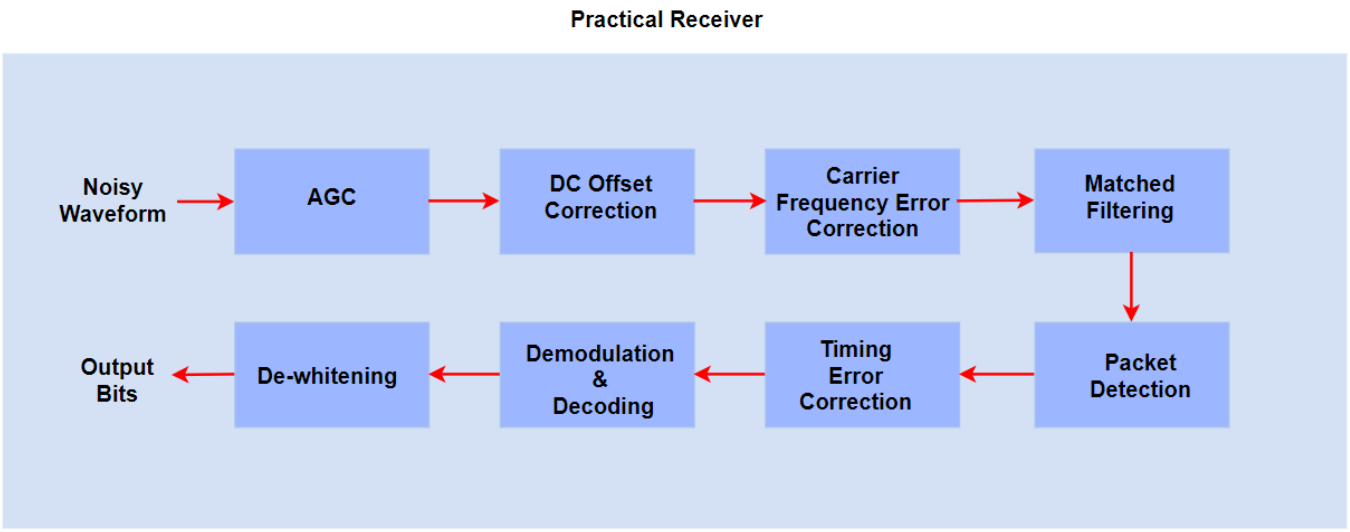
The noisy packets are processed through a practical Bluetooth LE receiver that performs these operations:

1. Automatic gain control (AGC)
2. DC removal
3. Carrier frequency offset correction
4. Matched filtering
5. Packet detection
6. Timing error detection
7. Demodulation and decoding

8 De-whitening

The end-to-end example chain is summarized in these block diagrams





The BER is obtained by comparing the transmitted and recovered data bits.

Configure Simulation Parameters

In this example, the distance between the transmitter and receiver is estimated based on the environment and the power levels of the signal at the transmitter and receiver. Configure the parameters using the `bluetoothRangeConfig` object.

Configure parameters related to the communication link between the transmitter and receiver

```
rangeConfig = bluetoothRangeConfig;
rangeConfig.Environment = Outdoor;                    % Environment
rangeConfig.Mode = LE1M;                              % PHY transmission mode
rangeConfig.ReceiverSensitivity = -73;                % Receiver sensitivity in d
rangeConfig.TransmitterPower = 0;                     % Transmit power in dBm
```

```
rangeConfig.TransmitterAntennaGain =  0 ;    % Transmitter antenna gain
rangeConfig.ReceiverAntennaGain =  0 ;       % Receiver antenna gain in
if strcmp(rangeConfig.Environment,'Industrial')                    % Link margin(dB) assumed in
    rangeConfig.LinkMargin = 7
elseif strcmp(rangeConfig.Environment,'Outdoor')
    rangeConfig.LinkMargin = 15
else
    rangeConfig.LinkMargin = 0
end
```

```
rangeConfig =
  bluetoothRangeConfig with properties:

                Environment: 'Outdoor'
             SignalPowerType: 'ReceiverSensitivity'
                        Mode: 'LE1M'
         ReceiverSensitivity: -73
                  LinkMargin: 15
            TransmitterPower: 0
      TransmitterAntennaGain: 0
         ReceiverAntennaGain: 0
        TransmitterCableLoss: 1.2500
           ReceiverCableLoss: 1.2500
    TransmitterAntennaHeight: 1
       ReceiverAntennaHeight: 1

   Read-only properties:
               FSPLDistance: 5.8256
             PathLossModel: 'TwoRayGroundReflection'
```

Configure parameters for waveform generation

```
sps = 8;                                          % Samples per symbol
dataLen =  254 ;                                    % Data length in bytes
channelIndex =  37 ;                                % Random channel index
```

Configure RF impairments

```
frequencyOffset =  5800 ; % Frequency offset in Hz
phaseOffset =  5 ;                                % Phase offset in degrees
initoff = 0.15*sps;                               % Static timing offset
stepsize = 20*1e-6;                               % Timing drift in ppm, Max range is +/- 50 ppm
dcOffset = 20;                                    % Percentage related to maximum amplitude value
```

Generate Bluetooth LE Waveform

Generate Bluetooth LE waveform based on waveform configuration parameters.

```
% Default access address for periodic advertising channels
accessAddress = [0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1]';

% Random data bits generation
```

```
txBits = randi([0 1],dataLen*8,1,'int8');

% Generate Bluetooth LE waveform
txWaveform = bleWaveformGenerator(txBits,'Mode',rangeConfig.Mode,...
                    'SamplesPerSymbol',sps,...
                    'ChannelIndex',channelIndex,...
                    'AccessAddress',accessAddress);
```

Configure noise and signal power at the receiver

The noise floor of the receiver is simulated with thermal noise. The height of the noise floor determines the SNR at the receiver. The noise figure of the receiver determines the level of noise floor.

```
NF = 6;                                                 % Noise figure (dB)
T = 290;                                                % Ambient temperature (K)
dBm2dBFactor = 30;                                      % Factor for converting dBm to dB

% Symbol rate based on the PHY transmission mode
symbolRate = 1e6;
if strcmp(rangeConfig.Mode,'LE2M')
    symbolRate = 2e6;
end
BW = sps*symbolRate; % Bandwidth (Hz)
k = 1.3806e-23;                    % Boltzmann constant (J/K)
noiseFloor = 10*log10(k*T*BW)+NF; % Nosie floor in dB

% Measure the path loss and signal power at the receiver.
[pldB,sigPowerdBm] = pathLoss(rangeConfig);
measuredPowerVector = sigPowerdBm - dBm2dBFactor;
snrdB = measuredPowerVector - noiseFloor; % SNR in dB
plLinear = 10^(pldB/20); % Convert path loss from dB to linear scale
```

Distort Bluetooth LE Waveform

Distort the generated Bluetooth LE waveform using RF impairments, path loss, and AWGN.

Add RF Impairments

The RF impairments are generated randomly and added to the Bluetooth LE waveform.

```
% Create and configure the System objects for impairments
initImp = helperBLEImpairmentsInit(rangeConfig.Mode,sps);

% Configure RF impairments
initImp.pfo.FrequencyOffset = frequencyOffset;              % Frequency offset in Hz
initImp.pfo.PhaseOffset = phaseOffset;                      % Phase offset in degrees
initImp.vdelay = (initoff:stepsize:initoff+stepsize*(length(txWaveform)-1))';
initImp.dc = dcOffset;

% Pass generated Bluetooth LE waveform through RF impairments
txImpairedWfm = helperBLEImpairmentsAddition(txWaveform,initImp);
```

Attenuate Impaired Bluetooth LE Waveform

Attenuate the impaired Bluetooth LE waveform.

```
% Attenuate Bluetooth LE waveform
attenWaveform  = txImpairedWfm.*10^(measuredPowerVector/20);
```

Add AWGN

Add AWGN to the attenuated Bluetooth LE waveform.

```
rxWaveform = awgn(attenWaveform,snrdB,'measured');
```

Simulation Results

Estimate and display the BER and the distance between the transmitter and the receiver by processing the distorted Bluetooth LE waveform through the practical receiver.

Receiver Processing

To retrieve the data bits, pass the attenuated, AWGN-distorted Bluetooth LE waveform through the practical receiver.

```
% Configure the receiver parameters in a structure
rxCfg = struct(Mode=rangeConfig.Mode,SamplesPerSymbol=sps,ChannelIndex=channelIndex,...
    DFPacketType='Disabled',AccessAddress=accessAddress);
rxCfg.CoarseFreqCompensator = comm.CoarseFrequencyCompensator(Modulation="OQPSK",...
                                        SampleRate=BW,...
                                        SamplesPerSymbol=2*sps,...
                                        FrequencyResolution=100);
rxCfg.PreambleDetector = comm.PreambleDetector(Detections="First");

% Recover data bits using practical receiver
[rxBits,accessAddress] = helperBLEPracticalReceiver(rxWaveform,rxCfg);
```

Estimate BER

Estimate value of the BER based on the retrieved and the transmitted data bits.

```
% Obtain BER by comparing the transmitted and recovered bits
if(length(txBits) == length(rxBits))
    ber = (sum(xor(txBits,rxBits))/length(txBits));
else
   disp('Unable to compute BER due to length mismatch in input and decoded bits')
end
```

Estimate Distance

Estimate the distance between the transmitter and the receiver.

```
% Estimate the distance between the transmitter and the receiver based on the environment
distance = bluetoothRange(rangeConfig);
```

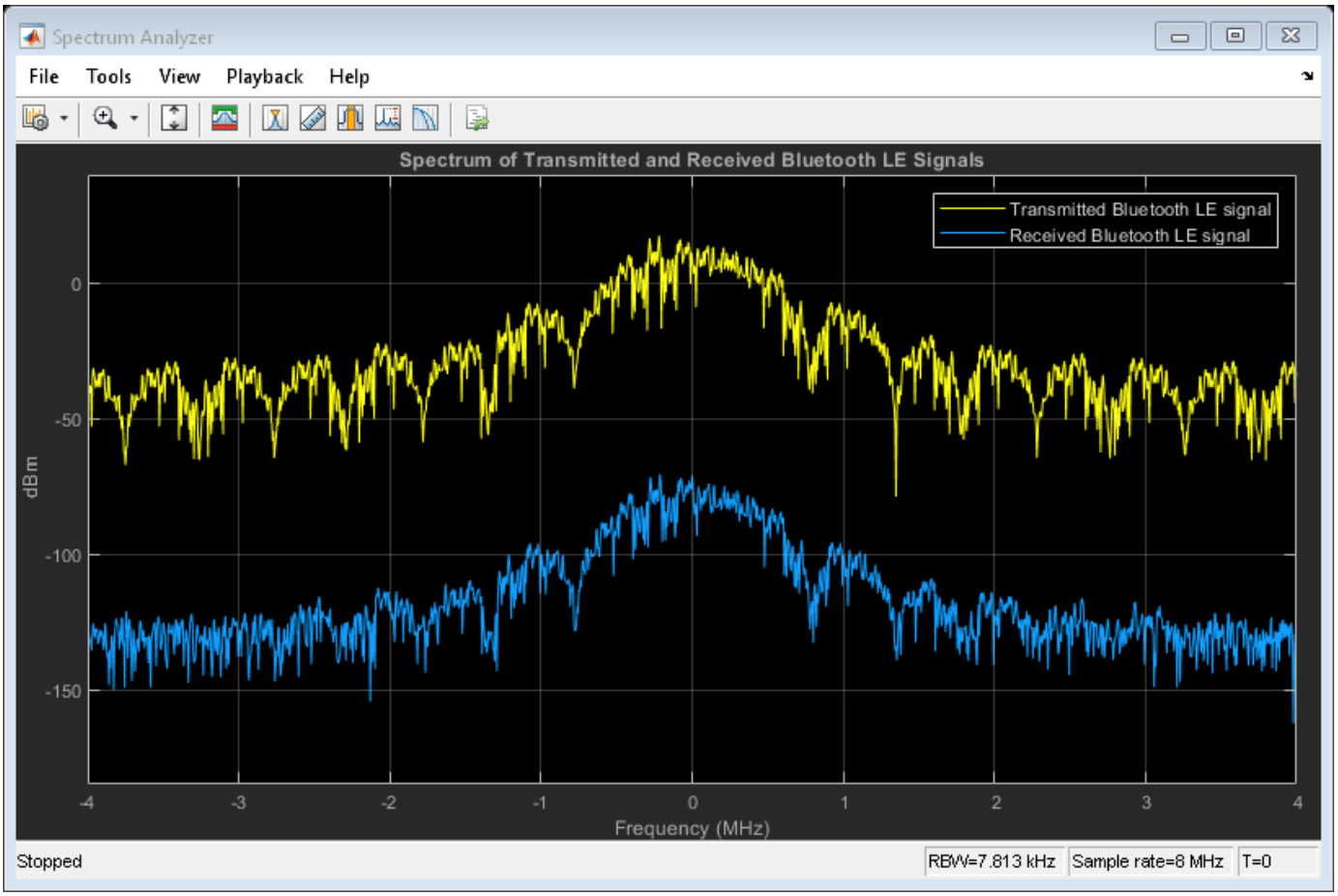
Display Results

Display the estimated results and plot the spectrum of the transmitted and received Bluetooth LE waveform.

```
% Display estimated BER and distance between the transmitter and the receiver.
disp(['Input configuration: ', newline , '    PHY transmission mode: ', rangeConfig.Mode,....
    newline,'    Environment: ', rangeConfig.Environment]);
```

```
Input configuration:
    PHY transmission mode: LE1M
    Environment: Outdoor
```

```
disp(['Estimated outputs: ', newline , '    Path loss : ', num2str(pldB), ' dB'....
    newline,'    Distance between the transmitter and receiver: ', num2str(round(distance(1))),
    '    Free space path loss distance: ', num2str(round(rangeConfig.FSPLDistance)), ' m', newlin

Estimated outputs:
    Path loss : 55.5 dB
    Distance between the transmitter and receiver: 5 to 8 m
    Free space path loss distance: 6 m
    BER: 0

% Plot the spectrum of the transmitted and received Bluetooth LE waveform
specAnalyzer = dsp.SpectrumAnalyzer('NumInputPorts',2,'SampleRate',symbolRate*sps,...
    'Title','Spectrum of Transmitted and Received Bluetooth LE Signals',...
   'ShowLegend',true,'ChannelNames',{'Transmitted Bluetooth LE signal','Received Bluetooth LE sig
specAnalyzer(txWaveform,rxWaveform);
release(specAnalyzer);
```



This example demonstrates an end-to-end Bluetooth LE simulation for different PHY transmission modes by considering the path loss model, RF impairments, and AWGN. The obtained simulation results display the path loss, estimated distance between the transmitter and receiver, and BER. The spectrum of the transmitted and received Bluetooth LE waveform is visualized by using a spectrum analyzer.

Appendix

The example uses these helper functions:

- helperBLEImpairmentsAddition.m: Add RF impairments to the Bluetooth LE waveform.
- helperBLEPracticalReceiver.m: Demodulate and decodes the received Bluetooth LE waveform.
- helperBLEImpairmentsInit.m: Initialize RF impairment parameters.

Selected Bibliography

[1] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". Version 5.3. https://www.bluetooth.com.

[2] *Path Loss Models Used in Bluetooth Range Estimator*. Bluetooth Special Interest Group (SIG). https://www.bluetooth.com.

[3] Rappaport, Theodore. *Wireless Communication – Principles and Practice*. Prentice Hall, 1996.

[4] *NIST Smart Grid Interoperability Panel Priority Action Plan 2: Guidelines for Assessing Wireless Standards for Smart Grid Applications*. National Institute of Standards and Technology, U.S. Department of Commerce, 2014, https://nvlpubs.nist.gov/.

See Also

Functions

`bleWaveformGenerator` | `bleIdealReceiver`

More About

- “End-to-End Bluetooth LE PHY Simulation with AWGN, RF Impairments and Corrections” on page 5-42
- “Bluetooth LE Bit Error Rate Simulation with AWGN” on page 1-19
- “Generate Bluetooth LE Waveform and Add RF Impairments” on page 9-25

End-to-End Bluetooth BR/EDR PHY Simulation with AWGN, RF Impairments and Corrections

This example shows an end-to-end simulation to measure the bit error rate (BER) and packet error rate (PER) for different Bluetooth® BR/EDR physical layer (PHY) packet types by using Bluetooth® Toolbox. These PHY packets are distorted by adding radio front-end (RF) impairments and the additive white Gaussian noise (AWGN). The distorted Bluetooth BR/EDR waveforms are processed at the practical receiver to get the BER and PER values. The obtained simulation results show the plots of BER and PER as a function of energy-to-noise density ratio (Eb/No).

Bluetooth BR/EDR Radio Specifications

Bluetooth is a short-range Wireless Personal Area Network (WPAN) technology, operating in the globally unlicensed industrial, scientific, and medical (ISM) band in the frequency range of 2.4 GHz to 2.485 GHz. In Bluetooth technology, data is divided into packets. Each packet is transmitted on one of the 79 designated Bluetooth channels. The bandwidth of each channel is 1 MHz. Bluetooth implements the frequency-hopping spread spectrum (FHSS) scheme to switch a carrier between multiple frequency channels by using a pseudorandom sequence known to the transmitter and the receiver.

The Bluetooth standard specifies these PHY modes:

Basic rate (BR) - Mandatory mode, uses Gaussian frequency shift keying (GFSK) modulation with a data rate of 1 Mbps.

Enhanced data rate (EDR) - Optional mode, uses phase shift keying (PSK) modulation with these two variants:

- **EDR2M:** Uses pi/4-DQPSK with a data rate of 2 Mbps
- **EDR3M:** Uses 8-DPSK with a data rate of 3 Mbps

This end-to-end Bluetooth BR/EDR PHY simulation determines BER and PER performance of one Bluetooth packet that has RF impairments and AWGN. Each packet is generated over a loop of a vector equal to length of the energy-to-noise density ratio (Eb/No) using the `bluetoothWaveformGenerator` function by configuring the `bluetoothWaveformConfig` object.

To accumulate the error rate statistics, the generated waveform is altered with RF impairments and AWGN before passing through the receiver.

These RF impairments are used to distort the packet:

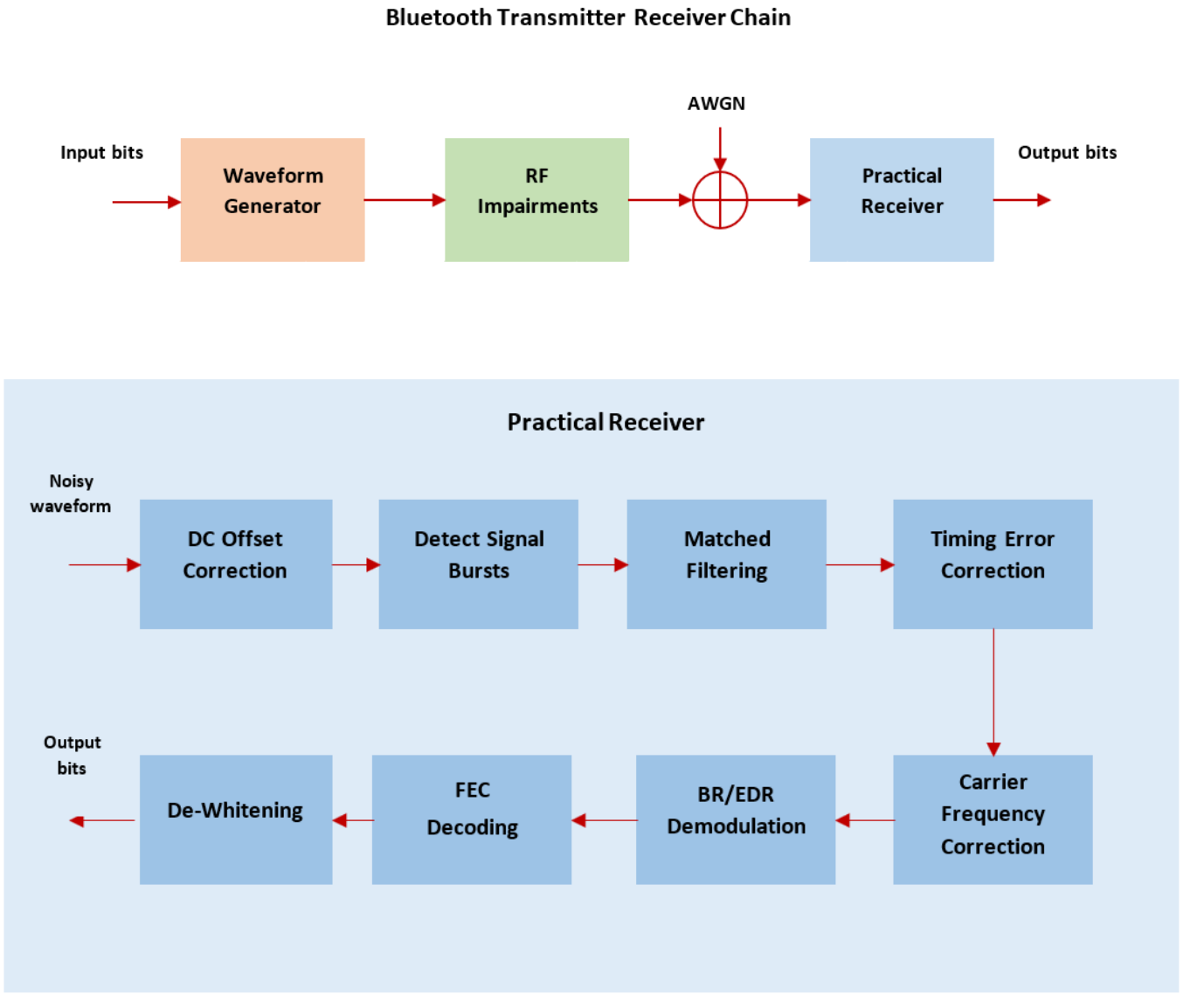
- DC offset
- Carrier frequency offset
- Static timing offset
- Timing drift

White Gaussian noise is added to the generated Bluetooth BR/EDR waveforms. The distorted and noisy waveforms are processed through a practical Bluetooth receiver performing these operations:

- Remove DC offset
- Detect the signal bursts

- Perform matched filtering
- Estimate and correct the timing offset
- Estimate and correct the carrier frequency offset
- Demodulate BR/EDR waveform
- Perform forward error correction (FEC) decoding
- Perform data dewhitening
- Perform header error check (HEC) and cyclic redundancy check (CRC)
- Outputs decoded bits and decoded packet statistics based on decoded lower address part (LAP), HEC, and CRC

This block diagram illustrates the processing steps for each Bluetooth BR/EDR PHY packet.



To determine the BER and the PER, compare the recovered output bits with the transmitted data bits.

Initialize Simulation Parameters

```
% Eb/No in dB
EbNo = 2:2:14;
% Maximum number of bit errors simulated at each Eb/No point
maxNumErrs = 100;
% Maximum number of bits accumulated at each Eb/No point
maxNumBits = 1e6;
% Maximum number of packets considered at each Eb/No point
maxNumPkts = 1000;
```

In this example, the values for maxNumErrs, maxNumBits, and maxNumPkts are selected for a short simulation time.

Configure Bluetooth BR/EDR Waveform

The Bluetooth BR/EDR waveform is configured by using the `bluetoothWaveformConfig` object. Configure the properties of the `bluetoothWaveformConfig` object as per your requirements. In this example, the PHY mode of transmission, the Bluetooth packet type, and the number of samples per symbol are configured.

```
phyMode = 'BR';                 % PHY transmission mode
bluetoothPacket = 'FHS';        % Type of Bluetooth packet, this value can be: {'ID',
                                % 'NULL','POLL','FHS','HV1','HV2','HV3','DV','EV3',
                                % 'EV4','EV5','AUX1','DM3','DM1','DH1','DM5','DH3',
                                % 'DH5','2-DH1','2-DH3','2-DH5','2-DH1','2-DH3',
                                %  '2-DH5','2-EV3','2-EV5','3-EV3','3-EV5'}
sps = 8;                        % Samples per symbol, must be greater than 1
```

Configure RF Impairments

Set frequency, time, and DC offset parameters to distort the Bluetooth BR/EDR waveform.

```
frequencyOffset = 6000;% In Hz
timingOffset = 0.5;    % In samples, less than 1 microsecond
timingDrift = 2;       % In parts per million
dcOffset = 2;          % Percentage w.r.t maximum amplitude value
symbolRate = 1e6;      % Symbol Rate

% Create timing offset object
timingDelayObj = dsp.VariableFractionalDelay;

% Create frequency offset object
frequencyDelay = comm.PhaseFrequencyOffset('SampleRate',symbolRate*sps);
```

Process Eb/No Points

For each Eb/No point, packets are generated and processed through these steps:

- Generate random bits
- Generate Bluetooth BR/EDR waveform
- Pass generated waveform through AWGN channel

- Add frequency offset
- Add timing offset
- Add DC offset
- Pass distorted waveform through practical receiver
- Calculate BER and PER

```
ber = zeros(1,length(EbNo));       % BER results
per = zeros(1,length(EbNo));       % PER results
bitsPerByte = 8;                   % Number of bits per byte
% Set code rate based on packet
if any(strcmp(bluetoothPacket,{'FHS','DM1','DM3','DM5','HV2','DV','EV4'}))
    codeRate = 2/3;
elseif strcmp(bluetoothPacket,'HV1')
    codeRate = 1/3;
else
    codeRate = 1;
end
% Set number of bits per symbol based on the PHY transmission mode
bitsPerSymbol = 1+ (strcmp(phyMode,'EDR2M'))*1 +(strcmp(phyMode,'EDR3M'))*2;

% Get SNR from EbNo values
snr = EbNo + 10*log10(codeRate) + 10*log10(bitsPerSymbol) - 10*log10(sps);
% Create a Bluetooth BR/EDR waveform configuration object
txCfg = bluetoothWaveformConfig('Mode',phyMode,'PacketType',bluetoothPacket,...
                                'SamplesPerSymbol',sps);
if strcmp(bluetoothPacket,'DM1')
    txCfg.PayloadLength = 17; % Maximum length of DM1 packets in bytes
end
dataLen = getPayloadLength(txCfg);  % Length of the payload
% Get PHY properties
rxCfg = getPhyConfigProperties(txCfg);

for iSnr = 1:length(snr)
        rng default
        % Initialize error computation parameters
        errorCalc = comm.ErrorRate;
        berVec = zeros(3,1);
        pktCount = 0;  % Counter for number of detected Bluetooth packets
        loopCount = 0; % Counter for number of packets at each SNR value
        pktErr = 0;
         while((berVec(2) < maxNumErrs) && (berVec(3) < maxNumBits) && (loopCount < maxNumPkts))
            txBits = randi([0 1],dataLen*bitsPerByte,1); % Data bits generation
            txWaveform = bluetoothWaveformGenerator(txBits,txCfg);

            % Add Frequency Offset
            frequencyDelay.FrequencyOffset = frequencyOffset;
            transWaveformCFO = frequencyDelay(txWaveform);

            % Add Timing Delay
            timingDriftRate = (timingDrift*1e-6)/(length(txWaveform)*sps);% Timing drift rate
            timingDriftVal = timingDriftRate*(0:1:(length(txWaveform)-1))';% Timing drift
            timingDelay = (timingOffset*sps)+timingDriftVal;   % Static timing offset and timing
            transWaveformTimingCFO = timingDelayObj(transWaveformCFO,timingDelay);

            % Add DC Offset
            dcValue = (dcOffset/100)*max(transWaveformTimingCFO);
```

```
            txImpairedWaveform = transWaveformTimingCFO + dcValue;

             % Add AWGN
            txNoisyWaveform = awgn(txImpairedWaveform,snr(iSnr),'measured');

            % Receiver Module
            [rxBits,decodedInfo,pktStatus]...
                                = helperBluetoothPracticalReceiver(txNoisyWaveform,rxCfg);
            numOfSignals = length(pktStatus);
            pktCount = pktCount+numOfSignals;
            loopCount = loopCount+1;

            % BER and PER Calculations
            L1 = length(txBits);
            L2 = length(rxBits);
            L = min(L1,L2);
            if(~isempty(L))
               berVec = errorCalc(txBits(1:L),rxBits(1:L));
            end
            pktErr = pktErr+sum(~pktStatus);
        end
        % Average of BER and PER
        per(iSnr) = pktErr/pktCount;
        ber(iSnr) = berVec(1);
        if ((ber(iSnr) == 0) && (per(iSnr) == 1))
            ber(iSnr) = 0.5; % If packet error rate is 1, consider average BER of 0.5
        end
        if ~any(strcmp(bluetoothPacket,{'ID','NULL','POLL'}))
            disp(['Mode ' phyMode ', '...
                'Simulated for Eb/No = ', num2str(EbNo(iSnr)), ' dB' ', '...
                'obtained BER:',num2str(ber(iSnr)),' obtained PER: ',...
                   num2str(per(iSnr))]);
        else
            disp(['Mode ' phyMode ', '...
                'Simulated for Eb/No = ', num2str(EbNo(iSnr)), ' dB' ', '...
                'obtained PER: ',num2str(per(iSnr))]);
        end
end
```

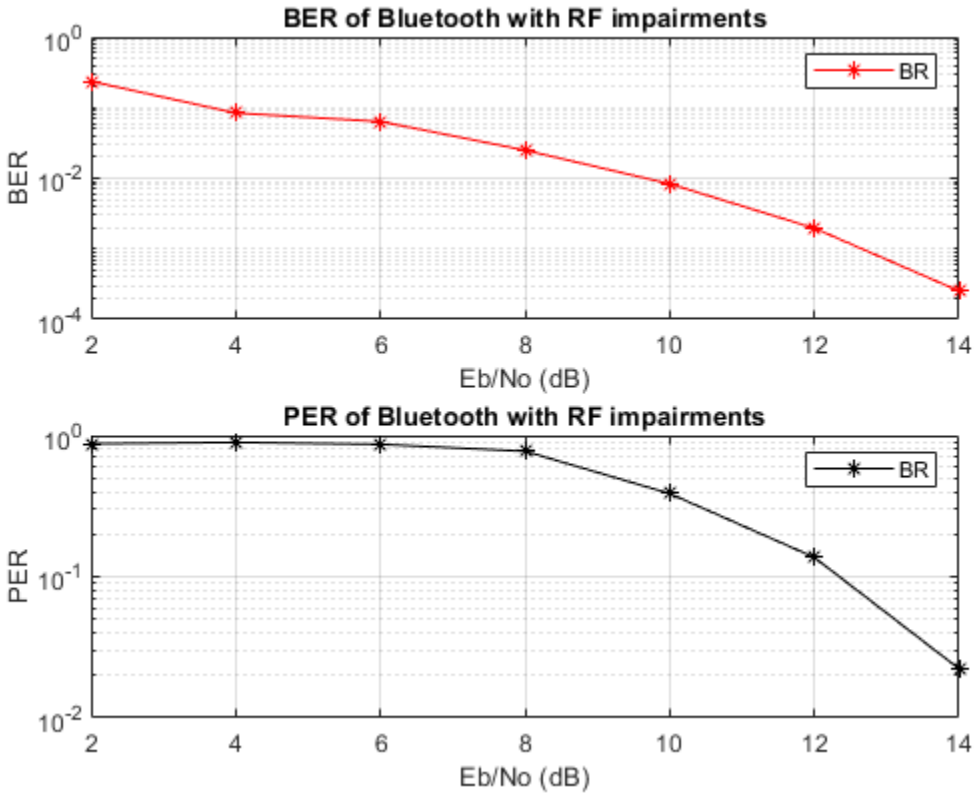
```
Mode BR, Simulated for Eb/No = 2 dB, obtained BER:0.23611 obtained PER: 0.875
Mode BR, Simulated for Eb/No = 4 dB, obtained BER:0.084028 obtained PER: 0.89474
Mode BR, Simulated for Eb/No = 6 dB, obtained BER:0.063492 obtained PER: 0.86667
Mode BR, Simulated for Eb/No = 8 dB, obtained BER:0.025 obtained PER: 0.77419
Mode BR, Simulated for Eb/No = 10 dB, obtained BER:0.0083333 obtained PER: 0.38824
Mode BR, Simulated for Eb/No = 12 dB, obtained BER:0.0019597 obtained PER: 0.13699
Mode BR, Simulated for Eb/No = 14 dB, obtained BER:0.00025304 obtained PER: 0.022267
```

Simulation Results

This section presents the BER and PER results with respect to the input Eb/No range for the considered PHY mode.

```
figure,
if any(strcmp(bluetoothPacket,{'ID','NULL','POLL'}))
    numOfPlots = 1; % Plot only PER
else
    numOfPlots = 2; % Plot both BER and PER
    subplot(numOfPlots,1,1),semilogy(EbNo,ber.','-r*');
```

```
    xlabel('Eb/No (dB)');
    ylabel('BER');
    legend(phyMode);
    title('BER of Bluetooth with RF impairments');
    hold on;
    grid on;
end
subplot(numOfPlots,1,numOfPlots),semilogy(EbNo,per.','-k*');
xlabel('Eb/No (dB)');
ylabel('PER');
legend(phyMode);
title('PER of Bluetooth with RF impairments');
hold on;
grid on;
```



Appendix

The example uses this helper function:

- helperBluetoothPracticalReceiver.m: Detects, synchronizes, and decodes the received Bluetooth BR/EDR waveform.

This example shows an entire end-to-end procedure to generate a Bluetooth BR/EDR packet. The generated Bluetooth BR/EDR waveform is distorted by adding RF impairments and AWGN. To get the BER and PER values, the distorted Bluetooth BR/EDR waveform is synchronized, demodulated, and decoded from the practical receiver.

Selected Bibliography

1 Bluetooth Special Interest Group (SIG). "Core System Package [BR/EDR Controller Volume]". *Bluetooth Core Specification*. Version 5.3, Volume 2. www.bluetooth.com

See Also

Functions

`bluetoothWaveformGenerator` | `bluetoothIdealReceiver`

More About

- “End-to-End Bluetooth BR/EDR PHY Simulation with WLAN Interference and Adaptive Frequency Hopping” on page 5-20
- “End-to-End Bluetooth BR/EDR PHY Simulation with Path Loss, RF Impairments, and AWGN” on page 5-35
- “Generate and Attenuate Bluetooth BR/EDR Waveform in Industrial Environment” on page 9-31

End-to-End Bluetooth BR/EDR PHY Simulation with WLAN Interference and Adaptive Frequency Hopping

This example presents an end-to-end simulation to demonstrate how adaptive frequency hopping (AFH) alters the frequency hopping sequence in Bluetooth® basic rate/enhanced data rate (BR/EDR) physical layer (PHY) and minimizes the impact of WLAN interference using Bluetooth® Toolbox. Unlike basic frequency hopping, AFH excludes the Bluetooth channels that are sources of interference. By excluding these channels from the list of available channels, AFH reassigns the transmission and reception of packets on channels with relatively less interference.

Using this example, you can:

- Generate a Bluetooth BR/EDR waveform.
- Add WLAN interference and additive white Gaussian noise (AWGN) to the generated Bluetooth BR/EDR waveform.
- Transmit and receive Bluetooth BR/EDR waveform by using basic frequency hopping and AFH techniques.
- Decode the received Bluetooth BR/EDR waveform.
- View and compare the packet error rate (PER) and bit error rate (BER) values of the Bluetooth BR/EDR waveforms in the presence of WLAN interference with AFH and basic frequency hopping.
- Visualize the power spectral density of Bluetooth BR/EDR waveforms in the presence of WLAN interference with AFH and basic frequency hopping.

Bluetooth BR/EDR PHY

The Bluetooth standard specifies two PHY modes: BR and EDR. The Bluetooth® Toolbox enables you to model Bluetooth BR/EDR communication system links, as specified in the Bluetooth Core Specification [2] on page 5-0 .

The Bluetooth BR mode is mandatory, whereas the EDR mode is optional. The Bluetooth BR/EDR radio implements a 1600 hops/s frequency hopping spread spectrum (FHSS) technique. The radio hops in a pseudo-random way on 79 designated Bluetooth channels. Each Bluetooth channel has a bandwidth of 1 MHz. Each channel is centered at (2402 + k) MHz, where k = 0, 1, ..., 78. The modulation technique on BR and EDR mode payloads is Gaussian frequency shift-keying (GFSK) and differential phase shift-keying (DPSK), respectively. The baud rate is 1 MSymbols/s. The Bluetooth BR/EDR radio uses time division duplex (TDD) in which data transmission occurs in one direction at one time. The transmission alternates in two directions, one after the other.

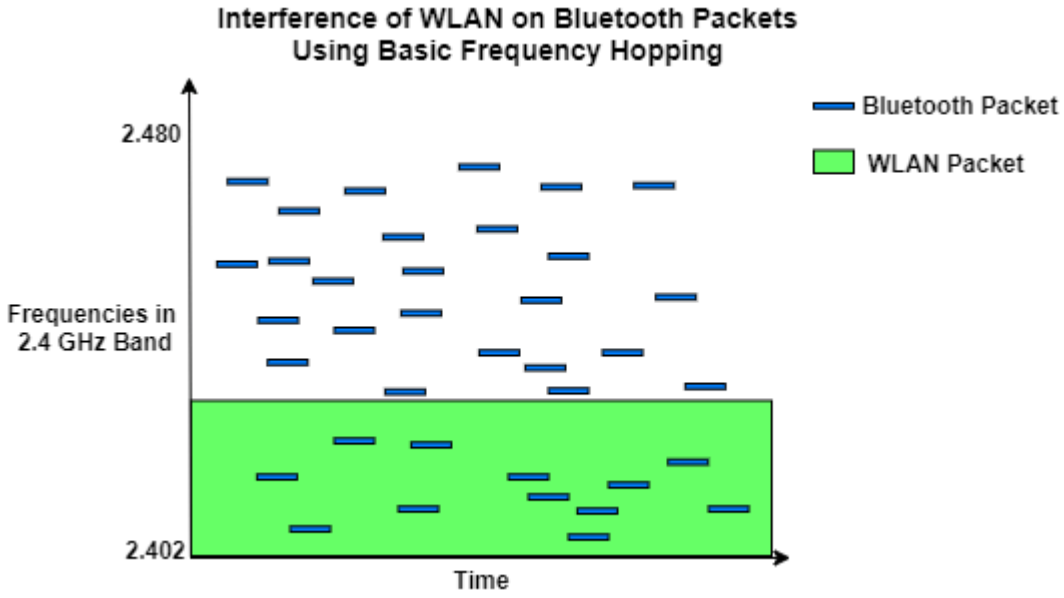
For more information about the Bluetooth BR/EDR radio and the protocol stack, see “Bluetooth Protocol Stack” on page 8-9. For more information about Bluetooth BR/EDR packet structures, see “Bluetooth Packet Structure” on page 8-27.

Adaptive Frequency Hopping

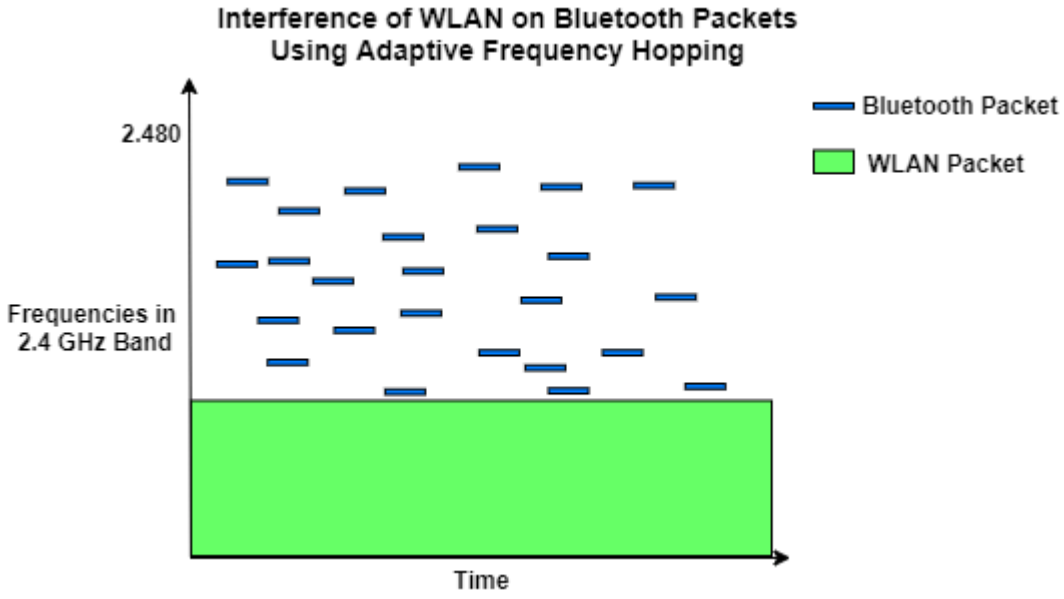
The objective of using FHSS in Bluetooth is to provide diversity that allows to minimize BER even if the interfering networks or the physical environment renders some channels unusable. Frequency hopping techniques can either implement a fixed sequence of channel hops such as with basic frequency hopping or adapt its hopping sequence dynamically with AFH to varying interference conditions.

Prior to AFH capability, Bluetooth devices implemented the basic frequency hopping scheme. In this approach, a Bluetooth radio hops in a pseudo-random way at the rate of 1600 hops/s. When another

wireless device operating in the same 2.4 GHz band comes into the environment, the basic frequency hopping scheme results in occasional collisions. For example, Bluetooth and WLAN are two such networks that operate in the 2.4 GHz frequency band. Bluetooth and WLAN radios often operate in the same physical scenario and on the same device. In these cases, Bluetooth and WLAN transmissions can interfere with each other. This interference impacts the performance and reliability of both networks. This figure shows a scenario in which Bluetooth and WLAN packet transmissions interfere with each other.



AFH enables Bluetooth to minimize collisions by avoiding sources of interference and excluding them from the list of available channels.This figure shows the previous scenario with AFH enabled.

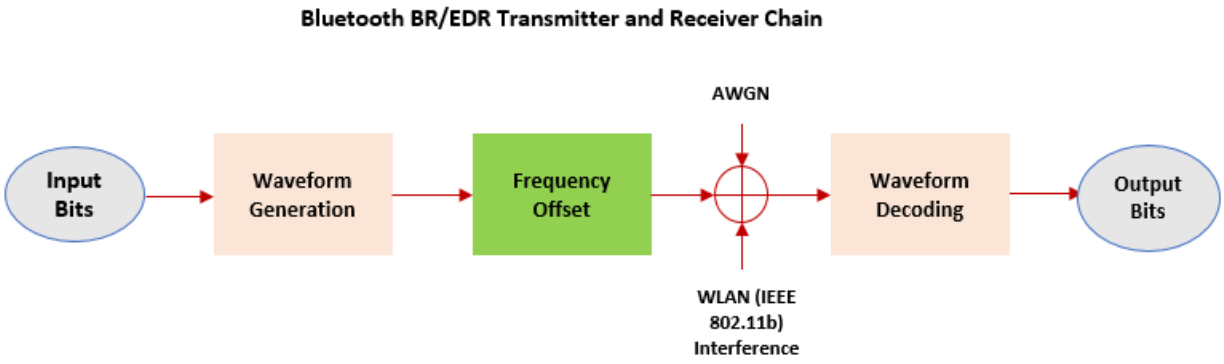


This procedure of remapping involves reducing the number of channels to be used by Bluetooth. The Bluetooth Core Specification [2] on page 5-0 require at least 20 channels for Bluetooth

transmissions. For more information about coexistence of Bluetooth and WLAN, see “Bluetooth-WLAN Coexistence” on page 8-53.

Bluetooth BR/EDR Transmitter and Receiver Chain

This example demonstrates an end-to-end Bluetooth BR/EDR waveform processing by using the frequency hopping mechanism defined in the Bluetooth Core Specification [2] on page 5-0 . The generated Bluetooth BR/EDR waveform is frequency modulated and then distorted with WLAN interference. This flowchart shows the Bluetooth transmitter and receiver chain.



Transmitter chain

1. Select a channel index for the transmission.
2. Generate random input bits.
3. Generate a Bluetooth BR/EDR waveform.
4. Apply a frequency offset based on the selected channel index.

Receiver chain

1. Select a channel index for reception.
2. Apply a frequency offset based on the selected channel index.
3. Decode the Bluetooth BR/EDR waveform to get the output bits.

Wireless Channel

1. Add WLAN (IEEE 802.11b) interference to Bluetooth BR/EDR waveform.
2. Add AWGN to Bluetooth BR/EDR waveform.

Results

The example displays these results for basic frequency hopping and AFH.

- The PER and BER for the simulations performed under an additive white gaussian noise (AWGN) channel for a given bit energy to noise density ratio (Eb/No) value
- The received signal spectrum and the spectrogram of the channel

- A plot displaying the selected channel index per transmission or reception slot

Configure Simulation Parameters

Configure the desired Bluetooth packet type, payload length, PHY mode, and simulation time.

```
simulationTime = 2*1e6; % Simulation time in microseconds
packetType     = DM1; % Specify baseband packet type
phyMode        = BR; % Specify PHY mode ('BR', 'EDR2M', 'EDR3M')
payloadLength  = 10; % Length of baseband packet in bytes
```

Configure Frequency Hopping

Use the `bluetoothFrequencyHop` object to select a channel index for the transmission and reception of Bluetooth BR/EDR waveforms.

```
% Bluetooth frequency hopping
frequencyHop = bluetoothFrequencyHop;
frequencyHop.SequenceType = 'Connection Adaptive';
```

Configure Bluetooth PHY

Use the helperBluetoothPHY helper object to model the Bluetooth BR/EDR waveform transmission and reception.

```
% Configure Bluetooth PHY transmission
phyTx = helperBluetoothPHY;
phyTx.Mode = phyMode;

% Configure Bluetooth PHY reception
phyRx = helperBluetoothPHY;
phyRx.Mode = phyMode;
```

Configure Channel and WLAN Interference:

Use helperBluetoothChannel object to configure the wireless channel. You can set the `EbNo` value for the AWGN channel. To generate the interfering WLAN waveform, use the helperBluetoothGenerateWLANWaveform function. Specify the sources of WLAN interference by using `wlanInterferenceSource` parameter. The WLAN signal is present between -10 to 10 MHz throughout the simulation. Use one of these options to specify the source of WLAN interference.

- ‘Generated’: To add WLAN signal from the WLAN Toolbox™, select this option.
- ‘BasebandFile’: To add a WLAN signal from a baseband file (`.bb`), select this option and specify the baseband file name in the `WLANBBFilename` property. If you do not specify the `.bb` file, the example uses the default `.bb` file, `WLANNonHTDSSS.bb`, to add the WLAN signal.
- ‘None’: No WLAN signal is added.

AWGN is present throughout the simulation.

```
% Specify as one of 'Generated' | 'BasebandFile' | 'None'
wlanInterferenceSource = BasebandFile;
wlanBBFilename = 'WLANNonHTDSSS.bb'; % Default baseband file

% Configure wireless channel
```

```
channel = helperBluetoothChannel;
channel.EbNo = 22; % Ratio of energy per bit (Eb) to the spectral noise density (No) in dB
channel.SIR = -20; % Signal to interference ratio in dB
if ~strcmpi(wlanInterferenceSource, 'None')
    % Generate the WLAN waveform
    wlanWaveform = helperBluetoothGenerateWLANWaveform(wlanInterferenceSource, wlanBBFilename);

    % Add the WLAN interference to Bluetooth channel
    addWLANWaveform(channel, wlanWaveform);
end
```

Simulation Setup

Initialize parameters to perform the end-to-end Bluetooth BR/EDR simulation.

```
slotTime = 625; % Bluetooth slot duration in microseconds

% Simulation time in terms of slots
numSlots = floor(simulationTime/slotTime);

% Slot duration, including transmission and reception
slotValue = phyTx.slotsRequired(packetType)*2;

% Number of Central transmission slots
numCentralTxSlots = floor(numSlots/slotValue);

% Total number of Bluetooth physical channels
numBluetoothChannels = 79;

% errorsBasic and errorsAdaptive store relevant bit and packet error
% information per channel. Each row stores the channel index, bit errors,
% packet errors, total bits, and BER per channel. errorsBasic and
% errorsAdaptive arrays store these values for basic frequency hopping
% and AFH, respectively.
[errorsBasic, errorsAdaptive] = deal(zeros(numBluetoothChannels,5));

% Initialize first column with channel numbers
[errorsBasic(:,1), errorsAdaptive(:,1)] = deal(0:78);

% Initialize variables for calculating PER and BER
[berBasic, berAdaptive, bitErrors] = deal(0);
badChannels = zeros(1,0);
totalTransmittedPackets = numCentralTxSlots;

% Number of bits per octet
octetLength = 8;

% Sample rate and input clock used in PHY processing
samplePerSymbol = 88;
symbolRate = 1e6;
sampleRate = symbolRate*samplePerSymbol;
inputClock = 0;

% Store hop index
hopIndex = zeros(1, numCentralTxSlots);

% Index to hop index vector
hopIdx = 1;
```

```
% Baseband packet structure
basebandData = struct(...
    'LTAddr',1,             ... % Logical transport address
    'PacketType',packetType,... % Packet type
    'Payload',zeros(1,phyTx.MaxPayloadSize), ... % Payload
    'PayloadLength',0,  ... % Payload length
    'LLID',[0; 0],      ... % Logical link identifier
    'SEQN',0,           ... % Sequence number
    'ARQN',1,           ... % Acknowledgment flag
    'IsValid',false);   ... % Flag to identify the status of cyclic redundancy check (CRC) and he

% Bluetooth signal structure
bluetoothSignal = struct(...
    'PacketType',packetType, ... % Packet type
    'Waveform',[],           ... % Waveform
    'NumSamples',[],         ... % Number of samples
    'SampleRate',sampleRate, ... % Sample rate
    'SamplesPerSymbol',samplePerSymbol,      ... % Samples per symbol in Hz
    'Payload',zeros(1,phyTx.MaxPayloadSize), ... % Payload
    'PayloadLength',0, ... % Payload length in bytes
    'SourceID',0,      ... % Source identifier
    'Bandwidth',1,     ... % Bandwidth
    'NodePosition',[0 0 0], ... % Node position
    'CenterFrequency',centerFrequency(phyTx), ... % Center frequency
    'StartTime',0, ... % Waveform start time in microseconds
    'EndTime',0,   ... % Waveform end time in microseconds
    'Duration',0); ... % Waveform duration in microseconds

% Clock ticks(one slot is 2 clock ticks)
clockTicks = slotValue*2;
```

To visualize the Bluetooth BR/EDR waveforms, create a `dsp.SpectrumAnalyzer` System object™.

```
% Spectrum analyzer for basic frequency hopping
spectrumAnalyzerBasic = dsp.SpectrumAnalyzer(...
    'Name','Bluetooth Basic Frequency Hopping', ...
    'ViewType','Spectrum and spectrogram', ...
    'TimeResolutionSource','Property', ...
    'TimeResolution',0.0005, ...    % In seconds
    'SampleRate',sampleRate, ...    % In Hz
    'TimeSpanSource','Property', ...
    'TimeSpan', 0.05, ...           % In seconds
    'FrequencyResolutionMethod', 'WindowLength', ...
    'WindowLength', 512, ...        % In samples
    'AxesLayout', 'Horizontal', ...
    'FrequencyOffset',2441*1e6, ... % In Hz
    'ColorLimits',[-20 15]);

% Spectrum analyzer for AFH
spectrumAnalyzerAdaptive = dsp.SpectrumAnalyzer(...
    'Name','Bluetooth Adaptive Frequency Hopping', ...
    'ViewType','Spectrum and spectrogram', ...
    'TimeResolutionSource','Property', ...
    'TimeResolution',0.0005, ...    % In seconds
    'SampleRate',sampleRate, ...    % In Hz
    'TimeSpanSource','Property', ...
    'TimeSpan',0.05, ...            % In seconds
```

```
    'FrequencyResolutionMethod','WindowLength', ...
    'WindowLength',512, ...          % In samples
    'AxesLayout','Horizontal', ...
    'FrequencyOffset',2441*1e6, ... % In Hz
    'ColorLimits',[-20 15]);
```

Simulations

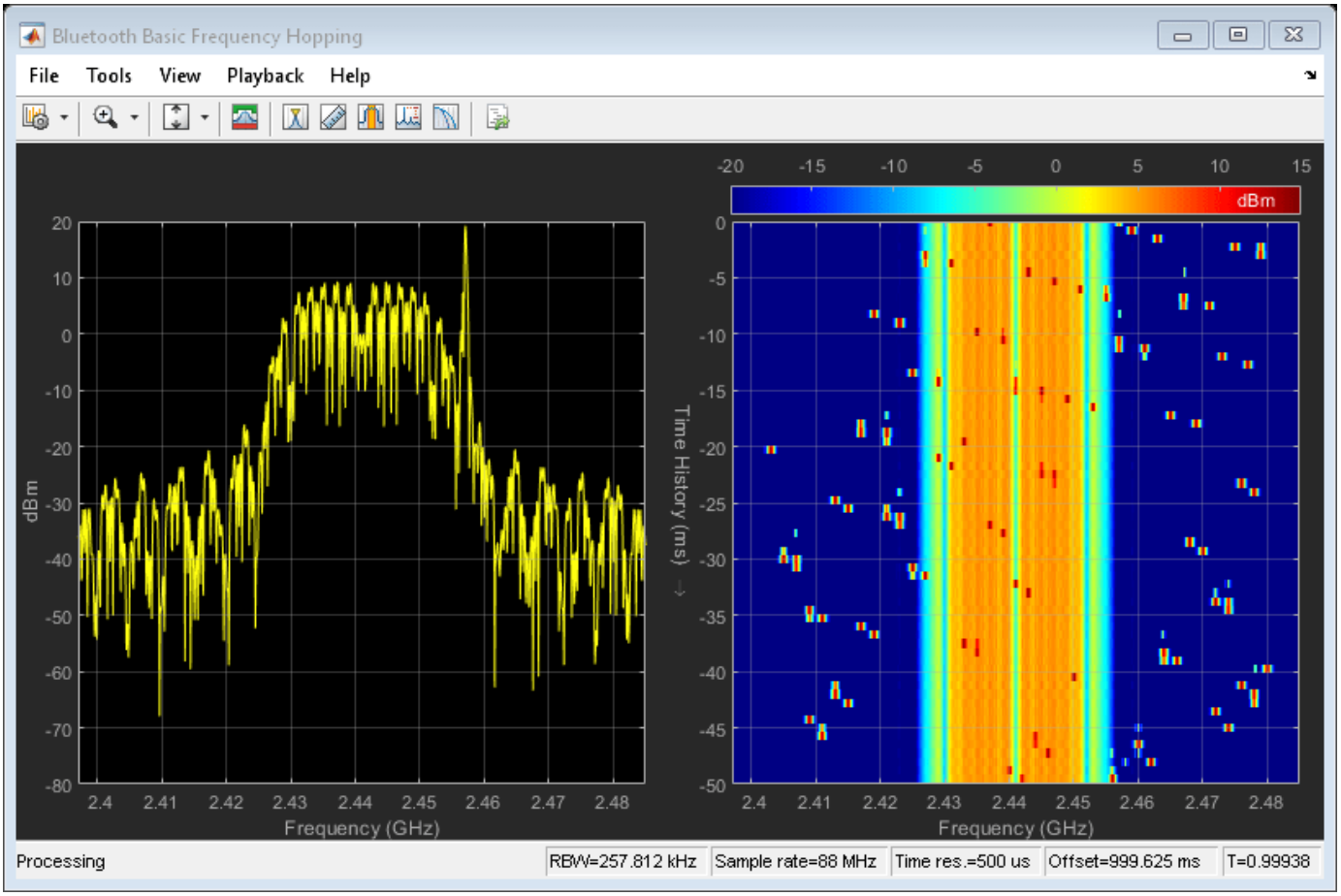
The Bluetooth transmitter and receiver chain is simulated using basic frequency hopping and AFH. Using per channel PER and BER results of basic frequency hopping, derive a list of used channels. The list of used channels is fed as an input to the simulation using AFH.

Basic Frequency Hopping

The simulation runs for all the specified number of Central transmission slots. Simulates the transmitter chain, receiver chain, and channel for each slot. At the end of the simulation, the example computes the PER and BER for all the Bluetooth BR/EDR waveforms.

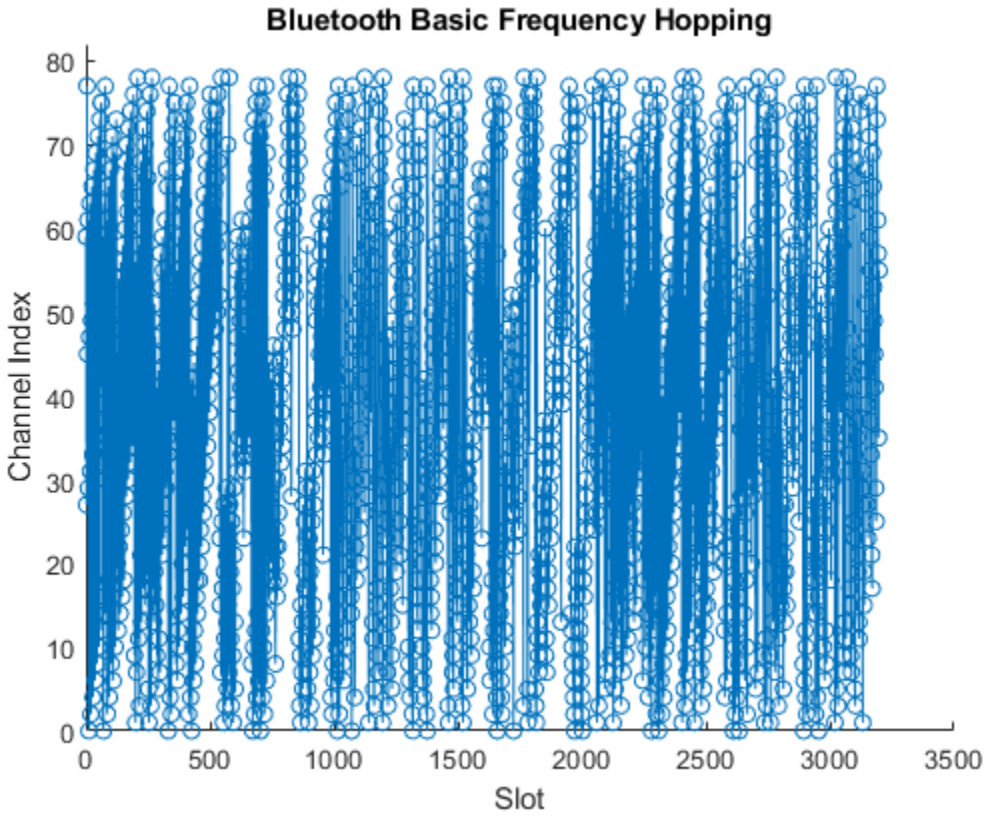
```
% Set the default random number generator ('twister' type with seed value 0).
% The seed value controls the pattern of random number generation. For high
% fidelity simulation results, change the seed value and average the
% results over multiple simulations.
sprev = rng('default');
for slotIdx = 0:slotValue:numSlots-slotValue
    % Update clock
    inputClock = inputClock + clockTicks;

    % Frequency hopping
    [channelIndex,~] = nextHop(frequencyHop,inputClock);

    % PHY transmission
    stateTx = 1; % Transmission state
    TxBits = randi([0 1],payloadLength*octetLength,1);
    basebandData.Payload = TxBits;
    basebandData.PayloadLength = payloadLength;

    % Generate whiten initialization vector from clock
    clockBinary = int2bit(inputClock,28,false)';
    whitenInitialization = [clockBinary(2:7)'; 1];

    % Update the PHY with request from the baseband layer
    updatePHY(phyTx,stateTx,channelIndex,whitenInitialization,basebandData);

    % Initialize and pass elapsed time as zero
    elapsedTime = 0;
    [nextTxTime,btWaveform] = run(phyTx,elapsedTime); % Run PHY transmission
    run(phyTx, nextTxTime); % Update next invoked time

    % Channel
    bluetoothSignal.Waveform = btWaveform;
    bluetoothSignal.NumSamples = numel(btWaveform);
    bluetoothSignal.CenterFrequency = centerFrequency(phyTx);
    channel.ChannelIndex = channelIndex;
    bluetoothSignal = run(channel,bluetoothSignal,phyMode);
    distortedWaveform = bluetoothSignal.Waveform;

    % PHY reception
    stateRx = 2; % Reception state
```

```
        % Update the PHY with request from the baseband layer
        updatePHY(phyRx,stateRx,channelIndex,whitenInitialization);
        [nextRxTime,~] = run(phyRx,elapsedTime,bluetoothSignal);
        bluetoothSignal.NumSamples = 0;
        run(phyRx,nextRxTime,bluetoothSignal); % Run PHY reception
        chIdx = channelIndex + 1;

        % Calculate error rate upon successful decoding the packet
        if phyRx.Decoded
            rxBitsLength = phyRx.DecodedBasebandData.PayloadLength*octetLength;
            RxBits = phyRx.DecodedBasebandData.Payload(1:rxBitsLength);

            % BER calculation
            txSymLength = length(TxBits);
            rxSymLength = length(RxBits);
            minSymLength = min(txSymLength,rxSymLength);
            if minSymLength > 0
                bitErrors = sum(xor(TxBits(1:minSymLength),RxBits(1:minSymLength)));
                totalBits = minSymLength;

                % Bit errors found in channel
                errorsBasic(chIdx,2) = errorsBasic(chIdx,2) + bitErrors;

                % Total bits transmitted in channel
                errorsBasic(chIdx,4) = errorsBasic(chIdx,4) + totalBits;
            end
            if ~phyRx.DecodedBasebandData.IsValid || bitErrors
                % Packet errors found in channel
                errorsBasic(chIdx,3) = errorsBasic(chIdx,3) + 1;
            end
        else
            % Packet errors found in channel
            errorsBasic(chIdx,3) = errorsBasic(chIdx,3) + 1;
        end
        hopIndex(hopIdx) = channelIndex;
        hopIdx = hopIdx + 1;

        % Plot spectrum
        spectrumAnalyzerBasic(btWaveform + wlanWaveform(1:numel(btWaveform)));
        pause(0.01);
    end
```



In the previous figure, the plot on the left shows the spectrum of the Bluetooth BR/EDR waveform distorted with WLAN interference in the frequency domain and passed through the AWGN channel. The plot on the right shows that the WLAN signal is present from –10 to 10 MHz. The results show that Bluetooth packets with the interfering WLAN signal overlap.

```
% Plot selected channel index per slot
figBasic = figure('Name','Basic frequency hopping');
axisBasic = axes(figBasic);
xlabel(axisBasic,'Slot');
ylabel(axisBasic,'Channel Index');
ylim(axisBasic,[0 numBluetoothChannels+3]);
title(axisBasic,'Bluetooth Basic Frequency Hopping');
hold on;
plot(axisBasic,0:slotValue:numSlots-slotValue,hopIndex,'-o');
```



The preceding plot displays the selected channel index per transmission or reception slot using basic frequency hopping. In this case all of the channels are used channels.

Channel Classification

The channels with more packet errors are marked as bad channels. Based on the bit and packet errors collected from the above simulation, calculate the PER and BER.

```
% Select 25 channels with highest packet errors as bad channels
[~,indexes] = sort(errorsBasic(:,3),'descend');
badChannelIdx = min(nnz(errorsBasic(:,3)),25);
if badChannelIdx ~= 0
    badChannels = indexes(1:badChannelIdx) - 1;
end
usedChannels = setdiff(0:numBluetoothChannels-1,badChannels);

% BER per channel calculation
errorsBasic(:,5) = errorsBasic(:,2)./errorsBasic(:,4);
errorsBasic(:,5) = fillmissing(errorsBasic(:,5),'constant',0);
ber = nonzeros(errorsBasic(:,5));
if ~isempty(ber)
    berBasic = mean(ber);
end

% PER calculation
packetErrorsBasic = sum(errorsBasic(:,3));
perBasic = packetErrorsBasic/totalTransmittedPackets;
```

```
% Reset
hopIdx = 1;
[inputClock,bitErrors] = deal(0);
fprintf('PER of Bluetooth BR/EDR waveforms using basic frequency hopping: %.4f\n',perBasic);
```

```
PER of Bluetooth BR/EDR waveforms using basic frequency hopping: 0.2787
```

```
fprintf('BER of Bluetooth BR/EDR waveforms using basic frequency hopping: %.4f\n',berBasic);
```

```
BER of Bluetooth BR/EDR waveforms using basic frequency hopping: 0.0550
```

Adaptive Frequency Hopping

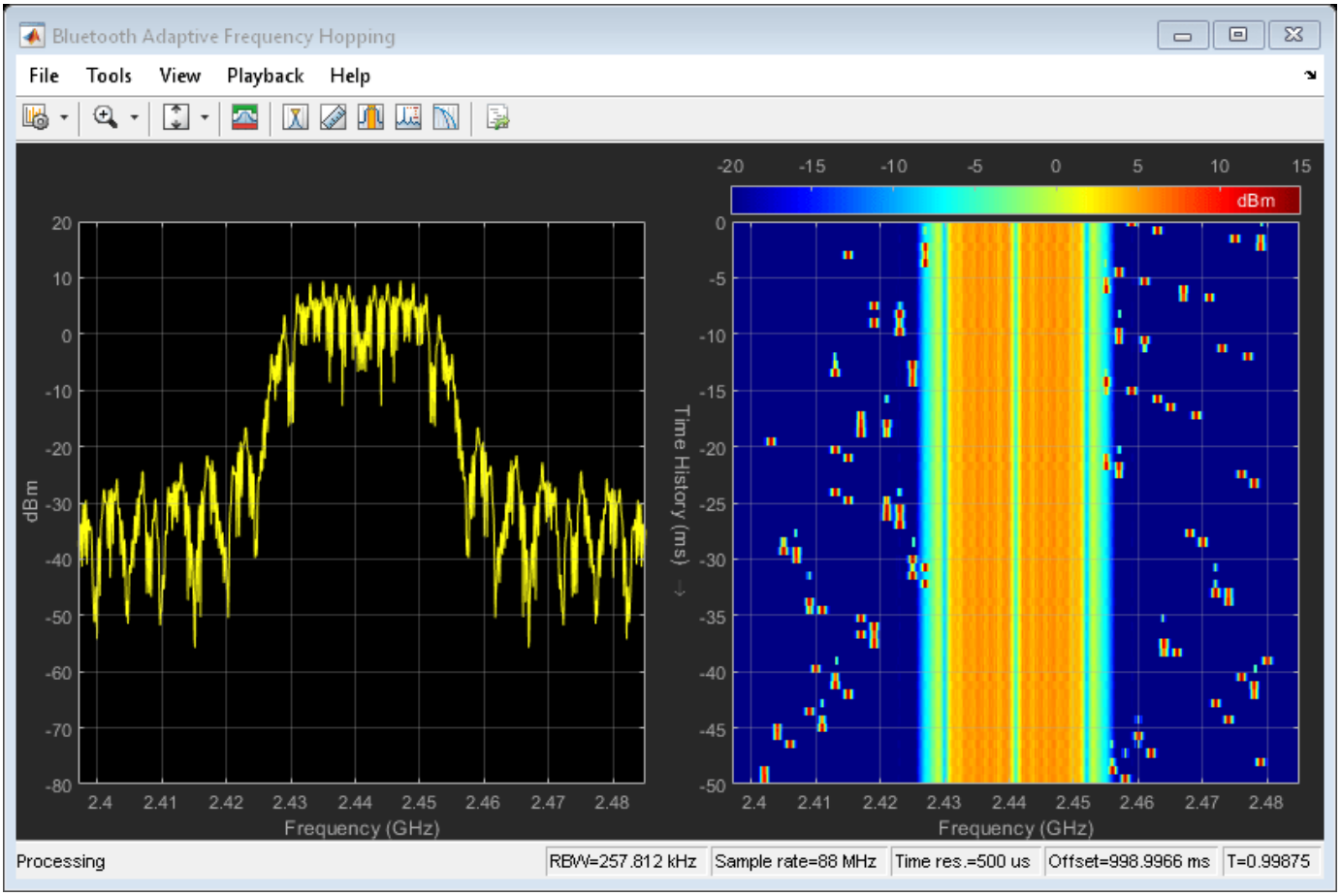
Set the value of the sequence type as `'Connection adaptive'` and specify the classified used channels.

```
frequencyHop.SequenceType = 'Connection adaptive';
frequencyHop.UsedChannels = usedChannels;
```

Run the simulation using AFH and compute the PER and BER.

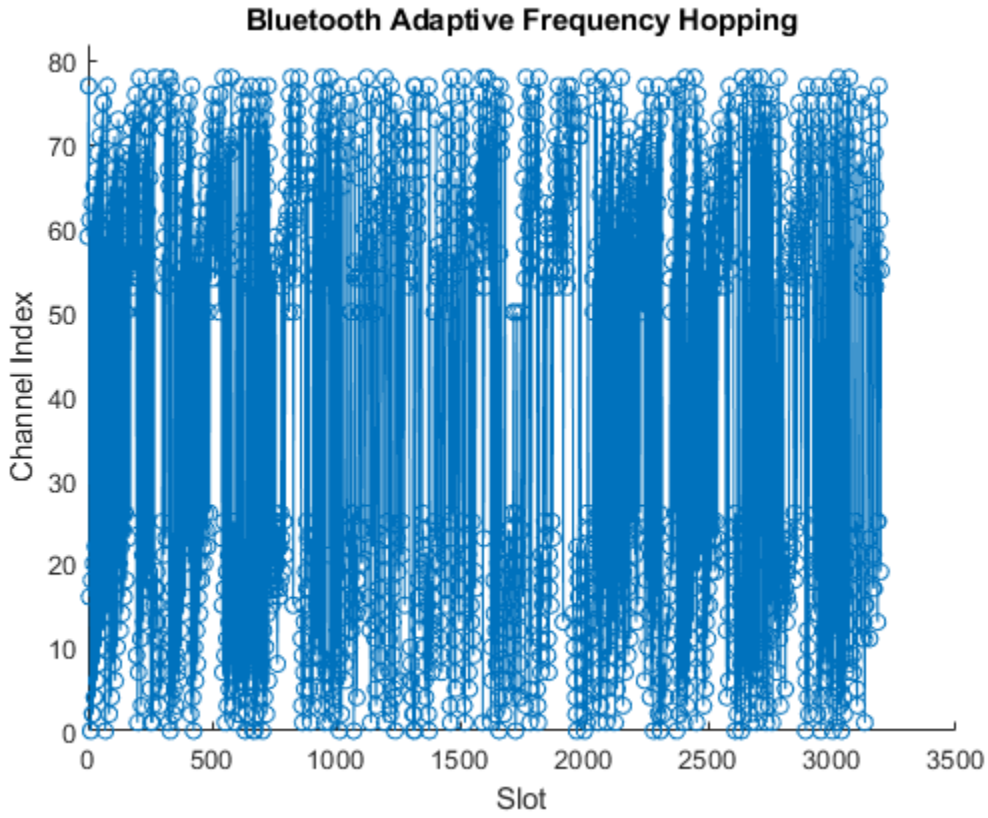
```
% Set the default random number generator ('twister' type with seed value 0).
% The seed value controls the pattern of random number generation. For high
% fidelity simulation results, change the seed value and average the
% results over multiple simulations.
rng('default');
for slotIdx = 0:slotValue:numSlots-slotValue
    % Update clock
    inputClock = inputClock + clockTicks;

    % Frequency hopping
    [channelIndex,~] = nextHop(frequencyHop,inputClock);

    % PHY transmission
    stateTx = 1; % Transmission state
    TxBits = randi([0 1],payloadLength*octetLength,1);
    basebandData.Payload = TxBits;
    basebandData.PayloadLength = payloadLength;

    % Generate whiten initialization vector from clock
    clockBinary = int2bit(inputClock,28,false)';
    whitenInitialization = [clockBinary(2:7)'; 1];

    % Update the PHY with request from the baseband layer
    updatePHY(phyTx,stateTx,channelIndex,whitenInitialization,basebandData);

    % Initialize and pass elapsed time as zero
    elapsedTime = 0;
    [nextTxTime,btWaveform] = run(phyTx,elapsedTime); % Run PHY transmission
    run(phyTx,nextTxTime); % Update next invoked time

    % Channel
    bluetoothSignal.Waveform = btWaveform;
    bluetoothSignal.NumSamples = numel(btWaveform);
    bluetoothSignal.CenterFrequency = centerFrequency(phyTx);
    channel.ChannelIndex = channelIndex;
    bluetoothSignal = run(channel,bluetoothSignal,phyMode);
    distortedWaveform = bluetoothSignal.Waveform;
```

```
        % PHY reception
        stateRx = 2; % Reception state

        % Update the PHY with request from the baseband layer
        updatePHY(phyRx,stateRx,channelIndex,whitenInitialization);
        [nextRxTime,~] = run(phyRx,elapsedTime,bluetoothSignal);
        bluetoothSignal.NumSamples = 0;
        run(phyRx,nextRxTime,bluetoothSignal); % Run PHY reception
        chIdx = channelIndex + 1;

        % Calculate error rate upon successful decoding the packet
        if phyRx.Decoded
            rxBitsLength = phyRx.DecodedBasebandData.PayloadLength*octetLength;
            RxBits = phyRx.DecodedBasebandData.Payload(1:rxBitsLength);

            % BER calculation
            txSymLength = length(TxBits);
            rxSymLength = length(RxBits);
            minSymLength = min(txSymLength,rxSymLength);
            if minSymLength > 0
                bitErrors = sum(xor(TxBits(1:minSymLength),RxBits(1:minSymLength)));
                totalBits = minSymLength;

                % Bit errors found in channel
                errorsAdaptive(chIdx,2) = errorsAdaptive(chIdx,2) + bitErrors;

                % Total bits transmitted in channel
                errorsAdaptive(chIdx,4) = errorsAdaptive(chIdx,4) + totalBits;
            end

            if ~phyRx.DecodedBasebandData.IsValid || bitErrors
                % Packet errors found in channel
                errorsAdaptive(chIdx, 3) = errorsAdaptive(chIdx, 3) + 1;
            end
        else
            % Packet errors found in channel
            errorsAdaptive(chIdx, 3) = errorsAdaptive(chIdx, 3) + 1;
        end
        hopIndex(hopIdx) = channelIndex;
        hopIdx = hopIdx + 1;

        % Plot spectrum
        spectrumAnalyzerAdaptive(btWaveform + wlanWaveform(1:numel(btWaveform)));
        pause(0.01);
    end
```



In the previous plot, you can observe that the transmission of Bluetooth packets does not overlap with the WLAN signal. AFH excludes the channels that are sources of WLAN interference and reassigns the transmission of Bluetooth packets on channels with relatively less interference.

```
% Plot selected channel index per slot
figAdaptive = figure('Name','Adaptive Frequency Hopping');
axisAdaptive = axes(figAdaptive);
xlabel(axisAdaptive,'Slot');
ylabel(axisAdaptive,'Channel Index');
title(axisAdaptive,'Bluetooth Adaptive Frequency Hopping');
ylim(axisAdaptive,[0 numBluetoothChannels+3]);
hold on;
plot(axisAdaptive,0:slotValue:numSlots-slotValue,hopIndex,'-o');
```



The preceding plot displays the selected channel index per transmission or reception slot using AFH. To minimize the packet and bit errors in the wireless channel, AFH selects only used channels for transmission or reception of Bluetooth BR/EDR waveforms

Compute the PER and BER of the Bluetooth BR/EDR waveforms with AFH.

```
% BER per channel calculation
errorsAdaptive(:,5) = errorsAdaptive(:,2)./errorsAdaptive(:,4);
errorsAdaptive(:,5) = fillmissing(errorsAdaptive(:,5),'constant',0);
ber = nonzeros(errorsAdaptive(:,5));
if ~isempty(ber)
    berAdaptive = mean(ber);
end

% PER calculation
packetErrorsAdaptive = sum(errorsAdaptive(:,3));
perAdaptive = packetErrorsAdaptive/totalTransmittedPackets;
fprintf('PER of Bluetooth BR/EDR waveforms using adaptive frequency hopping: %.4f\n',perAdaptive)

PER of Bluetooth BR/EDR waveforms using adaptive frequency hopping: 0.0569

fprintf('BER of Bluetooth BR/EDR waveforms using adaptive frequency hopping: %.4f\n',berAdaptive)

BER of Bluetooth BR/EDR waveforms using adaptive frequency hopping: 0.0019

% Restore previous setting of random number generation
rng(sprev);
```

The PER and BER values of the Bluetooth BR/EDR waveforms are less with AFH as compared with basic frequency hopping.

This example simulates an end-to-end transmitter-receiver chain to study how AFH mitigates interference between the Bluetooth BR/EDR and WLAN signals. The simulation results verify that the PER and the BER of the Bluetooth BR/EDR waveforms with WLAN interference is less with AFH as compared to basic frequency hopping.

Appendix

The example uses these helpers:

- helperBluetoothPHY: Configure and simulate Bluetooth PHY
- helperBluetoothChannel: Configure and simulate wireless channel
- helperBluetoothGenerateWLANWaveform: Generate WLAN waveform to be added as an interference to Bluetooth waveforms
- helperBluetoothPacketDuration: Calculate duration of Bluetooth packet

Selected Bibliography

1 Bluetooth® Technology Website. "Bluetooth Technology Website | The Official Website of Bluetooth Technology." Accessed November 29, 2021. https://www.bluetooth.com/.

2 Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification" Version 5.3, Accessed November 29, 2021. https://www.bluetooth.com/specifications/specs/.

See Also

Functions

`bluetoothFrequencyHop` | `bluetoothWaveformGenerator` | `bluetoothIdealReceiver`

More About

- “End-to-End Bluetooth BR/EDR PHY Simulation with AWGN, RF Impairments and Corrections” on page 5-13
- “End-to-End Bluetooth BR/EDR PHY Simulation with Path Loss, RF Impairments, and AWGN” on page 5-35
- “Generate and Attenuate Bluetooth BR/EDR Waveform in Industrial Environment” on page 9-31

End-to-End Bluetooth BR/EDR PHY Simulation with Path Loss, RF Impairments, and AWGN

This example uses Bluetooth® Toolbox to perform end-to-end simulation for the Bluetooth basic rate/enhanced data rate (BR/EDR) physical layer (PHY) transmission modes in the presence of the path loss model, radio front-end (RF) impairments, and additive white Gaussian noise (AWGN). The simulation results show the estimated value of the bit error rate (BER), path loss, and the impact of path loss on the spectrum of the waveform.

Path Loss Modeling in Bluetooth BR/EDR Network

Bluetooth is a short-range Wireless Personal Area Network (WPAN) technology, operating in the globally unlicensed industrial, scientific, and medical (ISM) band in the frequency range of 2.4 GHz to 2.485 GHz.

The Bluetooth Core Specification [1 on page 5-0] specifies these PHY modes.

Basic rate (BR) - Mandatory mode, uses Gaussian frequency shift keying (GFSK) modulation with a data rate of 1 Mbps.

Enhanced data rate (EDR) - Optional mode, uses phase shift keying (PSK) modulation with these two variants:

- **EDR2M:** Uses pi/4-DQPSK with a data rate of 2 Mbps
- **EDR3M:** Uses 8-DPSK with a data rate of 3 Mbps

For more information about Bluetooth BR/EDR protocol stack, see “Bluetooth Protocol Stack”.

For more information about Bluetooth BR/EDR packet structures, see “Bluetooth Packet Structure”.

Path loss or path attenuation is the decline in the power density of a given signal as it propagates from the transmitter to receiver through space. This reduction in power density occurs naturally over the distance and is impacted by the obstacles present in the environment in which the signal is being transmitted. The path loss is generally expressed in decibels (dB) and is calculated as:

$\mathrm{PL}_{\mathrm{dB}} = P_t - P_r$.

In this equation,

- $\mathrm{PL}_{\mathrm{dB}}$ is the path loss in dB.
- P_t is the transmitted signal power in dB.
- P_r is the received signal power in dB.

Path loss models describe the signal attenuation between the transmitter and receiver based on the propagation distance and other parameters such as frequency, wavelength, path loss exponent, and antenna gains. The example considers these path loss models:

- Free-space [3] on page 5-0
- Log-normal shadowing [3] on page 5-0
- Two-ray ground reflection [3] on page 5-0
- NIST PAP 02-Task 6 [4] on page 5-0

Bluetooth networks are operated in different environments such as home, office, industrial, and outdoor. A specific path loss model is used for each environment.

Environment	Path loss model
Outdoor	Two-Ray Ground Reflection
Industrial	Log-Normal Shadowing
Home	NIST PAP02-Task 6
Office	NIST PAP02-Task 6

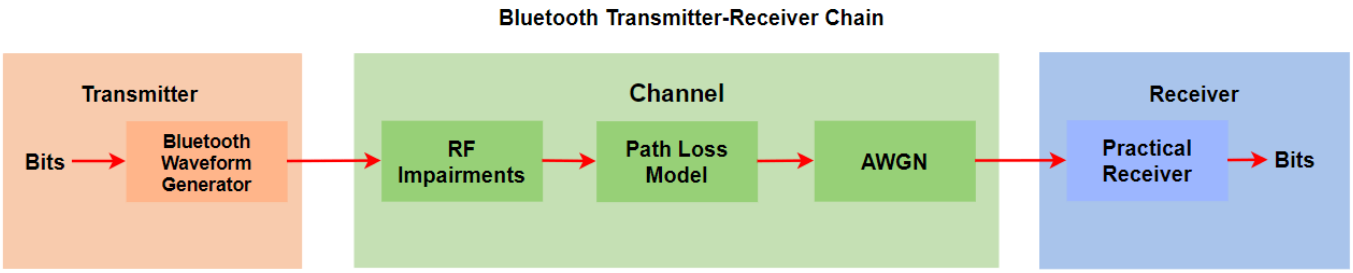
For more information about these path loss models, see “End-to-End Bluetooth LE PHY Simulation Using Path Loss Model, RF Impairments, and AWGN” on page 5-2 example.

This example estimates the BER and the path loss between the transmitter and receiver by considering a specific path loss model with RF impairments and AWGN added to the transmission packets.

End-to-End Bluetooth BR/EDR Simulation Procedure

To perform the end-to-end simulation in the presence of path loss, implement these steps.

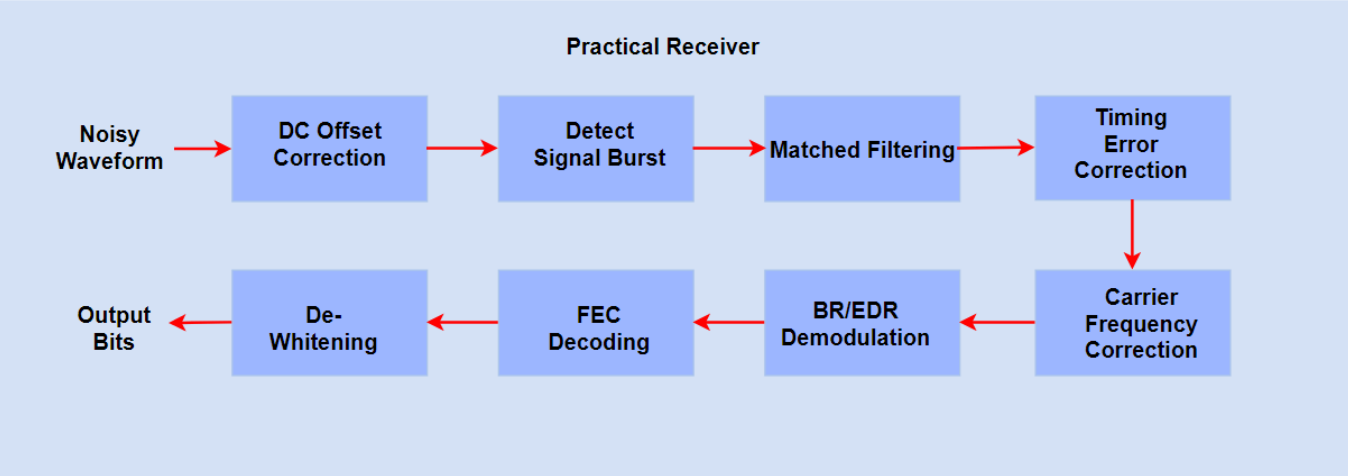
1. Generate random bits
2. Generate a Bluetooth BR/EDR waveform
3. Add impairments
4. Attenuate the waveform based on the path loss
5. Add AWGN
6. Display the spectrum of the transmitted and received waveforms



Pass the distorted and noisy waveforms through a practical receiver and perform these operations.

1. Remove DC offset
2. Detect the signal bursts
3. Perform matched filtering
4. Estimate and correct the timing offset

5 Estimate and correct the carrier frequency offset
6 Demodulate BR/EDR waveform
7 Perform forward error correction (FEC) decoding
8 Perform data dewhitening
9 Perform header error check (HEC) and cyclic redundancy check (CRC)
10 Outputs decoded bits



To estimate the bit error rate, compare the transmitted bits with the decoded bits.

Configure Simulation Parameters

Specify the path loss environment and the distance between the transmitter and receiver. Set the PHY transmission mode and the type of Bluetooth BR/EDR packet to be generated. Configure the RF impairments.

Configure parameters related to the communication link between the transmitter and receiver

```
environment = Outdoor;                  % Environment
distance = 5;                           % Distance between transmitter and receiver, in m
EbNo = 25;                              % Eb/No in dB
```

Configure parameters for waveform generation

```
phyMode = BR;                           % PHY transmission mode
bluetoothPacket = 'FHS';                % Type of Bluetooth BR/EDR packet. This value can
                                        % 'NULL','POLL','FHS','HV1','HV2','HV3','DV','EV3
                                        % 'EV4','EV5','AUX1','DM3','DM1','DH1','DM5','DH3
                                        % 'DH5','2-DH1','2-DH3','2-DH5','2-DH1','2-DH3',
                                        %  '2-DH5','2-EV3','2-EV5','3-EV3','3-EV5'}
sps = 8;                                % Samples per symbol
```

Configure RF impairments

```
frequencyOffset = 6000;% In Hz
timingOffset = 0.5;    % Offset within the symbol, in samples
timingDrift = 2;       % In parts per million
dcOffset = 2;          % Percentage with respect to maximum amplitude
```

Generate Bluetooth BR/EDR Waveform

Generate Bluetooth BR/EDR waveform based on the physical layer transmission mode, packet type, and samples per symbol.

```
% Create a Bluetooth BR/EDR waveform configuration object
txCfg = bluetoothWaveformConfig('Mode',phyMode,'PacketType',bluetoothPacket,...
    'SamplesPerSymbol',sps);
if strcmp(bluetoothPacket,'DM1')
    txCfg.PayloadLength = 17;       % Maximum length of DM1 packets in bytes
end
dataLen = getPayloadLength(txCfg);  % Length of the payload

% Generate Bluetooth BR/EDR waveform
bitsPerByte = 8;                              % Number of bits per byte
txBits = randi([0 1],dataLen*bitsPerByte,1); % Generate data bits
txWaveform = bluetoothWaveformGenerator(txBits,txCfg);
```

Add RF Impairments, Path Loss, and AWGN

Distort the generated Bluetooth BR/EDR waveform by adding the RF impairments.

```
% Create timing offset object
timingDelayObj = dsp.VariableFractionalDelay;

% Create frequency offset object
symbolRate = 1e6; % Symbol rate, in Hz
frequencyDelay = comm.PhaseFrequencyOffset('SampleRate',symbolRate*sps);

% Add Frequency Offset
frequencyDelay.FrequencyOffset = frequencyOffset;
txWaveformCFO = frequencyDelay(txWaveform);

% Add Timing Delay
packetDuration = bluetoothPacketDuration(phyMode,bluetoothPacket,dataLen);
filterSpan = 8*any(strcmp(phyMode,{'EDR2M','EDR3M'})); % To meet the spectral mask requirements
packetDurationSpan = packetDuration+filterSpan;
totalTimingDrift = zeros(length(txWaveform),1);
timingDriftRate = (timingDrift*1e-6)/(packetDurationSpan*sps);        % Timing drift rate
timingDriftVal = timingDriftRate*(0:1:((packetDurationSpan*sps)-1))'; % Timing drift
totalTimingDrift(1:(packetDurationSpan*sps)) = timingDriftVal;
timingDelay = (timingOffset*sps)+totalTimingDrift;                % Static timing offset and tim
txWaveformTimingCFO = timingDelayObj(txWaveformCFO,timingDelay);

% Add DC Offset
dcValue = (dcOffset/100)*max(txWaveformTimingCFO);
txImpairedWaveform = txWaveformTimingCFO + dcValue;
```

To obtain the path loss value, use helperBluetoothEstimatePathLoss.m function. To attenuate the Bluetooth BR/EDR waveform, add the path loss value to it.

```
% Obtain the path loss value in dB
[plLinear,pldB] = helperBluetoothEstimatePathLoss(environment,distance);

% Attenuate Bluetooth BR/EDR waveform
txAttenWaveform  = txImpairedWaveform./plLinear;
```

Add AWGN to the attenuated Bluetooth BR/EDR waveform.

```
% Set code rate based on packet type
if any(strcmp(bluetoothPacket,{'FHS','DM1','DM3','DM5','HV2','DV','EV4'}))
    codeRate = 2/3;
elseif strcmp(bluetoothPacket,'HV1')
    codeRate = 1/3;
else
    codeRate = 1;
end

% Set number of bits per symbol based on the PHY transmission mode
bitsPerSymbol = 1+ (strcmp(phyMode,'EDR2M'))*1 + (strcmp(phyMode,'EDR3M'))*2;

% Get SNR from EbNo values
snr = EbNo + 10*log10(codeRate) + 10*log10(bitsPerSymbol) - 10*log10(sps);

% Add AWGN
rxWaveform = awgn(txAttenWaveform,snr,'measured');
```

Receiver Processing

To retrieve the data bits, pass the attenuated, AWGN-distorted Bluetooth BR/EDR waveform through the practical receiver.

```
% Get PHY configuration properties
rxCfg = getPhyConfigProperties(txCfg);

% Receiver Module
rxBits = helperBluetoothPracticalReceiver(rxWaveform,rxCfg);
```

Simulation Results

Estimate BER based on the retrieved and transmitted bits.

```
% Calculate BER
if ((~any(strcmp(bluetoothPacket,{'ID','NULL','POLL'})))&&((length(txBits) == length(rxBits))))
    ber = (sum(xor(txBits,rxBits))/length(txBits));
    berDisplay = num2str(ber);
else % BER is not applicable either when packet is lost or when packet type is ID, NULL, POLL pac
    berDisplay = 'Not applicable';
end
```

Display Results

Display the estimated BER results. Plot the spectrum of the transmitted and received Bluetooth BR/EDR waveform.

```
% Display the estimated BER and distance between the transmitter and the receiver.
disp(['Input configuration: ', newline , '    PHY transmission mode: ', phyMode,....
```

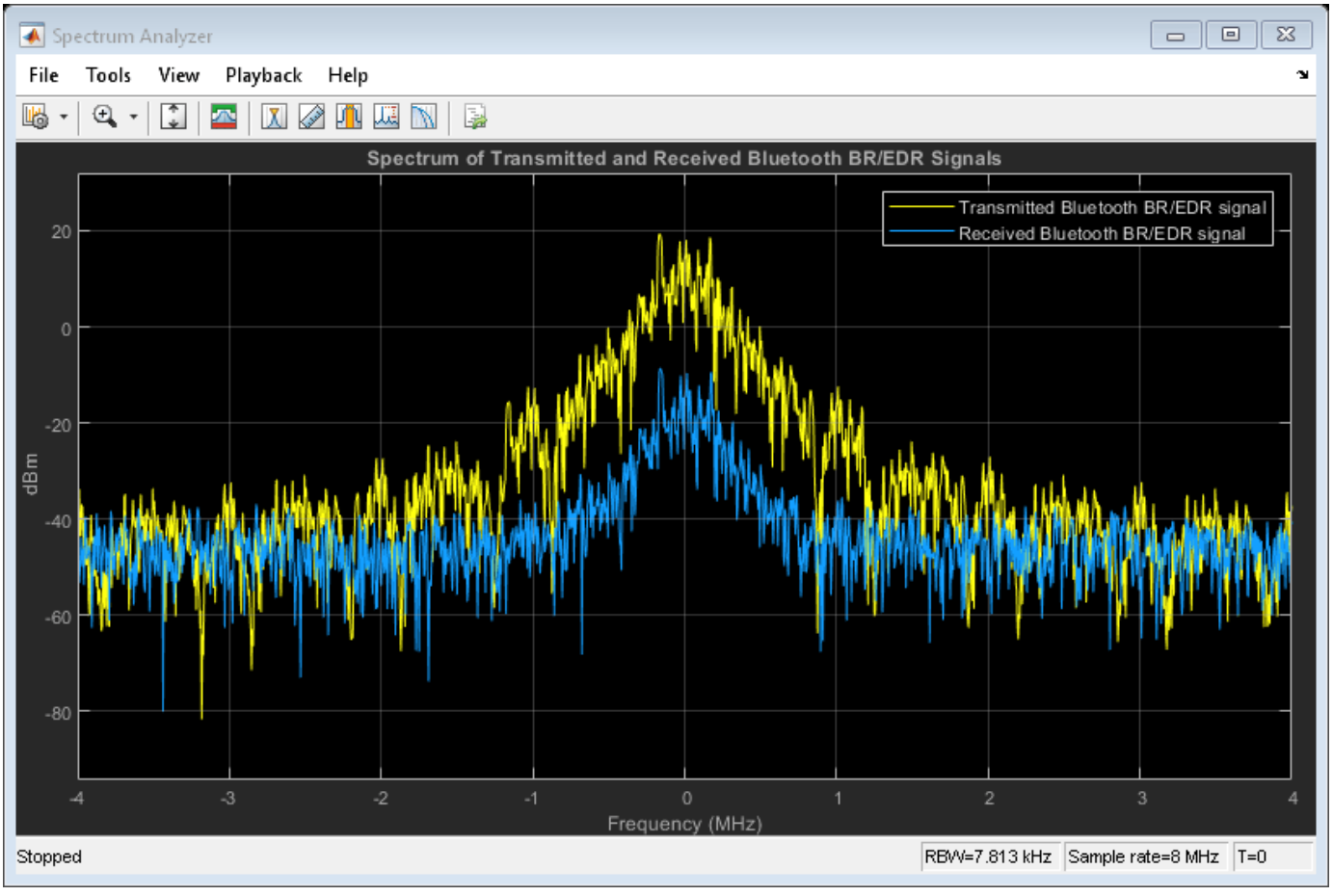
```
    newline,'    Environment: ', environment, newline ,...
    '    Distance between the transmitter and receiver: ', num2str(distance), ' m', newline ,...
    '    Eb/No: ', num2str(EbNo), ' dB']);
```

```
Input configuration:
    PHY transmission mode: BR
    Environment: Outdoor
    Distance between the transmitter and receiver: 5 m
    Eb/No: 25 dB
```

```
disp(['Estimated outputs: ', newline , '    Path loss : ', num2str(pldB), ' dB'....
    newline, '    BER: ', berDisplay]);
```

```
Estimated outputs:
    Path loss : 27.9588 dB
    BER: 0
```

```
% Plot the spectrum of the transmitted and received Bluetooth BR/EDR waveform
specAnalyzer = dsp.SpectrumAnalyzer('NumInputPorts',2,'SampleRate',symbolRate*sps,...
    'Title','Spectrum of Transmitted and Received Bluetooth BR/EDR Signals',...
    'ShowLegend',true,'ChannelNames',{'Transmitted Bluetooth BR/EDR signal','Received Bluetooth
specAnalyzer(txWaveform(1:packetDurationSpan*sps),rxWaveform(1:packetDurationSpan*sps));
release(specAnalyzer);
```



This example demonstrates an end-to-end Bluetooth BR/EDR simulation by considering the path loss model, distance between transmitter and receiver, RF impairments, and AWGN. The obtained

simulation results display the estimated path loss and BER. The spectrum of the transmitted and received Bluetooth BR/EDR waveforms is visualized by using a spectrum analyzer.

Appendix

The example uses these helper functions:

- helperBluetoothPracticalReceiver.m: Detects, synchronizes, and decodes the received Bluetooth BR/EDR waveform.
- helperBluetoothEstimatePathLoss.m: Estimates the path loss between the transmitter and receiver based on the path loss model and the distance between the transmitter and receiver.

Selected Bibliography

[1] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". Version 5.3. https://www.bluetooth.com.

[2] *Path Loss Models Used in Bluetooth Range Estimator*. Bluetooth Special Interest Group (SIG). https://www.bluetooth.com.

[3] Rappaport, Theodore. *Wireless Communication – Principles and Practice*. Prentice Hall, 1996.

[4] *NIST Smart Grid Interoperability Panel Priority Action Plan 2: Guidelines for Assessing Wireless Standards for Smart Grid Applications*. National Institute of Standards and Technology, U.S. Department of Commerce, 2014, https://nvlpubs.nist.gov/.

See Also

Functions

`bluetoothFrequencyHop` | `bluetoothWaveformGenerator` | `bluetoothIdealReceiver`

More About

- “End-to-End Bluetooth BR/EDR PHY Simulation with WLAN Interference and Adaptive Frequency Hopping” on page 5-20
- “End-to-End Bluetooth BR/EDR PHY Simulation with AWGN, RF Impairments and Corrections” on page 5-13
- “Generate and Attenuate Bluetooth BR/EDR Waveform in Industrial Environment” on page 9-31

End-to-End Bluetooth LE PHY Simulation with AWGN, RF Impairments and Corrections

This example shows how to measure the bit error rate (BER) and packet error rate (PER) for different modes of Bluetooth Low Energy (LE) [3] physical layer (PHY) packet transmissions that have radio front-end (RF) impairments and additive white gaussian noise (AWGN) added to them by using the Bluetooth® Toolbox.

Introduction

Bluetooth Special Interest Group (SIG) introduced Bluetooth LE for low power short range communications. Bluetooth LE devices operate in the globally unlicensed industrial, scientific and medical (ISM) band in the frequency range 2.4 GHz to 2.485 GHz. Bluetooth LE specifies a channel spacing of 2 MHz, which results in 40 RF channels. The Bluetooth LE standard specifies the **Link** layer which includes both **PHY** and **MAC** layers. Bluetooth LE applications include image and video file transfers between mobile phones, home automation, and the Internet of Things (IoT).

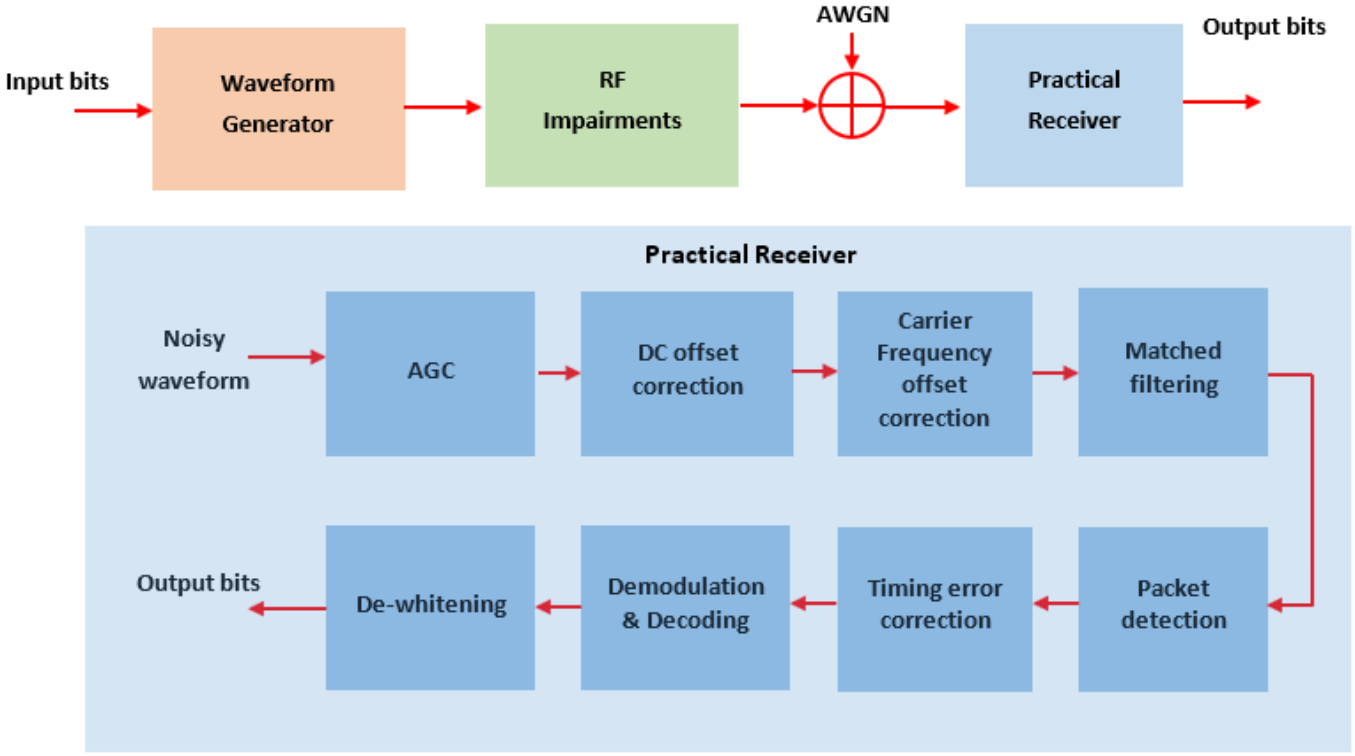
This end-to-end Bluetooth LE PHY simulation determines BER and PER performance of the four Bluetooth LE PHY transmission modes with RF impairments and AWGN added to the transmission packets. Nested for loops are used to compute error rates for each transmission mode at several bit energy to noise density ratio (Eb/No) settings. Inside the Eb/No loop, multiple transmission packets are generated using the `bleWaveformGenerator` function and altered with RF impairments and AWGN to accumulate the error rate statistics. Each packet is distorted by these RF impairments:

- DC offset
- Carrier frequency offset
- Carrier phase offset
- Timing drift

White gaussian noise is added to the transmitted Bluetooth LE waveforms. The noisy packets are processed through a practical Bluetooth LE receiver that performs the following operations:

- Automatic gain control (AGC)
- DC removal
- Carrier frequency offset correction
- Matched filtering
- Packet detection
- Timing error correction
- Demodulation and decoding
- Dewhitening

The processing steps for each packet are summarized in the following diagram:



The synchronized packets are then demodulated and decoded to recover the data bits. These recovered data bits are compared with the transmitted data bits to determine the BER and PER. BER and PER curves are generated for the following four PHY transmission throughput modes supported in Bluetooth LE:

- Uncoded PHY with data rate of 1 Mbps (LE1M)
- Uncoded PHY with data rate of 2 Mbps (LE2M)
- Coded PHY with data rate of 500 Kbps (LE500K)
- Coded PHY with data rate of 125 Kbps (LE125K)

Initialize the Simulation Parameters

```
EbNo = 2:4:10;                                  % Eb/No in dB
sps = 4;                                        % Samples per symbol, must be greater than 1
dataLen = 42;                                   % Data length in bytes, includes header, payload and
simMode = {'LE1M','LE2M','LE500K','LE125K'}; % PHY modes considered for the simulation
```

The number of packets tested at each Eb/No point is controlled by two parameters:

1. `maxNumErrors` is the maximum number of bit errors simulated at each Eb/No point. When the number of bit errors reaches this limit, the simulation at this Eb/No point is complete.
2. `maxNumPackets` is the maximum number of packets simulated at each Eb/No point and limits the length of the simulation if the bit error limit is not reached.

The numbers chosen for `maxNumErrors` and `maxNumPackets` in this example will lead to a very short simulation. For meaningful results we recommend increasing these numbers.

```
maxNumErrors = 100; % Maximum number of bit errors at an Eb/No point
maxNumPackets = 10; % Maximum number of packets at an Eb/No point
```

Simulating for Each Eb/No Point

This example also demonstrates how a `parfor` loop can be used instead of the `for` loop when simulating each Eb/No point to speed up a simulation. `parfor`, as part of the “Parallel Computing Toolbox”, executes processing for each Eb/No point in parallel to reduce the total simulation time. To enable the use of parallel computing for increased speed, comment out the 'for' statement and uncomment the 'parfor' statement below. If Parallel Computing Toolbox (TM) is not installed, 'parfor' will default to the normal 'for' statement.

```
numMode = numel(simMode);          % Number of modes
ber = zeros(numMode,length(EbNo)); % Pre-allocate to store BER results
per = zeros(numMode,length(EbNo)); % Pre-allocate to store PER results
bitsPerByte = 8;                   % Number of bits per byte

% Fixed access address Ideally, this access address value should meet the
% requirements specified in Section 2.1.2 of the Bluetooth specification.
accessAddress = [0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 ...
                           1 0 0 0 1 0 1 1 1 0 0 0 1]';
for iMode = 1:numMode

    phyMode = simMode{iMode};

    % Set signal to noise ratio (SNR) points based on mode.
    % For Coded PHYs (LE500K and LE125K), the code rate factor is included
    % in SNR calculation as 1/2 rate FEC encoder is used.
    if any(strcmp(phyMode,{'LE1M','LE2M'}))
        snrVec = EbNo - 10*log10(sps);
    else
        codeRate = 1/2;
        snrVec = EbNo + 10*log10(codeRate) - 10*log10(sps);
    end

%     parfor iSnr = 1:length(snrVec)  % Use 'parfor' to speed up the simulation
    for iSnr = 1:length(snrVec)   % Use 'for' to debug the simulation

        % Set random substream index per iteration to ensure that each
        % iteration uses a repeatable set of random numbers
        stream = RandStream('combRecursive','Seed',0);
        stream.Substream = iSnr;
        RandStream.setGlobalStream(stream);

        % Create an instance of error rate
        errorRate = comm.ErrorRate('Samples','Custom',...
                        'CustomSamples',1:(dataLen*bitsPerByte-1));

        % Create and configure the System objects for impairments
        initImp = helperBLEImpairmentsInit(phyMode,sps);

        % Configure the receiver parameters in a structure
        rxCfg = struct(Mode=phyMode,SamplesPerSymbol=sps,ChannelIndex=37,...
            DFPacketType='Disabled',AccessAddress=accessAddress);
        rxCfg.CoarseFreqCompensator = comm.CoarseFrequencyCompensator(Modulation="OQPSK",...
                                                    SampleRate=sps*(1+strcmp(phyMode,'LE2M'))*1e6,...
                                                    SamplesPerSymbol=2*sps,...
                                                    FrequencyResolution=100);
```

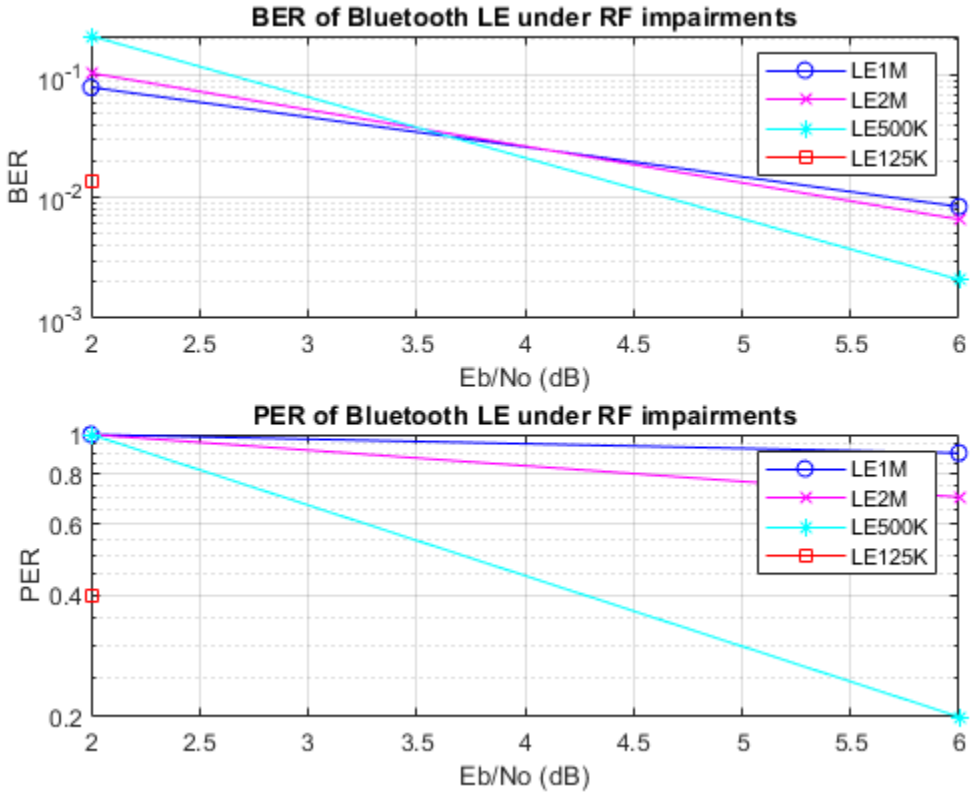
```
        rxCfg.PreambleDetector = comm.PreambleDetector(Detections="First");

        % Initialize error computation parameters
        [numErrs,perCnt] = deal(0);
        numPkt = 1;

        % Loop to simulate multiple packets
        while numErrs <= maxNumErrors && numPkt <= maxNumPackets

            % Generate Bluetooth LE waveform
            txBits = randi([0 1],dataLen*bitsPerByte,1,'int8'); % Data bits generation
            channelIndex = randi([0 39],1,1); % Random channel index value for each packet
            txWaveform = bleWaveformGenerator(txBits,'Mode',phyMode,...
                                              'SamplesPerSymbol',sps,...
                                              'ChannelIndex',channelIndex,...
                                              'accessAddress',accessAddress);

            % Define the RF impairment parameters
            initImp.pfo.FrequencyOffset = randsrc(1,1,-50e3:10:50e3); % In Hz, Max range is +/- 1
            initImp.pfo.PhaseOffset = randsrc(1,1,-10:5:10);          % In degrees
            initoff = 0.15*sps; % Static timing offset
            stepsize = 20*1e-6; % Timing drift in ppm, Max range is +/- 50 ppm
            initImp.vdelay = (initoff:stepsize:initoff+stepsize*(length(txWaveform)-1))'; % Varia
            initImp.dc = 20;     % Percentage w.r.t maximum amplitude value

            % Pass the generated waveform through RF impairments
            txImpairedWfm = helperBLEImpairmentsAddition(txWaveform,initImp);

            % Pass the transmitted waveform through AWGN channel
            rxWaveform = awgn(txImpairedWfm,snrVec(iSnr));

            % Recover data bits using practical receiver
            rxCfg.ChannelIndex = channelIndex;
            [rxBits,recAccessAddress] = helperBLEPracticalReceiver(rxWaveform,rxCfg);

            % Determine the BER and PER
            if(length(txBits) == length(rxBits))
                errors = errorRate(txBits,rxBits); % Outputs the accumulated errors
                ber(iMode,iSnr) = errors(1);       % Accumulated BER
                currentErrors = errors(2)-numErrs; % Number of errors in current packet
                if(currentErrors) % Check if current packet is in error or not
                    perCnt  = perCnt + 1;          % Increment the PER count
                end
                numErrs = errors(2);               % Accumulated errors
                numPkt = numPkt + 1;
            end
        end
        per(iMode,iSnr) = perCnt/(numPkt-1);

    disp(['Mode ' phyMode ', '...
        'Simulating for Eb/No = ', num2str(EbNo(iSnr)), ' dB' ', '...
        'BER:',num2str(ber(iMode,iSnr)), ', '...
        'PER:',num2str(per(iMode,iSnr))])
    end
end

Mode LE1M, Simulating for Eb/No = 2 dB, BER:0.080597, PER:1
Mode LE1M, Simulating for Eb/No = 6 dB, BER:0.0083582, PER:0.9
```

```
Mode LE1M, Simulating for Eb/No = 10 dB, BER:0, PER:0
Mode LE2M, Simulating for Eb/No = 2 dB, BER:0.10547, PER:1
Mode LE2M, Simulating for Eb/No = 6 dB, BER:0.0065672, PER:0.7
Mode LE2M, Simulating for Eb/No = 10 dB, BER:0, PER:0
Mode LE500K, Simulating for Eb/No = 2 dB, BER:0.21343, PER:1
Mode LE500K, Simulating for Eb/No = 6 dB, BER:0.0020896, PER:0.2
Mode LE500K, Simulating for Eb/No = 10 dB, BER:0, PER:0
Mode LE125K, Simulating for Eb/No = 2 dB, BER:0.013731, PER:0.4
Mode LE125K, Simulating for Eb/No = 6 dB, BER:0, PER:0
Mode LE125K, Simulating for Eb/No = 10 dB, BER:0, PER:0
```

Simulation Results

This section presents the BER and PER results w.r.t the input Eb/No range for the considered PHY modes

```
markers = 'ox*s';
color = 'bmcr';
dataStr = {zeros(numMode,1)};
for iMode = 1:numMode
    subplot(2,1,1),semilogy(EbNo,ber(iMode,:).',['-' markers(iMode) color(iMode)]);
    hold on;
    dataStr(iMode) = simMode(iMode);

    subplot(2,1,2),semilogy(EbNo,per(iMode,:).',['-' markers(iMode) color(iMode)]);
    hold on;
    dataStr(iMode) = simMode(iMode);
end
subplot(2,1,1),
grid on;
xlabel('Eb/No (dB)');
ylabel('BER');
legend(dataStr);
title('BER of Bluetooth LE under RF impairments');

subplot(2,1,2),
grid on;
xlabel('Eb/No (dB)');
ylabel('PER');
legend(dataStr);
title('PER of Bluetooth LE under RF impairments');
```



Reference Results

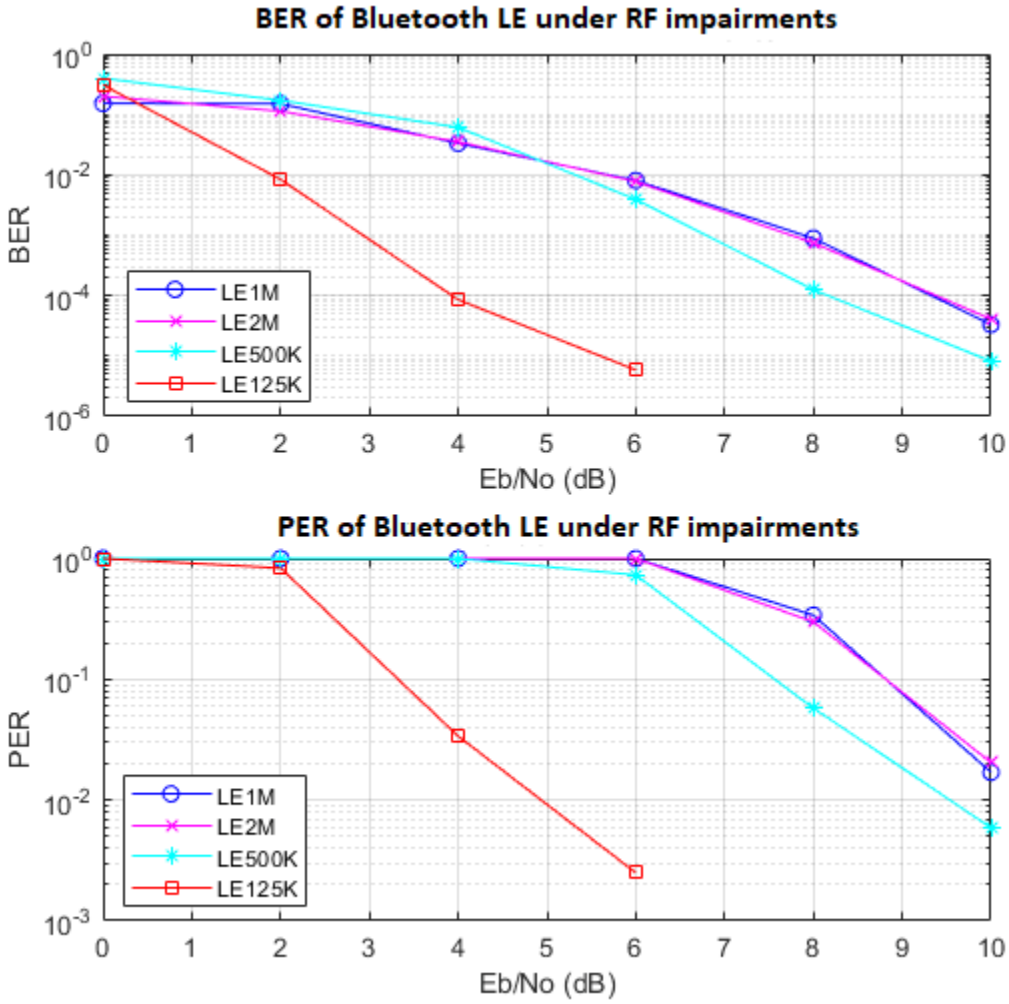
This section generates the reference BER, PER and Eb/No values for each PHY mode based on the receiver sensitivity and corresponding BER as specified in section 4.1 of the Bluetooth specification [3].

```
[refBER,refPER,refEbNo] = deal(zeros(numMode,1));
headerLen = 2; % Header length in bytes
crcLen = 3; % CRC length in bytes
payloadLen = dataLen-headerLen-crcLen; % Payload length in bytes
for iMode = 1:numMode
    [refBER(iMode),refPER(iMode),refEbNo(iMode)] = ...
                        helperBLEReferenceResults(simMode(iMode),payloadLen);
    disp(['Mode ' simMode{iMode} ', '...
        'Reference Eb/No = ', num2str(refEbNo(iMode)), ' dB' ', '...
        'BER = ',num2str(refBER(iMode)), ', '...
        'PER = ',num2str(refPER(iMode)), ', '...
        'for payload length of ',num2str(payloadLen), ' bytes'])
end
```

```
Mode LE1M, Reference Eb/No = 34.919 dB, BER = 0.001, PER = 0.30801, for payload length of 37 byte
Mode LE2M, Reference Eb/No = 34.919 dB, BER = 0.001, PER = 0.30801, for payload length of 37 byte
Mode LE500K, Reference Eb/No = 31.9087 dB, BER = 0.001, PER = 0.30801, for payload length of 37 b
Mode LE125K, Reference Eb/No = 31.9087 dB, BER = 0.001, PER = 0.30801, for payload length of 37 b
```

Further Exploration

The number of packets tested at each Eb/No value is controlled by `maxNumErrors` and `maxNumPackets` parameters. For statistically meaningful results these values should be larger than those presented in this example. To generate the figure below, the simulation ran using a data length of 128 bytes, `maxNumErrors = 1e3`, and `maxNumPackets = 1e4` for all the four transmission modes.



The figure shows that the reference BER and PER can be obtained at lower Eb/No points compared to the reference Eb/No value given in the Bluetooth specification. In this example, only the following impairments are added and passed through AWGN channel.

- DC offset
- Carrier frequency offset
- Carrier phase offset
- Timing drift

The reference Eb/No values generated based on the Bluetooth LE specification include margin for RF impairments and fading channel conditions that are not modeled in this simulation. As a result, the simulation results here outperform the standard reference results. If you modify this example to include additional impairments such as frequency drift, fading, and interference in the simulation the BER and PER curves will move right towards the reference Eb/No values generated based on the Bluetooth LE standard receiver characteristics in [3], Volume 6, Section 4.1.

Appendix

The example uses these helpers:

- <matlab:openExample('bluetooth/BLEPracticalReceiverExample','supportingFile','helperBLEImpairmentsAddition.m') helperBLEImpairmentsAddition >: Add RF impairments to the Bluetooth LE waveform
- <matlab:openExample('bluetooth/BLEPracticalReceiverExample','supportingFile','helperBLEPracticalReceiver.m') helperBLEPracticalReceiver >: Demodulate and decodes the received signal
- <matlab:openExample('bluetooth/BLEPracticalReceiverExample','supportingFile','helperBLEImpairmentsInit.m') helperBLEImpairmentsInit >: Initialize RF impairment parameters
- <matlab:openExample('bluetooth/BLEPracticalReceiverExample','supportingFile','helperBLEReferenceResults.m') helperBLEReferenceResults >: Generate reference BER, PER and Eb/No values

Summary

This example simulates a Bluetooth LE PHY packet transmissions that have RF impairments and AWGN added to them. It shows how to generate Bluetooth LE waveforms, demodulate and decode data bits using practical receiver and compute the BER and PER.

Selected Bibliography

1. Bluetooth Technology Website | The Official Website of Bluetooth Technology, Accessed November 22, 2021. https://www.bluetooth.com.
2. Volume 6 of the Bluetooth Core Specification, Version 5.3 Core System Package [Low Energy Controller Volume].

See Also

Functions

`bleWaveformGenerator` | `bleIdealReceiver` | `bleChannelSelection`

More About

- “End-to-End Bluetooth LE PHY Simulation Using Path Loss Model, RF Impairments, and AWGN” on page 5-2
- “Bluetooth LE Bit Error Rate Simulation with AWGN” on page 1-19
- “Generate Bluetooth LE Waveform and Add RF Impairments” on page 9-25

Multinode Communication

Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks

This example shows you how to perform energy profiling of nodes in a Bluetooth mesh network by using Bluetooth® Toolbox. Using this example, you can:

- Create and configure a Bluetooth mesh network.
- Compute energy consumption of mesh nodes in transmission, listen, sleep, and idle states by varying the number of Relay nodes, source-destination node pairs, Friend node-Low Power node (LPN) pair, and the application traffic.
- Estimate the lifetime of mesh node based on the configured hardware-specific energy parameters.
- Explore the impact of poll timeout and receive window size on the LPN lifetime.

The results confirm the expectation that LPN always consume less energy by spending more time in sleep, resulting in energy conservation and increased lifetime.

Bluetooth Mesh Energy Profiling

The Bluetooth Core Specification [2 on page 6-0] includes a Low Energy (LE) version for wireless personal area networks, referred to as Bluetooth LE or Bluetooth Smart. The Bluetooth mesh profile [3 on page 6-0] defines the fundamental requirements to implement a mesh networking solution for Bluetooth LE. Bluetooth mesh networking enables large-scale device networks in the applications such as smart lighting, industrial automation, sensor networking, and asset tracking. For information about Bluetooth LE protocol stack, see “Bluetooth Protocol Stack” on page 8-9.

Each Bluetooth mesh node possess some optional features enabling them to acquire additional capabilities. These features include the Relay, Proxy, Friend, and the LowPower features. The Bluetooth mesh nodes possessing these features are known as Relay nodes, Proxy nodes, Friend nodes, and LPNs, respectively. To reduce the duty cycles of the LPN and conserve energy, the LPN must establish a *Friendship* with a Friend node (mesh nodes supporting the Friend feature). This *Friendship* between the LPN and the Friend node enables the Friend node to store and forward messages addressed to the LPN. Forwarding by the Friend node occurs only when the LPN wakes up and polls the Friend node for messages awaiting delivery. This mechanism enables all of the LPNs to conserve energy and operate for longer durations. For more information about devices, nodes, and the Friendship in Bluetooth mesh network, see “Bluetooth Mesh Networking” on page 8-64.

In this example, the source nodes initiates mesh communication to a destination node and an LPN. During the simulation, the Friend node and LPN exchange Friendship messages. Each node computes the time spent in various states (transmission, listen, idle and sleep) and calculates its lifetime.

Configure Simulation Parameters

Specify the number of nodes (`numNodes`) along with their positions and simulation time. Set the seed for the random number generator to 1. The seed value controls the pattern of random number generation. For high fidelity simulation results, change the seed value for each run and average the results over multiple simulations.

```
% Set random number generator as "twister"
rng(1,"twister");

% Specify the simulation time in seconds
simulationTime = 5;
```

```
% Total number of nodes in the mesh network
numNodes = 21;
```

Get node positions from the MAT file. Specify the positions of Bluetooth mesh nodes as a `numNodes`-by-2 array, where `numNodes` is the number of nodes in the network. Each row specifies the Cartesian coordinates of a node, starting from the first node.

```
load("bleMeshNetworkNodePositions.mat");
```

Set some of the nodes as relay nodes, source-destination node pairs, friend node and LPN.

```
relayNodes = [3 6 7 8 9 12 13 14 17];
sourceDestinationPairs = [1 20; 21 16];
friend = 15;
lpn = 16;
```

Create Bluetooth Mesh Nodes

Use `bluetoothMeshProfileConfig` to create mesh profile configuration object. To create a Bluetooth mesh node, use the `bluetoothLENode` object. Specify the role as "`broadcaster-observer`" and assign the mesh profile to `MeshConfig`.

```
% Initialize array to store Bluetooth mesh nodes
meshNodes = cell(1,numNodes);
clear wirelessNode; % Clear context of wirelessNode

% Create Bluetooth mesh network
for nodeIndex = 1:numNodes
    % Create and configure Bluetooth mesh profile by specifying the element
    % address (unique to each node in the network). Set relay and network
    % message repetitions.
    meshCfg = bluetoothMeshProfileConfig(ElementAddress=dec2hex(nodeIndex,4),...
        NetworkTransmissions=3,RelayRetransmissions=3);

    % For the configured relayNodes, enable Relay feature
    if any(nodeIndex==relayNodes)
        meshCfg.Relay = true;
    end
    % For the configured Friend node, enable Friend feature
    if nodeIndex==friend
        meshCfg.Friend = true;
    % For the configured LPN, enable LowPower feature
    elseif nodeIndex==lpn
        meshCfg.LowPower = true;
    end

    % Create and configure Bluetooth mesh node by assigning the mesh profile.
    % Set receiver range, advertising interval (seconds) and scan interval (seconds).
    meshNode = bluetoothLENode("broadcaster-observer",MeshConfig=meshCfg, ...
        Position=[bleMeshNetworkNodePositions(nodeIndex,:) 0],Name="Node"+num2str(nodeIndex),...
        ReceiverRange=25,AdvertisingInterval=20e-3,ScanInterval=30e-3);

    % Store mesh nodes
    meshNodes{nodeIndex} = meshNode;
end
```

Configure Friendship Between Friend Node and LPN

Set friendship timing parameters (in seconds) such as poll timeout, receive window, and receive delay by using the `bluetoothMeshFriendshipConfig` object.

```
friendshipConfig = bluetoothMeshFriendshipConfig(PollTimeout=2,ReceiveWindow=180e-3,...
    ReceiveDelay=50e-3);
```

Configure the friendship between the friend node and LPN using `configureFriendship` method of `bluetoothMeshFriendshipConfig`.

```
friendNode = meshNodes{friend};
lowPowerNode = meshNodes{lpn};
configureFriendship(friendshipConfig,friendNode,lowPowerNode);
```

Add Application Traffic to Source Nodes

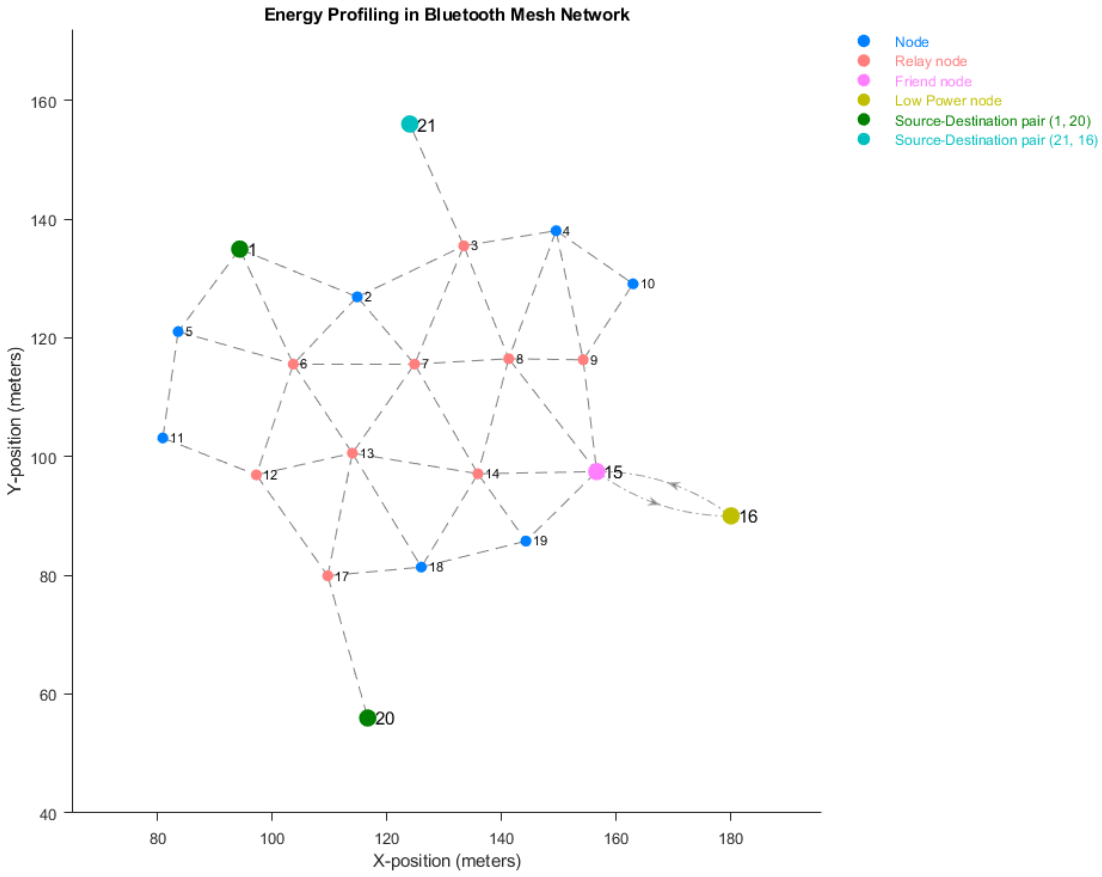
Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Configure the On-Off application traffic pattern by specifying the application data rate, packet size, on, and off state duration. Simulate mesh communication between the specified source-destination node pairs in the mesh network.

```
for srcIdx = 1:numel(sourceDestinationPairs)/2
    % Create network traffic object using networkTrafficOnOff. Set data
    % rate and packet size. Set on and off times based on the simulation time.
    traffic = networkTrafficOnOff(DataRate=1,PacketSize=15,GeneratePacket=true,...
        OnTime=simulationTime*0.3,OffTime=simulationTime*0.7);

    % You can control the maximum number of hops that the source node uses
    % to relay message by setting the time-to-live (TTL) value.
    ttl = 10;

    % Attach application traffic to source
    addTrafficSource(meshNodes{sourceDestinationPairs(srcIdx,1)},traffic,...
        SourceAddress=meshNodes{sourceDestinationPairs(srcIdx,1)}.MeshConfig.ElementAddress,...
        DestinationAddress=meshNodes{sourceDestinationPairs(srcIdx,2)}.MeshConfig.ElementAddress
        TTL=ttl);
end
```

Visualize Mesh Network

Visualize the mesh network scenario by using the helperBLEMeshVisualizeNetwork helper object. Assign the helper object to the `NetworkPlot` parameter in the helperBLEMeshEventCallback helper object. For information on how to visualize the message flow in the mesh network, see “Bluetooth Mesh Flooding in Wireless Sensor Networks” on page 6-10.

```
% Object to visualize mesh network
plotScenario = helperBLEMeshVisualizeNetwork(NumberOfNodes=numNodes,...
    NodePositionType="UserInput",Positions=bleMeshNetworkNodePositions,...
    VicinityRange=25,SimulationTime=simulationTime,...
    SourceDestinationPairs=sourceDestinationPairs,FriendPairs=[friend lpn],...
    Title="Energy Profiling in Bluetooth Mesh Network");

% Object to handle visualization callbacks
visualizeScenario = helperBLEMeshEventCallback(...
    NetworkPlot=plotScenario,Nodes=meshNodes);
init(visualizeScenario);
```



Run the simulation

Initialize the wireless network.

```
networkSimulator = helperWirelessNetwork(meshNodes);
```

Run all the nodes in the network for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

Update simulation progress at the end of the simulation and clear context of `wirelessNode`.

```
updateProgressBar(visualizeScenario.NetworkPlot,simulationTime);
```

Simulation Results

At each mesh node, the simulation captures these statistics.

- Application end-to-end packet latency in seconds
- Link layer (LL) throughput in Kbps
- Time spent in listen state, transmit state, idle state and sleep state in seconds

- Packet statistics at the application layer, network layer, transport layer, LL and physical layer

The workspace variable `statisticsAtEachNode` contains the cumulative value of the preceding statistics for all the nodes in the network. For a given simulation run, you can view the statistics for first five nodes. The network statistics for the first five nodes in the network are:

```
statisticsAtEachNode = helperBLEMeshStatistics(meshNodes);
statisticsForFirstFiveNodes = statisticsAtEachNode(1:min(numNodes,5),:)
```

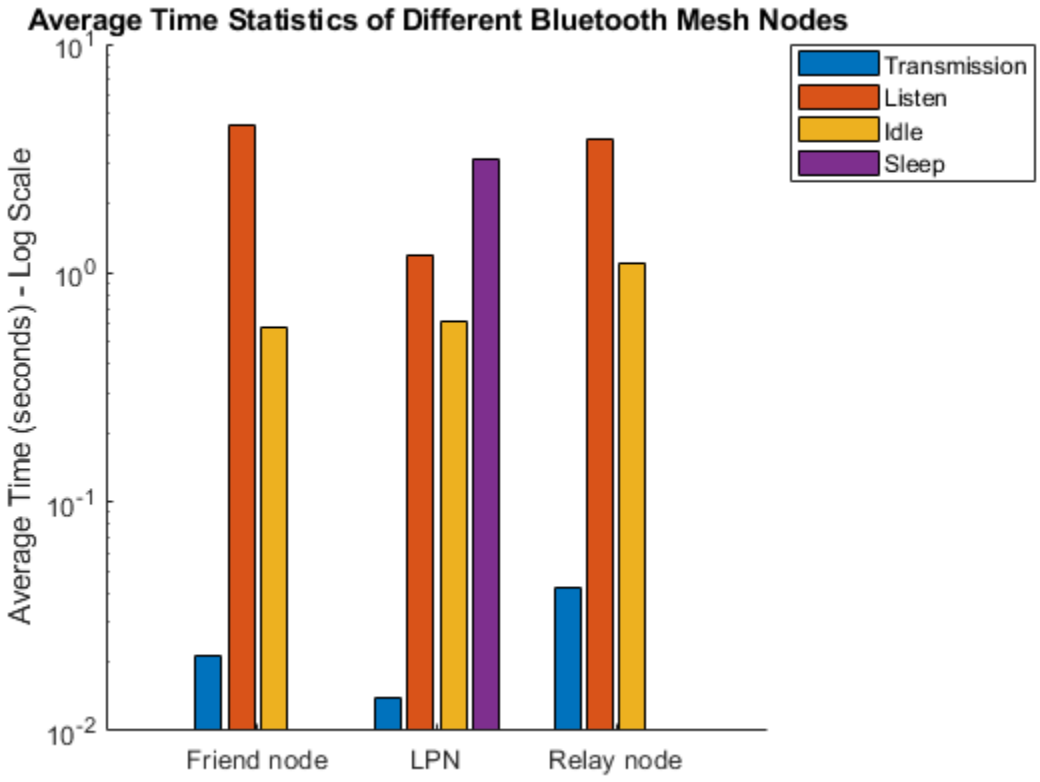
```
statisticsForFirstFiveNodes=5×22 table
            End-to-end packet Latency (seconds)    Throughput (Kbps)    PHY Tx Packets    PHY Rx
            ___________________________________    _________________    ______________    _____

    Node1                     0                         5.0554               117
    Node2                     0                              0                 0
    Node3                     0                          6.222               144
    Node4                     0                              0                 0
    Node5                     0                              0                 0
```

This plot shows the average time spent by different type of mesh nodes in different states. The results conclude that the LPN spend most of the time in sleep state, resulting in energy conservation and increased lifetime.

```
fprintf("Average time statistics of different Bluetooth mesh nodes are:\n");
```

```
Average time statistics of different Bluetooth mesh nodes are:
```

```
meshNodesAvgStats = helperBLEMeshNodeAverageTime(meshNodes);
```



Configure the traffic between the mesh nodes by using the `addTrafficSource` method of `bluetoothLENode`. The transmission time at the mesh node depends on the application traffic. The transmission time at the LPN depends on the poll timeout value.

Further Exploration

Calculate Lifetime of LPN:

Use helperBLEMeshNodeLifetime helper function to calculate the lifetime of a node in the Bluetooth mesh network at the end of simulation. To compute node lifetime, the simulation time and the mesh node is given as an input to the helperBLEMeshNodeLifetime helper function. The node lifetime is calculated by using the energy parameters that are hardware dependent. To update these hardware parameters, use the helperBLEMeshNodeLifetime helper function.

```
lifeTime = helperBLEMeshNodeLifetime(lowPowerNode,simulationTime);
```

```
Configured hardware parameters for a 1200 mAh battery are:
       Hardware parameters        Configured values (mA)
    ______________________________    ______________________

    Self-discharge                          0.0013699
    Transmission on channel 37                   7.57
    Transmission on channel 38                   7.77
    Transmission on channel 39                    7.7
    Listening                                    10.3
    Sleep                                         0.2
    Idle                                         1.19

Timing metrics at Node16 are:
     Time variables      Time (seconds)
    __________________    ______________

    Transmission time       0.016704
    Listen time                  1.2
    Sleep time                 3.1724
    Idle time                 0.60993
```

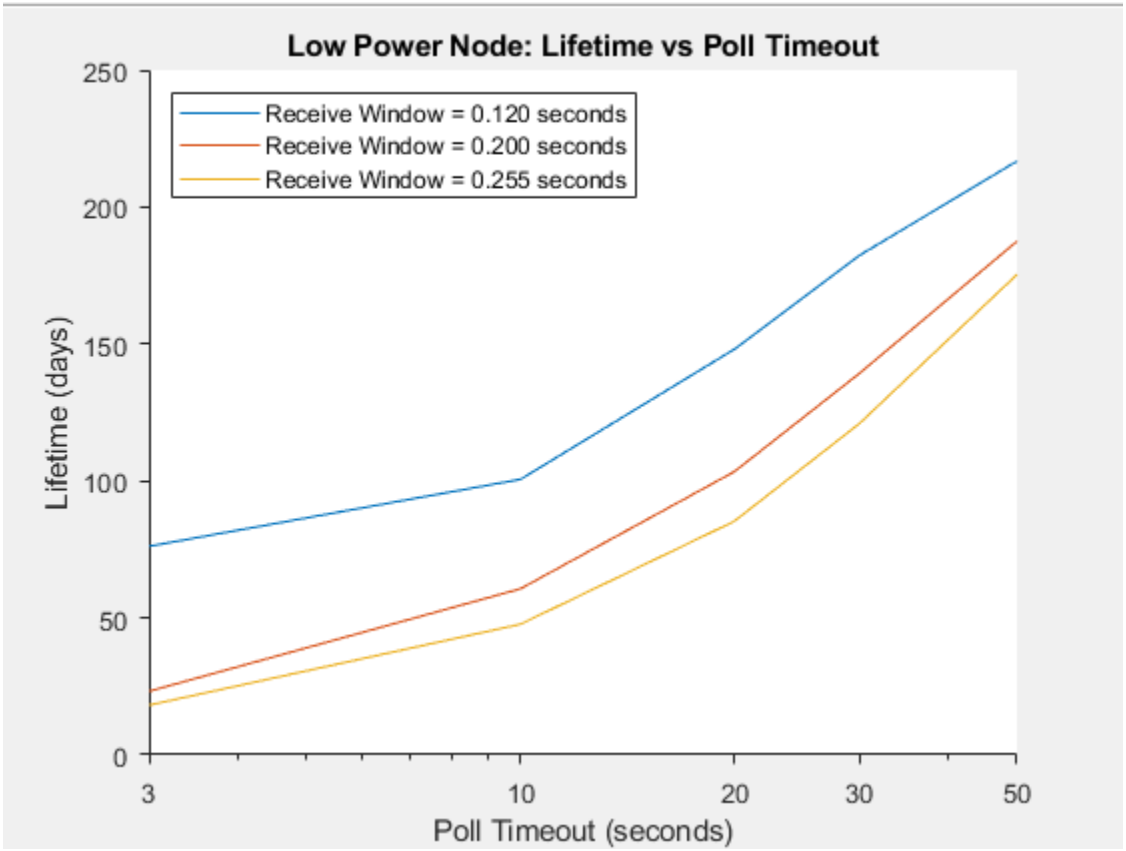
```
fprintf("Lifetime of %s is %.4f days.\n",lowPowerNode.Name,lifeTime);
```

```
Lifetime of Node16 is 18.0435 days.
```

Lifetime of LPN by Varying Poll Timeout

The lifetime of an LPN depends on the time for which the node is in the listen state. In a given poll timeout, an LPN is in listen or sleep state for most of the time. The receive window for each poll request of an LPN determines the time spent in listen state. The time spent in transmission state is negligible.

Visualize the impact of the poll timeout and receive window on the lifetime of LPN by using the helperBLEMeshLPNLifetimeVSPolltimeout helper function.



The preceding plot concludes that the lifetime of LPN is directly proportional to the poll timeout. The poll timeout specifies the maximum time between two consecutive requests from an LPN to Friend node. As the poll timeout increases, the LPN spends more time in sleep state that results in increasing the lifetime of the LPN.

This example shows how to create and configure a multinode Bluetooth mesh network and analyze the message exchange in the network. This example also enables you to analyze the behavior and the advantages of the Friendship between Friend node and LPN. To calculate the time spent by each node in different states, the Bluetooth mesh network is simulated with source-destination node pairs, and a Friend node-LPN pair. The plot of the average time spent by each node in different states shows that the LPN always consumes less energy by spending more time in sleep state. You can further explore the energy profiling of LPN by varying the poll timeout and receive window values.

Appendix

The example uses these helpers:

- helperWirelessNetwork: Simulate wireless network
- helperBLEMeshStatistics: Return statistics of each mesh node in the network
- helperBLEMeshEventCallback: Callback function to visualize message transmissions
- helperBLEMeshVisualizeNetwork: Bluetooth mesh network visualization
- helperBLEMeshNodeLifetime: Compute lifetime of a Bluetooth mesh node
- helperBLEMeshNodeAverageTime: Compute average time spent in various states by the Bluetooth mesh nodes

- helperBLEMeshLPNLifetimeVSPolltimeout: Compute the lifetime of Bluetooth mesh LPN for different poll timeout and receive window values

Selected Bibliography

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 25, 2021. https://www.bluetooth.com.
2. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". Version 5.3. https://www.bluetooth.com/.
3. Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Profile". Version 1.0.1. https://www.bluetooth.com/.

See Also

Functions

`configureFriendship`

Objects

`bluetoothLENode` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig`

More About

- “Bluetooth Mesh Networking” on page 8-64
- “Bluetooth Mesh Flooding in Wireless Sensor Networks” on page 6-10
- “Create, Configure and Simulate Bluetooth Mesh Network” on page 9-45
- “Create, Configure, and Visualize Bluetooth Mesh Network” on page 9-17
- “Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network” on page 9-49
- “Bluetooth LE Node Statistics” on page 8-78

Bluetooth Mesh Flooding in Wireless Sensor Networks

This example shows you how the managed flooding technique enables you to realize communication in a Bluetooth mesh network by using Bluetooth® Toolbox. Using this example, you can:

- Create and configure a Bluetooth mesh network.
- Visualize and analyze how managed flooding technique enables communication between the source and destination nodes, even after disabling some intermediate relay nodes.
- Visualize the message flow from the source to destination nodes.
- Analyze performance metrics such as network packet delivery ratio (PDR), end-to-end latency, throughput, and other node-related metrics.

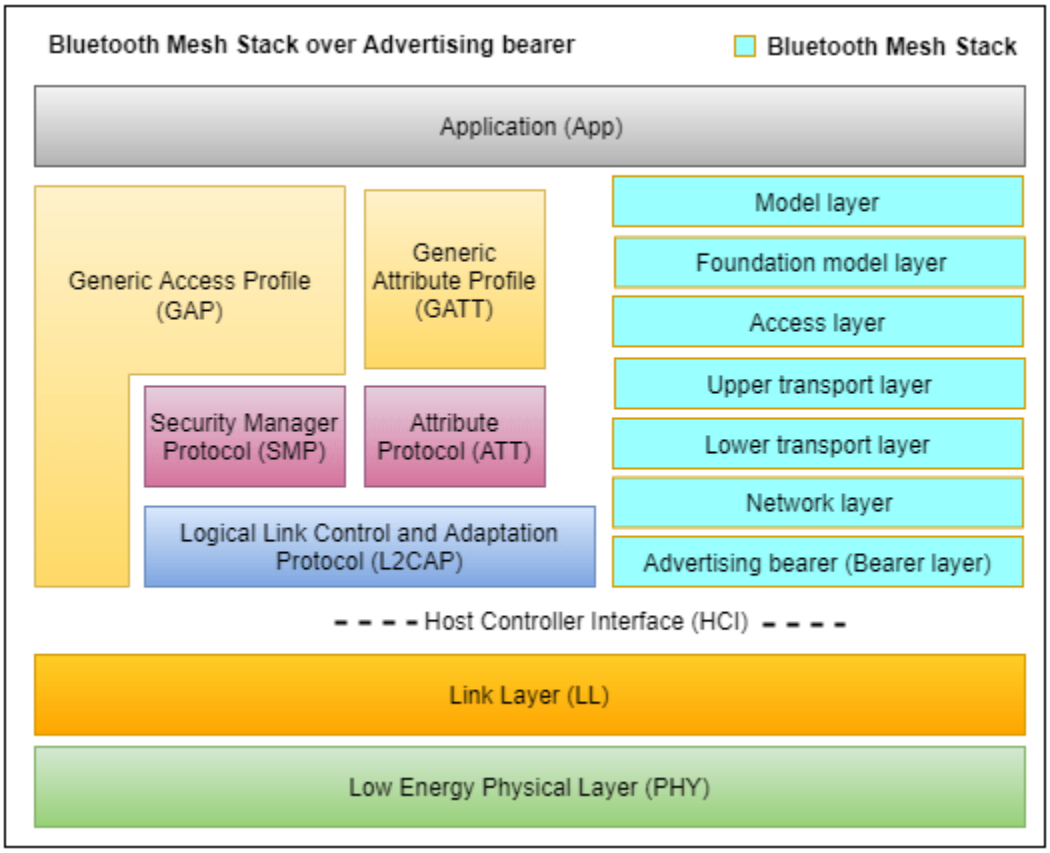
The example also shows you how to perform Monte Carlo simulations on the Bluetooth mesh network to identify critical relay nodes between the source and destination.

Bluetooth Mesh Stack

The Bluetooth Core Specification [2 on page 6-0] includes a Low Energy version for wireless personal area networks, referred to as Bluetooth Low Energy (LE) or Bluetooth Smart. Bluetooth LE was added to the standard for low energy devices generating small amounts of data, such as notification alerts used in applications such as home automation, health-care, fitness, and Internet of things (IoT). For more information about Bluetooth LE protocol stack, see “Bluetooth Protocol Stack” on page 8-9.

The Bluetooth Mesh Profile [3 on page 6-0] defines the fundamental requirements to implement a mesh networking solution for Bluetooth LE. Bluetooth mesh networking enables large-scale device networks in the applications such as smart lighting, industrial automation, sensor networking, and asset tracking.

This figure shows the Bluetooth mesh stack over the advertising bearer.



For more information about Bluetooth Mesh Profile, see “Bluetooth Mesh Networking” on page 8-64.

Bluetooth Mesh Network Flooding Scenarios

This example uses the advertising bearer to demonstrate managed flooding. This example enables you to create and configure two Bluetooth mesh network scenarios. Each scenario is a multinode mesh network. Each scenario classifies the mesh nodes as node, relay node, source node, and destination node.

- Node: Any device in the mesh network
- Relay node: A node that has the relay feature supported and enabled
- Source node: A node that originates and transmits packets
- Destination node: A node at which the packets from the source node are destined

The first mesh network scenario consists of 21 mesh nodes, including relay nodes, a source node, and a destination node. In the second scenario, the example:

- Disables the relay feature of some nodes
- Removes a node from the mesh network

For both the scenarios, you can visualize the message flow in the network and retrieve these statistics.

- Network PDR: Ratio of number of packets received at all the destination nodes to the number of packets transmitted by all the source nodes in the mesh network

- Application end-to-end packet latency in seconds
- Link layer (LL) throughput in Kbps
- Time spent in listen state, transmit state, idle state and sleep state in seconds
- Packet statistics at the application layer, network layer, transport layer, link layer and physical layer

Configure Simulation Parameters

Specify the number of nodes (`numNodes`) along with their positions and simulation time for both the scenarios. Set the seed for the random number generator to 1. The seed value controls the pattern of random number generation. For high fidelity simulation results, change the seed value for each run and average the results over multiple simulations.

```
% Set random number generator as "twister"
rng(1,"twister");

% Specify the simulation time for both scenarios in seconds
simulationTime = 0.3;

% To highlight message transmission in the mesh network visualization, set
% this flag to True.
highlightTransmissions = true;

% Number of nodes in the mesh network
numNodes = 21;
```

Get node positions from the MAT file. Specify the positions of Bluetooth mesh nodes as a `numNodes`-by-2 array, where `numNodes` is the number of nodes in the network. Each row specifies the Cartesian coordinates of a node, starting from the first node.

```
load("bleMeshNetworkNodePositions.mat");
```

Simulate Mesh Network Scenario One

The first mesh network scenario consists of 21 mesh nodes. Set some of the nodes as relay nodes and source-destination node pairs.

```
relayNodes = [3 6 7 8 9 12 13 14 15 17];
sourceDestinationPairs = [1 10];
```

Create Bluetooth Mesh Nodes

Use `bluetoothMeshProfileConfig` to create mesh profile configuration object. To create a Bluetooth mesh node, use the `bluetoothLENode` object. Specify the role as "`broadcaster-observer`" and assign the mesh profile to `MeshConfig`.

```
% Initialize array to store Bluetooth mesh nodes
nodesScenarioOne = cell(1,numNodes);
clear wirelessNode; % Clear context of wirelessNode

% Create Bluetooth mesh network for scenario one
for nodeIndex = 1:numNodes
    % Create and configure Bluetooth mesh profile by specifying the element
    % address (unique to each node in the network). Set network message
    % repetitions to 2, and network transmit interval as a random value in
```

```
    % the range [10, 30] milliseconds.
    meshCfg = bluetoothMeshProfileConfig(ElementAddress=dec2hex(nodeIndex,4),...
        NetworkTransmissions=2,NetworkTransmitInterval=randi([1 3])*10e-3);

    % For the configured relayNodes, enable Relay feature. Set relay
    % message repetitions to 3, and relay retransmit interval as a random
    % value in the range [10, 30] milliseconds.
    if any(nodeIndex==relayNodes)
        meshCfg.Relay = true;
        meshCfg.RelayRetransmissions = 3;
        meshCfg.RelayRetransmitInterval = randi([1 3])*10e-3;
    end

    % Create and configure Bluetooth mesh node by assigning the mesh profile.
    % Set receiver range, advertising interval (seconds) and scan interval (seconds).
    meshNode = bluetoothLENode("broadcaster-observer",MeshConfig=meshCfg,...
        Position=[bleMeshNetworkNodePositions(nodeIndex,:) 0],Name="Node"+num2str(nodeIndex),...
        ReceiverRange=25,AdvertisingInterval=20e-3,ScanInterval=30e-3);

    % Store mesh nodes
    nodesScenarioOne{nodeIndex} = meshNode;
end
```

Add Application Traffic to Source Nodes

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Configure the On-Off application traffic pattern by specifying the application data rate, packet size, on, and off state duration. Simulate mesh communication between specified source and destination pairs in the mesh network.

```
for srcIdx = 1:numel(sourceDestinationPairs)/2
    % Create network traffic object using networkTrafficOnOff. Set data rate
    % and packet size. Set on and off times based on the simulation time.
    traffic = networkTrafficOnOff(DataRate=1,PacketSize=15,GeneratePacket=true,...
        OnTime=simulationTime*0.3,OffTime=simulationTime*0.7);

    % You can control the maximum number of hops that the source node uses to
    % relay message by setting the time-to-live (TTL) value.
    ttl = 10;

    % Attach application traffic to source
    addTrafficSource(nodesScenarioOne{sourceDestinationPairs(srcIdx,1)},traffic, ...
        SourceAddress=nodesScenarioOne{sourceDestinationPairs(srcIdx,1)}.MeshConfig.ElementAddres
        DestinationAddress=nodesScenarioOne{sourceDestinationPairs(srcIdx,2)}.MeshConfig.ElementA
        TTL=ttl);
end
```

Visualize Mesh Network

Visualize the mesh network scenario by using the helperBLEMeshVisualizeNetwork helper object. Assign the helper object to the `NetworkPlot` parameter in the helperBLEMeshEventCallback helper object.

```
% Object to visualize mesh network
plotScenarioOne = helperBLEMeshVisualizeNetwork(NumberOfNodes=numNodes,...
    NodePositionType="UserInput",Positions=bleMeshNetworkNodePositions,...
    VicinityRange=25,SimulationTime=simulationTime,...
    SourceDestinationPairs=sourceDestinationPairs,...
```

```
    Title="Scenario 1: Bluetooth Mesh Flooding");

% Object to handle visualization callbacks
visualizeScenarioOne = helperBLEMeshEventCallback(NetworkPlot=plotScenarioOne,...
    HighlightTransmissions=highlightTransmissions,Nodes=nodesScenarioOne);
init(visualizeScenarioOne);
```

Visualize the path and message flow between the source and destination nodes by adding listeners. For more information, see `eventlisteners`.

```
for nodeIndex = 1:numNodes
    % Add listener at each node for visualizing the message flow in the network
    if highlightTransmissions
        addlistener(nodesScenarioOne{nodeIndex},"PacketTransmissionStarted",...
            @(src,evt) visualizeScenarioOne.visualizeTransmissions(src,evt));
    end
    % Add listener at each node for storing the message receptions in the network
    addlistener(nodesScenarioOne{nodeIndex},"PacketReceptionEnded", ...
        @(src,evt) visualizeScenarioOne.storeRx(src,evt));
end

% Add listener at destination nodes for visualizing the path between
% source and destination
for dstIdx = 1:numel(sourceDestinationPairs)/2
    addlistener(nodesScenarioOne{sourceDestinationPairs(dstIdx,2)},"MeshAppDataReceived",...
        @(src,evt) visualizeScenarioOne.showPath(src,evt));
end
```

Run the simulation

Initialize the wireless network.

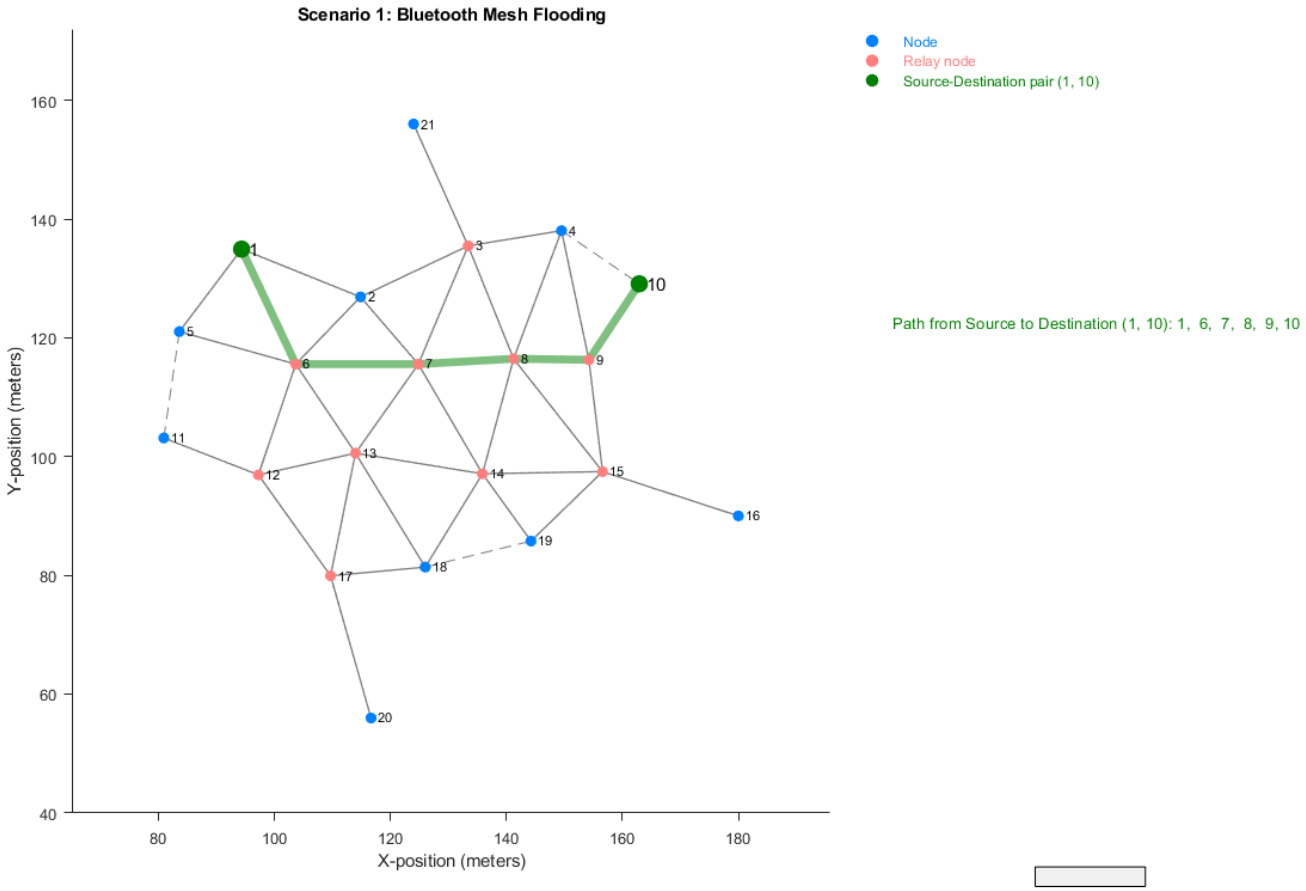
```
networkSimulator = helperWirelessNetwork(nodesScenarioOne);
```

Run all the nodes in the network for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

Update simulation progress at the end of the simulation.

```
updateProgressBar(visualizeScenarioOne.NetworkPlot,simulationTime);
```



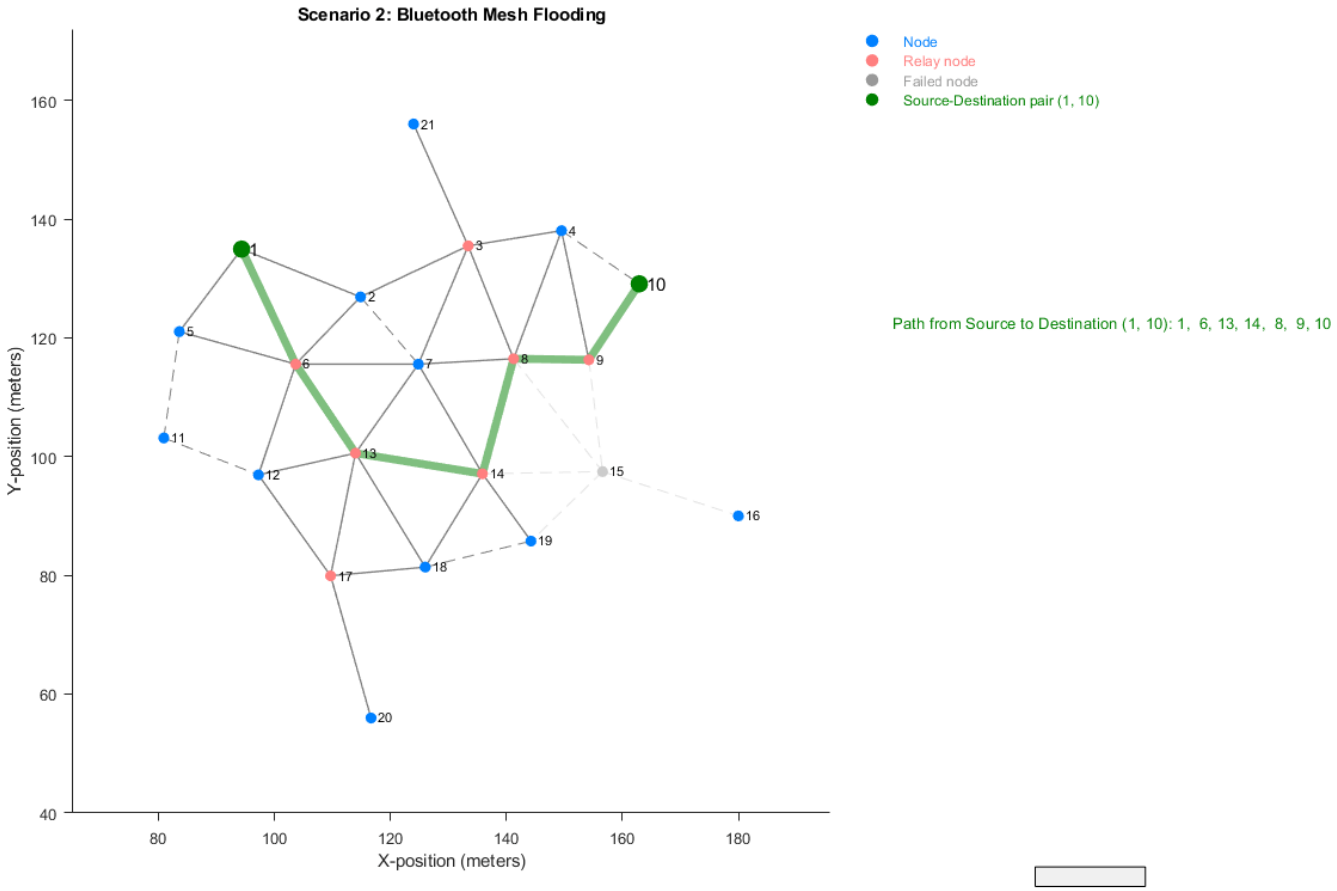
Simulate Mesh Network Scenario Two

The second mesh network scenario consists of 21 mesh nodes. In this scenario, the relay feature at Node7, Node12 is disabled and Node15 is removed from the network.

```
relayNodes = [3 6 8 9 13 14 17];
failedNode = 15;
```

Create, configure, and simulate the mesh network for scenario two by using the helperBLEMeshSimulateScenarioTwo helper object.

```
[nodesScenarioTwo,visualizeScenarioTwo] = helperBLEMeshSimulateScenarioTwo(bleMeshNetworkNodePos
    simulationTime,highlightTransmissions,sourceDestinationPairs,ttl,relayNodes,failedNode);
```



Simulation Results

View the table of statistics for both the scenarios by using the helperBLEMeshStatistics helper function.

```
statisticsScenarioOne = helperBLEMeshStatistics(nodesScenarioOne);
statisticsScenarioTwo = helperBLEMeshStatistics(nodesScenarioTwo);
```

Calculate PDR and path for both scenarios.

```
[pdrScenarioOne,pathScenarioOne] = meshResults(visualizeScenarioOne,statisticsScenarioOne)

pdrScenarioOne = 1

pathScenarioOne=1×4 table
    Source    Destination          Path          NumberOfHops
    ______    ___________    _________________    ____________

      1           10         {[1 6 7 8 9 10]}         5


[pdrScenarioTwo,pathScenarioTwo] = meshResults(visualizeScenarioTwo,statisticsScenarioTwo)

pdrScenarioTwo = 1
```

```
pathScenarioTwo=1×4 table
    Source    Destination            Path            NumberOfHops
    ______    ___________    ____________________    ____________

      1           10         {[1 6 13 14 8 9 10]}         6
```

The obtained results show that there exists a path between the source and destination nodes even if nodes fail randomly in the network.

Further Exploration

To obtain numerical results averaged over multiple simulations, the example implements the Monte Carlo method [4 on page 6-0]. To implement Monte Carlo simulations, use helperBLEMeshMonteCarloSimulations helper function. Each simulation run performs these steps.

1. Use a new seed to generate a random number
2. Randomly disable the relay nodes until a path exist between the source and destination nodes
3. Store the PDR

The Monte Carlo simulations outputs critical relay nodes required to ensure message delivery from the source to destination. The example performs Monte Carlo simulations by using these configuration parameters.

```
% Relay nodes
relayNodes = [3 6 7 8 9 12 13 14 15 17];

% Source and destination pair
sourceDestinationPairs = [1 10];

% TTL value for messages originated from the source
ttl = 10;
```

To view the simulation results, see `bleMeshMonteCarloResults` MAT file.

```
load("bleMeshMonteCarloResults.mat");
disp("Further exploration: Nodes [" + num2str(criticalRelaysInfo{1:5,1}') + ...
    "] are the top 5 critical relays for having communication between Node1 and Node10.");
```

```
Further exploration: Nodes [9   6   7   8  14] are the top 5 critical relays for having communica
```

Appendix

The example uses these helpers:

- helperWirelessNetwork: Simulate wireless network
- helperBLEMeshStatistics: Return statistics of each mesh node in the scenario
- helperBLEMeshEventCallback: Callback function to visualize message transmissions
- helperBLEMeshVisualizeNetwork: Bluetooth mesh network visualization
- helperBLEMeshSimulateScenarioTwo: Configure and run mesh network for scenario two
- helperBLEMeshMonteCarloSimulations: Bluetooth mesh network Monte Carlo simulations

Selected Bibliography

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 25, 2021. https://www.bluetooth.com.
2. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.
3. Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Profile." Version 1.0.1 https://www.bluetooth.com/.
4. Metropolis, Nicholas, and S. Ulam. "The Monte Carlo Method." *Journal of the American Statistical Association* 44, no. 247 (September 1949): 335–41. https://doi.org/10.1080/01621459.1949.10483310.

See Also

Functions

`configureFriendship`

Objects

`bluetoothLENode` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig`

More About

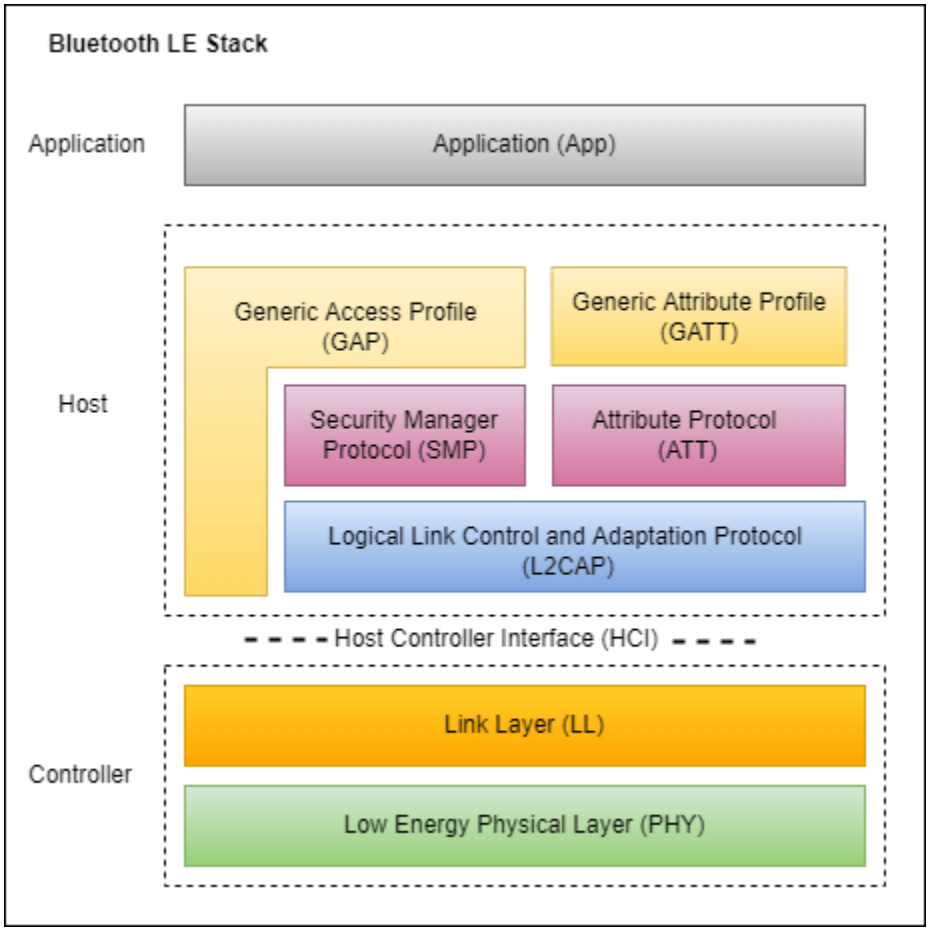
- “Bluetooth Mesh Networking” on page 8-64
- “Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks” on page 6-2
- “Create, Configure and Simulate Bluetooth Mesh Network” on page 9-45
- “Create, Configure, and Visualize Bluetooth Mesh Network” on page 9-17
- “Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network” on page 9-49
- “Bluetooth LE Node Statistics” on page 8-78

Bluetooth LE Link Layer Packet Generation and Decoding

This example shows how to generate and decode Bluetooth Low Energy (LE) link layer packets using Bluetooth® Toolbox.

Background

The Bluetooth Core Specification [2 on page 6-0] includes a Low Energy version for low-rate wireless personal area networks, that is referred to as Bluetooth Low Energy (LE) or Bluetooth Smart. The Bluetooth LE stack consists of: Generic Attribute Profile (GATT), Attribute Protocol (ATT), Security Manager Protocol (SMP), Logical Link Control and Adaptation Protocol (L2CAP), link layer (LL) and physical layer. Bluetooth LE was added to the standard for low energy devices generating small amounts of data, such as notification alerts used in such applications as home automation, health-care, fitness, and Internet of Things (IoT).



Packet Formats

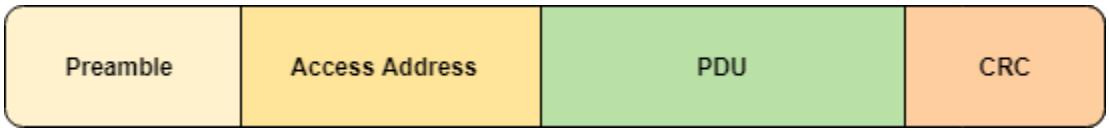
Bluetooth Core Specification [2 on page 6-0] defines two kinds of PHYs for Bluetooth LE. Each PHY has its own packet format.

(i) Uncoded PHYs (1 Mbps and 2 Mbps)

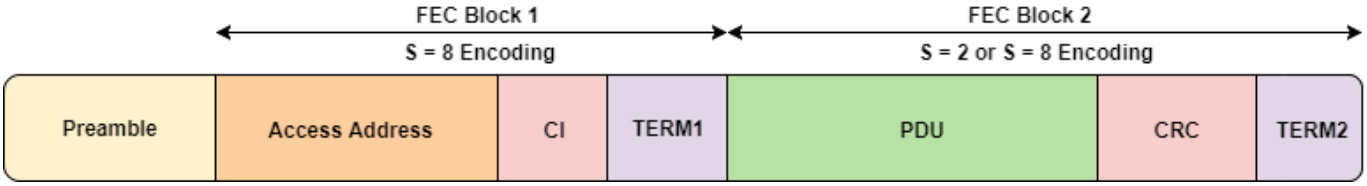
(ii) Coded PHYs (125 Kbps and 500 Kbps)

Coded PHYs use Forward Error Correction (FEC) encoding (with coding scheme S = 8 or S = 2) for the packets. The figures show the uncoded and coded PHY packet formats.

Format of LE Air Interface Packet for Uncoded PHY



Format of LE Air Interface Packet for Coded PHY



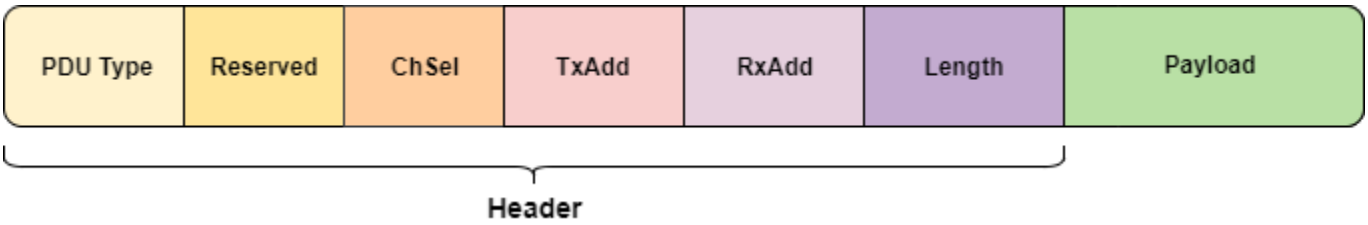
The Bluetooth® Toolbox generates LL packets that consist of Protocol Data Unit (PDU) and the Cyclic Redundancy Check (CRC) shown in the PHY packet.

Bluetooth LE classifies 40 RF channels into three advertising channels (Channel indices: 37, 38, 39) and thirty-seven data channels (Channel indices: 0 to 36). Bluetooth LE link layer defines two categories of PDUs, advertising channel PDUs and data channel PDUs. There are different PDU types within these two categories of PDUs. The access address field in the air interface packet format differentiates between a data channel PDU and an advertising channel PDU. Each category of PDU has its own format.

Advertising Channel PDUs

The advertising channel PDUs (see Section 2.3, Part-B, Vol-6 in [2 on page 6-0]) are used before a LL connection is created. These PDUs are transmitted only on the advertising channels and used in establishing the LL connection. The advertising channel PDU has a 16-bit header and a variable size payload.

The advertising channel PDU has the following packet format:



This example illustrates generation and decoding of advertising indication PDU. For a list of other advertising channel PDUs supported, see the `PDUType` property of `bleLLAdvertisingChannelPDUConfig` object.

Advertising indication: The advertising indication PDU is used when a device wants to advertise itself. This PDU contains the advertising data related to the application profile of the device.

Advertising Channel PDUs Generation

You can use the `bleLLAdvertisingChannelPDU` function to generate an advertising channel PDU. This function accepts a configuration object `bleLLAdvertisingChannelPDUConfig`. This object configures the fields required for generating an advertising channel PDU.

Advertising Indication Generation

To generate an 'Advertising indication' PDU, create a `bleLLAdvertisingChannelPDUConfig` object with `PDUType` set to `'Advertising indication'`.

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig('PDUType', 'Advertising indication');
```

Configure the fields:

```
% Advertiser address
cfgLLAdv.AdvertiserAddress = '012345ABCDEF';

% Advertising data
cfgLLAdv.AdvertisingData = '0201060D094261747465727920563 12E30'
```

```
cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Advertising indication'
         ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '012345ABCDEF'
          AdvertisingData: [17x2 char]
```

Generate an 'Advertising indication' PDU.

```
llAdvPDU = bleLLAdvertisingChannelPDU(cfgLLAdv);
```

Decoding Advertising Channel PDUs

You can use the `bleLLAdvertisingChannelPDUDecode` function to decode an advertising channel PDU. This function outputs the following information:

1. `status`: An enumeration of type `blePacketDecodeStatus`, specifying whether the LL decoding was successful.
2. `cfgLLAdv`: A LL advertising channel PDU configuration object of type `bleLLAdvertisingChannelPDUConfig`, which contains the decoded LL properties.

Provide the advertising channel PDU and an optional name-value pair specifying the format of the input data PDU to the `bleLLAdvertisingChannelPDUDecode` function. Default input format is 'bits'.

Decoding Advertising Indication

```
[llAdvDecodeStatus, cfgLLAdv] = bleLLAdvertisingChannelPDUDecode(llAdvPDU);
```

Observe the outputs

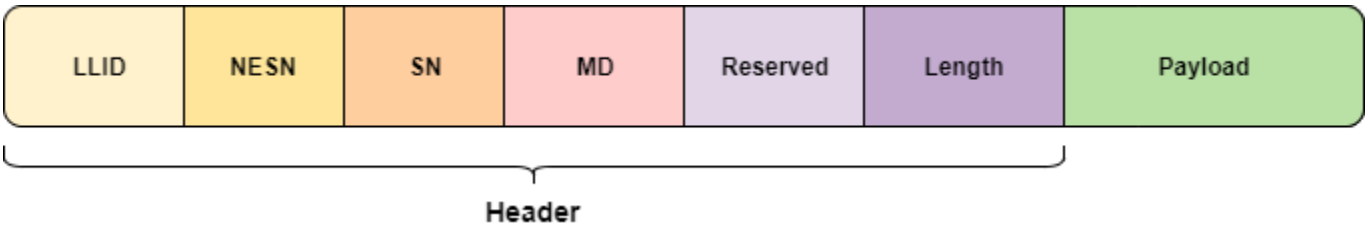
```
% Decoding is successful
if strcmp(llAdvDecodeStatus, 'Success')
```

```
    fprintf('Link layer decoding status is: %s\n\n', llAdvDecodeStatus);
    fprintf('Received Advertising channel PDU configuration is:\n');
    cfgLLAdv
% Decoding failed
else
    fprintf('Link layer decoding status is: %s\n', llAdvDecodeStatus);
end

Link layer decoding status is: Success

Received Advertising channel PDU configuration is:

cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Advertising indication'
         ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '012345ABCDEF'
          AdvertisingData: [17x2 char]
```

Data Channel PDUs

The data channel PDUs (see Section 2.4, Part-B, Vol-6 in [2 on page 6-0]) are used after a LL connection is created. The data channel PDUs consist of two sub-categories: **LL data PDUs** and **LL control PDUs**. The LL control PDUs are used for managing the LL connection and the LL data PDUs are used to carry the upper-layer data. The data channel PDU has a 16-bit header and a variable size payload.

The data channel PDUs have the following packet format:



This example illustrates generation and decoding of the following PDUs. For a list of other control PDU types and data PDU types supported see `Opcode` and `LLID` properties of `bleLLControlPDUConfig` and `bleLLDataChannelPDUConfig` objects, respectively.

1. *Channel map indication*: This LL control PDU is used to update the channel map at the peer device. This PDU contains the updated channel map indicating good and bad channels.
2. *Data (start fragment/complete)*: This LL data PDU is used to carry L2CAP data to the peer device.

Data Channel PDUs Generation

You can use the `bleLLDataChannelPDU` function to generate a data channel PDU. This function accepts a configuration object `bleLLDataChannelPDUConfig`, which configures the fields required for generating a data channel PDU. The `bleLLControlPDUConfig` is a sub-configuration object within `bleLLDataChannelPDUConfig`, control PDU payload fields are populated using the settings of this configuration object.

Data channel PDUs use the CRC initialization value obtained in the 'Connection indication' packet. The CRC initialization value used in the generation and decoding of packets.

```
% CRC initialization value
crcInit = 'ED321C';
```

LL Data PDU Generation

To generate a data PDU, create a `bleLLDataChannelPDUConfig` object with `LLID` set to `'Data (start fragment/complete)'`.

```
cfgLLData = bleLLDataChannelPDUConfig('LLID', ...
    'Data (start fragment/complete)');
```

Configure the fields:

```
% CRC initialization value
cfgLLData.CRCInitialization = crcInit;

% Sequence number
cfgLLData.SequenceNumber = 0;

% Next expected sequence number
cfgLLData.NESN = 1
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (start fragment/complete)'
                NESN: 1
      SequenceNumber: 0
            MoreData: 0
   CRCInitialization: 'ED321C'
```

A data PDU is used to transmit a payload from upper-layer. A 18-byte payload is used in this example.

```
% Payload
payload = '0E00050014010A001F004000170017000000';
```

Generate a data PDU using payload and configuration.

```
llDataPDU = bleLLDataChannelPDU(cfgLLData, payload);
```

LL Control PDU Generation

To generate a control PDU, create a `bleLLDataChannelPDUConfig` object with `LLID` set to `'Control'`.

```
cfgLLData = bleLLDataChannelPDUConfig('LLID', 'Control');
```

Configure the fields:

```
% CRC initialization value
cfgLLData.CRCInitialization = crcInit
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:
```

```
                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
   CRCInitialization: 'ED321C'
       ControlConfig: [1x1 bleLLControlPDUConfig]
```

You can configure the contents of an LL control PDU using `bleLLControlPDUConfig`.

Create a control PDU configuration object with `Opcode` set to `'Channel map indication'`.

```
cfgControl = bleLLControlPDUConfig('Opcode', 'Channel map indication');
```

Configure the fields:

```
% Used channels
cfgControl.UsedChannels = [9, 10, 12, 24, 28, 32];

% Connection event instant
cfgControl.Instant = 245
```

```
cfgControl =
  bleLLControlPDUConfig with properties:

          Opcode: 'Channel map indication'
         Instant: 245
    UsedChannels: [9 10 12 24 28 32]
```

Assign the updated control PDU configuration object to the `ControlConfig` property in the data channel PDU configuration object.

```
% Update the data channel PDU configuration
cfgLLData.ControlConfig = cfgControl;
```

Generate a control PDU with the updated configuration.

```
llControlPDU = bleLLDataChannelPDU(cfgLLData);
```

Decoding Data Channel PDUs

You can use the `bleLLDataChannelPDUDecode` function to decode a data channel PDU. This function outputs the following information:

1. `status`: An enumeration of type `blePacketDecodeStatus`, specifying whether the LL decoding was successful.
2. `cfgLLData`: An LL data channel PDU configuration object of type `bleLLDataChannelPDUConfig`, which contains the decoded LL properties.
3. `payload`: An n-by-2 character array representing the upper-layer payload carried by LL data PDUs.

Provide the data channel PDU, CRC initialization value and an optional name-value pair specifying the format of the input data PDU to the `bleLLDataChannelPDUDecode` function. Default input format is 'bits'.

Decoding LL Data PDU

```
[llDataDecodeStatus, cfgLLData, payload] = bleLLDataChannelPDUDecode(llDataPDU, crcInit);
```

Observe the outputs

```
% Decoding is successful
if strcmp(llDataDecodeStatus, 'Success')
    fprintf('Link layer decoding status is: %s\n\n', llDataDecodeStatus);
    fprintf('Received Data channel PDU configuration is:\n');
    cfgLLData
    fprintf('Size of the received upper-layer payload is: %d\n', ...
        numel(payload)/2);
% Decoding failed
else
    fprintf('Link layer decoding status is: %s\n', llDataDecodeStatus);
end
```

```
Link layer decoding status is: Success

Received Data channel PDU configuration is:

cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (start fragment/complete)'
                NESN: 1
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'


Size of the received upper-layer payload is: 18
```

Decoding LL Control PDU

```
[llControlDecodeStatus, cfgLLData] = bleLLDataChannelPDUDecode(llControlPDU, crcInit);
```

Observe the outputs

```
% Decoding is successful
if strcmp(llControlDecodeStatus, 'Success')
    fprintf('Link layer decoding status is: %s\n\n', llControlDecodeStatus);
    fprintf('Received Data channel PDU configuration is:\n');
    cfgLLData
    fprintf('Received control PDU configuration is:\n');
    cfgControl = cfgLLData.ControlConfig
% Decoding failed
else
    fprintf('Link layer decoding status is: %s\n', llControlDecodeStatus);
end
```

```
Link layer decoding status is: Success

Received Data channel PDU configuration is:

cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
```

```
            MoreData: 0
    CRCInitialization: '012345'
        ControlConfig: [1x1 bleLLControlPDUConfig]


Received control PDU configuration is:

cfgControl =
  bleLLControlPDUConfig with properties:

          Opcode: 'Channel map indication'
         Instant: 245
    UsedChannels: [9 10 12 24 28 32]
```

Exporting to a PCAP File

This example uses `blePCAPWriter` object to export the generated PDUs to a file with .pcap extension or .pcapng extension. To analyze and visualize this file, use a third part packet analyzer such as Wireshark.

Prepend access address

The PCAP format expects access address to be prepended to the LL packet.

```
% Advertising channel PDUs use the default access address
advAccessAddress = '8E89BED6';
advAccessAddressBinary = int2bit(hex2dec(advAccessAddress), 32, false);

% Data channel PDUs use the access address obtained from 'Connection
% indication' packet. A random access address is used for this example
connAccessAddress = 'E213BC42';
connAccessAddressBinary = int2bit(hex2dec(connAccessAddress), 32, false);

% Prepend access address
llPkts{1} = [advAccessAddressBinary; llAdvPDU];
llPkts{2} = [connAccessAddressBinary; llDataPDU];
llPkts{3} = [connAccessAddressBinary; llControlPDU];
```

Export to a PCAP file

Create an object of type `blePCAPWriter` and specify the packet capture file name.

```
% Create the Bluetooth LE PCAP Writer file object
pcapObj = blePCAPWriter("FileName", "bluetoothLELLPackets");
```

Use the `blePCAPWriter` function to write all the Bluetooth LE LL PDUs to a PCAP file. The constant `timestamp` specifies the capture time of a PDU. In this example, the capture time is same for all the PDUs.

```
timestamp = 124800; % timestamp (in microseconds)

% Write all the LL PDUs to the PCAP file
for idx = 1:numel(llPkts)
    write(pcapObj, llPkts{idx}, timestamp, 'PacketFormat', 'bits');
end

% Clear the object
clear pcapObj;
```

Visualization of the Generated Link Layer Packets

You can open the PCAP file containing the generated LL packets in a packet analyzer. The packets decoded by the packet analyzer match the standard compliant LL packets generated by the Bluetooth® Toolbox. The captured analysis of the packets is shown below.

- **Advertising indication**

```
Frame 1: 32 bytes on wire (256 bits), 32 bytes captured (256 bits)
Bluetooth
   [Source: SonyMobi_ab:cd:ef (01:23:45:ab:cd:ef)]
   [Destination: Broadcast (ff:ff:ff:ff:ff:ff)]
Bluetooth Low Energy Link Layer
   Access Address: 0x8e89bed6
 ⌄ Packet Header: 0x1740 (PDU Type: ADV_IND, ChSel: #1, TxAdd: Random)
      .... 0000 = PDU Type: ADV_IND (0x0)
      ...0 .... = RFU: 0
      ..0. .... = Channel Selection Algorithm: #1
      .1.. .... = Tx Address: Random
      0... .... = Reserved: False
      Length: 23
   Advertising Address: SonyMobi_ab:cd:ef (01:23:45:ab:cd:ef)
 ⌄ Advertising Data
    › Flags
    › Device Name: Battery V1.0
   CRC: 0xeebc04
```

- **LL data PDU (carrying L2CAP payload)**

```
Frame 2: 27 bytes on wire (216 bits), 27 bytes captured (216 bits)
Bluetooth
Bluetooth Low Energy Link Layer
   Access Address: 0xe213bc42
 ⌄ Data Header: 0x1206
      .... ..10 = LLID: Start of an L2CAP message or a complete L2CAP message with no fragmentation (0x2)
      .... .1.. = Next Expected Sequence Number: 1
      .... 0... = Sequence Number: 0
      ...0 .... = More Data: False
      000. .... = RFU: 0
      Length: 18
 › CRC: 0xbee203
Bluetooth L2CAP Protocol
   Length: 14
   CID: Low Energy L2CAP Signaling Channel (0x0005)
 ⌄ Command: LE Credit Based Connection Request
      Command Code: LE Credit Based Connection Request (0x14)
      Command Identifier: 0x01
      Command Length: 10
      LE PSM: Fixed, SIG Assigned (0x001f)
      [PSM: ATT (0x001f)]
      Source CID: Dynamically Allocated Channel (0x0040)
      MTU: 23
      MPS: 23
      Initial Credits: 0
```

- **LL control PDU (channel map indication)**

```
Frame 3: 17 bytes on wire (136 bits), 17 bytes captured (136 bits)
Bluetooth
Bluetooth Low Energy Link Layer
   Access Address: 0xe213bc42
 ⌄ Data Header: 0x0803
      .... ..11 = LLID: Control PDU (0x3)
      .... .0.. = Next Expected Sequence Number: 0
      .... 0... = Sequence Number: 0
      ...0 .... = More Data: False
      000. .... = RFU: 0
      Length: 8
   Control Opcode: LL_CHANNEL_MAP_REQ (0x01)
 ⌄ Channel Map: 0016001101
      .... ...0 = RF Channel 1 (2404 MHz - Data - 0): False
      .... ..0. = RF Channel 2 (2406 MHz - Data - 1): False
      .... .0.. = RF Channel 3 (2408 MHz - Data - 2): False
      .... 0... = RF Channel 4 (2410 MHz - Data - 3): False
      ...0 .... = RF Channel 5 (2412 MHz - Data - 4): False
      ..0. .... = RF Channel 6 (2414 MHz - Data - 5): False
      .0.. .... = RF Channel 7 (2416 MHz - Data - 6): False
      0... .... = RF Channel 8 (2418 MHz - Data - 7): False
      .... ...0 = RF Channel 9 (2420 MHz - Data - 8): False
      .... ..1. = RF Channel 10 (2422 MHz - Data - 9): True
      .... .1.. = RF Channel 11 (2424 MHz - Data - 10): True
      .... 0... = RF Channel 13 (2428 MHz - Data - 11): False
      ...1 .... = RF Channel 14 (2430 MHz - Data - 12): True
      ..0. .... = RF Channel 15 (2432 MHz - Data - 13): False
      .0.. .... = RF Channel 16 (2434 MHz - Data - 14): False
      0... .... = RF Channel 17 (2436 MHz - Data - 15): False
      .... ...0 = RF Channel 18 (2438 MHz - Data - 16): False
      .... ..0. = RF Channel 19 (2440 MHz - Data - 17): False
      .... .0.. = RF Channel 20 (2442 MHz - Data - 18): False
      .... 0... = RF Channel 21 (2444 MHz - Data - 19): False
      ...0 .... = RF Channel 22 (2446 MHz - Data - 20): False
      ..0. .... = RF Channel 23 (2448 MHz - Data - 21): False
      .0.. .... = RF Channel 24 (2450 MHz - Data - 22): False
      0... .... = RF Channel 25 (2452 MHz - Data - 23): False
      .... ...1 = RF Channel 26 (2454 MHz - Data - 24): True
      .... ..0. = RF Channel 27 (2456 MHz - Data - 25): False
      .... .0.. = RF Channel 28 (2458 MHz - Data - 26): False
      .... 0... = RF Channel 29 (2460 MHz - Data - 27): False
      ...1 .... = RF Channel 30 (2462 MHz - Data - 28): True
      ..0. .... = RF Channel 31 (2464 MHz - Data - 29): False
      .0.. .... = RF Channel 32 (2466 MHz - Data - 30): False
      0... .... = RF Channel 33 (2468 MHz - Data - 31): False
      .... ...1 = RF Channel 34 (2470 MHz - Data - 32): True
      .... ..0. = RF Channel 35 (2472 MHz - Data - 33): False
      .... .0.. = RF Channel 36 (2474 MHz - Data - 34): False
      .... 0... = RF Channel 37 (2476 MHz - Data - 35): False
      ...0 .... = RF Channel 38 (2478 MHz - Data - 36): False
      ..0. .... = RF Channel 0 (2402 MHz - Reserved for Advertising - 37): False
      .0.. .... = RF Channel 12 (2426 MHz - Reserved for Advertising - 38): False
      0... .... = RF Channel 39 (2480 MHz - Reserved for Advertising - 39): False
   Instant: 245
 ⌄ CRC: 0x563473
```

Conclusion

This example demonstrated generation and decoding of LL packets specified in the Bluetooth standard [2 on page 6-0]. You can use a packet analyzer to view the generated LL packets.

Selected Bibliography

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 25, 2021. https://www.bluetooth.com.
2. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.
3. "Development/LibpcapFileFormat - The Wireshark Wiki." https://wiki.wireshark.org/Development/LibpcapFileFormat
4. Group, The Tcpdump. "Tcpdump/Libpcap Public Repository." https://www.tcpdump.org.

See Also

Functions

`bleLLAdvertisingChannelPDU` | `bleLLAdvertisingChannelPDUDecode` | `bleLLDataChannelPDU` | `bleLLDataChannelPDUDecode`

Objects

`bleLLAdvertisingChannelPDUConfig` | `bleLLDataChannelPDUConfig` | `bleLLControlPDUConfig` | `blePCAPWriter`

More About

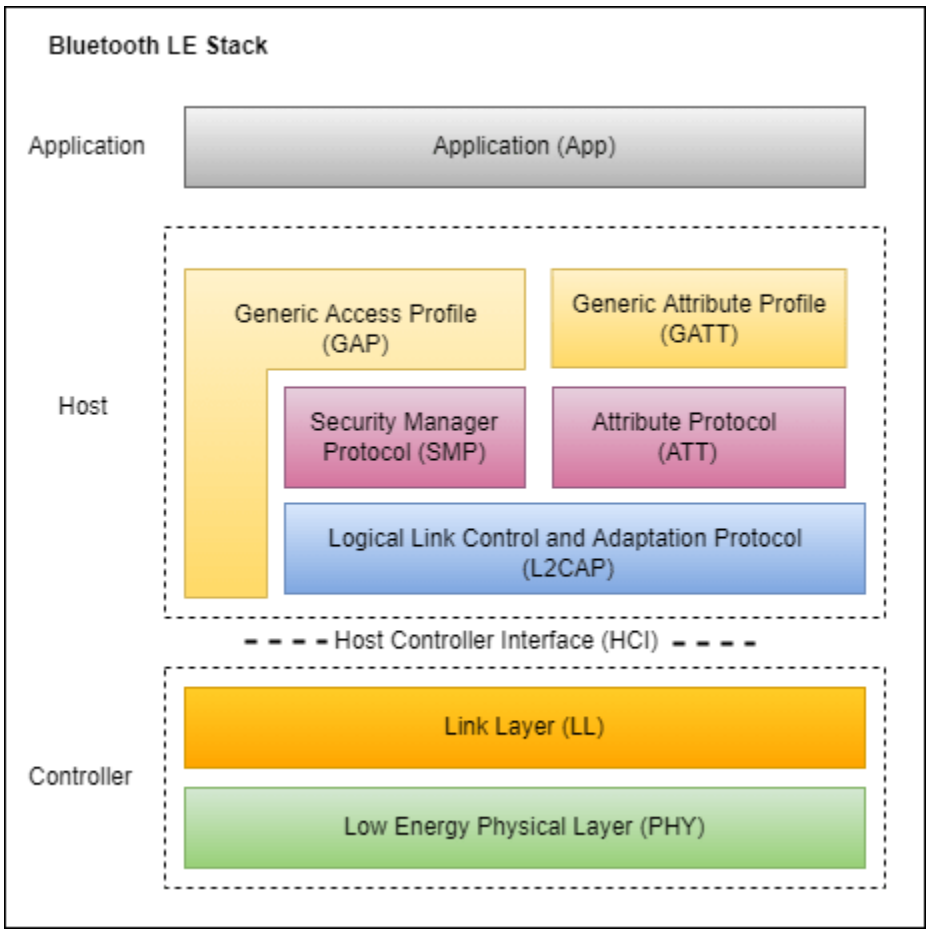
- “Generate and Decode Bluetooth Protocol Data Units” on page 9-2
- “Bluetooth LE L2CAP Frame Generation and Decoding” on page 6-30
- “Modeling of Bluetooth LE Devices with Heart Rate Profile” on page 6-39

Bluetooth LE L2CAP Frame Generation and Decoding

This example shows how to generate and decode Bluetooth Low Energy L2CAP frames using Bluetooth® Toolbox.

Background

The Bluetooth Core Specification [2 on page 6-0] includes a Low Energy (LE) version for low-rate wireless personal area networks, that is referred to as Bluetooth Low Energy (LE) or Bluetooth Smart. The Bluetooth LE stack consists of: Generic Attribute Profile (GATT), Attribute Protocol (ATT), Security Manager Protocol (SMP), Logical Link Control and Adaptation Protocol (L2CAP), Link layer and Physical layer. Bluetooth LE was added to the standard for low energy devices generating small amounts of data, such as notification alerts used in such applications as home automation, health-care, fitness, and Internet of Things (IoT).

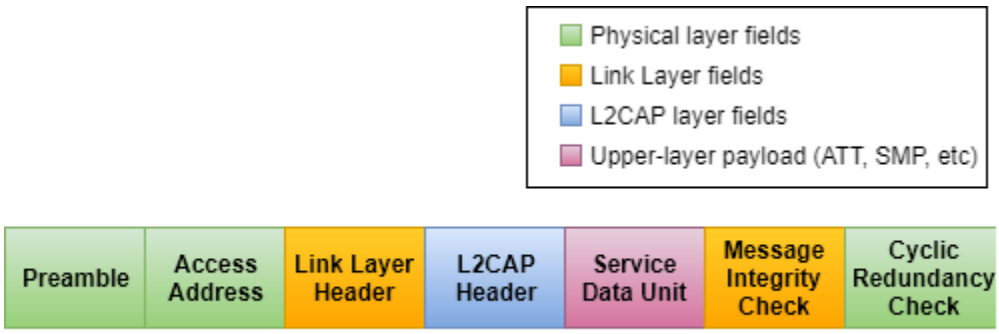


The L2CAP layer in Bluetooth LE corresponds to the higher sub-layer i.e. Logical Link Control (LLC) of the Data Link Layer in the OSI reference model. The L2CAP is above the PHY and Link Layer of Bluetooth LE. The Bluetooth LE specification optimized and simplified the L2CAP when compared to classic Bluetooth.

L2CAP in Bluetooth LE is responsible for:

1. Logical connection establishment
2. Protocol multiplexing
3. Segmentation and reassembly
4. Flow control per 'dynamic' L2CAP channel.

The L2CAP layer adds an L2CAP basic header to the higher-layer payload and passes the Protocol Data Unit (PDU) to the Link Layer below it.



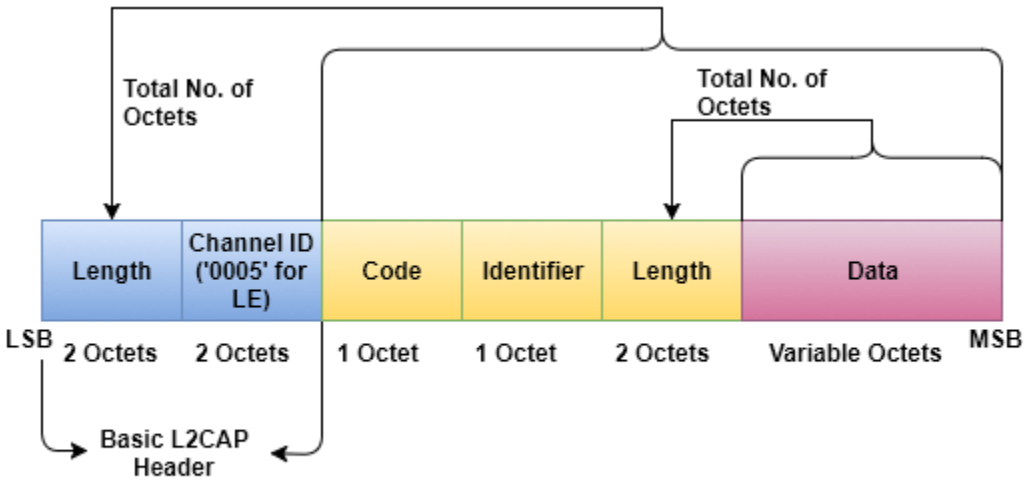
Format of Bluetooth LE Frame

L2CAP Frames

L2CAP Frames consist of two sub-categories: **Data frames** and **Signaling frames**. There are different types of frames within these two categories of frames. The *Data frames* are again sub-categorized into *B-frame* (Basic information frame) and *LE-frame* (Low Energy information frame). Each frame type has its own format.

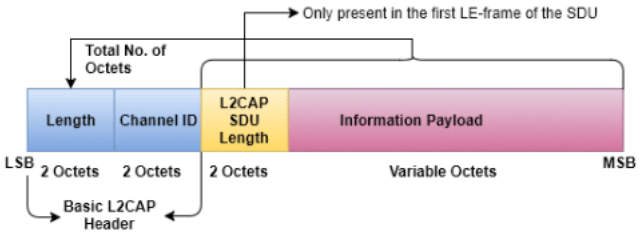
A **channel identifier (CID)** is the local name representing a logical channel endpoint on the device. For the protocols, such as the ATT and SMP, these channels are *fixed* by the Bluetooth Special Interest Group (SIG). For application specific profiles, such as Internet Protocol Support Profile (IPSP) and Object Transfer Profile (OTP), these channels are *dynamically* allocated.

Signaling frames are used with a fixed logical channel called signaling channel (**'0005'**) and used for logical connection establishment between peer devices using the LE credit based flow control mechanism. These signaling frames are also used for updating the connection parameters (Peripheral latency, Connection timeout, Minimum connection interval and Maximum connection interval) when connection parameters request procedure is not supported in the Link Layer.

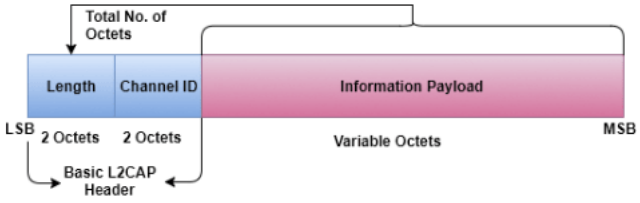


Format of Bluetooth LE L2CAP Signaling Frame

Data frames (B-frames and LE-frames) carry the upper-layer payload as 'Information Payload' in its frame format. *B-Frames* are used to carry fixed channels (ATT and SMP with fixed logical channels **'0004'** and **'0006'** respectively) payload. *LE-frames* are used to carry payload through dynamically created logical channels for application specific profiles, such as IPSP and OTP.



Format of L2CAP LE Information Frame (LE-Frame)



Format of L2CAP Basic Information Frame (B-Frame)

This example illustrates generation and decoding of the following frames. For a list of other signaling frames supported, see the `CommandType` property of `bleL2CAPFrameConfig` object.

1. Flow control credit: This signaling frame is sent to create and configure an L2CAP logical channel between two devices.

2. B-frames over fixed channels (ATT, SMP, etc.): This frame is used for carrying fixed channels payload in basic L2CAP mode.

3. LE-frames over dynamic channels (profiles like IPSP, OTP, etc.): This frame is used for carrying dynamic channels payload in LE credit based flow control mode.

L2CAP Frames Generation

You can use the `bleL2CAPFrame` function to generate an L2CAP frame. This function accepts a configuration object `bleL2CAPFrameConfig`. This object configures the fields required for generating an L2CAP frame.

Signaling frame generation

To generate a signaling frame, create a `bleL2CAPFrameConfig` object with `ChannelIdentifier` set to `'0005'`.

```
cfgL2CAP = bleL2CAPFrameConfig('ChannelIdentifier', '0005');
```

Configure the fields:

```
% Command type
cfgL2CAP.CommandType = 'Flow control credit';

% Source channel identifier
cfgL2CAP.SourceChannelIdentifier = '0041';

% LE credits
cfgL2CAP.Credits = 25
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Flow control credit'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0041'
                    Credits: 25
```

Generate a 'Flow control credit' command.

```
sigFrame = bleL2CAPFrame(cfgL2CAP);
```

B-frame generation

To generate a B-frame (carrying ATT PDU), create a `bleL2CAPFrameConfig` object with `ChannelIdentifier` set to `'0004'` (ATT channel ID).

```
cfgL2CAP = bleL2CAPFrameConfig('ChannelIdentifier', '0004')
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

    ChannelIdentifier: '0004'
```

A B-frame is used to transmit payload from the ATT upper-layer. A 5-byte ATT PDU is used as payload in this example.

```
payload = ['04';'01';'00';'FF';'FF'];
```

Generate an L2CAP B-frame using the payload and configuration.

```
bFrame = bleL2CAPFrame(cfgL2CAP, payload);
```

LE-frame generation

To generate an LE-frame, create a `bleL2CAPFrameConfig` object with `ChannelIdentifier` set to `'0035'`.

```
cfgL2CAP = bleL2CAPFrameConfig('ChannelIdentifier', '0035')
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

    ChannelIdentifier: '0035'
```

An LE-frame is used to transmit the payload of dynamic channels. A 2-byte payload is used in this example.

```
payload = ['01';'02'];
```

Generate an L2CAP LE-frame using the payload and configuration.

```
leFrame = bleL2CAPFrame(cfgL2CAP, payload);
```

Decoding L2CAP Frames

You can use the `bleL2CAPFrameDecode` function to decode an L2CAP frame. This function outputs the following information:

1. `status`: An enumeration of type `blePacketDecodeStatus`, which indicates whether or not the L2CAP decoding was successful.
2. `cfgL2CAP`: An L2CAP frame configuration object of type `bleL2CAPFrameConfig`, which contains the decoded L2CAP properties.

This function accepts a Bluetooth LE L2CAP frame as the input.

Decoding Signaling frame

```
[sigFrameDecodeStatus, cfgL2CAP] = bleL2CAPFrameDecode(sigFrame);
```

Observe the outputs

```
% Decoding is successful
if strcmp(sigFrameDecodeStatus, 'Success')
    fprintf('L2CAP decoding status is: %s\n\n', sigFrameDecodeStatus);
    fprintf('Received L2CAP signaling frame configuration is:\n');
    cfgL2CAP
% Decoding failed
else
    fprintf('L2CAP decoding status is: %s\n', sigFrameDecodeStatus);
end
```

```
L2CAP decoding status is: Success

Received L2CAP signaling frame configuration is:

cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Flow control credit'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0041'
                    Credits: 25
```

Decoding B-frame

```
[bFrameDecodeStatus, cfgL2CAP, payload] = bleL2CAPFrameDecode(bFrame);
```

Observe the outputs

```
% Decoding is successful
if strcmp(bFrameDecodeStatus, 'Success')
    fprintf('L2CAP decoding status is: %s\n\n', bFrameDecodeStatus);
    fprintf('Received L2CAP B-frame configuration is:\n');
    cfgL2CAP
    fprintf('Payload carried by L2CAP B-frame is:\n');
    payload
% Decoding failed
else
    fprintf('L2CAP decoding status is: %s\n', bFrameDecodeStatus);
end
```

```
L2CAP decoding status is: Success

Received L2CAP B-frame configuration is:

cfgL2CAP =
  bleL2CAPFrameConfig with properties:

    ChannelIdentifier: '0004'


Payload carried by L2CAP B-frame is:

payload = 5x2 char array
    '04'
    '01'
    '00'
    'FF'
    'FF'
```

Decoding LE-frame

```
[leFrameDecodeStatus, cfgL2CAP, payload] = bleL2CAPFrameDecode(leFrame);
```

Observe the outputs

```
% Decoding is successful
if strcmp(leFrameDecodeStatus, 'Success')
    fprintf('L2CAP decoding status is: %s\n\n', leFrameDecodeStatus);
    fprintf('Received L2CAP LE-frame configuration is:\n');
    cfgL2CAP
    fprintf('Payload carried by L2CAP LE-frame is:\n');
    payload
% Decoding failed
else
    fprintf('L2CAP decoding status is: %s\n', leFrameDecodeStatus);
end
```

```
L2CAP decoding status is: Success

Received L2CAP LE-frame configuration is:

cfgL2CAP =
  bleL2CAPFrameConfig with properties:
```

```
    ChannelIdentifier: '0035'

Payload carried by L2CAP LE-frame is:

payload = 2x2 char array
    '01'
    '02'
```

Exporting to a PCAP File

This example uses `blePCAPWriter` object to export the generated PDUs to a file with .pcap extension or .pcapng extension. To analyze and visualize this file, use a third part packet analyzer such as Wireshark [3 on page 6-0].

The PCAP format expects L2CAP frame to be enclosed within Link Layer packet and also expects the generated packet to be prepended with the access address. The following commands generate a PCAP file for the L2CAP frames generated in this example.

```
% Create a cell array of L2CAP frames
l2capFrames = {sigFrame, bFrame, leFrame};
llPackets = cell(1, numel(l2capFrames));
for i = 1:numel(llPackets)
    % Add Link Layer header to the generated L2CAP frame
    cfgLLData = bleLLDataChannelPDUConfig('LLID', 'Data (start fragment/complete)');
    llDataPDU = bleLLDataChannelPDU(cfgLLData, l2capFrames{i});
    % Prepend access address. A 4-byte access address is used in this example
    llPackets{i} = [int2bit(hex2dec('01234567'), 32, false); llDataPDU];
end
```

Export to a PCAP file

Create an object of type `blePCAPWriter` and specify the packet capture file name.

```
% Create the Bluetooth LE PCAP Writer file object
pcapObj = blePCAPWriter("FileName", "bluetoothLEL2CAPFrames");
```

Use the `blePCAPWriter` function to write all the Bluetooth LE LL PDUs to a PCAP file. The constant `timestamp` specifies the capture time of a PDU. In this example, the capture time is same for all the PDUs.

```
timestamp = 124800; % timestamp (in microseconds)

% Write all the LL PDUs to the PCAP file
for idx = 1:numel(llPackets)
    write(pcapObj, llPackets{idx}, timestamp, "PacketFormat", "bits");
end

% Clear the object
clear pcapObj;
```

Visualization of the Generated L2CAP Frames

You can open the PCAP file containing the generated L2CAP frames in a packet analyzer. The L2CAP frames decoded by the packet analyzer match the standard compliant L2CAP frames generated by Bluetooth® Toolbox. The captured analysis of the L2CAP frames is shown below.

- **Signaling frame (flow control credit)**

```
Frame 1: 21 bytes on wire (168 bits), 21 bytes captured (168 bits)
Bluetooth
Bluetooth Low Energy Link Layer
Bluetooth L2CAP Protocol
    Length: 8
    CID: Low Energy L2CAP Signaling Channel (0x0005)
  ⌄ Command: LE Flow Control Credit
        Command Code: LE Flow Control Credit (0x16)
        Command Identifier: 0x01
        Command Length: 4
        CID: Dynamically Allocated Channel (0x0041)
        Credits: 25
```

- **B-frame (carrying ATT PDU)**

```
Frame 2: 18 bytes on wire (144 bits), 18 bytes captured (144 bits)
Bluetooth
Bluetooth Low Energy Link Layer
Bluetooth L2CAP Protocol
    Length: 5
    CID: Attribute Protocol (0x0004)
Bluetooth Attribute Protocol
  ⌄ Opcode: Find Information Request (0x04)
        0... .... = Authentication Signature: False
        .0.. .... = Command: False
        ..00 0100 = Method: Find Information Request (0x04)
    Starting Handle: 0x0001
    Ending Handle: 0xffff
```

- **LE-frame (carrying dynamic channel payload)**

```
Frame 3: 17 bytes on wire (136 bits), 17 bytes captured (136 bits)
Bluetooth
Bluetooth Low Energy Link Layer
Bluetooth L2CAP Protocol
    Length: 4
    CID: Reserved (0x0035)
    Payload: 02000102
```

Conclusion

This example demonstrated generation and decoding of L2CAP frames specified in the Bluetooth [2 on page 6-0] standard. You can use a packet analyzer to view the generated L2CAP frames.

Selected Bibliography

1 Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 25, 2021. https://www.bluetooth.com.

2 Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

3 "Development/LibpcapFileFormat - The Wireshark Wiki." https://wiki.wireshark.org/Development/LibpcapFileFormat.

4 Group, The Tcpdump. "Tcpdump/Libpcap Public Repository." https://www.tcpdump.org.

See Also

Functions
`bleL2CAPFrame` | `bleL2CAPFrameDecode`

Objects
`bleL2CAPFrameConfig` | `blePCAPWriter`

More About

- “Generate and Decode Bluetooth Protocol Data Units” on page 9-2
- “Bluetooth LE Link Layer Packet Generation and Decoding” on page 6-19
- “Modeling of Bluetooth LE Devices with Heart Rate Profile” on page 6-39

Modeling of Bluetooth LE Devices with Heart Rate Profile

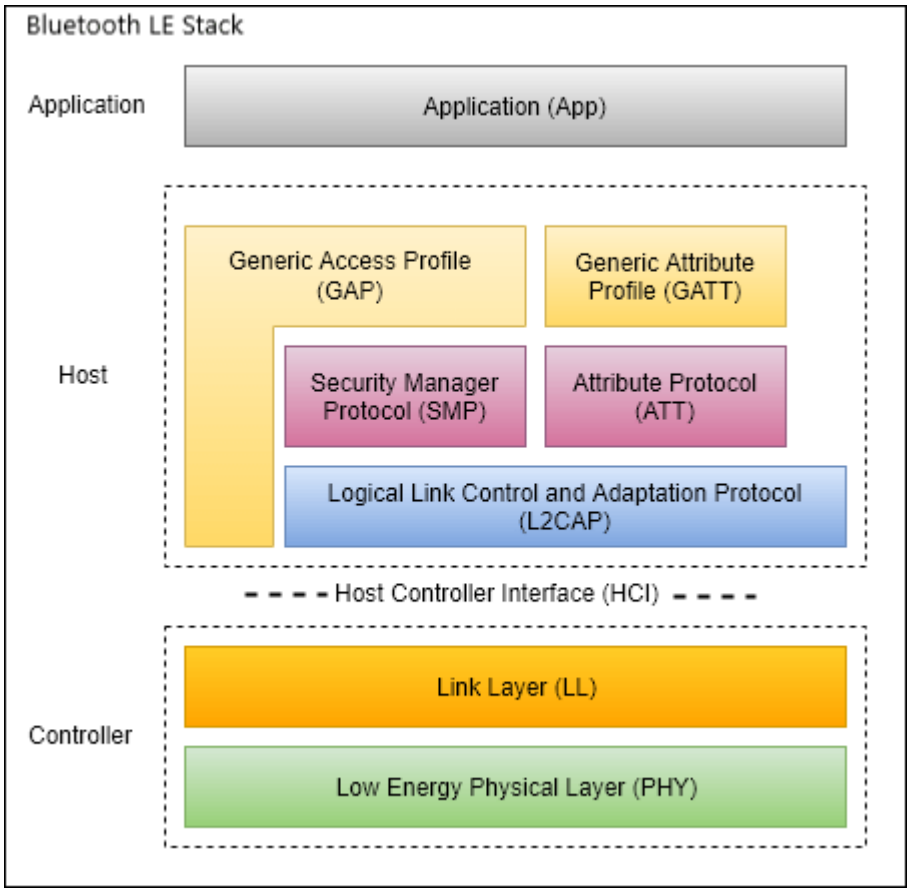
This example shows you how to model Bluetooth® low energy (LE) devices with the Heart Rate Profile (HRP) by using Bluetooth® Toolbox.

Using this example, you can:

- Create and configure a Bluetooth LE HRP client-server scenario with a smartphone as the client and a wrist band with a heart rate sensor as the server.
- Configure the client to perform service and characteristic discovery.
- Configure the client to enable receiving notifications for a characteristic from the server.
- Export the generated protocol data units (PDUs) to a file with `.pcap` extension.
- Visualize the generated PDUs by using a third party packet analyzer such as Wireshark [4 on page 6-0].

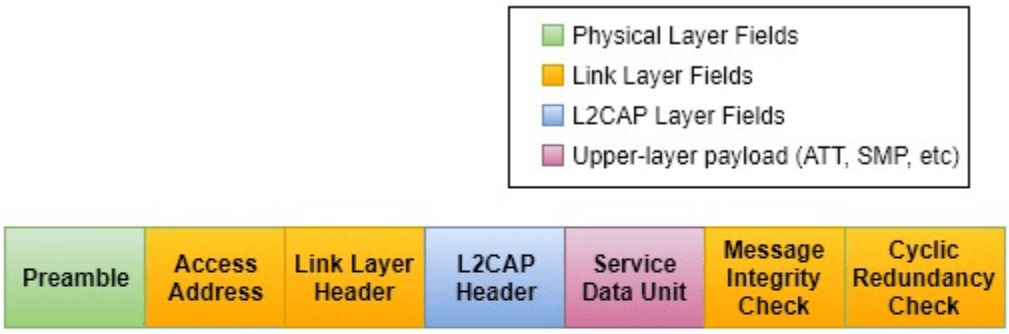
Background

The Bluetooth core specification [2 on page 6-0] includes a Low Energy version for low-rate wireless personal area networks, that is referred to as Bluetooth Low Energy (Bluetooth LE) or Bluetooth Smart. The Bluetooth LE stack consists of: Generic Attribute Profile (GATT), Attribute Protocol (ATT), Security Manager Protocol (SMP), Logical Link Control and Adaptation Protocol (L2CAP), Link Layer (LL) and Physical layer (PHY). Bluetooth LE was added to the standard for low energy devices generating small amounts of data, such as notification alerts used in such applications as home automation, health-care, fitness, and Internet of Things (IoT).



Attribute Protocol

The ATT is built on top of the L2CAP layer of Bluetooth LE. ATT defines a set of Protocol Data Units (PDUs) that are used for data exchange in GATT-based profiles.



Format of Bluetooth Low Energy(Bluetooth LE) frame

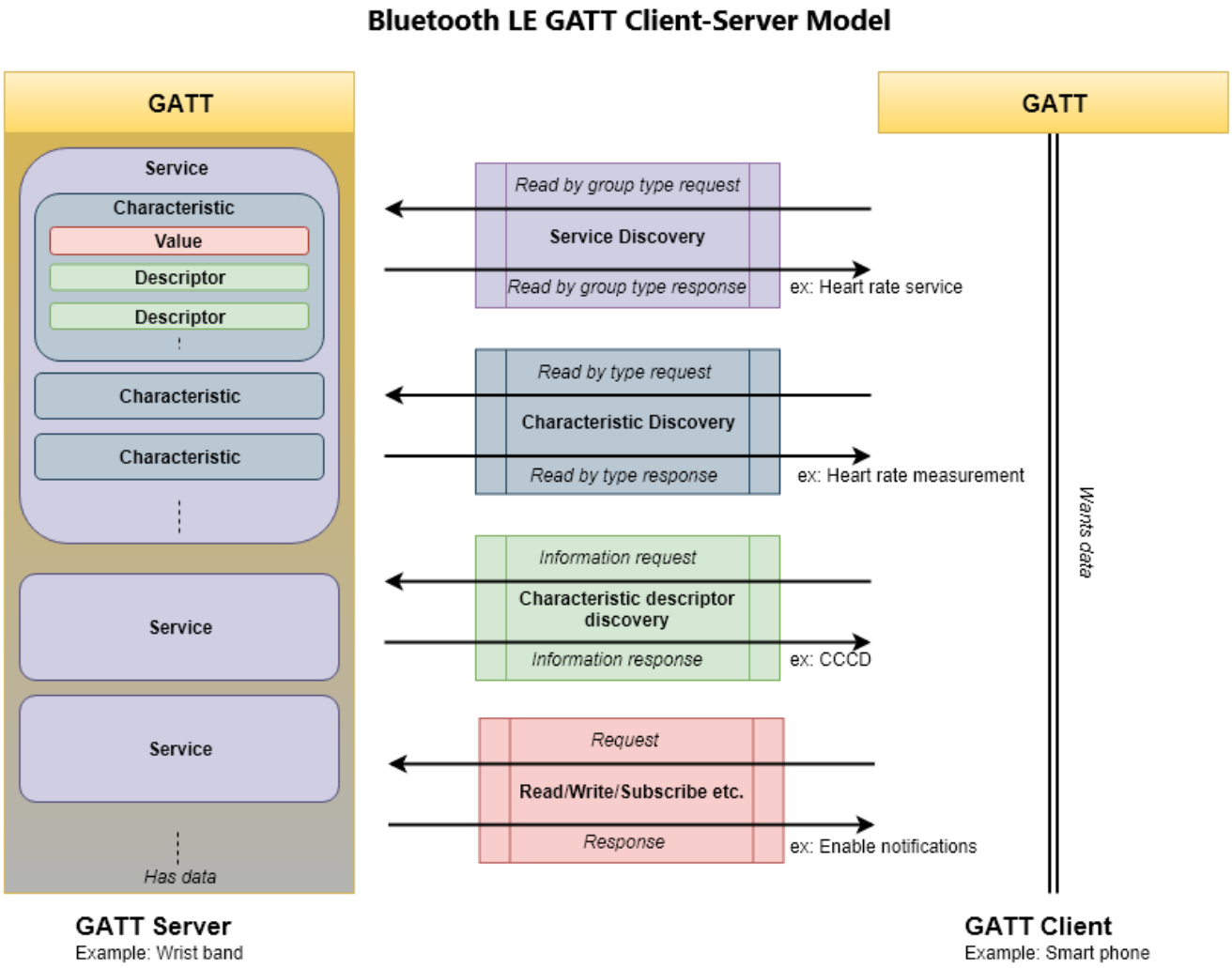
Service Data Unit - Attribute Protocol(ATT) PDU format

Generic Attribute Profile

The GATT is a service framework built using ATT. GATT handles the generation of requests or responses based on application data from the higher layers or ATT PDU received from the lower layer. It stores the information in the form of services, characteristics, and characteristic descriptors. It uses a client-server architecture.

GATT Terminology:

- **Service:** A service is a collection of data and associated behaviors to accomplish a particular function or feature. Example: A heart rate service that allows measurement of a heart rate.
- **Characteristic:** A characteristic is a value used in a service along with its permissions. Example: A heart rate measurement characteristic contains information about the measured heart rate value.
- **Characteristic descriptor:** Descriptors of the characteristic behavior. Example: A Client Characteristic Configuration Descriptor (CCCD), describes whether or not the server has to notify the client in a response containing the characteristic value.
- **GATT-Client:** Initiates commands and requests to the server, and receives responses, indications and notifications sent by the server.
- **GATT-Server:** Accepts incoming commands and requests from a client, and sends responses, indications, and notifications to the client.



Heart Rate Profile

Heart Rate Profile (HRP) [3 on page 6-0] is a GATT-based low energy profile defined by the Bluetooth Special Interest Group (SIG). The HRP defines the communication between a GATT-server of a heart rate sensor device, such as a wrist band, and a GATT-client, such as a smart phone or tablet. The HRP is widely used in fitness applications to collect heart rate measurements.

Bluetooth LE HRP Client-Server Scenario

In this scenario, the GATT-server is a wrist band with a heart rate sensor and the GATT-client is a smart phone.

```
% Create objects for GATT-server and GATT-client devices
gattServer = helperBLEGATTServer;
gattClient = helperBLEGATTClient;
```

Initially, the HRP client discovers the services, characteristics, characteristic descriptors defined at the server. After discovery, the client subscribes for heart rate measurement notifications.

Service Discovery

Clients perform a *service discovery* operation to get information about the available services. In service discovery, the client invokes `'Discover all primary services'` by sending a *Read by group type request* ATT PDU. The server responds with the available services and their associated handles by sending a 'Read by group type response' ATT PDU. A *handle* is a unique identifier of an attribute that are dynamically assigned by the server.

Client request for services at Server

The generateATTPDU object function generates an ATT PDU corresponding to the given sub-procedure as specified in the Bluetooth core specification.

```
% Preallocate a variable to store the generated link layer packets
pcapPackets = cell(1, 9);
count = 1;

% Configure a GATT client to discover services available at the server
gattClient.SubProcedure = 'Discover all primary services';
serviceDiscReqPDU = generateATTPDU(gattClient);

% Transmit the application data (|serviceDiscReqPDU|) to the server through
% PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(serviceDiscReqPDU);
count = count+1;
```

Receive Client request at Server

The server receives a *Read by group type request* from the client and sends the list of available services in a *Read by group type response* ATT PDU.

The receiveData object function decodes the incoming PDU as a GATT-server and returns the corresponding ATT PDU configuration object and the appropriate response PDU.

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode the received ATT PDU and generate response PDU, if applicable
[attServerRespPDU, serviceDiscReqCfg, gattServer] = receiveData(gattServer, receivedPDU);

fprintf("Received service discovery request at the server:\n")
```

```
Received service discovery request at the server:
```

```
serviceDiscReqCfg
```

```
serviceDiscReqCfg =
  bleATTPDUConfig with properties:

           Opcode: 'Read by group type request'
      StartHandle: '0001'
        EndHandle: 'FFFF'
    AttributeType: '2800'
```

```
% Transmit the application response data (|attServerRespPDU|) to the client
```

```
% through PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(attServerRespPDU);
count = count+1;
```

Receive Server response at Client

The receiveData object function decodes the incoming PDU as a GATT-client and returns the corresponding ATT PDU configuration object and the appropriate response PDU, if applicable.

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode received ATT PDU and generate response PDU, if applicable
[~, serviceDiscRespCfg] = receiveData(gattClient, receivedPDU);
gattClient.StartHandle = serviceDiscRespCfg.StartHandle;
gattClient.EndHandle = serviceDiscRespCfg.EndHandle;

% Expected response from the server: |'Read by group type response'| or
% |'Error response'|
if strcmp(serviceDiscRespCfg.Opcode, 'Error response')
    fprintf("Received error response at the client:\n")
    serviceDiscRespCfg
    serviceDiscRespMsg = ['Error response(''' serviceDiscRespCfg.ErrorMessage ''')'];
else
    fprintf("Received service discovery response at the client:\n")
    serviceDiscRespCfg
    service = helperBluetoothID.getBluetoothName(serviceDiscRespCfg.AttributeValue);
    serviceDiscRespMsg = ['Service discovery response(''' service ''')'];
end
```

```
Received service discovery response at the client:

serviceDiscRespCfg =
  bleATTPDUConfig with properties:

            Opcode: 'Read by group type response'
       StartHandle: '0001'
         EndHandle: '0006'
    AttributeValue: [2x2 char]
```

Characteristics Discovery

A service consists of multiple characteristics. For each service, there are information elements exchanged between a client and server. Each information element may contain descriptors of its behavior. A characteristic contains a value and its associated descriptors. After discovering the service, clients perform *characteristics discovery* to learn about the characteristics defined in the service. In characteristic discovery, the client invokes `'Discover all characteristics of service'` by sending 'Read by type request' ATT PDU. The server responds with the available characteristics and their associated handles by sending a 'Read by type response' ATT PDU.

Client request for characteristics at Server

```
% Configure a GATT client to discover all the available characteristics at
% the server
gattClient.SubProcedure = 'Discover all characteristics of service';
chrsticDiscReqPDU = generateATTPDU(gattClient);
```

```
% Transmit the application data (|chrsticDiscReqPDU|) to the server through
% PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(chrsticDiscReqPDU);
count = count+1;
```

Receive Client request at Server

Decodes the received request and return the list of available characteristics in a *Read by type response* ATT PDU.

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode received ATT PDU and generate response PDU, if applicable
[chrsticDiscRespPDU, chrsticDiscReqCfg, gattServer] = receiveData(gattServer, receivedPDU);

fprintf("Received characteristic discovery request at the server:\n")
```

```
Received characteristic discovery request at the server:
```

```
chrsticDiscReqCfg
```

```
chrsticDiscReqCfg =
  bleATTPDUConfig with properties:

           Opcode: 'Read by type request'
      StartHandle: '0001'
        EndHandle: '0006'
    AttributeType: '2803'
```

```
% Transmit the application response data (|chrsticDiscRespPDU|) to the
% client through PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(chrsticDiscRespPDU);
count = count+1;
```

Receive Server response at Client

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode received ATT PDU and generate response PDU, if applicable
[~, chrsticDiscRespCfg] = receiveData(gattClient, receivedPDU);

% Expected response from the server: |'Read by type response'| or |'Error
% response'|
if strcmp(chrsticDiscRespCfg.Opcode, 'Error response')
    fprintf("Received error response at the client:\n")
    chrsticDiscRespCfg
    chrsticDescRespMsg = ['Error response(''' chrsticDiscRespCfg.ErrorMessage ''')'];
else
    fprintf("Received characteristic discovery response at the client:\n")
    attributeValueCfg = helperBLEDecodeAttributeValue(...
        chrsticDiscRespCfg.AttributeValue, 'Characteristic');
    attributeValueCfg
```

```
    chrsticDescRespMsg = ['Characteristic discovery response(''' attributeValueCfg.Characteristi
end

Received characteristic discovery response at the client:

attributeValueCfg =
  helperBLEAttributeValueConfig with properties:

                    AttributeType: 'Characteristic'
                    BroadcastFlag: 'False'
                         ReadFlag: 'False'
          WriteWithoutResponseFlag: 'False'
                        WriteFlag: 'False'
                       NotifyFlag: 'True'
                     IndicateFlag: 'False'
    AuthenticatedSignedWritesFlag: 'False'
           ExtendedPropertiesFlag: 'False'
        CharacteristicValueHandle: '0003'
               CharacteristicType: 'Heart rate measurement'
```

Characteristic Descriptor Discovery

A characteristic may consists of multiple characteristic descriptors. After discovering the characteristic, clients perform *characteristic descriptors discovery* to learn about the list of descriptors and their handles. In characteristic descriptor discovery, the client invokes `'Discover all descriptors'` by sending 'Information request' ATT PDU. The server responds with the available characteristic descriptors and their associated handles by sending a 'Information response' ATT PDU.

Client request for characteristic descriptors at Server

```
% Configure a GATT client to discover all the available characteristic
% descriptors at the server
gattClient.SubProcedure = 'Discover all descriptors';
gattClient.StartHandle = dec2hex(hex2dec(chrsticDiscRespCfg.AttributeHandle)+1, 4);
chrsticDescDiscReqPDU = generateATTPDU(gattClient);

% Transmit the application data (|chrsticDescDiscReqPDU|) to the client
% through PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(chrsticDescDiscReqPDU);
count = count+1;
```

Receive Client request at Server

Decodes the received request and returns the list of available characteristic descriptors in a *Information response* ATT PDU.

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode received ATT PDU and generate response PDU, if applicable
[chrsticDescDiscRespPDU, chrsticDescDiscReqCfg, gattServer] = receiveData(gattServer, receivedPDU

fprintf("Received characteristic descriptor discovery request at the server:\n")

Received characteristic descriptor discovery request at the server:
```

```
chrsticDescDiscReqCfg
```

```
chrsticDescDiscReqCfg =
  bleATTPDUConfig with properties:

         Opcode: 'Information request'
    StartHandle: '0003'
      EndHandle: '0006'
```

```
% Transmit the application response data (|chrsticDescDiscRespPDU|) to the
% client through PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(chrsticDescDiscRespPDU);
count = count+1;
```

Receive Server response at Client

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode received ATT PDU and generate response PDU, if applicable
[~, chrsticDescDiscRespCfg] = receiveData(gattClient, receivedPDU);

% Expected response from the server: |'Information response'| or |'Error
% response'|
if strcmp(chrsticDescDiscRespCfg.Opcode, 'Error response')
    fprintf("Received error response at the client:\n")
    chrsticDescDiscRespCfg
    chrsticDescDiscRespMsg = ['Error response(''' chrsticDescDiscRespCfg.ErrorMessage ''')'];
else
    fprintf("Received characteristic descriptor discovery response at the client:\n")
    chrsticDescDiscRespCfg
    descriptor = helperBluetoothID.getBluetoothName(chrsticDescDiscRespCfg.AttributeType);
    chrsticDescDiscRespMsg = ['Characteristic descriptor discovery response(''' descriptor ''')'
end
```

```
Received characteristic descriptor discovery response at the client:
```

```
chrsticDescDiscRespCfg =
  bleATTPDUConfig with properties:

            Opcode: 'Information response'
            Format: '16 bit'
    AttributeHandle: '0004'
      AttributeType: '2902'
```

Subscribe for Notifications

After discovering the characteristic descriptors, the client may enable or disable *notifications* for its characteristic value. To enable notifications, the client must set the notification bit (first bit) of *Client Characteristic Configuration Descriptor (CCCD)* value by invoking `'Write characteristic value'` sub-procedure.

Client subscribe for notifications at Server

```
% Configure a GATT client to enable the notifications of Heart rate
% measurement characteristic
```

```
gattClient.SubProcedure = 'Write characteristic value';
gattClient.AttributeHandle = chrsticDescDiscRespCfg.AttributeHandle;
gattClient.AttributeValue = '0100';
enableNotificationReqPDU = generateATTPDU(gattClient);

% Transmit the application data (|enableNotificationReqPDU|) to the client
% through PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(enableNotificationReqPDU);
count = count+1;
```

Receive Client request at Server

Decodes the received request and sends the response in a *Write response* ATT PDU.

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode received ATT PDU and generate response PDU, if applicable
[enableNotificationRespPDU, enableNotificationReqCfg, gattServer] = receiveData(gattServer, rece

fprintf("Received enable notification request at the server:\n")
```

```
Received enable notification request at the server:
```

```
enableNotificationReqCfg
```

```
enableNotificationReqCfg =
  bleATTPDUConfig with properties:

             Opcode: 'Write request'
    AttributeHandle: '0004'
     AttributeValue: [2x2 char]
```

```
% Transmit the application response data (|enableNotificationRespPDU|) to
% the client through PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(enableNotificationRespPDU);
count = count+1;
```

Receive Server response at Client

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode received ATT PDU and generate response PDU, if applicable
[~, enableNotificationRespCfg] = receiveData(gattClient, receivedPDU);

% Expected response from the server: |'Write response'| or |'Error
% response'|
if strcmp(enableNotificationRespCfg.Opcode, 'Error response')
    fprintf("Received error response at the client:\n")
    enableNotificationRespCfg
    enableNotificRespMsg = ['Error response(''' enableNotificationRespCfg.ErrorMessage ''')'];
else
    fprintf("Received enable notification response at the client:\n")
    enableNotificationRespCfg
```

```
    enableNotificRespMsg = 'Notifications enabled(''Heart rate measurement '')';
end
```

```
Received enable notification response at the client:
```

```
enableNotificationRespCfg =
  bleATTPDUConfig with properties:

    Opcode: 'Write response'
```

Notifying the Heart Rate Measurement Value to the Client

When a client enables notifications for a characteristic, the server periodically notifies the value of characteristic (*Heart rate measurement*) to the client.

The HRP server notifies heart rate measurement to the client after its subscription.

Server sends notifications to Client

The notifyHeartRateMeasurement object function generates notification PDU as specified in the Bluetooth core specification.

```
% Reset the random number generator seed
rng default

% Measure heart rate value using sensor (generate a random number for heart
% rate measurement value)
heartRateMeasurementValue = randi([65 95]);

% Notify the heart rate measurement
[gattServer, notificationPDU] = notifyHeartRateMeasurement(gattServer, ...
    heartRateMeasurementValue);

% Transmit the application data (|notificationPDU|) to the client through
% PHY
[bleWaveform, pcapPackets{count}] = helperBLETransmitData(notificationPDU);
count = count+1;
```

Receive Server notifications at Client

```
% Decode the received Bluetooth LE waveform and retrieve the application
% data
receivedPDU = helperBLEDecodeData(bleWaveform);

% Decode received ATT PDU and generate response PDU, if applicable
[~, notificationCfg] = receiveData(gattClient, receivedPDU);

fprintf("Received notification at the client:\n")
```

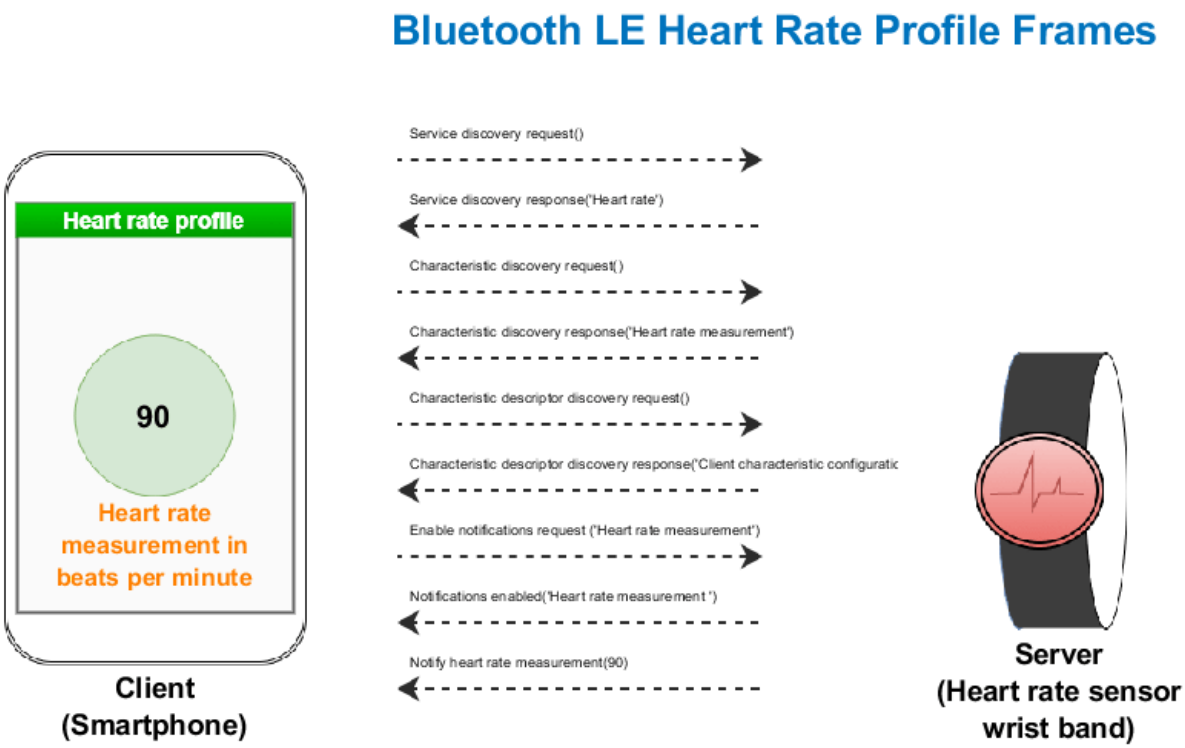
```
Received notification at the client:
```

```
% Decode the received heart rate measurement characteristic value
heartRateCharacteristicValue = helperBLEDecodeAttributeValue(...
    notificationCfg.AttributeValue, 'Heart rate measurement');
heartRateCharacteristicValue
```

```
heartRateCharacteristicValue =
  helperBLEAttributeValueConfig with properties:
```

```
              AttributeType: 'Heart rate measurement'
         HeartRateValueFormat: 'UINT8'
          SensorContactStatus: 'Contact detected'
      EnergyExpendedFieldFlag: 'Present, Units: Kilo Joules'
          RRIntervalFieldFlag: 'Present'
               HeartRateValue: 90
               EnergyExpended: 100
                   RRInterval: 10
```

```
heartRateMeasurementValue = heartRateCharacteristicValue.HeartRateValue;

% Visualize the Bluetooth LE GATT Client-Server model
helperBLEVisualizeHRPFrame(serviceDiscRespMsg, chrsticDescRespMsg, ...
    chrsticDescDiscRespMsg, enableNotificRespMsg, heartRateMeasurementValue);
```



Exporting to a PCAP File

This example uses `blePCAPWriter` object to export the generated PDUs to a file with `.pcap` extension or `.pcapng` extension. To analyze and visualize this file, use a third part packet analyzer such as Wireshark.

Create an object of type `blePCAPWriter` and specify the packet capture file name.

```
% Create the Bluetooth LE PCAP Writer file object
pcapObj = blePCAPWriter("FileName", "bluetoothLEHRP");
```

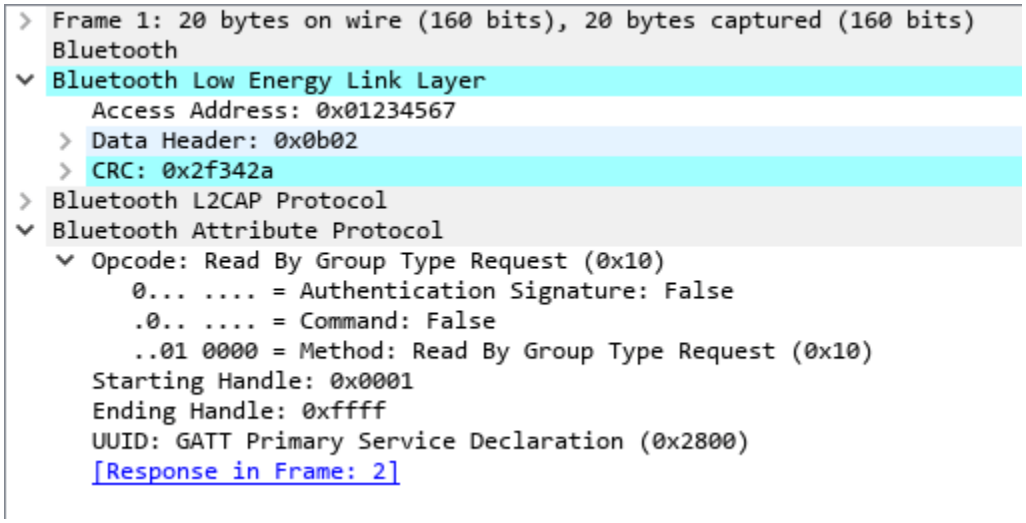
Use the `write` object function to write all the Bluetooth LE LL PDUs to a PCAP file. The constant `timestamp` specifies the capture time of a PDU. In this example, the capture time is same for all the PDUs.

```
timestamp = 124800; % timestamp (in microseconds)

% Write all the LL PDUs to the PCAP file
for idx = 1:numel(pcapPackets)
    write(pcapObj, pcapPackets{idx}, timestamp, "PacketFormat", "bits");
end

% Clear the object
clear pcapObj;

fprintf("Open generated pcap file 'bluetoothLEHRP.pcap' in a protocol analyzer to view the genera

Open generated pcap file 'bluetoothLEHRP.pcap' in a protocol analyzer to view the generated frame
```

Visualization of the Generated ATT PDUs

Since the generated heart rate profile packets are compliant with the Bluetooth standard, you can open, analyze and visualize the PCAP file using a third party packet analyzer such as Wireshark [4 on page 6-0]. The data shown in these figures uses the heart rate profile packets generated in this example.

- **Service discovery request**



- **Service discovery response**

```
> Frame 2: 21 bytes on wire (168 bits), 21 bytes captured (168 bits)
  Bluetooth
v Bluetooth Low Energy Link Layer
    Access Address: 0x01234567
  > Data Header: 0x0c02
  > CRC: 0x615414
> Bluetooth L2CAP Protocol
v Bluetooth Attribute Protocol
  v Opcode: Read By Group Type Response (0x11)
      0... .... = Authentication Signature: False
      .0.. .... = Command: False
      ..01 0001 = Method: Read By Group Type Response (0x11)
    Length: 6
  > Attribute Data, Handle: 0x0001, Group End Handle: 0x0006, UUID: Heart Rate
    [UUID: GATT Primary Service Declaration (0x2800)]
    [Request in Frame: 1]
```

- **Notifying heart rate measurement value**

```
> Frame 9: 22 bytes on wire (176 bits), 22 bytes captured (176 bits)
  Bluetooth
v Bluetooth Low Energy Link Layer
    Access Address: 0x01234567
  > Data Header: 0x0d02
  > CRC: 0xaa0ddb
> Bluetooth L2CAP Protocol
v Bluetooth Attribute Protocol
  v Opcode: Handle Value Notification (0x1b)
      0... .... = Authentication Signature: False
      .0.. .... = Command: False
      ..01 1011 = Method: Handle Value Notification (0x1b)
  v Handle: 0x0003 (Heart Rate: Heart Rate Measurement)
      [Service UUID: Heart Rate (0x180d)]
      [UUID: Heart Rate Measurement (0x2a37)]
  v Flags: 0x1e, RR Interval, Energy Expended, Sensor Support, Sensor Contact
      000. .... = Reserved: 0x0
      ...1 .... = RR Interval: True
      .... 1... = Energy Expended: True
      .... .1.. = Sensor Support: True
      .... ..1. = Sensor Contact: True
      .... ...0 = Value is UINT16: False
    Value: 90
    Energy Expended: 100
  v RR Intervals [count =  1]
      RR Interval: 10
```

Conclusion

This example demonstrated the modeling of Bluetooth LE devices with Heart Rate Profile using the GATT client-server scenario as specified in the Bluetooth core specification [2 on page 6-0]. You can use a packet analyzer to view the generated frames.

Appendix

The example uses these helpers:

- helperBLEGATTClient: Provide methods to the Generic Attribute profile
- helperBLEGATTServer: Create a GATT server object
- helperBLEAttributeValueConfig: Create configuration object for a Bluetooth LE attribute value
- helperBLEGenerateAttributeValue: Bluetooth LE attribute value generation
- helperBLEDecodeAttributeValue: Bluetooth LE attribute value decoder
- helperBLEDecodeData: Decode the received waveform and retrieve the application data
- helperBLETransmitData: Transmit application data by generating Bluetooth LE waveform
- helperBluetoothID: Bluetooth identifiers and their names assigned by the Bluetooth Special Interest Group (SIG)
- helperBLEVisualizeHRPFrame: Visualize the Heart Rate Profile (HRP) frames exchanged in HRP Example
- helperBLEPlotHRPFrames: Plot the data frame exchange between the Heart rate profile server and client

Selected Bibliography

1. Bluetooth® Technology Website. "Bluetooth Technology Website | The Official Website of Bluetooth Technology." Accessed November 29, 2021. https://www.bluetooth.com/.
2. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification" Version 5.3. https://www.bluetooth.com/.
3. Bluetooth Special Interest Group (SIG). "Heart Rate Profile" Version 1.0. https://www.bluetooth.com/.
4. “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed November 29, 2021. https://www.wireshark.org.
5. Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed November 29, 2021. https://www.tcpdump.org.
6. Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2021. https://www.ietf.org/.

See Also

Functions

`bleATTPDU` | `bleATTPDUDecode`

Objects

`bleATTPDUConfig` | `blePCAPWriter`

More About

- “Generate and Decode Bluetooth Protocol Data Units” on page 9-2
- “Bluetooth LE L2CAP Frame Generation and Decoding” on page 6-30
- “Bluetooth LE L2CAP Frame Generation and Decoding” on page 6-30

Evaluate the Performance of Bluetooth QoS Traffic Scheduling with WLAN Signal Interference

This example shows you how to evaluate the performance of the Bluetooth® scheduler by implementing multiple applications with different quality-of-service (QoS) requirements (throughput and latency) in a use-case scenario. Using this example, you can:

- Create and configure a use-case scenario of a home environment showing multiple Bluetooth applications in a piconet with WLAN interference.
- Emulate and configure the application traffic pattern by using the generic On-Off traffic model.
- Implement round-robin (RR) and QoS-based priority schedulers to schedule application traffic.
- Add your own custom scheduler.
- Specify the source of WLAN interference by adding the WLAN signal using the features of WLAN Toolbox™ or from a baseband file.
- Evaluate the performance of each Peripheral in the presence of a synchronous connection-oriented (SCO) link and by varying the scheduler.

The example supports adaptive frequency hopping (AFH) by classifying channels as good or bad based on the packet error rate (PER) of each channel. Visualize the power spectral density of Bluetooth waveforms with a WLAN signal interference using the Spectrum Analyzer.

Bluetooth Logical Transports and Application Profiles

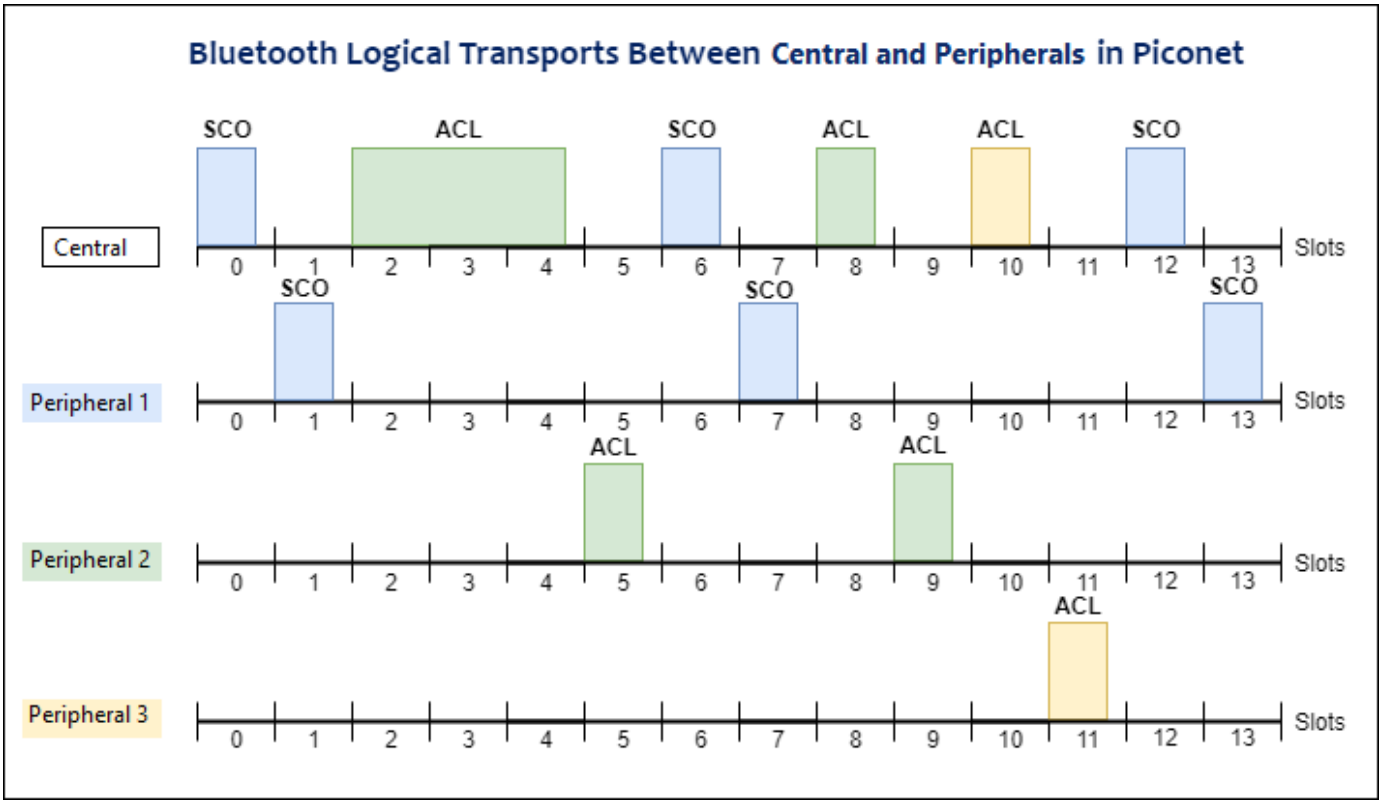
"Bluetooth Technology Overview" on page 8-2 supports communication over multiple logical transports with different applications running on it. These applications include audio streaming, gaming controls, wireless peripherals, and file transfer applications. Because multiple applications can exist in a Bluetooth piconet, different types of application traffic flow from the higher layers to the baseband.

Logical Transports

In a Bluetooth piconet, the Central and Peripheral exchange data over multiple logical transports. These logical transports are:

- Asynchronous connection-oriented (ACL)
- SCO
- Extended synchronous connection-oriented (eSCO)
- Active peripheral broadcast (APB)
- Connectionless peripheral broadcast (CPB)

This figure shows the communication between a Central and three Peripherals in a piconet over ACL and SCO logical transports. Because Bluetooth is a Central-driven time division duplex (TDD) system, the channel access in the piconet is controlled by the Central. The Peripheral can respond to only a transmission from the Central in the previous Tx slot. This process is called *polling*.



The Central polls a Peripheral with a poll packet (if no data exists) or a data packet in a Tx slot, and the Peripheral responds to the polling. The Central can poll any Peripheral of the SCO or ACL logical transport. For SCO links, the Central reserves the slots for the dedicated SCO Peripheral. The Central polls the ACL Peripherals in the remaining slots. The Peripheral responds to the Central with a data packet or a null packet.

Application Profiles

Bluetooth profiles (often called application profiles) are definitions of possible applications and specify general behaviors that Bluetooth-enabled devices use to communicate with other Bluetooth devices. The Bluetooth Special Interest Group (SIG) [2] on page 6-0 defines these profiles and the possible applications of each profile.

In general, Bluetooth application traffic can be categorized into these three classes.

- Streaming - These applications have latency and bandwidth requirements. For example, a headphone or a laptop.
- Human interface devices (HID) - These applications have low latency requirements. For example, a keyboard or a joystick.
- Best-effort traffic - These applications have no latency requirements. For example, file transfer using Bluetooth.

Each Peripheral corresponds to a specific application profile. Typically, the amount of traffic flow in each application profile varies. Also, the throughput and latency performance requirements of each application profile are different. In such scenarios, if the implementation uses an RR scheduler for polling the ACL Peripherals, and if the polled Peripherals do not have data to transmit, this results in

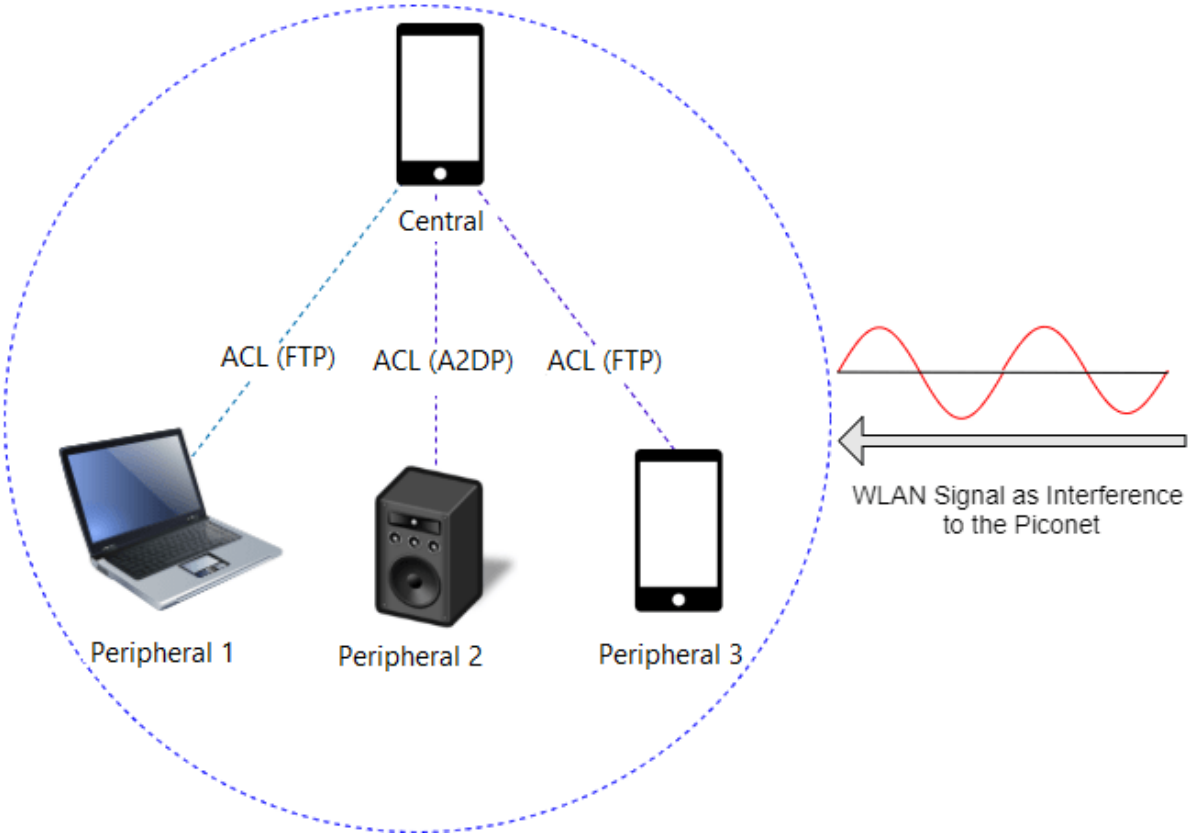
low bandwidth utilization by Peripherals. To help improve performance, prioritize the ACL Peripherals for polling based on the QoS requirements.

In this example, the Bluetooth nodes operate with the basic rate (BR) physical layer (PHY) and communicate with each other simultaneously by transmitting the application data packets (on the ACL link). This example simulates a use-case scenario consisting of multiple Bluetooth devices that communicate by emulating the traffic pattern of FTP and A2DP application profiles. For more details, see Use-Case Scenario on page 6-0 .

Use-Case Scenario

This example shows a use-case scenario of a home environment, where a smartphone (Central) connects to a laptop (Peripheral 1), wireless speaker (Peripheral 2), and smartphone (Peripheral 3). All of the devices are Bluetooth-BR-enabled. You can enable or disable presence of static WLAN signal interference in the vicinity of the Bluetooth piconet.

Use Case Scenario Showing Multiple Bluetooth Applications in a Piconet With WLAN Signal Interference



In the preceding figure:

- The Central transfers a file to the laptop (Peripheral 1) on the ACL link by emulating the FTP traffic pattern.

- The Central streams music in the wireless speaker (Peripheral 2) on the ACL link by emulating the A2DP traffic pattern.
- The Central transfers another file smartphone (Peripheral 3) on the ACL link by emulating the FTP traffic pattern.

In the preceding scenario, the performance of each Peripheral degrades due to these reasons.

- Concurrent communication by various Bluetooth applications
- Presence of WLAN signal interference in the wireless medium (if enabled)

This example shows how to simulate this use-case scenario and how to measure the communication performance. To communicate with the laptop (Peripheral 1), wireless speaker (Peripheral 2), and smartphone (Peripheral 3) on the ACL links, the Central uses the RR scheduling mechanism. The RR scheduling mechanism provides equal transmission and reception opportunities for Peripheral 1, Peripheral 2, and Peripheral 3. Because, Peripheral 2 has QoS requirements (related to throughput and latency), Peripheral 2 must be prioritized. The RR scheduling mechanism fails to prioritize the communication opportunities of Peripheral 2, resulting in the degradation of communication performance. To mitigate the performance degradation at Peripheral 2 and help improve performance the example uses the QoS-based priority scheduling mechanism at the Central. The example also shows how to add a custom scheduling algorithm, enabling you to schedule application traffic with specific performance requirements.

Configure Simulation Parameters

Configure the simulation parameters for the Bluetooth piconet, the application traffic, the wireless channel, and the WLAN signal interference.

Configure Bluetooth Piconet

The `NumPeripherals` parameter specifies the number of Peripherals in the Bluetooth piconet. The `LinkTraffic` parameter specifies the type of traffic over Bluetooth logical transports between a Central and the respective Peripheral. This table maps `LinkTraffic` to different logical transports. If the Central communicates with multiple Peripherals, `LinkTraffic` must be a vector.

`linkTraffic` Value	Logical Transport
1	ACL
2	SCO
3	ACL and SCO

```
% Set the simulation time in microseconds
simulationTime = 2*1e6;

% Specify to enable (true) or disable (false) visualization
enableVisualization = true;

simulationParameters = struct;
% Configure the number of Peripherals in the piconet
```

```
simulationParameters.NumPeripherals = 3 ;

% Configure the logical links between the Central and Peripherals. Each element
% represents the logical link between the Central and the respective Peripheral.
% If the Central is connected to multiple Peripherals, this value must be a row
% vector.
simulationParameters.LinkTraffic = [1 1 1];

% Specify the positions of Bluetooth nodes in the form of an n-by-3 array,
% where n is the number of nodes in the piconet. Each row specifies the
% cartesian coordinates of a node starting from the Central and followed by
% the Peripherals.
simulationParameters.NodePositions = [10 0 0; 20 0 0; 30 0 0; 40 0 0];

% Configure the frequency hopping sequence as Connection adaptive (for
% enabling AFH) or Connection basic
simulationParameters.SequenceType = Connection adaptive ;
```

Configure Application Traffic Pattern

This example shows how to use a generic On-Off model for emulating the application traffic pattern. To emulate the application traffic pattern of the ACL Peripherals (Peripheral 1, Peripheral 2, and Peripheral 3), use the `helperBluetoothNetworkTraffic`. To model the traffic pattern, specify the application token rate (bit rate), frame size, access latency, and the values for `OnTime` and `OffTime` properties of the object. If you specify SCO Peripherals, model the application traffic pattern based on the packet type. For each transmission, the data (random bits) is made available at the baseband layer. In the majority of the scenarios involving ACL links, the traffic flow is present at the source, and the sink sends only the acknowledgement. In this example, all of the sources are present at the Central.

```
% Load the Bluetooth application traffic pattern configuration
load('bluetoothTrafficConfig.mat');

% Specify the parameters for SCO application. To enable an SCO logical
% transport, set the linkTraffic value at the Peripheral index to 2 or 3.
% Specify the SCO packet type as 'HV1', 'HV2', or 'HV3' for the respective
% Peripheral that has SCO link traffic. Index 1 represents the Peripheral
% number, and index 2 represents the SCO packet type to be used by the
% Peripheral.
simulationParameters.SCOPacketType = {};

% Compute the number of ACL Peripherals
numACLPeripherals = nnz(simulationParameters.LinkTraffic ~= 2);
aclApplications = repmat({bluetoothTrafficConfig}, 1, numACLPeripherals);

% Update the configuration at the first ACL application (for example, a
% laptop). You can observe the variation in throughput based on the
% configured data rate and packet size.
aclApplications{1}.PeripheralNumber = 1;            % Peripheral LT address
aclApplications{1}.TrafficPattern = 'FTP';          % FTP traffic pattern
aclApplications{1}.PacketSize = 152;                % In bytes
aclApplications{1}.DataRateKbps = 224;              % In Kbps
aclApplications{1}.ApplicationOnTime = 2;           % In milliseconds
aclApplications{1}.ApplicationOffTime = 0;          % In milliseconds
aclApplications{1}.AccessLatency = 500;             % In milliseconds
```

```
aclApplications{1}.Role = 'Central';                % Install the application at 'Central', 'Peri

% Update the configuration at the second ACL application (for example, a
% wireless speaker).
aclApplications{2}.PeripheralNumber = 2;            % Peripheral LT address
aclApplications{2}.TrafficPattern = 'A2DP';         % A2DP traffic pattern
aclApplications{2}.PacketSize = 328;                % In bytes
aclApplications{2}.DataRateKbps = 237;              % In Kbps
aclApplications{2}.ApplicationOnTime = 2;           % In milliseconds
aclApplications{2}.ApplicationOffTime = 0;          % In milliseconds
aclApplications{2}.AccessLatency = 60;              % In milliseconds
aclApplications{2}.Role = 'Central';                % Install the application at 'Central', 'Peri

% Update the configuration at the third ACL application (for example, a
% smartphone).
aclApplications{3}.PeripheralNumber = 3;            % Peripheral LT address
aclApplications{3}.TrafficPattern = 'FTP';          % FTP traffic pattern
aclApplications{3}.PacketSize = 164;                % In bytes
aclApplications{3}.DataRateKbps = 172;              % In Kbps
aclApplications{3}.ApplicationOnTime = 2;           % In milliseconds
aclApplications{3}.ApplicationOffTime = 0;          % In milliseconds
aclApplications{3}.AccessLatency = 500;             % In milliseconds
aclApplications{3}.Role = 'Central';                % Install the application at 'Central', 'Peri

% Configure the application traffic objects
for appIdx = 1:numACLPeripherals
    appCfg = aclApplications{appIdx};
    app = helperBluetoothNetworkTraffic( ...
        'PacketSize',appCfg.PacketSize, ...     % In bytes
        'DataRate',appCfg.DataRateKbps, ...     % In Kbps
        'OnTime',appCfg.ApplicationOnTime, ...  % In milliseconds
        'OffTime',appCfg.ApplicationOffTime);   % In milliseconds
    aclApplications{appIdx}.AppTrafficPattern = app;
end
simulationParameters.ACLApplications = aclApplications;
```

Configure Scheduler

Configure the scheduler to be used at the baseband layer. To configure the RR and priority scheduler, use the `helperBluetoothRRScheduler` and `helperBluetoothPriorityScheduler` helper objects, respectively. You can also add a custom scheduler at the baseband layer. For information about how to implement and integrate a custom scheduler, see Compare Performance of Peripherals by varying Scheduling Algorithm on page 6-0 .

```
% Specify the scheduling scheme as 'RR' or 'Priority'
simulationParameters.Scheduler = Priority;
```

Configure Wireless Channel and WLAN Signal Interference

Configure the wireless channel by using the `helperBluetoothChannel` helper object and set the `SIR` value at each node. You can set the `EbNo` value for the AWGN channel. The AWGN is present throughout the simulation.

To generate the WLAN signal interference, use the `helperBluetoothGenerateWLANWaveform` helper function. Specify the sources of WLAN interference by using the `WLANInterference` parameter. Use one of these options to specify the source of the WLAN interference.

- `'Generated'` - To add a WLAN toolbox™ signal to interfere the communication between Bluetooth nodes, select this option. For details on how to add this signal, follow the steps shown in Add WLAN Signal Using WLAN Toolbox™ Features on page 6-0 .
- `'BasebandFile'` - To add a WLAN signal from a baseband file (`.bb`) to interfere the communication between Bluetooth nodes, select this option. You can specify the file name using the `WLANBBFilename` input argument. If you do not specify the `.bb` file, the example uses the default `.bb` file, `'WLANNonHTDSSS.bb'`, to add the WLAN signal.

The `'None'` option specifies that no WLAN signal is added to the Bluetooth signals, and the example uses this option by default.

```
% Configure wireless channel parameters
simulationParameters.EbNo = 22; % Ratio of energy per bit (Eb) to spectral noise density (No) in

% Configure the WLAN interference. Specify the WLAN interference as
% Generated, BasebandFile, or None. To use the wlanBBFilename property, set
% wlanInterference to BasebandFile.
simulationParameters.WLANInterference = None;
simulationParameters.WLANBBFilename =  'WLANNonHTDSSS.bb';

% Specify the signal to interference ratio, in dB, at each node. Specify
% this value as an n-element vector, where n is the number of nodes in the
% piconet, starting from the Central and followed by the Peripherals.
% Each value indicates the SIR at a node.
simulationParameters.SIR = [-12 -6 -10 -8];
```

Create Bluetooth Piconet

Create a Bluetooth piconet of nodes with an L2CAP layer, baseband layer, PHY, and channel. To configure the Bluetooth piconet from the configured parameters, use the `helperBluetoothCreatePiconet` helper function.

```
% Set the default random number generator ('twister' type with seed value 0).
% The seed value controls the pattern of random number generation. For high
% fidelity simulation results, change the seed value and average the
% results over multiple simulations.
rng('default');

% Specify the Tx power in dBm
simulationParameters.TxPower = 20;

% Specify the Bluetooth node receiver range (in meters).
simulationParameters.ReceiverRange = 40;

% Configure the channel classification parameters
simulationParameters.PERThreshold = 40;
simulationParameters.ClassificationInterval = 3000;
simulationParameters.RxStatusCount = 10;
simulationParameters.MinRxCountToClassify = 4;
simulationParameters.PreferredMinimumGoodChannels = 20;

% Set the total number of nodes in the piconet (one Central and multiple
% Peripherals)
numNodes = simulationParameters.NumPeripherals + 1;
```

```
% Configure the Bluetooth piconet
bluetoothNodes = helperBluetoothCreatePiconet(simulationParameters);
```

Visualize the Bluetooth waveforms by using the `dsp.SpectrumAnalyzer` System object™.

```
if enableVisualization
    spectrumAnalyzer = dsp.SpectrumAnalyzer( ...
        'ViewType','Spectrum and spectrogram', ...
        'TimeResolutionSource','Property', ...
        'TimeResolution',0.0005, ...                                    % In seconds
        'TimeSpanSource','Property', ...
        'TimeSpan',0.05, ...                                            % In seconds
        'FrequencyResolutionMethod','WindowLength', ...
        'WindowLength',512, ...                                         % In samples
        'AxesLayout','Horizontal', ...
        'FrequencyOffset',2441*1e6, ...                                 % In Hz
        'SampleRate',bluetoothNodes{1}.PHY.SamplesPerSymbol*1e6, ... % In Hz
        'ColorLimits',[-20 15]);
end
```

Simulation

Run the Bluetooth piconet simulation by calling each node instance. Distribute the Tx packets from each node to the Rx buffer of the other nodes, and then advance the simulation time to the next event of a node. Update the spectrum analyzer visualization.

```
% Specify the current simulation time, elapsed time, and next invoke times
% for all of the nodes in microseconds
curTime = 0;
elapsedTime = 0;
nextInvokeTimes = zeros(1, numel(bluetoothNodes));
slotPerSec = 1600;

% View buffer size at each ACL Peripheral
if enableVisualization
    channelNames = cell(1, numACLPeripherals);
    for idx=1:numACLPeripherals
        channelNames{idx} = bluetoothNodes{idx+1}.NodeName;
    end
    bufferSizeVisualization = timescope( ...
        'Name','Buffer Size of each ACL Peripheral at Central', ...
        'SampleRate',slotPerSec*1.25*simulationTime/1e6, ... % In Hz
        'AxesScaling','auto', ...
        'ShowLegend',true, ...
        'TimeSpanSource','property', ...
        'TimeSpan',simulationTime/1e6, ...                  % In seconds
        'YLabel','Buffer Size', ...
        'YLimits',[0 5], ...
        'ChannelNames',channelNames);
    bufferSizeVisualization(zeros(1, numACLPeripherals));
    if ~strcmpi(simulationParameters.WLANInterference, 'None')
        % Generate the WLAN waveform for visualization
        wlanWaveform = helperBluetoothGenerateWLANWaveform(...
            simulationParameters.WLANInterference, simulationParameters.WLANBBFilename);
    end
end

% Run the simulation
```

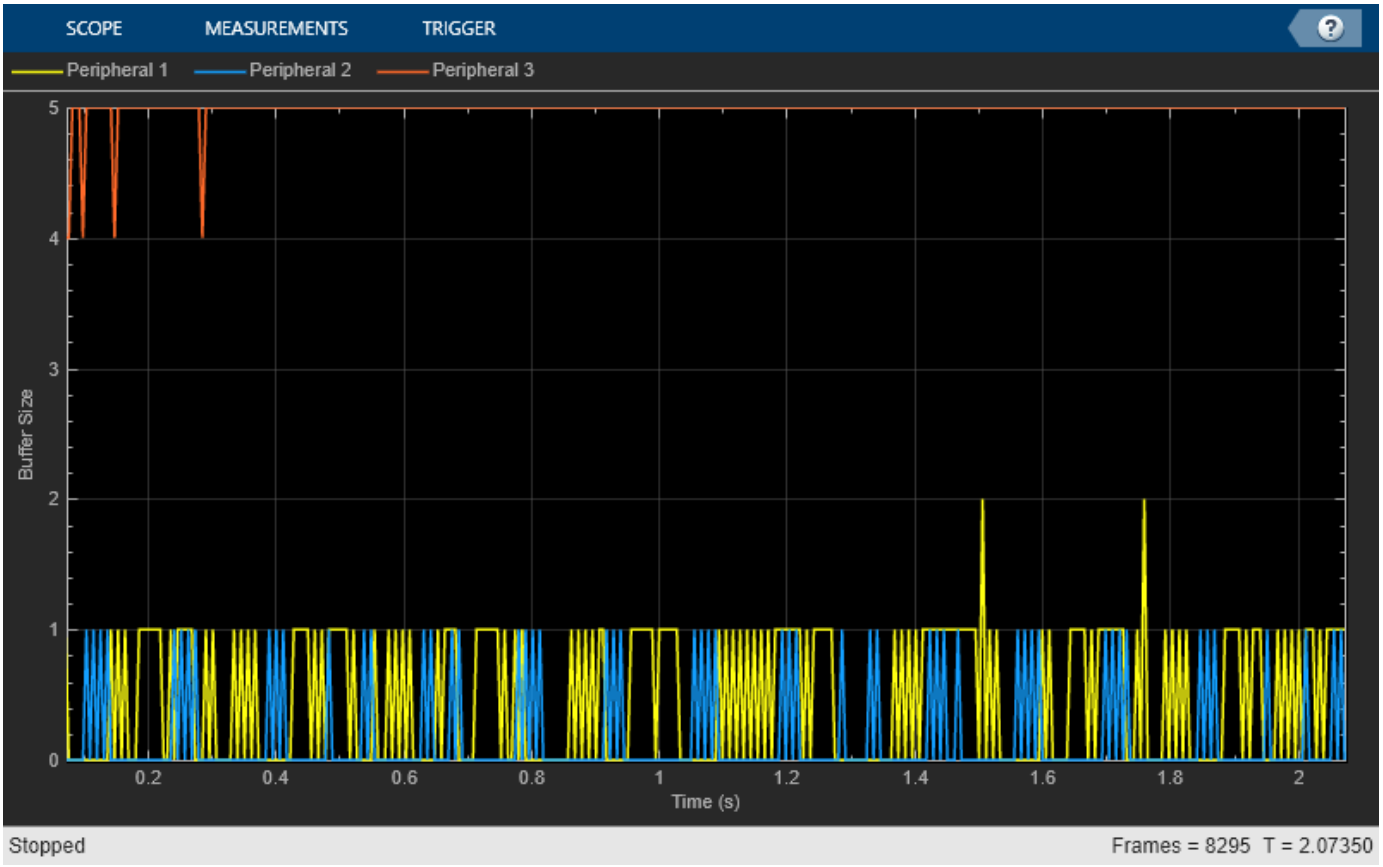
```
while(curTime < simulationTime)
    % Simulate the Bluetooth nodes
    for nodeIdx = 1:numel(bluetoothNodes)
        % Run the Bluetooth node instance
        nextInvokeTimes(nodeIdx) = runNode(bluetoothNodes{nodeIdx}, elapsedTime);

        % Update the visualization
        if enableVisualization
            % Update and plot the buffer status at the Central
            if nodeIdx == 1
                [~, emptySlots] = bluetoothNodes{1}.Baseband.aclLinkAddress;
                bufferSize = helperBluetoothLogicalTransports.MaxQueueSize - emptySlots;
                bufferSizeVisualization(bufferSize);
            end
        end
    end

    % Advance the current time by the elapsed time
    curTime = curTime + elapsedTime;

    % Distribute any transmitted packets from each node into the receiving
    % buffers of the other nodes
    [isPacketDistributed, txBuffer] = helperBluetoothDistributePackets(bluetoothNodes);

    % Update visualization
    if enableVisualization
        for txIdx = 1:numel(txBuffer)
            channelWaveform = txBuffer{txIdx}.Waveform(1:txBuffer{txIdx}.NumSamples);
            if ~strcmpi(simulationParameters.WLANInterference, 'None')
                % Add WLAN interference to the channel waveform.
                channelWaveform = channelWaveform + wlanWaveform(1:txBuffer{txIdx}.NumSamples);
            end

            % Plot the Bluetooth waveform.
            spectrumAnalyzer(channelWaveform);
        end
    end

    % If packets are distributed to the receiver nodes, run the nodes to
    % check the packet reception buffer
    if isPacketDistributed
        elapsedTime = 0;
        % Advance the simulation time to the next event at a node
    else
        elapsedTime = min(nextInvokeTimes(nextInvokeTimes ~= -1));
    end
end
```

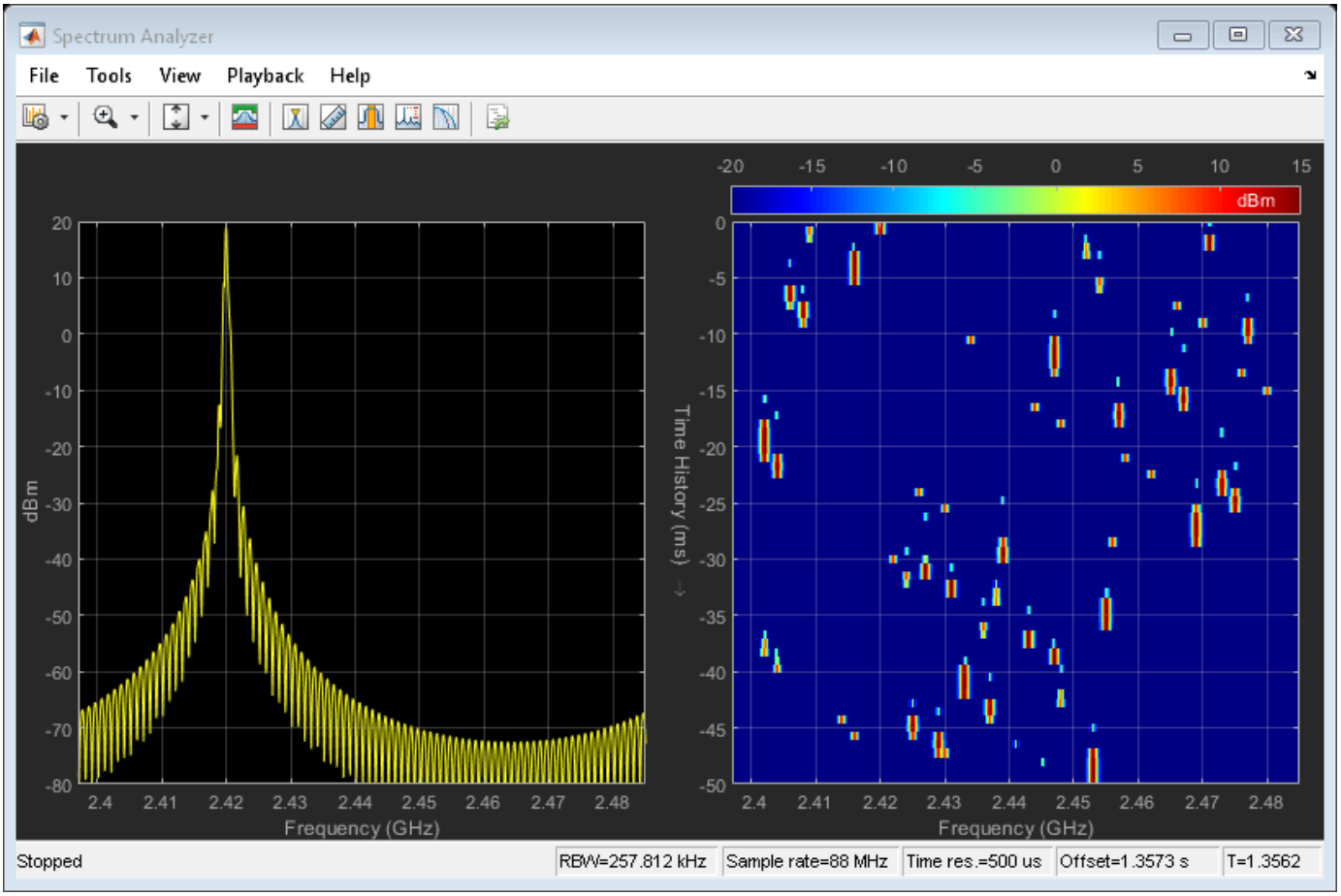
```
ans = 1x79 logical array

   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

```
ans = 1x79 logical array

   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

```
ans = 1x79 logical array

   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

```
% Release the System objects
if enableVisualization
    release(bufferSizeVisualization);
    release(spectrumAnalyzer);
end
```





The preceding Spectrum Analyzer plot shows the spectrum of the Bluetooth waveform distorted with WLAN signal interference (in the frequency domain) and passed through the AWGN channel. The right-side plot shows the overlapping of Bluetooth packets with the interfering WLAN signal. The WLAN waveform in the plot is present throughout the simulation.

Simulation Results

At each node, the simulation measures these metrics.

- Throughput at application (L2CAP)
- PER
- Bit error rate (BER)
- Packet statistics at the PHY, baseband layer, and L2CAP layer

In addition to those metrics, at each Peripheral, the simulation measures these metrics.

- Latency (only for ACL Peripherals)
- Fairness of scheduler

These metrics are generated during the simulation and stored in the `centralStats` and `peripheralsStats` table. For more information on these statistics, refer `helperBluetoothSchedulingStatistics` helper function. Get the metrics of each Bluetooth Peripheral in the piconet.

```
[centralStats, peripheralsStats] = helperBluetoothSchedulingStatistics(bluetoothNodes, ...
    simulationParameters.LinkTraffic, simulationParameters.SCOPacketType, simulationTime)
```

```
centralStats=1×11 table
               Throughput (Kbps)    PER    BER    TotalTxPackets    TotalRxPackets    ValidRxPacl
               _________________    ___    ___    ______________    ______________    ___________

    Central         569.41           0      0          712               711               711


peripheralsStats=3×17 table
                         Traffic Pattern    Application Data Rate (Kbps)    Throughput (Kbps)
                         _______________    ____________________________    _________________

    Peripheral1 (ACL)        "FTP"                    224                      223.74
    Peripheral2 (ACL)        "A2DP"                   237                      237.47
    Peripheral3 (ACL)        "FTP"                    172                       104.3
```

Further Exploration

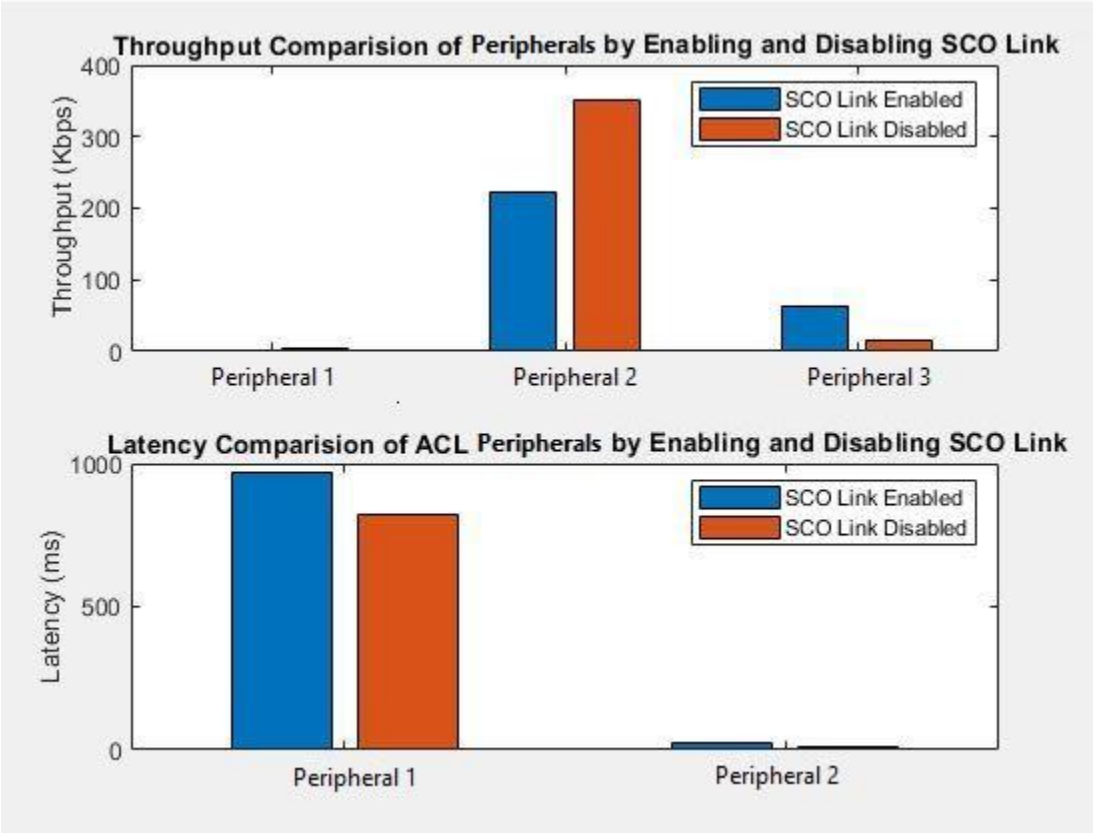
Because the presence of an SCO Peripheral in the piconet reserves the slots, the bandwidth for ACL Peripherals decreases. Consequently, the presence of an SCO Peripheral results in the suspension of most of the ongoing ACL transmissions. In the default configuration, as the simulation uses the priority scheduler, wireless speaker (Peripheral 2) is prioritized over laptop (Peripheral 1) and smartphone (Peripheral 2). In this case, audio streaming has a better throughput and lower latency than the file transfer.

You can further evaluate and compare the performance of Peripherals:

- In the presence of an SCO link
- By varying the scheduling algorithm

Compare Performance of Peripherals in Presence of SCO Link

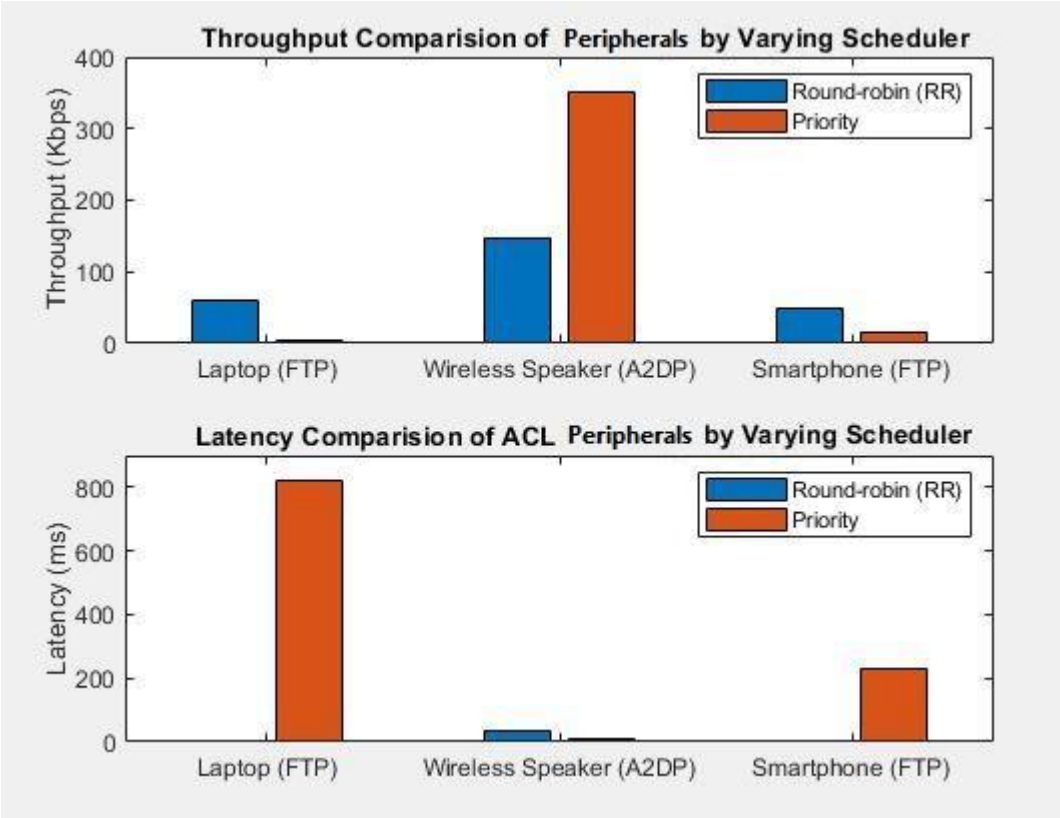
Simulate the scenario for 3 seconds with a QoS-based scheduler and with WLAN signal interference enabled. Compare the throughput and latency by enabling and then disabling the SCO link at Peripheral 3.



The presence of the SCO link degrades the throughput and latency performance of ACL Peripherals. And also, the wireless speaker (Peripheral 2) cannot achieve the required throughput if an SCO Peripheral in the piconet exists, even if you use a QoS-based priority scheduler. Because acknowledgement for SCO packets is not required, the latency is calculated only for the ACL Peripherals.

Compare Performance of Peripherals by varying Scheduling Algorithm

Simulate the scenario for 3 seconds with the RR and QoS-based priority scheduler. Enable WLAN signal interference for both scenarios throughout the simulation.



For the RR scheduler, the throughput and latency values are approximately same for the laptop, wireless speaker, and smartphone. The QoS-based priority scheduler achieves the desired throughput and low latency of the wireless speaker (Peripheral 2).

You can use parameters such as PER, BER, level of WLAN interference, and channel map to create your own custom scheduler and evaluate its performance using this simulation. For more details on experimenting with channel classification, see “Bluetooth Full Duplex Data and Voice Transmission in MATLAB” on page 1-24. Follow these steps to implement and integrate a custom Bluetooth scheduler.

1. Use the `helperBluetoothScheduler` helper object to create a new scheduler object.
2. Implement the algorithm by defining the `runScheduler` function. Modify the `helperBluetoothBaseband` helper object to update and retrieve the newly added parameters required by the algorithm.
3. Update the logic to attach the scheduler to the node in the `helperBluetoothCreatePiconet` helper function based on the algorithm.

Add WLAN Signal Using WLAN Toolbox™ Features

To add a WLAN signal using WLAN Toolbox™ features, set the value of `wlanInterference` to `Generated`. Use this code to add the generated WLAN signal as static signal interference to Bluetooth. Use this sample code snippet in WLAN signal generation using WLAN Toolbox™ features.

```
% % Create a WLAN waveform to interfere with Bluetooth waveforms,
% % which can be modified by using the features of the WLAN Toolbox.
% psduLength = 1000;
%
```

```
% % Create a configuration object for generating the WLAN waveform (802.11b)
% cfgNHT = wlanNonHTConfig('Modulation','DSSS', ...
%     'PSDULength', psduLength);
%
% % Create a random payload
% payload = randi([0 1], cfgNHT.PSDULength*8, 1);
%
% % Generate the WLAN waveform
% wlanWaveform = wlanWaveformGenerator(payload, cfgNHT);
```

You can add your custom signal generation code in the `helperBluetoothGenerateWLANWaveform` helper function. You can also write the respective signal WLAN spectrum masks and register to the `WLANSpectrum` property of the `helperBluetoothChannel` helper object as a function pointer.

This example simulates a home environment use-case scenario and demonstrates how to schedule Bluetooth FTP and A2DP application traffic on ACL link. The example uses the RR and QoS-based priority scheduling algorithms to schedule the application traffic. The results show that the QoS-based priority scheduling algorithm prioritizes A2DP traffic and gives a better throughput and latency performance of the wireless speaker (Peripheral 2). The results from further explorations indicate that the performance of ACL Peripherals decreases in the presence of SCO Peripherals.

Appendix

The example uses these helpers:

- helperBluetoothNode: Configure and simulate Bluetooth node
- helperBluetoothBaseband: Configure and simulate Bluetooth baseband layer
- helperBluetoothLogicalTransports: Configure logical transports between Bluetooth nodes
- helperBluetoothSlotTimer: Manage Bluetooth clock and the timing of slots
- helperBluetoothPHY: Configure and simulate Bluetooth PHY layer
- helperBluetoothPacketDuration: Duration of a Bluetooth packet
- helperBluetoothPracticalReceiver: Receive Bluetooth BR/EDR waveform
- helperBluetoothApplication: Configure and simulate Bluetooth application layer
- helperBluetoothBaseNode: Base class for Bluetooth node
- helperBluetoothChannel: Configure and simulate Bluetooth wireless channel
- helperBluetoothChannelClassification: Classify Bluetooth BR/EDR channels
- helperBluetoothLMPPDUConfig: Bluetooth link manager protocol (LMP) configuration object
- helperBluetoothLMPPDU: Generate Bluetooth LMP PDU
- helperBluetoothLMPPDUDecode: Decode Bluetooth LMP PDU
- helperBluetoothNetworkTraffic: Bluetooth application traffic pattern
- helperBluetoothPHYBase: Base class for Bluetooth PHY
- helperBluetoothQueue: Create a queue to buffer Bluetooth packets
- helperBluetoothScheduler: Base class for Bluetooth scheduler
- helperBluetoothGenerateWLANWaveform: Generate WLAN waveform to be added as interference to Bluetooth waveforms
- helperBluetoothCreatePiconet: Create Bluetooth piconet using the Bluetooth nodes
- helperBluetoothSchedulingStatistics: Return statistics of each Bluetooth node in the Bluetooth piconet

- helperBluetoothDistributePackets: Distribute Tx packets in the piconet
- helperBluetoothL2CAP: Create and process Bluetooth L2CAP channels
- helperBluetoothL2CAPFrame: Generate Bluetooth L2CAP frame
- helperBluetoothL2CAPFrameDecode: Decode Bluetooth L2CAP frame
- helperBluetoothRRScheduler: Create object for Bluetooth Round-robin scheduler
- helperBluetoothPriorityScheduler: Create object for Bluetooth priority scheduler

Selected Bibliography

1. Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 29, 2021. https://www.bluetooth.com/.
2. Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com.
3. Bluetooth Special Interest Group (SIG). "Traditional Profile Specifications." . https://www.bluetooth.com.

See Also

More About

- “Create, Configure, and Simulate Bluetooth LE Network” on page 9-38
- “Create Bluetooth Piconet by Enabling ACL Traffic, SCO Traffic, and AFH” on page 9-23
- “Packet Distribution in Bluetooth Piconet” on page 9-28

Estimate Packet Delivery Ratio of LE Broadcast Audio in Residential Scenario

This example shows you how to estimate the packet delivery ratio (PDR) of Bluetooth low energy (LE) audio isochronous broadcast streams in a residential scenario by using the Bluetooth® Toolbox.

Using this example, you can:

- Create and configure a residential scenario to simulate Bluetooth LE audio broadcast network.
- Configure the broadcast isochronous group (BIG) parameters at the broadcasters and receivers.
- Add a custom path loss model to the wireless channel.
- Add WLAN interference and explore the performance of the LE audio broadcast network with and without WLAN interference.
- Visualize PDR at different receiver locations in the residential scenario through a heatmap.

Isochronous Broadcasting in LE Audio

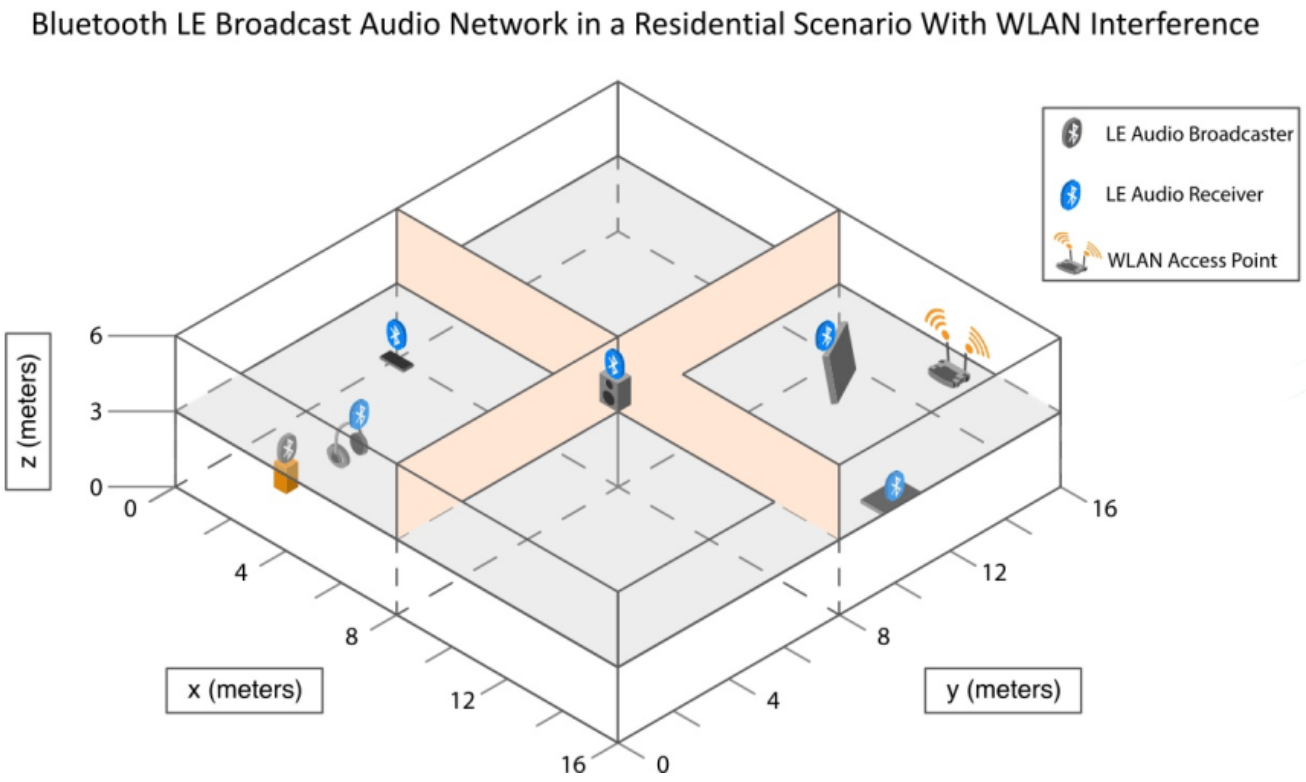
The Bluetooth Core Specification 5.2 [2 on page 6-0] defined by the Special Interest Group (SIG) enhances LE audio technology by adding the functionality to broadcast one or more audio streams to an unlimited number of audio receiver nodes in LE audio. To implement this functionality, the Bluetooth Core Specification 5.2 [2 on page 6-0] defines a new state, isochronous broadcasting, to the link layer (LL) state machine. In the isochronous broadcasting state, the LL transmits the isochronous data packets on an isochronous physical channel. If a Bluetooth LE node is in the isochronous broadcasting state then it is called as an isochronous broadcaster.

To realize connectionless broadcast isochronous communication, the Bluetooth Core Specification 5.2 [2 on page 6-0] defines broadcast isochronous streams (BIS) and broadcast isochronous group (BIG) events. A BIS is a logical transport that enables a Bluetooth LE node to transfer isochronous data (framed or unframed). A BIG contains one or more BISes that have the same isochronous interval.

For more information about BIS and BIG events, see “Bluetooth LE Audio” on page 8-41.

Residential Scenario Description

This example demonstrates a system-level simulation to evaluate the performance of a Bluetooth LE broadcast audio network in a residential scenario. The residential scenario consists of a building with two floors. This figure shows the residential scenario simulated in this example.



These are the characteristics of the residential scenario:

- Each floor consists of four rooms, each having dimensions 8 m x 8 m x 3 m.
- The building has one broadcaster, five receivers, and one WLAN interferer, placed in random x- and y- locations at a height of 1.5 meters from the floor.
- Each receiver is synchronized to the broadcaster.

Configure Simulation Parameters

Set the seed for the random number generator to 1 to ensure repeatability of results. The seed value controls the pattern of random number generation. For high fidelity simulation results, change the seed value and average the results over multiple simulations.

```
rng(1,"twister");
```

Residential Scenario

The `scenario` structure defines the size and layout of the residential building by using these parameters.

- `BuildingLayout` - Specifies the number of rooms along the length, breadth, and height of the building.
- `RoomSize` - Specifies the size of each room in meters.

```
scenario = struct;
scenario.BuildingLayout = [2 2 2];
scenario.RoomSize = [8 8 3];        % In meters
```

Initialize Broadcaster Nodes

Set the position of the broadcaster in the network.

```
broadcasterPosition = [2 2 1.5];   % x-, y-, and z- coordinates in meters
```

Create a Bluetooth LE node with the role set to `isochronous-broadcaster`. Specify the transmitter properties.

```
broadcasterNode = bluetoothLENode("isochronous-broadcaster");
broadcasterNode.Name = "Broadcaster";
broadcasterNode.Position = broadcasterPosition(1,:);
broadcasterNode.TransmitterPower = 10                          % In dBm
```

```
broadcasterNode =
  bluetoothLENode with properties:

        TransmitterPower: 10
         TransmitterGain: 0
           ReceiverRange: 100
            ReceiverGain: 0
     ReceiverSensitivity: -100
             NoiseFigure: 0
    InterferenceFidelity: 0
                    Name: 'Broadcaster'
                Position: [2 2 1.5000]

   Read-only properties:
                    Role: 'isochronous-broadcaster'
               BIGConfig: [1x1 bluetoothLEBIGConfig]
          TransmitBuffer: [1x1 struct]
                      ID: 22
```

Initialize Receiver Nodes

Specify the number of receivers and their respective positions in the network.

```
numReceivers = 5;
receiverPositions = [9 7 4.5; ...
    4 4 4.5; ...
    12 12 4.5; ...
    3 3 1.5; ...
    14 12 1.5];                    % x-, y-, and z- coordinates in meters
```

Create Bluetooth LE nodes with the role set as `synchronized-receiver`.

```
receiverNodes = cell(1,numReceivers);
for rxIdx = 1:numReceivers
    rxNode = bluetoothLENode("synchronized-receiver");
    rxNode.Name = ['Receiver ' num2str(rxIdx)];
    rxNode.Position = receiverPositions(rxIdx,:);
    receiverNodes{rxIdx} = rxNode;
end
```

Configure BIG parameters

Create a Bluetooth LE BIG configuration object with default BIG parameters.

```
cfgBIG = bluetoothLEBIGConfig

cfgBIG =
  bluetoothLEBIGConfig with properties:

        SeedAccessAddress: '78E52493'
                  PHYMode: 'LE1M'
                   NumBIS: 1
              ISOInterval: 0.0050
               BISSpacing: 0.0022
              SubInterval: 0.0022
                   MaxPDU: 251
              BurstNumber: 1
    PretransmissionOffset: 0
          RepetitionCount: 1
             NumSubevents: 1
           BISArrangement: 'sequential'
                BIGOffset: 0
        ReceiveBISNumbers: 1
             UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... ]
            InstantOffset: 6
    BaseCRCInitialization: '1234'
```

Assign the configuration to the broadcaster and receiver nodes.

```
for rxIdx = 1:numReceivers
    configureBIG(cfgBIG,broadcasterNode,receiverNodes{rxIdx});
end
```

Add Application Traffic to Broadcaster

Attach application traffic source to the broadcaster node. Configure the on-off application traffic pattern, by using the `networkTrafficOnOff` object, at the broadcaster node by specifying the application data rate, packet size, and on and off state duration.

```
trafficSource = networkTrafficOnOff(OnTime=inf,DataRate=500,PacketSize=cfgBIG.MaxPDU, ...
    GeneratePacket=true,ApplicationData=randi([0 255], 100, 1));
addTrafficSource(broadcasterNode, trafficSource);
```

WLAN Signal Interference

To add WLAN signal interference, enable the `wlanInterference` parameter.

`wlanInterference =` true ;

The interfering WLAN nodes add interference in the Bluetooth network by transmitting WLAN signals periodically based on the configuration. They do not model the physical layer (PHY) and MAC behavior of WLAN.

Set the properties of the interfering WLAN nodes. Specify the source of WLAN interference by using the `WaveformSource` parameter of the `helperInterferingWLANNode` helper function. Specify the source of the WLAN interference by using one of these options.

- `'Generated'` - To add a WLAN Toolbox™ signal to interfere the communication between Bluetooth nodes, select this option. For more information about how to add WLAN signal

interference generated using the WLAN Toolbox features, see Add WLAN Signal Using WLAN Toolbox Features on page 6-0 .

- `'BasebandFile'` - To add a WLAN signal from a baseband file (`.bb`) to interfere the communication between Bluetooth nodes, select this option. You can specify the file name using the `BasebandFile` parameter. If you do not specify the `.bb` file, the example uses the default `.bb` file, `'WLANHESUBandwidth20.bb'`, to add the WLAN signal.

```
if wlanInterference
    wlanNode = helperInterferingWLANNode;
    wlanNode.Name = "WLAN node";
    wlanNode.Position = [14 14 6];          % x-, y- and z- coordinates in meters
    wlanNode.TransmitterPower = 5;          % In dBm
end
```

Create and Simulate Broadcast Isochronous Network

Create a broadcast isochronous network consisting of the LE audio broadcast nodes and WLAN interfering nodes (if present).

```
nodes = [{broadcasterNode} receiverNodes];
if wlanInterference
    nodes = [nodes {wlanNode}];
end
```

Initialize the broadcast isochronous network.

```
networkSimulator = helperWirelessNetwork(nodes);
```

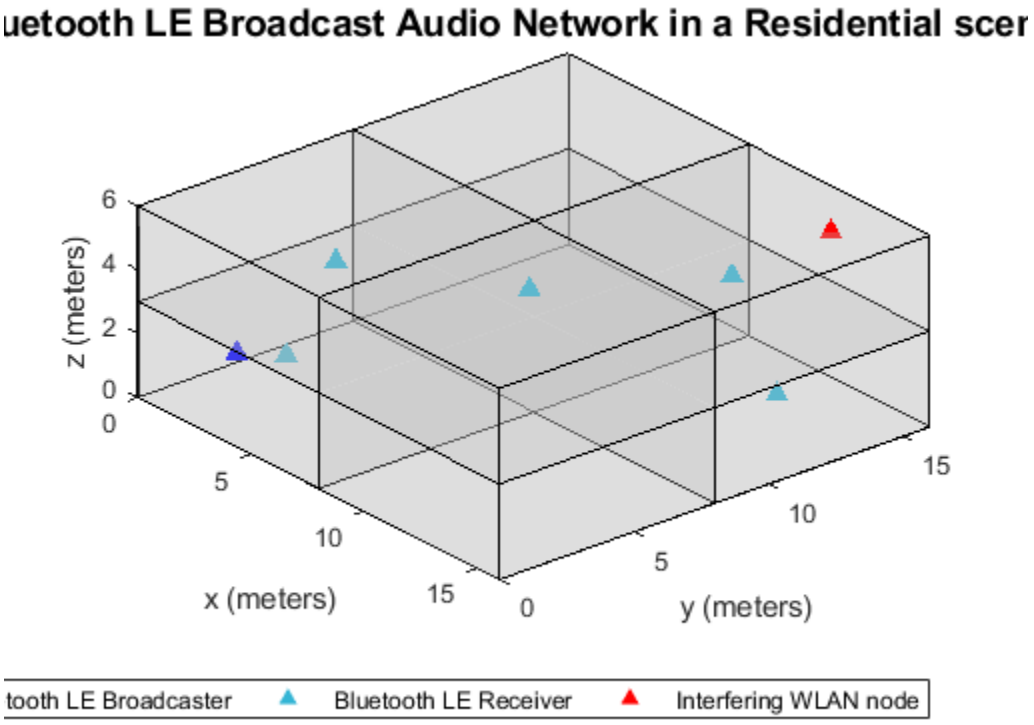
Visualize the Network

Create the building triangulation from the scenario parameters.

```
buildingTriangulation = hTGaxResidentialTriangulation(scenario);
```

Visualize the residential scenario in 3-D.

```
helperVisualizeResidentialScenario(buildingTriangulation,{broadcasterNode;receiverNodes;wlanNode
    ["Bluetooth LE Broadcaster","Bluetooth LE Receiver","Interfering WLAN node"],...
    "Bluetooth LE Broadcast Audio Network in a Residential scenario");
```



Wireless Channel

Set the `customPathlossModel` to true to add your own custom path loss model. Specify the function handle of the custom model by using the `pathloss` parameter. If you set the `customPathlossModel` to false, the example uses a free-space path loss model.

`customPathlossModel =` true ;

This example uses the TGax residential propagation model [4 on page 6-0] to determine pathloss between the nodes. Path loss depends on the number of walls, number of floors, and the distance between nodes. Create a path loss model by using the hTGaxResidentialPathLoss helper function. Obtain the path loss between each pair of nodes in the network by using a function handle. Add a residential path loss model to the simulator.

```
if customPathlossModel
    propModel = hTGaxResidentialPathLoss(Triangulation=buildingTriangulation,...
        FacesPerWall=1);                                          % Propagation model
    pathloss = helperGeneratePathLossTable(propModel,nodes);      % Pathloss function
    addChannelModel(networkSimulator,pathloss);
end
```

Run the Simulation

Set the simulation time in seconds.

```
simulationTime = 1;
```

Run all the nodes in the network for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

Simulation Results

At each receiver node, the simulation measures these metrics:

- PDR
- Average LL packet latency
- LL throughput
- Packet statistics at PHY and LL
- Time spent in listen state and sleep state

View the table of statistics by using the helperBLEBIGStatistics helper function.

```
receiverStatistics = helperBLEBIGStatistics(nodes)
```

```
receiverStatistics=5×17 table
                     Position         PDR     LL Average Packet Latency (seconds)    Successful
                  ________________    _____    ___________________________________    __________

    Receiver 1     9      7    4.5     0.74                0.002089                             1
    Receiver 2     4      4    4.5    0.965                0.002089                             1
    Receiver 3    12     12    4.5     0.67                0.002089                             1
    Receiver 4     3      3    1.5     0.83                0.002089                             1
    Receiver 5    14     12    1.5    0.965                0.002089                             1
```

Further Exploration

Add WLAN Signal Using WLAN Toolbox Features

To add a WLAN signal using WLAN Toolbox features, follow these steps:

Set the value of `WaveformSource` parameter of the `helperInterferingWLANNode` to '`Generated`'.

```
% wlanNode = helperInterferingWLANNode;
% wlanNode.WaveformSource = 'Generated';
```

Create a WLAN packet format configuration object and assign it to the node.

```
% cfgHT = wlanHTConfig("ChannelBandwidth","CBW40");
% wlanNode.FormatConfig = cfgHT;
```

Set the bandwidth of the signal.

```
% wlanNode.Bandwidth = 40e6;
```

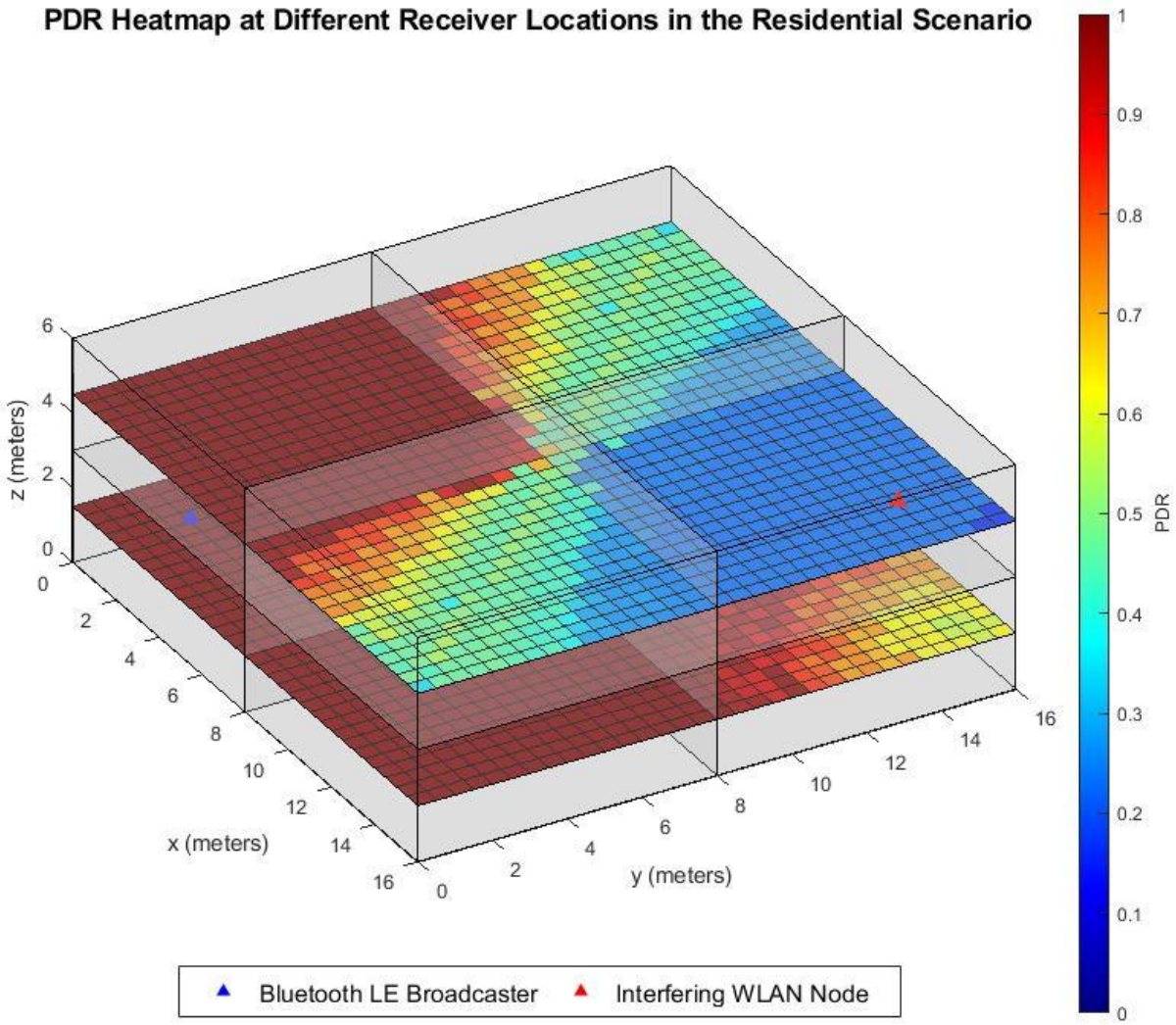
Heatmap of PDR

To observe the variation in PDR when a receiver moves within the building, you can run the simulation for different positions of a receiver with a fixed broadcaster. Store the receiver positions in a matrix, `receiverPositions`, and the corresponding PDR values in a column vector, `pdr`. Observe the variation of PDR with respect to the distance from the broadcaster by using this code snippet.

```
% helperVisualizeResidentialScenario(buildingTriangulation,{broadcasterNode;wlanNode},...
%     ["Bluetooth LE Broadcaster","Interfering WLAN node"],...
```

```
%     "PDR Heatmap at Different Receiver Locations in the Residential scenario");
% pdr = receiverStatistics.PDR;
% plotHeatmap(scenario,receiverPositions,pdr);
%
% % Function to plot the heatmap of PDR values
% function plotHeatmap(scenario, receiverPositions, pdr)
%     numFloors = scenario.BuildingLayout(3);
%     floorDim = [scenario.RoomSize(1:2).*scenario.BuildingLayout(1:2) scenario.RoomSize(3)];
%     floorHeights = scenario.RoomSize(3):scenario.RoomSize(3):scenario.RoomSize(3)*numFloors;
%
%     % Create x-. and y-axis inputs for surf function
%     [X, Y] = meshgrid(0:1:max(floorDim(1), floorDim(2)));
%
%     for f = 1:numFloors
%         pdrValues = zeros(size(X));
%         floorIdx = receiverPositions(:,3)>(floorHeights(f)-floorDim(3)) & receiverPositions(:,3
%         receiversInFloor = receiverPositions(floorIdx, :);
%         receiversPDR = pdr(floorIdx);
%         for idx = 1:numel(receiversInFloor)/3
%             index = (X == round(receiversInFloor(idx,1)) & Y == round(receiversInFloor(idx,2)))
%             pdrValues(index) = receiversPDR(idx);
%         end
%         Z = (floorHeights(f)-floorDim(3)/2)*ones(size(X));
%         surf(X,Y,Z,pdrValues);
%     end
%     c = colorbar;
%     c.Label.String = 'PDR';
%     colormap('jet');
%     caxis([0 1]);
% end
```

This heatmap plots the set of PDRs captured at different positions of a receiver.



You can analyze the improvement in PDR in the presence of interference by changing the configuration parameters. To observe an increase in PDR, you can:

- Increase the transmission power of the broadcaster node by using the `TransmitterPower` property of the `bluetoothLENode` object.
- Increase the number of retransmissions by using the `RepetitionCount` property of the `bluetoothLEBIGConfig` object.
- Create time diversity in the copies of a packet by setting a nonzero pre-transmission offset by using the `PretransmissionOffset` property of the `bluetoothLEBIGConfig` object.
- Avoid hopping to bad channels by setting the good channels using the `updateChannelList` object function of the broadcaster node.

Appendix

The example uses these helpers:

- helperWirelessNetwork - Simulate a wireless network
- helperInterferingWLANNode - Configure and simulate an interfering WLAN node
- helperBLEBIGStatistics - Return statistics of each receiver in the scenario
- helperVisualizeResidentialScenario - Visualize the residential scenario and network in 3D
- hTGaxResidentialTriangulation - Create the residential scenario geometry
- hTGaxResidentialPathLoss - Configure and create a residential pathloss model
- hTGaxIndoorLinkInfo - Return the number of floors, the number of walls, and the distance between points for a link
- helperGeneratePathLossTable - Return the path loss factor in the residential scenario for a node pair

Selected Bibliography

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 10, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "*Bluetooth Core Specification*." Version 5.2. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "*Bluetooth Core Specification*." Version 5.3. https://www.bluetooth.com/.

[4] "*TGax Channel Models*.", doc. IEEE 802.11-14/0882r4.

See Also

Functions

`addTrafficSource` | `configureBIG` | `statistics`

Objects

`bluetoothLENode` | `bluetoothLEBIGConfig`

More About

- “Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network” on page 9-41
- “Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario” on page 9-53
- “Bluetooth LE Audio” on page 8-41
- “Bluetooth LE Node Statistics” on page 8-78

7

Code Generation and Deployment

What is C Code Generation from MATLAB?

You can use Bluetooth Toolbox together with MATLAB® Coder™ to:

- Create a MEX file to speed up your MATLAB application.
- Generate ANSI®/ISO® compliant C/C++ source code that implements your MATLAB functions and models.
- Generate a standalone executable that runs independently of MATLAB on your computer or another platform.

In general, the code you generate using the toolbox is portable ANSI C code. In order to use code generation, you need a MATLAB Coder license. For more information, see “Get Started with MATLAB Coder” (MATLAB Coder).

Using MATLAB Coder

Creating a MATLAB Coder MEX file can substantially accelerate your MATLAB code. It is also a convenient first step in a workflow that ultimately leads to completely standalone code. When you create a MEX file, it runs in the MATLAB environment. Its inputs and outputs are available for inspection just like any other MATLAB variable. You can then use MATLAB tools for visualization, verification, and analysis.

The simplest way to generate MEX files from your MATLAB code is by using the `codegen` function at the command line. For example, if you have an existing function, `myfunction.m`, you can type the commands at the command line to compile and run the MEX function. `codegen` adds a platform-specific extension to this name. In this case, the `"mex"` suffix is added.

```
codegen myfunction.m
myfunction_mex;
```

Within your code, you can run specific commands either as generated C code or by using the MATLAB engine. In cases where an isolated command does not yet have code generation support, you can use the `coder.extrinsic` command to embed the command in your code. This means that the generated code reenters the MATLAB environment when it needs to run that particular command. This is also useful if you want to embed commands that cannot generate code (such as plotting functions).

To generate standalone executables that run independently of the MATLAB environment, create a MATLAB Coder project inside the MATLAB Coder Integrated Development Environment (IDE). Alternatively, you can call the `codegen` command in the command line environment with appropriate configuration parameters. A standalone executable requires you to write your own `main.c` or `main.cpp` function. See “Generating Standalone C/C++ Executables from MATLAB Code” (MATLAB Coder) for more information.

C/C++ Compiler Setup

Before using `codegen` to compile your code, you must set up your C/C++ compiler. For 32-bit Windows platforms, MathWorks® supplies a default compiler with MATLAB. If your installation does not include a default compiler, you can supply your own compiler. For the current list of supported compilers, see Supported and Compatible Compilers on the MathWorks website. Install a compiler that is suitable for your platform, then read “Setting Up the C or C++ Compiler” (MATLAB Coder).

After installation, at the MATLAB command prompt, run `mex -setup`. You can then use the `codegen` function to compile your code.

Functions and System Objects That Support Code Generation

See Also

Functions
`codegen` | `mex`

More About

- “Code Generation Workflow” (MATLAB Coder)
- Generate C Code from MATLAB Code Video

Bluetooth Topic Pages

Bluetooth Technology Overview

Bluetooth [1] wireless technology is the air interface intended to replace the cables connecting portable and fixed electronic equipment. Bluetooth device manufacturers have the flexibility to include optional core specification features to optimize and differentiate product offers.

Bluetooth is equated with the implementation specified by the Bluetooth Core Specification [3] group of standards maintained by the Bluetooth Special Interest Group (SIG) industry consortium. The Bluetooth Toolbox functionalities enables you to model Bluetooth low energy (LE) and Bluetooth basic rate/enhanced data rate (BR/EDR) communications system links, as specified in the Core System Package [Low Energy Controller volume], Specification Volume 6. It also enables you to explore variations on implementations for future evolution of the standard. Bluetooth BR/EDR and Bluetooth LE devices operate in the same unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) frequency band as Wi-Fi®.

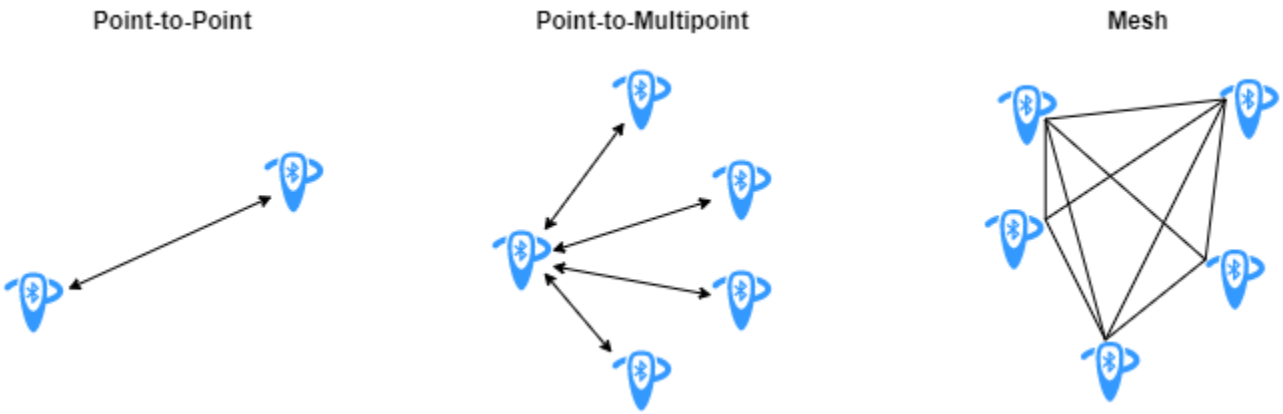
In Bluetooth BR/EDR, the radio hops in a pseudo-random way on 79 designated Bluetooth channels. Each Bluetooth BR/EDR channel has a bandwidth of 1 MHz. Each frequency is located at (2402 + k) MHz, where k = 0,1,...78.

In Bluetooth LE, the operating radio frequency is in the range 2.4000 GHz to 2.4835 GHz, inclusive. The channel bandwidth is 2 MHz and the operating band is divided into 40 channels, k = 0, ..., 39. The center frequency of the k^{th} channel is located at 2402 + k × 2 MHz.

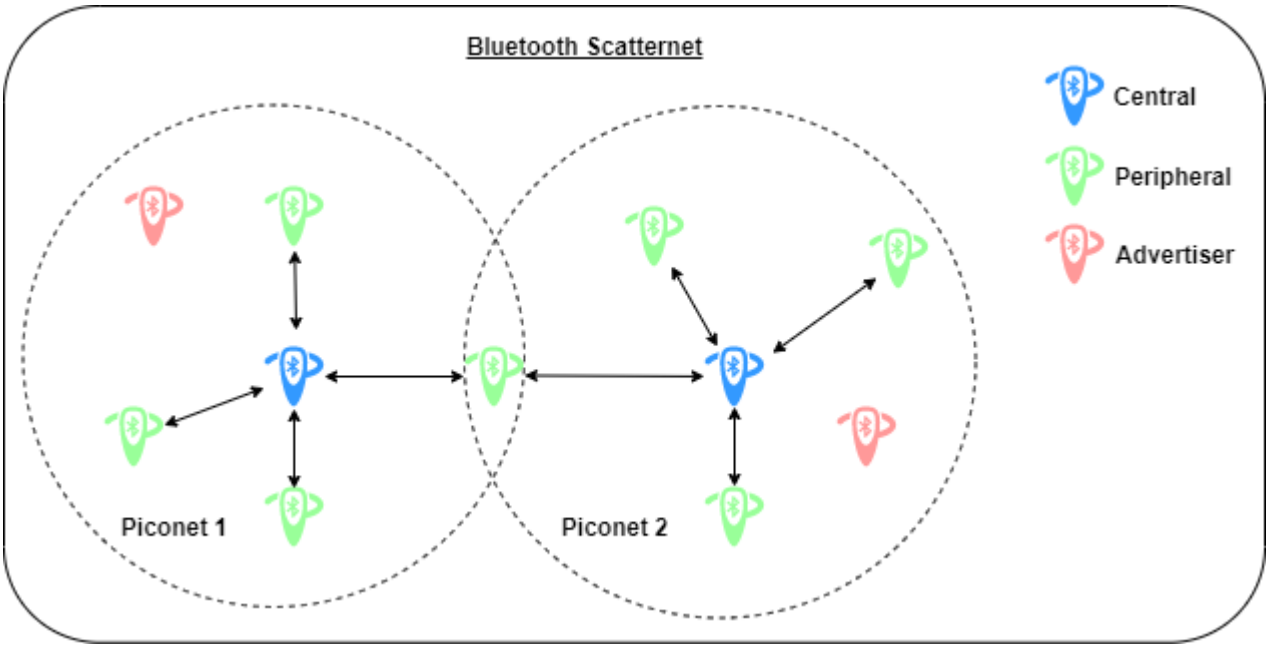
For more information about the specifications of Bluetooth BR/EDR and LE, see “Comparison of Bluetooth BR/EDR and Bluetooth LE Specifications” on page 8-6.

Bluetooth Connection Topologies

Most Bluetooth LE devices communicate with each other using a simple point-to-point (one-to-one communication) or point-to-multipoint (one-to-many communication) topology as shown in this figure.



Devices using one-to-one communication operate in a Bluetooth *piconet*. As shown in this figure, each piconet consists of a device in the role of *Central*, with other devices in the *Peripheral* or *Advertiser* roles. Before joining the piconet, each *Peripheral* node is in an advertiser role. Multiple piconets connect to each other, forming a Bluetooth *scatternet*.



For example, a smartphone with an established one-to-one connection to a heart rate monitor over which it can transfer data is an example of point-to-point connection.

On the contrary, the Bluetooth mesh enables you to set up many-to-many communication links between Bluetooth devices. In a Bluetooth mesh, devices can relay data to remote devices that are not in the direct communication range of the source device. This enables a Bluetooth mesh network to extend its radio range and encompass a large geographical area containing a large number of devices. Another advantage of the Bluetooth mesh over point-to-point and point-to-multipoint topologies is the capability of self healing. The self-healing capability of the Bluetooth mesh implies that the network does not have any single point of failure. If a Bluetooth device disconnects from the mesh network, other devices can still send and receive messages from each other, which keeps the network functioning. For more information about Bluetooth mesh networking, see “Bluetooth Mesh Networking” on page 8-64.

Solution Areas

This table summarizes prominent solution areas of Bluetooth BR/EDR and Bluetooth LE.

Application	Bluetooth BR/EDR	Bluetooth LE
Audio streaming applications such as: • Bluetooth headphones or earbuds • Bluetooth speakers • Bluetooth watches	Supported	Supported

Application	Bluetooth BR/EDR	Bluetooth LE
Location and direction finding applications such as: • Asset tracking • Indoor navigation services • Beacon-based services	Not supported	Supported
Data transmission applications such as: • Medical and health equipments • Sports and fitness equipments • Peripherals and accessories	Not supported	Supported
Device network applications such as: • Monitoring systems and services • Automation systems • Control systems	Not supported	Supported

New Use Cases and Enhancements

These are some of the prominent new Bluetooth use cases and capabilities introduced by the SIG.

- COVID-19 pandemic response solutions — To tackle the challenges of COVID-19 pandemic, many private and Government institutions have turned towards the Bluetooth technology for innovative solutions that can realize and accelerate reopening efforts across the world, and enable safer and faster treatment of patients during the COVID-19 pandemic and future disease outbreaks. To achieve these solutions, these three use cases leverage Bluetooth technology.
  - Exposure notification systems (ENS): Public ENS use the Bluetooth technology present in the smartphones to apprise people when they have been in close proximity with a person who was later diagnosed with COVID-19.
  - Safe return solutions: Public venues such as stadiums, offices, and universities etc. are looking to Bluetooth technology to provide safe return solutions to help them take necessary steps to reopen or continue to operate safely during the pandemic times. Some of the prominent solutions in these use case include occupancy management, exposure management, hygiene management, and touchless access and control.
  - Safe treatment solutions: Medical institutions are leveraging Bluetooth technology to improve the quality and efficiency of diagnosis and treatment. Some of these solutions include safe facility management, safe patient diagnosis and monitoring, remote patient care and monitoring.
- Networked lighting control — Networked lighting control systems feature an intelligent network of individually addressable and sensor-rich luminaries and control elements that enable each device of the system to send and receive data. Bluetooth networked lighting control systems are

deployed in offices, retail, healthcare, factories, and other commercial places to provide a combination of significant energy savings, improved occupant well-being and productivity, and efficient building operations and predictive maintenance. The key advantages of shifting from wired to wireless solutions for networked lighting control are reduced operation and maintenance cost, greater design and configuration flexibility, and future extensibility.

- Bluetooth LE audio — The Bluetooth Core Specification 5.2 [2] introduced the next generation of Bluetooth audio called the LE audio. LE audio operates on the Bluetooth LE standard. LE audio is the next generation of Bluetooth audio, which supports development of the same audio products and use cases as the classic audio. It also enables creation of new products and use cases and presents additional features and capabilities to help improve the performance of classic audio products. Some of the key features and use cases of LE audio include enabling audio sharing, providing multistream audio, and supporting hearing aids. For more information about LE audio, see “Bluetooth LE Audio” on page 8-41.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 14, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.2. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

More About

- “Comparison of Bluetooth BR/EDR and Bluetooth LE Specifications” on page 8-6
- “Bluetooth Protocol Stack” on page 8-9

Comparison of Bluetooth BR/EDR and Bluetooth LE Specifications

Bluetooth technology [1], operating on the 2.4 GHz unlicensed industrial, scientific, and medical (ISM) frequency band, uses low-power radio frequency to enable short-range communication at a low cost. The two variants of the Bluetooth technology are –

- Bluetooth basic rate/enhanced data rate (BR/EDR) or classic Bluetooth
- Bluetooth low energy (LE) or Bluetooth Smart

The Bluetooth Core Specification [2], specified by the Special Interest Group (SIG) consortium, defines the technologies required to create interoperable Bluetooth BR/EDR and Bluetooth LE devices.

Bluetooth BR/EDR radio is primarily designed for low power, high data throughput operations. In Bluetooth BR/EDR, the radio hops in a pseudo-random way on 79 designated Bluetooth channels. Each Bluetooth BR/EDR channel has a bandwidth of 1 MHz. Each frequency is located at $(2402 + k)$ MHz, where $k = 0,1, \ldots, 78$.

In 2010, the SIG introduced Bluetooth LE with the Bluetooth 4.0 version. The Bluetooth LE radio is designed and optimized to support applications and use cases that have a relatively low duty cycle. For example, suppose a person wears a heart rate monitoring device for several hours. Because this device transmits only a few bytes of data every second, its radio is in the 'on' state for a very short period of time. In Bluetooth LE, the operating radio frequency is in the range from 2.4000 GHz to 2.4835 GHz. The channel bandwidth is 2 MHz, and the operating band is divided into 40 channels, ($k = 0, 1, \ldots, 39$). The center frequency of the kth channel is located at $(2402 + k \times 2)$ MHz.

This table summarizes and compares different features of Bluetooth BR/EDR and Bluetooth LE.

Feature	Bluetooth BR/EDR	Bluetooth LE
Frequency band	Operates on a 2.4 GHz Industrial, Scientific, and Medical (ISM) band, with the values in the range from 2.4000 GHz to 2.4835 GHz	Operates on 2.4 GHz ISM band, with the values in the range from 2.4000 GHz to 2.4835 GHz
Channels	79 channels	40 channels (37 data channels and 3 advertising channels)
Channel bandwidth	1 MHz	2 MHz
Spread spectrum technique	1600 hops/sec frequency-hopping spread spectrum (FHSS)	FHSS
Modulation scheme	• Gaussian frequency shift keying (GFSK) • π/4 differential quadrature phase shift keying (DQPSK) • 8 differential phase shift keying (DPSK)	GFSK

Feature	Bluetooth BR/EDR	Bluetooth LE
Power usage	1 W (reference value)	~0.01x W to 0.5x W of reference (depending on the use case scenario)
Maximum transmission power	• Class 1: 100 mW (20 dBm) • Class 2: 2.5 mW (4 dBm) • Class 3: 1 mW (0 dBm)	• Class 1: 100 mW (20 dBm) • Class 1.5: 10 mW (10 dBm) • Class 2: 2.5 mW (4 dBm) • Class 3: 1 mW (0 dBm)
Data rate	• BR PHY (GFSK): 1 Mb/s • EDR PHY (π/4 DQPSK): 2 Mb/s • EDR PHY (8 DPSK): 3 Mb/s	• LE Coded PHY (S = 8): 125 Kb/s • LE Coded PHY (S = 2): 500 Kb/s • LE 1M PHY: 1 Mb/s • LE 2M PHY: 2 Mb/s
Device discovery	Inquiry or paging	Advertising
Device address privacy	None	Private device addressing supported
Encryption algorithm	E0/SAFER+	AES-CCM
Audio capable	Yes	Yes (Bluetooth LE audio is introduced in Bluetooth Core Specification 5.2)
Network topology	Point-to-point (including piconet)	• Point-to-point (including piconet) • Broadcast • Mesh

This table summarizes prominent applications of Bluetooth BR/EDR and Bluetooth LE.

Application	Bluetooth BR/EDR	Bluetooth LE
Audio streaming applications such as: • Bluetooth headphones or earbuds • Bluetooth speakers • Bluetooth watches	Supported	Supported
Location and direction finding applications such as: • Asset tracking • Indoor navigation services • Beacon-based services	Not supported	Supported

Application	Bluetooth BR/EDR	Bluetooth LE
Data transmission applications such as: • Medical and health equipments • Sports and fitness equipments • Peripherals and accessories	Not supported	Supported
Device network applications such as: • Monitoring systems and services • Automation systems • Control systems	Not supported	Supported

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 14, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

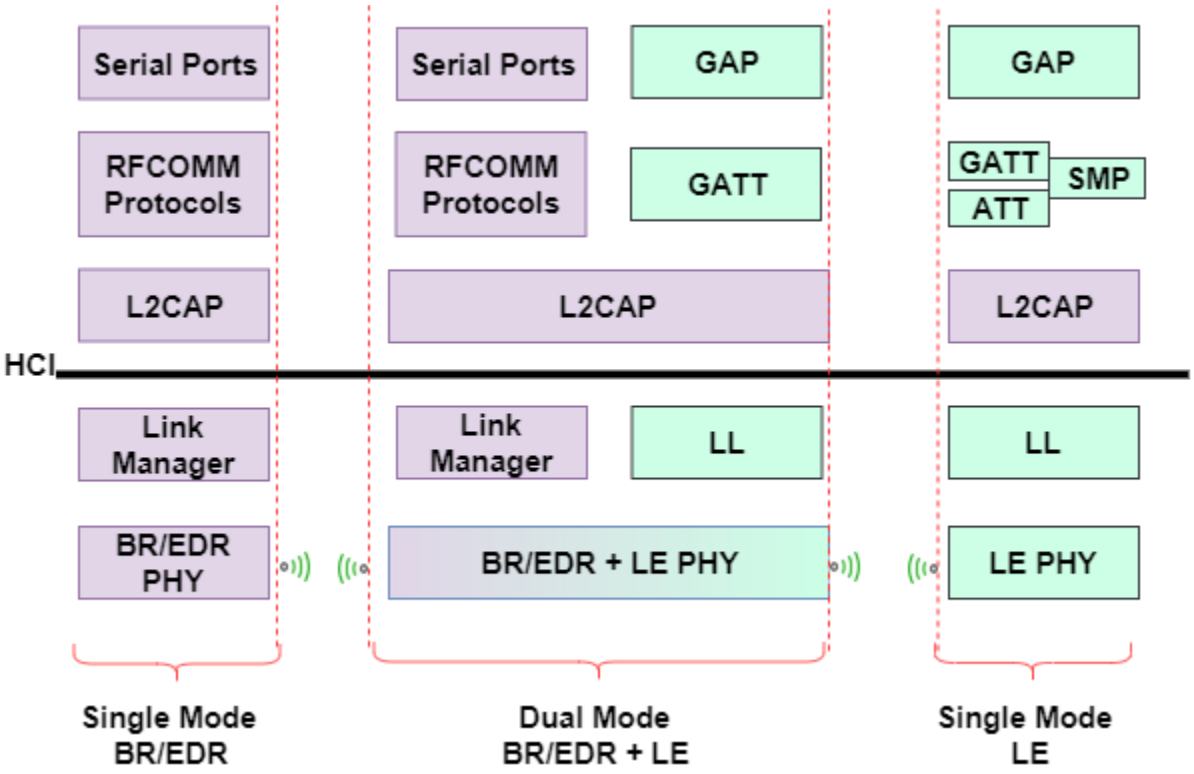
More About

• “Bluetooth Technology Overview” on page 8-2
• “Bluetooth Protocol Stack” on page 8-9
• “Bluetooth Packet Structure” on page 8-27

Bluetooth Protocol Stack

The Bluetooth Special Interest Group (SIG) [1] and [2] defines the protocol stack for Bluetooth low energy (LE) and Bluetooth basic rate/enhanced data rate (BR/EDR) technology. The fundamental objectives of these specifications is to develop interactive services and applications over interoperable radio components and data communication protocols.

This figure shows the architecture of the Bluetooth stack.



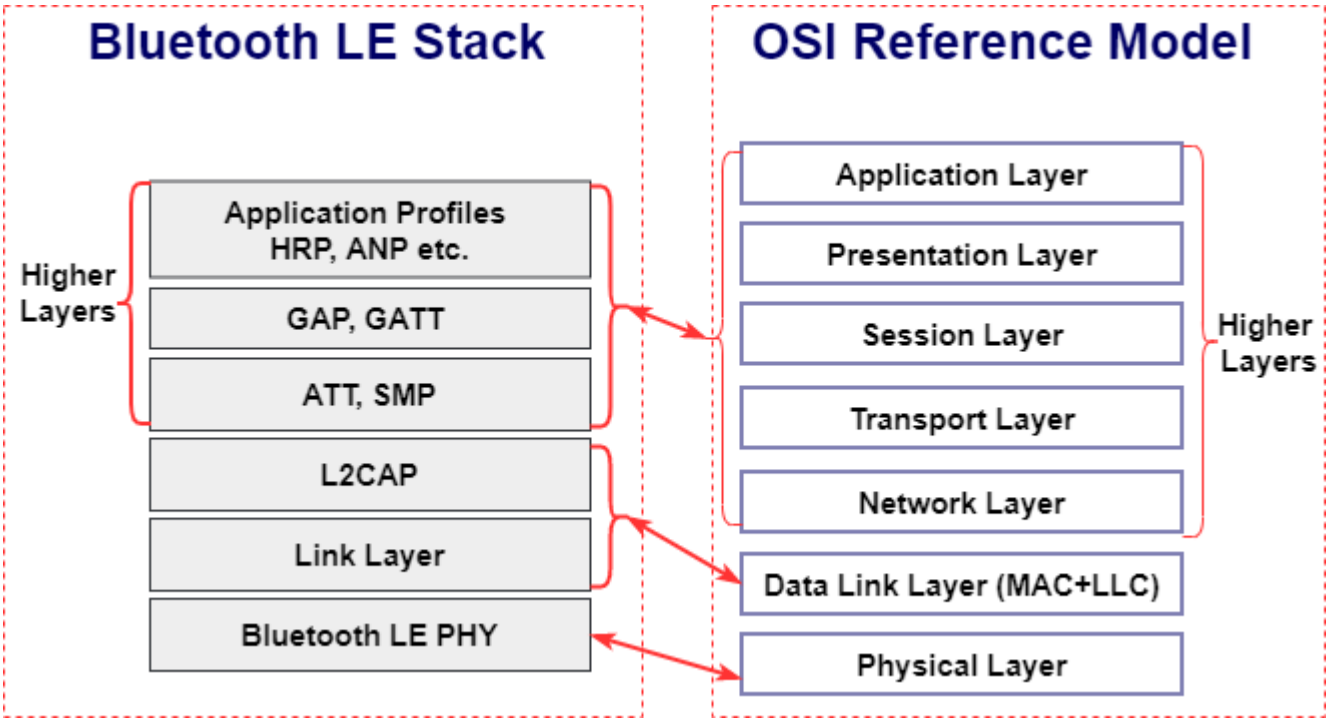
Bluetooth devices can be one of these two types:

- Single mode – Supports a BR/EDR or LE profile
- Dual mode – Supports BR/EDR and LE profiles

The subsequent sections provide details about the architecture of “Bluetooth LE Protocol Stack” on page 8-9 and “Bluetooth BR/EDR Protocol Stack” on page 8-14.

Bluetooth LE Protocol Stack

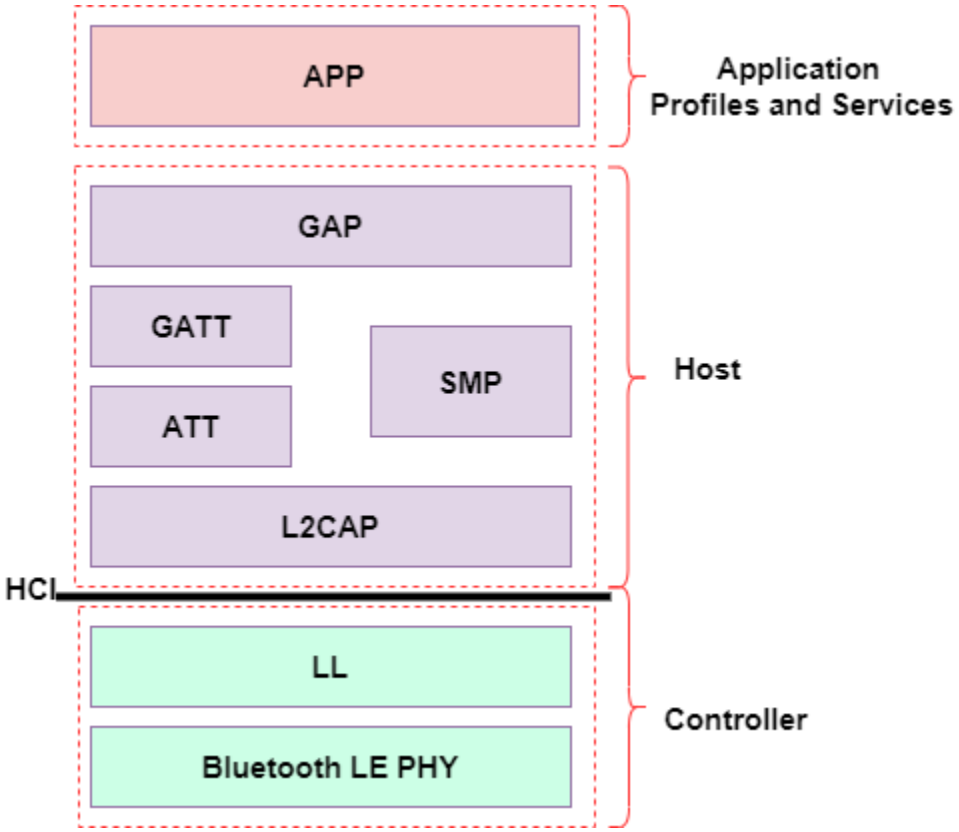
This figure compares the Bluetooth LE protocol stack to the Open System Interconnection (OSI) reference model.



In the preceding figure, the Bluetooth LE protocol stack is shown along with the OSI reference model.

- There is one-to-one mapping at the physical layer (PHY)
- The OSI data link layer (DLL) maps to the Bluetooth LE logical link control and adaptation protocol (L2CAP) and link layer (LL)
- In the Bluetooth LE stack, the higher layers provide application layer services, device roles and modes, connection management, and security protocol

The functionality of the Bluetooth LE protocol stack is divided between three main layers: the Controller, the Host, and Application Profiles and Services.



Controller

The controller layer includes the Bluetooth LE PHY, the LL, and the controller-side host controller interface (HCI).

Bluetooth LE PHY

The Bluetooth LE PHY air interface operates in the same unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) frequency band as Wi-Fi. The Bluetooth LE PHY air interface also includes these characteristics:

- Operating radio frequency (RF) is in the range 2.4000 GHz to 2.4835 GHz, inclusive.
- The channel bandwidth is 2 MHz. The operating band is divided into 40 channels, $k = 0, \ldots, 39$. The center frequency of the kth channel is $2402 + k \times 2$ MHz.
  - User data packets are transmitted using channels in the range [0, 36].
  - Advertising data packets are transmitted in channels 37, 38, and 39.
- Gaussian frequency shift-keying (GFSK) modulation scheme is implemented.
- The Bluetooth LE PHY uses frequency-hopping spread spectrum (FHSS) to reduce interference and to counter the impact of fading channels. The time between frequency hops can vary from 7.5 ms to 4 s and is set at the connection time for each Peripheral.
- Support for throughput at 1 Mbps is mandatory for specification version 4.x compliant devices. At a data rate of 1 Mbps, the transmission is uncoded.
- Optionally, devices compliant with the Bluetooth Core Specification version 5.1 support these additional data rates:

- Coded transmission at bit rates of 500 kbps or 125 kbps
- Uncoded transmission at a bit rate of 2 Mbps

LL

The LL performs tasks similar to the medium access control (MAC) layer of the OSI model. In Bluetooth, the LL interfaces directly with the Bluetooth LE PHY and manages the link state of the radio to define the role of a device as Central, Peripheral, Advertiser, or Scanner.

Controller-Side HCI

The HCI on the controller side handles the interface between the host and the controller. The HCI defines a set of commands and events for transmission and reception of packet data. When receiving packets from the controller, the HCI extracts raw data at the controller to send to the host.

Host

The host includes the host-side HCI, L2CAP, attribute protocol (ATT), generic attribute profile (GATT), security manager protocol (SMP), and generic access profile (GAP).

Host-Side HCI

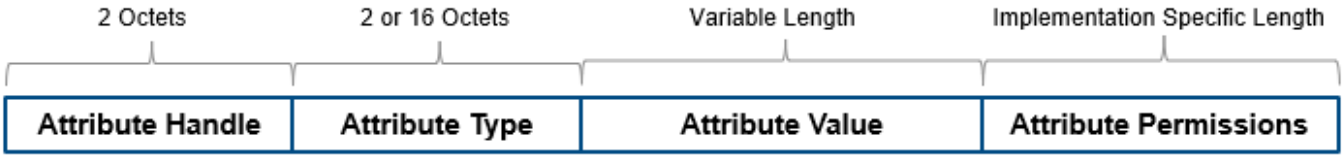
The HCI on the host side handles the interface between the host and the controller. The HCI defines a set of commands and events for transmission and reception of packet data. When transmitting data, the HCI translates raw data into packets to send them from the host to the controller.

L2CAP

The L2CAP encapsulates data from the Bluetooth LE higher layers into the standard Bluetooth LE packet format for transmission or extracts data from the standard Bluetooth LE LL packet on reception according to the link configuration specified at the ATT and SMP layers.

ATT

The ATT transfers attribute data between clients and servers in GATT-based profiles. The ATT defines the roles of the client-server architecture. The roles typically correspond to the Central and the Peripheral as defined in the link layer. In general, a device could be a client, a server, or both, irrespective of whether it is a Central or a Peripheral. The ATT also performs data organization into attributes as shown in this figure.



Device attributes are represented as:

- The attribute handle is a 16-bit identifier value assigned by the server to enable a client to reference those attributes.
- The attribute type is a universally unique identifier (UUID) defined by Bluetooth SIG. For example, UUID 0x2A37 represents a heart-rate measurement.
- The attribute value is a variable length field. The UUID associated with and the service class of the service record containing the attribute value, determine the length of the attribute value field.

- Attribute permissions are sets of permission values associated with each attribute. These permissions specify read and write privileges for an attribute, and the security level required for read and write permission.

GATT

The GATT provides a reference framework for all GATT-based profiles. The GATT encapsulates the ATT and is responsible for coordinating the exchange of profiles in a Bluetooth LE link. Profiles include information and data such as handle assignment, a UUID, and a set of permissions.

For devices that implement the GATT profile,

- The client is the device that initiates commands and requests toward the server. The client can receive responses, indications, and notifications.
- The server is the device that accepts incoming commands and requests from the client. The server sends responses, indications, and notifications to the client.

The GATT uses client-server architecture. The roles are not fixed and are determined when a device initiates a defined procedure. Roles are released when the procedure ends.

The terminology used in the GATT includes:

- Service — A collection of data and associated behaviors used to accomplish a particular function or feature
- Characteristic — A value used in a service along with appropriate permissions
- Characteristic descriptor — A description of the associated characteristic behavior
- GATT-Client — A GATT-Client initiates commands and requests towards the server and can receive responses, indications, and notifications sent by the server
- GATT-Server — A GATT-Server accepts incoming commands and requests from a client and sends responses, indications, and notifications to the client

SMP

The SMP applies security algorithms to encrypt and decrypt data packets. This layer defines the initiator and the responder, corresponding to the Central and the Peripheral, once the connection is established.

GAP

The GAP specifies roles, modes, and procedures of a device. It also manages the connection establishment and security. The GAP interfaces directly with the Application Profiles and Services (App) layer.

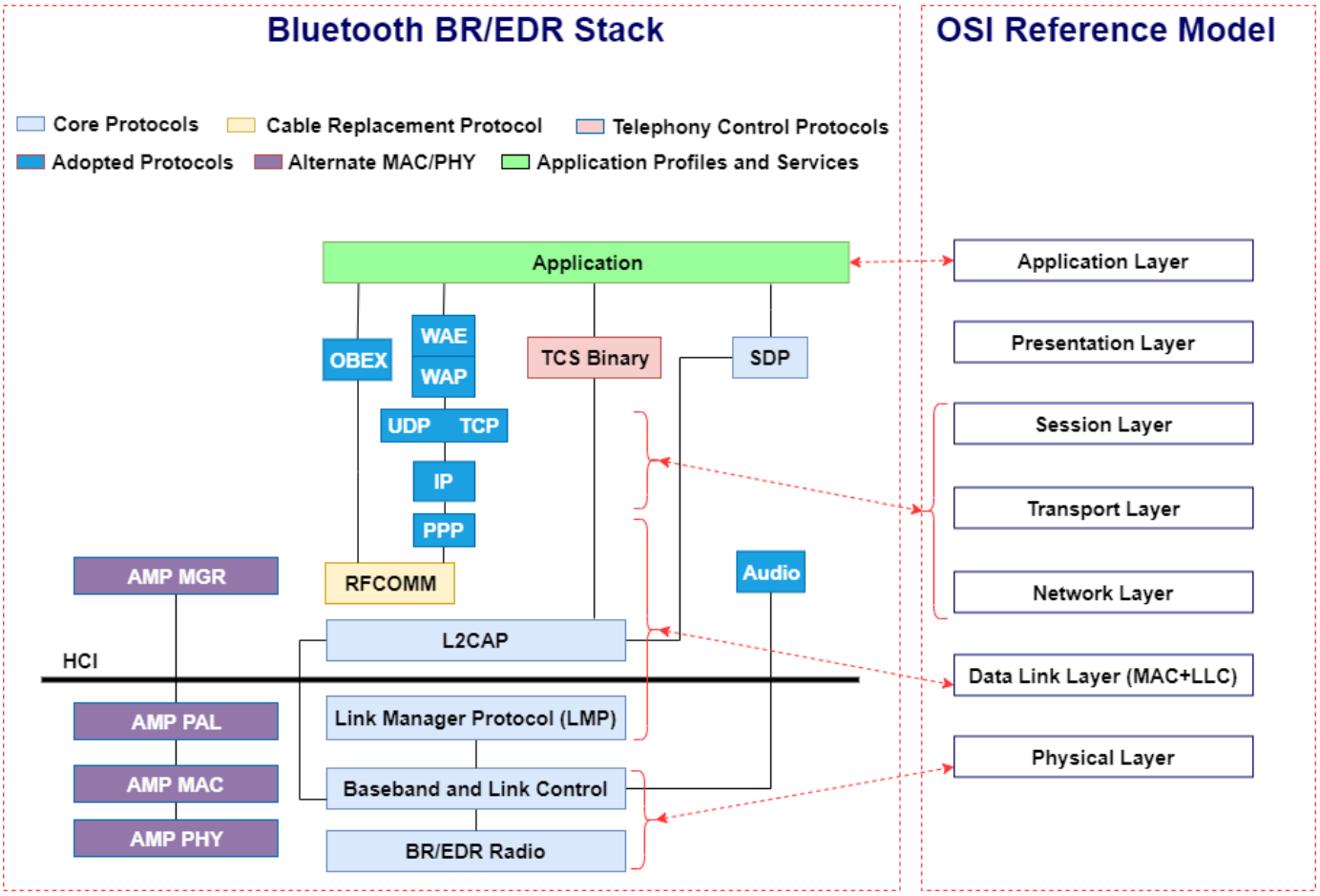
APP Layer

The App layer is the direct user interface defining profiles that afford interoperability between various applications. The Bluetooth core specification enables vendors to define proprietary profiles for use cases not defined by SIG profiles.

Note For more information about the Bluetooth LE protocol stack architecture, see Volume 3, Part C, Sections 2 and 2.1 of the Bluetooth Core Specification [1].

Bluetooth BR/EDR Protocol Stack

This figure compares the block diagram of the Bluetooth BR/EDR protocol stack and with the OSI reference model.



The mapping of BR/EDR stack to the OSI reference model is as shown below:

- The “BR/EDR Radio” on page 8-15 and “Baseband and Link Control” on page 8-15 layers of the Bluetooth BR/EDR stack map to the OSI PHY layer.
- The “Link Manager Protocol (LMP)” on page 8-15, “L2CAP” on page 8-15, “Cable Replacement Protocol” on page 8-15 (RFCOMM), and “PPP” on page 8-16 layers of the Bluetooth BR/EDR stack map to the OSI data link layer.
- The user datagram protocol (UDP), transmission control protocol (TCP), and internet protocol (IP) layers of Bluetooth BR/EDR stack map to a combined, network, transport and session layers of the OSI reference model.
- There is one-to-one mapping at the application layer.

Core Protocols

The Bluetooth core protocols and the Bluetooth radio are required by most of the Bluetooth devices. The core protocols include these layers.

BR/EDR Radio

The BR/EDR radio is the lowest defined layer of the Bluetooth specification. The BR mode is mandatory, whereas the EDR mode is optional. This layer defines the requirements of the Bluetooth transceiver device operating in the 2.4 GHz ISM frequency band. It implements a 1600 hops/sec FHSS technique. The radio hops in a pseudo-random way on 79 designated Bluetooth channels. Each Bluetooth channel has a bandwidth of 1 MHz. Each frequency is located at (2402 + k) MHz, where k = 0,1,...78. The modulation technique for BR and EDR mode is GFSK and differential phase shift keying (DPSK), respectively. The baud rate is 1 Msymbols/s. The Bluetooth BR/EDR radio uses the time division duplex (TDD) topology in which data transmission occurs in one direction at one time. The transmission alternates in two directions, one after the other.

Baseband and Link Control

The baseband and link control layer enables the PHY RF link between different Bluetooth devices, forming a piconet. The baseband handles the channel processing and timing, and the link control handles the channel access control. This layer provides these two different types of PHY RF links with their corresponding baseband packets:

- Synchronous connection-oriented (SCO) – Supports real-time audio traffic
- Asynchronous connection-oriented (ACL) – Supports data packet transmission

Link Manager Protocol (LMP)

The LMP layer is primarily responsible for link setup and link configuration between different Bluetooth devices. These processes include establishing security functions such as authentication and encryption by generating, exchanging, and checking link and encryption keys. Furthermore, this layer controls the power modes and duty cycles of the Bluetooth radio device and the connection states of a Bluetooth unit in a piconet.

L2CAP

The L2CAP adapts higher-layer protocols over the baseband. It shields the higher-layer protocols from the details of the lower-layer protocols. The L2CAP provides connection-oriented and connectionless services to the higher-layer protocols. This includes protocol multiplexing capability, segmentation and reassembly operations, and group abstractions.

SDP

Discovery services are an important aspect of the Bluetooth framework. The service discovery protocol (SDP) provides a means for applications to query services and the characteristics of services, following which a connection can be established between two or more Bluetooth devices. The SDP is quite different from service discovery in traditional network-based environments. The SDP is built on top of the L2CAP.

Cable Replacement Protocol

The cable replacement protocol in the Bluetooth BR/EDR stack uses RFCOMM to provide emulation of serial ports over L2CAP. RFCOMM emulates RS-232 control and data signals over the Bluetooth baseband and provides transport capabilities for higher-layer services that use a serial interface as a transport mechanism. RFCOMM also provides multiple simultaneous connections to one device and enables connections to multiple devices.

Telephony Control Protocols

The telephony control protocol specification, binary (TCS binary), defines the call control signaling to establish data and voice calls between Bluetooth devices. It is built on top of the L2CAP. Moreover, TCS binary defines mobility management procedures for handling Bluetooth devices.

Adopted Protocols

In addition to the core protocols, the Bluetooth BR/EDR stack includes protocols adopted from other standard bodies. These adopted protocols are defined in specifications issued by other standard-making organizations and are incorporated into the Bluetooth framework.

PPP

The point-to-point protocol (PPP) is an Internet Engineering Task Force (IETF) [3] standard protocol for transporting IP datagrams over a point-to-point link. The PPP runs over the RFCOMM to realize point-to-point connections.

TCP, UDP, and IP

These layers are the IETF-defined foundation protocols of the TCP/IP protocol suite.

- TCP – This protocol provides a reliable virtual connection between devices to realize data communication. The TCP treats the data as a stream of bytes and transmits them without any errors or duplication.
- UDP – This protocol is an alternative to the TCP and provides an unreliable datagram connection between devices. As there is no end-to-end connection in UDP, data is transmitted link-by-link without any guarantee of service.
- IP – This layer is a network layer protocol that enables a datagram service between devices, supporting both the TCP and UDP.

The use of the TCP, UDP, and IP in the Bluetooth BR/EDR stack enables communication with any other device connected to the Internet.

OBEX

The object exchange (OBEX) protocol is a session-level protocol developed by the Infrared Data Association (IrDA) to exchange objects. The OBEX protocol provides functionality similar to that of HTTP, but in a simpler manner. HTTP is an application layer protocol and layered above the TCP/IP. The OBEX protocol provides the client with a reliable transport for connecting to a server. It also provides a model for representing objects and operations.

WAE and WAP

Bluetooth BR/EDR stack incorporates the wireless application environment (WAE) and wireless application protocol (WAP) into its architecture. The advantages of using WAE/WAP features in the Bluetooth stack are:

- Build application gateways that act as an interface between WAP servers and some other application on the PC
- Provide functions such as remote control and data fetching from the PC to the Bluetooth handset
- Reuse the upper software application developed for the WAP application environment

Application Profiles and Services

For more information, see “APP Layer” on page 8-13.

Alternate MAC/PHY

The alternate MAC/PHY (AMP) manager is a secondary controller in the Bluetooth core system. After an L2CAP connection is established between two devices over the BR/EDR radio, the AMP manager can discover the AMPs that are available on the other device. If an AMP is common between two devices, the Bluetooth core system provides mechanisms for moving data traffic from the BR/EDR controller to an AMP controller.

Each AMP manager consists of a protocol adaptation layer (PAL) on top of a MAC and PHY. The PAL maps the Bluetooth protocols to the specific protocols of the underlying MAC and PHY.

L2CAP channels can be created on, or moved to, an AMP. If an AMP physical link has a link supervision timeout, then L2CAP channels can be moved back to BR/EDR radio. To minimize power consumption in the device, AMPs are enabled or disabled as required.

HCI

The HCI provides a command interface to the BR/EDR radio, baseband controller, and the link manager. It is a single standard interface for accessing the Bluetooth baseband capabilities, the hardware status, and the control registers.

Note For more information about the Bluetooth BR/EDR protocol stack architecture, see Volume 1, Part A, Sections 2 and 2.1 of the Bluetooth Core Specification [1].

References

[1] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[2] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 6, 2021. https://www.bluetooth.com/.

[3] IETF. “Internet Standards.” Accessed November 6, 2021. https://www.ietf.org/.

[4] Bluetooth Protocol Stack - an Overview | ScienceDirect Topics. Accessed November 15, 2021. https://www.sciencedirect.com/.

See Also

More About

- “Bluetooth Technology Overview” on page 8-2
- “Comparison of Bluetooth BR/EDR and Bluetooth LE Specifications” on page 8-6
- “Bluetooth Packet Structure” on page 8-27

Bluetooth Location and Direction Finding

Bluetooth technology [1] uses low-power radio frequency to enable short-range communication at a low cost. The Bluetooth Core Specification [2] provided by the Bluetooth Special Interest Group (SIG) added a location and direction finding feature in the Bluetooth low energy (LE). The communication in Bluetooth LE is realized using these two distinct physical layers (PHYs).

- LE Uncoded: This PHY is further segregated into LE 1M PHY and LE 2M PHY. LE 1M is the default PHY and provides a symbol rate of 1 Msym/s. Support for LE 1M is mandatory in all devices that support Bluetooth LE. LE 2M provides a symbol rate of 2 Msym/s. The support for LE 2M is optional for the devices supporting the Bluetooth LE controller.
- LE Coded: This PHY is equipped for longer range communication. It has the potential to quadruple the range that can be achieved whilst reducing the data rate. Support for LE Coded PHY is optional for devices supporting the Bluetooth LE controller.

Bluetooth direction finding can use either LE 1M or LE 2M PHY, but not the LE Coded PHY.

Location and Direction Finding Services in Bluetooth

For several years, Bluetooth has been used to provide different types of location and direction finding services. On a high-level, these services can be split into two categories.

- Proximity solutions: This category consists of point of interest (PoI) information applications (for example, museums that provide the user information about the artefacts in the room). This category also includes item-finding solutions such as Bluetooth tags that help to find lost or misplaced items. In these solutions, the Bluetooth tags periodically transmit Bluetooth LE broadcast frames. The access point (AP) scans these frames to obtain the Bluetooth tag information and sends it to the location server through the access controller (AC). In PoI proximity applications, determining what point or PoIs are in close proximity of the calculated location is necessary.
- Positioning systems: This category includes location-based services to leverage Bluetooth to find the physical position of the device. The prominent use case examples in this category are real-time locating systems used for asset tracking, people tracking, and indoor positioning systems used to enable path-finding solutions that help people navigate through intricate indoor scenarios. Indoor positioning use cases need applications that estimate the accurate location of the beacons they encounter so that the location of the tracked device corresponding to the known location of the beacon can be calculated. In some cases, the position of a beacon might need to be determined in three dimensions, considering its x- and y-coordinates in horizontal plane and its elevation above or below some reference altitude. The application can determine the position of its host device only if it knows the direction from which the received signal is coming, the approximate distance to that beacon, and the location of the beacon.

  In applications involving smartphones, when calculating the direction of the signal, the application must consider the orientation in three dimensional space of the phone.

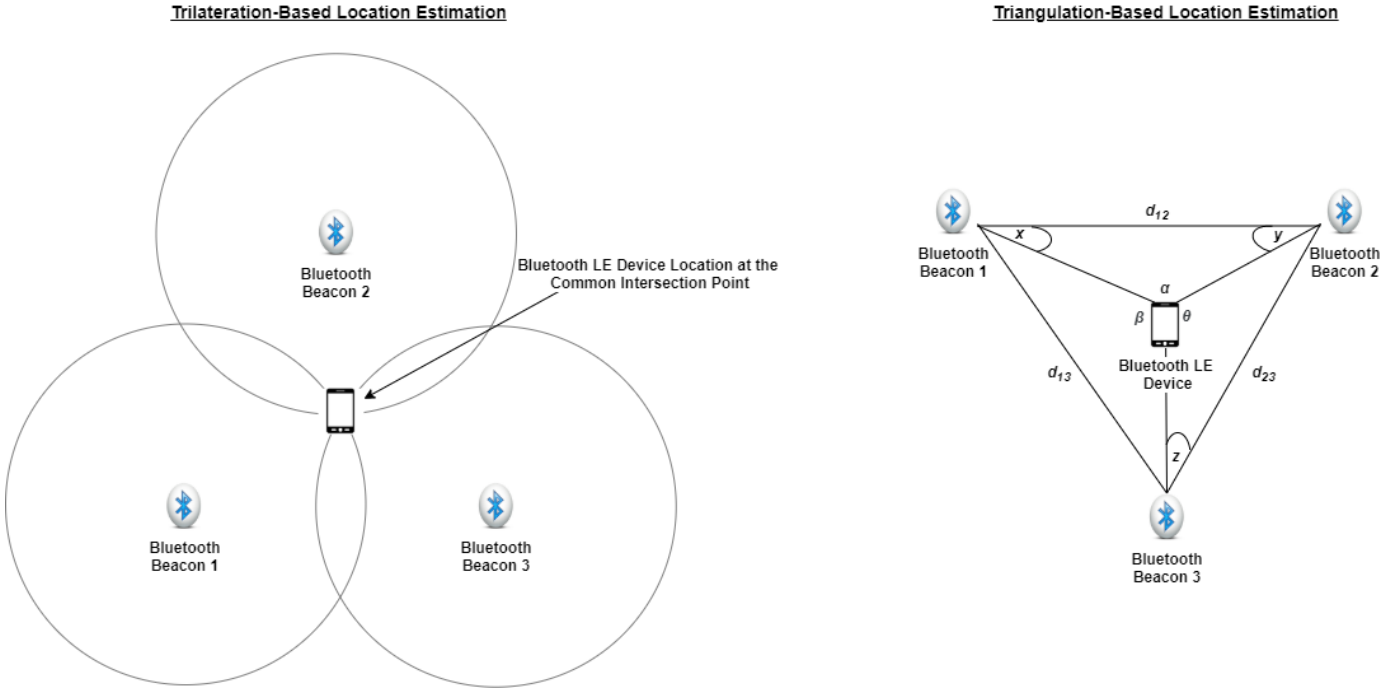
Beyond the previously mentioned location-finding services, the applications themselves must undertake these common considerations.

- Determining antenna array details: To accurately receive and process IQ sample data, applications must have details of the antenna array in the local device (for angle of arrival (AoA)) or remote device (angle of departure (AoD)). Application profiles describe how applications can obtain the antenna array description from remote devices. Expect the APIs to emerge for retrieving details of antenna arrays in local and remote devices.

- Configuring constant tone extension (CTE) parameters: Parameters such as the length of the CTE, the length of antenna switching pattern, and the number of packets that include the CTE to transmit per periodic advertising event govern the CTE production. These parameters can be set through new host controller interface (HCI) commands.
- Configuring and enabling IQ sampling: The Bluetooth Core Specification [2] defines a series of parameters to configure and initiate IQ sampling. These parameters include sample slot duration (either 1 µs or 2 µs), the length of the switching pattern, and the IDs of the antennas to be included in the sampling pattern.
- Developing algorithms and calculating angles from IQ sample data: The Bluetooth SIG does not designate any one particular algorithm as the standard direction-finding algorithm. The choice of algorithm is left to the application layer to address. Generally, this is the area in which manufacturers and developers compete.

Location Estimation Techniques in Bluetooth

Bluetooth beacon technology is an application of the Bluetooth LE standard. A beacon broadcasts a distinct ID. An application on a Bluetooth LE device receiving that ID looks into a database to recognize the transmitting beacon and then provides the user with information related to the location of that beacon. This figure shows the techniques used to estimate the distance between the Bluetooth LE device and the beacon.



- Trilateration-based location estimation: Trilateration is one of the most commonly used technique to estimate the device location. In this technique, the locations of at least two reference Bluetooth beacons and the distance between them must be known. However, to accurately determine the relative location of a node, three beacons are needed. The trilateration technique uses the received signal strength indicator (RSSI) value to compute the distance between the Bluetooth beacons and the Bluetooth LE device. The RSSI value helps to determine the proximity of two Bluetooth LE devices by providing meter-level accuracy. The RSSI value indicates the strength of the beacon's signal as seen from the receiving Bluetooth LE device. As the RSSI value increases,

the beacon signal strengthens. This relationship helps indicate when the Bluetooth LE device is in close proximity of the beacon. Because the direction of the beacon signal cannot be determined by trilateration, the location of Bluetooth LE device can be at any point on the circumference of the circle. However, the ideal location of the Bluetooth LE device is at the common intersection point of the three circles. Due to lack of information related to the direction of the beacon signal, the three circles might not always have a common intersection point.

In trilateration, the advantage of using the RSSI value is that it does not need any additional hardware or incur any additional communication overhead. On the contrary, the accuracy of the RSSI-based approach is impeded by the accuracy of the path-loss model you select. Also, this approach is not accurate enough for several use cases. Even if the reference RSSI value is efficiently calibrated when first installing the Bluetooth beacon, the calculated RSSI value is influenced by the environmental conditions such as the presence of people and humidity levels. The RSSI-based approach gives poor accuracy, particularly in indoor scenarios that are filled with obstacles such as walls and furniture. These obstacles are the source of multipath fading and make the relation between distance and RSSI inaccurate.

- Triangulation-based location estimation: Triangulation is a technique of calculating the position of a point that relies on a known distance between two or three reference points and the angles measured using the Bluetooth direction finding feature between those reference points to that point. These angles can be AoA or AoD. For more information, see “Angle of Arrival (AoA)” on page 8-21 and “Angle of Departure (AoD)” on page 8-21. Unlike trilateration, which implements only the distance measurements, the triangulation technique uses angle measurements. With this technique, you can calculate the location of any point in 2-D given the three angles between the point and other three reference points. However, in 2-D space, a minimum of two angles is required to estimate the location of any point. With reference to the preceding figure, d_{12}, d_{23}, and d_{13} denote the distances between the Bluetooth beacons 1–2, 2–3, and 1–3 respectively. Angles x, y, and z are the known angle measurements between the Bluetooth LE device and Bluetooth beacons 1, 2, and 3, respectively. Using these known measurements, the triangulation technique enables you to compute angles α, β, and θ. Consequently, the location of the Bluetooth LE device is obtained. Triangulation is a complex technique that requires information about the location and spatial rotation of the Bluetooth beacons. However, due to AoA and AoD capabilities, triangulation gives a more accurate location of the Bluetooth LE device as compared to the trilateration technique.

To accurately determine the Bluetooth LE device location, more advanced solutions must implement multiple Bluetooth beacons and complex algorithms based on trilateration and triangulation techniques.

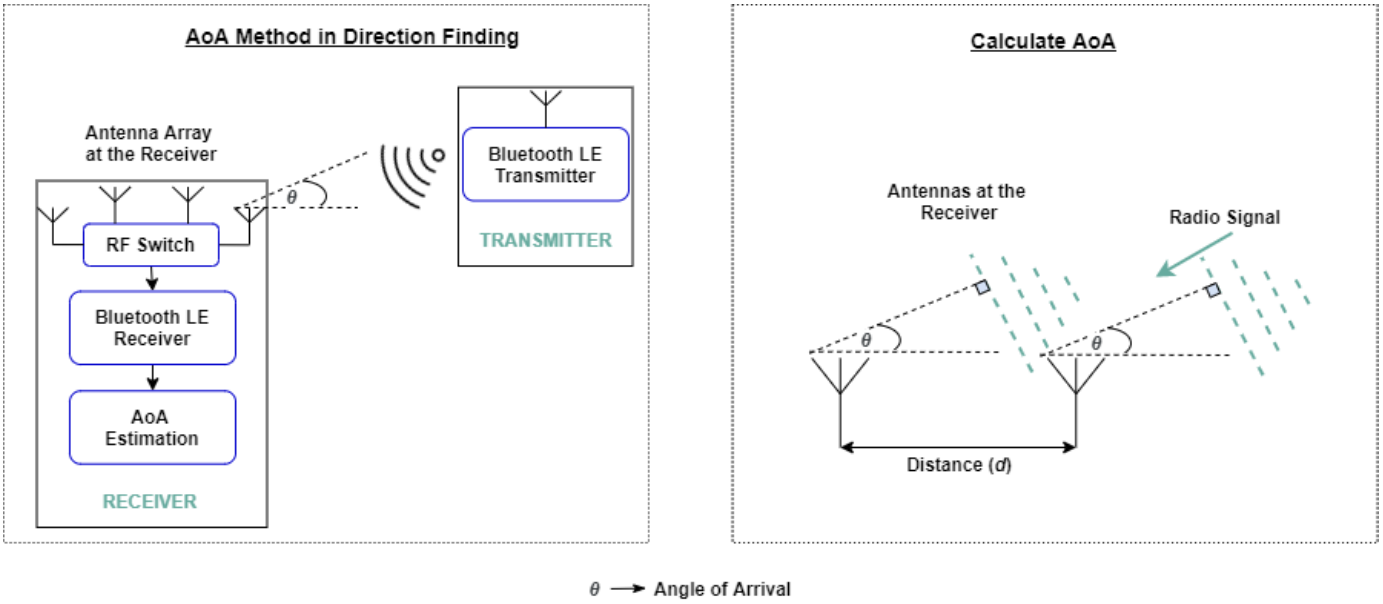
Bluetooth Direction-Finding Capabilities

The Bluetooth Core Specification [2] introduced new features that support high-accuracy direction finding. The controller specification is enhanced so that the specialized hardware that incorporates an antenna array can be used to determine the direction of a received Bluetooth LE signal. The HCI is modified so that data acquired by the controller can be made available to higher layers of the stack where direction calculations can take place. Bluetooth direction finding offers two distinct methods, each of which exploits the same underlying basis. These direction finding methods are – AoA and AoD.

Note Bluetooth direction-finding capabilities, AoA and AoD, are introduced in the Bluetooth Core Specification 5.1 [2].

Angle of Arrival (AoA)

A Bluetooth LE device can send its direction-related information to another peer Bluetooth LE device by transmitting direction-finding enabled packets using a single antenna. The peer Bluetooth LE device consisting of an RF switch and an antenna array switches antennas and captures the received in-phase (I) and quadrature (Q) samples. The Bluetooth LE device uses these I and Q samples to compute the phase difference in the radio signal received by various elements of the antenna array. Consequently, the calculated phase difference is used to estimate the AoA. This figure illustrates the concept of the AoA method.



The transmitter device uses a single antenna, whereas the receiver device uses an antenna array handled by the RF switch. At the receiver, d denotes the distance between two antennas. The phase difference, ψ, between the signals arriving at the two antennas is calculated as:

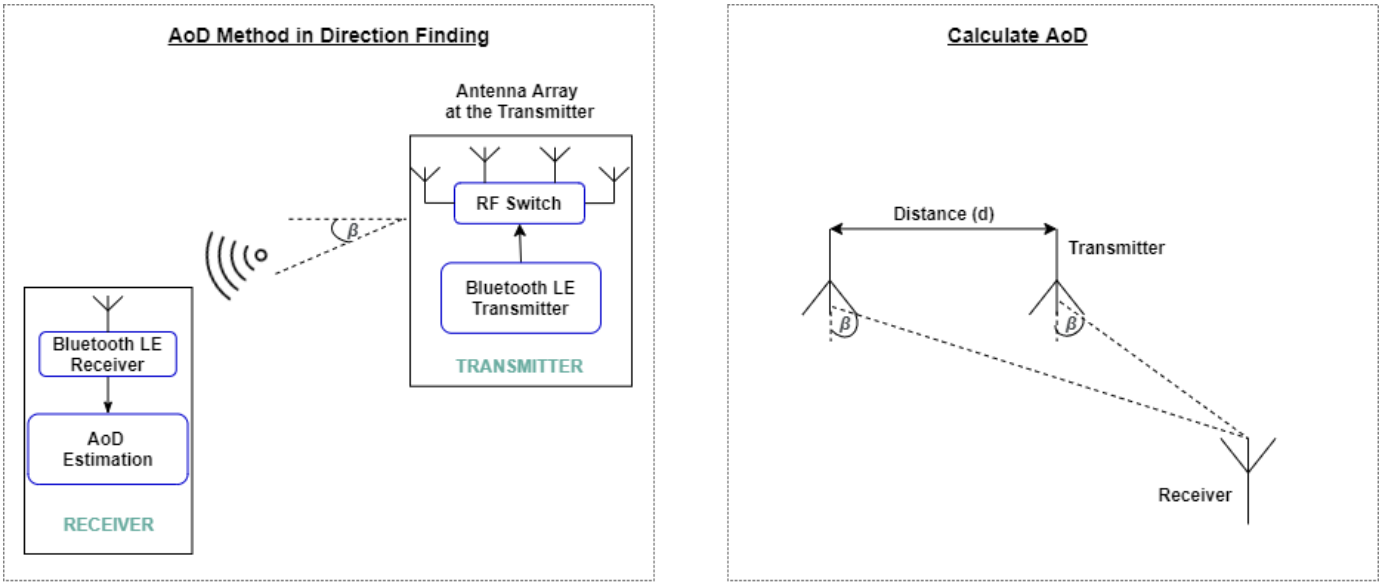
$$\psi = \frac{2\pi d \cos\theta}{\lambda}$$

λ is the signal wavelength and θ is the AoA. To avoid the aliasing effect, the maximum value of d must be λ/2. Rearranging the above equation, the AoA is calculated as:

$$\theta = \cos^{-1}\left(\frac{\psi\lambda}{2\pi d}\right)$$

Angle of Departure (AoD)

Unlike in AoA, the AoD method consists of a single antenna at the receiver and multiple antennas at the transmitter. A Bluetooth LE transmitter consisting of an RF switch and antenna array can make its AoD detectable by sending direction-finding packets and then switching antennas in the antenna array during the transmission. The Bluetooth LE receiver receives the packets using a single antenna and captures the I and Q samples. The direction of the signal is determined from different propagation delays of the Bluetooth LE signal between multiple antennas of the antenna array and the single receiving antenna. This figure illustrates the concept of the AoD method.



The receiver device uses a single antenna, whereas the transmitter device has an antenna array handled by the RF switch. At the transmitter, d denotes the distance between two antennas. The phase difference, ψ, between the signals arriving at the two antennas is calculated as:

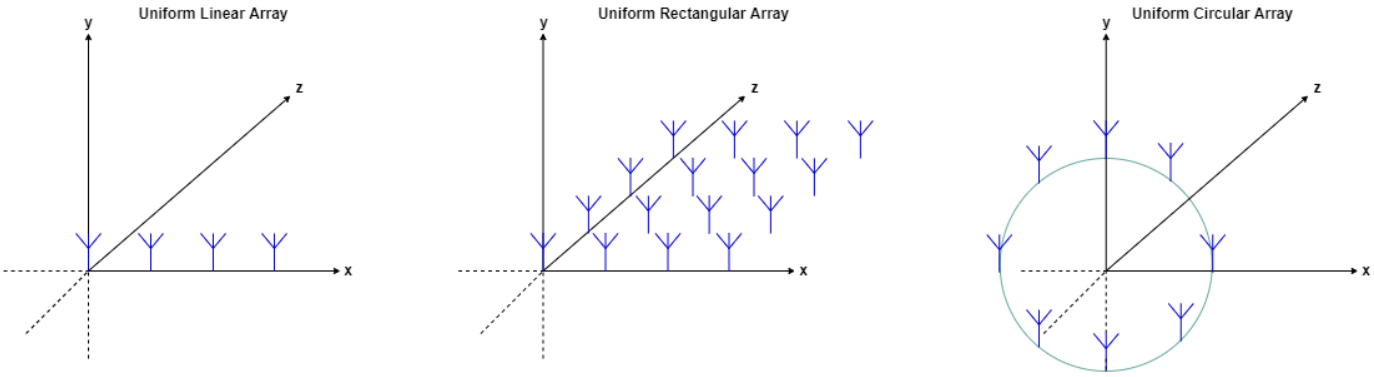
$$\psi = \frac{2\pi d \sin\beta}{\lambda}$$

λ is the signal wavelength and β is the AoD. By rearranging the above equation, AoD is calculated as:

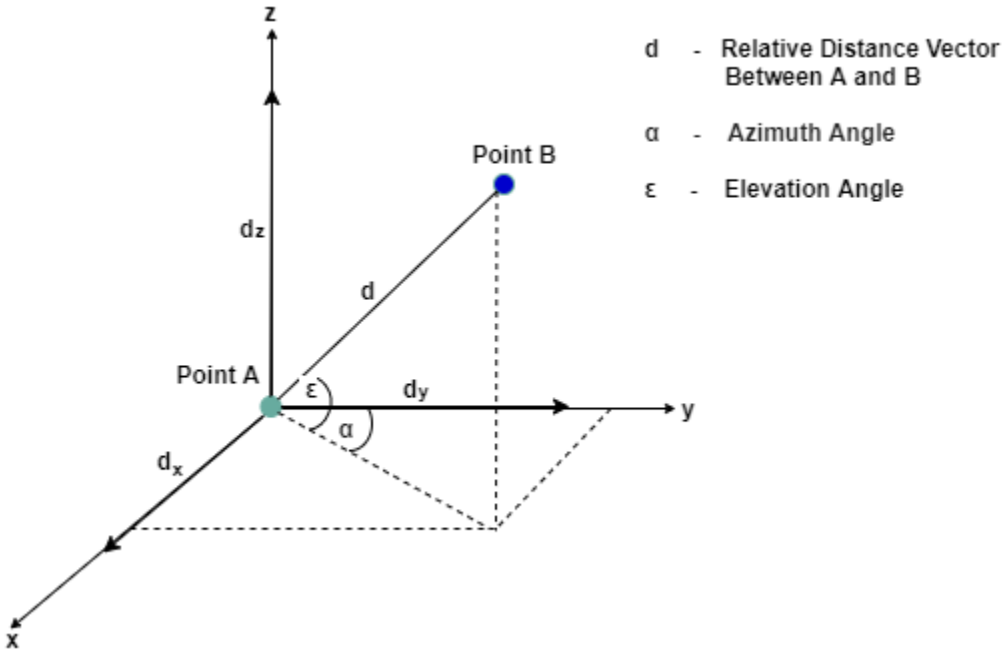
$$\beta = \sin^{-1}\left(\frac{\psi\lambda}{2\pi d}\right)$$

Antenna Arrays

The fundamental use of antenna arrays is to direct a radiated signal toward a desired angular sector. The number, geometrical design, relative amplitudes, and relative phases of the elements of the antenna array depend on the desired angular pattern. Once the antenna array is designed to focus in a specific direction, the array can also be steered in another direction by changing the relative phases of the array elements. This figure shows some commonly used antenna array designs.



In the uniform linear array (ULA) case, antenna elements are located in a single line. In the uniform rectangular array (URA) case, antenna elements are positioned along a rectangular grid. The uniform circular array (UCA) enables antenna elements to be placed along the circumference of the circle. The geometrical designs of ULAs are simple and enable only a single angle to be calculated from a signal. More complex antenna array designs can enable two or three angles to be determined. Calculating the elevation and azimuth angles of the signal relative to a reference plane is common in these antenna arrays. This figure shows the concept of elevation and azimuth angles.



d is the relative distance vector between points A and B. d_x, d_y, and d_z denote the components of d along x-, y-, and z-axis, respectively. Using this information, the azimuth angle (α) and elevation angle (ϵ) between points A and B is calculated as:

$$\alpha = \tan^{-1}(\frac{d_y}{d_x})$$

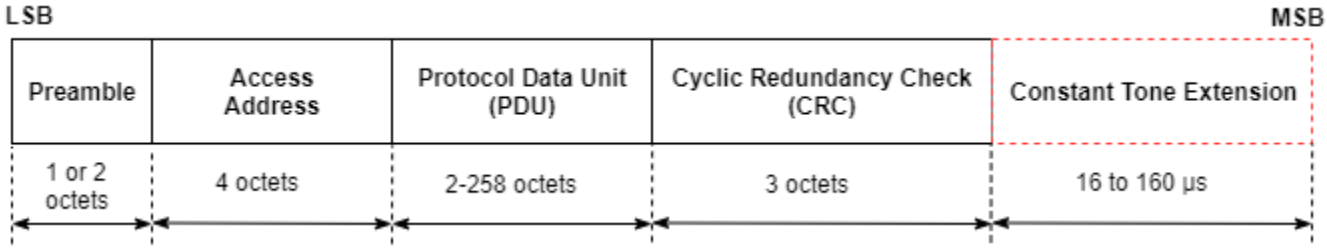
$$\varepsilon = \tan^{-1}(\frac{d_z}{\sqrt{d_x^2 + d_y^2}})$$

Bluetooth Direction-Finding Signals

Bluetooth direction-finding signals are an important part of the Bluetooth direction-finding technique. Direction-finding signals provide a source of constant signal material to which the IQ sampling can be applied. New link layer (LL) protocol data units (PDUs) are identified for direction finding between two connected Bluetooth LE devices. Moreover, the Bluetooth Core Specification [2] enables you to use existing advertising PDUs for connectionless direction-finding purposes. In these cases, additional information known as CTE is appended to the PDUs. To calculate AoA and AoD, the Bluetooth direction-finding signals use these Bluetooth LE packet structure fields.

Constant Tone Extension (CTE)

This figure shows the packet structure for the Bluetooth LE uncoded PHY operating on LE 1M and LE 2M. The CTE field is appended at the end of the packet structure.



CTE contains a series of symbols, each representing a binary 1. The number of symbols in the CTE field are configured by the higher layers so that a suitable amount of data and time is available for IQ sampling.

Note For more information about the CTE, see Volume 6, Part B, Section 2.5.1 of the Bluetooth Core Specification [2].

Frequency Deviation

In a given radio channel, Bluetooth uses two frequencies, one to denote digital 0s and the other to denote digital 1s. These two frequencies are computed by adding or subtracting the *frequency deviation* to or from the center frequency of the channel. Any change in the frequency also changes the wavelength. The wavelength is an important factor in calculating direction from IQ samples. Therefore, CTE consists solely of digital 1s. This implies that the entire CTE is transmitted at a single frequency and has a constant wavelength.

Cyclic Redundancy Check (CRC)

Each Bluetooth LE packet contains a CRC field that is used in error detection. The Bluetooth LE transmitter calculates a CRC value from the remainder of the packet to be transmitted, appends the CRC to the end of the packet, and transmits the packet. The Bluetooth LE receiver performs the same calculation and compares the computed CRC value with the appended CRC value. If the CRC values are unequal, a communication error has occurred. This causes a change in one or more of the transmitted bits. In this case, the packet is ignored by the Bluetooth LE receiver and can be retransmitted by the Bluetooth LE transmitter.

Note The CTE value in the direction-finding packets is not included in the CRC calculation.

Message Integrity Check (MIC)

If a connection between the Bluetooth LE transmitter and receiver is encrypted and authenticated, the LL PDU includes a MIC field. The MIC value is used to authenticate the sender of the PDU.

Note The CTE value in the direction-finding packets is not included in the MIC calculation.

Whitening

Whitening refers to the process of scrambling the bits to avoid lengthy sequences of 1s and 0s in the transmitted bit stream. The lengthy sequences of 1s and 0s might cause the receiver to lose its frequency lock and act as though the center frequency has moved up or down. Bluetooth LE uses whitening to scramble the PDU and CRC fields of all LL packets.

Note The CTE value in the direction-finding packets is not subject to the whitening process.

Connectionless and Connection-Oriented Direction Finding

The Bluetooth Core Specification [2] enables the AoA and AoD to be used in either connectionless or connection-oriented communication. However, in typical use cases, the AoD is used with connectionless communication and the AoA is used with connection-oriented communication. This table shows four possible permutations of using the AoA and AoD with connectionless and connection-oriented communication.

Type of Connection	AoA	AoD
Connectionless	Bluetooth LE controller support is optional.	Bluetooth LE controller support is optional. Using the AoD with connectionless communication is typical.
Connection-oriented	Bluetooth LE controller support is optional. Using the AoA with connection-oriented communication is typical.	Bluetooth LE controller support is optional.

Connectionless direction finding implements Bluetooth periodic advertising. The CTE is appended to otherwise standard AUX_SYNC_IND PDUs. Connection-oriented direction finding conveys the CTE using new LL_CTE_RSP packets that are transmitted over the connection as an acknowledgment to LL_CTE_REQ PDUs. In both of these cases, a variety of setup and configuration steps must be completed before IQ sampling is initiated and the CTE-bearing packets are generated.

With the Bluetooth-direction finding capability, the proximity and positioning systems operating at submeter accuracy can be developed for use cases such as indoor positioning, path finding, asset tracking, and directional discovery. The Bluetooth direction-finding capability elevates proven engineering techniques for signal direction. This capability also standardizes the interfaces, interactions, and prominent intrinsic operations of the Bluetooth LE stack. Precise direction finding is now interoperable across different manufacturers and can be widely adopted to create a new generation of advanced Bluetooth location and direction finding services.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.1. https://www.bluetooth.com/.

[3] Suryavanshi, Nitesh B., K. Viswavardhan Reddy, and Vishnu R. Chandrika. “Direction Finding Capability in Bluetooth 5.1 Standard.” *In Ubiquitous Communications and Network Computing*, edited by Navin Kumar and R. Venkatesha Prasad, 53–65. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Cham: Springer International Publishing, 2019.

[4] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions

`bleAngleEstimate` | `bleCTEIQSample` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects

`bleAngleEstimateConfig`

More About

- “Parameterize Bluetooth LE Direction Finding Features” on page 9-10
- “Estimate Bluetooth LE Node Position” on page 9-35
- “Bluetooth LE Positioning by Using Direction Finding” on page 3-2
- “Bluetooth LE Direction Finding for Tracking Node Position” on page 3-15

Bluetooth Packet Structure

The Bluetooth Special Interest Group (SIG) [1] and [2] defines different packet structures for Bluetooth low energy (LE) and Bluetooth basic rate/enhanced data rate (BR/EDR) devices.

Bluetooth LE Packet Structure

Bit Ordering in Bluetooth LE Packets

When defining packets and messages in the baseband specification, the bit ordering follows the little-endian format. In this format, these rules apply:

- The least significant bit (LSB) corresponds to b_0.
- LSB is the first bit sent over the air.
- When illustrating the packet structure, the LSB is shown on the left side.

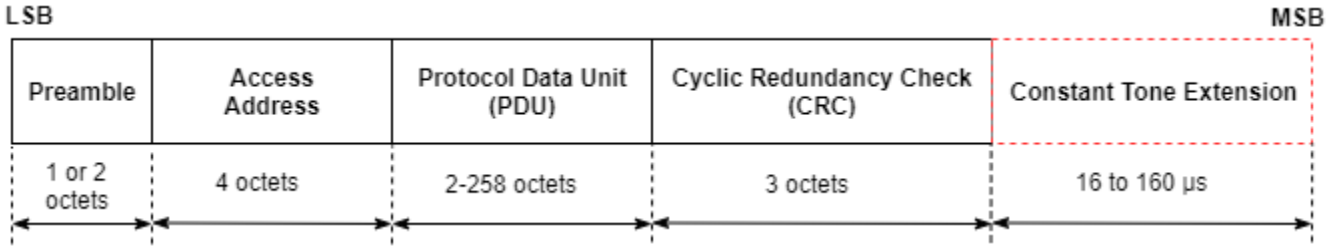
Moreover, data fields generated internally at the baseband level (packet header and payload header length), must be transmitted with the LSB first. For example, a 3-bit parameter is sent as: $b_0b_1b_2$ = 110 over the air, where 1 is sent first and 0 is sent last.

The Bluetooth LE devices use packet formats for: “Bluetooth LE Uncoded Physical Layer (PHY)” on page 8-27, “Bluetooth LE Coded PHY” on page 8-28, “Advertising Physical Channel PDU” on page 8-30, “Data Physical Channel PDU” on page 8-31, and “Constant Tone Extension and In-Phase Quadrature (IQ) Sampling” on page 8-34.

Note For more information about Bluetooth LE packet structure, see Volume 6, Part B, Section 2 of the Bluetooth Core Specification [2].

Bluetooth LE Uncoded Physical Layer (PHY)

The Bluetooth Core Specification [2] defines two physical layer (PHY) transmission modes (LE 1M and LE 2M) for uncoded PHY. This figure shows the packet structure for the Bluetooth LE uncoded PHY operating on LE 1M and LE 2M.



Each packet contains four mandatory fields (preamble, access-address, protocol data unit (PDU), and cyclic redundancy check (CRC)) and one optional field (constant tone extension (CTE)). The preamble is transmitted first, followed by the access address, PDU, CRC, and CTE (if present) in that order. The entire packet is transmitted at the same symbol rate of 1 Msym/s or 2 Msym/s.

Preamble

All link layer (LL) packets contain a preamble, which is used in the receiver to perform frequency synchronization, automatic gain control (AGC) training, and symbol timing estimation. The preamble

is a fixed sequence of alternating 0 and 1 bits. For the Bluetooth LE packets transmitted on the LE 1M PHY and LE 2M PHY, the preamble size is 1 octet and 2 octets, respectively.

Access address

The access address is a 4-octet value. Each LL connection between any two devices and each periodic advertising train has a distinct access address. Each time the Bluetooth LE device needs a new access address, the LL generates a new random value adhering to these requirements:

- The value must not be the access address for any existing LL connection on this device.
- The value must not be the access address for any enabled periodic advertising train.
- The value must have no more than six successive 1s or 0s.
- The value must not be the access address for any advertising channel packets.
- The value must not be a sequence that differs from the access address of advertising physical channel packets by only 1 bit.
- All four octets for the value must not be equal.
- The value must have a minimum of two transitions in the most significant 6 bits.

If the random value does not satisfy the above requirements, a new random value is generated until it meets all of the requirements.

PDU

When a Bluetooth LE packet is transmitted on either the primary or secondary advertising physical channel or the periodic physical channel, the PDU is defined as the “Advertising Physical Channel PDU” on page 8-30. When a packet is transmitted on the data physical channel, the PDU is defined as the “Data Physical Channel PDU” on page 8-31.

CRC

The size of the CRC is 3 octets and is calculated on the PDU of all LL packets. If the PDU is encrypted, then the CRC is calculated after encryption of the PDU is complete. The CRC polynomial has the form $x^{24}+x^{10}+x^9+x^6+x^4+x^3+x+1$.

For more information about CRC generation, see Volume 6, Part B, Section 3.1.1 of the Bluetooth Core Specification [2].

CTE

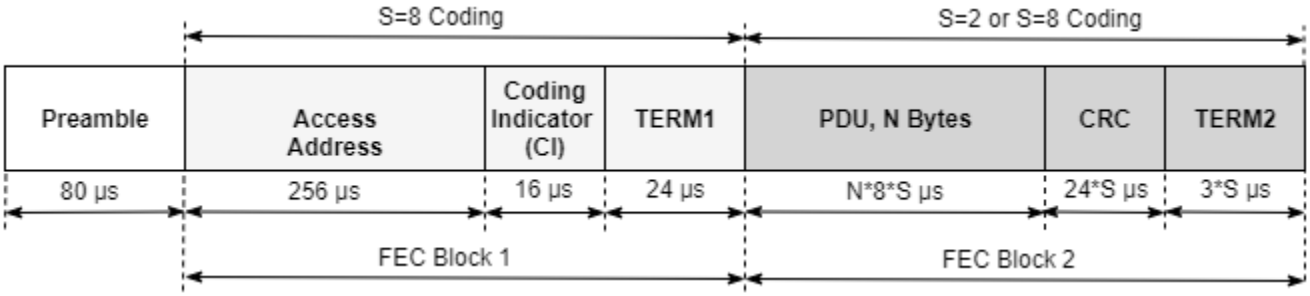
The CTE consists of a constantly modulated series of unwhitened 1s. This field has a variable length that ranges from 16 µs to 160 µs.

For more information about the CTE, see Volume 6, Part B, Section 2.5.1 of the Bluetooth Core Specification [2].

Note For more information about Bluetooth LE uncoded PHY packet structure, refer to Volume 6, Part B, Section 2.1 of Bluetooth Core Specification [2].

Bluetooth LE Coded PHY

This figure shows the packet structure for the Bluetooth LE coded PHY and is implemented for Bluetooth LE packets on all the physical channels.



Each Bluetooth LE packet consists of a preamble and these two forward error correcting (FEC) blocks:

- FEC block 1— This block contains three fields: access address, coding indicator (CI), and TERM1. This block implements an *S*=8 coding scheme, where each bit represents eight symbols. This gives a data rate of 125 Kbps.
- FEC block 2— This block contains these three fields: PDU, CRC, and TERM2. This block implements an *S*=8 or *S*=2 coding scheme. In the *S*=2 coding scheme, each bit represents two symbols. Therefore, the data rate is 500 Kbps.

The Bluetooth LE coded PHY does not contain the CTE.

Preamble

The Bluetooth LE coded PHY preamble is 80 symbols in length and contains 10 repetitions of the symbol pattern '00111100' (in the transmission order).

Access address

The length of Bluetooth LE coded PHY access address is 256 symbols. For more information, see “Access address” on page 8-28. In addition to the requirements listed in the access address subsection of the “Bluetooth LE Uncoded Physical Layer (PHY)” on page 8-27 section, the new value for the access address of the Bluetooth LE coded PHY must also meet these requirements:

- The value must have at least three 1s in the last significant bits.
- The value must have no more than 11 transitions in the least significant 16 bits.

CI

The CI field consists of two bits as shown in this table:

Bits in CI	Description
`00b`	FEC block 2 coded using *S*=8
`01b`	FEC block 2 coded using *S*=2
All other values	Reserved for future use

PDU

The PDU in the Bluetooth LE coded PHY packet structure has the same formatting as the “PDU” on page 8-28 in the Bluetooth LE uncoded PHY packet.

CRC

The CRC in the Bluetooth LE coded PHY packet structure has the same formatting as the “CRC” on page 8-28 in the Bluetooth LE uncoded PHY packet.

TERM1 and TERM2

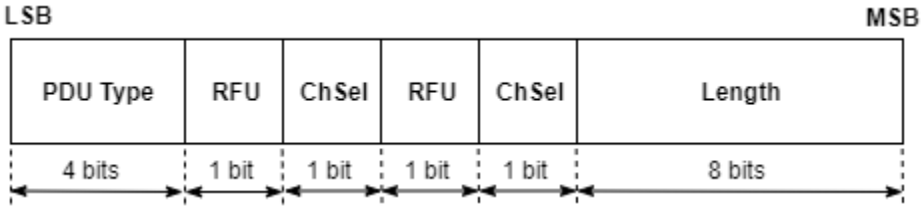
Each FEC block contains a terminator at the end of the block. That terminator is referred to as TERM1 and TERM2. Each terminator is 3 bits long and forms the termination sequence during the FEC encoding process.

Note For more information about Bluetooth LE coded PHY packet structure, see Volume 6, Part B, Section 2.2 of the Bluetooth Core Specification [2].

Advertising Physical Channel PDU

The packet structure format of the advertising physical channel PDU is shown in this figure.

The advertising physical channel PDU has a 16-bit header and a variable-size payload. The 16-bit header field of the advertising physical channel PDU is shown in this figure.



The PDU type field in the advertising channel PDU header defines different types of PDUs that can be transmitted on the Bluetooth LE coded PHY. This table maps different types of PDUs with the physical channels and the PHYs on which the Bluetooth LE packet might appear. The table also indicates the PHY transmission modes supported for each type of advertising physical channel PDU.

PDU Type	PDU Name	Physical Channel	LE 1M Support	LE 2M Support	LE Coded Support
0000b	ADV_IND	Primary Advertising	Yes		
0001b	ADV_DIRECT_IND	Primary Advertising	Yes		
0010b	ADV_NONCONN_IND	Primary Advertising	Yes		
0011b	SCAN_REQ	Primary Advertising	Yes		
	AUX_SCAN_REQ	Secondary Advertising	Yes	Yes	Yes

PDU Type	PDU Name	Physical Channel	LE 1M Support	LE 2M Support	LE Coded Support
0100b	SCAN_RSP	Primary Advertising	Yes		
0101b	CONNECT_IND	Primary Advertising	Yes		
	AUX_CONNECT_REQ	Secondary Advertising	Yes	Yes	Yes
0110b	ADV_SCAN_IND	Primary Advertising	Yes		
0111b	ADV_EXT_IND	Primary Advertising	Yes		Yes
	AUX_ADV_IND	Secondary Advertising	Yes	Yes	Yes
	AUX_SCAN_RSP	Secondary Advertising	Yes	Yes	Yes
	AUX_SYNC_IND	Periodic	Yes	Yes	Yes
	AUX_CHAIN_IND	Secondary Advertising and Periodic	Yes	Yes	Yes
1000b	AUX_CONNECT_RSP	Secondary Advertising	Yes	Yes	Yes
All other values	Reserved for future use				

The RFU field is reserved for future use. The `ChSel`, `TxAdd`, and `RxAdd` fields of the advertising physical channel PDU header contain information specific to the type of PDU defined for each advertising physical channel PDU separately. If the `ChSel`, `TxAdd`, or `RxAdd` fields are not defined as used in a given PDU, then they are considered as reserved for future use.

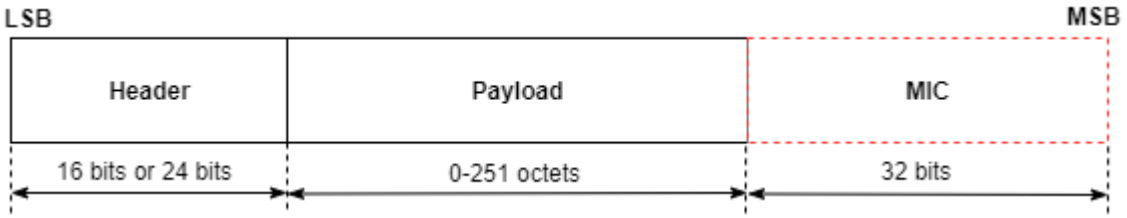
The `Length` field of the advertising physical channel PDU header denotes the length of the payload in octets. The valid range of the Length field is 1 to 255 octets.

The `Payload` field in the advertising physical channel PDU packet structure is specific to the type of PDUs listed in the preceding table.

Note For more information about advertising physical channel PDUs, see Volume 6, Part B, Section 2.3 of the Bluetooth Core Specification [2].

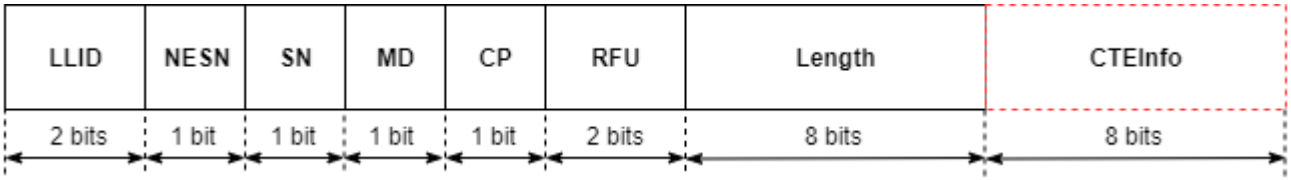
Data Physical Channel PDU

The packet structure format of the data physical channel PDU is shown in this figure.



The data physical channel PDU has a 16-bit or 24-bit header, a variable length payload in the range [0, 251] octets, and can include a 32-bit message integrity check (MIC) field. The MIC is not included in an unencrypted LL connection or in an encrypted LL connection with a data channel PDU containing an empty payload. The MIC is included in an encrypted LL connection with a data channel PDU containing a nonzero length payload. In this case, the MIC is calculated as specified in Volume 6, Part E, Section 1 of the Bluetooth Core Specification [2].

The header field of the data physical channel PDU is shown in this figure.



The data physical channel PDU header includes these fields:

- Link layer identifier (LLID) — This field indicates whether the packet is an LL data PDU or LL control PDU.
  - `00b` — Reserved for future use
  - `01b` — LL Data PDU, which can be a continuation fragment of an logical link control and adaptation (L2CAP) message or an empty PDU
  - `10b` — LL Data PDU, which can be a start of an L2CAP message or a complete L2CAP message with no fragmentation
  - `11b` — LL control PDU
- Next expected sequence number (NESN): The LL uses this field to either acknowledge the last data physical channel PDU sent by the peer or to request the peer to resend the last data physical channel PDU. For more information about NESN, see Volume 6, Part B, Section 4.5.9 of the Bluetooth Core Specification [2].
- Sequence number (SN): The LL uses this field to identify the Bluetooth LE packets sent by it. For more information about the SN, see Volume 6, Part B, Section 4.5.9 of the Bluetooth Core Specification [2].
- More data (MD): This field indicates that the Bluetooth LE device has more data to send. If neither of Central and Peripheral Bluetooth LE device has set the MD bit in their packets, the packet from the Peripheral closes the connection event. If the Central and Peripheral devices have set the MD bit, the Central can continue the connection event by sending another packet, and the Peripheral must listen after sending its packet. For more information about MD, see volume 6, Part B, Section 4.5.6 of the Bluetooth Core Specification [2].
- CTEInfo present (CP): This field indicates whether the data physical channel PDU header has a CTEInfo field and, subsequently whether the data physical channel packet has a CTE. For more information about the packet structure of the CTEInfo field, see Volume 6, Part B, Section 2.5.2 of the Bluetooth Core Specification [2].
- Length: This field indicates the size, in octets, of the payload and MIC, if present. The size of this field is in the range [0, 255] octets.
- CTEInfo: This field indicates the type and length of the CTE.

The two types of data physical channel PDUs are: “LL Data PDU” on page 8-32 and “LL Control PDU” on page 8-33.

LL Data PDU

The LL uses the LL data PDU to send L2CAP data. The LLID field in the LL data channel PDU header is set to either `01b` or `10b`. An LL data PDU is referred to as an empty PDU if

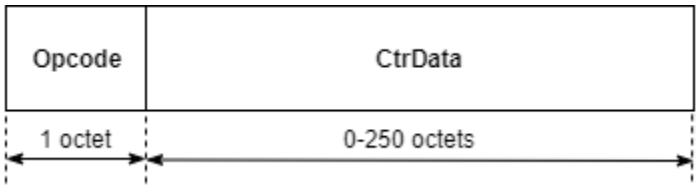
- The LLID field of the LL data channel PDU header is set to `01b`.

- The Length field of the LL data channel PDU header is set to `00000000b`.

An LL data PDU with the LLID field in the header set to `10b` does not have the Length field set to `00000000b`.

LL Control PDU

The LL uses the LL data PDU to control the LL connection. If the LLID field of data physical channel PDU header is set to `11b`, the data physical channel PDU contains an LL control PDU. This figure shows the LL control PDU payload.



The Opcode field defines different types of LL control PDUs as shown in this table.

Opcode	LL Control PDU
0x00	LL_CONNECTION_UPDATE_IND
0x01	LL_CHANNEL_MAP_IND
0x02	LL_TERMINATE_IND
0x03	LL_ENC_REQ
0x04	LL_ENC_RSP
0x05	LL_START_ENC_REQ
0x06	LL_START_ENC_RSP
0x07	LL_UNKNOWN_RSP
0x08	LL_FEATURE_REQ
0x09	LL_FEATURE_RSP
0x0A	LL_PAUSE_ENC_REQ
0x0B	LL_PAUSE_ENC_RSP
0x0C	LL_VERSION_IND
0x0D	LL_REJECT_IND
0x0E	LL_SLAVE_FEATURE_REQ
0x0F	LL_CONNECTION_PARAM_REQ
0x10	LL_CONNECTION_PARAM_RSP
0x11	LL_REJECT_EXT_IND
0x12	LL_PING_REQ
0x13	LL_PING_RSP
0x14	LL_LENGTH_REQ
0x15	LL_LENGTH_RSP
0x16	LL_PHY_REQ

Opcode	LL Control PDU
0x17	LL_PHY_RSP
0x18	LL_PHY_UPDATE_IND
0x19	LL_MIN_USED_CHANNELS_IND
0x1A	LL_CTE_REQ
0x1B	LL_CTE_RSP
0x1C	LL_PERIODIC_SYNC_IND
0x1D	LL_CLOCK_ACCURACY_REQ
0x1E	LL_CLOCK_ACCURACY_RSP
All other values	Reserved for future use

The CtrData field in the LL control PDU is specific to the value of the Opcode field. For more information about different LL control PDUs and their corresponding CtrData field structure, see Volume 6, Part B, Sections 2.4.2.1 to 2.4.2.28 of the Bluetooth Core Specification [2].

Constant Tone Extension and In-Phase Quadrature (IQ) Sampling

The length of the CTE is variable and in the range [16, 160] µs. This field contains a constantly modulated series of 1s with no whitening applied. The CTE is of two types: antenna switching during CTE transmission (AoD) and antenna switching during CTE reception (AoA). When receiving a packet containing an AoD CTE, the receiver does not need to switch antennae. When receiving a packet containing an AoA CTE, the receiver performs antenna switching according to the switching pattern configured by the host. In both cases, the receiver takes an IQ sample at each microsecond during the reference period and an IQ sample each sample slot. The controller reports the IQ samples to the host. The receiver samples the entire CTE regardless of its length, unless this conflicts with other activities. For more information about CTE, see Volume 6, Part B, Sections 2.5.1 to 2.5.3 of the Bluetooth Core Specification [2].

When requested by the host, the receiver performs IQ sampling when receiving a valid Bluetooth LE packet with a CTE. However, when receiving a Bluetooth LE packet with a CTE and an incorrect CRC, the receiver might perform IQ sampling. For more information about IQ sampling, see Volume 6, Part B, Section 2.5.4 of the Bluetooth Core Specification [2].

Note For more information about data physical channel PDUs, see Volume 6, Part B, Section 2.4 of the Bluetooth Core Specification [2].

Bluetooth BR/EDR Packet Structure

Bit Ordering in Bluetooth BR/EDR Packets

The bit ordering in Bluetooth BR/EDR packets follows the same format as the “Bit Ordering in Bluetooth LE Packets” on page 8-27.

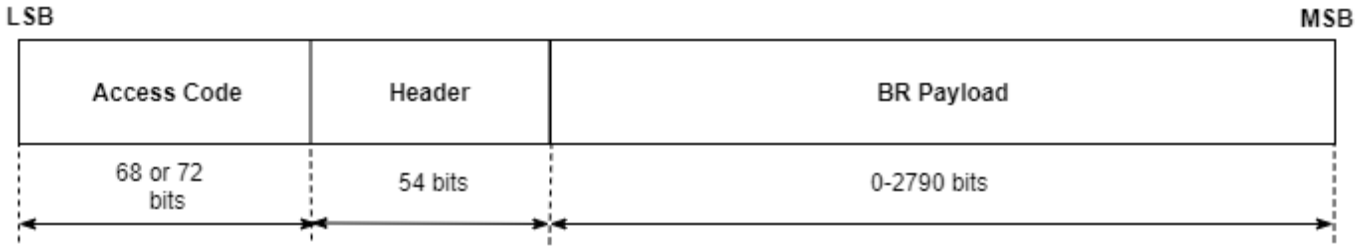
Bluetooth BR/EDR devices use packet formats for: “BR Mode” on page 8-35, “EDR Mode” on page 8-35, “Access Code” on page 8-35, “Packet Header” on page 8-37, “Packet Types” on page 8-38, and “Payload Format” on page 8-39.

Note For more information about Bluetooth BR/EDR packet structure, see Volume 2, Part B, Section 6 of the Bluetooth Core Specification [2].

General Format

BR Mode

The general format of Bluetooth BR packets is shown in this figure. Each packet consists of these fields: the access code (68 or 72 bits), header (54 bits), and payload in the range [0, 2790] bits.

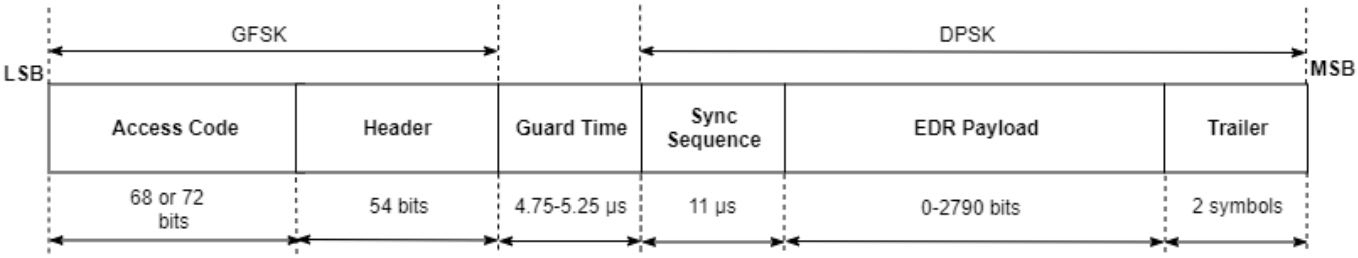


The Bluetooth Core Specification [2] defines different types of packets. A packet can consist of:

- The shortened access code only
- The access code and the packet header
- The access code, the packet header, and the payload

EDR Mode

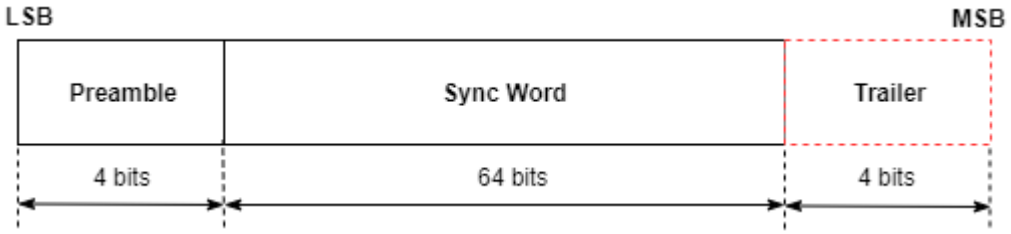
The general format of Bluetooth EDR packets is shown in this figure.



The format and modulation of the access code and the packet header fields are similar to that of BR packets. Following the header field, the EDR packets have a guard time in the range [4.75, 5.25] µs, a sync sequence (11 µs), payload in the range [0, 2790] bits, and trailer (two symbols) fields.

Access Code

Each packet starts with an access code. If a packet header follows, the access code is 72 bits long. Otherwise, the length of the access code is 68 bits. In this case, the access code is referred to as a shortened access code. The shortened access code does not contain a trailer. The access code is used for synchronization, DC offset compensation, and identification of all packets exchanged on the physical channel. The shortened access code is used in paging and inquiry. In this case, the access code itself is used as a signaling message, and neither a header nor a payload is present. This figure shows the packet structure of the access code.



Different access code types use different lower address parts (LAPs) to construct the sync word. A summary of different access code types is shown in this table.

Access Code Type	LAP	Access Code Length (Bits)	Description
Channel access code (CAC)	Central	72	This access code is used in the connection state, synchronization train substate, and synchronization scan substate. It is derived from the LAP of the Central's `BD_ADDR` .
Device access code (DAC)	Paged device	68 or 72	This access code is used during page, page scan, and page response substates. It is derived from the paged devices's `BD_ADDR`.
Dedicated inquiry access code (DIAC)	Dedicated	68 or 72	This access code is used in the inquiry substate for dedicated inquiry operations.
General inquiry access code (GIAC)	Reserved	68 or 72	This access code is used in the inquiry substate for general inquiry operations.

For DAC, DIAC, and GIAC access code types, the access code length of 72 bits is used only in combination with frequency hopping sequence (FHS) packets. When used as self-contained messages without a header, the DAC, DIAC and GIAC do not include trailer bits and are of length 68 bits.

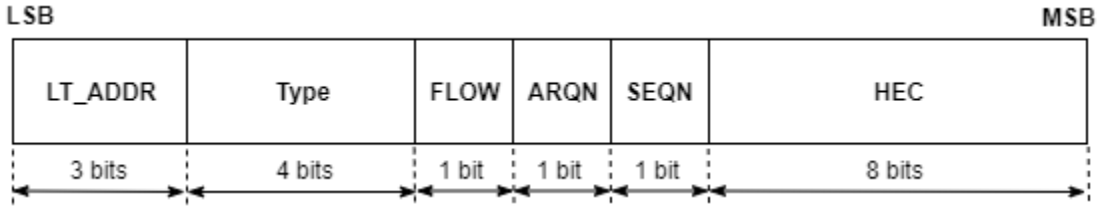
The CAC consists of a preamble, sync word, and trailer.

- Preamble: It is a fixed 4-symbol pattern of 1s and 0s that facilitates DC compensation. If the LSB of the following sync word is `1` or `0`, the preamble sequence is `1010` or `0101` (in transmission order), respectively.
- Sync word: It is a 64-bit code word derived from a 24-bit LAP address. The construction guarantees a large Hamming distance between sync words based on different LAPs. The autocorrelation properties of the sync word improve timing acquisition.
- Trailer: It is appended to the sync word as soon as the packet header follows the access code. The trailer is a fixed 4-symbol pattern of 1s and 0s. The trailer together with the three MSBs of the sync word form a 7-bit pattern of alternating 1s and 0s which is used for extended DC compensation. The trailer sequence is either `1010` or `0101` (in transmission order) depending on whether the MSB of the sync word is `0` or `1`, respectively.

Note For more information about access code in Bluetooth BR/EDR, see Volume 2, Part B, Section 6.3 of the Bluetooth Core Specification [2].

Packet Header

The structure of the Bluetooth BR/EDR packet header is shown in this figure.



This table provides a brief description about the packet header fields.

Packet Header Field	Size of the Field (Bits)	Description
Logical transport address (LT_ADDR)	3	This field indicates the destination Peripheral(s) for a packet in a Central-to-Peripheral transmission slot and indicates the source Peripheral for a Peripheral-to-Central transmission slot.
Type	4	This field specifies the type of packet used. The Bluetooth Core Specification [2] defines 16 different types of BR/EDR packets. The value in this field depends on the value of LT_ADDR field in the packet. This field determines the number of slots occupied by the current packet.
Flow control (FLOW)	1	This field implements the flow control of BR/EDR packets over the asynchronous connection-oriented logical (ACL) transport. When the receive buffer for the ACL logical transport is full, a 'STOP' indication (FLOW = 0) is returned to stop the other device from transmitting data temporarily. When the receive buffer can accept data, a 'GO' indication (FLOW = 1) is returned.
Automatic repeat request number (ARQN)	1	This field informs the source of a successful transfer of payload data with the CRC. This field is reserved for future use on the connectionless Peripheral broadcast (CSB) logical transport.
Sequence number (SEQN)	1	This field provides a sequential numbering scheme to order the data packet stream. This field is reserved for future use on the CSB logical transport.

Packet Header Field	Size of the Field (Bits)	Description
Header error check (HEC)	8	This field checks the packet header integrity. Before generating the HEC, the HEC generator is initialized with an 8-bit value. These 8 bits correspond to the upper address part (UAP). After the initialization, the HEC generator calculates the HEC value for the 10 header bits. Before checking the HEC, the receiver initializes the HEC check circuitry with the appropriate 8-bit UAP. If the HEC does not check the packet header integrity, the entire packet is discarded.

Note For more information about packet header used in Bluetooth BR/EDR, see Volume 2, Part B, Section 6.4 of the Bluetooth Core Specification [2].

Packet Types

The packets used in the piconet are related to these logical transports on which they are used.

- Synchronous connection-oriented (SCO): It is a circuit-switched connection that reserved slots between the Central and a specific Peripheral.
- Extended SCO (eSCO): Similar to SCO, it reserves slots between the Central and a specific Peripheral. eSCO supports a retransmission window following the reserved slots. Together, the reserved slots and the retransmission window form the complete eSCO window.
- ACL: It provides a packet-switched connection between the Central and all active Peripherals participating in the piconet. ACL supports asynchronous and isochronous services. Between a Central and a Peripheral, only a single ACL logical transport must exist.
- CSB: It is used to transport profile broadcast data from a Central to multiple Peripherals. A CSB logical transport is unreliable.

This table summarizes the packets defined for the SCO, eSCO, ACL, and CSB logical transport types.

Note The column entries followed by "D" means data field only. "C.1" implies that the MIC value is mandatory when encryption with AES-CCM is enabled. Otherwise, MIC is excluded. For more information about different packet types used in Bluetooth BR/EDR, see Volume 2, Part B, Sections 6.5 and 6.7 of the Bluetooth Core Specification [2].

Packet Type	TYPE Code	Slot Occupancy	Payload Header (Bytes)	User Payload (Bytes)	FEC	MIC	CRC	Logical Transport Types Supported
ID	N/A	1	N/A	N/A	N/A	N/A	N/A	N/A
NULL	0000	1	N/A	N/A	N/A	N/A	N/A	SCO, eSCO, ACL, CSB

Packet Type	TYPE Code	Slot Occupancy	Payload Header (Bytes)	User Payload (Bytes)	FEC	MIC	CRC	Logical Transport Types Supported
POLL	0001	1	N/A	N/A	N/A	N/A	N/A	SCO, eSCO, ACL
FHS	0010	1	N/A	18	2/3	N/A	Yes	SCO, ACL
DM1	0011	1	1	0-17	2/3	C.1	Yes	SCO, ACL, CSB
DH1	0100	1	1	0-27	No	C.1	Yes	ACL, CSB
DM3	1010	3	2	0-121	2/3	C.1	Yes	ACL, CSB
DH3	1011	3	2	0-183	No	C.1	Yes	ACL, CSB
DM5	1110	5	2	0-224	2/3	C.1	Yes	ACL, CSB
DH5	1111	5	2	0-339	No	C.1	Yes	ACL, CSB
2-DH1	0100	1	2	0-54	No	C.1	Yes	ACL, CSB
2-DH3	1010	3	2	0-367	No	C.1	Yes	ACL, CSB
2-DH5	1110	5	2	0-679	No	C.1	Yes	ACL, CSB
3-DH1	1000	1	2	0-83	No	C.1	Yes	ACL, CSB
3-DH3	1011	3	2	0-552	No	C.1	Yes	ACL, CSB
3-DH5	1111	5	2	0-1021	No	C.1	Yes	ACL, CSB
HV1	0101	1	N/A	10	1/3	No	No	SCO
HV2	0110	1	N/A	20	2/3	No	No	SCO
HV3	0111	1	N/A	30	No	No	No	SCO
DV	1000	1	1 D	10+(0-9) D	2/3 D	No	Yes D	SCO
EV3	0111	1	N/A	1-30	No	No	Yes	eSCO
EV4	1100	3	N/A	1-120	2/3	No	Yes	eSCO
EV5	1101	3	N/A	1-180	No	No	Yes	eSCO
2-EV3	0110	1	N/A	1-60	No	No	Yes	eSCO
2-EV5	1100	3	N/A	1-360	No	No	Yes	eSCO
3-EV3	0111	1	N/A	1-90	No	No	Yes	eSCO
3-EV5	1101	3	N/A	1-540	No	No	Yes	eSCO

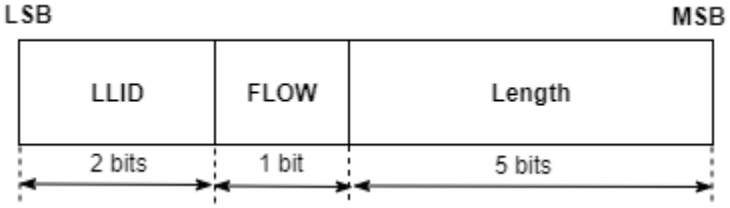
Payload Format

The Bluetooth Core Specification [2] defines two types of payload field formats: synchronous data field (for ACL packets) and asynchronous data field (for SCO and eSCO packets). However, the DV packets contain both the synchronous and asynchronous data fields.

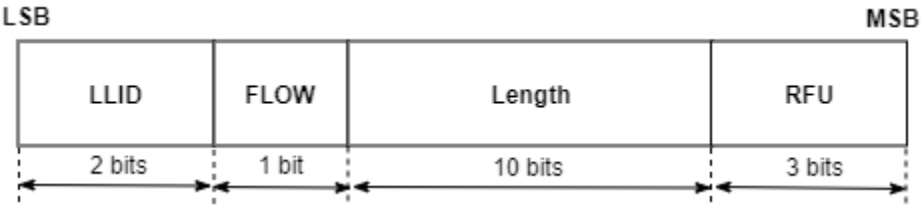
- Synchronous data field: In SCO, which supports only the BR mode, the length of the synchronous data field is fixed. The synchronous data field contains only the synchronous data body portion and does not have a payload header. In BR eSCO, the synchronous data field consists of these two segments: a synchronous data body and a CRC code. In this case, no payload header is present. In

EDR eSCO, the synchronous data field consists of a guard time, synchronization sequence, synchronous data body, CRC code, and trailer. In this case, no payload header is present.

- Asynchronous data field: The BR ACL packets have an asynchronous data field consisting of payload header, payload body, MIC (if applicable), and CRC (if applicable). This figure shows the 8-bit payload header format for BR single-slot ACL packets.



EDR ACL packets have an asynchronous data field consisting of guard time, synchronization sequence, payload header, payload body, MIC (if applicable), CRC (if applicable), and trailer. This figure shows the 16-bit payload header format for EDR multislot ACL packets.



Note For more information about the payload format, see Volume 2, Part B, Sections 6.6.1 and 6.6.2 of the Bluetooth Core Specification [2].

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

More About

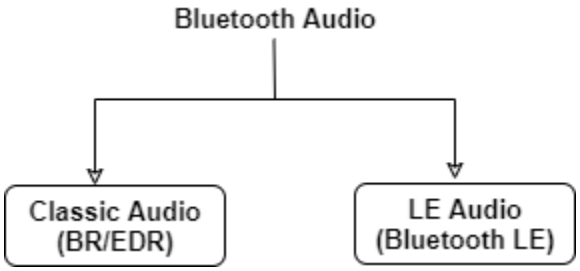
- “Bluetooth Technology Overview” on page 8-2
- “Bluetooth Protocol Stack” on page 8-9
- “Comparison of Bluetooth BR/EDR and Bluetooth LE Specifications” on page 8-6

Bluetooth LE Audio

The Bluetooth Core Specification 5.2 [2] defined by the Bluetooth Special Interest Group (SIG) introduced the next generation of Bluetooth audio called the low energy (LE) audio. LE audio operates on the Bluetooth LE standard. For more information about the Bluetooth LE stack, see “Bluetooth Protocol Stack” on page 8-9.

What Is LE Audio?

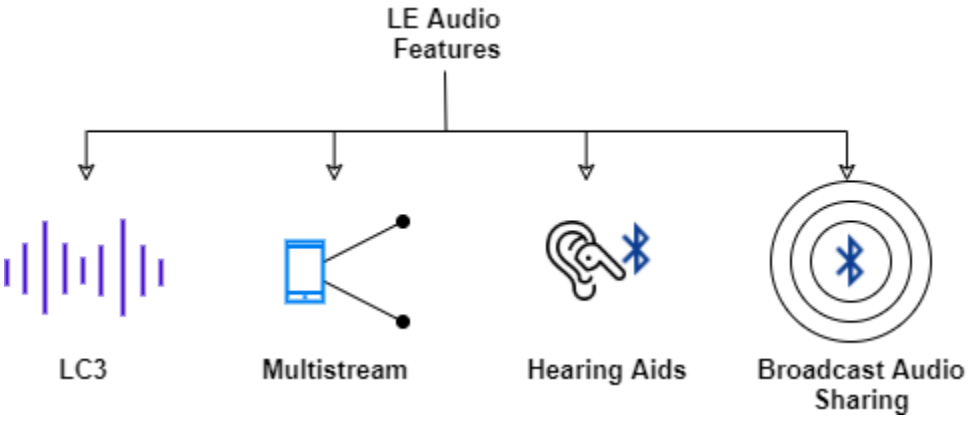
This figure shows the taxonomy of Bluetooth audio.



Bluetooth audio can be classified as – classic audio (operates on the basic rate/enhanced data rate (BR/EDR) physical layer (PHY)) and LE audio (operates on the Bluetooth LE PHY). LE audio is the next generation of Bluetooth audio, which supports development of the same audio products and use cases as the classic audio. It also enables creation of new products and use cases and presents additional features and capabilities to help improve the performance of classic audio products. Some of the key features and use cases of LE audio include enabling audio sharing, providing multistream audio, and supporting hearing aids. For more information about LE audio features and use cases, see “Features of LE Audio” on page 8-41 and “Use Cases of LE Audio” on page 8-51, respectively.

Features of LE Audio

This figure illustrates the salient features of LE audio.



Low Complexity Communication Codec (LC3)

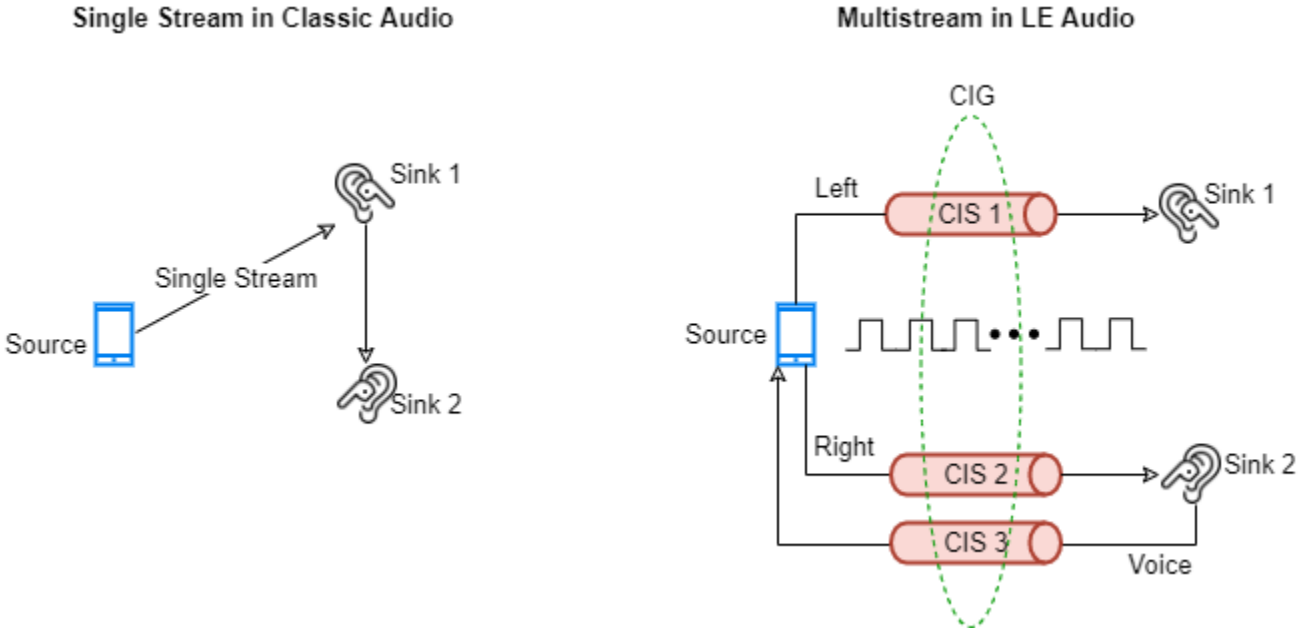
LE audio includes a new high quality, low power codec, known as LC3. It supports a wide range of sample rates, bit rates, and frame rates giving the product developers maximum flexibility to optimize

their products to deliver the best possible audio experience to end users. As compared to the subband codec (SBC) implemented by classic audio, LC3 is much more efficient in processing and delivering audio. A comparison between LC3 and SBC related to the standard stereo listening test [1] verifies that LC3 delivers high quality audio at low data rates. The results shown in [1] show that even at half of the bit rate, LC3 provides far superior audio experience than SBC.

The intrinsic shortcomings of SBC resulted in the manufacturers of audio equipment such as Bluetooth headphones turning to proprietary solutions such as audio codec 3 (AC3) and AptX. Such proprietary solutions need specific hardware support and add costs over standards-based implementations. The introduction of LC3 removes the dependency on the proprietary solutions, resulting in lower device costs. LC3 enables the product developers to have an efficient tradeoff between sound quality and power consumption. The high quality, low power characteristic of LC3 enables the product developers to optimize the longevity of the device battery.

Multistream Audio

Multistream audio enables you to transmit multiple, independent, and synchronized audio streams between an audio source device, such as a smartphone, and one or more audio sink devices like earbuds or earphones. To support multistream audio, the Bluetooth Core Specification 5.2 [2] introduced the connected isochronous stream (CIS) and connected isochronous group (CIG). For more information about the CIS and CIG, see “CIS and CIG” on page 8-45. This figure shows how LE audio enables you to send multiple audio streams between a source and sink.



Classic Bluetooth audio supports only a single point-to-point audio stream over the advanced audio distribution profile (A2DP). However, LE audio enables you to handle multiple isochronous audio streams with synchronization between them. The multistream support of LE audio can improve the performance of truly wireless earphones by providing a better stereo imaging experience, making the use of voice assistant services more seamless, and making switching between multiple audio source devices smoother [1].

Hearing Aids Support

LE audio provides exclusive support for hearing aids. Typically, hearing aid devices require low and efficient power consumption. LE audio supports high quality, low power capability of LC3 and the efficient power consumption characteristic of the Bluetooth LE standard. LE audio-supported hearing aids are interoperable, enabling you to connect to most smartphones, TVs, and laptops and making these devices much more accessible to people with hearing loss.

Broadcast Audio Sharing

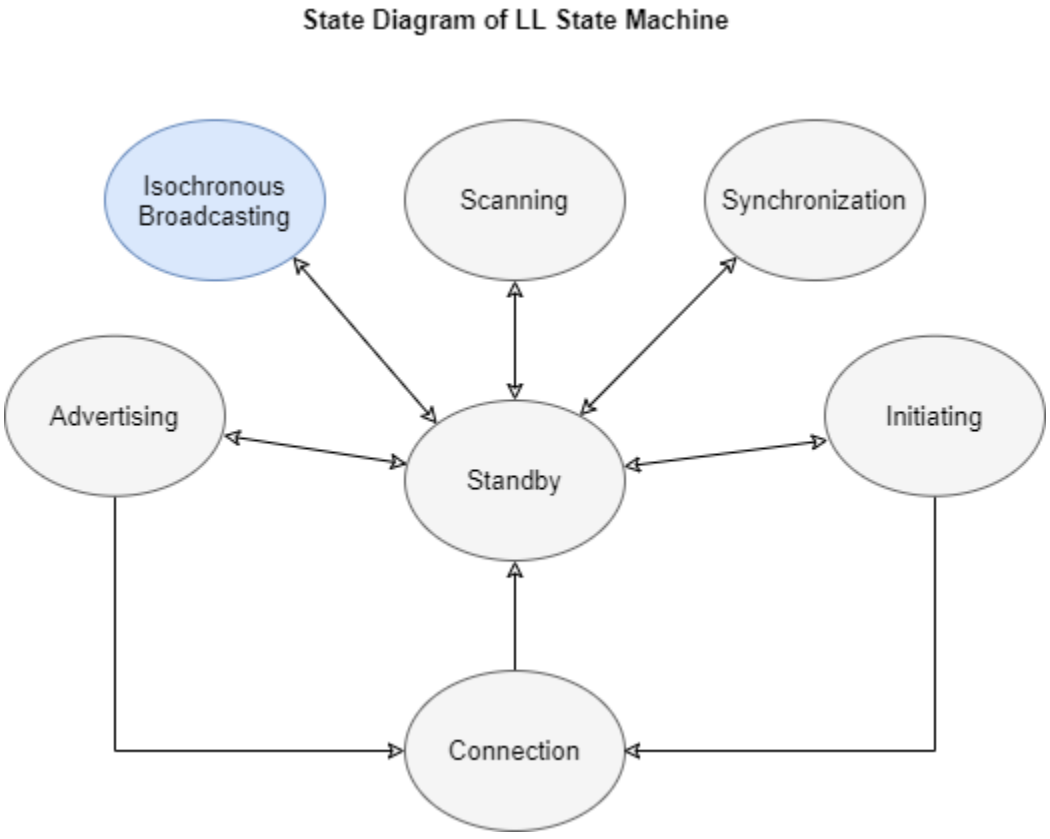
LE audio now supports the ability to broadcast one or more audio streams to an unlimited number of audio sink devices. Broadcast audio opens significant new opportunities for innovation, such as a new Bluetooth use case, audio sharing. Broadcast audio sharing can be personal (share audio with people around you) or location-based (share audio in public places like airports).

Support For LE Audio In Bluetooth Core Specification

The Bluetooth Core Specification 5.2 [2] introduced these updates pertaining to LE audio.

Changes to Link Layer (LL) State Machine

The functioning of the LL is described in terms of a state machine. This figure shows the state diagram of the LL state machine.



The Bluetooth Core Specification 5.2 [2] added a new state, Isochronous Broadcasting, to the LL state machine. In the Isochronous Broadcasting state, the LL transmits the isochronous data packets on an

isochronous physical channel. The Isochronous Broadcasting state can be entered from the Standby state. If a device is in the Isochronous Broadcasting state, then it is referred to as an isochronous broadcaster.

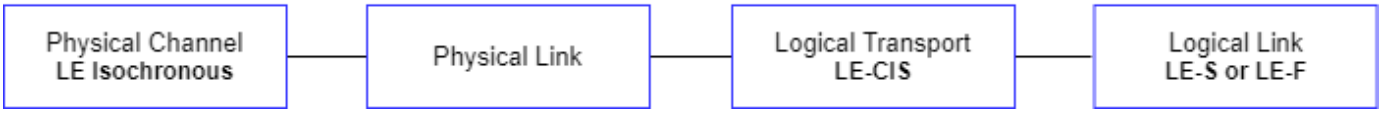
Note For more information about different states of the LL state machine, see Volume 6, Part B, Section 1.1 of the Bluetooth Core Specification 5.2 [2].

LE Isochronous Channels and the Bluetooth Data Transport Architecture

The LE isochronous channels feature enables you to transfer latency-sensitive data between the devices. This feature provides a mechanism to ensure synchronization between multiple sink devices receiving data from the same source. The expired data (data that violates the time-bound validity period) that is not transmitted, is discarded. Consequently, the receiving devices receive data that is valid with respect to its age and acceptable latency.

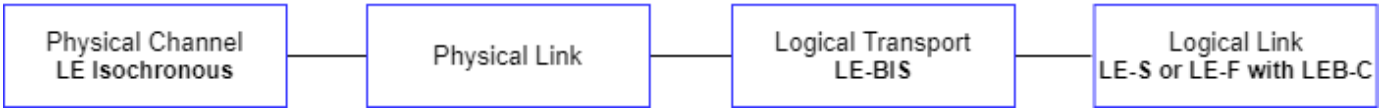
The Bluetooth data transport architecture now supports LE isochronous channels. The LE isochronous channels can be connection-oriented or connectionless. In both cases, the isochronous communication is realized using the new LE isochronous physical channel. This physical channel uses frequency hopping and specifies the timing of the first packet. This timing acts as a reference point for the timing of the subsequent packets. The LE isochronous physical channel can operate on an LE Uncoded (LE1M and LE2M) or LE Coded Bluetooth LE PHY. The LE isochronous physical channel uses LE-Stream (LE-S) and LE-Frame (LE-F) logical links to transmit audio data and framed data packets, respectively. The connection-oriented isochronous channels use LE-CIS logical transport and support bidirectional communication. This figure shows the procedure of connection-oriented isochronous channel data transport.

Connection-oriented Isochronous Channel Data Transport



A single LE-CIS stream provides point-to-point isochronous communication between two connected devices. A flushing period is specified for the LE-CIS logical transport. Any packet that has not been transmitted within the flushing period is discarded. This figure shows the procedure of connectionless isochronous channel data transport.

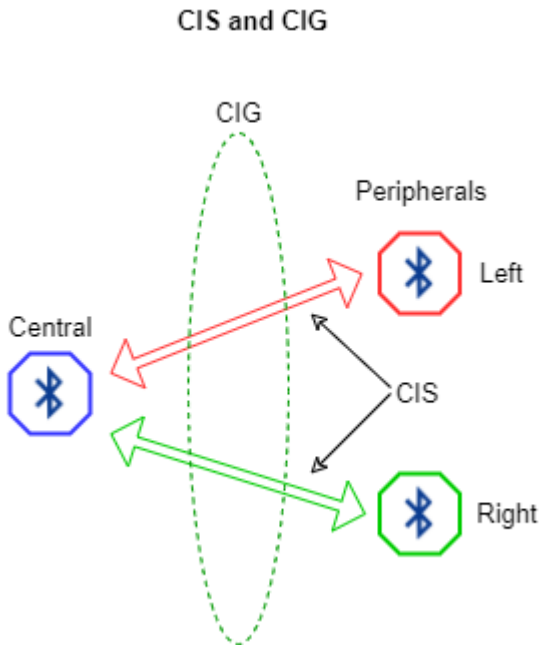
Connectionless Isochronous Channel Data Transport



Connectionless isochronous communication uses broadcast isochronous streams (BIS) and supports only unidirectional communication. The BIS uses LE-S or LE-F logical links over the LE isochronous physical channel for user data, with the new LE-broadcast control (LEB-C) logical link used for control requirements. A single BIS can stream identical copies of data to multiple receiver devices.

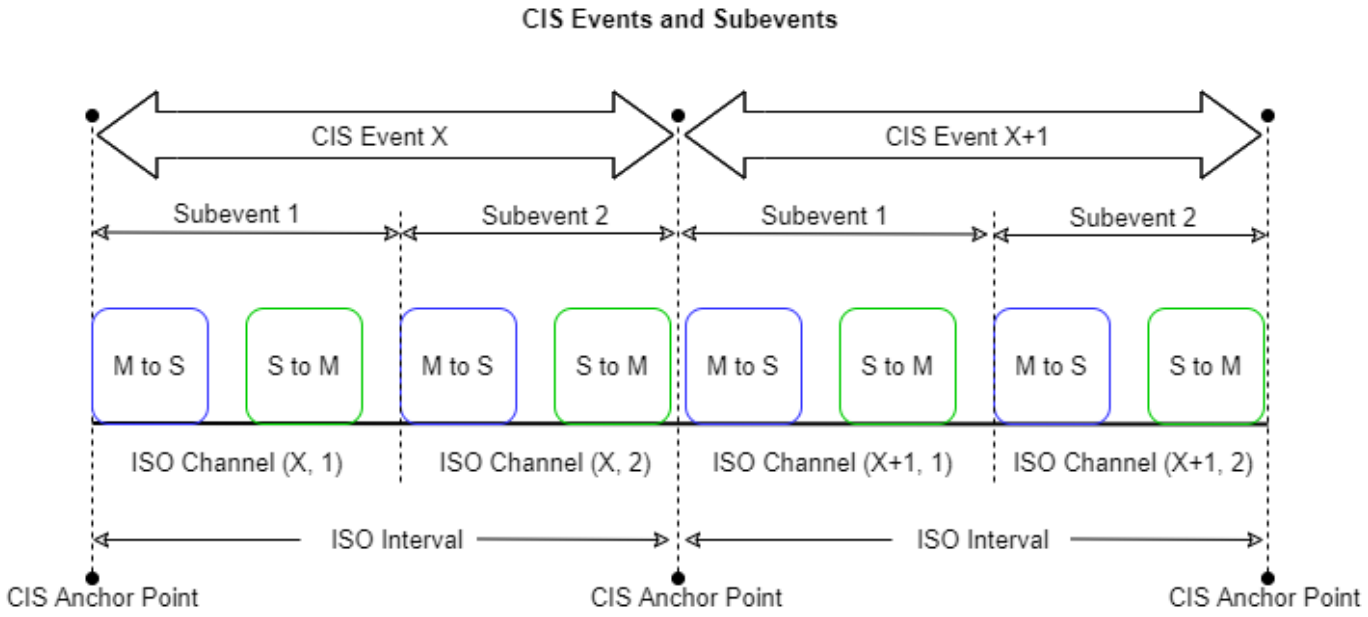
CIS and CIG

A CIS is a logical transport that enables connected devices to transfer isochronous data unidirectionally and bidirectionally. The isochronous data can be transferred either in an LE-S or LE-F logical link by using the CIS logical transport. Each CIS is associated with an LE asynchronous connection (ACL). A CIS supports variable-size packets and the transmission of one or more packets in each isochronous event. This capability enables LE audio to support a range of data rates. A CIG consists of two or more CISs that have the same ISO interval (time between the anchor points of adjacent CISs) and that are expected to have a time relationship at the application layer, or of a single CIS. This figure shows the CIS and CIG servicing left and right stereo ear buds.



The maximum number of CISs in a CIG is 31. A CIG comes into existence when its first CIS is created, and it ceases to exist when all of its constituent CISs are destroyed.

This figure illustrates the concept of CIS events and subevents.



Each CIS event occurs at a regular ISO Interval, which is in the range from 5 ms to 4 s in multiples of 1.25 ms. Each CIS event is partitioned into one or more subevents. In a CIS, during a subevent, the Central transmits once and the Peripheral responds as shown in the preceding figure. A CIS event is an opportunity for the Central and Peripheral to exchange CIS protocol data units (PDUs). A subevent ends at the end of the Peripheral's packet, if any, and at the end of the Central's packet. The isochronous channel is changed at the end of each subevent. The LL closes a CIS event at the end of its last subevent.

All CISes in a CIG has the same Central but may have different Peripheral's. A CIG event consists of the corresponding CIS events of the CISes currently making up that CIG. Each CIG event starts at the anchor point of the earliest (in transmission order) CIS of the CIG and ends at the end of the last subevent of the latest CIS of the same CIG event. Any two CIG events on the same CIG do not overlap because the last CIS event of a given CIG event ends before the first CIS anchor point of the next CIG event.

Consider a use case where an audio stream from a smartphone (the source) is to be played in the left and right buds (the two sinks) of LE earphones. The left and right buds each establish a CIS stream with the source device. Both the CIS streams are part of the same CIG. A fragment of audio produced by the source is encoded into a packet and a copy is transmitted to each sink device over its stream, one at a time during a series of consecutive CIS events. The audio playback must not start until all devices in the CIG have received the packet.

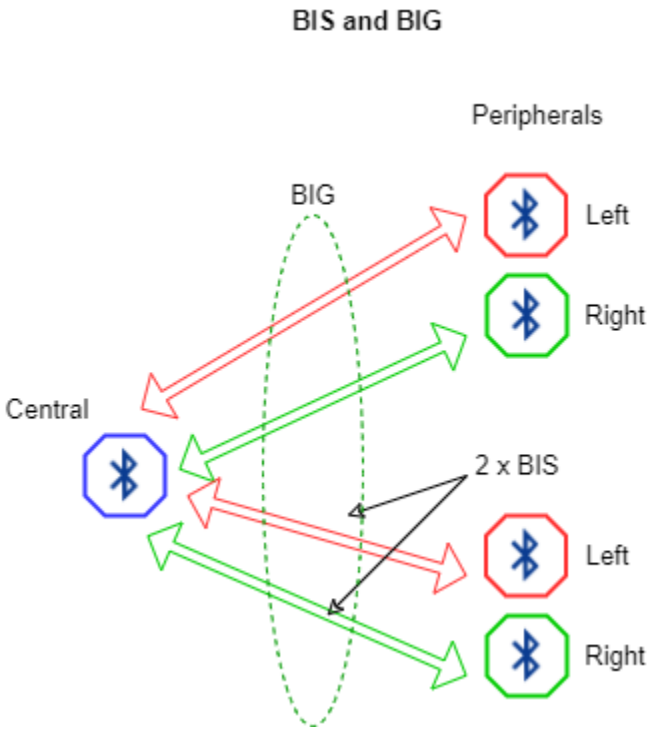
Note

- For more information about CIS, see Volume 6, Part B, Section 4.5.13 of the Bluetooth Core Specification 5.2 [2].
- For more information about CIG, see Volume 6, Part B, Section 4.5.14 of the Bluetooth Core Specification 5.2 [2].

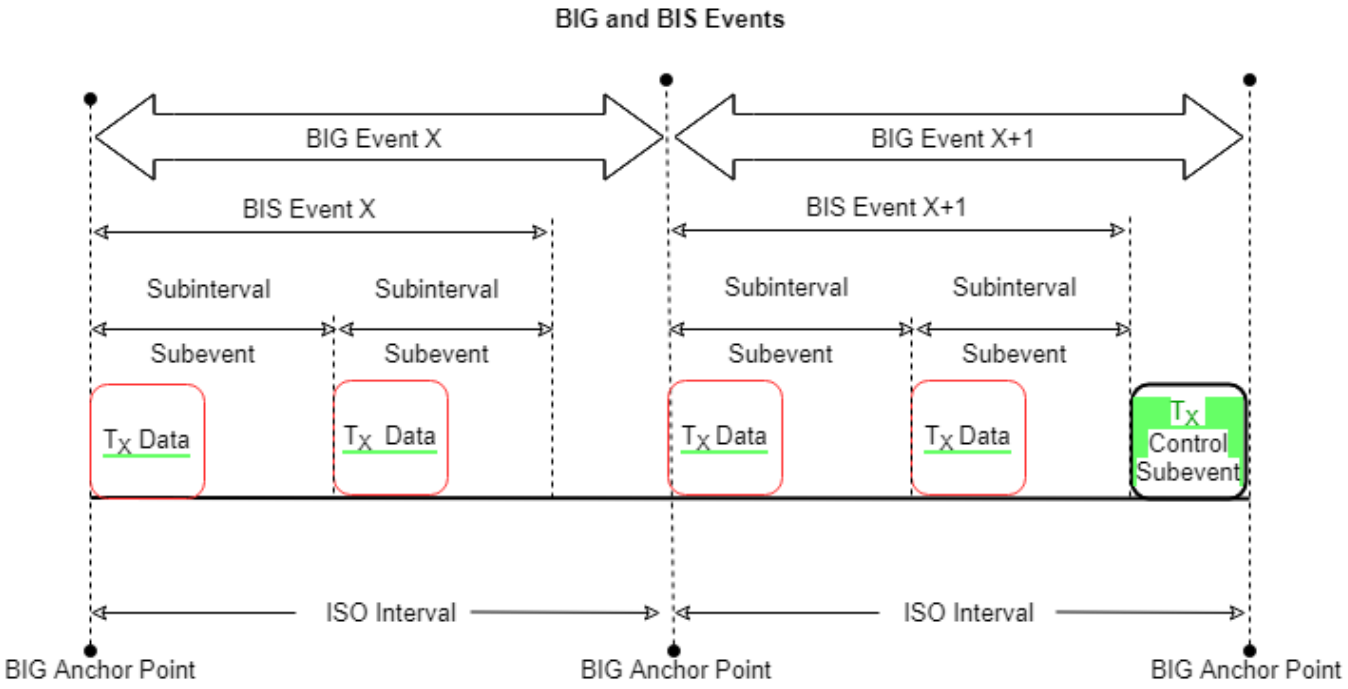
BIS and Broadcast Isochronous Group (BIG)

A BIS is a logical transport that enables a device to transfer isochronous data (framed or unframed). A BIS supports variable-size packets and the transmission of one or more packets in each isochronous event, enabling LE audio to support a range of data rates. The data traffic is unidirectional from the broadcasting device. Therefore, no acknowledgment protocol exists, making broadcast isochronous traffic unreliable. To improve the reliability of packet delivery, the BIS supports multiple retransmissions.

A BIG contains two or more BISs that have the same ISO interval and that are expected to have a time relationship at the application layer, or of a single BIS. The maximum number of BISs in a BIG is 31. This figure shows the BIS and BIG servicing a pair of left and right stereo ear buds.



For each BIS within a BIG, a schedule of transmission time slots (known as events and subevents) exist. This figure shows the concept of BIS and BIG events and subevents.



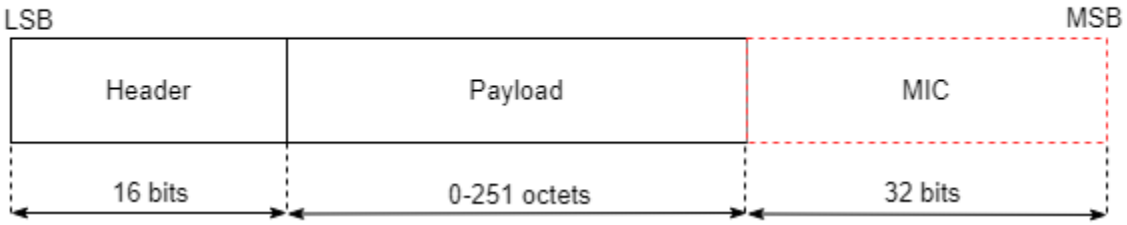
Each BIS event starts at the BIS anchor point and ends after its last subevent. Each BIG event starts at the BIG anchor point and ends after the control subevent, if one exists. If a control subevent does not exist, the BIG event ends at the end of the last BIS event. A BIS subevent enables an isochronous broadcaster to transmit a BIS PDU and enables a synchronized receiver to receive it. The LL must transmit one BIS data PDU at the start of each subevent of the isochronous broadcasting event. For each BIS event, the source of the data must send a burst of data consisting of burst number (BN) payloads. The subevents of each BIS event are each partitioned into groups of BN subevents. Each BIG event contains an optional control subevent. If a control subevent is present, the LL transmits a single BIG control PDU at the start of the control subevent to send control information about the BIG. The LL does not transmit a BIG Control PDU at any other time.

Note For more information about BIG and BIS, see Volume 6, Part B, Section 4.4.6 of the Bluetooth Core Specification 5.2 [2].

Isochronous Physical Channel Protocol Data Unit (PDU)

The isochronous physical channel PDU contains a 16-bit header, a variable size payload, and an optional message integrity check (MIC) field. This figure shows the packet structure of isochronous physical channel PDU.

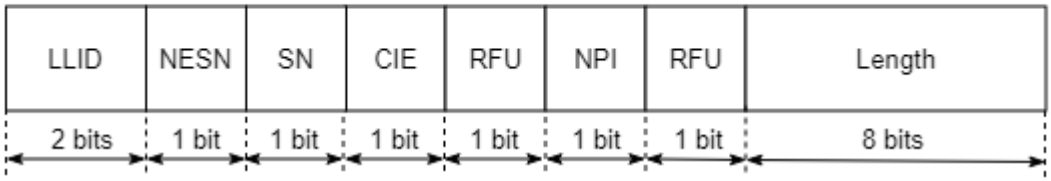
Isochronous Physical Channel PDU



The format of the Header and Payload fields depend on the type of isochronous physical channel PDU that is being used. The isochronous physical channel PDU is a CIS PDU or a BIS PDU when used on a CIS or BIS, respectively. The MIC field is included in all PDUs that contain a nonzero Payload transmitted on an encrypted CIS or BIS. If a PDU is sent on an nonencrypted CIS or BIS or has a zero-length Payload, then the MIC field is not present.

This figure shows the packet structure of an isochronous physical channel PDU Header for a CIS PDU.

Isochronous Physical Channel PDU Header for CIS PDU



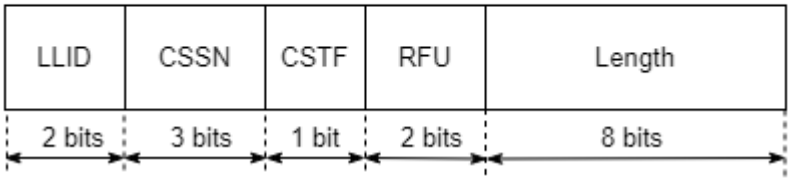
A CIS PDU can be a CIS data PDU or a CIS null PDU. A CIS Data PDU carries isochronous data, whereas a CIS null PDU is used when no data exists to be sent. This table explains the contents of the Header field.

Header Field Name	Description
LL identifier (LLID)	The LLID field indicates the type of content of the Payload field of the CIS Data PDU. These are the valid values of this field. • 0b00 – Unframed CIS data PDU (end fragment of an service data unit (SDU) or a complete SDU) • 0b01 – Unframed CIS data PDU (start or continuation fragment of an SDU) • 0b10 – Framed CIS data PDU (one or more segments of an SDU) • 0b11 – Reserved for future use For a CIS null PDU, the LLID is reserved for future use (RFU).
Next expected sequence number (NESN)	The LL uses this field to either acknowledge the last PDU sent by the peer device, or to request the peer device to resend the last PDU sent.

Header Field Name	Description
Sequence number (SN)	This field sets the identification number for LL packets. For a CIS null PDU, the SN is RFU.
Close isochronous event (CIE)	The device uses this field to close a CIS event early.
Null PDU indicator (NPI)	This field indicates whether the CIS PDU is a CIS data PDU or a CIS null PDU. If the CIS PDU is a CIS null PDU, then LL sets this field.
Length	This field indicates the size (in octets) of the Payload and MIC, if included.

This figure shows the packet structure of an isochronous physical channel PDU Header for a BIS PDU.

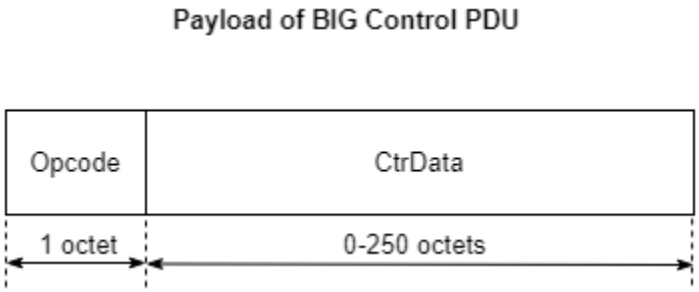
Isochronous Physical Channel PDU Header for BIS PDU



A BIS PDU can be a BIS data PDU or a BIG control PDU. A BIS data PDU carries isochronous data. A BIG control PDU sends control information for a BIG. This table explains the contents of the Header field.

Header Field Name	Description
LLID	The LLID field indicates the type of content of the Payload field of the BIS data PDU. These are the valid values of this field. • 0b00 – Unframed BIS data PDU (end fragment of an SDU or a complete SDU) • 0b01 – Unframed BIS data PDU (start or continuation fragment of an SDU) • 0b10 – Framed BIS data PDU (one or more segments of an SDU) • 0b11 – BIG control PDU
Control subevent sequence number (CSSN)	The LL uses this field to indicate the start of a BIG event that contains the first transmission of a new BIG control PDU.
Control subevent transmission flag (CSTF)	The LL uses this field to indicate whether it has scheduled a BIG control PDU to be transmitted in a BIG event.
Length	This field indicates the size (in octets) of the Payload and MIC, if included.

This figure shows the packet structure of the Payload field in a BIG control PDU.



The Opcode field specifies different types of BIG control PDUs. The Opcode field specifies the CtrData field in the Payload of BIG control PDU. For a given Opcode, the length of the CtrData field is fixed.

Note

- For more information about CIS PDU, see Volume 6, Part B, Section 2.6.1 of the Bluetooth Core Specification 5.2 [2].
- For more information about BIS PDU, see Section 2.6.2 of the Bluetooth Core Specification 5.2 [2].
- For more information about BIG control PDU, see Volume 6, Part B, Section 2.6.3 of the Bluetooth Core Specification 5.2 [2].

Isochronous Adaptation Layer (ISOAL)

To support LE audio, the Bluetooth Core Specification 5.2 [2] introduced the ISOAL in the Bluetooth stack that is present in the controller above the LL. The ISOAL enables the lower and upper layers of the stack to work together. This flexibility enables the size of isochronous data packets created and used by the upper layers to be distinct from the size used by the CIS or BIS logical transport in the LL. The ISOAL provides segmentation, fragmentation, reassembly, and recombination services for the conversion of the SDUs from the upper layer to the PDUs of the baseband resource manager and vice versa. The ISOAL enables the upper layer to use timing intervals that differ from those used by the LL so that the rate of SDUs exchanged with the upper layers is not the same as the rate with which they are exchanged with the LL. The isochronous communication mechanism uses the host controller interface (HCI) as the interface from the upper layer to the ISOAL. The SDUs are transferred to and from the upper layer by either using HCI ISO data packets or over an implementation-specific transport.

Note For more information about ISOAL, see Volume 6, Part G of the Bluetooth Core Specification 5.2 [2].

Use Cases of LE Audio

This table shows some prominent use cases of LE audio.

Use Case	Description
Personal audio sharing	With personal audio sharing, people can share their Bluetooth audio experience with others around them. For example, group of friends can simultaneously enjoy music playing on one smartphone through their LE supported headphones. This is an example of a private group of audio sink devices sharing a single audio source.
Public assisted hearing	The dialogue of a theater play can be broadcast such that all LE hearing aid users in the audience can hear the dialogue.
Public television	At the gymnasium, all attendees with LE headphones or ear buds can listen to the television audio stream.
Multi-language flight announcements	Passengers at the airport or in an aircraft can connect their LE headphones to the flight information system, specify their preferred language, and listen to the flight information in that language.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 14, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.2. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification Version 5.2 Feature Overview. https://www.bluetooth.com/.

[4] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions
`configureBIG`

Objects
`bluetoothLEBIGConfig` | `bluetoothLENode`

More About

- “Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network” on page 9-41
- “Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario” on page 9-53
- “Estimate Packet Delivery Ratio of LE Broadcast Audio in Residential Scenario” on page 6-70

Bluetooth-WLAN Coexistence

Due to the ubiquitous deployment of wireless networks and devices on the unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) frequency band, multiple homogenous and heterogeneous networks (Bluetooth, Wi-Fi, and ZigBee®) operating in this band are likely to coexist in a physical scenario. The wireless personal area network (WPAN) represented by the Bluetooth [1] and wireless local area network (WLAN) represented by IEEE® 802.11 standard both operate in the 2.4 GHz ISM frequency band. Bluetooth and WLAN radios often operate in the same physical scenario and some times in the same device. In these cases, Bluetooth and WLAN transmissions can interfere with each other, impacting the performance and reliability of both networks.

IEEE 802.15.2 Task Group [3] considers proposals for mechanisms to improve the level of coexistence between Bluetooth and WLAN devices and publishes the recommended practices derived from these.

Bluetooth and IEEE 802.11 WLAN Specifications

Bluetooth technology uses low-power radio frequency to enable short-range communication. Bluetooth is equated with the implementation specified by the Bluetooth Core Specification [2] group of standards maintained by the Bluetooth Special Interest Group (SIG) industry consortium. The Bluetooth Toolbox enables you to model Bluetooth low energy (LE), Bluetooth LE mesh, and Bluetooth basic rate/enhanced data rate (BR/EDR) communications system links, as specified in the Bluetooth Core Specification. Bluetooth BR/EDR and Bluetooth LE devices operate in the unlicensed 2.4 GHz ISM frequency band.

The Bluetooth BR mode is mandatory, whereas EDR mode is optional. The Bluetooth BR/EDR radio implements a 1600 hops/sec frequency-hopping spread spectrum (“FHSS” on page 8-54) technique. The radio hops in a pseudo-random way on 79 designated Bluetooth channels. Each Bluetooth channel has a bandwidth of 1 MHz. Each frequency is located at (2402 + k) MHz, where k = 0,1,..., 78. The modulation technique for BR and EDR mode is Gaussian frequency shift-keying (GFSK) and differential phase shift-keying (DPSK), respectively. The baud rate is 1 Msymbols/s. The Bluetooth BR/EDR radio uses the time division duplex (TDD) topology in which data transmission occurs in one direction at one time. The transmission alternates in two directions, one after the other.

In Bluetooth LE, the operating band is divided into 40 channels, k = 0, 1, ..., 39, with a channel bandwidth of 2 MHz. The range of RF center frequencies is [2402, 2480] MHz. The user data packets are transmitted using channels in the range [0, 36]. The advertising data packets are transmitted on channels 37, 38, and 39. Bluetooth LE also implements GFSK modulation. The Bluetooth LE physical layer (PHY) uses FHSS to reduce interference and to counter the impact of fading channels. The time between frequency hops can vary from 7.5 ms to 4 s and is set at the connection establishment for each Peripheral with the Central. The Central device provides the synchronization reference. The Peripheral device synchronizes to the clock and frequency-hopping pattern of the Central device. The support for the data rate at 1 Mbps is mandatory for specification version 4.x compliant devices. At a data rate of 1 Mbps, the data transmission is uncoded. Optionally, devices compliant with the Bluetooth Core Specification 5.x support these additional data rates:

- Coded transmission at bit rates of 500 kbps or 125 kbps
- Uncoded transmission at a bit rate of 2 Mbps

To explore the Bluetooth BR/EDR and Bluetooth LE protocol stack, see “Bluetooth Protocol Stack” on page 8-9. For information about different packet structures implemented in Bluetooth BR/EDR and Bluetooth LE transmissions, see “Bluetooth Packet Structure” on page 8-27. To study the

fundamentals of Bluetooth mesh networking and its applications, see “Bluetooth Mesh Networking” on page 8-64.

The IEEE 802.11 (Wi-Fi) standard is a wireless technology that connects devices and an infrastructure in a WLAN. WLAN is compliant with various IEEE 802.11 standards. Some of the prominent and widely implemented standards are 802.11 a/b/g/n/ac/ax. The 802.11a standard uses the 5 GHz unlicensed national information infrastructure (U-NII) band and provides at least 23 nonoverlapping 20 MHz wide channels instead of three nonoverlapping 20 MHz-wide channels offered by the 2.4 GHz band. The 802.11ac standard also operates in only the 5 GHz frequency band. As per Part 15 of the U.S. Federal Communications Commission (FCC) Rules and Regulations, 802.11b, 802.11g, and 802.11n standards use the 2.4 GHz. Devices that use these standards suffer interference in the 2.4 GHz band from Bluetooth devices. To mitigate this interference, devices that use 802.11b, 802.11g, or 802.11n standards implement direct-sequence spread spectrum (“DSSS” on page 8-55), “Orthogonal Frequency-Division Multiplexing” on page 8-56 (OFDM), and multiple-input, multiple-output (MIMO) OFDM signaling techniques, respectively. Devices that use the 802.11n or 802.11ax (Wi-Fi-6) standard operate in dual-band at 2.4 GHz and 5 GHz. The 802.11ax standard enhances the existing 802.11 a/b/g/n/ac standards even if they are not fully upgraded to 802.11ax. The OFDM-based channel access technique of 802.11ax standard is completely backward-compatible with traditional enhanced distributed channel access/carrier-sense multiple access (EDCA/CSMA). IEEE 802.11ax provides maximum compatibility, coexisting efficiently with 802.11a/n/ac devices.

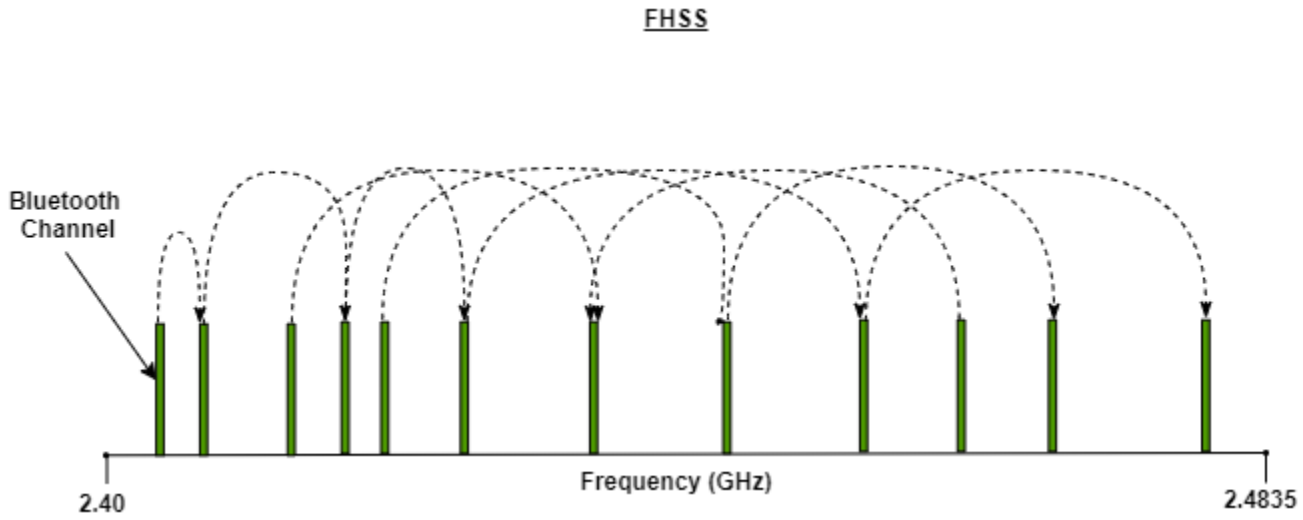
For more information about WLAN radio frequency channels, see “WLAN Radio Frequency Channels” (WLAN Toolbox). For more information about WLAN packet structures, see “WLAN PPDU Structure” (WLAN Toolbox) and “Packet Size and Duration Dependencies” (WLAN Toolbox).

Spread Spectrum Techniques

Bluetooth and WLAN technologies operate using the spread spectrum signal structuring. This signal structuring technique enables a narrowband signal such as a stream of 1s and 0s, to spread across a given frequency spectrum and transform into a wideband signal. Bluetooth devices implement the basic FHSS technique defined in the Bluetooth Core Specification [2]. This basic frequency-hopping technique is modified into an adaptive frequency hopping (AFH) technique to mitigate interference. WLAN devices use the DSSS technique.

FHSS

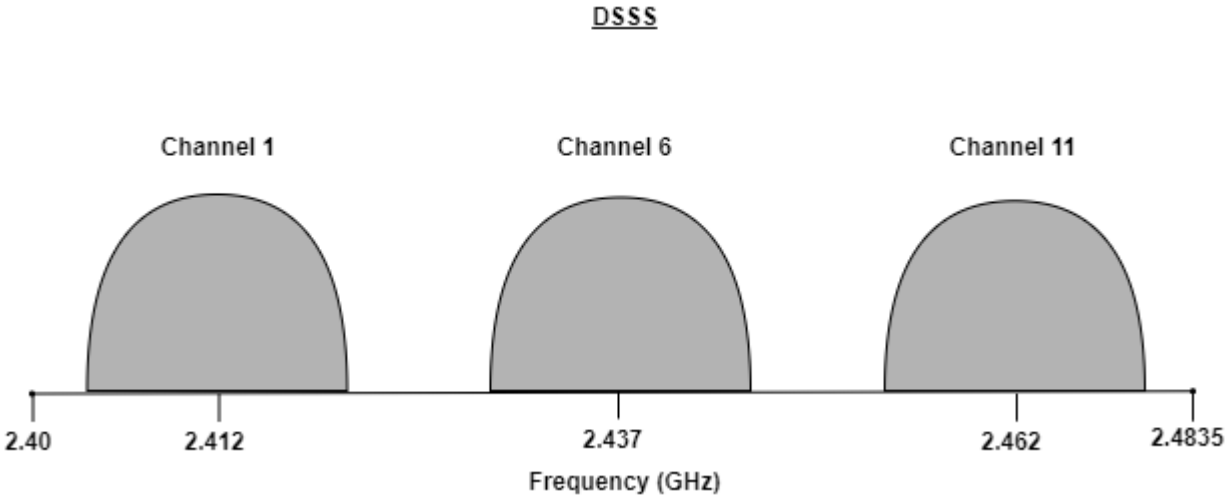
The basic Bluetooth frequency-hopping technique or the FHSS spreads the narrowband signal by *hopping* across different channels on the 2.4 GHz frequency spectrum. This figure shows how the FHSS transmits a Bluetooth signal on different frequencies at specific intervals to spread the signal across a relatively wide operating band.



The transmitting and receiving Bluetooth devices adhere to a specific hopping sequence during a particular session so that the receiving device can anticipate the frequency of the next transmission. In this case, Bluetooth makes full use of the 2.4 GHz frequency spectrum.

DSSS

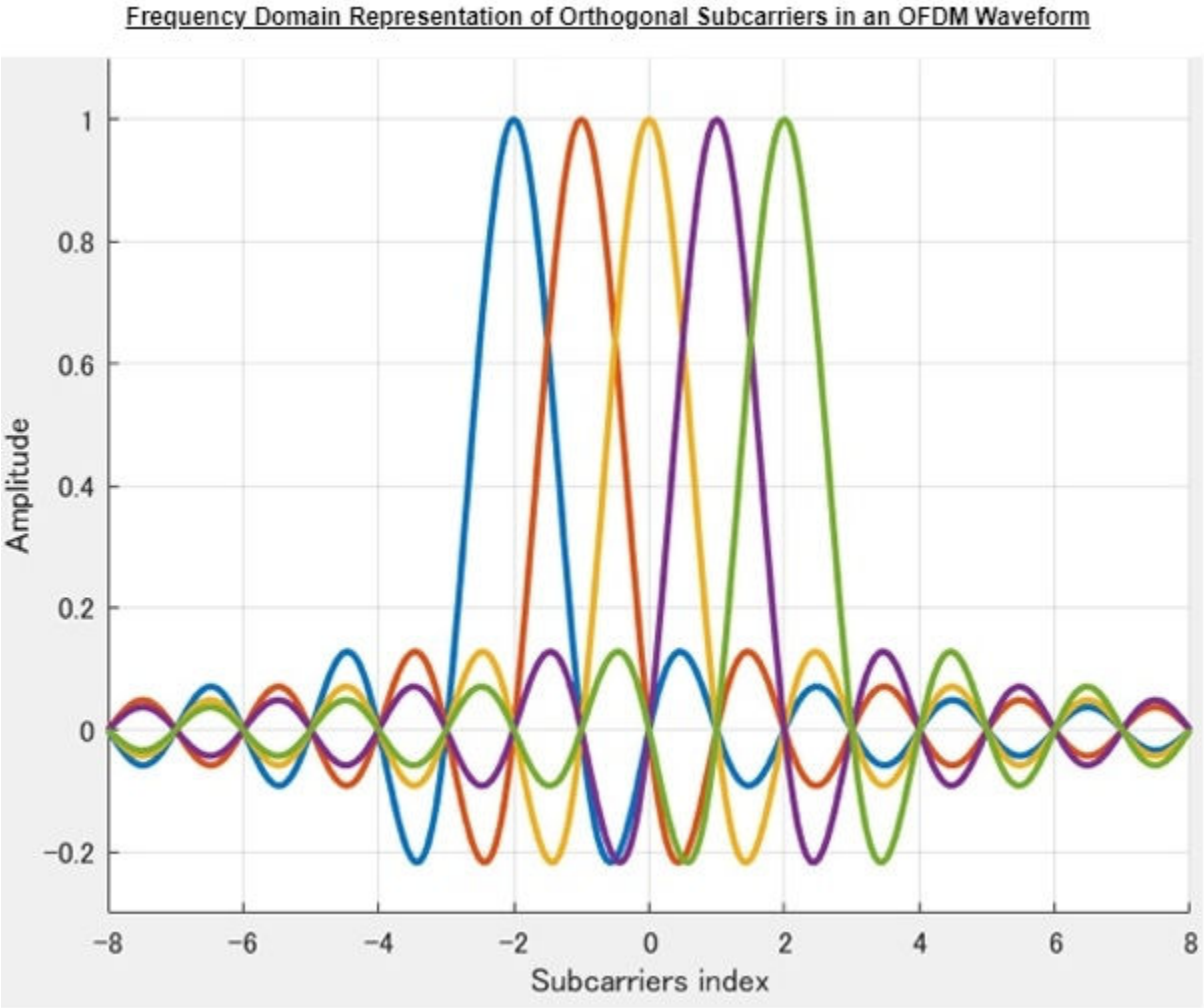
With the DSSS, the narrowband data signal is divided and simultaneously transmitted on multiple frequencies within a specific frequency band. This figure shows how the DSSS continually transmits the data signal across different channels.



The DSSS adds redundant data bits known as *chips*, to the data signal to denote 1s and 0s. The ratio of chips to data is called the *spreading ratio*. Increasing the ratio increases the immunity of the WLAN signal to interference. This is because if part of the transmission is corrupted, the data can still be recovered from the remaining part of the chipping code. The DSSS technique provides greater transmission rates than the FHSS. The DSSS also protects against data loss through redundant simultaneous data transmission. However, because DSSS floods the channel with redundant transmissions, it is more vulnerable to interference from Bluetooth devices operating on the same frequency band.

Orthogonal Frequency-Division Multiplexing

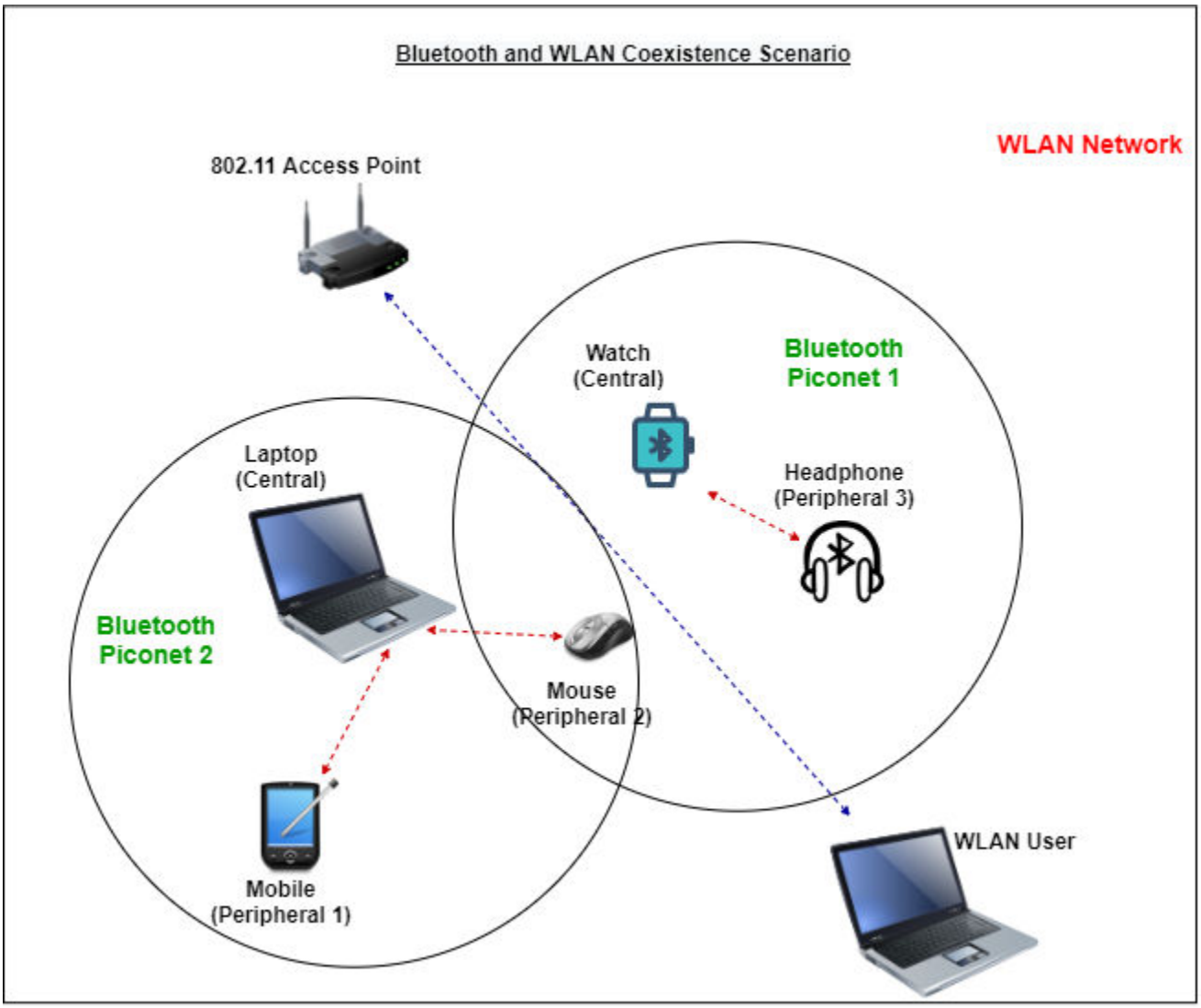
OFDM is a flexible, multicarrier modulation technique implemented by IEEE standards 802.11g/n/ac/ax. OFDM partitions the channel bandwidth into multiple narrow-band orthogonal subcarriers to carry the information. This partitioning enables the removal of guard bands. However, because the orthogonal subcarriers are unrelated, they can overlap each other. Therefore, OFDM is bandwidth efficient. This figure shows the frequency domain representation of the orthogonal subcarriers in an OFDM waveform.



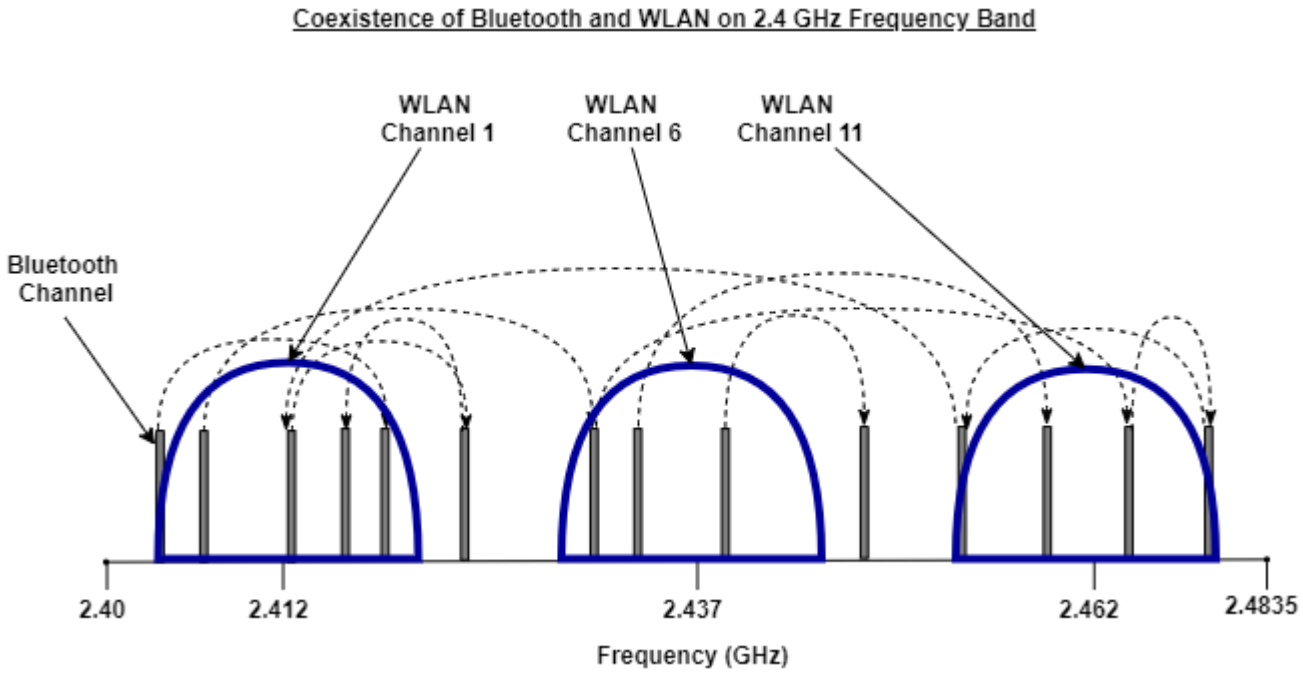
The use of narrow-band subchannels (compared to a single wideband channel) helps mitigate channel fading. As each subchannel operates at a low data rate, OFDM is very resilient to intersymbol interference and interframe interference. As data is transmitted simultaneously on multiple orthogonal subcarriers, OFDM can provide very high throughput. To further maximize the throughput, you can use OFDM with MIMO, extended rate physical (ERP), and multiuser (MU) technologies.

Bluetooth-WLAN Coexistence Problem

As Bluetooth and WLAN devices operate in the same 2.4 GHz frequency band, a mutual interference exist between the two wireless networks. This interference results in performance degradation. For example, consider the scenario shown in this figure. The scenario consists of two Bluetooth piconets collocated with a WLAN.



If the transmission in piconet 1 is overlapped in time and frequency by transmissions from piconet 2 and/or the WLAN, a Bluetooth packet can be lost. This figure shows how the Bluetooth and WLAN devices share the 2.4 GHz frequency spectrum.



If the Bluetooth packets transmitted through the FHSS hops to the portion of the frequency spectrum occupied by the DSSS WLAN transmitter, then mutual interference occurs. This interference results in packet collisions. Factors such as the distance between WLAN and Bluetooth devices, the data traffic present in these two networks, power levels of the devices, and data rate of the WLAN network impact the level of interference. Additionally, different types of data traffic have different levels of sensitivity to the interference. For example, voice traffic can be more sensitive to interference than data traffic.

Bluetooth in Presence of 802.11b WLAN Interferer

A transmission that uses one spread spectrum technique interferes with a receiver that uses different spread spectrum technique. 802.11b WLAN devices operate in 22 MHz bandwidth. In Bluetooth, 22 of the 79 hopping channels are subject to interference. A frequency-hopping system like Bluetooth is vulnerable to interference from the adjacent channels as well. This vulnerability increases the total number of interference channels from 22 to 24. Based on these assumptions, the results shown in [3] quantify the packet error rate (PER) in Bluetooth transmissions with a 802.11b WLAN interferer. The results show that the network throughput decreases and network delay increases for Bluetooth in the presence of 802.11b interference.

802.11b WLAN in Presence of Bluetooth Interferer

When a Bluetooth device hops into the 802.11b passband, a packet collision can occur with the WLAN device. This collision occurs because 22 of the 79 Bluetooth channels fall within the WLAN passband. As 802.11b devices support four data rates (1, 2, 5, and 11 Mbps), the transmission time of the WLAN packets may vary significantly for packets carrying the exact same data. Increasing the duration of the WLAN packet increases the likelihood that the packet collides with an interfering Bluetooth packet. If automatic data rate scaling is implemented and enabled in the WLAN device, the Bluetooth interference can cause the WLAN device to scale to a lower data rate. Lower data rate increase the temporal duration of the WLAN packets. This increase in packet duration can lead to frequent packet collisions with the interfering Bluetooth packets. In some implementations, the frequent packet

collisions can result in WLAN scaling down its data rate to 1 Mbps. In this case, to ensure reliable packet delivery, the IEEE 802.11 medium access (MAC) layer incorporates an acknowledgement (ACK) and retransmission mechanism.

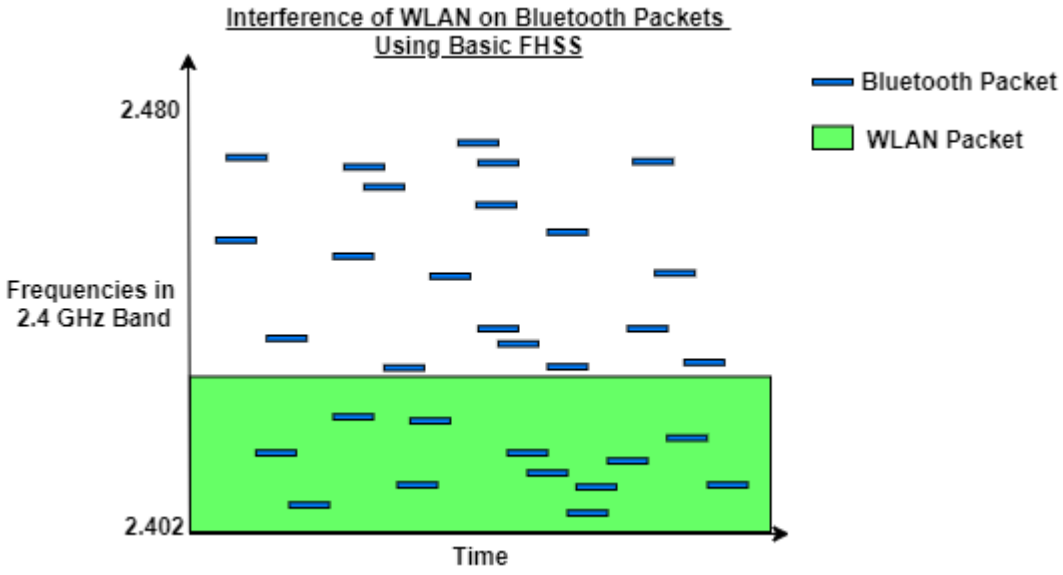
Coexistence Mechanisms

Interference between Bluetooth and WLAN can be addressed by two coexistence mechanisms – noncollaborative and collaborative.

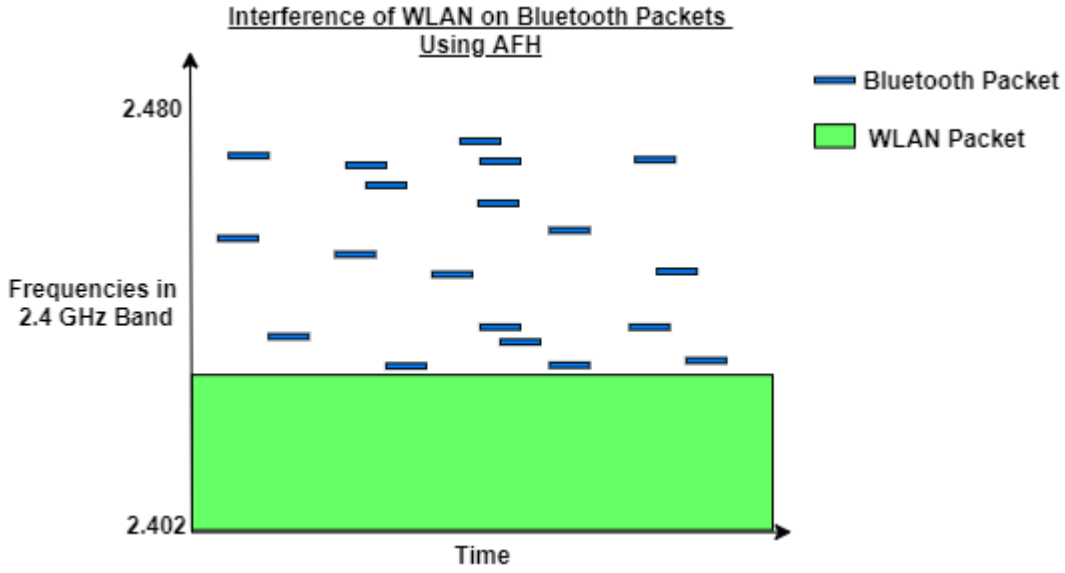
Noncollaborative Coexistence

Noncollaborative mechanisms do not exchange information between two wireless networks. These coexistence mechanisms are applicable only after a WLAN or Bluetooth piconet is established and the data is to be transmitted. These coexistence mechanisms do not help in the process of establishing a WLAN or Bluetooth piconet. As per the recommended practices mentioned in [3], these noncollaborative coexistence mechanisms are used to mitigate interference between Bluetooth and WLAN.

- Adaptive frequency hopping (AFH) — Prior to the emergence of AFH, Bluetooth devices implemented the basic FHSS signal structuring scheme. The FHSS scheme often resulted in Bluetooth and WLAN packet transmissions interfering with each other, as shown in this figure.



On the contrary, AFH enables Bluetooth to adapt to its environment by identifying fixed sources of WLAN interference and excluding them from the list of available channels. This figure shows the previous scenario with AFH enabled.



AFH dynamically alters the frequency hopping sequence to avoid the interference observed by the Bluetooth devices. AFH operates through these four processes.

- AFH capability discovery: This process informs the Central about the Peripheral(s) that support AFH and the associated parameters.
- Channel classification: This process classifies the channels as *good* or *bad*. Channel classification takes place in the Central and optionally in the Peripheral(s).
- Channel classification information exchange: This process uses AFH link manager protocol (LMP) commands to exchange information between the Central and the supporting Peripheral(s) in the piconet.
- Adaptive hopping: This process adaptively selects *good* channels for frequency hopping.

For more information about how AFH mitigates interference and enables coexistence between Bluetooth and WLAN, see “End-to-End Bluetooth BR/EDR PHY Simulation with WLAN Interference and Adaptive Frequency Hopping” on page 5-20.

Note For more information about AFH, see Annex B of IEEE 802.15.2 Task Group [3].

- Adaptive interference suppression — This mechanism is exclusively related to signal processing in the WLAN physical layer (PHY). The adaptive interference suppression mechanism requires a Bluetooth receiver collocated with a WLAN receiver. The WLAN receiver has no prior knowledge of the timing or frequency used by the Bluetooth network. The WLAN receiver uses an adaptive filter to estimate and cancel the interfering signal.

Note For more information about adaptive interference suppression, see Clause 8 of IEEE 802.15.2 Task Group [3].

- Adaptive packet selection and scheduling — Bluetooth transmissions involve various packet types with different configurations such as packet length and degree of error protection used. By selecting the best packet type according to the channel condition of the upcoming frequency hop, better throughput and network performance can be achieved. Additionally, packet transmissions can be scheduled efficiently so that the Bluetooth devices transmit during hops that are outside of

the WLAN frequencies and refrain from transmitting while in-band. This type of packet transmission scheduling minimizes mutual interference and also increases the throughput of Bluetooth networks.

Note For more information about adaptive packet selection and scheduling, see Clause 9 of IEEE 802.15.2 Task Group [3].

- Packet scheduling for synchronous connection-oriented (SCO) links — Voice applications are among the most sought-after applications for Bluetooth devices but are vulnerable to interference. Interference from an in-band WLAN network degrades the voice quality of the Bluetooth SCO link, making it inaudible to the users. This noncollaborative coexistence mechanism recommends improvements that can significantly improve the quality-of-service (QoS) for SCO links. The fundamental idea is to enable the SCO link the flexibility of selecting hops that are out-of-band with the collocating WLAN spectrum for transmission. The duty cycle of the SCO link does not change.

Note For more information about packet scheduling for SCO links, see Annex A of IEEE 802.15.2 Task Group [3].

- Packet scheduling for asynchronous connection-oriented logical (ACL) links — This mechanism defines a procedure to minimize the impact of WLAN interference on Bluetooth devices by using these two components.

  - Channel classification: It is performed on every Bluetooth receiver and is based on the measurements conducted per frequency or channel to locate the presence of interference. A channel is considered as *good* if it can correctly decode a received packet. Otherwise, the channel is considered as *bad*. Good and bad channels are classified based on different criteria such as the received signal strength indicator (RSSI), PER, or negative ACKs.
  - Central delay policy: It uses the information available in the channel classification table to avoid packet transmission in a *bad* channel. Because the Central device controls and manages all transmissions in a piconet, the delay rule must be implemented in the Central device only. Also, a Peripheral transmission must follow each Central transmission. Therefore, the Central checks the receiving frequency of the Peripheral and its own receiving frequency before choosing to transmit a packet in a given frequency hop.

Note For more information about packet scheduling for ACL links, see Clause 10 of IEEE 802.15.2 Task Group [3].

Collaborative Coexistence

In collaborative coexistence mechanisms, two wireless networks collaborate and exchange network-related information. As per the recommended practices stated in [3], the three collaborative coexistence mechanisms are:

- Alternating wireless medium access (AWMA) — In the AWMA mechanism, a WLAN radio and a Bluetooth radio are collocated in the same physical unit, enabling a wired connection between the two radios. The collaborative coexistence mechanism uses this wired connection to coordinate access to the wireless medium between WLAN and Bluetooth. The AWMA mechanism uses part of the wireless IEEE 802.11 beacon interval for the Bluetooth operations. From a timing perspective, the medium assignment alternates between usage following the IEEE 802.11 procedures and usage following the Bluetooth procedures. Each wireless network limits its transmissions to the appropriate time segment, thus preventing mutual interference between the two networks.

Note For more information about AWMA, see Clause 5 and Annex I of IEEE 802.15.2 Task Group [3].

- Packet traffic arbitration (PTA) — In the PTA mechanism, the WLAN station and the Bluetooth device are collocated. The PTA control entity provides per-packet authorization of all transmissions. This mechanism can deny permission for transmission if it has chances of collisions. The PTA mechanism dynamically coordinates sharing of the wireless medium based on the traffic load of WLAN and Bluetooth. If a collision occurs, the PTA mechanism prioritizes transmission based on the priorities of different packets. Using the PTA mechanism in case of high variability in the WLAN and Bluetooth traffic load or whenever a Bluetooth SCO link needs to be supported.

Note For more information about PTA, see Clause 6 and Annex J of IEEE 802.15.2 Task Group [3].

- Deterministic interference suppression — In this mechanism, a null is inserted in the WLAN receiver at the frequency of the Bluetooth signal. Because Bluetooth devices hop to a new frequency for each packet transmission, the WLAN receiver must know the hopping pattern and timing of the Bluetooth device. The hopping pattern and timing is obtained by using a Bluetooth receiver as part of the WLAN receiver. Deterministic interference suppression is a collocated, collaborative coexistence mechanism.

Note For more information about deterministic interference suppression, see Clause 7 and Annex K of IEEE 802.15.2 Task Group [3].

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 17, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] *P802.15.2/D09 - IEEE Draft Recommended Practice for Information Technology Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks Specific Requirements - Part 15.2: Coexistence of Wireless Personal Area Networks With Other Wireless Devices Operating in Unlicensed Frequency Bands.* LAN/MAN Standards Committee, IEEE Computer Society, 2003, https://ieeexplore.ieee.org/document/4040972.

[4] Macleod, M D. “14 - Coding.” *In Telecommunications Engineer’s Reference Book*, edited by Fraidoon Mazda, 14–1. Butterworth-Heinemann, 1993. https://www.sciencedirect.com/science/article/pii/B9780750611626500204.

[5] Xiao, Yang, and Yi Pan. “Coexistence of Bluetooth Piconets and Wireless LAN.” *In Emerging Wireless LANs, Wireless PANs, and Wireless MANs: IEEE 802.11, IEEE 802.15, 802.16 Wireless Standard Family*, 151–85. Wiley, 2009. https://ieeexplore.ieee.org/abstract/document/8040602.

[6] “IEEE SA - The IEEE Standards Association - Home.” Accessed May 4, 2020. https://standards.ieee.org/.

[7] IEEE P802.11ax/D4.1. “*Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 1: Enhancements for High Efficiency WLAN*.” Draft

Standard for Information technology — Telecommunications and information exchange between systems. Local and metropolitan area networks — Specific requirements.

See Also

More About

- “Configure Bluetooth BR/EDR Channel with WLAN Interference and Pass the Waveform Through Channel” on page 9-20
- “WLAN Protocol Stack” (WLAN Toolbox)
- “WLAN PPDU Structure” (WLAN Toolbox)
- “Packet Size and Duration Dependencies” (WLAN Toolbox)

Bluetooth Mesh Networking

The Bluetooth technology [1] uses low-power radio frequency to enable short-range communication at a low cost. In 2010, Bluetooth 4.0 introduced a new variant known as Bluetooth low energy (LE), or Bluetooth Smart. Bluetooth LE supports a point-to-multipoint or broadcast communication that is useful for a short-range navigation beacon mode such as real-time locating systems used for asset-tracking and people tracking. Auxiliary improvements to Bluetooth technology were released with the introduction of Bluetooth 5.0 [2]. In addition to Bluetooth 5.0, the Bluetooth Special Interest Group (SIG) defined a new connectivity model for Bluetooth LE, known as the Mesh Profile [4]. The Bluetooth LE Mesh Profile establishes the option of many-to-many communication links for Bluetooth LE devices and is optimized for creating large scale Internet of Things (IoT) networks. The mesh stack defined by the Bluetooth LE Mesh Profile is located on top of the Bluetooth LE core specification. For more information about the mesh stack, see “Bluetooth Mesh Stack” on page 8-64. This new mesh networking capability of Bluetooth is ideally suited for building automation, large scale sensor networks, and other IoT solutions that require tens and hundreds of devices to be reliably and securely set up.

Motivation for Bluetooth Mesh Networking

As mesh networking topologies offer the best way to satisfy different increasingly common and popular communications requirements, Bluetooth mesh networking was introduced. Some of the fundamental requirements include:

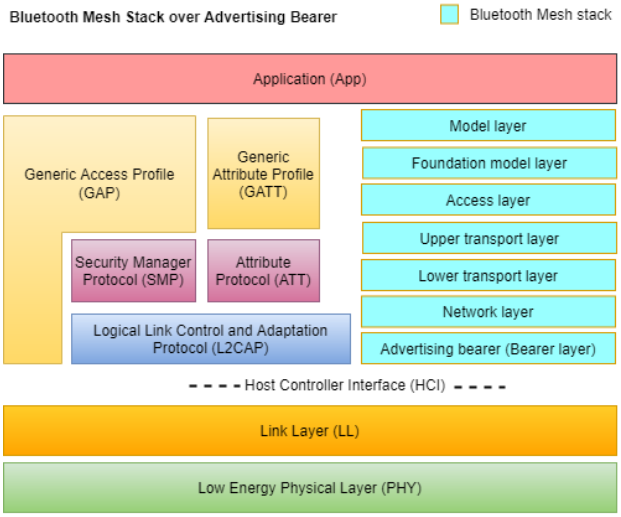
- Low energy consumption
- Extension of the coverage area through multihop communication
- Increased network scalability through efficient use of radio resources
- Interoperability with different standards
- Increased communications security through authentication and encryption
- Improved system reliability through multipath message-oriented communication
- Ability to deliver optimal network responsiveness

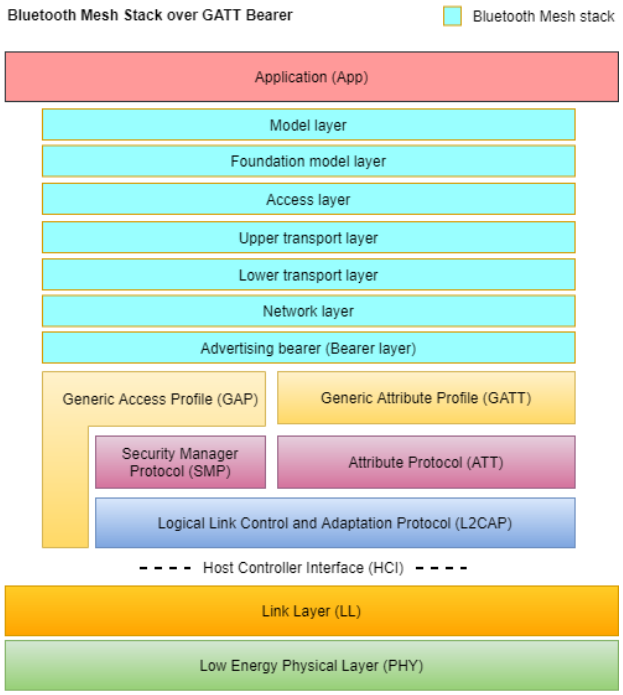
Other low-power wireless communication technologies, such as ZigBee and Thread, also support mesh networking topologies. However, these technologies often face issues such as low data rates, restricted number of *hops* when relaying data across the mesh, limitations in scalability often caused by the way radio channels are used, and delays when following procedures to change the device composition of the mesh topology. Additionally, these wireless communication technologies are not supported by standard smartphones, tablets, and PCs. The Bluetooth mesh meets the previously mentioned requirements and creates an industry-standard mesh communications technology based on Bluetooth LE.

Bluetooth mesh networking integrates the trusted, global interoperability with an evolved, trusted ecosystem to create industrial-grade device networks.

Bluetooth Mesh Stack

This figure illustrates how the Bluetooth mesh stack fits into the standard Bluetooth LE protocol stack. The figure shows the Bluetooth mesh stack over the Bluetooth LE advertising bearer and generic attribute (GATT) bearer.





The Bluetooth mesh stack consists of these layers:

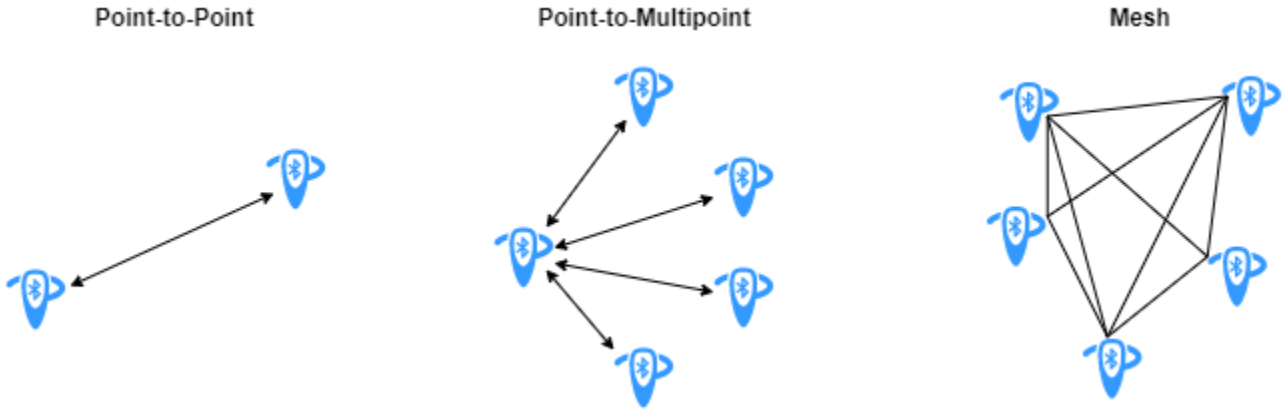
- Model layer: This layer defines the models, messages, and states required for use-case scenarios. For example, to change the state of a light to On or Off, the Bluetooth nodes use the Generic OnOff Set message from the Generic OnOff model.
- Foundation model layer: This layer defines the models, messages, and states required to configure and manage the mesh network. This layer also configures element, publish, and subscription addresses.
- Access layer: This layer defines the interface to the upper transport layer and the format of the application data. It also controls the encryption and decryption of the application data in the upper transport layer.
- Upper transport layer: This layer defines functionalities such as encryption, decryption, and authentication of the application data and is designed to provide confidentiality of access messages. This layer is also responsible for generating transport control messages (*Friendship* and heartbeat) internally and transmits those messages to a peer upper transport layer. The network layer encrypts and authenticates these messages.
- Lower transport layer: This layer defines functionalities such as segmentation and reassembly of upper transport layer messages into multiple lower transport layer messages to deliver large upper transport layer messages to other nodes. This layer also defines the friend queue used by the Friend node to store the lower transport layer messages for a Low Power node.
- Network layer: This layer defines functionalities such as encryption, decryption, and authentication of the lower transport layer messages. This layer transmits the lower transport layer messages over the bearer layer and relays the mesh messages when the *Relay* feature is enabled. The network layer also defines the message cache containing all recently seen network messages. If the received message is found to be in the cache, then it is discarded.
- Bearer layer: This layer defines the interface between the Bluetooth mesh stack and the Bluetooth LE protocol stack. This layer is also responsible for creating a mesh network by provisioning the

mesh nodes. The two types of bearers supported by the Bluetooth mesh are advertising bearer and GATT bearer.

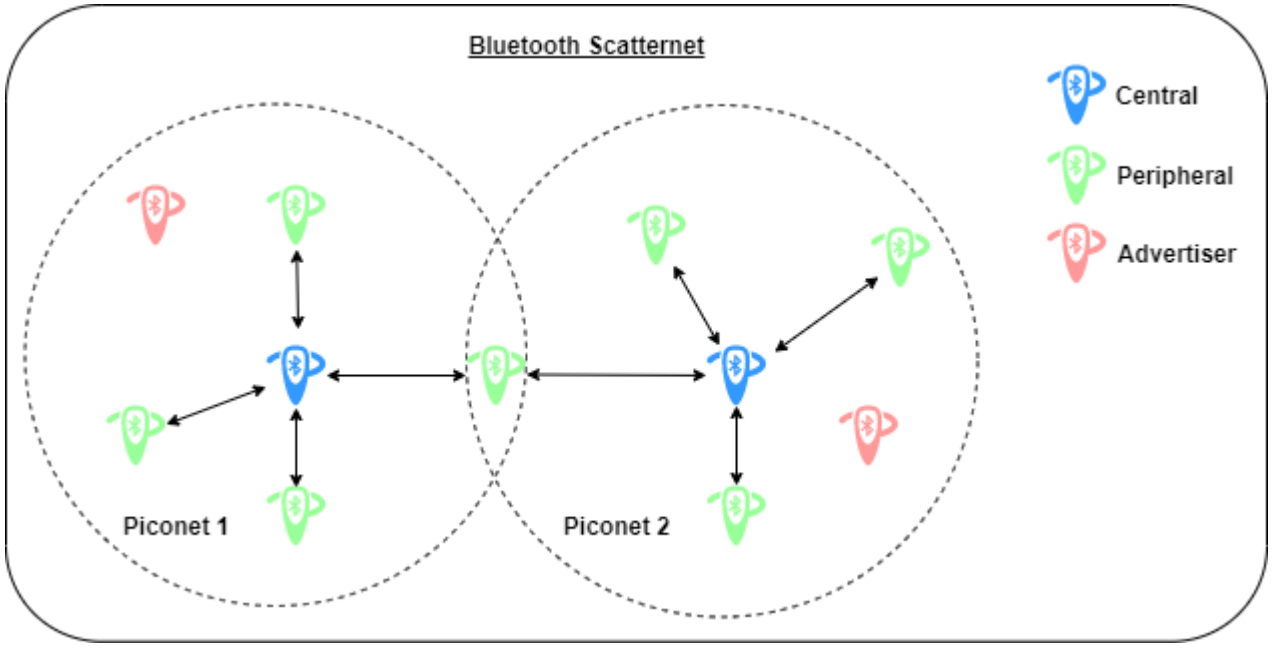
Bluetooth LE is the wireless communications protocol stack of which the Bluetooth mesh makes use. For more information about Bluetooth LE protocol stack, see “Bluetooth LE Protocol Stack” on page 8-9.

Bluetooth Connection Topologies

Most Bluetooth LE devices communicate with each other using a simple point-to-point (one-to-one communication) or point-to-multipoint (one-to-many communication) topology as shown in this figure.



Devices using one-to-one communication operate in a Bluetooth *piconet*. As shown in this figure, each piconet consists of a device in the role of *Central*, with other devices in the *Peripheral* or *Advertiser* roles. Before joining the piconet, each *Peripheral* node is in an advertiser role. Multiple piconets connect to each other, forming a Bluetooth *scatternet*.



For example, a smartphone with an established one-to-one connection to a heart rate monitor over which it can transfer data is an example of point-to-point connection.

On the contrary, the Bluetooth mesh enables you to set up many-to-many communication links between Bluetooth devices. In a Bluetooth mesh, devices can relay data to remote devices that are not in the direct communication range of the source device. This enables a Bluetooth mesh network to extend its radio range and encompass a large geographical area containing a large number of devices. Another advantage of the Bluetooth mesh over point-to-point and point-to-multipoint topologies is the capability of self healing. The self-healing capability of the Bluetooth mesh implies that the network does not have any single point of failure. If a Bluetooth device disconnects from the mesh network, other devices can still send and receive messages from each other, which keeps the network functioning.

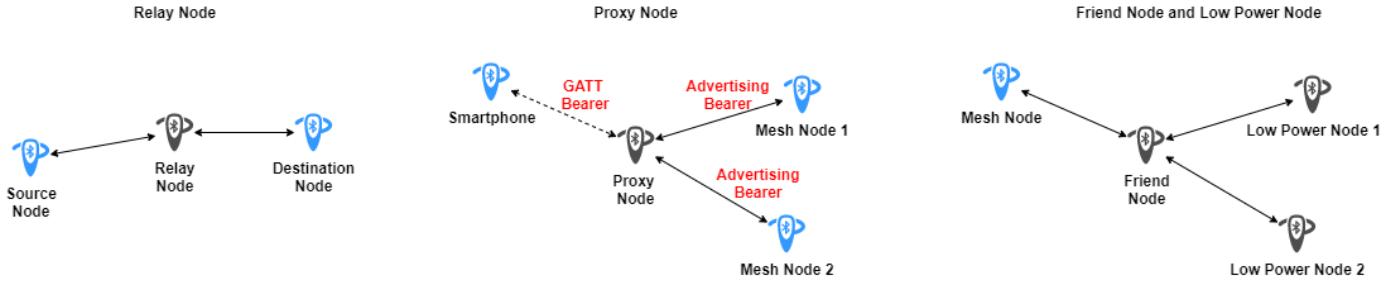
Fundamentals of Bluetooth Mesh Networking

These concepts of Bluetooth mesh networking serve as the foundation to study the functionality of a Bluetooth mesh network.

Devices and Nodes

Devices that belong to a Bluetooth mesh network are known as *nodes*. Devices that are not a part of a Bluetooth mesh network are known as *unprovisioned devices*. The process of transforming an unprovisioned Bluetooth device into a node is called “Provisioning” on page 8-71. Each node can send and receive messages either directly or through relaying from node to node. This figure shows a web of Bluetooth nodes spread across the MathWorks, Natick office.

Each Bluetooth mesh node might possess some optional features enabling them to acquire additional, special capabilities. These features include the *Relay*, *Proxy*, *Friend*, and the *Low Power* features. The Bluetooth mesh nodes possessing these features are known as Relay nodes, Proxy nodes, Friend nodes, and Low Power nodes (LPNs), respectively.

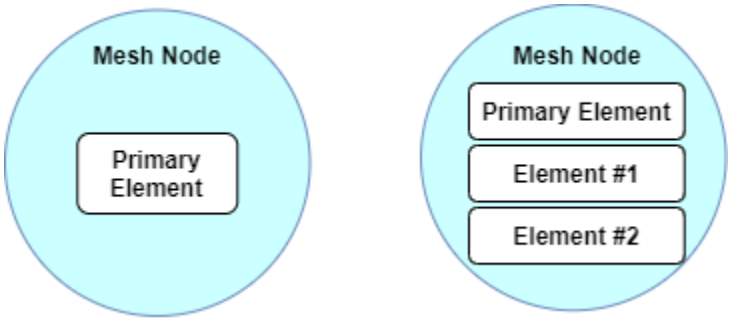


- Relay nodes: Bluetooth mesh nodes that possess the Relay feature are known as Relay nodes. These nodes use the relaying mechanism to retransmit the received messages through multiple hops. Depending on the power source and computational capacity, a mesh node might become a Relay node.
- Proxy nodes: To enable communication between a Bluetooth LE device that do not possess a Bluetooth mesh stack and the nodes in the mesh network, proxy nodes can be used. A Proxy node acts as an intermediary and utilizes the proxy protocol with generic attribute profile (GATT) operations. For example, as shown in the preceding figure, a smartphone that does not support the Bluetooth mesh stack interacts with mesh nodes by a Proxy node through GATT operations.
- Friend node: Bluetooth mesh nodes that do not have any power constraints are good exemplars for being Friend nodes. LPNs work in collaboration with Friend nodes. A Friend node stores messages destined to an LPN and delivers the messages to the LPN whenever the LPN polls the Friend node for the *waiting messages*. The relationship between an LPN and a Friend node is called “Friendship” on page 8-72.
- Low Power node: Bluetooth mesh nodes that are power constrained can use the Low Power feature to minimize the On time of the radio and conserve energy. Such nodes are known as LPNs. LPNs are predominantly concerned with sending messages but have a need to occasionally receive messages. For example, a temperature monitoring sensor that is powered by a small coin cell battery sends a temperature reading once per minute whenever the temperature is above or below the configured threshold values. If the temperature stays within the thresholds, the LPN sends no message.

Note For more information about Bluetooth mesh features, see Sections 3.4.6, 3.6.6.3, 3.6.6.4, and 7.2 of the Bluetooth Mesh Profile [4].

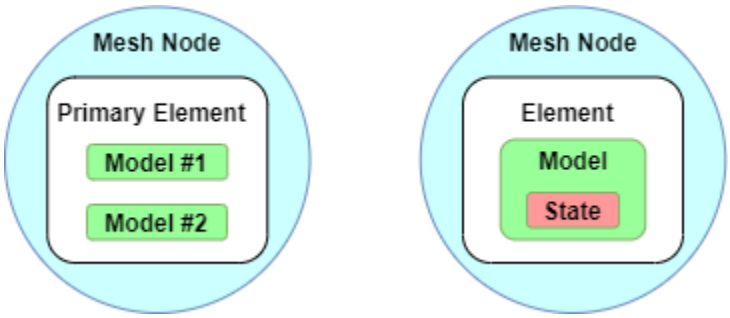
Elements, Models, and States

Some Bluetooth mesh nodes possess one or more independent constituent parts known as *elements*. This figure shows that a mesh node must have at least one element (primary element) but can have multiple elements.



Elements consists of entities that define the functionality of a node and the condition of the element. Each element in a mesh node has a unique *unicast address* that enables each element to be addressed.

This figure shows the mesh node and its constituents. The basic functionality of a mesh node is defined and implemented by models. Models reside inside elements, and each element must have at least one model.



Bluetooth Core Specifications [2] defines these three categories of models.

- Server model: This model defines a collection of states, state transitions, state bindings, and messages that the element containing the model can send or receive. It also defines behaviors pertaining to messages, states, and state transitions.
- Client model: This model defines the messages that it can send or receive in order to acquire values of multiple states defined in the corresponding server model.
- Control model: This model comprises of a server model and client model. The server model enables communication with other client models, and the client model enables communication with server models.

A *state* is a value of a certain type that defines the condition of elements. Additionally, states also have associated behaviors. For example, consider a simple light bulb that can be either On or Off. The Bluetooth mesh defines a state called generic *OnOff*. When the light bulb acquires this state and a value of *On*, the bulb is illuminated. Similarly, when the light bulb acquires the generic *OnOff* state and a value of *Off*, the bulb is switched off.

Note For more information about Bluetooth mesh models and states, see Section 4 of the Bluetooth Mesh Profile [4].

Addresses and Messages

Bluetooth Core Specifications [2] defines these four types of addresses.

- Unassigned address: Nonconfigured elements or elements without any designated addresses have an unassigned address. Mesh nodes with unassigned addresses are not involved in messaging.
- Unicast address: During provisioning, a provisioner assigns a unicast address to each element in a node. Unicast addresses can appear in the source address field of a message, the destination address field of a message, or both. Messages sent to unicast addresses are processed by only one element. For more information about provisioning, see “Provisioning” on page 8-71.
- Virtual address: A virtual address represents a set of destination addresses. Each virtual address logically represents a 128-bit label universally unique identifier (UUID). The Bluetooth nodes can publish or subscribe to these addresses.
- Group address: Group addresses are types of multicast addresses that represent multiple elements from one or more nodes. Group addresses can be fixed (allocated by Bluetooth SIG) or dynamically assigned.

Communication in Bluetooth mesh networks is realized through messages. A message can be a control message or an access message.

- Control message: These messages are involved in the actual functioning of the Bluetooth mesh network. For example, heartbeat and friend request messages are types of control messages.
- Access message: These messages enable client models to retrieve or set the values of states in server models. Access messages can be acknowledged or unacknowledged. Acknowledged messages are transmitted to each receiving element. The receiving element acknowledges the messages by sending a status message. No response is sent to an unacknowledged message. Bluetooth mesh network status messages are an example of unacknowledged messages.

For every state, the server model supports a set of messages. For example, these message can include a client model requesting the value of a state or requesting to change a state and a server model sending messages about the states or a change in the state.

Messages are identified by opcodes and have associated parameters. A unique opcode defines these three types of mesh messages:

- GET message: This mesh message requests the state value from one or more nodes.
- SET message: This mesh message changes the value of a given state.
- STATUS message: This mesh message is used is these scenarios.
  - In response to a GET message containing the state value
  - In response to an unacknowledged SET message
  - Sent independently to report the status of an element

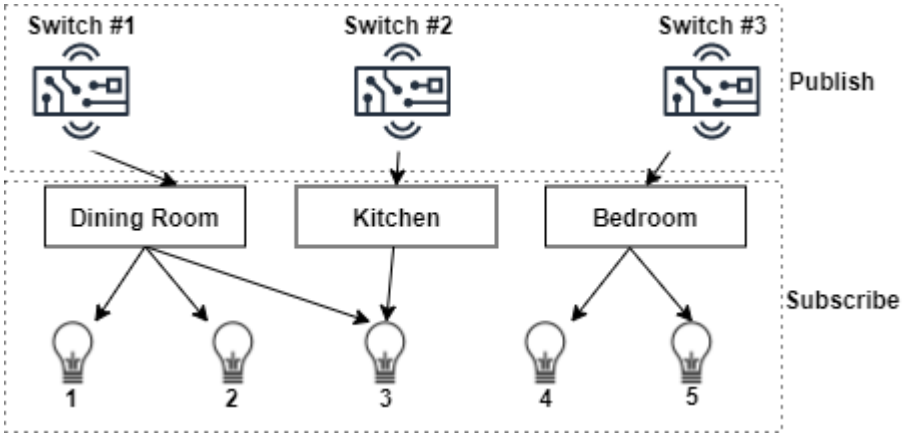
Note For more information about Bluetooth mesh addresses, see Section 3.4.2 of the Bluetooth Mesh Profile [4].

Publish/Subscribe Message-Oriented Communication System

Bluetooth mesh networking implements a publish/subscribe message-oriented communication system. Such an approach ensures that different types of products can coexist in a mesh network

without being affected by messages from devices they do not need to listen to. The act of sending a message is known as *publishing*. Based on the configuration, the mesh nodes select messages sent to specific addresses for processing. This technique is known as *subscribing*. A publisher node sends messages to those nodes that have subscribed to the publisher. Typically, mesh messages are addressed to group or virtual addresses.

Consider the example shown in this figure. Each room can subscribe to messages from the specific light bulbs for that room. Additionally, these messages can be unicast, multicast, broadcast, or any combination of these three options.



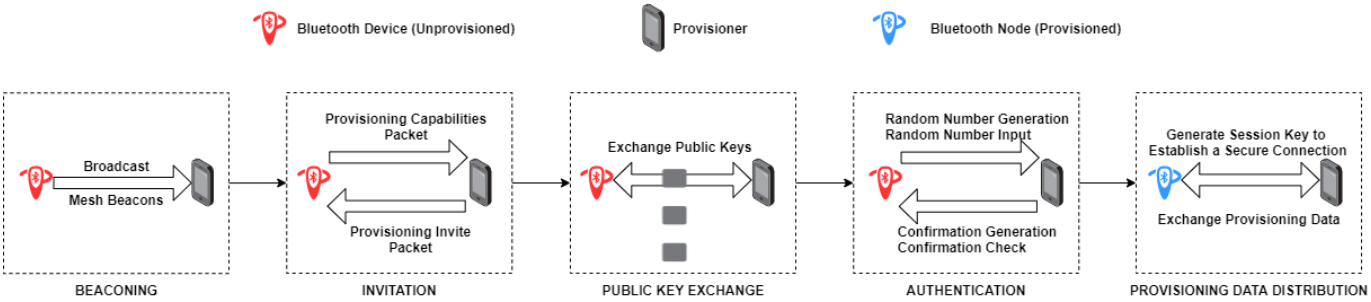
Switch #1 publishes to the group address Dining Room. Light bulbs 1, 2, and 3 each subscribe to the Dining Room address and therefore process messages published to this address. Switch #2 publishes to the group address Kitchen. Light bulb 3 subscribes to the Kitchen address and is the only bulb controlled by Switch #2. Similarly, Switch #3 publishes to the group address Bedroom and hence controls light bulbs 4 and 5. This example also shows that the mesh nodes can subscribe to messages addressed to more than one unique address.

The group and virtual addresses used in the publish/subscribe communication system enables removing, replacing, or adding new nodes to the mesh network without any reconfiguration. For example, an additional light bulb can be added in Kitchen using the “Provisioning” on page 8-71 process and then configured to subscribe to the Kitchen group address. In this process, no other light bulbs are impacted.

Note For more information about the Bluetooth mesh publish/subscribe communication, see Section 2.3.8 of the Bluetooth Mesh Model Specification [5].

Provisioning

Provisioning is the process by which a Bluetooth device (unprovisioned device) joins the mesh network and becomes a Bluetooth mesh node. This process is controlled by a *provisioner*. A provisioner and the unprovisioned device follow a fixed procedure as defined in the Bluetooth Mesh Profile [4]. A provisioner is typically a smartphone running a provisioning application. The process of provisioning can be accomplished by one or more provisioners. This figure shows the five steps of provisioning.

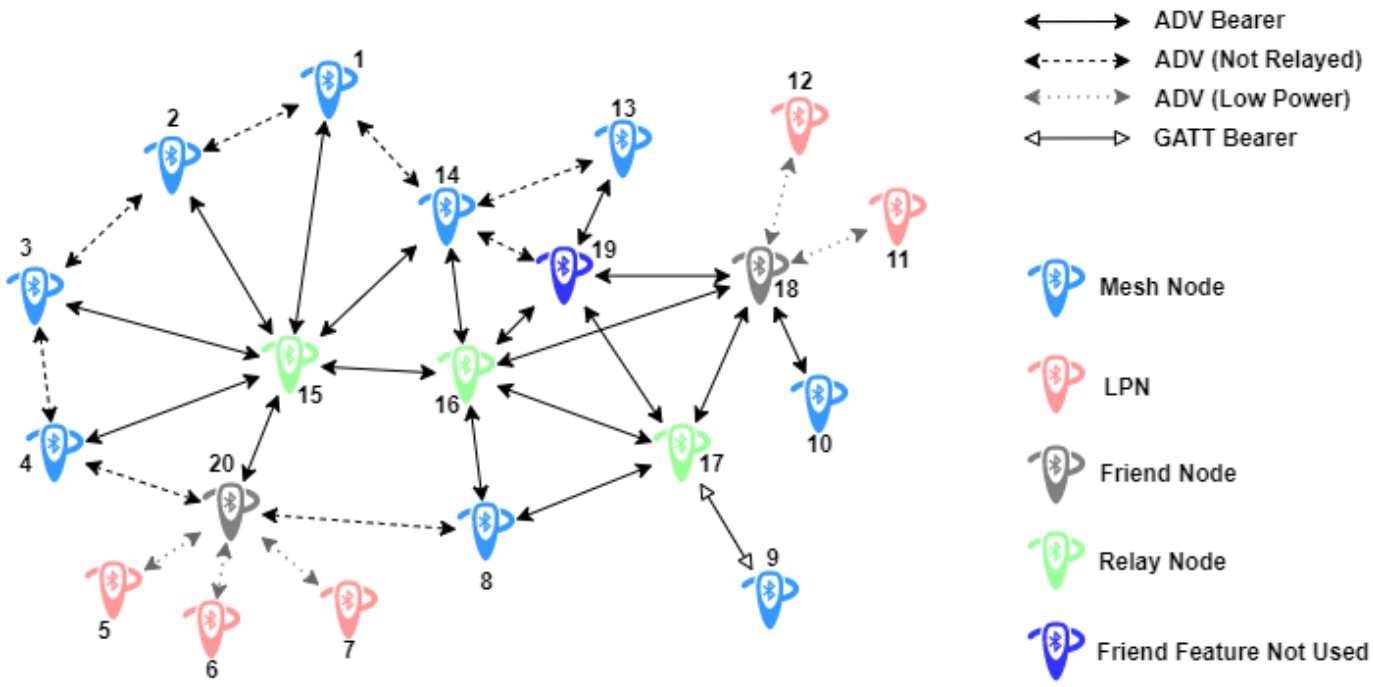


1. Beaconing: In this step, the unprovisioned Bluetooth device advertises its availability to be provisioned by sending the *mesh beacon* advertisements in the advertisement packets. A typical way to trigger beaconing is through a specified sequence of button clicks on the unprovisioned Bluetooth device.
2. Invitation: In this step, the provisioner invites the unprovisioned Bluetooth device for provisioning by sending a *provisioning invite protocol data unit* (PDU). The unprovisioned Bluetooth device responds with information about its capabilities by sending a *provisioning capabilities PDU*.
3. Public key exchange: In this step, the provisioner and the unprovisioned device exchange their public keys. These public keys can be static or ephemeral, either directly or using an out-of-band (OOB) method.
4. Authentication: In this step, the unprovisioned device outputs a random, single or multidigit number to the user in some form, using an action appropriate to its capabilities. The authentication method depends on the capabilities of both devices used. Irrespective of the authentication method that the Bluetooth node uses, the authentication also includes a confirmation value generation step and a confirmation check step.
5. Provisioning data distribution: After successfully completing the authentication step, the provisioner and the unprovisioned device generate a session key by using their private keys and the exchanged peer public keys. The provisioner and the unprovisioned device use the session key to secure the subsequent exchange of data needed to complete the provisioning process. This process includes the distribution of a security key called the *network key* (NetKey). After provisioning is completed, the provisioned device acquires the NetKey, a mesh security parameter called *IV Index*, and a unicast address assigned by the provisioner. At this point, the Bluetooth device can be termed as a Bluetooth mesh node.

Note For more information about the Bluetooth mesh provisioning, see section 5 of the Bluetooth Mesh Profile [4].

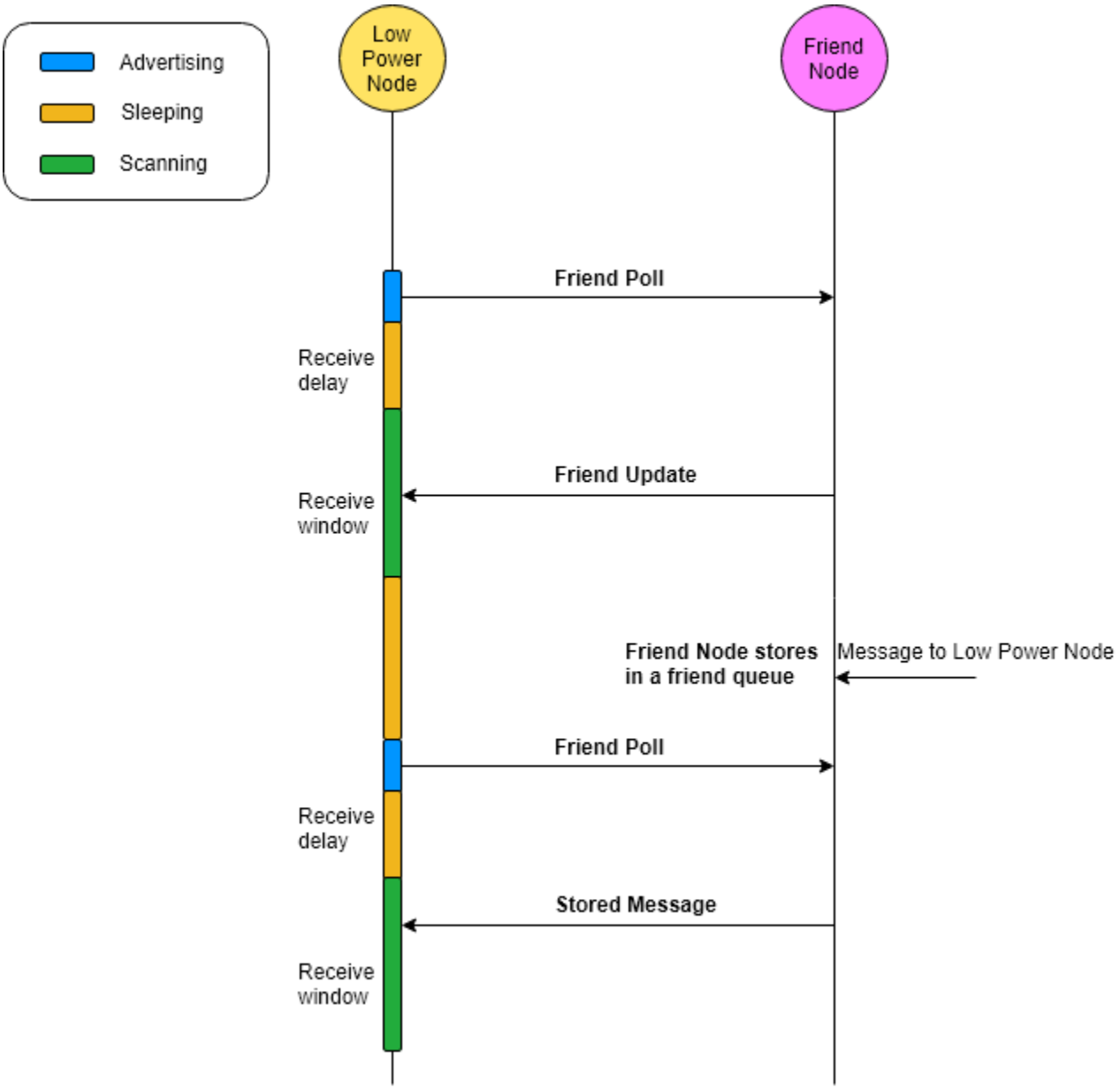
Friendship

To reduce the duty cycles of the LPN and conserve energy, the LPN must establish a *Friendship* with a mesh node supporting the Friend feature. This figure from [4] shows the relationship between LPNs and Friend nodes.



LPNs 5, 6, and 7 have a Friendship relationship with Friend node 20. Friend node 18 has Friendship with LPNs 11 and 12. Subsequently, Friend node 20 stores and forwards messages addressed to LPNs 5, 6, and 7. Similarly, Friend node 18 stores and forwards messages addressed to LPNs 11 and 12. Forwarding by the Friend node occurs only when the LPN wakes up and polls the Friend node for messages awaiting delivery. This mechanism enables all of the LPNs to conserve energy and operate for longer durations.

This figure shows the Bluetooth mesh messages exchanged between an LPN and a Friend node to establish Friendship.



The Bluetooth nodes use these timing parameters to establish Friendship:

- Receive delay: This parameter specifies the time between when an LPN sends a request and listens for a response from the Friend node. The LPN is in sleep state for the complete duration of the receive delay.
- Receive window: This parameter specifies the time for which an LPN listens for a response from a Friend node. The LPN is in the scanning state for the complete duration of the receive window.
- Poll timeout: This parameter specifies the maximum time between two successive requests from an LPN. Within the poll timeout, if the Friend node or the LPN fails to a receive request or response from the other node, the Friendship is terminated.

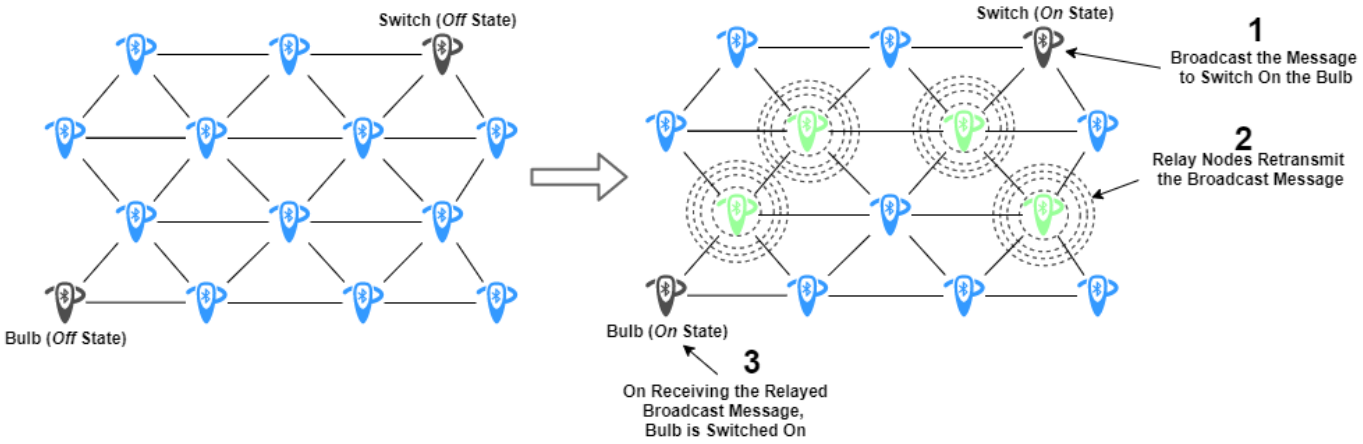
  Periodically, LPNs poll Friend nodes for any data messages stored in the friend queue. After polling the Friend node, the LPN enters a sleep state for the duration of receive delay. The Friend

node uses the receive delay to prepare the response for the LPN. After the receive delay, the Friend node responds to the LPN before the sum of the receive delay and the receive window. For more information about Friendship, see “Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks” on page 6-2 and “Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network” on page 9-49 examples.

Note For more information about Friendship, see Section 3.6.6 of the Bluetooth Mesh Profile [4].

Managed Flooding

Many mesh networks implement routing mechanisms to relay messages in the network. Another mechanism to relay messages is to flood the network with messages being relayed without any consideration of the optimal routes to reach their respective destinations. Bluetooth mesh networking uses an approach known as *managed flooding* that comprises of both of these mechanisms. Bluetooth mesh networking leverages the strengths of the flooding approach and optimizes its operations such that it is both reliable and efficient. This figure demonstrates the process of managed flooding in the Bluetooth mesh.



The figure illustrates communication between a switch and a connected light bulb in a Bluetooth mesh. Initially, the switch and bulb are in the *Off* state. Changing the switch to the *On* state broadcasts a message to turn on the bulb. All of the mesh nodes in range of the switch hear the message, but only the relay nodes retransmit the message. The message is relayed in this manner across the network until it reaches the bulb and turns on the bulb. This process is termed as managed flooding. To optimize this process, Bluetooth mesh implements these measures.

- Heartbeat messages: These messages are transmitted by mesh nodes periodically to indicate to other mesh nodes that the node sending the heartbeat is still active. Heartbeat messages contain information that enables the receiving nodes to determine the number of hops between it and the sending node.
- Time to live (TTL): This field is present in all Bluetooth mesh PDUs. It manages the maximum number of hops over which a message is relayed. Setting the TTL value enables mesh nodes to have control over relaying and to conserve energy. Heartbeat messages enable nodes to determine the optimal TTL value required for each published message.

- Message cache: Every mesh node contains a message cache to determine whether it has seen a message before. If the node has seen the message before, the node discards the message and avoids unnecessary processing higher up the stack.
- Friendship: For more information about the Friendship mechanism, see “Friendship” on page 8-72.

For more information about managed flooding in Bluetooth mesh networks, see “Bluetooth Mesh Flooding in Wireless Sensor Networks” on page 6-10 example.

Applications of Bluetooth Mesh Networking

The addition of mesh capabilities to Bluetooth creates opportunities for applying Bluetooth mesh networking in automation and IoT domains. These are some of the prominent applications of Bluetooth mesh networking.

- Smart home automation — Bluetooth mesh networking can be used to simplify the smart home automation processes by enabling a mesh network of devices (such as smart bulbs, thermostats, and vents) readily established and provisioned with the user's smartphone. A Bluetooth mesh network of such connected devices can be used to relay messages through multiple paths, thus increasing the communication reliability and network scalability. Bluetooth mesh does not have any single point of failure. This prevents service outages if a mesh node fails. For example, consider a home scenario with a Bluetooth mesh network of all lighting devices. If some of the lighting devices in the mesh network fail, the messages from the rest of the mesh can still reach the user’s control device.
- Beaconing — One of the prominent use case of Bluetooth mesh networking is the *beaconing*. In beaconing, an external event triggers a mesh node to transmit data. This data can include sensor information, location information, or point-of-interest information. Any mesh node can integrate one or more beacon standards (such as iBeacon of Apple or EddyStone from Google) and can be transformed into a virtual Bluetooth beacon while operating as a Bluetooth mesh node. This approach can enable new use-case scenarios, such as indoor positioning, asset tracking, and point-of-interest information delivery. Use cases involving Bluetooth direction finding (introduced in Bluetooth Core Specifications 5.1 [3]) implement beaconing to support high-accuracy direction finding. Bluetooth mesh combined with direction-finding features such as angle of arrival and angle of departure can pave way for many commercial IoT-based use case scenarios. For more information about beaconing and Bluetooth direction-finding capabilities, refer “Bluetooth Location and Direction Finding” on page 8-18 topic.
- Automated irrigation systems and plant lighting — Bluetooth mesh networks can be used to develop intelligent solutions in automated irrigation systems and plant lighting. For example, as land resources decline, many plants are grown in greenhouses for higher harvesting yields. Automated indoor planting needs a combination of appropriate light source with a smart control system and a Bluetooth mesh network. In this scenario, Bluetooth mesh modules are placed into the light sources. This mechanism enables the mesh modules to automate the control of the light source, soil moisture, air temperature, moisture, humidity, and automatic irrigation.
- Low latency — Bluetooth mesh networks can be useful in low-latency use-case scenarios. In networks where round-trip time is of a high significance, specific functional nodes and relay nodes can be used to minimize the communication delay while maintaining coverage and reliability.

Mesh networking with the Bluetooth standard can be used in intelligent IoT solutions to facilitate home, commercial, and industrial automation. In summary, Bluetooth mesh networking is comparable to all other Bluetooth connectivity and establishes hub-less networks that expand the coverage and reliability of Bluetooth systems.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.0. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[4] Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Profile." Version 1.0.1. https://www.bluetooth.com/.

[5] Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Model Specification." Version 1.0.1. https://www.bluetooth.com/.

See Also

More About

- “Create, Configure and Simulate Bluetooth Mesh Network” on page 9-45
- “Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network” on page 9-49
- “Bluetooth Mesh Flooding in Wireless Sensor Networks” on page 6-10
- “Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks” on page 6-2

Bluetooth LE Node Statistics

The Bluetooth low energy (LE) network simulation captures these statistics as a structure. This structure is returned as an output by the `statistics` object function of the `bluetoothLENode` object. The fields of this structure depend on the value of the Role property of the `bluetoothLENode` object.

Statistics of Central and Peripheral Nodes

If you set the Role property of the `bluetoothLENode` object to `"central"` or `"peripheral"`, these are the fields of the structure.

Structure Field	Description
`App`	Application statistics, returned as a structure. If the network contains multiple Peripheral nodes, this field returns an array of structures.
`LL`	Link layer (LL) statistics, returned as a structure. If the network contains multiple Peripheral nodes, this field returns an array of structures.
`PHY`	Physical layer (PHY) statistics, returned as a structure.

These are the fields of the `App` structure.

Structure Field	Description
`DestinationID`	Destination node ID for the packets transmitted from the application layer
`TransmittedPackets`	Total number of packets transmitted by the application layer
`TransmittedBytes`	Total number of bytes transmitted from the application layer
`ReceivedPackets`	Total number of packets received at the application layer
`ReceivedBytes`	Total number of bytes received at the application layer
`AveragePacketLatency`	Average packet latency in seconds

These are the fields of the `LL` structure.

Structure Field	Description
`TransmittedPackets`	Total number of packets transmitted from the LL
`TransmittedEmptyPackets`	Total number of empty packets transmitted from the LL
`TransmittedDataPackets`	Total number of data packets transmitted from the LL

Structure Field	Description
`RetransmittedDataPackets`	Total number of data packets retransmitted from the LL
`TransmittedControlPackets`	Total number of control packets transmitted from the LL
`RetransmittedControlPackets`	Total number of control packets retransmitted from the LL
`TransmitQueueOverflow`	Total number of times LL transmit queue overflowed
`TransmittedBytes`	Total number of transmitted and retransmitted bytes from the LL
`TransmittedPayloadBytes`	Total number of unique bytes from the higher layers transmitted from the LL
`AcknowledgedPackets`	Total number of packets acknowledged at the LL
`ReceivedPackets`	Total number of packets received by the LL
`ReceivedDataPackets`	Total number of data packets received by the LL
`ReceivedControlPackets`	Total number of control packets received by the LL
`ReceivedDuplicatePackets`	Total number of duplicate packets received by the LL
`CRCFailedPackets`	Total number of cyclic redundancy check (CRC) failed packets received by the LL
`ReceivedBytes`	Total number of bytes received by the LL
`ReceivedPayloadBytes`	Total number of bytes received by the LL from the higher layers
`ReceivedEmptyPackets`	Total number of empty packets received by the LL
`SleepTime`	Time spent by LL in the sleep state, in seconds
`IdleTime`	Time spent by LL in the idle state, in seconds
`ListenTime`	Time spent by LL in the listen state, in seconds
`TransmissionTime`	Time spent by LL in the transmission state, in seconds
`AveragePacketLatency`	Average packet latency in seconds
`AverageRoundTripTime`	Average roundtrip time captured at the LL in seconds
`PacketLossRatio`	Ratio of total number of transmitted packets to the total number of acknowledged packets
`Throughput`	Throughput calculated at the LL based on the unique application bytes transmitted, in kb/s

These are the fields of the `PHY` structure.

Structure Field	Description
`TransmittedPackets`	Total number of packets transmitted by the PHY
`TransmittedBits`	Total number of bits transmitted by the PHY
`ReceivedPackets`	Total number of packets received at the PHY
`DecodeFailures`	Total number of packets that are unsuccessfully decoded at the PHY receiver
`InvalidAccessAddressPackets`	Total number of decoded packets whose decoded access address does not match the expected access address
`InvalidLengthPackets`	Total number of decoded packets whose length (in bytes) is less than the length field in the decoded header
`ReceivedBits`	Total number of bits received and successfully decoded
`PacketCollisions`	Total number of packets that experienced collision from other packets
`CoChannelCollisions`	Total number of packets that experienced co-channel collisions
`CollisionsWithBLE`	Total number of packets that experienced collision in the time domain with Bluetooth LE packets
`CollisionsWithNonBLE`	Total number of packets that experienced collision in the time domain with non-Bluetooth LE packets
`CollisionsWithBLEAndNonBLE`	Total number of packets that experienced collision in the time domain with Bluetooth LE packets and non-Bluetooth LE packets

Statistics of Broadcaster-Observer Node

If you set the Role property of the `bluetoothLENode` object to `"broadcaster-observer"`, these are the fields of the structure. The fields of the `PHY` structure for a node with the `"broadcaster-observer"` role are same as those for a node with the `"central"` or `"peripheral"` role.

Structure Field	Description
`App`	Application statistics, returned as a structure. If the network contains multiple Peripheral nodes, this field returns an array of structures.
`Transport`	Mesh transport layer statistics, returned as a structure.
`Network`	Mesh network layer statistics, returned as a structure.
`LL`	Mesh LL layer statistics, returned as a structure.
`PHY`	PHY statistics, returned as a structure.

These are the fields of the `App` structure.

Structure Field	Description
`SourceAddress`	Source address of messages originated from the mesh application layer
`DestinationAddress`	Destination address of messages originated from the mesh profile
`TransmittedPackets`	Total number of packets transmitted by the mesh application layer
`TransmittedBytes`	Total number of bytes transmitted from the mesh application layer
`ReceivedPackets`	Total number of packets received at the mesh application layer
`ReceivedBytes`	Total number of bytes received at the mesh application layer
`AveragePacketLatency`	Average packet latency in seconds

These are the fields of the `Transport` structure.

Structure Field	Description
`TransmittedDataMessages`	Total number of data messages transmitted from the transport layer
`ReceivedDataMessages`	Total number of data messages received by the transport layer
`TransmittedControlMessages`	Total number of control messages transmitted by the transport layer
`ReceivedControlMessages`	Total number of control messages received by the transport layer
`TransmittedDataBytes`	Total number of data bytes transmitted by the transport layer
`ReceivedDataBytes`	Total number of data bytes received by the transport layer

These are the fields of the `Network` structure.

Structure Field	Description
`TransmittedMessages`	Total number of messages transmitted from the network layer
`ReceivedMessages`	Total number of messages received by the network layer
`AcceptedMessages`	Total number of messages accepted at the network layer for further processing
`RelayedMessages`	Total number of messages relayed at the network layer

Structure Field	Description
`DroppedMessages`	Total number of messages dropped at the network layer

These are the fields of the `LL` structure.

Structure Field	Description
`TransmittedPackets`	Total number of packets transmitted from the LL
`ReceivedPackets`	Total number of packets received by the LL
`DroppedPackets`	Total number of packets dropped by the LL
`TransmittedBytes`	Total number of bytes transmitted and retransmitted from the LL
`ReceivedBytes`	Total number of bytes received by the LL
`CRCFailedPackets`	Total number of CRC failed packets received by the LL
`SleepTime`	Time spent by the LL in the sleep state, in seconds
`IdleTime`	Time spent by the LL in the idle state, in seconds
`ListenTime`	Time spent by the LL in the listen state, in seconds
`TransmissionTime`	Time spent by the LL in the transmission state, in seconds
`Throughput`	Throughput calculated at the LL based on the unique application bytes transmitted, in kb/s

Statistics of Isochronous-Broadcaster and Receiver Nodes

If you set the Role property of the `bluetoothLENode` object to `"isochronous-broadcaster"` or `"synchronized-receiver"`, these are the fields of the structure. The fields of the PHY structure for a node with the `"isochronous-broadcaster"` or `"synchronized-receiver"` roles are the same as those for a node with the `"central"`, `"peripheral"`, or `"broadcaster-observer"` role.

Structure Field	Description
`App`	Application statistics, returned as a structure.
`LL`	LL statistics, returned as a structure.
`PHY`	PHY statistics, returned as a structure

These are the fields of the `App` structure.

Structure Field	Description
`TransmittedPackets`	Total number of packets transmitted by the application layer
`TransmittedBytes`	Total number of bytes transmitted from the application layer

Structure Field	Description
`ReceivedPackets`	Total number of packets received at the application layer
`ReceivedBytes`	Total number of bytes received at the application layer
`AveragePacketLatency`	Average packet latency averaged over all the values captured in the packet latency vector, in seconds

These are the fields of the `LL` structure.

Structure Field	Description
`StandbyTime`	Time spent by the LL in the standby state, in seconds
`SleepTime`	Time spent by the LL in the sleep/idle state, in seconds
`ListenTime`	Time spent by the LL in the listen state, in seconds
`TransmissionTime`	Time spent by the LL in the transmit state, in seconds
`ReceivedDataPackets`	Number of data protocol data units (PDUs) decoded successfully at the receiver
`ReceivedEmptyPackets`	Number of empty packets received
`ReceivedDuplicatePackets`	Number of duplicate data packets received
`AggregatePacketlatency`	Sum of packet latencies of all the successfully received data packets in seconds
`ReceivedBytes`	Total number of bytes received successfully
`TransmittedDataPackets`	Number of unique data packets transmitted by the broadcaster in each broadcast isochronous stream (BIS)
`TransmittedControlPackets`	Number of control packets transmitted by the broadcaster
`TransmittedEmptyPackets`	Number of empty packets transmitted by the broadcaster
`CRCFailedPackets`	Number of packets at the receiver with CRC failure
`AveragePacketLatency`	Average packet latency in seconds

See Also

Functions
`statistics`

Objects
`bluetoothLENode` | `bluetoothLEConnectionConfig` | `bluetoothLEBIGConfig` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig`

More About

- “Create, Configure, and Simulate Bluetooth LE Network” on page 9-38
- “Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network” on page 9-41
- “Create, Configure and Simulate Bluetooth Mesh Network” on page 9-45
- “Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network” on page 9-49
- “Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario” on page 9-53

Tutorials

Generate and Decode Bluetooth Protocol Data Units

The Bluetooth Toolbox enables you to model the Bluetooth low energy (LE) link layer (LL), the logical link control and adaptation protocol (L2CAP), the generic access profile (GAP), the attribute protocol (ATT) layer and the Bluetooth basic rate/enhanced data rate (BR/EDR) baseband and link control layer. Use the configuration objects and functions to parameterize, generate, and decode the Bluetooth protocol data units (PDUs) based on Bluetooth Core Specification 5.3 [2].

Generate and Decode Bluetooth LE ATT PDU

This example enables you to generate and decode a Bluetooth LE ATT PDU using the `bleATTPDU` and `bleATTPDUDecode` functions, respectively. Use the `bleATTPDUConfig` configuration object to set the configuration parameters for Bluetooth LE ATT PDU generation.

Create a Bluetooth LE ATT PDU configuration object, specifying the opcode as "`Information request`".

```
cfgATT = bleATTPDUConfig(Opcode="Information request")
```

```
cfgATT =
  bleATTPDUConfig with properties:

         Opcode: 'Information request'
    StartHandle: '0001'
      EndHandle: 'FFFF'
```

Generate a Bluetooth LE ATT PDU.

```
attPDU = bleATTPDU(cfgATT)
```

```
attPDU = 5x2 char array
    '04'
    '01'
    '00'
    'FF'
    'FF'
```

Decode the generated Bluetooth LE ATT PDU. The returned status indicates decoding is successful.

```
[status,cfgATTDecode] = bleATTPDUDecode(attPDU)
```

```
status =
  blePacketDecodeStatus enumeration

    Success


cfgATTDecode =
  bleATTPDUConfig with properties:

         Opcode: 'Information request'
    StartHandle: '0001'
      EndHandle: 'FFFF'
```

Generate and Decode Bluetooth LE LL Data Channel PDU

This example enables you to generate and decode a Bluetooth LE LL data channel PDU using the `bleLLDataChannelPDU` and `bleLLDataChannelPDUDecode` functions, respectively. Use the `bleLLDataChannelPDUConfig` configuration object to set the configuration parameters for Bluetooth LE LL data channel PDU generation.

Create a default Bluetooth LE LL data channel PDU configuration object for a data PDU. Set the link layer identifier to "`Data (start fragment/complete)`".

```
cfgLLData = bleLLDataChannelPDUConfig
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (continuation fragment/empty)'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'
```

```
cfgLLData.LLID = "Data (start fragment/complete)"
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (start fragment/complete)'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'
```

Set the CRC value.

```
cfgLLData.CRCInitialization = '123456';
```

Generate a Bluetooth LE LL data channel PDU by using the created configuration object and the upper-layer payload, "`030004000A0100`".

```
dataPDU = bleLLDataChannelPDU(cfgLLData,"030004000A0100")
```

```
dataPDU = 96×1

     0
     1
     0
     0
     0
     0
     0
     0
     1
     1
     ⋮
```

Decode the generated Bluetooth LE LL data channel PDU by initializing the cyclic redundancy check (CRC) value. The returned status indicates decoding is successful.

```
crcInit = '123456';
[status,cfgRx,llPayload] = bleLLDataChannelPDUDecode(dataPDU,crcInit)
```

```
status =
  blePacketDecodeStatus enumeration

    Success


cfgRx =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (start fragment/complete)'
                NESN: 0
       SequenceNumber: 0
             MoreData: 0
    CRCInitialization: '012345'


llPayload = 7x2 char array
    '03'
    '00'
    '04'
    '00'
    '0A'
    '01'
    '00'
```

Generate and Decode Bluetooth LE LL Control PDU

This example enables you to generate and decode a Bluetooth LE LL control PDU using the `bleLLDataChannelPDU` and `bleLLDataChannelPDUDecode` functions, respectively. Use the `bleLLControlPDUConfig` configuration object to set the configuration parameters for Bluetooth LE LL control PDU generation.

Create a default Bluetooth LE LL control PDU configuration object.

```
cfgControl = bleLLControlPDUConfig
```

```
cfgControl =
  bleLLControlPDUConfig with properties:

               Opcode: 'Connection update indication'
           WindowSize: 1
         WindowOffset: 0
    ConnectionInterval: 6
     PeripheralLatency: 0
     ConnectionTimeout: 10
              Instant: 0
```

Create a Bluetooth LE LL data channel PDU configuration object, specifying the control PDU configuration object.

```
cfgLLData = bleLLDataChannelPDUConfig(ControlConfig=cfgControl);
```

Specify the link layer identifier and CRC initialization values.

```
cfgLLData.LLID = "Control";
cfgLLData.CRCInitialization = "E23456"
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
   CRCInitialization: 'E23456'
       ControlConfig: [1x1 bleLLControlPDUConfig]
```

Generate a Bluetooth LE LL control PDU.

```
controlPDU = bleLLDataChannelPDU(cfgLLData)
```

```
controlPDU = 136×1

     1
     1
     0
     0
     0
     0
     0
     0
     0
     0
      ⋮
```

Decode the generated Bluetooth LE LL control PDU by initializing the CRC value. The returned status indicates decoding is successful.

```
crcInit = 'E23456';                                                          % Received duri
[status,cfgLLControl,llPayload] = bleLLDataChannelPDUDecode(controlPDU,crcInit)
```

```
status =
  blePacketDecodeStatus enumeration

    Success


cfgLLControl =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
   CRCInitialization: '012345'
       ControlConfig: [1x1 bleLLControlPDUConfig]
```

```
llPayload =

  1x0 empty char array
```

Generate and Decode Bluetooth LE LL Advertising Channel PDU

This example enables you to generate and decode a Bluetooth LE LL Advertising Channel PDU using the `bleLLAdvertisingChannelPDU` and `bleLLAdvertisingChannelPDUDecode` functions, respectively. Use the `bleLLAdvertisingChannelPDUConfig` configuration object to set the configuration parameters for Bluetooth LE LL Advertising Channel PDU generation.

Create a default Bluetooth LE LL Advertising Channel PDU configuration object. Specify the PDU type as "`Scan response`".

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig;
cfgLLAdv.PDUType = "Scan response";
```

Set the channel selection algorithm to "`Algorithm2`" and hop increment count to `10`.

```
cfgLLAdv.ChannelSelection = "Algorithm2";
cfgLLAdv.HopIncrement = 10
```

```
cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                 PDUType: 'Scan response'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
         ScanResponseData: [3x2 char]
```

Generate a Bluetooth LE LL Advertising Channel PDU.

```
advLLPDU = bleLLAdvertisingChannelPDU(cfgLLAdv)
```

```
advLLPDU = 112×1

     0
     0
     1
     0
     0
     1
     1
     0
     1
     0
     ⋮
```

Decode the generated Bluetooth LE LL Advertising Channel PDU. The returned status indicates decoding is successful.

```
[status,cfgLLAdvDecode] = bleLLAdvertisingChannelPDUDecode(advLLPDU)
```

```
status =
  blePacketDecodeStatus enumeration
```

```
    Success


cfgLLAdvDecode =
  bleLLAdvertisingChannelPDUConfig with properties:

                 PDUType: 'Scan response'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
         ScanResponseData: [3x2 char]
```

Generate and Decode Bluetooth LE GAP Data Block

This example enables you to generate and decode a Bluetooth LE GAP data block using the `bleGAPDataBlock` and `bleGAPDataBlockDecode` functions, respectively. Use the `bleGAPDataBlockConfig` configuration object to set the configuration parameters for Bluetooth LE GAP data block generation.

Create a Bluetooth LE GAP data block configuration object, specifying the advertising data type as "`Advertising interval`" and enabling simultaneous support for LE and BR/EDR modes at the Host.

```
cfgGAP = bleGAPDataBlockConfig(AdvertisingDataTypes="Advertising interval",LE="Host")
```

```
cfgGAP =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {'Advertising interval'}
     AdvertisingInterval: 32
```

Generate a Bluetooth LE GAP data block.

```
dataBlock = bleGAPDataBlock(cfgGAP)
```

```
dataBlock = 4x2 char array
    '03'
    '1A'
    '20'
    '00'
```

Decode the generated Bluetooth LE GAP AD block. The returned status indicates decoding is successful.

```
[status,cfgGAP] = bleGAPDataBlockDecode(dataBlock)
```

```
status =
  blePacketDecodeStatus enumeration

    Success


cfgGAP =
  bleGAPDataBlockConfig with properties:
```

```
    AdvertisingDataTypes: {'Advertising interval'}
     AdvertisingInterval: 32
```

Generate and Decode Bluetooth LE L2CAP Frame

This example enables you to generate and decode a Bluetooth LE L2CAP frame using the `bleL2CAPFrame` and `bleL2CAPFrameDecode` functions, respectively. Use the `bleL2CAPFrameConfig` configuration object to set the configuration parameters for Bluetooth LE L2CAP frame generation.

Create a default Bluetooth LE L2CAP frame configuration object, specifying the signaling command type as "`Credit Based Connection request`".

```
cfgL2CAP = bleL2CAPFrameConfig(CommandType="Credit based connection request")
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Credit based connection request'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0040'
                      LEPSM: '001F'
        MaxTransmissionUnit: 23
          MaxPDUPayloadSize: 23
                    Credits: 1
```

Generate a Bluetooth LE L2CAP frame.

```
L2CAPFrame = bleL2CAPFrame(cfgL2CAP)
```

```
L2CAPFrame = 18x2 char array
    '0E'
    '00'
    '05'
    '00'
    '14'
    '01'
    '0A'
    '00'
    '1F'
    '00'
    '40'
    '00'
    '17'
    '00'
    '17'
    '00'
    '01'
    '00'
```

Decode the generated Bluetooth LE L2CAP frame. The returned status indicates decoding is successful.

```
[status,cfgL2CAP,sdu] = bleL2CAPFrameDecode(L2CAPFrame)

status =
  blePacketDecodeStatus enumeration

    Success


cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Credit based connection request'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0040'
                      LEPSM: '001F'
        MaxTransmissionUnit: 23
          MaxPDUPayloadSize: 23
                    Credits: 1


sdu =

  1x0 empty char array
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed September 1, 2020. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

More About

- “Bluetooth Technology Overview” on page 8-2
- “Comparison of Bluetooth BR/EDR and Bluetooth LE Specifications” on page 8-6
- “Bluetooth Protocol Stack” on page 8-9
- “Bluetooth LE Link Layer Packet Generation and Decoding” on page 6-19
- “Bluetooth LE L2CAP Frame Generation and Decoding” on page 6-30
- “Modeling of Bluetooth LE Devices with Heart Rate Profile” on page 6-39

Parameterize Bluetooth LE Direction Finding Features

The Bluetooth Core Specification 5.1 [2] provided by the Bluetooth Special Interest Group (SIG) added direction finding features in Bluetooth low energy (LE) technology. The Bluetooth direction-finding capabilities, angle of arrival (AoA) and angle of departure (AoD), are introduced in the Bluetooth Core Specification 5.1 [2]. For more information about Bluetooth LE direction finding, see the "Bluetooth Location and Direction Finding" on page 8-18 topic and "Bluetooth LE Positioning by Using Direction Finding" on page 3-2 and "Bluetooth LE Direction Finding for Tracking Node Position" on page 3-15 examples.

The Bluetooth Toolbox enables you to configure and simulate these Bluetooth LE direction finding capabilities.

Set Simulation Parameters for Bluetooth LE Location and Direction Finding

Specify the number of Bluetooth LE locators and the dimensions in which the Bluetooth LE node position is to be determined. To estimate the 2-D or 3-D position of a Bluetooth LE node, specify at least two or three locators, respectively.

```
numDimensions = 2;
numLocators = 3;
```

Specify the bit energy-to-noise density ratio (Eb/No) range (in dB) and the number of iterations to simulate each Eb/No point.

```
EbNo = 6:2:16;
numIterations = 200;
```

Specify the direction finding method, the direction finding packet type, and the physical layer (PHY) transmission mode. The PHY transmission mode must be LE1M or LE2M for a connection-oriented constant tone extension (CTE) and LE1M for a connectionless CTE.

```
dfMethod = AoA;
dfPacketType = ConnectionCTE;
phyMode = LE1M;
```

Specify the antenna array parameters. The antenna array size must be a scalar or vector for 2-D or 3-D positioning, respectively. The scalar or vector array size represents a uniform linear array (ULA) or uniform rectangular array (URA), respectively. Specify the normalized element spacing between the antenna elements with respect to the wavelength. Specify the antenna switching pattern as a 1-by-M row vector, where M is in the range [2, $\frac{74}{\text{slotDuration}} + 1$].

```
arraySize = 16;
elementSpacing = 0.5;
switchingPattern = 1:prod(arraySize);
```

Specify the Bluetooth LE waveform generation parameters. The length of the CTE must be in microseconds, in the range [16, 160], with a step size of 8 microseconds.

```
slotDuration = 2;          % Slot duration in microseconds
cteLength = 160;
sps = 8;
chanIndex = 17;
crcInit = '555551';
accAddress = '01234567';
payloadLength = 1;         % Payload length in bytes
```

Create Bluetooth LE Angle Estimation Configuration Object

Create a default Bluetooth LE angle estimation configuration object by using the `bleAngleEstimateConfig` object. This object enables you to configure different parameters for Bluetooth LE angle estimation.

```
cfgAngle = bleAngleEstimateConfig
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

           ArraySize: 4
      ElementSpacing: 0.5000
        SlotDuration: 2
    SwitchingPattern: [1 2 3 4]
```

Specify a URA antenna design by setting the antenna array size of the configuration object to [4 4]. Set the row element spacing and column element spacing to 0.4 and 0.3, respectively. Specify the value of the antenna switching pattern.

```
cfgAngle.ArraySize = [4 4];
cfgAngle.ElementSpacing = [0.4 0.3];
cfgAngle.SwitchingPattern = 1:16
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

           ArraySize: [4 4]
      ElementSpacing: [0.4000 0.3000]
        SlotDuration: 2
    SwitchingPattern: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
```

Generate Random Positions for Bluetooth LE Locators

A Bluetooth LE locator represents a receiving device and a transmitting device in AoA and AoD calculation, respectively. To place a Bluetooth LE node at the origin and the locators randomly in the

2-D or 3-D space, use helperBLEGeneratePositions function. Specify the number of locators as 3 and the number of dimensions as 2. The function returns the 2-D position of the Bluetooth LE node at the origin, a matrix representing the position of the three locators, and the AoA or AoD (in degrees) between the Bluetooth LE node and the locators.

```
[nodePos,locatorPos,angle] = helperBLEGeneratePositions(3,2)
```

```
nodePos = 2×1

     0
     0


locatorPos = 2×3

  -23.7249  -57.5071  -12.5811
  -77.9415   69.9823   -1.7241


angle = 3×1

   73.0701
  -50.5887
    7.8032
```

Generate Bluetooth LE Direction Finding Packet

Set the simulation parameters to generate a Bluetooth LE direction finding packet.

```
dfPacketType = ConnectionCTE;
cteLength = 160;
dfMethod = AoA;
payloadLength = 1;                % Payload length in bytes
crcInit = '555551';
slotDuration = 2;                 % Slot duration in microseconds
```

Derive the type of CTE based on the slot duration and the direction finding method.

```
if strcmp(dfMethod,'AoA')
    cteType = [0;0];
else
    cteType = [0;1];
    if slotDuration == 1
        cteType = [1;0];
    end
end
```

To generate a direction finding packet corresponding to the type of CTE, use helperBLEGenerateDFPDU function.

```
dfPacket = helperBLEGenerateDFPDU(dfPacketType,cteLength,cteType,payloadLength,crcInit);
```

Perform Antenna Steering and Switching on Bluetooth LE Waveform

Set Bluetooth LE direction finding simulation parameters to perform antenna steering and switching on a Bluetooth LE waveform.

```
dfPacketType = ConnectionCTE;
cteLength = 160;
dfMethod = AoA;
payloadLength = 1;          % Payload length in bytes
crcInit = '555551';
slotDuration = 2;           % Slot duration in microseconds
phyMode = LE1M;
sps = 8;
chanIndex = 17;
```

Derive the type of CTE based on the slot duration and the direction finding method.

```
if strcmp(dfMethod,'AoA')
    cteType = [0;0];
else
    cteType = [0;1];
    if slotDuration == 1
        cteType = [1;0];
    end
end
```

Create a default Bluetooth LE angle estimation configuration object. Specify the antenna slot duration.

```
obj = bleAngleEstimateConfig;
obj.SlotDuration = slotDuration;
```

Generate a direction finding packet corresponding to the type of CTE by using the helperBLEGenerateDFPDU function.

```
dfPacket = helperBLEGenerateDFPDU(dfPacketType,cteLength,cteType,payloadLength,crcInit);
```

Using the direction finding packet, generate the Bluetooth LE waveform.

```
bleWaveform = bleWaveformGenerator(dfPacket,'Mode',phyMode,'SamplesPerSymbol',sps, ...
    'ChannelIndex',chanIndex,'DFPacketType',dfPacketType);
```

Use the helperBLESteerSwitchAntenna function to steer the Bluetooth LE waveform by 45 degrees in azimuth and 0 degrees in elevation and switch between the antennas according to the switching pattern.

```
dfWaveform = helperBLESteerSwitchAntenna(bleWaveform,45, ...
    phyMode,sps,dfPacketType,payloadLength,obj);
```

Decode Bluetooth LE Waveform with Connection-Oriented CTE

Set the simulation parameters to decode the Bluetooth LE waveform.

```
dfPacketType = ConnectionCTE;
cteLength = 160;
dfMethod = AoA;
payloadLength = 1;          % Payload length in bytes
crcInit = '555551';
slotDuration = 2;           % Slot duration in microseconds
phyMode = LE1M;
sps = 8;
chanIndex = 17;
```

Derive the type of CTE based on the slot duration and the direction finding method.

```
if strcmp(dfMethod,'AoA')
    cteType = [0;0];
else
    cteType = [0;1];
    if slotDuration == 1
        cteType = [1;0];
    end
end
```

Create a default Bluetooth LE angle estimation configuration object. Specify the antenna slot duration.

```
obj = bleAngleEstimateConfig;
obj.SlotDuration = slotDuration;
```

To generate a direction finding packet corresponding to the type of CTE, use the helperBLEGenerateDFPDU function.

```
dfPacket = helperBLEGenerateDFPDU(dfPacketType,cteLength,cteType,payloadLength,crcInit);
```

Generate the Bluetooth LE waveform by using the direction finding packet.

```
bleWaveform = bleWaveformGenerator(dfPacket, ...
    'Mode',phyMode, ...
    'SamplesPerSymbol',sps, ...
    'ChannelIndex',chanIndex, ...
    'DFPacketType',dfPacketType);
```

Get the in-phase and quadrature (IQ) samples by decoding the Bluetooth LE waveform.

```
[bits,accAddr,iqSamples] = bleIdealReceiver(bleWaveform, ...
    'Mode',phyMode, ...
    'SamplesPerSymbol',sps, ...
```

```
    'ChannelIndex',chanIndex, ...
    'DFPacketType',dfPacketType, ...
    'SlotDuration',slotDuration);
```

Estimate AoA of Bluetooth LE Waveform

Create a Bluetooth LE angle estimation configuration object, specifying the values of the antenna array size, slot duration, and antenna switching pattern.

```
cfgAngle = bleAngleEstimateConfig('ArraySize',2,'SlotDuration',2, ...
    'SwitchingPattern',[1 2]);
```

Estimate the AoA of the Bluetooth LE waveform by using the `bleAngleEstimate` function. The function accepts IQ samples and the Bluetooth LE angle estimation configuration object as inputs. You can either use the IQ samples obtained by decoding the Bluetooth LE waveform or use the IQ samples corresponding to the connection data channel protocol data unit (PDU).

Specify the IQ samples of a connection data PDU with an AoA CTE of 2 *µs* slots, CTE time of 16 *µs*, and azimuth rotation of 70 degrees.

```
IQsamples = [0.8507 + 0.5257i; -0.5257 + 0.8507i; -0.8507 - 0.5257i; ...
    0.5257 - 0.8507i; 0.8507 + 0.5257i; -0.5257 + 0.8507i; ...
    -0.8507 - 0.5257i; 0.5257 - 0.8507i; -0.3561 + 0.9345i];
```

Estimate the AoA of the Bluetooth LE waveform.

```
angle = bleAngleEstimate(IQsamples,cfgAngle)
```

```
angle = 70
```

Estimate Bluetooth LE Transmitter Position in 2-D Network Using Angulation

Set the positions of the Bluetooth LE receivers (locators).

```
rxPosition = [-18 -40;-10 70];          % In meters
```

Specify the azimuth angle of the signal between each Bluetooth LE receiver and transmitter.

```
azimuthAngles = [29.0546 -60.2551];     % In degrees
```

Specify the localization method. Because the angle of the signal between each Bluetooth LE receiver and transmitter is known, set the localization method to `'angulation'`.

```
localizationMethod = "angulation";
```

Estimate the position of the Bluetooth LE transmitter. The actual position of the Bluetooth LE transmitter is at the origin: (0, 0).

```
txPosition = blePositionEstimate(rxPosition,localizationMethod, ...
    azimuthAngles)
```

```
txPosition = 2×1
10^-4 ×
```

```
0.2374
0.1150
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed September 1, 2020. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.1. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions
`bleAngleEstimate` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects
`bleAngleEstimateConfig`

More About

- “Bluetooth Location and Direction Finding” on page 8-18
- “Bluetooth LE Direction Finding for Tracking Node Position” on page 3-15
- “Bluetooth LE Positioning by Using Direction Finding” on page 3-2
- “Estimate Bluetooth LE Node Position” on page 9-35

Create, Configure, and Visualize Bluetooth Mesh Network

Using this example, you can:

1. Create a Bluetooth® mesh network by configuring the nodes as source, destination, and relay.
2. Create and configure Bluetooth mesh profile.
3. Add On-Off application traffic between the mesh nodes.
4. Visualize Bluetooth mesh network.

Specify the total number of Bluetooth LE mesh nodes and their respective positions in the network. This example creates a 6-node Bluetooth mesh network consisting of a relay node, an end node, and two source-destination pairs. For more information about the functionalities of these nodes, see “Bluetooth Mesh Networking” on page 8-64.

```
totalNodes = 6;
meshNodesPositions = [15 25; 15 5; 30 15; 45 5; 45 25; 30 30];  % In meters
```

Specify the relay node and source-destination node pairs. In this example, Node1 and Node2 are source nodes and the corresponding destination nodes are Node4 and Node5.

```
relayNode = 3;
sourceDestinationNodePairs = [1 4; 2 5];
```

Set the Bluetooth mesh profile configuration parameters. Create Bluetooth mesh nodes with `"broadcaster-observer"` role.

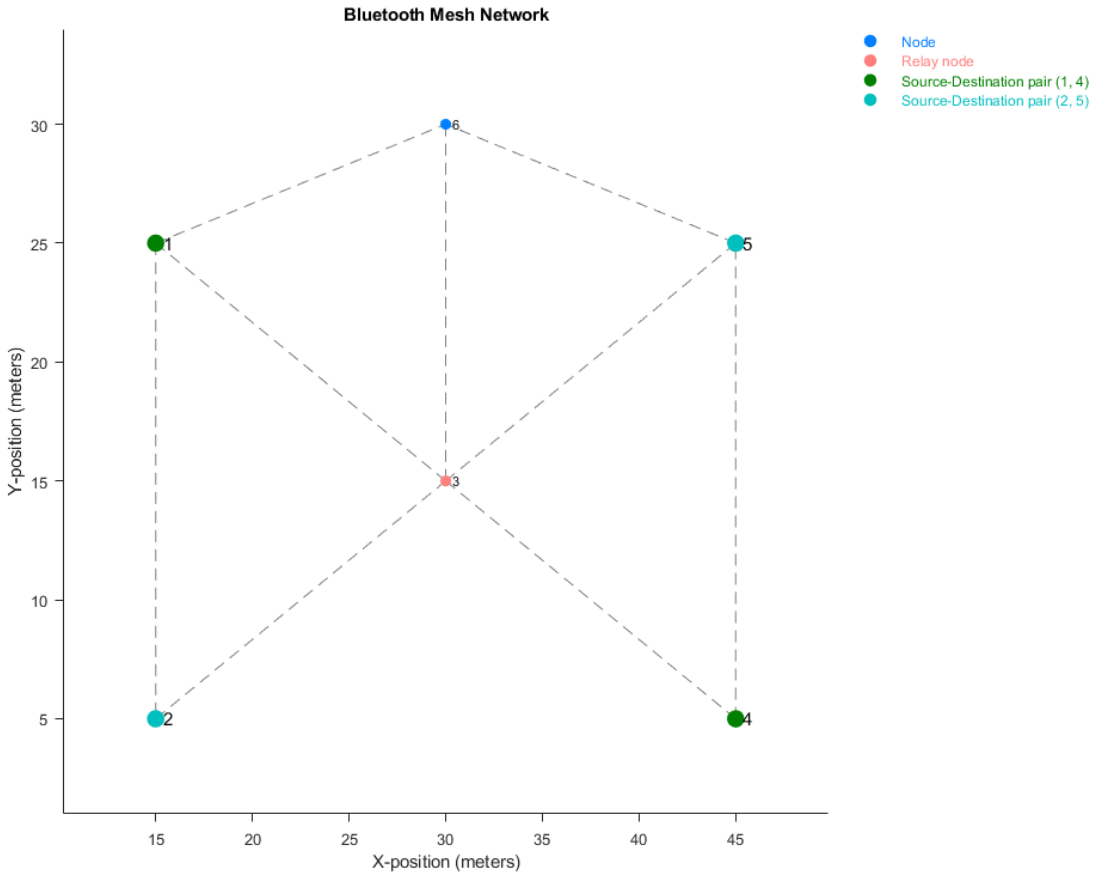
```
meshNodes = cell(1,totalNodes);
for nodeIdx = 1:totalNodes
    meshCfg = bluetoothMeshProfileConfig(ElementAddress=dec2hex(nodeIdx,4));
    if any(nodeIdx,relayNode)
        meshCfg.Relay = true;
    end
    meshNode = bluetoothLENode("broadcaster-observer", MeshConfig=meshCfg, ...
        Position=[meshNodesPositions(nodeIdx,:) 0],ReceiverRange=25, ...
        AdvertisingInterval=20e-3, ScanInterval=30e-3);
    meshNodes{nodeIdx} = meshNode;
end
```

Add traffic between Node1 - Node4 and Node2 - Node5 source-destination node pairs by using the `networkTrafficOnOff` object. The `networkTrafficOnOff` object enables you to create an On-Off application traffic pattern.

```
% Add traffic between Node1 and Node4
traffic = networkTrafficOnOff(DataRate=1,PacketSize=15,GeneratePacket=true);
addTrafficSource(meshNodes{1},traffic, ...
    SourceAddress=meshNodes{1}.MeshConfig.ElementAddress, ...
    DestinationAddress=meshNodes{4}.MeshConfig.ElementAddress, ...
    TTL=3);

% Add traffic between Node2 and Node5
traffic = networkTrafficOnOff(DataRate=1,PacketSize=10,GeneratePacket=true);
addTrafficSource(meshNodes{2},traffic, ...
    SourceAddress=meshNodes{2}.MeshConfig.ElementAddress, ...
    DestinationAddress=meshNodes{5}.MeshConfig.ElementAddress, ...
    TTL=3);
```

Visualize the Bluetooth mesh network by using the `helperBLEMeshVisualizeNetwork` helper function. This helper function displays the created Bluetooth mesh network.

```
meshNetworkGraph = helperBLEMeshVisualizeNetwork();                    % Object for Bluetooth me
meshNetworkGraph.NumberOfNodes = totalNodes;                           % Total number of mesh no
meshNetworkGraph.NodePositionType = 'UserInput';                       % Option to assign node p
meshNetworkGraph.Positions = meshNodesPositions;                       % List of all node positi
meshNetworkGraph.VicinityRange = 25;                                   % Transmission and recept
meshNetworkGraph.Title = 'Bluetooth Mesh Network';                     % Title of plot
meshNetworkGraph.SourceDestinationPairs = sourceDestinationNodePairs;  % Source-destination node
meshNetworkGraph.NodeType = [1 1 2 1 1 1];                             % State of mesh node
meshNetworkGraph.DisplayProgressBar = false;                           % Display progress bar
meshNetworkGraph.createNetwork();                                      % Display mesh network
```



References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 25, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Profile." Version 1.0.1. https://www.bluetooth.com/.

[4] Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Model Specification." Version 1.0.1. https://www.bluetooth.com/.

See Also

More About

- “Bluetooth Mesh Networking” on page 8-64
- “Create, Configure and Simulate Bluetooth Mesh Network” on page 9-45
- “Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network” on page 9-49
- “Bluetooth Mesh Flooding in Wireless Sensor Networks” on page 6-10
- “Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks” on page 6-2

Configure Bluetooth BR/EDR Channel with WLAN Interference and Pass the Waveform Through Channel

Using this example, you can:

1. Create and configure Bluetooth® basic rate/enhanced data rate (BR/EDR) channel.
2. Specify the source of WLAN interference as a baseband file or WLAN Toolbox™.
3. Generate WLAN interference and add it to the Bluetooth BR/EDR channel.
4. Generate Bluetooth BR/EDR waveform and pass it through the channel.

Create and Configure Bluetooth BR/EDR Channel with WLAN Interference

Configure a Bluetooth BR/EDR channel by using the `helperBluetoothChannel` object, which creates a Bluetooth BR/EDR channel model object with configurable properties.

```
bluetoothBREDRChannel = helperBluetoothChannel
```

```
bluetoothBREDRChannel =
  helperBluetoothChannel with properties:

    ChannelIndex: 0
            FSPL: 1
    NodePosition: [0 0 0]
            EbNo: 10
             SIR: 0
```

Set the ratio of energy per bit to noise power spectral density (Eb/No) for the additive white Gaussian noise (AWGN) channel to 22 dB. Specify signal-to-interference ratio (SIR) as -15 dB.

```
bluetoothBREDRChannel.EbNo = 22;
bluetoothBREDRChannel.SIR = -15;
```

Specify the source of WLAN interference by using the `wlanInterference` property. Use one of these options to specify the source of the WLAN interference.

- `'Generated'`: To add a WLAN (802.11b) signal (requires the WLAN Toolbox™ software), select this option.
- `'BasebandFile'`: To add a WLAN signal from a baseband file (`.bb`), select this option. You can specify the file name using the `wlanBBFilename` input argument. If you do not specify the `.bb` file, the example uses the default `.bb` file, `'WLANNonHTDSSS.bb'`, to add the WLAN signal.
- `'None'`: To disable WLAN interference, select this option.

Specify the source of WLAN interference as `'BasebandFile'`and specify a baseband file.

```
wlanInterference = BasebandFile;
wlanBBFilename =  'WLANNonHTDSSS.bb';
```

Generate the WLAN signal interference, by using the `helperBluetoothGenerateWLANWaveform` function. Add the generated WLAN signal interference to the Bluetooth BR/EDR channel.

```
if ~strcmpi(wlanInterference,'None')
    wlanWaveform = helperBluetoothGenerateWLANWaveform(wlanInterference,wlanBBFilename);
```

```
    addWLANWaveform(bluetoothBREDRChannel,wlanWaveform);
end
```

Generate and Pass Bluetooth BR/EDR Waveform Through the Channel

Create a Bluetooth BR/EDR signal structure, specifying different configurable properties of the waveform.

```
bluetoothSignal = struct(...
    'PacketType','DM1',...                  % Packet type
    'Waveform',[],...                       % Waveform
    'NumSamples',[],...                     % Number of samples
    'SampleRate',1e6,...                    % Sample rate
    'SamplesPerSymbol',8,...                % Samples per symbol
    'Payload',zeros(1,3200), ...            % Payload
    'PayloadLength',0, ...                  % Payload length
    'SourceID',0,...                        % Source identifier
    'Bandwidth',1,...                       % Bandwidth
    'NodePosition',[0 0 0],...              % Node position
    'CenterFrequency',2402,...              % Center frequency
    'StartTime',0,...                       % Waveform start time
    'EndTime',0,...                         % Waveform end time
    'Duration',0);...                       % Waveform duration
```

Create a Bluetooth BR/EDR waveform configuration object. Specify the packet type as HV1.

```
cfgWaveform = bluetoothWaveformConfig;
cfgWaveform.PacketType = 'HV1';
```

Create a bit vector containing concatenated payloads.

```
numBits = getPayloadLength(cfgWaveform)*8;              % Byte to bit conversion
message = randi([0 1],numBits,1);
```

Generate a Bluetooth BR/EDR waveform.

```
txWaveform = bluetoothWaveformGenerator(message,cfgWaveform);
```

Pass the generated waveform through the Bluetooth BR/EDR channel.

```
bluetoothSignal.Waveform = txWaveform;
bluetoothSignal.NumSamples = numel(txWaveform);
bluetoothSignal = run(bluetoothBREDRChannel,bluetoothSignal,cfgWaveform.Mode);
wirelessWaveform = bluetoothSignal.Waveform;
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 17, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] *P802.15.2/D09 - IEEE Draft Recommended Practice for Information Technology Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks Specific Requirements - Part 15.2: Coexistence of Wireless Personal Area Networks With Other Wireless Devices Operating in Unlicensed Frequency Bands.* LAN/MAN

Standards Committee, IEEE Computer Society, 2003, https://ieeexplore.ieee.org/document/4040972.

See Also

Functions
`bluetoothWaveformGenerator`

Objects
`bluetoothWaveformConfig`

More About

- “Bluetooth-WLAN Coexistence” on page 8-53
- “End-to-End Bluetooth BR/EDR PHY Simulation with WLAN Interference and Adaptive Frequency Hopping” on page 5-20

Create Bluetooth Piconet by Enabling ACL Traffic, SCO Traffic, and AFH

Bluetooth® Toolbox capabilities enable you to create and configure a Bluetooth piconet with basic rate (BR) physical layer (PHY) mode. The toolbox provides functionalities to configure asynchronous connection-oriented (ACL) link, synchronous connection-oriented (SCO) link, or both between the Central and Peripheral. You can also configure the frequency hopping techniques as basic frequency hopping or adaptive frequency hopping (AFH).

Configure the simulation parameters of the Bluetooth piconet by creating a structure. Specify the number of Peripherals in the piconet. A piconet can contain a maximum of seven Peripherals.

```
simulationParameters = struct;
simulationParameters.NumPeripherals = 2 ;
```

Calculate the total number of nodes in the piconet (one Central and multiple Peripherals).

```
numNodes = simulationParameters.NumPeripherals + 1;
```

Specify the type of logical link between the Central and Peripherals. Valid logical link values depend on how many Peripherals are connected to the Central.

- If the Central is connected to one Peripheral, you must specify the logical link value as a one-element vector of 1 (ACL link), 2 (SCO link), or 3 (ACL and SCO links).
- If the Central is connected to multiple Peripherals, you must specify the logical link value as an *n*-element row vector, where *n* is the number of Peripherals. Each element must be 1 (ACL link), 2 (SCO link), or 3 (ACL and SCO links).

To enable an ACL logical transport, set the logical link traffic to 1 or 3. You can specify the ACL packet type as `'DM1'`, `'DH1'`, `'DM3'`, `'DH3'`, `'DM5'`, or `'DH5'`. To enable an SCO logical transport, set the logical link traffic to 2 or 3. You can specify the SCO packet type as `'HV1'`, `'HV2'`, or `'HV3'` for the respective Peripheral that has SCO link traffic.

Enable ACL and SCO traffic, specifying the type of ACL and SCO packet as `'DM5'` and `'HV3'`, respectively.

```
simulationParameters.LinkTraffic = [1 2];
simulationParameters.ACLPacketType = DM5 ;
simulationParameters.SCOPacketType = {2,'HV3'};
```

To enable AFH, set the sequence type to Connection adaptive.

```
simulationParameters.SequenceType = Connection adaptive ;
```

Initialize a cell array to store the Bluetooth nodes.

```
btNodes = cell(1,numNodes);
```

Specify the distance (in meters) between two Bluetooth nodes.

```
interNodeDistance = 10;
```

Set the positions of the Bluetooth nodes.

```
simulationParameters.NodePositions = zeros(numNodes,3);
for nodeIdx = 1:numNodes
    simulationParameters.NodePositions(nodeIdx,:) = [nodeIdx*interNodeDistance 0 0];      % Set n
end
```

Set the node configuration parameters related to the wireless channel and channel classification.

```
simulationParameters.EbNo = 22;                              % Ratio of energy per bit (Eb) to spec
simulationParameters.WLANInterference = 'None';
simulationParameters.SIR = [-15 -16 -14 -13 -12 -11 -10];  % Signal to interference ratio in dB
simulationParameters.PERThreshold = 40;                      % Packet error rate
simulationParameters.ClassificationInterval = 3000;          % Classification interval in slots
simulationParameters.RxStatusCount = 10;                     % Status of maximum number of received
simulationParameters.MinRxCountToClassify = 4;               % Status of minimum number of received
simulationParameters.PreferredMinimumGoodChannels = 20;      % Preferred number of good channels re
simulationParameters.TxPower = 20;                           % Transmit power in dBm
simulationParameters.ReceiverRange = 40;                     % Bluetooth node receiver range in met
```

Create a Bluetooth piconet by using the `helperBluetoothCreatePiconet` helper function. This helper function creates a piconet supporting BR PHY mode.

```
helperBluetoothCreatePiconet(simulationParameters);
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 17, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

More About

- “Bluetooth Packet Structure” on page 8-27
- “Modeling of Bluetooth LE Devices with Heart Rate Profile” on page 6-39
- “Evaluate the Performance of Bluetooth QoS Traffic Scheduling with WLAN Signal Interference” on page 6-54

Generate Bluetooth LE Waveform and Add RF Impairments

Using this example, you can:

1. Generate a Bluetooth® low energy (LE) waveform.
2. Configure the parameters of radio frequency (RF) impairments.
3. Impair the generated Bluetooth LE waveform and plot the spectrum of the transmitted and impaired Bluetooth LE waveform.

Specify the data length. Create a message column vector of the specified data length containing random binary values.

```
dataLength = 2056;                              % In bits
message = randi([0 1],dataLength,1);
symbolRate = 1e6;
```

Specify the values of the physical layer (PHY) mode, channel index, samples per symbol, and access address.

```
phyMode = 'LE125K';
chanIdx = 2;
sps = 4;
accAdd = [1 1 1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 0 1 ...
    0 1 0 1 1 0 0].';
```

Generate the Bluetooth LE waveform.

```
txWaveform = bleWaveformGenerator(message,'Mode',phyMode,'SamplesPerSymbol',sps,'ChannelIndex',ch
```

Initialize the RF impairments for the specified PHY mode and samples per symbol by using the `helperBLEImpairmentsInit` function. The helper function returns a structure with phase frequency offset and variable fractional delay fields.

```
initRFImp = helperBLEImpairmentsInit(phyMode,sps)
```

```
initRFImp = struct with fields:
         pfo: [1x1 comm.PhaseFrequencyOffset]
    varDelay: [1x1 dsp.VariableFractionalDelay]
```

Specify the values of the frequency offset and phase offset.

```
initRFImp.pfo.FrequencyOffset = 150;          % In Hz
initRFImp.pfo.PhaseOffset = -1;               % In degrees
```

Specify the values of static timing offset, timing drift, variable timing offset, and DC offset.

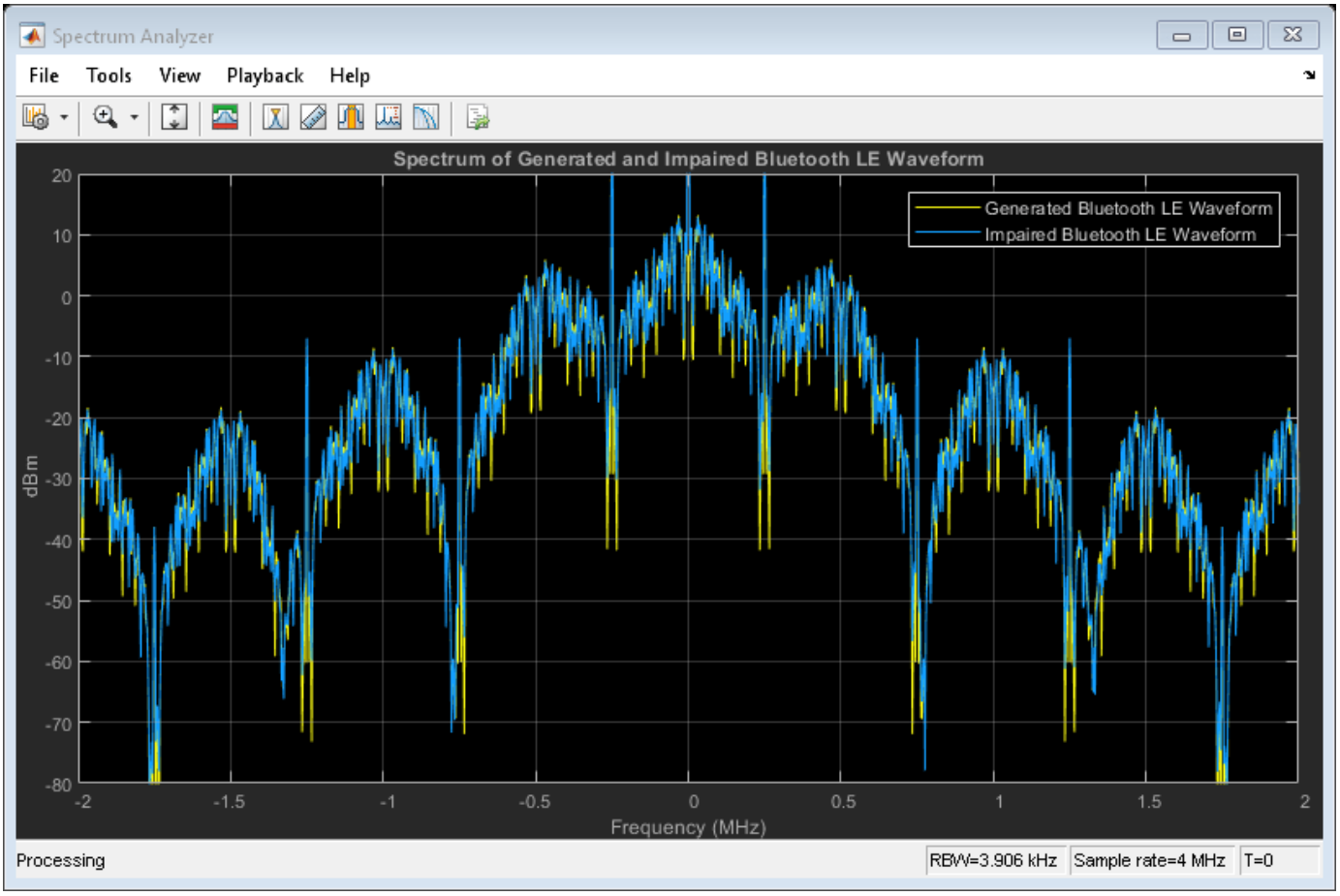
```
staticTimingOff = 0.15*sps;
timingDrift = 10;
initRFImp.vdelay = (staticTimingOff:timingDrift:staticTimingOff + timingDrift * (length(txWavefo
initRFImp.dc = 6 + 20;
```

Add RF impairments to the generated Bluetooth LE waveform by using the `helperBLEImpairmentsAddition` function.

```
txImpairedWaveform = helperBLEImpairmentsAddition(txWaveform,initRFImp);
```

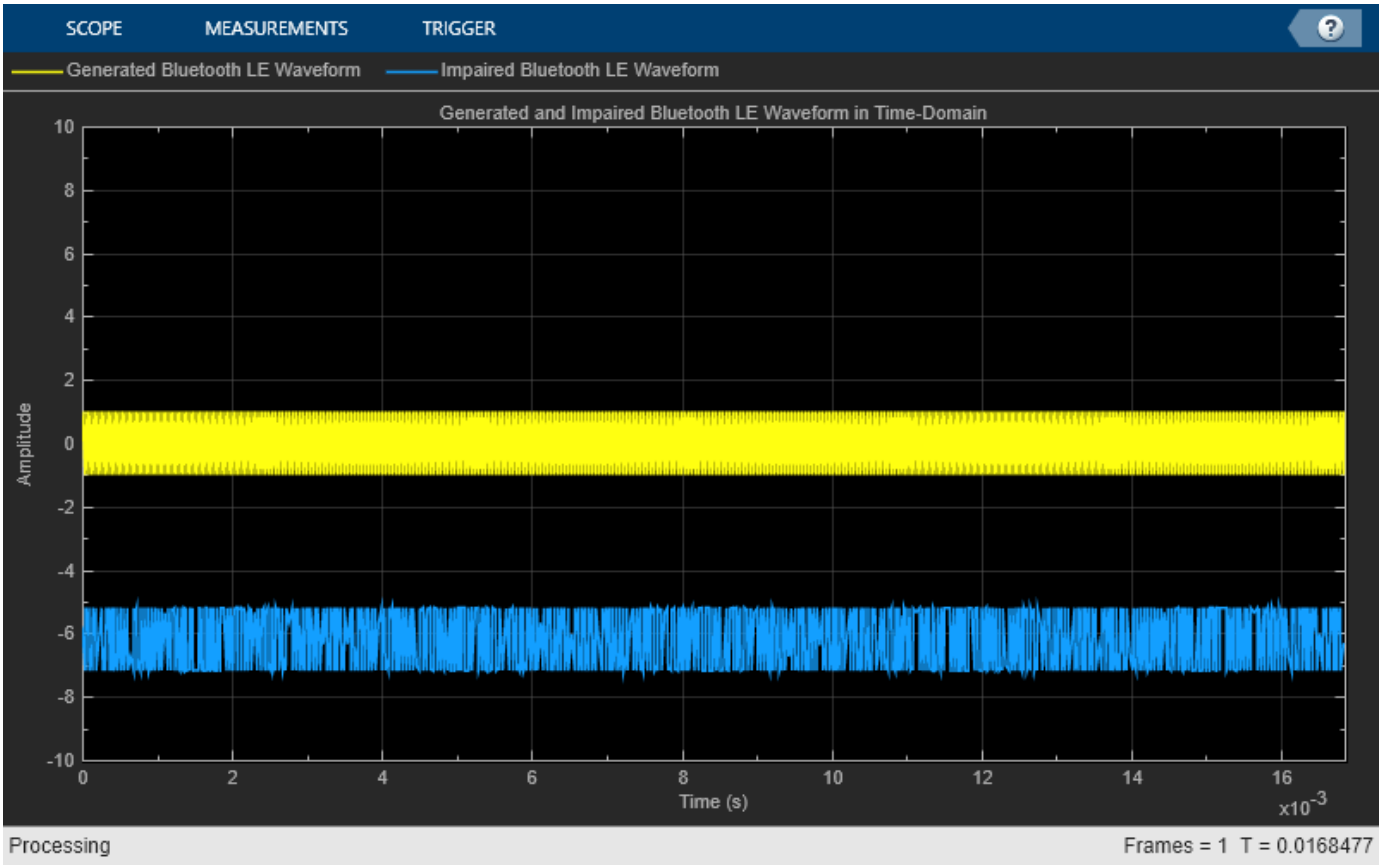
Create a default spectrum analyzer System object. Then, set the sample rate of the frequency spectrum. Visualize the generated and impaired Bluetooth LE waveform in the spectrum analyzer.

```
scope = dsp.SpectrumAnalyzer;
scope.SampleRate = sps*symbolRate;
scope.NumInputPorts = 2;
scope.Title = 'Spectrum of Generated and Impaired Bluetooth LE Waveform';
scope.ShowLegend = true;
scope.ChannelNames = {'Generated Bluetooth LE Waveform','Impaired Bluetooth LE Waveform'};
scope(txWaveform,txImpairedWaveform);
```



Visualize the generated and impaired Bluetooth LE waveform in time-domain.

```
timeScope = timescope('SampleRate',symbolRate*sps,'TimeSpanSource','Auto','ShowLegend',true);
timeScope.Title = 'Generated and Impaired Bluetooth LE Waveform in Time-Domain';
timeScope.ChannelNames = {'Generated Bluetooth LE Waveform','Impaired Bluetooth LE Waveform'};
timeScope(real(txWaveform),real(txImpairedWaveform));
```



References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 27, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions

`bleWaveformGenerator` | `bleIdealReceiver`

More About

- “End-to-End Bluetooth LE PHY Simulation Using Path Loss Model, RF Impairments, and AWGN” on page 5-2
- “End-to-End Bluetooth BR/EDR PHY Simulation with Path Loss, RF Impairments, and AWGN” on page 5-35
- “End-to-End Bluetooth LE PHY Simulation with AWGN, RF Impairments and Corrections” on page 5-42

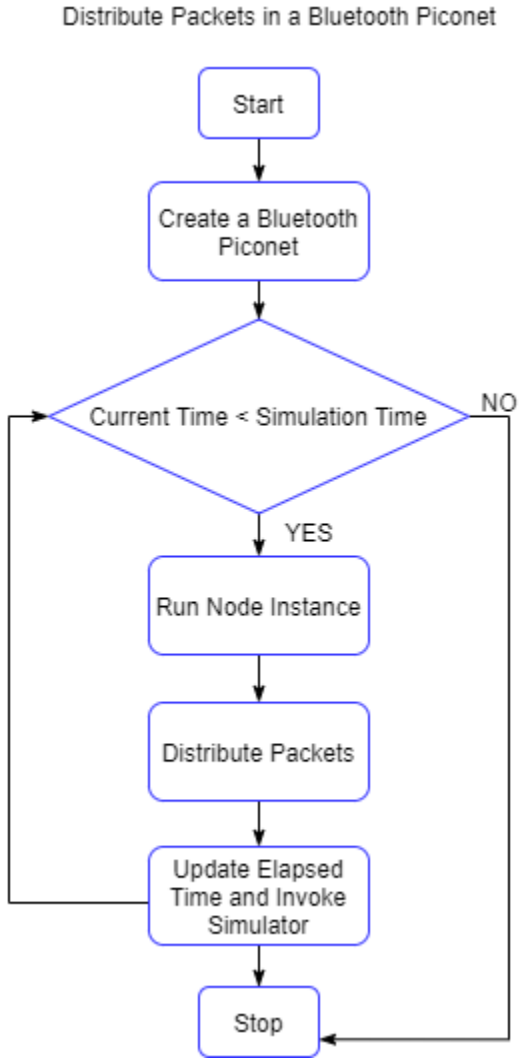
Packet Distribution in Bluetooth Piconet

Bluetooth Toolbox features enable you to distribute packets in a Bluetooth piconet using discrete time simulation (DTS). In DTS, you can call the node only if it has an operation to perform. To increase the speed of the simulation, the DTS implements these two core time values.

- Next invoke time — At this time, the simulator runs all of the node instances. This value is given by each node through discrete time operations such as sending data, receiving data, retransmissions, or transferring data from the higher layer to lower layer. The simulator is called at the time that is the minimum of the next invoke time values given by each node.
- Elapsed time — This value is the time elapsed between the last and current call of the simulator.

Packet Distribution

To distribute packets in a Bluetooth piconet using DTS, follow these steps.



Create Bluetooth Piconet

Create a Bluetooth piconet and configure the nodes as Central and Peripheral. For information about how to create a Bluetooth piconet, see “Create Bluetooth Piconet by Enabling ACL Traffic, SCO Traffic, and AFH” on page 9-23. In that example, a cell array, `btNodes`, is created representing the Bluetooth piconet. `btNodes` contains all of the nodes in the piconet with the complete stack enabled. The Bluetooth Toolbox features enables you to create a Bluetooth piconet with basic rate (BR) physical layer (PHY) mode.

Run Node Instance

Set the simulation time, current time, elapsed time, and next invoke time.

```
simulationTime = 2*1e6;                                % In microseconds
currentTime = 0;
nextInvokeTime = zeros(1,numel(btNodes));
```

Simulate the Bluetooth nodes by running node instance for each Bluetooth node.

```
while(curTime < simulationTimeInUs)
  for nodeIdx = 1:numel(btNodes)                                                % Simulate all the Blu
    nextInvokeTimes(nodeIdx) = runNode(btNodes{nodeIdx},elapsedTime);           % Run the Bluetooth no
  end
```

Distribute Packets

To distribute packets from each node to the receiving buffer of other nodes, use `helperBluetoothDistributePackets` function. This helper function accepts `btNodes` as an input and returns the transmission flag, `isPacketDistributed`, indicating whether the channel is free or not.

```
isPacketDistributed = helperBluetoothDistributePackets(btNodes);
```

Update Elapsed Time and Invoke Simulator

Based on the transmission flag, update the elapsed time. If no packets are to be distributed, update the elapsed time to the next event at a node.

```
  if isPacketDistributed
    elapsedTime = 0;
  else
    elapsedTime = min(nextInvokeTimes(nextInvokeTimes ~= -1));
  end
end
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 27, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

More About

- “Bluetooth Packet Structure” on page 8-27
- “Create Bluetooth Piconet by Enabling ACL Traffic, SCO Traffic, and AFH” on page 9-23

Generate and Attenuate Bluetooth BR/EDR Waveform in Industrial Environment

This example shows you how to generate a Bluetooth basic rate/enhanced data rate (BR/EDR) waveform in an industrial environment and attenuate the waveform by using a path loss model. The Bluetooth® Toolbox functionalities enable you to add free space, log distance, log normal shadowing, or two-ray ground reflection path loss models to the Bluetooth waveform.

Using this example, you can:

1. Generate a Bluetooth BR/EDR waveform.
2. Configure the parameters of a log distance path loss model.
3. Attenuate the Bluetooth BR/EDR waveform and plot the spectra of the transmitted and attenuated Bluetooth BR/EDR waveforms.

Create a default Bluetooth BR/EDR waveform configuration object.

```
BREDRcfg = bluetoothWaveformConfig();
```

Set the packet type, transmission mode, samples per symbol, and symbol rate.

```
BREDRcfg.PacketType = 'FHS';                % Packet type
BREDRcfg.Mode = 'BR';                       % Physical layer (PHY) transmission mode
BREDRcfg.SamplesPerSymbol = 8               % Samples per symbol
```

```
BREDRcfg =
  bluetoothWaveformConfig with properties:

                      Mode: 'BR'
                PacketType: 'FHS'
             DeviceAddress: '0123456789AB'
    LogicalTransportAddress: [3x1 double]
          HeaderControlBits: [3x1 double]
            ModulationIndex: 0.3200
           SamplesPerSymbol: 8
               WhitenStatus: 'On'
       WhitenInitialization: [7x1 double]
```

```
symbolRate = 1e6;                           % Symbol rate in Hz
```

Calculate the payload length (in bytes) for the Bluetooth BR/EDR configuration object by using the `getPayloadLength` object function. Then, calculate the payload length (in bits) by multiplying the calculated payload length in bytes by 8.

```
numBits = getPayloadLength(BREDRcfg)*8;     % Length of the payload in bits
```

Generate the payload for a single packet by using a random input bit vector.

```
txBits = randi([0 1],numBits,1);            % Generate data bits
```

Generate the Bluetooth BR/EDR waveform.

```
BREDRWaveform = bluetoothWaveformGenerator(txBits,BREDRcfg);
```

Set the environment to outdoor, industrial, home, or office. Specify the distance (in meters) between the transmitter and receiver.

```
environment = 'Industrial';                 % Environment
distance = 15;                              % Distance in meters
```

Set the frequency of the transmitted signal (in Hz) and the path loss exponent, which depends on the propagation environment. If you set the environment as outdoor, home, or office, you do not need to specify the path loss exponent value.

```
fc = 2.402*1e9;                             % Frequency of the signal
plExp = 5                                   ;           % Path loss exponent
```

Obtain the path loss value in linear scale by using the `helperBluetoothEstimatePathLoss` helper function. Attenuate the generated Bluetooth BR/EDR waveform by using this path loss value.

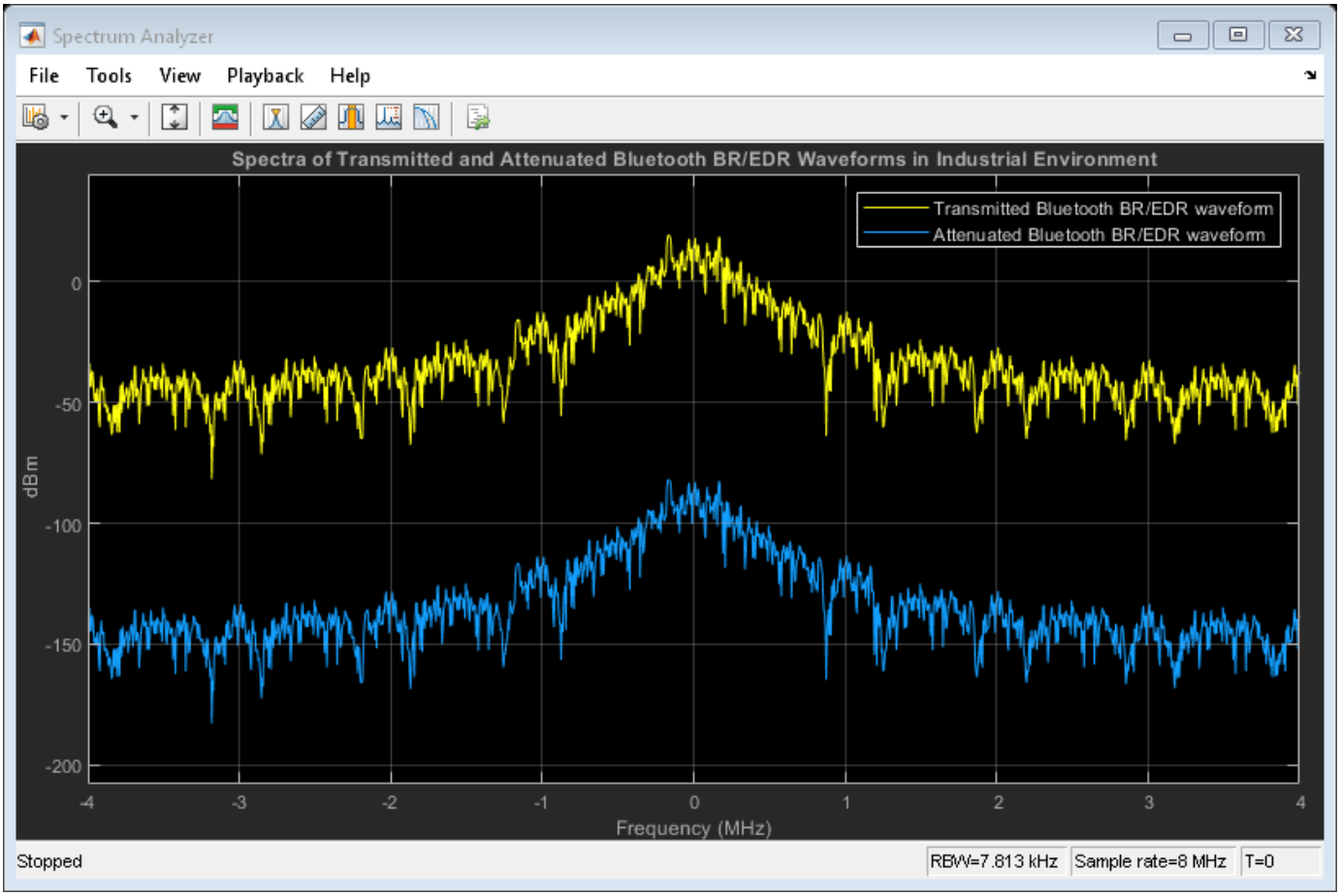
```
plLinear = helperBluetoothEstimatePathLoss(environment,distance,fc,plExp);
rxWaveform  = BREDRWaveform./plLinear;
```

Compute Bluetooth BR/EDR packet duration (in microseconds).

```
packetDuration = bluetoothPacketDuration(BREDRcfg.Mode,BREDRcfg.PacketType,numBits/8);
```

Plot the spectra of the transmitted and attenuated Bluetooth BR/EDR waveforms in an industrial environment.

```
specAnalyzer = dsp.SpectrumAnalyzer('NumInputPorts',2, ...
    'SampleRate',symbolRate*BREDRcfg.SamplesPerSymbol, ...
    'Title','Spectra of Transmitted and Attenuated Bluetooth BR/EDR Waveforms in Industrial Envi
    'ShowLegend',true, ...
    'ChannelNames',{'Transmitted Bluetooth BR/EDR waveform','Attenuated Bluetooth BR/EDR wavefor
specAnalyzer(BREDRWaveform(1:packetDuration*BREDRcfg.SamplesPerSymbol), ...
    rxWaveform(1:packetDuration*BREDRcfg.SamplesPerSymbol));
release(specAnalyzer);
```



References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 27, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions

`bluetoothWaveformGenerator` | `bluetoothIdealReceiver`

Objects

`bluetoothWaveformConfig`

More About

- “Bluetooth Packet Structure” on page 8-27

- “End-to-End Bluetooth BR/EDR PHY Simulation with Path Loss, RF Impairments, and AWGN” on page 5-35

Estimate Bluetooth LE Node Position

Bluetooth Toolbox features enable you to estimate the 2-D or 3-D position of a Bluetooth low energy (LE) node by using lateration, angulation, or distance-angle localization methods.

Estimate Bluetooth LE Receiver Position in 3-D Network Using Lateration

Set the positions of the Bluetooth LE transmitters (locators).

```
txPosition = [-5 -15 -30 -12.5;8.6603 -15 -17.3205 -21.6506; ...
    -17.3205 21.2132 20 43.3013];                                % In meters
```

Specify the distance between each Bluetooth LE transmitter and receiver.

```
distance = [13.2964 33.4221 40 55.0728];                         % In meters
```

Specify the localization method. Because the distance between each Bluetooth LE transmitter and receiver is known, set the localization method to `'lateration'`.

```
localizationMethod = "lateration";
```

Estimate the position of the Bluetooth LE receiver. The actual position of the Bluetooth LE receiver is [-7.5, 4.33, -5].

```
rxPosition = blePositionEstimate(txPosition,localizationMethod, ...
    distance)
```

```
rxPosition = 3×1

   -7.5001
    4.3304
   -4.9999
```

Estimate Bluetooth LE Transmitter Position in 2-D Network Using Angulation

Set the positions of the Bluetooth LE receivers (locators).

```
rxPosition = [-18 -40;-10 70];          % In meters
```

Specify the azimuth angle of the signal between each Bluetooth LE receiver and transmitter.

```
azimuthAngles = [29.0546 -60.2551];     % In degrees
```

Specify the localization method. Because the angle of the signal between each Bluetooth LE receiver and transmitter is known, set the localization method to `'angulation'`.

```
localizationMethod = "angulation";
```

Estimate the position of the Bluetooth LE transmitter. The actual position of the Bluetooth LE transmitter is at the origin: (0, 0).

```
txPosition = blePositionEstimate(rxPosition,localizationMethod, ...
    azimuthAngles)
```

```
txPosition = 2×1
10^-4 ×

    0.2374
    0.1150
```

Estimate Bluetooth LE Transmitter Position in 3-D Network Using Distance-Angle

Set the position of the Bluetooth LE receiver (locator).

```
rxPosition = [-10; 30; 40];          % In meters
```

Specify the distance between the Bluetooth LE receiver and transmitter.

```
distance = [23.2964];                % In meters
```

Specify the azimuth and elevation angles of the signal between the Bluetooth LE receiver and transmitter.

```
direction = [29.0546; -20.2551];     % In degrees
```

Specify the localization method. Because the distance and angle of the signal between the Bluetooth LE receiver and transmitter is known, set the localization method to "`distance-angle`".

```
localizationMethod = "distance-angle";
```

Estimate the position of the Bluetooth LE transmitter. If one receiver (locator) is present in the range of a Bluetooth LE transmitter, the "`distance-angle`" localization method estimates the location of the transmitter by using both of these measurements.

- Angle of arrival (AoA) or angle of departure (AoD) of a direction finding signal
- Received signal strength indicator (RSSI) and path loss

```
txPosition = blePositionEstimate(rxPosition,localizationMethod,distance,direction)
```

```
txPosition = 3×1

    9.1054
   40.6141
   31.9348
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions
`blePositionEstimate` | `bleAngleEstimate`

Objects
`bleAngleEstimateConfig`

More About

- “Bluetooth Location and Direction Finding” on page 8-18
- “Parameterize Bluetooth LE Direction Finding Features” on page 9-10
- “Bluetooth LE Direction Finding for Tracking Node Position” on page 3-15
- “Bluetooth LE Positioning by Using Direction Finding” on page 3-2

Create, Configure, and Simulate Bluetooth LE Network

This example shows you how to simulate a Bluetooth low energy (LE) network by using Bluetooth® Toolbox.

Using this example, you can:

1. Create and configure a Bluetooth LE piconet with Central and Peripheral nodes.
2. Create and configure a link layer (LL) connection between Central and Peripheral nodes.
3. Add application traffic from the Central to Peripheral nodes.
4. Simulate Bluetooth LE network and retrieve the statistics of the Central and Peripheral nodes.

Create a Bluetooth LE node, specifying the role as `"central"`. Specify the name and position of the node.

```
centralNode = bluetoothLENode("central");
centralNode.Name = "CentralNode";
centralNode.Position = [0 0 0];                 % In x-, y-, and z-coordinates, in meters
```

Create a Bluetooth LE node, specifying the role as `"peripheral"`. Specify the name and position of the node.

```
peripheralNode = bluetoothLENode("peripheral");
peripheralNode.Name = "PeripheralNode";
peripheralNode.Position = [10 0 0]              % In x-, y-, and z-coordinates, in meters
```

```
peripheralNode =
  bluetoothLENode with properties:

        TransmitterPower: 20
         TransmitterGain: 0
           ReceiverRange: 100
            ReceiverGain: 0
     ReceiverSensitivity: -100
             NoiseFigure: 0
    InterferenceFidelity: 0
                    Name: 'PeripheralNode'
                Position: [10 0 0]

   Read-only properties:
                    Role: 'peripheral'
        ConnectionConfig: [1x1 bluetoothLEConnectionConfig]
          TransmitBuffer: [1x1 struct]
                      ID: 4
```

Create a default Bluetooth LE configuration object to share the LL connection between the Central and Peripheral nodes.

```
cfgConnection = bluetoothLEConnectionConfig;
```

Specify the connection interval and connection offset. Throughout the simulation, the object establishes LL connection events for the duration of each connection interval. The connection offset is from the beginning of the connection interval.

```
cfgConnection.ConnectionInterval = 0.01; % In seconds
cfgConnection.ConnectionOffset = 0;      % In seconds
```

Specify the active communication period after the connection event is established between the Central and Peripheral nodes.

```
cfgConnection.ActivePeriod = 0.01 % In seconds
```

```
cfgConnection =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0100
         AccessAddress: '5DA44270'
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: '012345'
    SupervisionTimeout: 1
               PHYMode: 'LE1M'
         InstantOffset: 6
      ConnectionOffset: 0
          ActivePeriod: 0.0100
```

Configure the connection between Central and Peripheral nodes by using the `configureConnection` object function of the `bluetoothLEConnectionConfig` object.

```
configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Enable packet generation to generate an application packet with a payload.

```
traffic = networkTrafficOnOff(DataRate=100, ...
                              PacketSize=10, ...
                              GeneratePacket=true);
```

Add application traffic from the Central to the Peripheral node by using the `addTrafficSource` object function.

```
addTrafficSource(centralNode,traffic,"DestinationNode",peripheralNode.Name);
```

Create a Bluetooth LE network consisting of a Central and a Peripheral node.

```
nodes = {centralNode peripheralNode};
```

Initialize the Bluetooth LE network simulation by using the `helperWirelessNetwork` helper object. This helper object uses these object functions.

- `runNode`: Run the Bluetooth LE nodes.
- `pushChannelData` and `channelInvokeDecision`: Apply the channel impairments to the transmitted packet and distribute the impaired packet to the receiving buffers of the nodes that intends to receive the packet.

```
networkSimulator = helperWirelessNetwork(nodes);
```

Set the simulation time in seconds and run the simulation.

```
simulationTime = 0.5;
run(networkSimulator,simulationTime);
```

Retrieve application, link layer (LL), and physical layer (PHY) statistics corresponding to the broadcaster and receiver nodes. For more information about the statistics, see “Bluetooth LE Node Statistics” on page 8-78.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
    Name: 'CentralNode'
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
    Name: 'PeripheralNode'
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions
`addTrafficSource` | `statistics` | `configureConnection`

Objects
`bluetoothLENode` | `bluetoothLEConnectionConfig`

More About

- “Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network” on page 9-41
- “Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario” on page 9-53
- “Create, Configure and Simulate Bluetooth Mesh Network” on page 9-45
- “Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network” on page 9-49
- “Bluetooth LE Node Statistics” on page 8-78

Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network

This example shows you how to simulate a Bluetooth low energy (LE) isochronous broadcast audio network by using Bluetooth® Toolbox.

Using this example, you can:

1. Create and configure a Bluetooth LE piconet with an isochronous broadcaster and receivers.
2. Add application traffic at the broadcaster.
3. Simulate the broadcast isochronous network and retrieve the statistics of the broadcaster and receivers.

Create a Bluetooth LE node, specifying the role as `"isochronous-broadcaster"`. Specify the name and position of the node.

```
broadcasterNode = bluetoothLENode("isochronous-broadcaster");
broadcasterNode.Name = "Broadcaster";
broadcasterNode.Position = [0 0 0];                              % In x-, y-, and z-coordinates, in
```

Create two Bluetooth LE nodes, specifying the role as `"synchronized-receiver"`. Specify the name and position of the nodes.

```
receiverNode1 = bluetoothLENode("synchronized-receiver");
receiverNode1.Name = "Receiver1";
receiverNode1.Position = [10 0 0];
receiverNode2 = bluetoothLENode("synchronized-receiver");
receiverNode2.Name = "Receiver2";
receiverNode2.Position = [20 0 0]
```

```
receiverNode2 =
  bluetoothLENode with properties:

        TransmitterPower: 20
         TransmitterGain: 0
           ReceiverRange: 100
            ReceiverGain: 0
     ReceiverSensitivity: -100
             NoiseFigure: 0
    InterferenceFidelity: 0
                    Name: 'Receiver2'
                Position: [20 0 0]

   Read-only properties:
                    Role: 'synchronized-receiver'
               BIGConfig: [1x1 bluetoothLEBIGConfig]
          TransmitBuffer: [1x1 struct]
                      ID: 31
```

Create a default Bluetooth LE broadcast isochronous group (BIG) configuration object.

```
cfgBIG = bluetoothLEBIGConfig
```

```
cfgBIG =
  bluetoothLEBIGConfig with properties:
```

```
        SeedAccessAddress: '78E52493'
                  PHYMode: 'LE1M'
                   NumBIS: 1
              ISOInterval: 0.0050
               BISSpacing: 0.0022
              SubInterval: 0.0022
                   MaxPDU: 251
              BurstNumber: 1
    PretransmissionOffset: 0
          RepetitionCount: 1
             NumSubevents: 1
           BISArrangement: 'sequential'
                BIGOffset: 0
        ReceiveBISNumbers: 1
             UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... ]
            InstantOffset: 6
    BaseCRCInitialization: '1234'
```

Configure the broadcaster and receiver nodes so they use the default BIG parameters.

```
configureBIG(cfgBIG,broadcasterNode,receiverNode1);
configureBIG(cfgBIG,broadcasterNode,receiverNode2);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Enable packet generation to generate an application packet with a payload.

```
traffic = networkTrafficOnOff(DataRate=500, ...
                              PacketSize=10, ...
                              GeneratePacket=true);
```

Add application traffic at the broadcaster node by using the `addTrafficSource` object function.

```
addTrafficSource(broadcasterNode,traffic);
```

Create a broadcast isochronous network consisting of LE broadcast audio nodes.

```
nodes = {broadcasterNode receiverNode1 receiverNode2};
```

Initialize the broadcast isochronous network simulation by using the `helperWirelessNetwork` helper object. This helper object uses these object functions.

- `runNode`: Run the Bluetooth LE nodes.
- `pushChannelData` and `channelInvokeDecision`: Apply the channel impairments to the transmitted packet. Distribute the impaired packet to the receiving buffers of the nodes that intend to receive the packet.

```
networkSimulator = helperWirelessNetwork(nodes);
```

Set the simulation time in seconds and run the simulation.

```
simulationTime = 0.5;
run(networkSimulator,simulationTime);
```

Retrieve application, link layer (LL), and physical layer (PHY) statistics corresponding to the broadcaster and receiver nodes. For more information about the statistics, see “Bluetooth LE Node Statistics” on page 8-78.

```
broadcasterStats = statistics(broadcasterNode)
```

```
broadcasterStats = struct with fields:
    Name: 'Broadcaster'
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

```
receiver1Stats = statistics(receiverNode1)
```

```
receiver1Stats = struct with fields:
    Name: 'Receiver1'
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

```
receiver2Stats = statistics(receiverNode2)
```

```
receiver2Stats = struct with fields:
    Name: 'Receiver2'
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions
`addTrafficSource` | `statistics` | `configureBIG`

Objects
`bluetoothLENode` | `bluetoothLEBIGConfig`

More About

- “Bluetooth LE Audio” on page 8-41
- “Create, Configure, and Simulate Bluetooth LE Network” on page 9-38
- “Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario” on page 9-53
- “Estimate Packet Delivery Ratio of LE Broadcast Audio in Residential Scenario” on page 6-70

- “Bluetooth LE Node Statistics” on page 8-78

Create, Configure and Simulate Bluetooth Mesh Network

This example shows you how to simulate a Bluetooth mesh network by using Bluetooth® Toolbox.

Using this example, you can:

1. Create and configure a Bluetooth mesh network.
2. Enable or disable relay feature of the mesh node.
3. Add application traffic between the source and destination nodes.
4. Simulate Bluetooth mesh network and retrieve the statistics of mesh nodes.

Create a Bluetooth mesh profile configuration object, specifying the element address of the source node.

```
cfgMeshSource = bluetoothMeshProfileConfig(ElementAddress="0001")
```

```
cfgMeshSource =
  bluetoothMeshProfileConfig with properties:

             ElementAddress: '0001'
                      Relay: 0
                     Friend: 0
                   LowPower: 0
       NetworkTransmissions: 1
    NetworkTransmitInterval: 0.0100
                        TTL: 127
```

Create a Bluetooth LE node, specifying the role as `"broadcaster-observer"`. Specify the position of the source node. Assign the mesh profile configuration to the source node.

```
sourceNode = bluetoothLENode("broadcaster-observer");
sourceNode.Position = [0 0 0];
sourceNode.MeshConfig = cfgMeshSource;
```

Create a Bluetooth mesh profile configuration object, specifying the element address and enabling the relay feature of the Bluetooth LE node.

```
cfgMeshRelay = bluetoothMeshProfileConfig(ElementAddress="0002",Relay=true)
```

```
cfgMeshRelay =
  bluetoothMeshProfileConfig with properties:

             ElementAddress: '0002'
                      Relay: 1
                     Friend: 0
                   LowPower: 0
       NetworkTransmissions: 1
    NetworkTransmitInterval: 0.0100
                        TTL: 127
       RelayRetransmissions: 1
    RelayRetransmitInterval: 0.0100
```

Create a Bluetooth LE node, specifying the role as `"broadcaster-observer"`. Specify the position of the relay node. Assign the mesh profile configuration to the relay node.

```
relayNode = bluetoothLENode("broadcaster-observer");
relayNode.Position = [25 0 0];
relayNode.MeshConfig = cfgMeshRelay;
```

Create a Bluetooth mesh profile configuration object, specifying the element address of the Bluetooth LE node.

```
cfgMeshDestination = bluetoothMeshProfileConfig(ElementAddress="0003")
```

```
cfgMeshDestination =
  bluetoothMeshProfileConfig with properties:

            ElementAddress: '0003'
                     Relay: 0
                    Friend: 0
                  LowPower: 0
       NetworkTransmissions: 1
    NetworkTransmitInterval: 0.0100
                       TTL: 127
```

Create a Bluetooth LE node, specifying the role as `"broadcaster-observer"`. Specify the position of the destination node. Assign the mesh profile configuration to the destination node.

```
destinationNode = bluetoothLENode("broadcaster-observer");
destinationNode.Position = [50 0 0];
destinationNode.MeshConfig = cfgMeshDestination;
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the on time, data rate in kb/s, and packet size in bytes. Generate an application packet with a payload by enabling packet generation.

```
traffic = networkTrafficOnOff(OnTime=inf, ...
    DataRate=1, ...
    PacketSize=15, ...
    GeneratePacket=true);
```

Add application traffic between the source and destination nodes.

```
addTrafficSource(sourceNode,traffic, ...
    SourceAddress=cfgMeshSource.ElementAddress, ...
    DestinationAddress=cfgMeshDestination.ElementAddress,TTL=10);
```

Create a Bluetooth mesh network consisting of the source node, relay node, and destination node.

```
nodes = {sourceNode relayNode destinationNode};
```

Simulate the Bluetooth LE mesh network by using the `helperWirelessNetwork` helper object. This helper object uses these object functions.

- `runNode`: Run the Bluetooth LE nodes.
- `pushChannelData` and `channelInvokeDecision`: Apply the channel impairments to the transmitted packet and distributes the impaired packet to the receiving buffers of the nodes that intends receive the packet.

```
networkSimulator = helperWirelessNetwork(nodes);
```

Set the simulation time and run the simulation.

```
simulationTime = 1;                   % In seconds
run(networkSimulator,simulationTime);
```

Retrieve application, link layer (LL) , and physical layer (PHY) statistics related to the source, relay, and destination nodes by using the `statistics` object function. For more information about the statistics, see “Bluetooth LE Node Statistics” on page 8-78.

```
sourceStats = statistics(sourceNode)
```

```
sourceStats = struct with fields:
         Name: 'Node5'
          App: [1x1 struct]
    Transport: [1x1 struct]
      Network: [1x1 struct]
           LL: [1x1 struct]
          PHY: [1x1 struct]
```

```
relayStats = statistics(relayNode)
```

```
relayStats = struct with fields:
         Name: 'Node6'
          App: [1x1 struct]
    Transport: [1x1 struct]
      Network: [1x1 struct]
           LL: [1x1 struct]
          PHY: [1x1 struct]
```

```
destinationStats = statistics(destinationNode)
```

```
destinationStats = struct with fields:
         Name: 'Node7'
          App: [1x1 struct]
    Transport: [1x1 struct]
      Network: [1x1 struct]
           LL: [1x1 struct]
          PHY: [1x1 struct]
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions
`addTrafficSource` | `statistics`

Objects
`bluetoothLENode` | `bluetoothMeshProfileConfig`

More About

- “Create, Configure, and Simulate Bluetooth LE Network” on page 9-38
- “Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network” on page 9-41
- “Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario” on page 9-53
- “Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network” on page 9-49
- “Bluetooth LE Node Statistics” on page 8-78
- “Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks” on page 6-2
- “Bluetooth Mesh Flooding in Wireless Sensor Networks” on page 6-10

Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network

This example shows you how to simulate a Bluetooth mesh network by using Bluetooth® Toolbox.

Using this example, you can:

1. Create and configure a Bluetooth mesh network consisting of a source node, Friend node, a Low Power node (LPN), and a relay node.
2. Configure and establish friendship between the LPN and Friend node.
3. Add application traffic between the source node and LPN and relay the traffic by using the Friend node and relay node.
4. Simulate Bluetooth mesh network and retrieve the statistics of mesh nodes.

Create a default Bluetooth mesh profile configuration object, specifying the element address of the source node.

```
cfgMeshSource = bluetoothMeshProfileConfig(ElementAddress="0001");
```

Create a Bluetooth LE node, specifying the role as `"broadcaster-observer"`. Specify the position of the source node. Assign the mesh profile configuration to the source node.

```
sourceNode = bluetoothLENode("broadcaster-observer",AdvertisingInterval=30e-3,ScanInterval=30e-3)
sourceNode.Position = [0 0 0];
sourceNode.MeshConfig = cfgMeshSource;
```

Create a default Bluetooth mesh profile configuration object, specifying the element address and enabling the relay feature of the Bluetooth LE node.

```
cfgMeshRelay = bluetoothMeshProfileConfig(ElementAddress="0002",Relay=true);
```

Create a Bluetooth LE node, specifying the role as `"broadcaster-observer"`. Specify the position of the relay node. Assign the mesh profile configuration to the relay node.

```
relayNode = bluetoothLENode("broadcaster-observer",AdvertisingInterval=30e-3,ScanInterval=30e-3)
relayNode.Position = [25 0 0];
relayNode.MeshConfig = cfgMeshRelay;
```

Create a default Bluetooth mesh profile configuration object, specifying the element address and enabling the friend feature of the Bluetooth LE node.

```
cfgMeshFriend = bluetoothMeshProfileConfig(ElementAddress="0003",Friend=true);
```

Create a Bluetooth LE node, specifying the role as `"broadcaster-observer"`. Specify the position of the friend node. Assign the mesh profile configuration to the friend node.

```
friendNode = bluetoothLENode("broadcaster-observer",AdvertisingInterval=30e-3,ScanInterval=30e-3)
friendNode.Position = [50 0 0];
friendNode.MeshConfig = cfgMeshFriend;
```

Create a default Bluetooth mesh profile configuration object, specifying the element address of the Bluetooth LE node.

```
cfgMeshLPN = bluetoothMeshProfileConfig(ElementAddress="0004",LowPower=true);
```

Create a Bluetooth LE node, specifying the role as `"broadcaster-observer"`. Specify the position of the LPN node. Assign the mesh profile configuration to the LPN.

```
lowPowerNode = bluetoothLENode("broadcaster-observer",AdvertisingInterval=30e-3,ScanInterval=30e
lowPowerNode.Position = [75 0 0];
lowPowerNode.MeshConfig = cfgMeshLPN;
```

Create a default Bluetooth mesh friendship configuration object. Specify the poll timeout to terminate a friendship, the receive window supported by Friend node, and the receive delay requested by LPN, in seconds.

```
cfgFriendship = bluetoothMeshFriendshipConfig;
cfgFriendship.PollTimeout = 6;
cfgFriendship.ReceiveWindow = 180e-3;
cfgFriendship.ReceiveDelay = 50e-3;
```

Configure the connection between the friend node and LPN by using the `configureFriendship` object function of the `bluetoothMeshFriendshipConfig` object.

```
configureFriendship(cfgFriendship,friendNode,lowPowerNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the on time, data rate in kb/s, and packet size in bytes. Generate an application packet with a payload by enabling packet generation.

```
traffic = networkTrafficOnOff(OnTime=inf, ...
    DataRate=1, ...
    PacketSize=15, ...
    GeneratePacket=true);
```

Add application traffic between the source and destination node.

```
addTrafficSource(sourceNode,traffic, ...
    SourceAddress=cfgMeshSource.ElementAddress, ...
    DestinationAddress=cfgMeshLPN.ElementAddress,TTL=5);
```

Create a Bluetooth mesh network consisting of the source node, relay node, friend node and destination node.

```
nodes = {sourceNode relayNode friendNode lowPowerNode};
```

Simulate the Bluetooth LE mesh network by using the `helperWirelessNetwork` helper object. This helper object uses these object functions.

- `runNode`: Run the Bluetooth LE nodes.
- `pushChannelData` and `channelInvokeDecision`: Apply the channel impairments to the transmitted packet and distributes the impaired packet to the receiving buffers of the nodes that intends receive the packet.

```
networkSimulator = helperWirelessNetwork(nodes);
```

Set the simulation time and run the simulation.

```
simulationTime = 5;                   % In seconds
run(networkSimulator,simulationTime);
```

Retrieve application, link layer (LL) , and physical layer (PHY) statistics related to the source, relay, and destination nodes by using the `statistics` object function. For more information about the statistics, see “Bluetooth LE Node Statistics” on page 8-78.

```
sourceStats = statistics(sourceNode)
```

```
sourceStats = struct with fields:
         Name: 'Node46'
          App: [1x1 struct]
    Transport: [1x1 struct]
      Network: [1x1 struct]
           LL: [1x1 struct]
          PHY: [1x1 struct]
```

```
relayStats = statistics(relayNode)
```

```
relayStats = struct with fields:
         Name: 'Node47'
          App: [1x1 struct]
    Transport: [1x1 struct]
      Network: [1x1 struct]
           LL: [1x1 struct]
          PHY: [1x1 struct]
```

```
friendStats = statistics(friendNode)
```

```
friendStats = struct with fields:
         Name: 'Node48'
          App: [1x1 struct]
    Transport: [1x1 struct]
      Network: [1x1 struct]
           LL: [1x1 struct]
          PHY: [1x1 struct]
```

```
LPNStats = statistics(lowPowerNode)
```

```
LPNStats = struct with fields:
         Name: 'Node49'
          App: [1x1 struct]
    Transport: [1x1 struct]
      Network: [1x1 struct]
           LL: [1x1 struct]
          PHY: [1x1 struct]
```

Display the number of data messages transmitted by the Friend node.

```
fprintf("Number of data messages transmitted by the Friend node = %d\n",friendStats.Transport.Tra
```

```
Number of data messages transmitted by the Friend node = 2
```

Display the number of received data messages at the LPN.

```
fprintf("Number of data messages received by the LPN = %d\n",LPNStats.Transport.ReceivedDataMessa
```

```
Number of data messages received by the LPN = 2
```

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Functions
`addTrafficSource` | `statistics` | `configureFriendship`

Objects
`bluetoothLENode` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig`

More About

- “Create, Configure and Simulate Bluetooth Mesh Network” on page 9-45
- “Create, Configure, and Simulate Bluetooth LE Network” on page 9-38
- “Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network” on page 9-41
- “Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario” on page 9-53
- “Bluetooth LE Node Statistics” on page 8-78
- “Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks” on page 6-2
- “Bluetooth Mesh Flooding in Wireless Sensor Networks” on page 6-10

Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario

This example shows you how to visualize a Bluetooth low energy (LE) isochronous broadcast audio network in a residential scenario by using Bluetooth® Toolbox.

Using this example, you can:

1. Create and configure a Bluetooth LE broadcast audio residential scenario consisting of an isochronous broadcaster, synchronized receivers, and a WLAN node.
2. Create and configure the WLAN node without modeling the physical layer (PHY) and MAC behavior of the WLAN node.
3. Create the building triangulation from the scenario.
4. Visualize the 3-D residential scenario of the Bluetooth LE broadcast audio network.

Specify the size and layout of the residential building by using the `BuildingLayout` and `RoomSize` parameters. The `BuildingLayout` parameter specifies the number of rooms along the length, width, and height of the building. The `RoomSize` parameter specifies the size of each room in meters.

```
scenario = struct;
scenario.BuildingLayout = [3 3 2];
scenario.RoomSize = [10 8 6];        % In meters
```

Create a Bluetooth LE node, specifying the role as `isochronous-broadcaster`. Specify the position of the broadcaster.

```
broadcasterNode = bluetoothLENode("isochronous-broadcaster");
broadcasterNode.Position = [2 2 3];                          % x-, y-, and z-coordinates, in me
```

Specify the number of receivers and their respective positions.

```
numReceivers = 6;
receiverPositions = [7 6 9; ...
    6 4 4; ...
    25 18 6; ...
    18 12 10; ...
    5 20 6; ...
    12 15 3];                        % x-, y-, and z-coordinates, in meters
```

Create Bluetooth LE nodes with the role set as `synchronized-receiver`. Assign the receiver positions to the nodes.

```
receiverNodes = cell(1,numReceivers);
for rxIdx = 1:numReceivers
    rxNode = bluetoothLENode("synchronized-receiver");
    rxNode.Name = ['Receiver ' num2str(rxIdx)];
    rxNode.Position = receiverPositions(rxIdx,:);
    receiverNodes{rxIdx} = rxNode;
end
```

Create a WLAN node by using the `helperInterferingWLANNode` helper object. Specify the position of the WLAN node. This WLAN node does not model the PHY and MAC behavior of WLAN.

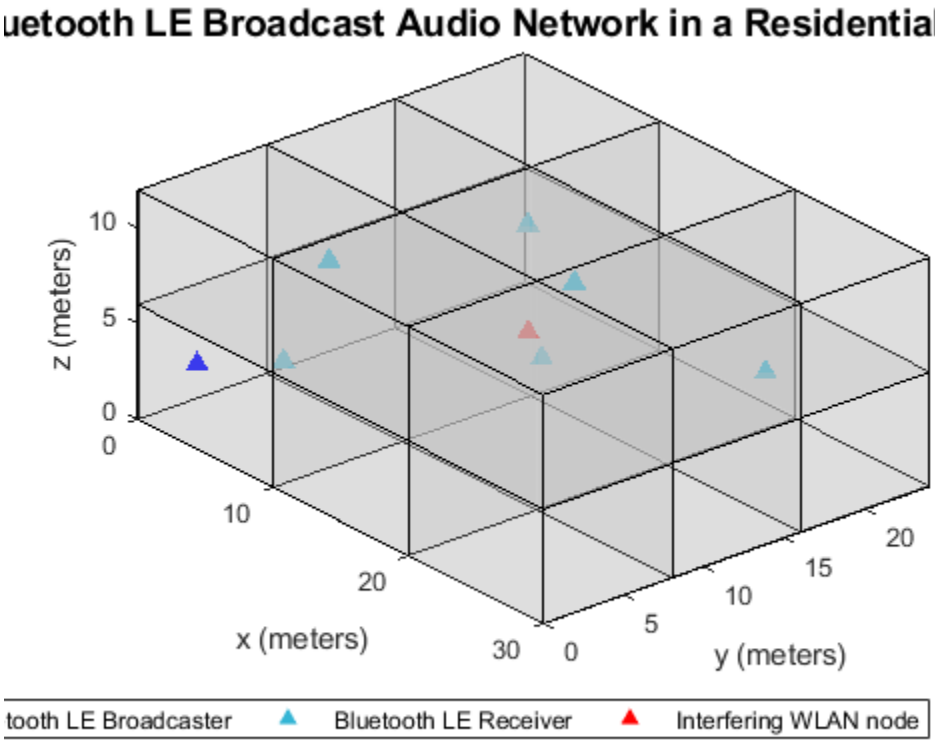
```
wlanNode = helperInterferingWLANNode;
wlanNode.Name = "WLAN node";
wlanNode.Position = [11 15 4];          % x-, y- and z- coordinates in meters
```

Create the building triangulation from the scenario parameters by using the `hTGaxResidentialTriangulation` helper function.

```
buildingTriangulation = hTGaxResidentialTriangulation(scenario);
```

Visualize the residential scenario in 3-D by using the `helperVisualizeResidentialScenario` helper function.

```
helperVisualizeResidentialScenario(buildingTriangulation,{broadcasterNode;receiverNodes;wlanNode
    ["Bluetooth LE Broadcaster","Bluetooth LE Receiver","Interfering WLAN node"],...
    "Bluetooth LE Broadcast Audio Network in a Residential scenario");
```



References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode`

More About

- “Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network” on page 9-41
- “Bluetooth LE Node Statistics” on page 8-78
- “Estimate Packet Delivery Ratio of LE Broadcast Audio in Residential Scenario” on page 6-70