Intra-Piconet Scheduling in Bluetooth®

Rachid Ait Yaiz

ISBN 90–365–2049–5
ISSN 1381–3617, CTIT Ph.D.-thesis series no. 04–62

Centre for Telematics and Information Technology,
P.O. Box 217, 7500 AE Enschede, The Netherlands

INTRA-PICONET SCHEDULING

IN BLUETOOTH®

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op donderdag 8 juli 2004 om 16.45 uur

door

Rachid Ait Yaiz

geboren op 9 maart 1974

te Amsterdam

Dit proefschrift is goedgekeurd door:

prof.dr.ir. B.R.H.M. Haverkort (promotor)
dr.ir. G.J. Heijenk (assistent promotor)
prof.dr.ir. I.G.M.M. Niemegeers (promotor)

Abstract

The trend of dynamically interconnecting the personal devices that people carry with them has led to the introduction of personal area networks (PANs) and personal networks (PNs). The Bluetooth® wireless access technology is believed to be a potential enabler of PANs and PNs. This dissertation focuses on Bluetooth intra-piconet scheduling (also referred to as Bluetooth polling) that helps in making the Bluetooth technology a successful enabler of PANs and PNs.

In order for the Bluetooth technology to be such a successful enabler, its polling mechanism should be efficient. At the same time, the polling mechanism should also be fair. Finally, the polling mechanism must be able to provide quality of service (QoS), which is needed to support audio and video applications. Conventional polling mechanisms are less suitable for Bluetooth as they do not take the Bluetooth specification into account. Current Bluetooth polling mechanisms are either not able to poll in a fair and efficient manner, or they do not provide the needed QoS.

In this thesis, a new polling mechanism, named Predictive Fair Poller (PFP), is developed. This polling mechanism predicts the availability of data for each slave, and it keeps track of fairness. Based on these two aspects, it decides which slave to poll next such that the efficiency and fairness are optimized.

Further, two new QoS-capable polling mechanisms are developed, namely the fixed-interval poller and the variable-interval poller. These pollers follow the IETF's Guaranteed Service approach, hence providing both a rate guarantee and a delay guarantee. With respect to Bluetooth polling, this is new. The fixed-interval poller plans polls to slaves with fixed intervals, whereas the variable-interval poller postpones polls for slaves, whenever possible, in order to save bandwidth. The fixed-interval poller and the variable-interval poller provide, with some predefined maximum deviation, a rate guarantee, which leads to a delay guarantee, provided that the traffic sources comply to their traffic flow specification. These two types of guarantees are the main QoS types that are needed for audio and video applications. Additionally, retransmission strategies are developed that minimize the influence of bad radio environments on the provisioning of these QoS types.

The mechanisms and techniques developed in this work are evaluated by means of simulation studies. These studies show that PFP is fair and efficient. In particular, the studies show that PFP performs at least as good as and sometimes better than existing Bluetooth polling mechanisms. Furthermore, the studies show that the variable-interval poller outperforms the fixed-interval poller, and that it is able to guarantee delay bounds that approach the delay bounds that can be guaranteed using a synchronous connection-oriented (SCO) channel. Moreover, the variable-interval poller is able to do so while consuming less resources. As the variable-interval poller can also perform retransmissions, this saved bandwidth can be used to avoid the link quality problems of SCO channels in bad radio environments, while keeping up QoS.

Acknowledgments

The road to the Ph.D. degree is heavy, long, and lonely, but more important, made bearable with the support of family, friends, and colleagues. Now that I arrived at this stage, I would like to thank them for their support.

I would like to start with my supervisors prof.dr.ir. Boudewijn Haverkort and prof.dr.ir. Ignas Niemegeers, and my daily supervisor dr.ir. Geert Heijenk. They helped me in realizing the dissertation that you are reading now. Besides them, I also thank dr.ir. Victor Nicola, dr.ir. Erik van Doorn, ir. Martin van der Zee, dr. Phil Chimento, and ir. Pierre Jansen for the valuable discussions I have had with them. Other colleagues that I would like to thank include Aiko, Alex, Assed, Bert-Jan, Carel, Djoerd, Enno, Ewout, Fred, Georgios, Hans, Helen, Hommad, Ing, Jaap, Jan, José, Kees, Lucia, Maarten, Malohat, Marc, Marcos, Maurice, Michael, Minh, Nikolay, Patrick, Pieter-Tjerk, Richard, Sonia, Val, and Vlora. Many of these colleagues allowed me to use their computer when I was desperately looking for more processing power for my simulation studies. I would also like to thank the people from Ericsson and WMC that contributed in one way or another to this work. These include Fredrik Alriksson, Arie Huijgen, Per Johansson, Ulf Jönsson, Johan Nielsen, Simon Oosthoek, and John de Waal.

Besides the support that I have received from my colleagues, I have also received a lot of support from my friends, which I would like to thank here. These friends include AbdelFettah, AbdelHafid, AbdelMajid, AbdelMonim, AbdelOuahab, Ahmed, Bedir, Hadi, Hamda, Hommad, Mehmet, Mohammed, Mouad, Omar, Osman, Rachid, Said, Salaheddine, Ziad, and all the members of the UT-Moslems community. This community really gave me the true and nice feeling that I am not alone.

Last but not least, I would like to thank my family, especially those in the Netherlands and in Morocco, for their support and caring during this heavy, long, and lonely road. My affection for them, as well as my gratitude to them cannot be expressed in a thousand words, so I will simply say: Thank you . . .

Rachid Ait Yaiz
July 8, 2004
Apeldoorn, The Netherlands

Contents

Chapter 1

Introduction

This dissertation is concerned with Bluetooth intra-piconet scheduling, also referred to as Bluetooth polling. The dissertation focuses on the development of Bluetooth polling mechanisms which help in making the Bluetooth technology a successful enabler of Personal Area Networks and Personal Networks.

This chapter introduces the dissertation by providing background information, motivating the work, presenting the main contributions, and describing the outline of the dissertation.

1.1 Background

The technological advances in electronics and telecommunications of the last few decades have led to the development of numerous personal devices that we carry with us. These personal devices are used for, for instance, improving our business, tightening our personal contacts, health-care, entertainment purposes, and many more. Examples of these devices are computers, personal digital assistants (PDAs), phones, cameras, medical devices, audio players, and game consoles.

The trend of dynamically networking these personal devices has led to the introduction of *personal area networks* (PANs), which are used to enhance our personal living environment. The concept of personal area networks is nowadays evolving to the newer concept of *personal networks* (PNs) [NHdG02]. These consist of a core personal area network extended on-demand with remote personal devices, remote foreign devices, and local foreign devices. This concept is illustrated in Figure 1.1, where person A is equipped with several devices that are connected through his PAN. His PAN is also extended with the network in his car, his network at home, his network at work, and the PAN of person B (e.g., his wife) forming a so-called personal network.

The concept of personal networks opens the door for new services and application areas not envisaged before. For instance, imagine a revalidating person equipped with sensors that monitor functions of his body. A personal network comprising these sensors, his mobile phone, and the remote health-care center's equipment will make it possible to keep track of the revalidation process of the person, without requiring him to stay in the revalidation center.

Another usage scenario of personal networks is the following. Suppose you are in a plane returning from a business trip. While in the plane, you compose several emails on your laptop. As your cellular phone must be turned off during the flight, these emails stay in the out-box of your email program as you cannot send them. Once leaving the plane, you turn on your cellular phone, which, after checking your laptop for pending outgoing emails, sends the emails you composed in the plane. Once you enter your car to drive back home, a friend of yours phones you through your cellular phone, and a live picture of him appears on a screen in your car. Meanwhile, your family at home knows that you are almost home as they are exactly aware of your car's current location. Your car makes its location known through your mobile phone.

One of the potential enablers of PANs and PNs is the Bluetooth wireless access technology [JKKG01]. Bluetooth [BT001][Haa00] is a technology that was initially developed as a replacement for cables between devices [Haa98]. However, it evolved to a technology that can be used to simultaneously network multiple devices.

Bluetooth nodes (devices) are either a master or a slave, and communication only takes place between the master and a slave, and never directly between two slaves or two masters. One master and up to seven slaves can be affiliated with each other and form a so-called piconet. On a time-division basis, a Bluetooth node can be a master in one piconet and/or a slave in one or more other piconets, making it possible to interconnect piconets forming a so-called scatternet.

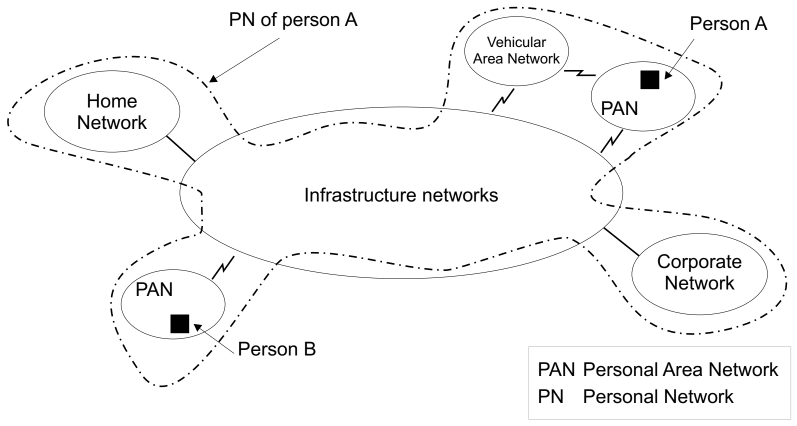


Figure 1.1: *Personal Network*

1.2 Motivation

In a Bluetooth piconet, bandwidth is divided among the slaves using a polling mechanism, which is also referred to as intra-piconet scheduling mechanism. This polling mechanism is applied by the master and is highly determining for the performance of the Bluetooth connections. In order for the Bluetooth technology to be a successful enabler for personal (area) networks, at least the following requirements must be met:

- Since the wireless resources are scarce, the bandwidth must be divided in an efficient manner (see Section 3.1.2.1). For instance, if a personal area network includes a Bluetooth-enabled computer and a Bluetooth printer, while no document is being printed, it is a waste of wireless resources to keep polling the printer.
- The bandwidth should be divided in a fair manner (see Section 3.1.2.3). For instance, if a personal area network includes a Bluetooth-enabled computer, a Bluetooth mouse,

and a Bluetooth printer, it would be efficient to poll only the printer. However, the computer will then not react to moving the mouse.

- The polling mechanism must be suitable for providing quality of service (QoS) (see Section 4.1.2). For instance, if a personal area network includes a Bluetooth-enabled computer, Bluetooth speakers, and a Bluetooth Internet access point, the audio you are listening to should not be disturbed by an Internet download session that you initiate.
- The QoS to be provided by the polling mechanism should be minimally influenced by non-ideal radio environments (see Section 5.1). For instance, if a personal area network includes a Bluetooth-enabled mobile phone and a Bluetooth headset, a phone conversation should be performed as good as possible in the presence of interference.

Conventional polling mechanisms (e.g. [Tak86, BLW91, Box91], and [BR87]) are less suitable for Bluetooth because of several reasons. Most important, they cannot be properly mapped on the Bluetooth system, where polling an empty queue results in a wasted time slot, where master-to-slave queues and slave-to-master queues are coupled, and where there is no knowledge about the availability of data at a slave.

A simple polling mechanism that can be used in Bluetooth is the 1-limited Round Robin poller. Using this polling mechanism, the slaves in a Bluetooth piconet are granted transmission time in a cyclic manner, independent of the need for transmission time. This polling mechanism is unable to fulfill the aforementioned requirements.

Advanced polling mechanisms have been defined (e.g., [JKJ99]) to (partly) fulfill the first two requirements, i.e., to be efficient, and to be fair. However, these polling mechanisms are unable to provide QoS. The Bluetooth standard defines a so-called SCO channel (see Chapter 2) in which slaves in a Bluetooth piconet are polled with fixed predefined intervals. This SCO channel can be used to provide QoS to some types of traffic flows (e.g., CBR traffic). In SCO channels, no retransmissions are performed, which make them vulnerable to bad radio environments. At the cost of lower bandwidth, SCO provides strong forward error correction schemes to cope with bad radio environments. Related work will be presented in the corresponding chapters.

The main goal of this thesis is the design of new Bluetooth polling mechanisms. First, we aim to design a polling mechanism that, in case of best effort traffic, divides bandwidth among the slaves in a fair and efficient manner. Second, we aim to design a polling mechanism that is able to provide quality of service. These polling mechanisms help in making the Bluetooth technology a successful enabler of personal area networks, and thus of personal networks.

1.3 Contributions

This dissertation provides a number of contributions, which are described in the following:

- *Efficient and fair scheduling of best effort traffic in Bluetooth*
  A new polling mechanism, named Predictive Fair Poller, has been designed that is able to divide, in an efficient and fair manner, bandwidth among the slaves in a Bluetooth

piconet. This polling mechanism keeps track of the fairness and the probability of data being available. Based on these aspects it decides which slave to poll next.

- *Scheduling QoS traffic in Bluetooth*
  Two polling mechanisms have been designed that are able to provide both delay guarantees and rate guarantees. These types of QoS support are necessary for, among others, audio and video applications in a Bluetooth piconet. The first polling mechanism plans polls for slaves with fixed intervals, whereas the second polling mechanism plans polls for slaves with variable intervals, as it postpones polls, whenever possible, in order to save as much as possible bandwidth. Furthermore, each planned poll has a deadline that must be met, i.e., a deadline before which the poll must be executed. The bandwidth saved by the second polling mechanism can be used to achieve a higher best effort throughput, or for performing retransmissions of lost packets.

- *Scheduling QoS traffic in Bluetooth in a non-ideal radio environment*
  Mechanisms and techniques have been defined to minimize the effect of bad radio environments on the QoS scheduling in Bluetooth. They are based on performing retransmissions of lost packets as soon as possible, without causing the aforementioned poll deadlines to be missed.

1.4 Organization of the dissertation

The remainder of this dissertation is organized as follows:

- *Chapter 2* presents background information about the Bluetooth technology and about Internet quality of service. Readers familiar with these concepts may skip this chapter.

- *Chapter 3* discusses the development of a polling mechanism that is able to divide, in an efficient and fair manner, bandwidth among the slaves in a Bluetooth piconet. For comparison, the stability, efficiency, and fairness of the 1-limited Round Robin poller are studied with analytical means. By means of simulation, the developed polling mechanism is evaluated and compared with some of the existing solutions.

- *Chapter 4* discusses the development of two polling mechanisms that are able to provide rate and delay guarantees. By means of simulations, these polling mechanisms are evaluated and compared with existing Bluetooth solutions to providing QoS.

- *Chapter 5* discusses the development of mechanisms that minimize the effect of bad radio environments on the polling mechanisms introduced in Chapter 4. By means of simulations, the resulting polling mechanisms are evaluated and compared with the existing Bluetooth solutions to providing QoS in a bad radio environment.

- *Chapter 6* concludes this dissertation by summarizing the achievements presented in this dissertation. Furthermore, it presents directions for further research.

Chapter 2

Preliminaries

This chapter presents background information on the Bluetooth technology and on Internet QoS, while focusing on parts relevant to this dissertation. The background information presented in this chapter is mainly extracted from [Haa98], [Haa00], and [BT001] in case of the Bluetooth technology, and from [XN99] and [SSG97] in case of Internet QoS. This chapter is organized as follows. Section 2.1 describes the Bluetooth technology, focusing on the baseband layer and the so-called L2CAP layer. Section 2.2 describes the Internet Engineering Task Force's (IETF) way of providing Integrated Services and Differentiated Services, focusing on the Guaranteed Service approach of providing Integrated Services.

2.1 The Bluetooth technology

2.1.1 Introduction

As the number of personal devices and their accessories persons carry with them grew, the number of accompanying cables and connectors also grew. Simultaneously, the need for a way to get rid of these cables and connectors emerged. Although many personal devices can use infrared links (IrDA) for wireless communications, the use of IrDA has some inherent disadvantages. Infrared links have a very limited range, require a line-of-sight, and do not support simultaneous connections between more than two devices.

In 1994, Ericsson Mobile Communication AB started a study to investigate the feasibility of a low-power, low-cost radio interface between mobile phones and accessories [Haa98]. Inherent to radio links, this radio interface would not have the disadvantages of infrared links. The study ultimately resulted in a widely supported new wireless access mechanism named Bluetooth.

Bluetooth is a wireless access technology that operates in the 2.4 GHz ISM (Industrial Scientific Medical) band. Bluetooth nodes are either a master or a slave, and communication only takes place between a master and a slave, and never directly between two slaves or two masters. One master and up to seven slaves can be affiliated with each other and form a so-called piconet. On a time-division basis, a Bluetooth node can be a master in one piconet and/or a slave in one or more other piconets, making it possible to interconnect piconets forming a so-called scatternet. Figure 2.1 shows an example of three piconets forming a scatternet. In Figure 2.1, node n1, node n9, and node n14 are the masters of piconet A, piconet B, and piconet C, respectively. On a time-division basis, node n5 is a slave in both piconet A and piconet C, and node n9 is the master of piconet B and a slave in piconet C. This makes it possible for all the nodes in piconet A, piconet B, and piconet C to communicate with each other. For instance, node n3 can communicate with node n12 through node n1, node n5, node n9, and node n14.

Figure 2.2 shows a usage scenario of Bluetooth. More specifically, it shows a Personal Area

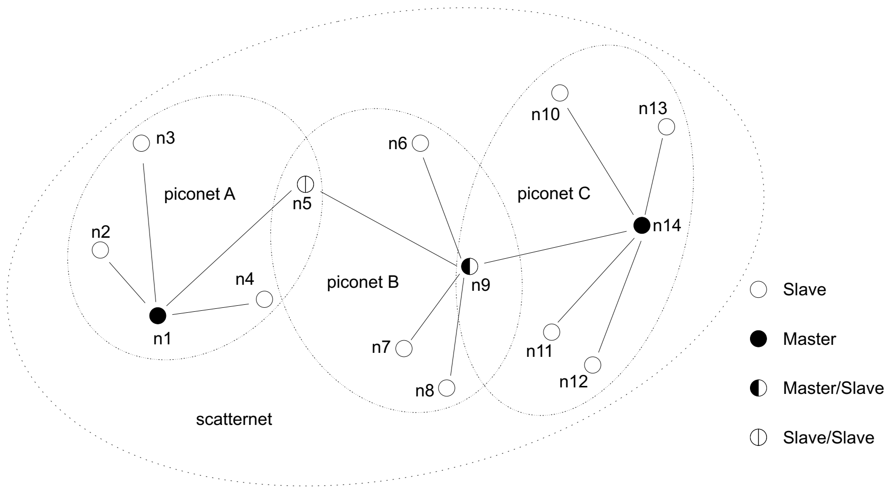


Figure 2.1: *Three piconets forming a scatternet*

Network that is created using a single piconet. This piconet is connected to an infrastructure network through the master, which serves here as an accesspoint to the infrastructure network. For instance, the accesspoint is a Bluetooth-enabled UMTS phone, which provides a wireless connection to an infrastructure network. The setup allows its user, for instance, to browse the Internet on the laptop and the PDA, to have a phone conversation through the headset, to play a game against a remote opponent, to inform a friend about his exact location, and to let the revalidation center keep track of his revalidation process. In this dissertation, we focus on scheduling in a single piconet, possibly connected to an infrastructure, as illustrated in Figure 2.2.

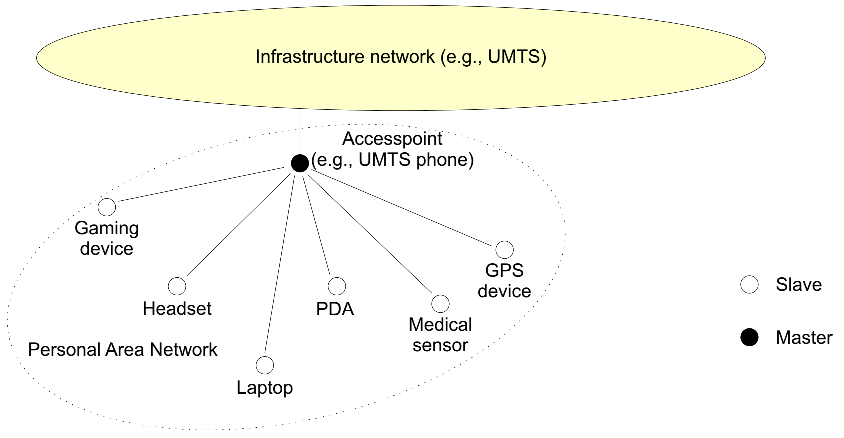


Figure 2.2: *Example of a Personal Area Network that consists of a single piconet connected to an infrastructure network.*

2.1.2 Bluetooth protocol stack

The protocol stack of Bluetooth is shown in Figure 2.3. The radio frequency (RF) layer is concerned with the actual transmission of data over the air interface. The baseband layer carries out the low-level link routines. Furthermore, it is concerned with low-level operations such as encryption, error coding, and retransmission schemes. The link manager (LM) layer is responsible for setup and management of baseband connections. It attaches and detaches slaves, it establishes so-called SCO links, and it handles the low power modes hold, sniff, and park. The logical link control and adaptation protocol (L2CAP) layer provides data services to higher-layer protocols. It is concerned with higher-level protocol multiplexing, packet segmentation and reassembly (SAR), and the exchange of quality of service (QoS) information. The L2CAP layer interfaces with various other protocol layers such as the so-called RFCOMM layer, the transmission convergence sublayer (TCS), the Service Discovery Protocol (SDP) layer, and other protocol layers, including the Internet Protocol (IP) layer. The RFCOMM protocol is used for serial port emulation on top L2CAP, which provides a packet-oriented channel. TCS provides a cordless telephony protocol. SDP is used to discover the services that are available as well as to determine the characteristics of these services. Finally, IP is used for connecting, for instance, personal digital assistants (PDAs) and laptops to the Internet. Note that Audio is transported using so-called SCO voice channels, which are defined in the baseband layer. However, packetized audio data, e.g. IP telephony, may be transported using L2CAP channels.

The following sections describe, in more detail, the protocol layers relevant to this dissertation.

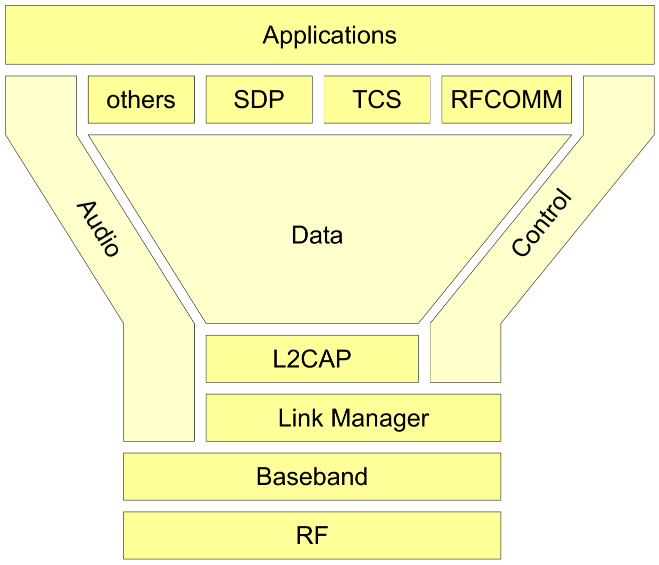


Figure 2.3: *Bluetooth protocol stack [Haa00]*

2.1.3 Radio Frequency (RF) Layer

Bluetooth operates in the 2.4 GHz ISM (Industrial Scientific Medical) band. In most countries around the world the range of this frequency band is 2400 - 2483.5 MHz. In some countries a limited frequency range is used due to national limitations. For instance, the frequency range in France is 2446.5 - 2483.5 MHz. The radio uses frequency hopping to spread the energy across the ISM band in 79 (23 in France) RF channels of 1 MHz.

Bluetooth uses Gaussian Frequency Shift Keying (GFSK) with a modulation index between 0.28 and 0.35. A logical one is represented by a positive frequency deviation, and a logical zero is represented by a negative frequency deviation. The Bluetooth symbol rate is 1 Ms/s, which results in a raw bit rate of 1 Mbps.

2.1.4 Baseband

As mentioned before, Bluetooth nodes are either a master or a slave, and communication only takes place between the master and a slave, and never directly between two slaves or two masters. One master and up to seven slaves can be affiliated with each other and form a so-called piconet. On a time division basis, a Bluetooth node can be a master in one piconet and/or a slave in one or more other piconets, making it possible to interconnect piconets forming a so-called scatternet (see Figure 2.1).

The Bluetooth channel is represented by a pseudo-random[1] hopping sequence that hops through 79 (or 23) RF frequencies at a nominal hop rate of 1600 hops/s. The hopping sequence in a piconet is determined based on the Bluetooth device address and clock of the master of that piconet, while the participants in the piconet are time- and hop-synchronized to the channel. As the Bluetooth device address and clock of the master determine the hopping sequence, a Bluetooth node can be master in only one piconet.

The channel is divided into 1600 time slots per second. Time slots are either downlink slots, i.e., from the master to a slave, or uplink slots, i.e., from the addressed slave to the master. Data is exchanged between the master and a slave using baseband packets that cover one, three or five time slots, while the whole baseband packet is transmitted using the RF hop frequency of its first slot.

Bluetooth supports two types of links between a master and a slave: a *Synchronous Connection-Oriented (SCO)* link and an *Asynchronous Connection-Less (ACL)* link. Baseband packets sent over an SCO link (SCO packets) always cover one time slot while baseband packets sent over an ACL link can cover one, three, or five time slots. In case of an SCO link between the master and a slave, the master polls that slave at regular intervals. The addressed slave can then respond with an SCO packet. In case of an ACL link, polling can be done in many different ways. The difference between the polling mechanisms is related to the order which slaves are polled in and to the service discipline used to serve a slave. Figure 2.4 shows an example of polling slaves, in the absence of SCO channels, in a 1-limited Round Robin manner. The master implicitly polls the slaves by sending data to them. In case the master has no data for

[1]A pseudo-random sequence of numbers is a sequence of numbers that appears to be random, but which is determined by a random number generator.

the slave to be polled, the master explicitly polls that slave by sending a POLL packet, which is a packet with no payload. When polled, a slave responds with a single packet. In case the polled slave has no data to be transmitted, it responds with a NULL packet, which is a packet with no payload.

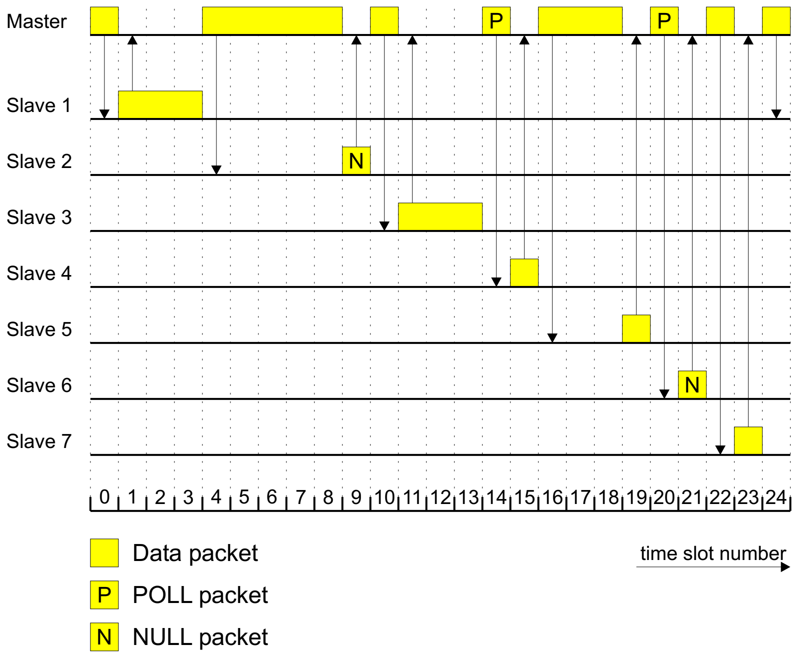


Figure 2.4: *Polling in Bluetooth*

2.1.4.1 Baseband Packets

The general Baseband packet format is shown in Figure 2.5. Generally, a baseband packet consists of an access code, a header, and a payload field. The size of the access code is 72 bits if a packet header follows and 68 bits otherwise. The access code identifies all baseband packets exchanged on the channel of the piconet. The size of the baseband packet header is 54 bits, and the size of the payload ranges between 0 and 2745 bits.

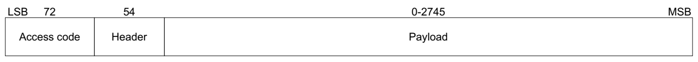


Figure 2.5: *General packet format [BT001]*

Three types of access codes are defined in the standard and are used in different operating modes: the Channel Access Code (CAC), the Device Access Code (DAC) and the Inquiry Access Code (IAC). The CAC is included in all packets exchanged on the piconet channel and identifies the piconet. The DAC is used for, among others, paging and response to paging. Finally, two variations of the IAC are defined: General Inquiry Access Code (GIAC) and Dedicated Inquiry Access Code (DIAC). The GIAC can be used to discover which other Bluetooth nodes are in range, whereas the DIAC can be used for the same purpose, though being restricted to a dedicated group of Bluetooth nodes that share a common characteristic. As opposed to the CAC, the DAC and IAC do not always include a header. In that case the size of the DAC and IAC is 68 bits.

As can be seen in Figure 2.6, the access code consists of a preamble, a sync word, and a trailer. The preamble is a fixed zero-one pattern and is used for DC compensation. The sequence is 1010 if the first bit of the syncword is one, and is 0101 otherwise. The sync word for the CAC is a 64-bit code word that is derived from the 24-bit lower address part (LAP) of the unique 48-bit Bluetooth device address (BD_ADDR) of the master. Reserved dedicated LAPs are used for the sync word for the GIAC and the DIAC, while the slave's LAP is used for the sync word of the DAC. Similar to the preamble, the trailer is a fixed zero-one pattern that is 1010 if the last bit of the sync word is zero and is 0101 otherwise. Note that the trailer is only appended if a packet header follows the access code.

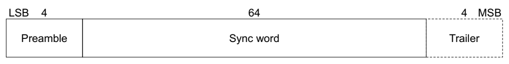


Figure 2.6: *Access code format [BT001]*

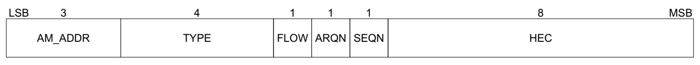


Figure 2.7: *Header format [BT001]*

The packet header consists of 18 information bits and is 1/3 FEC encoded[2], which results in a 54 bits header. The header (before FEC encoding) is shown in Figure 2.7 and consists of:

- a 3-bit active member address (AM_ADDR), which is assigned to each slave that becomes member of a piconet. Packets exchanged between the master and a particular slave include the active member address of that slave. As soon as a slave is disconnected or parked (see Section 2.1.4.3) it gives up its assigned active member address. Broadcasting packets and so-called FHS packets include an all-zero active member address.
- 4-bit type code (TYPE), which identifies the baseband packet type. Its interpretation depends on the physical link type (ACL or SCO). Note that the packet type also determines the number of slots covered by the packet.
- 1-bit flow control (FLOW), which is used for flow control of packets over the ACL link.

[2]The rate 1/3 FEC code is a simple 3-times repetition FEC, whereas the rate 2/3 FEC is a (15,10) shortened Hamming code.

- 1-bit acknowledgment indication (ARQN), which is used to acknowledge the correct receipt of payload data with cyclic redundancy check (CRC).
- 1-bit sequence number (SEQN), which is used to filter out retransmissions.
- 8-bit header error check (HEC), which is used to check the integrity of the header.

Three groups of packet types can be distinguished. A group of common packet types that are common to ACL links and SCO links, a group of packet types for ACL links and a group of packet types for SCO links.

Five packet types are common to ACL links and SCO links and are:

- ID packet, which consists of the DAC or the IAC and which is used in paging and inquiry procedures.
- NULL packet, which is a packet consisting of only the CAC and a packet header and which is used to return acknowledgment and flow control information in the absence of data to be transmitted.
- POLL packet, which is similar to the NULL packet, except that the POLL packet is not part of the ARQ scheme and that it does not affect the ARQN and the SEQN fields. The POLL packet must be confirmed by the recipient by responding with a packet even if no data is available.
- FHS packet, which is a control packet that is, among others, used for making known the Bluetooth device address and the clock of the sender.
- DM1 packet, which is used to support link messages in both an ACL link and an SCO link. Its details will be presented when discussing the ACL packet types.

Seven packet types are defined for use in an ACL link and are:

- DM1, DM3 and DM5, which are the Data - Medium rate packets. DM1 may cover up to one slot, DM3 may cover up to three slots, and DM5 may cover up to 5 slots. Besides a 16-bit CRC code, the payload contains up to 18 information bytes, 123 information bytes, and 226 information bytes for DM1, DM3, and DM5 respectively. In the information bytes, DM1 includes a 1-byte payload header, whereas DM3 and DM5 include a 2-byte payload header. The information bytes plus CRC code are coded with a rate 2/3 FEC, which adds five parity bits to each 10-bit segment.
- DH1, DH3 and DH5, which are the Data - High rate packets. DH1 may cover up to one slot, DH3 may cover up to three slots, and DH5 may cover up to 5 slots. Besides a 16-bit CRC code, the payload contains up to 28 information bytes, 185 information bytes, and 341 information bytes for DH1, DH3, and DH5 respectively. In the information bytes, DH1 includes a 1-byte payload header, whereas DH3 and DH5 include a 2-byte payload header. The information bytes are not FEC encoded.
- AUX1 packet, which is similar to the DH1 packet, except that is does not contain a CRC code. Consequently it can contain up to 30 information bytes, while a 1-byte payload header is included in the information bytes.

Note that the maximum payload size of a baseband packet is not linearly proportional to the number of slots covered by that baseband packet. Hence, for the same group of ACL packet types (DH or DM), the larger the baseband packet, the higher the net number of bytes per slot. Furthermore, DM packets protect the payload with a FEC at the cost of a lower maximum payload size. Consequently, a slot rate cannot directly be translated into a bit rate.

Four packet types are defined for use in an SCO link and are:

- HV1 (High-quality Voice) packet, which carries 10 information bytes that are encoded with a rate 1/3 FEC. The HV1 packet contains no payload header and no CRC and is never retransmitted.
- HV2 packet, which carries 20 information bytes that are encoded with a rate 2/3 FEC. The HV2 packet contains no payload header and no CRC and is never retransmitted.
- HV3 packet, which carries 30 information bytes that are not protected by FEC. The HV3 packet contains no payload header and no CRC and is never retransmitted.
- DV (Data Voice) packet, which is a packet that is divided into a 80-bit (10-byte) voice field and a 150-bit data field. The voice field is not protected by FEC, while the data field is coded by a rate 2/3 FEC. As opposed to the voice field, the data field is retransmitted if needed.

The packets described above contain a voice field and/or a data field. As opposed to voice fields, which contain no payload header and no CRC, the data fields consist of a payload header, a payload body, and a CRC code (not for AUX1). The payload header format of a single-slot packet is shown in Figure 2.8, whereas the payload header format of a multi-slot packet is shown in Figure 2.9. As can be seen, the payload header consists of a L_CH field, a FLOW field, and a LENGTH field. The L_CH field specifies the logical channel, i.e., it specifies whether the current baseband packet is the first fragment of an L2CAP message, a continuation fragment of an L2CAP message, or a so-called LMP message (see also Section 2.1.5). The FLOW field controls the flow on the logical channels. Finally, the LENGTH field indicates the length of the payload.

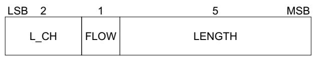


Figure 2.8: *Payload Header format of a single-slot packet [BT001]*

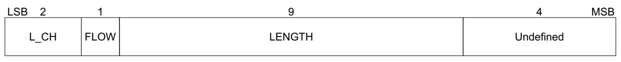


Figure 2.9: *Payload Header format of a multi-slot packet [BT001]*

The ACL and SCO packet types are summarized in Table 2.1 and Table 2.2, respectively. The tables shows, for each packet type, the size of payload headers (if any), the user payload size, the forward error correction type, whether a CRC is included or not, and the maximum rate that can be achieved using that packet type. The symmetric maximum rate is achieved when the same packet type is always used in the opposite direction, whereas the asymmetric maximum rate is achieved when a single slot packet is always used in the opposite direction. For instance, in case of a DH5 packet, at most $\frac{1600}{5+5}$ DH5 packets can be transmitted in one direction in the symmetric case, which corresponds to a maximum rate of 433.9 kbps. In the asymmetric case, at most $\frac{1600}{5+1}$ DH5 packets can be transmitted, which corresponds to a maximum bit rate of 723.2 kbps.

Type	Payload header (bytes)	User payload (bytes)	FEC	CRC	Symmetric maximum rate (kbps)	Asymmetric maximum rate (kbps)
DM1	1	0-17	2/3	yes	108.8	108.8
DH1	1	0-27	no	yes	172.8	172.8
DM3	2	0-121	2/3	yes	258.1	387.2
DH3	2	0-183	no	yes	390.4	585.6
DM5	2	0-224	2/3	yes	286.7	477.8
DH5	2	0-339	no	yes	433.9	723.2
AUX1	1	0-29	no	no	185.6	185.6

Table 2.1: *Overview of ACL packet types [BT001]*

Type	Payload header (bytes)	User payload (bytes)	FEC	CRC	Symmetric maximum rate (kbps)
HV1	na	10	1/3	no	64.0
HV2	na	20	2/3	no	64.0
HV3	na	30	no	no	64.0
DV	1 D	10+(0-9)D	2/3 D	yes D	64.0 + 57.6 D

Table 2.2: *Overview of SCO packet types. Items followed by a 'D' relate to the data field [BT001]*

2.1.4.2 Error correction

Besides the possibility of using a FEC for protection of information bits, an ARQ scheme is defined for the data. This ARQ scheme makes it possible to retransmit baseband packets that are not correctly received. The size of the ARQ field is 1-bit, which implies that there can be at most one outstanding unacknowledged packet per connection. Sometimes, higher-layer packets have a predefined lifetime, i.e., a time period outside which these packets become useless for the receiver. Consequently, it is a waste of bandwidth if fragments of these packets keep being retransmitted. In order to limit the time in which fragments of an L2CAP packet can be retransmitted, the possibility of flushing payload has been introduced. After a predefined flush timeout[3] T_{flush}, remaining segments of the L2CAP message being transmit-

[3]In this dissertation, it is assumed that the the flush timeout of an L2CAP packet is counted from the moment when the corresponding higher-layer packet arrives at the L2CAP layer

ted are flushed, and a subsequent L2CAP message (if any) becomes available for transmission. In Chapter 5, we make use of the flush timeout for providing QoS in a non-ideal radio environment.

2.1.4.3 Bluetooth link controller states

The Bluetooth link controller, which carries out the baseband protocol and other low-level link routines is in one of the following states: *standby*, *inquiry*, *inquiry scan*, *page*, *page scan*, or *connection*.

The *standby* state is the default state, in which the Bluetooth node is in a low-power mode. The standby can be left for performing inquiry or paging, or for scanning for inquiry or paging messages. In the *page scan* state, a slave listens to pages from masters that are in the *page* state and that try to activate and connect to a slave using its device access code (DAC). The DAC can be obtained through inquiry. The *inquiry* and *inquiry scan* states are similar to the *page* and *page scan* state respectively, except that instead of a device access code (DAC) inquiry messages include the general inquiry access code (GIAC) or a dedicated inquiry access code (DIAC). Including the GIAC makes it possible to discover any Bluetooth device that is within range, while including the DIAC makes it possible to discover Bluetooth devices of a certain type that are within range.

After successful connection, the Bluetooth node resides in the *connection state*, in which four modes can be distinguished, namely *active* mode, and three low power modes: *sniff* mode, *hold* mode, and *park* mode. In the *active* mode, a Bluetooth node actively participates on the channel, i.e., masters poll slaves, and slaves reply if polled. The *sniff* mode is a mode in which the master can only start transmission to a slave in specified and regularly spaced time slots. For instance, this makes it possible for a Bluetooth node to participate in multiple piconets, forming a so-called scatternet. During the *hold* mode, the ACL link between the master and a slave can be put on hold, while SCO links, if any, will still be supported. Finally, the *park* mode can be entered by a slave if participation on the channel is temporarily not needed. In that case, the slave to be parked gives up its active member address (AM_ADDR) and gets a parked member address (PM_ADDR) and access request address (AR_ADDR) instead. The PM_ADDR is used in case the master wants to unpark the slave, whereas the AR_ADDR is used by the slave when it requests to be unparked.

2.1.5 Link Manager Protocol

The Link Manager Protocol (LMP) is concerned with setup and management of baseband connections. It is for instance responsible for attaching and detaching slaves, for establishing SCO links, and for handling of the low power modes hold, sniff, and park.

With respect to link configuration, the LMP provides the capability of guaranteeing a poll interval T_{poll}, which is the maximum time between subsequent transmissions from the master to a slave on the ACL link. Furthermore, the baseband packet types to use are also negotiated and controlled by the LMP.

2.1.6 Logical Link Control and Adaptation Protocol

The Logical Link Control and Adaptation Protocol (L2CAP) provides data services to higher-layer protocols. It is concerned with higher-level protocol multiplexing, packet segmentation and reassembly (SAR), and the exchange of quality of service (QoS) information.

As baseband packets are of a relatively low maximum size, segmentation and reassembly capabilities are needed in order to transport higher-layer packets larger than the maximum baseband packet size. Considering the case in which L2CAP runs directly over the baseband protocol, a higher-layer packet is transported by an L2CAP packet, which is segmented into possibly multiple baseband packets. Each baseband packet carries an indication whether it is a first segment or a continuation segment, while the first segments also carries an indication of the length of the L2CAP packet. Once the baseband at the receiving side receives all segments belonging to an L2CAP packet, these baseband packets are reassembled at the L2CAP layer.

L2CAP channels between L2CAP entities in remote devices are either connection-oriented or connectionless. In both cases, L2CAP packets consist of a 2-byte length indicator field, a 2-byte channel ID (CID) field, up to 65533 bytes of information in case of a connectionless channel, and up to 65535 bytes of information in case of connection-oriented channel. In case of a connectionless channel, a Protocol/Service Multiplexer (PSM) field exists between the channel ID field and the information bytes. The PSM field values 0x0001, 0x0003, and 0x0005 correspond to Service Discovery Protocol, RFCOMM, and Telephony Control Protocol, respectively.

Signaling commands passed between L2CAP entities on remote devices use a CID value of 0x0001. Through this signaling connection requests, configuration requests, disconnection requests, echo requests, information requests, and their responses can be exchanged. The connection request and connection response are used to create an L2CAP channel between two devices. This channel can be terminated using the disconnection request and disconnection response. Echo requests and echo responses are used to test the link or to exchange vendor-specific information. The information request and information response is used to exchange implementation-specific information.

The configuration requests and configuration responses are used to establish an initial logical link transmission contract between two L2CAP entities. Furthermore, they are used to renegotiate such a contract whenever needed. The configuration parameter options that can be exchanged include the maximum transfer unit (MTU) option, the flush timeout option, and the quality of service (QoS) option.

The MTU option specifies the maximum L2CAP payload the configuration requester can accept for that channel. The flush timeout option specifies the amount of time the configuration requester will try to successfully transmit an L2CAP segment before flushing the L2CAP packet. The quality of service option is used to specify a flow specification similar the one proposed in [Par92], and which includes a token bucket specification [Par93]. This option makes it possible to specify whether a best effort service or a guaranteed service is needed. However, L2CAP implementations are not required to provide guaranteed service.

2.2 Internet Quality of Service

Traditionally, the Internet provides a *Best Effort (BE)* type of service, which means that the service in terms of, for instance, bandwidth, packet delay, and packet drop probabilities will be *as good as possible*. As real-time applications such as IP telephony and audio/video streaming became increasingly available, the need to provide a service better than best effort grew. This is especially the case in networks where bandwidth is scarce, e.g., in wireless networks.

The Internet Engineering Task Force (IETF) has proposed two service architectures for meeting this demand, namely the Integrated Services [BCS94] architecture and the Differentiated Services [BBC$^+$98] architecture.

2.2.1 Integrated Services

The Integrated Services model is based on per flow resource reservations. Before starting the actual data transmission, a communication path is set up between the sender and the receiver of the data flow which QoS is needed for. The IETF has proposed two service classes for the Integrated Services model, i.e., Guaranteed Service (GS) [SSG97] and Controlled-Load (CL) [Wro97]. This section briefly describes these service classes.

2.2.1.1 Guaranteed Service

The Guaranteed Service class (GS) [SSG97] makes use of the concept that packet delay in a network is a function of the arrival pattern of packets, the packet sizes, and the way these packets are served throughout the network. Guaranteed Service is based on the idea that if a GS flow is described (by the GS sender) using a token bucket [Par92][Par93] flow specification, and if the intermediate network between the GS sender and the GS receiver conforms the *fluid model* at service rate R (see Figure 2.10 for an illustration), then an end-to-end fluid model delay bound $d^{\mathrm{B}}_{\mathrm{fm}}$ can be computed.

The token bucket traffic specification consists of minimum policed unit m, maximum transfer unit M, token rate r^{t}, peak rate r^{p}, and bucket size b, where M, m, and b are specified in bytes, and where r^{t} and r^{p} are specified in bytes/sec. When testing conformance to the traffic specification, all packets with a size lower than the minimum policed unit m will be counted as being of size m. Consequently, traffic sources are encouraged to use packet sizes at least equal to the minimum policed unit. The maximum transfer unit M is the allowed maximum size of a packet.

The token bucket traffic specification specifies an envelope of the traffic source. Tokens are generated with rate r^{t}, and up to a maximum of b tokens can be stored in the token bucket.

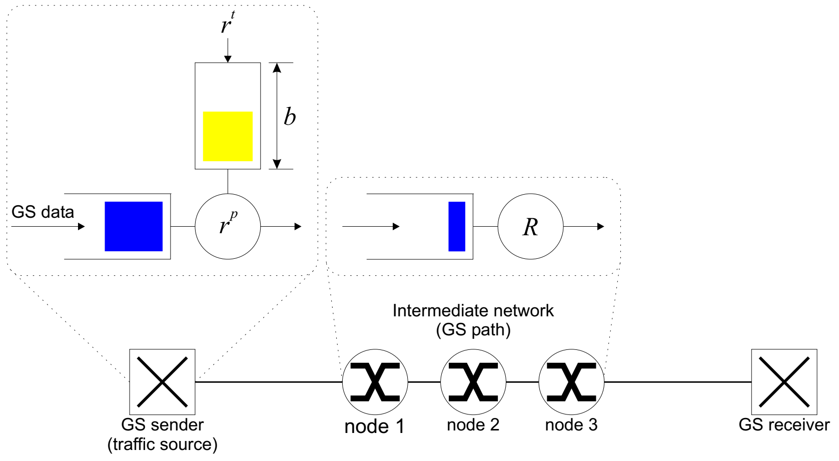


Figure 2.10: *Guaranteed Service model*

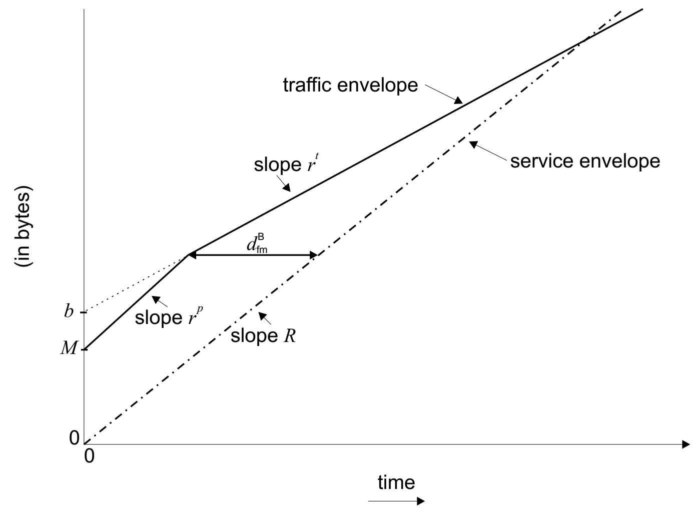


Figure 2.11: *Delay bound calculation for* $r^{\mathrm{t}} \leq R < r^{\mathrm{p}}$

If the bucket is full, newly generated tokens are lost. As long as there are tokens in the token bucket, data can be transmitted at peak rate[4] r^{p}, while removing one stored token for each transmitted byte. Figure 2.11 and Figure 2.12 show the traffic envelope and the service envelope in case of $r^{\mathrm{t}} \leq R < r^{\mathrm{p}}$ and $r^{\mathrm{t}} \leq r^{\mathrm{p}} \leq R$, respectively (see also [GGPR96]). The traffic envelope is characterized by the token bucket specification, and with the service envelope we refer to the maximum amount of service that can been provided by a dedicated

[4]According to [Par92], the peak rate can be seen as a form of a leaky bucket. When a packet of size L is sent, an amount of L credits are placed into an empty bucket, which drains credits at the peak rate. Furthermore, packets may only be sent if the bucket is empty.

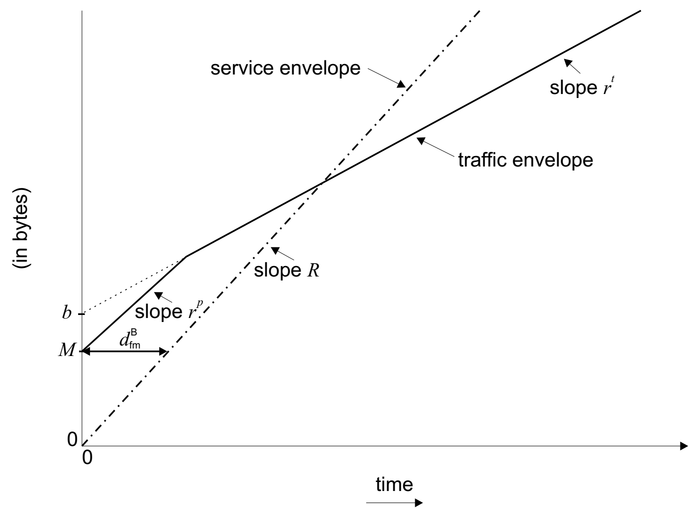


Figure 2.12: *Delay bound calculation for* $r^{\mathrm{t}} \leq r^{\mathrm{p}} \leq R$

wire at service rate R. From these figures, it follows that the end-to-end fluid model delay bound is given by

$$d^{\mathrm{B}}_{\mathrm{fm}} = \begin{cases} \frac{b-M}{R}\frac{r^{\mathrm{p}}-R}{r^{\mathrm{p}}-r^{\mathrm{t}}} + \frac{M}{R}, & r^{\mathrm{t}} \leq R < r^{\mathrm{p}}, \\ \frac{M}{R}, & r^{\mathrm{t}} \leq r^{\mathrm{p}} \leq R. \end{cases} \tag{2.1}$$

The Guaranteed Service architecture allows the intermediate network to deviate from the fluid model within a priori agreed-upon bounds. For this, during the Guaranteed Service setup, each of the intermediate nodes exports its deviation from the fluid model service rate R in terms of an additional delay. This deviation is expressed in both a rate-dependent deviation, known as the C *error term*, and a rate-independent deviation, known as the D *error term*. Given a requested service rate R, the end-to-end delay bound is then given by

$$d^{\mathrm{B}} = \begin{cases} \frac{b-M}{R}\frac{r^{\mathrm{p}}-R}{r^{\mathrm{p}}-r^{\mathrm{t}}} + \frac{M+C_{\mathrm{tot}}}{R} + D_{\mathrm{tot}}, & r^{\mathrm{t}} \leq R < r^{\mathrm{p}}, \\ \frac{M+C_{\mathrm{tot}}}{R} + D_{\mathrm{tot}}, & r^{\mathrm{t}} \leq r^{\mathrm{p}} \leq R. \end{cases} \tag{2.2}$$

where $C_{\mathrm{tot}} = \sum_i C_i$ and $D_{\mathrm{tot}} = \sum_i D_i$ are, respectively, the rate-dependent deviation and the rate-independent deviation summed over the intermediate nodes between the GS sender and the GS receiver.

Guaranteed Service is set up as follows. The Guaranteed Service traffic source (GS source) sends a token bucket specification of its traffic together with a requested delay bound to the receiver of the traffic (GS receiver). The intermediate nodes between the GS sender and the GS receiver compute their deviation from the fluid model and export this information to the receiver of the GS flow. Using the token bucket traffic specification, the exported deviations, and the requested delay bound, the requested service rate R that leads to the requested delay bound d^{B} is computed (for instance, by the receiver of the GS flow). Finally, the computed service rate R is requested (for instance, by the receiver) from the intermediate nodes between the GS sender and GS receiver.

The exchange of the aforementioned traffic specification, exported error terms, and requested service rate can be done using the *resource reservation protocol* (RSVP) [BZB+97]. With RSVP, a so-called Path message, containing the traffic specification and the delay request, is sent from the GS sender to the GS receiver, while the intermediate nodes add their C and D error terms to this message. Once the Path message arrives at the GS receiver, the GS receiver sends a so-called Resv message back to the GS sender. The Resv message contains the requested service R and is used to request this service rate from the intermediate nodes. Note, that the use of RSVP is not required by the Guaranteed Service architecture, which is independent of the former.

2.2.1.2 Controlled-Load service

The Controlled-Load (CL) [Wro97] service class is concerned with providing applications a behavior that approximates the behavior these applications would experience under lightly loaded network conditions, i.e., low packet drop ratio and low delay. For this, the Controlled-Load service requesting applications specify their traffic using a token bucket flow specification [Par92]. If the network elements in the CL service path accept the flow from the CL service requesting application, they commit to provide this flow a behavior that approximates the best effort behavior in the same CL service path under lightly loaded network conditions. As opposed to the GS class, the CL service class provides a relative QoS rather than an absolute QoS.

2.2.2 Differentiated Services

With Integrated Services, the amount of state information at the Integrated Services supporting network elements is proportional to the number of flows for which Integrated Services is provided. Furthermore, a complicated classification of all the data traffic must be performed. Consequently, Integrated Services does not scale well in networks with large numbers of flows, such as the Internet core. Therefore, the IETF introduced the Differentiated Services [BBC+98] (DS) architecture, which achieves better scalability by maintaining state information for aggregates of flows rather than for individual flows.

With Differentiated Services, customers have a Service Level Agreement (SLA) with their Internet Service Provider (ISP). The SLA specifies the supported service classes and the amount of traffic allowed for that service class. The DS field [NBBB98] is the Differentiated Services' interpretation of either the IPv4 Type of Service (TOS) field [Alm92], or the IPv6 Traffic Class byte, and is used for marking flow aggregates. Differentiated Services defines the layout of this DS field as well as a set of per-hop behaviors (PHBs), where a PHB is the forwarding behavior experienced at a DS-compliant node by packets with the same value of the DS field. By marking the DS field of individual packets, customers can get different types of services that are built with the PHBs.

The IETF has defined the Assured Forwarding (AF) PHB group [HBWW99] and the Expedited Forwarding (EF) PHB [DCB$^+$02]. The AF PHB group defines four independently forwarded AF classes. In each of these classes, IP packets can be assigned one of three different levels of drop precedence. The EF PHB can be used to build a service that can be characterized by low delay, low jitter, low packet loss ratio, and guaranteed rate. The services that can be built with the PHBs apply to aggregates of flows rather that to individual flows. In the Differentiated Services, individual flows have no means for requesting delay or rate guarantees.

2.2.3 Summary

The Integrated Services architecture provides the means to strictly control the quality of service. However, Integrated Services does not scale in core networks, where the number of flows is high. The Differentiated Services architecture does not have this scalability problem as it provides QoS to aggregates of flows, rather than to individual flows. Bernet et al. [BFY$^+$00] have proposed a framework for Integrated Services operation over Differentiated Services networks. Such a framework makes it possible to use the Integrated Services approach in the access networks (e.g. wireless access network), where the number of flows is relatively low, and to use the Differentiated Services approach in the core network, where the number of flows is high.

Chapter 3

Scheduling Best Effort Traffic in Bluetooth

This chapter discusses the development of a Bluetooth polling mechanism that performs at least as good as the existing Bluetooth polling mechanisms, while being able to provide QoS if properly set. The chapter is structured as follows. Section 3.1 presents the problem description. Section 3.2 presents the work from the literature related to this chapter. Section 3.3 explains the operation of the poller being developed, namely the Predictive Fair Poller (PFP) [HAY01, AYH01b]. Section 3.4 presents an analysis of the stability, efficiency, and fairness of the 1-limited Round Robin poller. Section 3.5 evaluates PFP and the most important alternative polling mechanisms in a simulation study, and Section 3.6 concludes this chapter.

3.1 Problem Description

3.1.1 Introduction

Bluetooth is a time-slotted wireless access technology where each second is divided into 1600 time slots. Time slots are either downlink slots, i.e., from the master to a slave, or uplink slots, i.e., from the addressed slave to the master. Data is exchanged between the master and a slave using Baseband packets that cover one, three or five time slots. Various other protocols can be used on top of Bluetooth. For instance, IP over Bluetooth can be used, where IP packets cover one or more baseband packets.

As mentioned in Chapter 2, the traffic within a piconet is controlled by the master of that piconet such that a slave is only allowed to transmit if it was addressed (by the master) in the previous time slot. In other words, the master polls the slaves giving them an opportunity to transmit data. A master polls a slave either implicitly or explicitly, where an implicit poll of a slave means that the master polls the slave by sending a packet containing data to that slave. An explicit poll of a slave means that the master sends a packet with no payload (POLL packet) to that slave, for instance when data destined for that slave is not available. Similarly, a slave that has no data destined for the master responds to a poll with a packet with no payload (NULL packet). Figure 2.4 shows an example of a master polling seven slaves.

3.1.2 Goals for a Poller

Bluetooth supports two types of links between a master and a slave: a *Synchronous Connection-Oriented* (SCO) link and an *Asynchronous Connection-Less* (ACL) link. Packets sent over an SCO link (SCO packets) cover one time slot, whereas packets sent over an ACL link can cover one, three or five time slots.
In case of an SCO link between the master and a slave, the master polls that slave at regular intervals. The addressed slave can then respond with an SCO packet. In case of an ACL link, polling can be done in many different ways. The difference between the polling mechanisms

is related to the order in which slaves are polled and the service discipline used to serve a slave. Goals for a poller are to obtain high efficiency, fairness and a low mean response time. In the following subsections we discuss these performance goals, and relate them to concrete metrics.

3.1.2.1 Efficiency

Slaves are only allowed to transmit if they were addressed in the previous time slot. If a master wants to allow a slave to transmit, it can send a data packet to that slave if a data packet is available. Otherwise, the master must send a POLL packet, i.e., a baseband packet with no payload. On its turn, if the addressed slave has data to send back to the master, it will send that data. Otherwise, it will send a NULL packet, i.e., a baseband packet with no payload. We define the *efficiency of a poller* over a period $(a, b]$ as the ratio of the number of time slots used for data transport to the sum of the number of POLL packets, the number of NULL packets and the number of time slots used for data transport, i.e.,

$$\eta_{(a,b]} = \frac{d_{(a,b]}}{d_{(a,b]} + p_{(a,b]} + n_{(a,b]}}, \tag{3.1}$$

where $p_{(a,b]}$ is the number of POLL packets during $(a, b]$, $n_{(a,b]}$ is the number of NULL packets during $(a, b]$, and $d_{(a,b]}$ is the number of time slots used for data transport during $(a, b]$.

As can be seen in (3.1), given the same amount of used time slots ($d_{(a,b]} + p_{(a,b]} + n_{(a,b]}$), a higher number of POLL or NULL packets leads to a lower efficiency.

3.1.2.2 Total mean response time

Let us first define the waiting time W_i of an L2CAP packet i as the time between the moment when the L2CAP packet becomes available and the moment when the transmission of the first segment of that L2CAP packet starts. Furthermore, let us define the service time S_i of an L2CAP packet i as the time between the moment when the transmission of the first segment of that L2CAP packet starts and the moment when the transmission of the last segment of that L2CAP packet ends. Figure 3.1 shows the waiting time and service time of an L2CAP packet that will be transmitted using one DH3 packet and one DH1 packet. The total mean response time MRT is the mean of the sum of the service time and the waiting time taken over all the L2CAP packets transmitted in the piconet, i.e.,

$$MRT = \frac{1}{n}\sum_{i=1}^{n}(W_i + S_i) \tag{3.2}$$

where the L2CAP packets transmitted in the piconet are numbered from 1 to n.

There are different ways for a polling mechanism to achieve the same value of efficiency. However, the different algorithms will not necessarily lead to the same total mean response time.

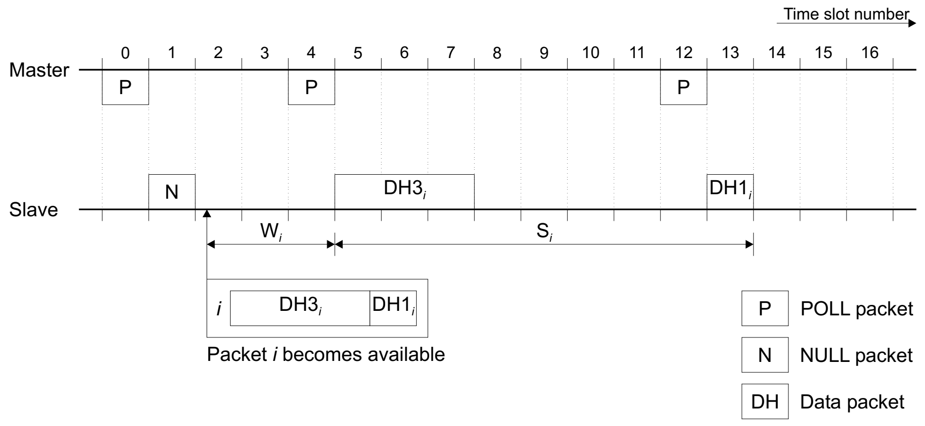


Figure 3.1: *Transmission diagram of an L2CAP packet*

3.1.2.3 Fairness

In order to examine the fairness of a system, an allocation metric and a formula resulting in a fairness index are needed. The allocation metric is the metric for which the fairness is to be investigated, while the fairness index shows how fair the system is with respect to the given allocation metric. Given allocation metric x_i, we use the definition of fairness index presented in [JCH84]:

$$f(\boldsymbol{x}) = \frac{\left\{\frac{1}{n}\sum_{i=1}^{n} x_i\right\}^2}{\frac{1}{n}\sum_{i=1}^{n} x_i^2}. \tag{3.3}$$

This fairness index is bounded between zero and one, making it possible to show how near the system is to total fairness ($f(x) = 1$) or to total unfairness ($f(x) = 0$). Furthermore, whenever a single allocation changes, the fairness index will also change. Finally, this fairness index is a linear measure of fairness. For instance, if there are U units to be divided among n users and we give m ($m \leq n$) users $\frac{U}{m}$ units, then the fairness index will be equal to $\frac{m}{n}$.

The next step is to decide what x_i should represent. In other words, an allocation metric should be chosen. Possible allocation metrics are:

- Throughput
- Fraction of fair share (of resources)
- Waiting time, service time or response time
- Inverse fraction of reference waiting time

We have chosen fairness based on fraction of fair share and fairness based on inverse fraction of reference waiting time. We will further define and justify our selection.

Fairness based on Fraction of Fair Share

Consider a system in which an amount of resources are to be divided among users requesting resources, and in which each user i requests an amount of resources d_i. The fair shares of resources fs_i that should be allocated for each user i are determined based on the user demands d_i in two stages. During the first stage, resources are equally divided between users that need these resources, while no user gets more resources than it is requesting. The first stage ends as soon as each user is allocated the resources it is requesting or as soon as there are no more resources to divide. During the second stage, remaining resources, if any, are divided among the users proportional to the resources that were allocated to them during the first stage.

Figure 3.2 shows an algorithm that performs the first stage of determining fair shares of resources. First, this algorithm (step 1) checks whether the sum of the demands is lower than the total amount of resources. In that case, each user is allocated an amount of resources equal to its demand and the algorithm stops. Step 2 in the algorithm calculates the amount of remaining resources C that can be divided among the users for which a fair share is not determined yet. Based on this amount of remaining resources C and on the number of users for which a fair share is not determined yet, step 3 calculates the fair share of resources A_{fair} that these users can get at most. In step 4, if a user does not need more than this fair share A_{fair}, it is allocated a share equal to its demand and it is removed from the list of remaining users for which a fair share is not determined yet. If this list does not change during step 4, this indicates that these remaining users need more than fair share A_{fair}. In that case, these users are allocated an amount of resources equal to A_{fair} (step 6) and the algorithm stops. Otherwise, the algorithm returns to step 2. The following two examples show how resources are divided among users based on this algorithm:

1. Suppose 100 units of resource must be divided among 4 users with demands $\boldsymbol{d} = \{5, 10, 15, 20\}$. After the first stage, and as the sum of the demands is lower than the available resources, each user gets a share equal to its demand and thus $fs = \{5, 10, 15, 20\}$. During the second stage, the remaining resources are divided among the users and the fair allocation of resources will be $fs = \{10, 20, 30, 40\}$
2. Consider the same amount of resources and the same number of users as in example 1, but let the demands be $\boldsymbol{d} = \{15, 25, 40, 50\}$. After the first stage, the allocated resources will be $fs = \{15, 25, 30, 30\}$. As the sum of the demands is higher than the available resources, no more resources will be divided during the second stage. Hence, the fair allocation of resources is the allocation that follows from the first stage.

We state that a system is fair based on fraction of fair share if each user gets the same fraction of its fair share of resources. An allocation metric that will let the fairness index have this meaning is the fraction of fair share

$$x_i = ffs_i = \frac{s_i}{fs_i}, \tag{3.4}$$

where s_i is the amount of resources actually given to user i (actual share) and fs_i is the fair share of resources for user i. This is similar to the fraction of demand presented in [JCH84].

With respect to a Bluetooth polling mechanism, the resources mentioned above correspond to the wireless link resources, which can be represented as bit rates, slot rates or poll rates. However, as mentioned in chapter 2, bit rates, slot rates and poll rates cannot directly be

a. Let C_{tot} be the total amount of resources

b. Let C be the remaining amount of resources that can be allocated, while initially $C = C_{tot}$

c. Let S be the set of remaining users to have their fair share determined, while initially $S = \{1, 2, ..., n\}$, thus with cardinality $\mathrm{card}(S) = n$

d. Let d_i be the resource demands of user i

e. Let A_i be the fair share for user i, while initially $A_i = 0$, $i = 1, 2, ..., n$

f. Let A_{fair} be the fair share given the remaining amount of resources C and the remaining users S to have their fare share determined.

1. If $\sum_{i=1}^{n} d_i < C_{tot}$ then $\forall i \in S$ let $A_i = d_i$ and exit
2. $C = C_{tot} - \sum_{i=1}^{n} A_i$
3. $A_{fair} = \frac{C}{card(S)}$
4. $\forall i \in S$ with $d_i \leq A_{fair}$ let $A_i = d_i$ and remove user i from S
5. If S changed during step 4 and $card(S) \geq 1$ then goto step 2.
6. $\forall i \in S$ let $A_i = A_{fair}$

Figure 3.2: *First stage of fair share determination procedure [RJC87]*

translated into each other. Hence, it must be decided which resource a Bluetooth polling mechanism must fairly divide among the users requesting it.

Chapter 2 also mentioned that the larger the used baseband packet size, the higher the slot efficiency (average number of bytes per slot). Hence, in order to achieve higher slot efficiencies and thus saving more time slots, Bluetooth nodes should be encouraged to transmit their data as efficient as possible. This can be achieved by considering the available polls as the available resources to be fairly divided, i.e., by fairly dividing polls between users that need to be polled. In that case, slaves will try to transmit as much as possible data per poll.

Fairness based on inverse fraction of reference waiting time

It is possible that two polling mechanisms show the same value for the fairness based on fraction of fair share, the same value for the efficiency and the same total mean response time. Nonetheless, one of the mechanism can be preferred over the other. For instance, consider a system with a master with capacity C and two slaves, where at both slaves a large burst of data, equal to CT_h (with T_h in seconds), arrives at time $t = 0$ sec. Consider the following two polling mechanisms (M1 and M2):

M1: The master polls slave 1 is during time period $[0, \frac{1}{2}T_h)$, whereas slave 2 is polled during time period $[\frac{1}{2}T_h, T_h)$.

M2: The master polls slave 1 and slave 2 in a 1-limited Round-Robin fashion during time period $[0, T_h)$.

Taken over the time period $[0, T_h)$, both polling mechanisms will obtain the same fairness based on fraction of fair share (equal to 1), the same efficiency, and the same total mean response time (equal to $\frac{T_h}{2}$). However, with polling mechanism M2, packets from both slave 1 and slave 2 experience the same mean response time, whereas with polling mechanism M1 packets from slave 1 experience a lower mean response time than packets from slave 2 do. We state that polling mechanism M2 is more fair than polling mechanism M1, and introduce an additional performance metric to express this, namely fairness based on inverse fraction of reference waiting time.

The reference mean waiting time $W_{\mathrm{R}i}$ of packets from a user i is defined as the mean waiting time that would be experienced by those packets if they were transmitted over a link with a constant bit rate R_i. The constant bit rate R_i corresponds to the average throughput that the same user would experience in the Bluetooth polling system given the same packet size distribution, a fair share fs_i (in polls/sec), and an arrival rate that is higher than the fair share (fair poll rate).

Consider a slave i at which packets of an average size L_i arrive according to an arrival process that is characterized by arrival rate λ_i and variability parameter $c_{a_i}^2$, which is defined as the squared coefficient of variation of the inter-arrival time. Furthermore, fs_i is the fair share of resources that slave i should get. The continuous bit rate that is used to determine the reference waiting time $W_{\mathrm{R}i}$ is

$$R_i = fs_i L_i. \tag{3.5}$$

The reference mean waiting time of a slave i is determined using the waiting time approximations presented in [Whi83], namely

$$W_{\mathrm{R}i} \cong \frac{\tau_i \rho_i (c_{a_i}^2 + c_{s_i}^2) g_i}{2(1-\rho_i)}, \tag{3.6}$$

where $g_i \equiv g_i(\rho, c_{a_i}^2, c_{s_i}^2)$ is defined as

$$g_i(\rho, c_{a_i}^2, c_{s_i}^2) = \begin{cases} e^{\left(-\frac{2(1-\rho_i)}{3\rho_i}\frac{(1-c_{a_i}^2)^2}{c_{a_i}^2+c_{s_i}^2}\right)}, & \text{for } c_{a_i}^2 < 1, \\ 1, & \text{for } c_{a_i}^2 \geq 1, \end{cases} \tag{3.7}$$

where the service time is characterized by mean service time $\tau_i = \frac{L_i}{R_i}$ and variability parameter $c_{s_i}^2$. As we assume a constant bit rate R_i, the variability parameter of the service time equals the variability of the packet sizes.

Given a determined reference mean waiting time $W_{\mathrm{R}i}$ for slave i and the actual mean waiting time $\overline{W_i}$ we let the fairness index of (3.3) have as meaning fairness based on inverse fraction of reference waiting time by defining the allocation metric x_i as

$$x_i = \frac{1}{\frac{\overline{W_i}}{W_{\mathrm{R}i}}} = \frac{W_{\mathrm{R}i}}{\overline{W_i}}. \tag{3.8}$$

3.2 Related work

Scheduling the traffic from a master to a slave and vice versa is referred to as Bluetooth polling, Bluetooth MAC scheduling or intra-piconet scheduling, while scheduling the node's participation in different piconets is referred to as inter-piconet scheduling. Bluetooth polling mechanisms are studied in [CGK01, JKJ99, MKM04, KBS99, CKR$^+$00, CKK$^+$01, DGR$^+$01, BCG01, RBK01], and [PH03]. We will briefly describe them in this section.

3.2.1 Cyclic polling

The 1-limited Round-Robin (RR) poller is a very simple poller that polls slaves in a cyclic manner, regardless of the availability of data. Every time a slave is polled, that slave is allowed to transmit exactly one Baseband packet. In case of symmetric loads, the 1-limited Round Robin poller obtains the maximum obtainable fairness and efficiency. However, consider a polling system with N slaves with asymmetric loads, where some slaves require more than $\frac{1}{N}$ of the total available number of polls and other slaves require less than $\frac{1}{N}$ of the total available number of polls. Also consider the total needed number of polls to be less than the total available number of polls. In this case the poller fails both to be fair based on fraction of fair share and fails to be efficient. The reason for the poller not being fair is that the poller will give the slaves requiring more than $\frac{1}{N}$ of the total available number of polls no more than $\frac{1}{N}$ of the total available number of polls. The reason for the poller not being efficient is that the poller will poll the slaves requiring less than $\frac{1}{N}$ of the total available number of polls even when they have no data to transmit.

Besides the 1-limited round robin poller, other pollers that poll slaves in a cyclic manner exist. The exhaustive Round Robin poller polls the slave in a cyclic manner, and the next slave is polled only if all the data available at the current slave is served. The exhaustive Round Robin poller can handle load that is asymmetrically distributed among the slave, but has the disadvantage that a busy slave can consume all the bandwidth. Note that a non-cyclic version of this poller, namely Exhaustive Pseudo-cyclic Master queue length poller (EPM) has been proposed in [CGK01]. The main difference with the exhaustive Round Robin poller is that, at the beginning of each cycle, the cycle order is defined according to a decreasing master to slave queue length order.

The n-limited Round Robin poller is also a cyclic poller, where the next slave is polled if n packets are served from the current slave or if the current slave has no more packets to be served. The n-limited Round Robin poller can handle load that is asymmetrically distributed among the slaves up to a given asymmetry level as long as n is set correctly. An extended version of the n-limited Round Robin poller is the Limited and Weighted Round Robin poller (LWRR), which is proposed in [CGK01]. It adopts a weighted Round Robin algorithm with dynamically changed weights. Each slave is assigned a weight, which is initially set to MP (Maximum Priority), and which is at least equal to 1. Each time a slave i is polled and no data is exchanged, the weight of that slave is decreased by 1. As soon a slave i is polled and data is exchanged, the weight of that slave is increased to MP.

Another cyclic poller is the Deficit Round Robin (DRR) poller [SV96], which is similar to the 1-limited Round Robin poller, except that each slave is assigned a given time quantum during a polling cycle. If a queued packet is larger than the time quantum, that packet is not served, and the remainder from the current quantum is added to the quantum in the next cycle. In other words, DRR keeps track of deficits, i.e., slaves are compensated for missed service time in a next cycle.

3.2.2 Fair Exhaustive Polling

The Fair Exhaustive Poller (FEP) is described in [JKJ99]. The main idea behind FEP is to maintain two lists. A list of active nodes (AN) and a list of non-active nodes (NN). A slave belongs either to the list of active nodes or to the list on non-active nodes depending on the activity of that slave. When the polling starts, the poller moves all the slaves to the list of active nodes and the nodes in the active list are polled in a Round-Robin way. Inactivity of a slave i can be detected by using an inactivity metric. For instance, this inactivity metric can be the number of successive unsuccessful polls of slave i, where an unsuccessful poll refers to a poll in the absence of data to be transmitted, or some function of the hit ratio of that slave, which is the success ratio of a poll for that slave. Whenever the inactivity metric of slave i exceeds a predefined threshold, slave i is moved to the list of inactive nodes. Furthermore, a maximum inter-poll time T_{poll_i} is defined for each slave i. After being in the list of inactive nodes for at most a time T_{poll_i}, the slave is polled again. If a polled slave i responds with data (successful poll), that slave is returned to the list of active nodes.

3.2.3 Adaptive Cycle-Limited Scheduling

The Adaptive Cycle-Limited Scheduling algorithm, which is introduced in [MKM04], is based on two main concepts. First the piconet cycle time is limited in order to maintain a maximum poll interval. Second, it tries to minimize packet delays by adjusting the time allocated to each slave. The highly loaded slaves will get a larger portion of the cycle time, while the lowly loaded slaves will get a smaller portion.

3.2.4 HOL Priority and HOL K-Fairness Scheduling

Kalia et al. introduced two scheduling policies, which they labeled as Master-Slave Queue-State-Dependent Packet Scheduling policies [KBS99] [KBS00]. The policies they introduced assume knowledge of the head-of-line (HOL) packet at both the master and the slaves. In the presence of an SCO channel, and thus in case of the availability of a (small) fixed number of time slots between an SCO packet from a slave and the next SCO packet from the master, master-slave pairs are classified in different classes depending on the amount of wasted slots when serving the slave belonging to that master-slave connection. For instance, consider the case in which the number of slots between an SCO packet from a slave and the next SCO packet from the master is four time slots. If the master has a head-of-line packet of three time slots destined for a particular slave and if that slave has no data packet available for transmission, then that master-slave connection belongs to the class of master-slave connections that cause one time slot to be wasted. Having these classes of master-slave connections they defined two policies to serve slaves belonging to these classes:

- *HOL Priority Policy*: the master-slave pairs are served in a weighted Round Robin[1] manner, where each master-slave pair has a priority that depends on the class they belong to (the higher the number of wasted slots, the lower the priority). A delay comparison of the HOL Priority Policy with other polling schemes that also have knowledge of the slave to master queues can be found in [CGK01].

[1]Each slave is polled a number of times, that depends on its weight, before the next slave is polled.

- *HOL K-Fairness Policy (HOL-KFP)*: the backlogged master-slave pairs are visited in a Round Robin manner. However, up to a bound, master-slave pairs belonging to classes with higher wastage transfer service to master-slave pairs belonging to classes with lower wastage.

3.2.5 Flow-bit based polling

The polling mechanisms of this type are described in [DGR$^+$01] and use the flow bit present in the payload header field of the baseband packet. A variable flow is defined, which is set to one if the flow bit in either direction is set to one. Three polling mechanisms are described:

- *Adaptive Flow-based Polling (AFP)*: Each master-slave connection starts with a negotiated minimum polling interval P_0, where a polling interval is the maximum interval before which the slave belonging to this master-slave connection must be served. Furthermore, AFP uses an adaptive polling interval that changes based on the variable flow. If for a master-slave connection $\mathit{flow} = 1$ and the head-of-line packet to the particular slave is a data packet then the data packet is transmitted and the variable polling interval is set to its minimum, i.e., $P = P_0$. If for a master-slave pair $\mathit{flow} = 0$ and the head-of-line packet is a data packet then the data packet is transmitted and the variable polling interval is kept unchanged. Furthermore, if a POLL packet is transmitted and a NULL packet is received then the variable polling interval is doubled unless a maximum polling interval P_{thresh} is reached.

- *Sticky poller*: The slaves are served in a cyclic manner. However, if for a master-slave pair $\mathit{flow} = 1$ then when the turn of the particular slave arrives it will be served a maximum num_sticky times before serving the next slave. On the other hand, if for a master-slave pair $\mathit{flow} = 0$ then when the turn of the particular slave arrives it will be served only once.

- *Sticky Adaptive Flow-based Polling (StickyAFP)* is similar to AFP except for the following: if for a master-slave pair $\mathit{flow} = 1$ and the head-of-line packet to the particular slave is a data packet then when the turn of the particular slave arrives it will be served a maximum num_sticky times before serving the next slave.

3.2.6 Efficient Double Cycle Scheduling Algorithm

The Efficient Double Cycle Scheduling Algorithm (EDC) is described in [BCG01]. The main idea behind this algorithm is decoupling the scheduling of the transmissions in the uplink direction and the transmissions in the downlink direction. The authors introduced the idea of a double polling cycle: an uplink polling sub cycle ($Cycle_{UP}$) and a downlink polling sub cycle ($Cycle_{DW}$). Before each cycle the master selects a set of slaves that are eligible for polling, which are referred to as $E(UP)$ and $E(DW)$. $E(DW)$ is determined by considering the local master to slave queues. On the other hand, $E(UP)$ is determined based on the poll results. If a poll resulted in the slave responding with a NULL packet then that slave will be skipped a number w_k of $Cycle_{UP}$'s, while w_k is increased each time a NULL packet is received up to a maximum $w_{\max}$. As soon as a slave responds to a poll with a data packet then w_k will be set to one and that slave will be included in the next $E(UP)$.

3.2.7 Demand-based Bluetooth Scheduling

The demand-based Bluetooth scheduling algorithm is described in [RBK01]. Considering the polling positions (slots) of synchronous slaves and the polling positions of slaves that belong to more than one piconet (shared slaves), this scheduling algorithm is used to schedule transmissions to (and from) the remaining asynchronous slaves, which are referred to as asynchronous dedicated slaves (ADS). Each ADS has a current polling period n and a time-stamp t that is increased by n whenever that ADS is polled. Initially each ADS polling period is set to the number of asynchronous dedicated slaves D. During a slot that is not used by synchronous slaves or shared slaves, the ADS with the smallest time stamp is polled. If the polling of an ADS resulted in a POLL and a NULL packet, then the polling interval n of that ADS is increased with a constant value m unless a maximum polling interval n_{max} will be exceeded. In that case the master would consider parking that ADS. On the other hand, if the polling of an ADS resulted in data being transmitted in either or both direction then the polling interval of that ADS will be reset to D. Note that the maximum polling interval n_{max} is chosen such that if the polling period of an ADS is $n = n_{max}$ then that slave is inactive enough to be parked.

3.2.8 Sniff-based polling

One of the operation modes of a Bluetooth node described in the Bluetooth specification is the sniff mode. The sniff mode is used to reduce a slave's listen duty cycle. Consequently, if a slave is in sniff mode then the master can only start transmission to that slave in specified time slots. These time slots are called sniff slots and are spaced regularly with an interval of T_{sniff}. The polling mechanisms described in this section are introduced in [CKR+00] and [CKK+01], and make use of the sniff mode. Considering the fact that polling an inactive slave is a waste of power and bandwidth and considering the fact that switching a slave to sniff mode and back also costs power and bandwidth the main idea behind this mechanisms is to switch a slave to sniff mode whenever switching a slave to sniff mode and back costs less power and bandwidth than staying in active mode (normal operation mode). The following switching policies have been defined in [CKR+00] and [CKK+01]:

- *Mean Policy (MEAN)*: Assuming that the inter-arrival time between the last burst of data packets and the next burst of data packets is correlated to the inter-arrival times of the previous bursts of data packets, the master and the slave store the mean of the last several inter-arrival times of data packet bursts. The minimum of the mean of the inter-arrival time of bursts of data packets at the master destined for a particular slave and the mean of the inter-arrival time of bursts of data packets at that particular slave destined to the master is taken to be the expected next inter-arrival time of bursts of data packets to or from that particular slave. Consequently, that slave is switched to sniff mode with a sniff interval equal to that minimum. As soon as the master or that particular slave receives more than two packets (destined for the other side of the master-slave connection) at their queues then the slave is switched back from sniff mode when the master polls the slave again.

- *Last Inter-Burst Time (LIBT)*: Assuming high correlation between two consecutive inter-arrival times of bursts of data packets, the last inter-arrival time of bursts of data packets is used as the predicted value for the next inter-arrival time of bursts of data packets. This is done for both traffic from the master to a particular slave and from that particular slave to the master. The minimum of the two expected inter-arrival times is used as a sniff interval when switching that particular slave to the sniff mode. The criteria for switching that particular slave back from sniff mode is the same as in the MEAN policy.

- *Queue Status based Polling Interval (QSPI)*: Whenever there is no data destined to send to a particular slave then that slave is switched to sniff mode with a fixed value for the sniff interval. In sniff mode, the slave can be in one of two state: state I with the mentioned fixed sniff interval and state II with double the sniff interval from state I. When the slave is switched to sniff mode it will be first in state I. If a slave in state I is polled and that slave has no data destined for the master then that slave is moved to state II. If a slave in state I is polled and that slave has one packet destined for the master then that slave remains in state I, otherwise it is switched back from sniff mode. On the other hand, if a slave in state II is polled while it has two packets destined for the master then it is moved to state I. If a slave in state II is polled and it has more then two packet destined for the master then it is switched back from sniff mode, otherwise the slave remains in state II.

- *Adaptive Probability-based Polling Interval (APPI)*: Slaves are put in sniff mode using probabilistic estimates of inactivity which are based on the previous traffic arrival pattern.

3.2.9 Adaptive Share Polling

The Adaptive Share Polling (ASP) mechanism, which is proposed in [PH03], aims at avoiding the following two situations. First, it tries to avoid polling slaves that do not have data available for transmission by keeping track of the share of bandwidth needed by each slave. Second, it prevents the slaves from listening when knowing that they will not be polled for a while. For that, ASP puts slaves in hold mode for periods that depend on the traffic load.

3.3 Predictive Fair Polling

As mentioned in Section 3.1.2, it is a goal for pollers to be efficient, to be fair, and to obtain low delays. The 1-limited Round-Robin poller is neither efficient nor fair when the load is asymmetrically distributed among the slaves. On the other hand, the other pollers mentioned in the previous section are to a certain extent fair and efficient, which is sufficient for most best effort applications (e.g. web browsing, email, file transfer etc…). However, QoS applications (e.g. audio and video transfer) will play an important role in the usage of the Bluetooth technology. With respect to QoS traffic handling, the pollers mentioned in the previous section will not be able to handle QoS traffic as such. Consequently, we decided to come up with a poller that performs as good as the existing pollers in the best effort case, and that can be extended to handle QoS traffic.

We have introduced a poller named Predictive Fair Poller (PFP) [HAY01, AYH01b], which takes both efficiency and fairness into account. It predicts for each slave whether data is available or not and it keeps track of the fairness. Based on these two aspects it decides which slave to poll next. In the Best Effort case, the Predictive Fair Poller estimates the fair share of resources for each slave and keeps track of the fractions of these fair shares that each slave has been given. The Predictive Fair Poller can be used to poll Best Effort traffic in a fair and efficient manner by keeping track of both the fairness based on these fractions of fair share and the predictions. In the QoS case, QoS requirements are negotiated with the slaves and translated to fair QoS treatments. The poller keeps track of the fractions of these fair QoS treatments that each slave has been given. Similar to the Best Effort case, the Predictive Fair Poller can be used to poll QoS traffic such that the QoS requirements are met by keeping track of the fairness based on these fractions of the fair QoS treatments.

First, we show and explain the building blocks of the Predictive Fair Poller. Subsequently, we discuss the implementation of these building blocks for the Best Effort case. For the sake of simplicity we discuss the Predictive Fair Poller only for traffic destined from the slaves to the master. However, we will indicate how the Predictive Fair Poller can also handle traffic from the master to the slaves.

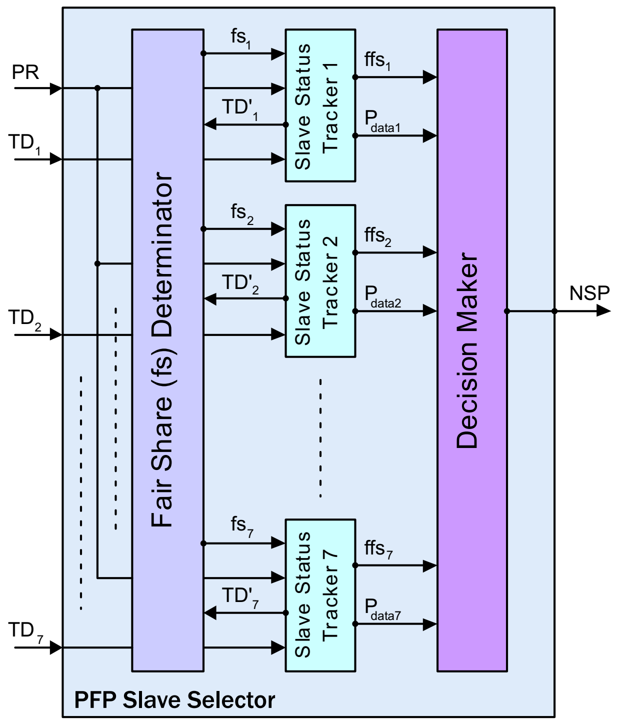
3.3.1 Building blocks of PFP

The selection of the next slave to be polled is performed by the PFP Slave Selector, which is shown in Figure 3.3. It is located in the master and requires knowledge of the results (PR) of its poll decisions. The PFP Slave Selector can be fed with a Traffic Demand (TD_i) for each slave i in order to support QoS traffic. However, if a slave does not make its traffic demand known to the master then the Traffic Demand TD_i^{e} estimated by the Traffic Demand Estimator in the Slave Status Tracker (see also Figure 3.4) will be used instead. In other words, the Traffic Demand (TD_i') is either the Traffic Demand (TD_i) made known by slave i (e.g. QoS case) or the Traffic Demand TD_i^{e} estimated by the Traffic Demand Estimator in case slave i did not make its Traffic Demand known to the master (Best Effort case).

The Fair Share Determinator in the PFP Slave Selector uses the Traffic Demands ($TD_1'..TD_7'$) to determine the Fair Share (fs_i) of bandwidth for each slave i.

Besides a Traffic Demand Estimator the Slave Status Tracker also contains a Fraction of Fair Share Determinator and a Data Availability Predictor. The Fraction of Fair Share Determinator in Slave Status Tracker i uses the Fair Share (fs_i) determined by the Fair Share Determinator and the Poll Results (PR) to determine the Fraction of Fair Share of bandwidth (ffs_i) that slave i has been given. The Data Availability Predictor in Slave Status Tracker i uses the Traffic Demand (TD_i') and the Poll Results (PR) to determine the probability (P_{data_i}) of data being available for transmission from slave i to the master.

The probabilities (P_{data_i}) of data being available for transmission from each slave i to the master and the Fractions of Fair Share of bandwidth (ffs_i) that each slave i has been given are used by the Decision Maker to decide which slave to poll next. The decision rules depend on the requirements on both the efficiency and the fairness.

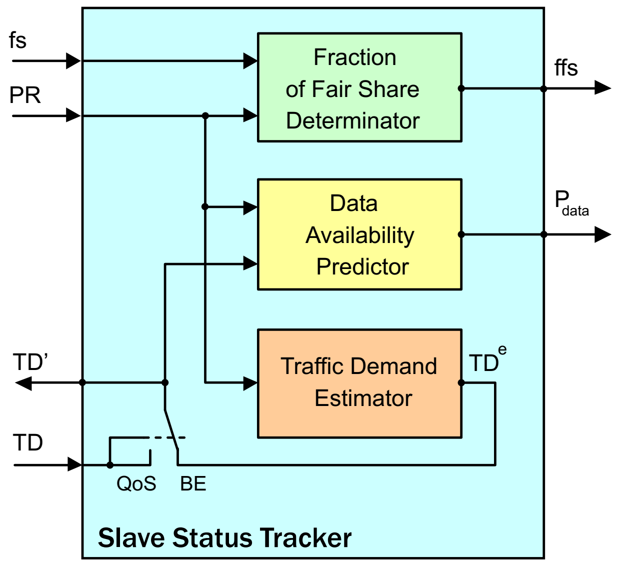


TD	Provided Traffic Demand (QoS)
TD'	Traffic Demand
PR	Poll Result of the last poll
fs	Fair Share
ffs	Fraction of Fair Share
NSP	Next Slave to Poll
P_{data}	Probability of data being available for transmission

Figure 3.3: *Block diagram of the PFP Slave Selector*

3.3.2 Implementation of PFP for the Best Effort case

In the Best Effort case the poller is not given any information about the offered load. This means that neither the L2CAP packet sizes (and thus the number of Baseband packets belonging to the same L2CAP packet) nor the inter-arrival times of these L2CAP packets are known to the poller. Furthermore, the distribution of the inter-arrival times of these L2CAP packets is also unknown. As a result, we decided to assume that the load is generated according to a Poisson process, i.e., the L2CAP packets are generated with exponential inter-arrival



TD	Provided Traffic Demand (QoS)
TD^e	Estimated Traffic Demand (BE)
TD'	Traffic Demand
PR	Poll Result of the last poll
fs	Fair Share
ffs	Fraction of Fair Share
P_{data}	Probability of data being available for transmission

Figure 3.4: *Block diagram of the Slave Status Tracker*

times. Depending on their size, L2CAP packets may be segmented into multiple segments (see also Section 2.1.6). During this section we will use the following terminology:

- Macro poll is a poll to which a slave responds with a first segment of an L2CAP packet or with a NULL packet.
- Macro success is a macro poll resulting in a first segment of an L2CAP packet.
- Macro poll time is the time at which a macro poll takes place.
- Macro success time is the time at which a macro success takes place.
- Macro success ratio is the ratio of the total number of macro successes to the total number of macro polls.
- Micro poll is a poll to which a slave responds with a continuation segment of an L2CAP packet.
- Micro poll time is the time at which a micro poll takes place.

3.3.2.1 Markov chain analysis

By means of a Markov chain analysis, we present an expression for the probability P_{data_k} of data being available at the queue in slave k, which will be determined by the Data Availability Predictor (see Figure 3.4 and Section 3.3.2.3). For the sake of simplicity, we restrict ourselves to L2CAP packets that fit in a single baseband packet, and thus to the case in which micro polls do not occur. As a result, a poll always refers to a macro poll in this section. In Section 3.3.2.2, this restriction is released again.

If packet arrivals are generated according to a Poisson process we can give a formula for the probability P_{data_k} of data being available at the queue in slave k. In order to derive this formula we introduce the following Markov chain for one slave (see also Figure 3.5):

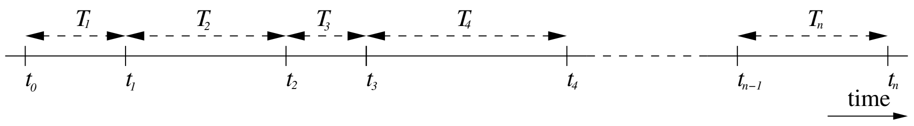


Figure 3.5: *Polling instants at a slave*

Let time t_0 be the last time this slave has been unsuccessfully polled (i.e., there was no packet available for transmission to the master). Time period T_1 is the time period between the last unsuccessful poll and first successful poll, and time periods $T_2..T_{n-1}$ are the time periods between two successful polls (i.e., there was at least one packet available for transmission at the slave at $t_1^-, t_2^- ... t_{n-1}^-$, where $t^- = t - \delta$ with $\delta \downarrow 0$). Now, let X_n be the number of L2CAP packets available for transmission at the slave at t_n^-. X_n is a time-inhomogeneous Markov process that represents the number of L2CAP packets available for transmission at the slave and is therefore non-negative. Furthermore, at most one packet can be drained when polling a slave, which results in the following relationship

$$\begin{array}{ll} X_n \geq X_{n-1} - 1 & \text{, for } X_{n-1} > 0, \\ X_n \geq 0 & \text{, for } X_{n-1} = 0. \end{array} \tag{3.9}$$

Whenever just before a poll, there is at least one packet available for transmission at the polled slave (i.e., $X_{n-1} = i > 0$), then one packet will be drained during that poll and the probability of $X_n = j$ packets ($j \geq i - 1$) being available for transmission just before the subsequent poll is the probability of $j - (i - 1)$ arrivals between the two polls. On the other hand, if just before a poll, no packet is available for transmission at the polled slave then no packet is drained during that poll and the probability of $X_n = j$ packets ($j \geq 0$) being available for transmission just before the subsequent poll is the probability of j arrivals between the two polls. This leads to the following transition probabilities

$$\begin{aligned} P_{ij}(n) &= P(X_n = j \mid X_{n-1} = i) && (3.10) \\ &= \begin{cases} e^{-\lambda T_n} \frac{(\lambda T_n)^{j-i+1}}{(j-i+1)!}, \quad j \geq (i-1), & i > 0, \\ 0, \quad j < i-1, & \\ e^{-\lambda T_n} \frac{(\lambda T_n)^{j}}{j!}, & i = 0, \\ 0, & i < 0, \end{cases} && (3.11) \end{aligned}$$

where λ is the arrival rate of packets. Furthermore, let us define $q_j(n)$ as the probability of j packets being available at the slave at t_n^-

$$q_j(n) = P(X_n = j \mid X_{n-1} \geq 1, ..., X_1 \geq 1, X_0 = 0). \tag{3.12}$$

Rewriting the probability in (3.12) as a joint probability gives

$$q_j(n) = \frac{P(X_n = j, X_{n-1} \geq 1 \mid X_{n-2} \geq 1, ..., X_1 \geq 1, X_0 = 0)}{P(X_{n-1} \geq 1 \mid X_{n-2} \geq 1, ..., X_1 \geq 1, X_0 = 0)}. \tag{3.13}$$

Breaking up the probability in the numerator in (3.13) into the possible ways of $X_{n-1} \geq 1$ gives

$$q_j(n) = \sum_{i=1}^{\infty} \frac{P(X_n = j, X_{n-1} = i \mid X_{n-2} \geq 1, ..., X_1 \geq 1, X_0 = 0)}{P(X_{n-1} \geq 1 \mid X_{n-2} \geq 1, ..., X_1 \geq 1, X_0 = 0)}. \tag{3.14}$$

Rewriting the probability in the numerator as a conditional probability gives

$$\begin{aligned} q_j(n) &= \\ & \sum_{i=1}^{\infty} \{ P(X_n = j \mid X_{n-1} = i, X_{n-2} \geq 1, ..., X_1 \geq 1, X_0 = 0) \\ & \cdot \frac{P(X_{n-1} = i \mid X_{n-2} \geq 1, ..., X_1 \geq 1, X_0 = 0)}{P(X_{n-1} \geq 1 \mid X_{n-2} \geq 1, ..., X_1 \geq 1, X_0 = 0)} \}. && (3.15) \end{aligned}$$

As $(X_n, n \geq 0)$ is a Markov chain, state X_n is only dependent on state X_{n-1}, and hence

$$q_j(n) = \frac{\sum_{i=1}^{\infty} \{ P_{ij}(n) P(X_{n-1} = i \mid X_{n-2} \geq 1, X_{n-3} \geq 1, ..., X_1 \geq 1, X_0 = 0) \}}{P(X_{n-1} \geq 1 \mid X_{n-2} \geq 1, X_{n-3} \geq 1, ..., X_1 \geq 1, X_0 = 0)}. \tag{3.16}$$

From (3.12) and (3.16), it follows that

$$q_j(n) = \frac{\sum_{i=1}^{\infty} q_i(n-1) P_{ij}(n)}{1 - q_0(n-1)}. \tag{3.17}$$

As $P_{ij}(n) = 0$ for $i > j + 1$ (see (3.11)), (3.17) can be rewritten as

$$q_j(n) = \frac{\sum_{i=1}^{j+1} q_i(n-1) P_{ij}(n)}{1 - q_0(n-1)}, \tag{3.18}$$

with

$$q_j(1) = P_{0j}(1) = e^{-\lambda T_1} \frac{(\lambda T_1)^j}{j!}. \tag{3.19}$$

The desired probability of data being available for transmission at time t_n^- at slave is

$$P_{data}(t_n^-) = 1 - q_0(n). \tag{3.20}$$

Expanding (3.20) for the first four polls while assuming the last unsuccessful poll took place at t_0 results in (3.21) to (3.24):

$$\begin{aligned}
P_{data}(t_1^-) &= 1 - q_0(1) \\
&= e^{(-\lambda T_1)}. && (3.21) \\
P_{data}(t_2^-) &= 1 - q_0(2) \\
&= \frac{\lambda T_1 e^{(-\lambda(T_1+T_2))}}{-1 + e^{(-\lambda T_1)}}. && (3.22) \\
P_{data}(t_3^-) &= 1 - q_0(3) \\
&= \frac{e^{(-\lambda(T_1+T_2+T_3))}\lambda^2 T_1(2T_2+T_1)}{-1 + e^{(-\lambda T_1)} + e^{(-\lambda(T_1+T_2))}\lambda T_1}. && (3.23) \\
P_{data}(t_4^-) &= 1 - q_0(4) \\
&= \{e^{(-\lambda(\sum_{i=1}^{4} T_i))}\lambda^3 T_1(2T_3T_2 + T_3T_1 + {T_2}^2 + T_1T_2 + \frac{{T_1}^2}{3})\} \\
&\quad \cdot\{(-2 + 2e^{(-\lambda T_1)} + 2e^{(-\lambda(T_1+T_2))}\lambda T_1 + 2e^{(-\lambda(T_1+T_2+T_3))}\lambda^2 T_1 T_2 \\
&\quad + e^{(-\lambda(T_1+T_2+T_3))}\lambda^2 {T_1}^2)\}^{-1}. && (3.24)
\end{aligned}$$

It can be seen that the expressions are becoming larger for increasing number of successive successful polls. Besides the size of the expression, the number of inter-poll times that must be remembered also (linearly) increases with the number of successive successful polls. We believe that an algorithm based on (3.18) is hard to implement. We will therefore make some simplifications.

An extensive search for a way to write $P_{data}(t_m)$ as a function of $P_{data}(t_1)..P_{data}(t_{m-1})$ or as a function of the numerators and the denominators of $P_{data}(t_1)..P_{data}(t_{m-1})$ did not give any result. Consequently, we decided to simplify the model.

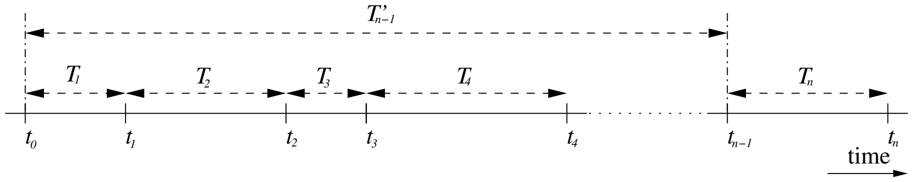


Figure 3.6: *Polling instants at a slave: a Simplification*

In order to get a simple approximation for the probability of data being available for transmission at a slave at time t_n^- we do not consider the separate time periods anymore, but the time period T'_{n-1} from the last unsuccessful poll to the last poll and the time period T_n from the last poll until now, where

$$T'_n = \sum_{i=1}^{n} T_i = \begin{cases} T_n, & n = 1. \\ T'_{n-1} + T_n, & n > 1, \end{cases} \tag{3.25}$$

Given the case depicted in Figure 3.6, we know that there was at least one arrival during T_1, at least two arrivals during $T_1 + T_2$, and so on. In other words, the probability of data being available for transmission at t_n is the probability of at least 1 packet arrival during T_n plus the probability of zero packet arrivals during T_n and more than $n-1$ packet arrivals during T_{n-1}, of which $n-1$ arrivals arrive such that the $n-1$ preceding polls are successful, i.e.,

$$P_{data}(t_n^-) = P'_{data}(t_n^-) + (1 - P'_{data}(t_n^-))P''_{data}(t_n^-), \tag{3.26}$$

where

$$P'_{data}(t_n^-) = P(\text{at least 1 arrival during } T_n), \tag{3.27}$$

and

$$P''_{data}(t_n^-) = \begin{cases} 0, & n = 1, \\ \frac{P(\text{at least } n \text{ arrivals during } T'_{n-1} \text{ given } n-1 \text{ successful polls})}{P(\text{at least } n-1 \text{ arrivals during } T'_{n-1} \text{ given } n-1 \text{ successful polls})}, & n > 1. \end{cases} \tag{3.28}$$

The approximation we apply concerns the $n-1$ arrivals during T'_{n-1}. We ignore the fact that each of these arrivals took place in a specific time period, i.e., the first arrival in T_1, the second arrival in $T_1 + T_2$ but after the first arrival, and so on. Because we ignore this in both the numerator and the denominator of (3.28), we conjecture and will show in Figure 3.7 that the effect of ignoring this will be acceptable. (3.28) will be replaced by

$$P''_{data}(t_n^-) = \begin{cases} 0, & n = 1, \\ \frac{P(\text{at least } n \text{ arrivals during } T'_{n-1})}{P(\text{at least } n-1 \text{ arrivals during } T'_{n-1})}, & n > 1. \end{cases} \tag{3.29}$$

Based on this approximation and taking into account the fact that the arrivals are generated by a Poisson process with parameter λ, the probability of data being available for transmission at t_n^- is

$$P_{data}(t_n^-) = \begin{cases} (1 - e^{-\lambda T_n}), & n = 1, \\ (1 - e^{-\lambda T_n}) + e^{-\lambda T_n} \frac{\sum_{i=n}^{\infty} e^{-\lambda T'_{n-1}} \frac{(\lambda T'_{n-1})^i}{i!}}{\sum_{i=n-1}^{\infty} e^{-\lambda T'_{n-1}} \frac{(\lambda T'_{n-1})^i}{i!}}, & n > 1, \end{cases} \tag{3.30}$$

and thus

$$P_{data}(t_n^-) = \begin{cases} (1 - e^{-\lambda T_n}), & n = 1, \\ (1 - e^{-\lambda T_n}) + e^{-\lambda T_n} \frac{1 - \sum_{i=0}^{n-1} e^{-\lambda T'_{n-1}} \frac{(\lambda T'_{n-1})^i}{i!}}{1 - \sum_{i=0}^{n-2} e^{-\lambda T'_{n-1}} \frac{(\lambda T'_{n-1})^i}{i!}}, & n > 1. \end{cases} \tag{3.31}$$

In order to examine the validity of (3.31) consider a polling system where the poller polls a queue as soon as P_{data} equals a predefined poll threshold p_{th}, and where infinitely small data packets arrive at the queue according to a Poisson process with rate λ. Only if the queue is non-empty and the poller polls that queue, then exactly one packet will be served. Based on the law of large numbers, if P_{data} exactly represents the probability of data being available for transmission, then the hit ratio of the poller (ratio of the number of successful polls to the total number of polls) should be equal to the probability $P_{data} = p_{th}$ at which the poller polls a slave.

We have written a simulation program where packets arrive at a queue according to a Poisson process with rate λ. The probability P_{data} of data being available at that queue is calculated using the simplified formula in (3.31). As soon as $P_{data} = p_{th}$, that queue is polled and the head of line packet, if available, is served. Figure 3.7 shows the hit ratio of the considered poller for different poll thresholds. Note that a poll sequence number of $n = m$ is a poll that is preceded by $m-1$ successful polls, i.e., a poll with sequence number $n = 1$ is the first poll after an unsuccessful poll. As we can see, the average hit ratio approaches the poll threshold p_{th} for each poll sequence number, which suggests that (3.31) is a good approximation for (3.20).

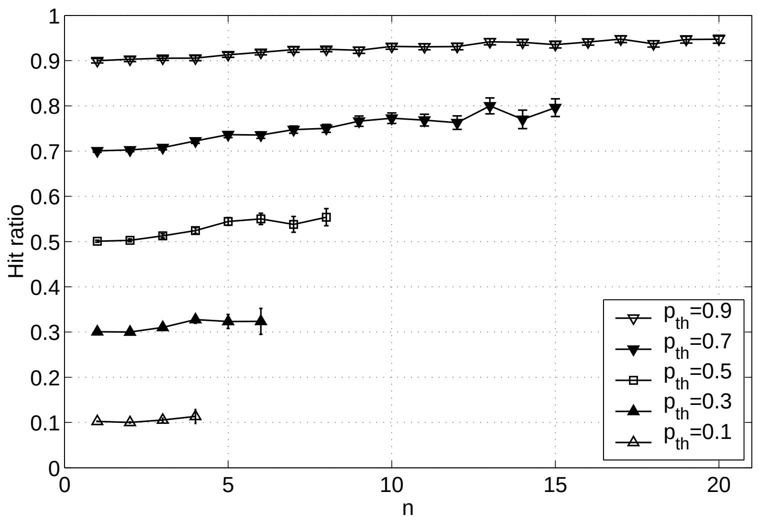


Figure 3.7: *Hit ratio as a function of poll sequence number n for different p_{th}*

3.3.2.2 Traffic Demand Estimator

Because an exponential inter-arrival time of packets is assumed, the Traffic Demand Estimator only needs to estimate the arrival rate of the L2CAP packets. For this the Traffic Demand Estimator calculates, with smoothing factor α_{tde}, an exponential moving average of the macro success rate taken over the last n_{tde} macro successes, i.e.,

$$TD_i^{\mathrm{e}} = (1-\alpha_{\mathrm{tde}})\hat{TD}_i^{\mathrm{e}} + \alpha_{\mathrm{tde}}\frac{n_{\mathrm{tde}}-1}{t_i^{\mathrm{ms}}[1]-t_i^{\mathrm{ms}}[n_{\mathrm{tde}}]}, \tag{3.32}$$

where $\hat{TD}_i^{\mathrm{e}}$ is the previous value of TD_i^{e}, and where $t_i^{\mathrm{ms}}[n]$ is the n-th last macro success time of slave i. Note that the arrival rate can only be determined correctly if the macro success ratio is less than unity. In case the macro success ratio equals unity, the actual arrival rate of packets is probably higher than the macro success rate, meaning that there is probably a queue overflow at the slave.

3.3.2.3 Data Availability Predictor

The Data Availability Predictor predicts whether a particular slave has a baseband packet waiting for transmission. In case the last poll to that slave resulted in a successful poll, the Data Availability Predictor checks whether the L2CAP packet, which the last baseband packet belongs to, is completely received. One way of doing this is looking at the payload size of the last received Baseband packet from that slave, where a full Baseband packet serves as an indication that it is likely that a continuation Baseband packet is waiting at the slave. If the Data Availability Predictor finds out that a continuation baseband packet is waiting at the particular slave, then it indicates that another baseband packet is waiting at the particular slave, i.e.,

$$P_{data_i} = 1. \tag{3.33}$$

In case the last poll to a particular slave resulted in a NULL packet to be received, or in case the Data Availability Predictor finds out that no continuation baseband packets are waiting at the particular slave, the Data Availability Predictor calculates the probability of data being available at the particular slave using (3.31), i.e.,

$$P_{data_i} = \begin{cases} (1-e^{-TD_i^{\mathrm{e}}T_n}), & n=1, \\ \\ (1-e^{-TD_i^{\mathrm{e}}T_n}) + e^{-TD_i^{\mathrm{e}}T_n}\frac{1-\sum_{i=0}^{n-1} e^{-TD_i^{\mathrm{e}}T'_{n-1}}\frac{(TD_i^{\mathrm{e}}T'_{n-1})^i}{i!}}{1-\sum_{i=0}^{n-2} e^{-TD_i^{\mathrm{e}}T'_{n-1}}\frac{(TD_i^{\mathrm{e}}T'_{n-1})^i}{i!}}, & n>1, \end{cases} \tag{3.34}$$

where $n-1$ is the number of macro successes since the last unsuccessful poll, and where T_n and T'_{n-1} are defined as is Figure 3.6. T_n and T'_{n-1} are extracted from poll result PR.

3.3.2.4 Fair Share Determinator

For the Best Effort case we want the poller to divide the available polls among the slaves according to the two-stages method explained in Section 3.1.2.3. Consider packets arriving at a each slave i with rate TD_i^{e}. The instantaneous fair share fs'_i of each slave i is determined according to the two-stages method explained in Section 3.1.2.3, where the demands are given by $d_i = TD_i^{\mathrm{e}}$ and where C_{tot} is the total amount of available polls. The fair share is determined by calculating, with smoothing factor α_{fs}, the exponential moving average of the instantaneous fair share, i.e.,

$$fs_i = (1-\alpha_{\mathrm{fs}})\hat{fs}_i + \alpha_{\mathrm{fs}}fs'_i, \tag{3.35}$$

where $\hat{fs}_i$ is the previous value of fs_i.

3.3.2.5 Fraction of Fair Share Determinator

The Fraction of Fair Share Determinator must first calculate the share s_i that slave i has been given. The instantaneous share s'_i is unity if the last poll was to slave i, and zero otherwise. Share s_i is the determined by taking, after each poll, and with smoothing factor α_s, the exponential moving average of the instantaneous share, i.e.,

$$s_i = (1-\alpha_s)\hat{s_i} + \alpha_s s'_i, \tag{3.36}$$

where $\hat{s_i}$ is the previous value of s_i. Using share s_i and fair share fs_i, the Fraction of Fair Share is defined as

$$ffs_i = \begin{cases} \frac{s_i}{fs_i}, & \text{if } s_i < fs_i, \\ 1, & \text{otherwise.} \end{cases} \tag{3.37}$$

3.3.2.6 Decision Maker

For each slave i a probability P_{data_i} of a Baseband packet being available for transmission to the master and a Fraction of Fair Share ffs_i is applied to the Decision Maker. Based on these inputs the Decision Maker decides which slave to poll next. It is clear that it is urgent to poll a slave with $P_{data} = 1$ and $ffs = 0$ while it is not needed to poll a slave with $P_{data} = 0$ and $ffs = 1$. The decision to make in the area between these two extremes depends on the policy.
We define a variable U_i for each slave i. We name it *poll urgency* and define it as

$$U_i = (1-\gamma_{\text{U}})P_{data_i} + \gamma_{\text{U}}(1-ffs_i), \quad 0 \leq \gamma_{\text{U}} \leq 1. \tag{3.38}$$

Most of the time, making a poller extremely efficient ($\gamma_{\text{U}} \to 0$) results in the poller being not fair (compare with a Round Robin poller with exhaustive service discipline), while basing poll decisions only on the fairness ($\gamma_{\text{U}} \to 1$) will lead to a poller that is not necessarily efficient. As a result, both the probabilities P_{data_i} and the Fractions of Fair Share ffs_i should have impact on the polling decisions. Therefore, the tuning variable γ_{U} is introduced to tune the impact of the probability P_{data_i} and the Fraction of Fair Share ffs_i on the poll urgency. Furthermore, the Decision Maker selects the slave i with the highest poll urgency value U_i.

Polling each time slave i that has the highest probability P_{data_i} will lead to an efficient and fair distribution of bandwidth among the slaves as long as there is at most one slave i with $P_{data_i} = 1$ at each poll moment. If it is predicted that more than one slave definitely has a Baseband packet available for transmission to the master then the fairness must also be considered. On the other hand, considering only the fairness while making a decision will lead to a polling scheme that is not necessarily efficient. As a result, the tuning variable γ_{U} should be greater than zero and less than one.

3.3.3 Simplification of PFP

In Section 3.3.2, the inter-arrival time of packets was assumed to be exponentially distributed. However, in reality, packets often arrive in bursts. Consequently, the presence of more baseband packets at a slave is likely if the last poll to that slave was successful.

The simplification of PFP that is proposed here concerns the simplification of the Data Availability Predictor in the sense that it assumes data being available at a particular slave after each successful poll to that slave. If the last poll to a particular slave is unsuccessful, then the time between that poll and the first arrival at that slave is assumed to be exponentially distributed with mean $\frac{1}{TD_i^{\mathrm{e}}}$, where TD_i^{e} is the arrival rate determined by the Traffic Demand Estimator as described in Section 3.3.2.2. The resulting probability P_{data_i} of data being available at a slave i is then given by

$$P_{data_i} = \begin{cases} (1 - e^{-TD_i^{\mathrm{e}}T}), & n = 1, \\ \\ 1, & n > 1, \end{cases} \tag{3.39}$$

where $n - 1$ is the number of polls to slave i since the last unsuccessful poll to that slave, and where T is the time duration since the last poll to that slave. As can be seen from (3.31) and (3.39), the determination of the probability of data being available is simplified. We will examine the influence of this simplification in Section 3.5.

3.3.4 Extension to PFP for duplex traffic handling

In order to make duplex traffic handling possible, two Slave Status Trackers are implemented for each master-slave pair, one for traffic from master to slave and one for traffic from slave to master. The major difference between the two slave status trackers is the Data Availability Predictor. In case of traffic from the master to the slaves, the poller knows whether data is available for transmission to a slave or not. Thus, the probability is either zero or one. The implementation of the other building blocks remains as described before. This includes the Decision Maker which still selects the slave with the highest poll urgency level (with respect to traffic from master to slave or traffic from slave to master) regardless of the poll urgency level with respect to traffic in the opposite direction.

3.4 Analysis of stability, efficiency, and fairness

As opposed to the polling sequences of the Fair Exhaustive Poller and the Predictive Fair Pollers, the polling sequence of the 1-limited Round Robin poller is independent of the traffic load. Consequently, the stability, efficiency, and fairness based on fraction of fair share of the 1-limited Round Robin poller can be obtained analytically. In this section we study the stability, efficiency, and fairness based on fraction of fair share of the 1-limited Round Robin poller taking the level of asymmetry of the load into account. In Section 3.5, we compare the simulation results of the 1-limited Round Robin poller with the analytically obtained results.

Consider N slaves and a master forming a piconet, while n_h ($0 < n_h < N$) slaves are highly loaded and the other slaves are lowly loaded. The load of each slave i is produced by a stochastic process that generates L2CAP packets with a rate λ_i, where

$$\lambda_1..\lambda_{N-n_h} = \lambda_l \geq 0, \tag{3.40}$$

and

$$\lambda_{N-n_h+1}..\lambda_N = \lambda_h, \tag{3.41}$$

where $\lambda_h > 0$ and $\lambda_h \geq \lambda_l$.
Furthermore, consider each L2CAP packet needs for transmission an average number of data slots $\overline{d}$, an average number of polls $\overline{p}$ and average number of wasted slots $\overline{w}$ (e.g., POLL packets). If the number of available time slots per second (total slot rate) is C_{tdd} then the load ρ is defined as

$$\rho = \frac{(\sum_{i=1}^{N} \lambda_i)\overline{d}}{C_{tdd}} = \frac{((N-n_h)\lambda_l + n_h\lambda_h)\overline{d}}{C_{tdd}}. \tag{3.42}$$

The level of asymmetry of the load in the piconet can be described with the coefficient of variation of the arrival rates which is defined here as ([Jai91][MR99])

$$\text{CV}(\boldsymbol{\lambda}) = \frac{\sqrt{\text{Var}}}{\mu} = \frac{\sqrt{\frac{\sum_{i=1}^{N}\lambda_i^2 - \frac{1}{N}(\sum_{i=1}^{N}\lambda_i)^2}{N-1}}}{\frac{1}{N}\sum_{i=1}^{N}\lambda_i}. \tag{3.43}$$

Substituting λ_i from (3.40) and (3.41) in (3.43) gives

$$\text{CV}(\boldsymbol{\lambda}) = \frac{N\sqrt{\frac{(N-n_h)n_h(\lambda_h-\lambda_l)^2}{N(N-1)}}}{(N-n_h)\lambda_l + n_h\lambda_h} = \frac{(\lambda_h - \lambda_l)\sqrt{\frac{N(N-n_h)n_h}{N-1}}}{(N-n_h)\lambda_l + n_h\lambda_h}. \tag{3.44}$$

Note that the maximum coefficient of variation of the arrival rates for which λ_l is non-negative is achieved when $\lambda_l = 0$ and is given by

$$\hat{\text{CV}}(\boldsymbol{\lambda}) = \sqrt{\frac{N(N-n_h)}{(N-1)n_h}}, \tag{3.45}$$

while the minimum coefficient of variation of the arrival rates is zero and is achieved when $\lambda_l = \lambda_h$.

Because of the assumption on the arrival rates $(\lambda_1, ..., \lambda_N)$ each combination of the load and the coefficient of variation of the arrival rates results in a unique combination of λ_l and λ_h. In other words, the arrival rates λ_l and λ_h can be written as a function of ρ and $\text{CV}(\boldsymbol{\lambda})$, i.e.,

$$\lambda_l = \frac{C_{tdd}\rho}{\overline{d}N}\left(1 - \text{CV}(\boldsymbol{\lambda})\sqrt{\frac{(N-1)n_h}{N(N-n_h)}}\right), \tag{3.46}$$

and

$$\lambda_h = \frac{C_{tdd}\rho}{\overline{d}N}\left(1 + \text{CV}(\boldsymbol{\lambda})\sqrt{\frac{(N-1)(N-n_h)}{Nn_h}}\right). \tag{3.47}$$

Substituting $\hat{\text{CV}}(\boldsymbol{\lambda})$ from (3.45) in (3.46) and (3.47) gives

$$\lambda_l = \frac{C_{tdd}\rho}{\overline{d}N}\left(1 - \frac{\text{CV}(\boldsymbol{\lambda})}{\hat{\text{CV}}(\boldsymbol{\lambda})}\right), \tag{3.48}$$

and

$$\lambda_h = \frac{C_{tdd}\rho}{\overline{d}N}\left(1 + \frac{\text{CV}(\boldsymbol{\lambda})}{\hat{\text{CV}}(\boldsymbol{\lambda})}\frac{N - n_h}{n_h}\right). \tag{3.49}$$

3.4.1 Stability of the 1-limited Round Robin poller

In order to study the stability of the the 1-limited Round Robin poller in Bluetooth we follow an approach described in [Kue79]. For that, we model the system as a multi-queue single-server system where:

- N queues are served by a single server in a cyclic manner with a 1-limited service discipline
- Queues $1, ..., (N - n_h)$ are loaded each with a burst[2] arrival rate $\lambda_l \geq 0$. We call these queues the lowly loaded queues.
- Queues $(N - n_h + 1), ..., N$ are loaded each with a burst arrival rate $\lambda_h \geq \lambda_l$. We call these queues the highly loaded queues.
- Each burst consists on average of $\overline{p}$ baseband packets. This models the fact that each L2CAP packet consists on average of $\overline{p}$ baseband packets.
- The switchover time u_i for queue i (also known as walking time or reply interval [Tak86]) is constant and equal to 2 time slots. Thus for each queue i

$$u_i = \frac{2}{C_{tdd}}, \tag{3.50}$$

  where C_{tdd} is the total slot rate.
- As an L2CAP packet needs on average $\overline{d} + \overline{w}$ time slots to be serviced, each baseband packet needs on average $\frac{\overline{d}+\overline{w}}{\overline{p}}$ time slots to be serviced. However, the switchover time already accounts for a part of the slots needed to serve a packet. Consequently, the modeled average service time h_i for a packet at queue i is

$$h_i = \frac{\frac{\overline{d}+\overline{w}}{\overline{p}}}{C_{tdd}} - u_i = \frac{\frac{\overline{d}+\overline{w}}{\overline{p}} - 2}{C_{tdd}}. \tag{3.51}$$

- The random cycle time is T_c with an average cycle time $c = \text{E}[T_c]$
- The average number of arriving baseband packets at queue i during a cycle is

$$m_i = \lambda_i \overline{p} c. \tag{3.52}$$

[2]Note that the burst is due to the fact that an L2CAP packet may comprise multiple baseband packets.

The system is said to be unstable if one or more queues are unstable. This is the case if either the highly loaded queues exceed their stability boundary (i.e., $m_h \geq 1$) or if the lowly loaded queues (and thus all the queues) exceed their stability boundary (i.e., $m_l \geq 1$ and $m_h \geq 1$). On the other hand, the queues are stable if they operate below their stability boundary.

If the highly loaded queues operate near to their stability boundary (i.e., $m_h \rightarrow 1$) then the average cycle length will be

$$c_h^* = \sum_{i=1}^{N} u_i + \sum_{i=1}^{N-n_h} \lambda_i \overline{p} c_h^* h_i + \sum_{i=N-n_h+1}^{N} h_i, \tag{3.53}$$

$$= \frac{2N}{C_{tdd}} + (N-n_h)\lambda_l \overline{p} c_h^* \frac{\frac{\overline{d}+\overline{w}}{\overline{p}} - 2}{C_{tdd}} + n_h \frac{\frac{\overline{d}+\overline{w}}{\overline{p}} - 2}{C_{tdd}}. \tag{3.54}$$

The first term of (3.53) accounts for the total switchover time during a cycle. Furthermore, the lowly loaded queues are stable and thus, on average, the number of packets from a lowly loaded queue that are served during a cycle equals the average number of packets that arrive at that queue during a cycle. This is accounted for by the second term of (3.53). The highly loaded queues operate near to their stability boundary and thus have a packet available for transmission each cycle. This is accounted for by the last term of (3.53).
Extracting c_h^* from (3.54) gives

$$c_h^* = \frac{2N + n_h(\frac{\overline{d}+\overline{w}}{\overline{p}} - 2)}{C_{tdd} - (N-n_h)\lambda_l(\overline{d}+\overline{w}-2\overline{p})}. \tag{3.55}$$

If the lowly loaded queues operate near to their stability boundary (i.e., $m_l \rightarrow 1$) then the highly loaded queues also operate near to their stability boundary (i.e., $m_h \rightarrow 1$). The average cycle length will then be

$$c_l^* = \sum_{i=1}^{N} u_i + \sum_{i=1}^{N} h_i, \tag{3.56}$$

$$= \frac{2N}{C_{tdd}} + N\frac{\frac{\overline{d}+\overline{w}}{\overline{p}} - 2}{C_{tdd}}, \tag{3.57}$$

$$= \frac{N}{C_{tdd}}(\frac{\overline{d}+\overline{w}}{\overline{p}}). \tag{3.58}$$

Again, the first term of (3.56) accounts for the total switchover time. Furthermore, all the slaves operate near to their stability boundary and thus have a packet available for transmission every time they are polled. This can be seen in the last term of (3.56).

The queues are served in a cyclic manner with a 1-limited service discipline, which means that at most one packet is served from each queue during a cycle. Consequently, the system is stable if at each queue, on average, at most one packet arrives during a cycle. This means that the system is stable if both

$$m_l = \lambda_l \overline{p} c_l^* < 1, \tag{3.59}$$

and

$$m_h = \lambda_h \overline{p} c_h^* < 1. \tag{3.60}$$

Rewriting (3.59) and (3.60) respectively gives

$$\lambda_l < \frac{C_{tdd}}{N(\overline{d} + \overline{w})}, \tag{3.61}$$

and

$$\lambda_h < \frac{C_{tdd} - (N - n_h)\lambda_l(\overline{d} + \overline{w} - 2\overline{p})}{n_h(\overline{d} + \overline{w}) + 2(N - n_h)\overline{p}}. \tag{3.62}$$

From (3.42) we know that:

$$\lambda_l = \frac{C_{tdd}\rho - n_h \lambda_h \overline{d}}{\overline{d}(N - nh)}. \tag{3.63}$$

Substituting λ_l from (3.63) in (3.62) gives

$$\lambda_h < \frac{C_{tdd} - \frac{\overline{d}+\overline{w}-2\overline{p}}{\overline{d}}(C_{tdd} - n_h \lambda_h \overline{d})}{n_h(\overline{d} + \overline{w}) + 2(N - n_h)\overline{p}}. \tag{3.64}$$

Solving the inequality from (3.64) gives

$$\lambda_h < \frac{C_{tdd}(\overline{d}(1 - \rho) + (2\overline{p} - \overline{w})\rho)}{2N\overline{d}\overline{p}}, \tag{3.65}$$

Substituting λ_h from (3.65) in (3.63) gives

$$\lambda_l > \frac{C_{tdd}(2N\rho\overline{p} - n_h(\overline{d}(1 - \rho) + (2\overline{p} - \overline{w})\rho))}{2(N - n_h)N\overline{d}\overline{p})}. \tag{3.66}$$

The conditions from (3.65) and (3.66) can be rewritten as a condition for the coefficient of variation of the arrival rates by substituting λ_l and λ_h in (3.44), i.e.,

$$\mathrm{CV}(\boldsymbol{\lambda}) < \sqrt{\frac{Nn_h}{(N-1)(N-n_h)}} \frac{\overline{d} + \overline{w}}{2\overline{p}} (\frac{\overline{d}}{\rho(\overline{d} + \overline{w})} - 1). \tag{3.67}$$

Substituting $\hat{\mathrm{CV}}(\boldsymbol{\lambda})$ from (3.45) in (3.67) we write

$$\mathrm{CV}(\boldsymbol{\lambda}) < \frac{n_h}{N - n_h} \hat{\mathrm{CV}}(\boldsymbol{\lambda}) \frac{\overline{d} + \overline{w}}{2\overline{p}} \left(\frac{\overline{d}}{\rho(\overline{d} + \overline{w})} - 1 \right). \tag{3.68}$$

In the previous subsection it is mentioned that the minimum value for the coefficient of variation of the arrival rates is zero. As a result of (3.68) and because $\hat{\mathrm{CV}}(\boldsymbol{\lambda})$, ρ, $\overline{d}$, $\overline{w}$, $\overline{p}$, n_h, and $(N - n_h)$ are non-negative, the following must also hold

$$\left(\frac{\overline{d}}{\rho(\overline{d} + \overline{w})} - 1 \right) > 0, \tag{3.69}$$

and thus

$$\rho < \frac{\overline{d}}{\overline{d} + \overline{w}}. \tag{3.70}$$

In other words, the minimum load for which no polling mechanism can be stable is defined as

$$\hat{\rho} = \frac{\overline{d}}{\overline{d} + \overline{w}}. \tag{3.71}$$

Note that this result can also be obtained by substituting λ_l from (3.61) in (3.62) and ultimately substituting λ_l and the new λ_h in (3.42).

Summarizing, we state that the system is stable if both requirements from (3.70) and (3.68) are met. Furthermore, as $\mathrm{CV}(\boldsymbol{\lambda}) \leq \hat{\mathrm{CV}}(\boldsymbol{\lambda})$, it can be seen that (3.68) is always met if

$$\frac{n_h}{N - n_h} \frac{\overline{d} + \overline{w}}{2\overline{p}} \left(\frac{\overline{d}}{\rho(\overline{d} + \overline{w})} - 1\right) > 1, \tag{3.72}$$

and thus if

$$\rho < \frac{n_h \overline{d}}{n_h(\overline{d} + \overline{w}) + 2(N - n_h)\overline{p}}. \tag{3.73}$$

3.4.2 Efficiency of the 1-limited Round Robin poller

We defined the efficiency in Section 3.1.2.1. Stating the definition differently, the efficiency is the ratio of the average service time[3] per cycle to the average cycle time. With respect to efficiency we distinguish four operation areas:

Operation area I

This operation area is defined as the operation area in which both the lowly loaded queues and the highly loaded queues are stable. The system is then stable, and hence operation area I is defined by (3.70) and (3.68).

The average cycle time in a stable system is

$$\begin{aligned} c &= \sum_{i=1}^{N} u_i + \sum_{i=1}^{N} (\lambda_i \overline{p} c h_i), & (3.74) \\ &= \frac{2N}{C_{tdd}} + c \left\{ (N - n_h)\lambda_l \overline{p} \frac{\frac{\overline{d}+\overline{w}}{\overline{p}} - 2}{C_{tdd}} + n_h \lambda_h \overline{p} \frac{\frac{\overline{d}+\overline{w}}{\overline{p}} - 2}{C_{tdd}} \right\}. & (3.75) \end{aligned}$$

The first term of (3.74) accounts for the total switchover time during a cycle. Furthermore, all the queues are stable and thus, on average, the number of packets from a queue that are served during a cycle equals the average number of packets that arrive at that queue during a cycle. This is accounted for by the second term of (3.74).

[3]Note that the real service time $\frac{\frac{\overline{d}}{\overline{p}}}{C_{tdd}}$ is now considered instead of the modeled service time of (3.51).

Extracting c from (3.75) gives

$$c = \frac{2N}{C_{tdd} - \{(N-n_h)\lambda_l + n_h\lambda_h\}(\overline{d} + \overline{w} - 2\overline{p})}. \tag{3.76}$$

According to our definition, the efficiency is

$$\eta = \frac{c\{((N-n_h)\lambda_l\overline{p} + n_h\lambda_h\overline{p})\}\frac{\frac{\overline{d}}{\overline{p}}}{C_{tdd}}}{c} = \frac{((N-n_h)\lambda_l + n_h\lambda_h)\overline{d}}{C_{tdd}} = \rho. \tag{3.77}$$

This was expected because in a stable system all the packets are served.

Operation area II

This operation area is defined as the operation area in which only the highly loaded queues are unstable, while

$$\rho < \hat{\rho}. \tag{3.78}$$

As $\lambda_l \leq \lambda_h$, unstable lowly loaded queue imply that (see also (3.61))

$$\lambda_h \geq \lambda_l \geq \frac{C_{tdd}}{N(\overline{d} + \overline{w})}. \tag{3.79}$$

Substituting λ_l and λ_h from (3.79) in (3.42) will show that (3.78) will not hold if the lowly loaded queues are unstable. Hence, whenever the system is unstable while (3.78) holds, it means that the highly loaded queues are the only unstable queues. Consequently, operation area II is defined by (3.78) and (see also (3.68))

$$\mathrm{CV}(\boldsymbol{\lambda}) \geq \frac{n_h}{N-n_h}\hat{\mathrm{CV}}(\boldsymbol{\lambda})\frac{\overline{d} + \overline{w}}{2\overline{p}}(\frac{\hat{\rho}}{\rho} - 1). \tag{3.80}$$

If only the highly loaded slaves are unstable then the average cycle time c_h^* is given by (3.55). According to our definition the efficiency will be

$$\eta = \frac{c_h^*\left\{(N-n_h)\lambda_l\overline{p}\frac{\frac{\overline{d}}{\overline{p}}}{C_{tdd}}\right\} + n_h\frac{\frac{\overline{d}}{\overline{p}}}{C_{tdd}}}{c_h^*}. \tag{3.81}$$

Substituting c_h^* from (3.55) in (3.81) gives

$$\eta = \frac{n_h\overline{d}}{n_h(\overline{d} + \overline{w}) + 2(N-n_h)\overline{p}} + \frac{2\lambda_l\overline{p}\frac{\overline{d}}{C_{tdd}}N(N-n_h)}{n_h(\overline{d} + \overline{w}) + 2(N-n_h)\overline{p}}. \tag{3.82}$$

Substituting λ_l from (3.48) in (3.82) gives

$$\eta = \frac{n_h\overline{d}}{n_h(\overline{d} + \overline{w}) + 2(N-n_h)\overline{p}} + \frac{2\overline{p}\rho(N-n_h)(1 - \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})})}{n_h(\overline{d} + \overline{w}) + 2(N-n_h)\overline{p}}. \tag{3.83}$$

Note that the minimum efficiency in an unstable system served by a 1-limited Round Robin poller is equal to the maximum load at which the system is stable regardless of the coefficient of variation of the arrival rates (see (3.73) and the first term of (3.83)).

Operation area III

This operation area is defined as the operation area in which only the highly loaded queues are unstable, and

$$\rho \geq \hat{\rho}. \tag{3.84}$$

If the lowly loaded queues approach their stability boundary then the average cycle time c_l^* is given by (3.58). The lowly loaded queues are stable as long as

$$\lambda_l \overline{p} c_l^* < 1, \tag{3.85}$$

thus as long as

$$\lambda_l < \frac{C_{tdd}}{N(\overline{d} + \overline{w})}. \tag{3.86}$$

From (3.42) we know that

$$\lambda_h = \frac{C_{tdd}\rho - (N - n_h)\lambda_l \overline{d}}{\overline{d} n_h}, \tag{3.87}$$

which implies that

$$\lambda_h > \frac{C_{tdd}\left\{N\rho(\overline{d} + \overline{w}) - (N - n_h)\overline{d}\right\}}{N n_h \overline{d}(\overline{d} + \overline{w})}. \tag{3.88}$$

The conditions from (3.86) and (3.88) can be rewritten as a condition for the coefficient of variation of the arrival rates by substituting λ_l and λ_h in (3.44), i.e.,

$$\mathrm{CV}(\boldsymbol{\lambda}) > \sqrt{\frac{N(N - n_h)}{(N - 1)n_h}} \left(1 - \frac{\overline{d}}{\rho(\overline{d} + \overline{w})}\right). \tag{3.89}$$

Substituting $\hat{\mathrm{CV}}(\boldsymbol{\lambda})$ from (3.45) and $\hat{\rho}$ from (3.71) in (3.89) gives

$$\mathrm{CV}(\boldsymbol{\lambda}) > \hat{\mathrm{CV}}(\boldsymbol{\lambda})(1 - \frac{\hat{\rho}}{\rho}). \tag{3.90}$$

In other words, the second operation area in which only the highly loaded queues are unstable (operation area III) is defined by (3.84) and (3.90).

In operation area III, only the highly loaded queues are unstable. Hence, the efficiency will be the same as in operation area II (see (3.83)).

Operation area IV

This operation area is defined as the operation area in which both the lowly loaded queues and the the highly loaded queues are unstable. According to the above, this is the case if

$$\rho \geq \hat{\rho} \quad \text{and} \quad \mathrm{CV}(\boldsymbol{\lambda}) \leq \hat{\mathrm{CV}}(\boldsymbol{\lambda})(1 - \frac{\hat{\rho}}{\rho}). \tag{3.91}$$

In that case the average cycle time is

$$\begin{aligned} c^* &= \sum_{i=1}^{N} u_i + \sum_{i=1}^{N} h_i, && (3.92) \\ &= \frac{2N}{C_{tdd}} + N\frac{\frac{\overline{d}+\overline{w}}{\overline{p}} - 2}{C_{tdd}}, && (3.93) \\ &= \frac{N}{C_{tdd}}(\frac{\overline{d}+\overline{w}}{\overline{p}}). && (3.94) \end{aligned}$$

Hence, the efficiency will be given by

$$\eta = \frac{N\frac{\frac{\overline{d}}{\overline{p}}}{C_{tdd}}}{c^*}. \quad (3.95)$$

Substituting c^* from (3.94) in (3.95) gives

$$\eta = \frac{\overline{d}}{\overline{d}+\overline{w}} = \hat{\rho}. \quad (3.96)$$

The above is summarized as

$$\eta = \begin{cases} \rho, & \text{Operation area I,} \\ \frac{n_h\overline{d}}{n_h(\overline{d}+\overline{w})+2(N-n_h)\overline{p}} + \frac{2\overline{p}\rho(N-n_h)(1-\frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})})}{n_h(\overline{d}+\overline{w})+2(N-n_h)\overline{p}}, & \text{Operation area II and III,} \\ \hat{\rho}, & \text{Operation area IV,} \end{cases} \quad (3.97)$$

while the operation areas are shown in Table 3.1. Note that operation area I is the operation area in which a system served by a 1-limited Round Robin poller is stable. Queues served by pollers that are able to handle traffic that is asymmetrically distributed among the slaves will also be stable in operation area II.

3.4.3 Fairness of the 1-limited Round Robin poller

We investigate the fairness based on fraction of fair share for the 1-limited Round Robin in the four operation areas listed in Table 3.1.

Operation area I and II

In these operation areas, the total load is less than the available resources. Consequently, each user should be allocated an amount of resources proportional to its offered load. As the number of polls per L2CAP packet is $\overline{p}$, the fair share of resources for each lowly loaded slave is given by

$$fs_l = \lambda_l \overline{p} k, \quad (3.98)$$

Operation area	Conditions	
I	$\rho < \hat{\rho}$	$\mathrm{CV}(\boldsymbol{\lambda}) < \frac{n_h}{n-n_h}\hat{\mathrm{CV}}(\boldsymbol{\lambda})\frac{\overline{d}+\overline{w}}{2\overline{p}}(\frac{\hat{\rho}}{\rho}-1)$
II		$\mathrm{CV}(\boldsymbol{\lambda}) \geq \frac{n_h}{n-n_h}\hat{\mathrm{CV}}(\boldsymbol{\lambda})\frac{\overline{d}+\overline{w}}{2\overline{p}}(\frac{\hat{\rho}}{\rho}-1)$
III	$\rho \geq \hat{\rho}$	$\mathrm{CV}(\boldsymbol{\lambda}) > \hat{\mathrm{CV}}(\boldsymbol{\lambda})(1-\frac{\hat{\rho}}{\rho})$
IV		$\mathrm{CV}(\boldsymbol{\lambda}) \leq \hat{\mathrm{CV}}(\boldsymbol{\lambda})(1-\frac{\hat{\rho}}{\rho})$

Table 3.1: *Operation areas of the 1-limited Round Robin poller*

while the fair share of resources for each highly loaded slave is given by

$$fs_h = \lambda_h \overline{p} k, \tag{3.99}$$

where $k \geq 1$ is a constant that depends on the amount of excess resources. Substituting λ_l and λ_h from (3.48) and (3.49) in (3.98) and (3.99), respectively, gives

$$fs_l = \frac{C_{tdd}\rho\overline{p}k}{\overline{d}N}(1 - \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})}), \tag{3.100}$$

and

$$fs_h = \frac{C_{tdd}\rho\overline{p}k}{\overline{d}N}(1 + \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})}\frac{N-n_h}{n_h}). \tag{3.101}$$

Let share s_i be the actual poll rate for each user i. As the 1-limited Round Robin poller equally divides the available polls among the users, each user will get the same share s, i.e.,

$$s_i = s, \quad \forall i. \tag{3.102}$$

According to (3.4), the fractions of fair share of resources will be given by

$$f\!fs_l = \frac{s_l}{fs_l} = \frac{\overline{d}Ns}{C_{tdd}\rho\overline{p}k}\frac{1}{(1 - \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})})}, \tag{3.103}$$

and

$$f\!fs_h = \frac{s_h}{fs_h} = \frac{\overline{d}Ns}{C_{tdd}\rho\overline{p}k}\frac{1}{(1 + \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})}\frac{N-n_h}{n_h})}. \tag{3.104}$$

According to (3.3), the fairness based on fraction of fair share is given by

$$f(f\!fs) = \frac{\left\{\hat{\mathrm{CV}}(\boldsymbol{\lambda}) + \mathrm{CV}(\boldsymbol{\lambda})(\frac{N}{n_h}-2)\right\}^2}{\left\{\hat{\mathrm{CV}}(\boldsymbol{\lambda}) + \mathrm{CV}(\boldsymbol{\lambda})(\frac{N}{n_h}-2)\right\}^2 + \mathrm{CV}(\boldsymbol{\lambda})^2(\frac{N}{n_h}-1)}. \tag{3.105}$$

Given that the considered system is operating in operation area I or II, it can be seen from (3.105) that the fairness based on fraction of fair share only depends on the number of users, the number of highly loaded users, and on the coefficient of variation of the arrival rates. In other words, and with respect to the considered system, the fairness based on fraction of fair share is independent of the total load, as long as system is operating in operation area I or II.

Operation area III

In operation area III, the total load is higher than the load that can be handled by any polling system. However, in this operation area, the load of the lowly loaded users is low such that their fair share equals their required share, i.e.,

$$fs_l = \frac{C_{tdd}\rho\overline{p}}{\overline{d}N}\left(1 - \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})}\right). \tag{3.106}$$

On the other hand, the load of the highly loaded users is high such that their required share cannot be granted. Consequently, the fair share for the highly loaded users equals the resources not allocated to the lowly loaded users divided by the number of highly loaded users, i.e.,

$$fs_h = \frac{1}{n_h}\left\{\frac{C_{tdd}\hat{\rho}\overline{p}}{\overline{d}} - (N - n_h)fs_l\right\}. \tag{3.107}$$

Substituting fs_l from (3.106) in (3.107) results in

$$fs_h = \frac{C_{tdd}\rho\overline{p}}{\overline{d}N}\left\{\frac{N}{n_h}\frac{\hat{\rho}}{\rho} - \frac{N-n_h}{n_h}\left(1 - \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})}\right)\right\}. \tag{3.108}$$

As the 1-limited Round Robin poller equally divides the available polls among the users, each user will get the same share s (see (3.102)). According to (3.4), the fractions of fair share of resources will be given by

$$ffs_l = \frac{s_l}{fs_l} = \frac{\overline{d}Ns}{C_{tdd}\rho\overline{p}}\frac{1}{(1 - \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})})}, \tag{3.109}$$

and

$$ffs_h = \frac{s_h}{fs_h} = \frac{\overline{d}Ns}{C_{tdd}\rho\overline{p}}\frac{1}{\frac{N}{n_h}\frac{\hat{\rho}}{\rho} - \frac{N-n_h}{n_h}(1 - \frac{\mathrm{CV}(\boldsymbol{\lambda})}{\hat{\mathrm{CV}}(\boldsymbol{\lambda})})}. \tag{3.110}$$

According to (3.3), the fairness based on fraction of fair share is then given by

$$f(ffs) = \frac{\left\{\hat{\mathrm{CV}}(\boldsymbol{\lambda}) + \xi\mathrm{CV}(\boldsymbol{\lambda})(\frac{N}{n_h} - 2)\right\}^2}{\left\{\hat{\mathrm{CV}}(\boldsymbol{\lambda}) + \xi\mathrm{CV}(\boldsymbol{\lambda})(\frac{N}{n_h} - 2)\right\}^2 + (\xi\mathrm{CV}(\boldsymbol{\lambda}))^2(\frac{N}{n_h} - 1)}, \tag{3.111}$$

where

$$\xi = \frac{\rho}{\hat{\rho}} - \frac{\hat{\mathrm{CV}}(\boldsymbol{\lambda})}{\mathrm{CV}(\boldsymbol{\lambda})}(\frac{\rho}{\hat{\rho}} - 1). \tag{3.112}$$

As opposed to the fairness based on fraction of fair share in operation area I and II, the fairness in operation area III also depends on the ratio of the load ρ to the minimum load for which no polling system can be stable $\hat{\rho}$.

Operation area IV

In this operation area, none of the required loads can be afforded. Consequently, all the users should be allocated the same fair share, i.e.,

$$fs_l = fs_h = \frac{C_{tdd}\hat{\rho}\overline{p}}{\overline{d}N}. \tag{3.113}$$

As the 1-limited Round Robin poller equally divides the available polls among the users, each user will get the same share s (see (3.102)). According to (3.4), the fractions of fair share of resources will be given by

$$ffs_l = ffs_h = \frac{s_h}{fs_h} = \frac{\overline{d}Ns}{C_{tdd}\hat{\rho}\overline{p}}. \tag{3.114}$$

According to (3.3), the fairness based on fraction of fair share is then given by

$$f(ffs) = 1. \tag{3.115}$$

3.5 Simulation studies

We evaluate the Predictive Fair Poller (see Section 3.3), the Simplified Predictive Fair Poller (see Section 3.3.3), the Fair Exhaustive Poller (see section 3.2.2), and the 1-limited Round Robin poller (see Section 3.2.1) by means of simulations. The Predictive Fair Poller uses (3.33) and (3.34) for the calculation of P_{data}, whereas the Simplified Predictive Fair Poller uses (3.39) for the same purpose. Through these simulations we show the shortcomings of the 1-limited Round Robin poller and show that both the Fair Exhaustive Poller and the Predictive Fair Pollers do not have these shortcomings. We compare the Predictive Fair Poller with the 1-limited Round Robin poller because it is the most simple poller, and with the Fair Exhaustive Poller because we believe that Fair Exhaustive Poller is the best alternative poller available.

We present simulation results of these pollers in three Best Effort traffic scenarios. In the first scenario, fixed-size IP packets[4] that fit in a single baseband are generated with exponential inter-arrival times. In the second scenario, variable-size IP packets that may comprise multiple baseband packets are generated with exponential inter-arrival times. Finally, the third simulation scenario is an FTP/TCP scenario in which the traffic is generated by FTP/TCP sources.

The simulation tool we used is Network Simulator (ns2) [ns2] with CMU wireless extensions [CMU99] based Bluetooth extensions [Nie00] from Ericsson Switchlab together with our ns2 implementation of both the Fair Exhaustive Poller and the Predictive Fair Pollers.

[4]In this dissertation, it is assumed that the IP layer is situated directly on top of the L2CAP layer, and that each IP packet is transported using one L2CAP packet

In Section 3.5.1, we will briefly describe the simulation model. In Section 3.5.2 we present the simulation results of the Poisson scenario with fixed IP packet sizes. We present the simulation results of the Poisson scenario with variable IP packet sizes in Section 3.5.3. Finally, we present the simulation results of the FTP/TCP scenario in Section 3.5.4

3.5.1 Description of the Simulation Model

Simulations are performed in three best effort scenarios. In all the scenarios, the network setup of Figure 3.8 is used. In the first two scenarios, a Poisson process at each slave Si generates data (IP packets) that is destined for the master M, while the wired nodes W0..W7 are not involved in these two scenarios. In the third scenario, that we call the FTP/TCP scenario, a TCP connection is setup between each slave Si and wired node Wi, and FTP is used to transports data from slave Si to wired node Wi.

Simulations are performed in scenarios in which there are lowly loaded slaves and highly loaded slaves. In the first two scenarios this is achieved by setting the rate of the Poisson processes at each slave. In the third scenario, this is achieved by choosing the right capacity for each link Li.

For each of the simulated pollers, the maximum inter-poll time is set at $T_{\mathrm{poll}} = 0.2$ sec. With respect to the Fair Exhaustive Poller, a slave is moved to the list of inactive slaves after a single unsuccessful poll. With respect to the Predictive Fair Pollers, the parameters of the traffic demand estimator (see Section 3.3.2.2) are $\alpha_{\mathrm{tde}} = 0.05$ and $n_{\mathrm{tde}} = 10$. Furthermore, the parameters of the fair share determinator (see Section 3.3.2.4) and the fraction of fair share determinator (see Section 3.3.2.5) are $\alpha_{\mathrm{fs}} = \alpha_{\mathrm{s}} = 0.01$. Finally, the parameter of the decision maker (see Section 3.3.2.6) is $\gamma_{\mathrm{U}} = 0.5$. The aforementioned values are chosen based on the results of initial test simulations.

Note that in this chapter, the simulations are performed in a perfect radio environment where no transmission errors occur and where retransmissions are not needed. Furthermore, no paging or inquiry procedures take place after the initial part of the simulations.

3.5.2 The Poisson scenario with fixed IP packet sizes

3.5.2.1 Purpose of the simulation

The purpose of simulating in this simulation scenario is to show that the 1-limited Round Robin poller is not always able to handle traffic that is asymmetrically distributed among the slaves. Furthermore, by means of this simulation, we investigate whether the Predictive Fair Pollers perform at least as good as the Fair Exhaustive Poller.

By using fixed-size IP packets that fit in a single baseband packet, also the baseband packets arrive with exponentially distributed inter-arrival times. The simplified Predictive Fair Poller assumes that baseband packets arrive in bursts. This simulation shows the effect of this assumption on the performance of the simplified Predictive Fair Poller.

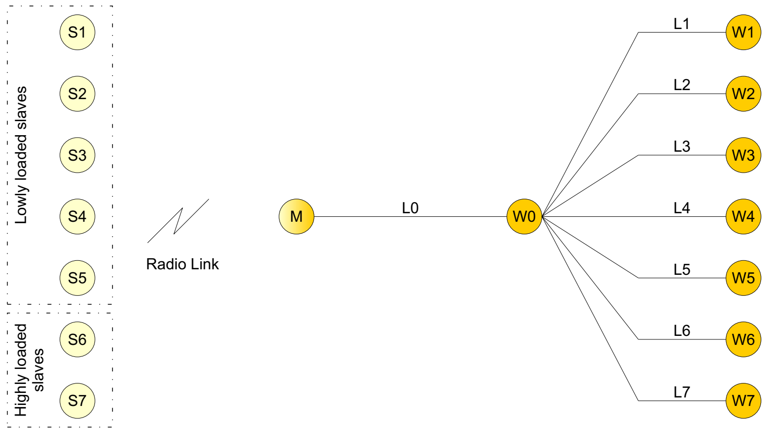


Figure 3.8: *Network setup for the simulations*

3.5.2.2 Description of the simulation scenario

In this scenario we simulate under the following assumptions (see also Figure 3.8):

- Seven slaves (S1,...,S7) and a master (M) form a piconet, while two of the seven slaves are highly loaded, i.e.,
$$N = 7 \quad \text{and} \quad n_h = 2. \tag{3.116}$$
- There is only upstream traffic, i.e., from the slaves to the master.
- The slaves generate IP packets according to Poisson processes with arrival rates $\lambda_1, ..., \lambda_7$, while
$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = \lambda_l \geq 0, \tag{3.117}$$
and
$$\lambda_6 = \lambda_7 = \lambda_h \geq \lambda_l. \tag{3.118}$$
- The available Bluetooth Baseband packet type is DH1 with a maximum payload size of 27 bytes.
- Each IP packet fits in one DH1 baseband packet and needs one POLL packet. This results in an average of $\overline{d} = 1$ data slot, an average of $\overline{p} = 1$ polls and an average of $\overline{w} = 1$ wasted time slots per IP packet.
- The number of slots per second available for transmission of data or for transmission of a POLL or NULL packet is
$$C_{tdd} = 1600 \text{ slots/sec.} \tag{3.119}$$
- According to (3.42) the load of the piconet is given by
$$\rho = \frac{(5\lambda_l + 2\lambda_h)\overline{d}}{C_{tdd}} = \frac{(5\lambda_l + 2\lambda_h)}{1600}, \tag{3.120}$$

while according to (3.71) the minimum load for which no poller can be stable is given by

$$\hat{\rho} = \frac{1}{2}. \tag{3.121}$$

- According to (3.44) the coefficient of variation of the arrival rates is given by

$$\mathrm{CV}(\boldsymbol{\lambda}) = \frac{\sqrt{\frac{35}{3}}(\lambda_h - \lambda_l)}{5\lambda_l + 2\lambda_h}, \tag{3.122}$$

while according to (3.45) the maximum coefficient of variation of the arrival rates is

$$\hat{\mathrm{CV}}(\boldsymbol{\lambda}) = \sqrt{\frac{35}{12}}. \tag{3.123}$$

- According to (3.48) and (3.49) the arrival rates are given by

$$\lambda_l = \frac{1600\rho}{7}(1 - \sqrt{\frac{12}{35}}\mathrm{CV}(\boldsymbol{\lambda})), \tag{3.124}$$

and

$$\lambda_h = \frac{1600\rho}{7}(1 + \frac{5}{2}\sqrt{\frac{12}{35}}\mathrm{CV}(\boldsymbol{\lambda})). \tag{3.125}$$

The simulations are performed in a lowly loaded piconet ($\rho = 0.1$) and in a highly loaded piconet ($\rho = 0.45$). We compare the 1-limited Round Robin poller (RR), the Fair Exhaustive Poller (FEP), the Predictive Fair Poller (PFP), and the simplified Predictive Fair Poller (simplified PFP) taking into account the efficiency (η), the total mean response time (MRT), the fairness based on fraction of fair share ($f(ffs)$), and the fairness based on inverse fraction of reference waiting time ($f(ifrw)$). These performance metrics are explained in Section 3.1.2

3.5.2.3 Expectations for the efficiency

According to Section 3.4.2, the efficiency of the Round Robin poller will be given by (confer (3.97))

$$\eta = \begin{cases} \rho, & \text{Operation area I,} \\ \frac{1}{7} + \frac{5}{7}\rho(1 - \sqrt{\frac{12}{35}}\mathrm{CV}(\boldsymbol{\lambda})), & \text{Operation area II and III,} \\ \frac{1}{2}, & \text{Operation area IV.} \end{cases} \tag{3.126}$$

where the operation areas are shown in Table 3.2 (confer Table 3.1) and Figure 3.9. This figure shows in which operation area the 1-limited Round Robin poller operates given a particular load ρ and coefficient of variation of the arrival rates $\mathrm{CV}(\boldsymbol{\lambda})$. The system served by a 1-limited Round Robin poller is only stable in operation area I.
As can be seen in Figure 3.9 and as can be calculated using (3.73), the system served by the 1-limited Round Robin poller is stable regardless of the coefficient of variation of the arrival rates ($\mathrm{CV}(\boldsymbol{\lambda})$) if

Operation area	Conditions	
I	$\rho < \frac{1}{2}$	$\mathrm{CV}(\boldsymbol{\lambda}) < \frac{2}{5}\sqrt{\frac{35}{12}}(\frac{1}{2\rho} - 1)$
II		$\mathrm{CV}(\boldsymbol{\lambda}) \geq \frac{2}{5}\sqrt{\frac{35}{12}}(\frac{1}{2\rho} - 1)$
III	$\rho \geq \frac{1}{2}$	$\mathrm{CV}(\boldsymbol{\lambda}) \geq \sqrt{\frac{35}{12}}(1 - \frac{1}{2\rho})$
IV		$\mathrm{CV}(\boldsymbol{\lambda}) < \sqrt{\frac{35}{12}}(1 - \frac{1}{2\rho})$

Table 3.2: *Operation areas of the 1-limited Round Robin poller in the Poisson scenario with fixed IP packet sizes*

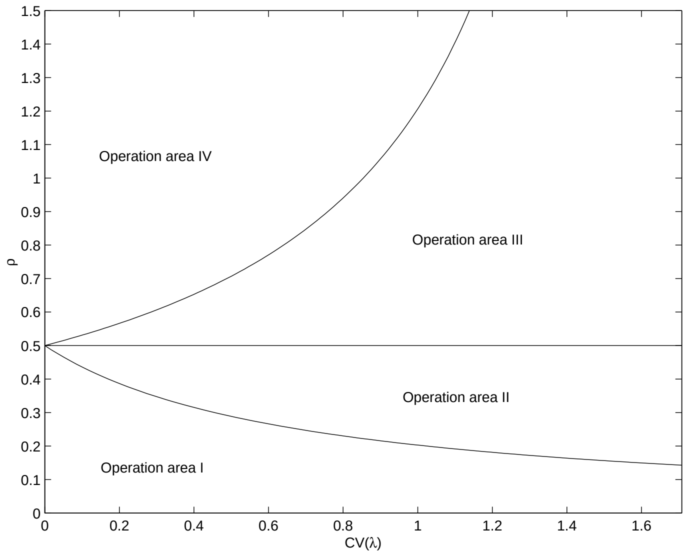


Figure 3.9: *Operation areas of the 1-limited Round Robin poller in the Poisson scenario with fixed IP packet sizes*

$$\rho < \frac{n_h \overline{d}}{n_h(\overline{d} + \overline{w}) + 2(N - n_h)\overline{p}} = \frac{1}{7}. \tag{3.127}$$

As a result, the system served by the 1-limited Round Robin poller will be stable in the lowly loaded piconet ($\rho = 0.1$) regardless of the coefficient of variation of the arrival rates ($\mathrm{CV}(\boldsymbol{\lambda})$), and a maximum efficiency of $\eta = \rho = 0.1$ is expected to be achieved. However, in the highly loaded piconet ($\rho = 0.45$), the system will be stable as long as (see also operation area I in table 3.2)

$$\mathrm{CV}(\boldsymbol{\lambda}) < \frac{2}{5}\sqrt{\frac{35}{12}}\left(\frac{1}{2\rho} - 1\right) = 0.076. \tag{3.128}$$

Table 3.3 shows the calculated values for the efficiency (η) as a function of the coefficient of variation of the arrival rates ($\mathrm{CV}(\boldsymbol{\lambda})$) in a highly loaded piconet ($\rho = 0.45$).

$\mathrm{CV}(\boldsymbol{\lambda})$	0	0.2	0.4	0.6	0.8	1	1.2
η	ρ	0.43	0.39	0.35	0.31	0.28	0.24
Capacity loss	0%	5.2%	13.6%	21.9%	30.3%	38.6%	47%

Table 3.3: *Expected efficiency (η) of the 1-limited Round Robin Poller as function of $\mathrm{CV}(\boldsymbol{\lambda})$ in a highly loaded piconet ($\rho = 0.45$)*

Because the Fair Exhaustive Poller, the Predictive Fair Poller and the simplified Predictive Fair Poller are assumed to be able of handling traffic that is asymmetrically distributed among the slaves, we expect that these pollers will be stable as long as they operate in operation area I or II, i.e., as long as

$$\rho < \hat{\rho} = \frac{1}{2}. \tag{3.129}$$

As mentioned before, we simulate in a lowly loaded piconet ($\rho = 0.1$) and in highly loaded piconet ($\rho = 0.45$). As (3.129) holds in both cases, the system operates in operation area I or II in both cases. Consequently, we expect the Fair Exhaustive Poller, the Predictive Fair Poller and the simplified Predictive Fair Poller to be stable in both cases. This implies that they achieve the maximum efficiency of $\eta = \rho$.

3.5.2.4 Expectations for the fairness

As the system operates in operation area I or II, the value of the fairness based on fraction of fair share for the 1-limited Round Robin poller will be given by (3.105), and will be independent of the total load ρ. Figure 3.10 shows the fairness based on fraction of fair share as a function of the coefficient of variation of the arrival rates for the 1-limited Round Robin poller. For instance, the solid line corresponds to the fairness in case the system operates in operation area I or II. In the remaining cases, the fairness also depends on the total load.

The Fair Exhaustive Poller polls slaves that it assumes to be active. In a lowly loaded piconet, the Fair Exhaustive Poller will most of the time assume slaves to be inactive. As a result, the Fair Exhaustive Poller will often poll the slaves in a 1-limited Round Robin manner, leading to a fairness that is near to the fairness achieved by the Round Robin poller. In a highly loaded piconet, the Fair Exhaustive Poller can distinguish better between active slaves and inactive slaves. As a result the Fair Exhaustive poller will most of the time poll the active slaves in a Round Robin manner. In other words, the Fair Exhaustive Poller will equally divide most of the bandwidth among the active slaves, while allocating less bandwidth to the inactive slaves. Consequently, its fairness based on fraction of fair share will approach one.

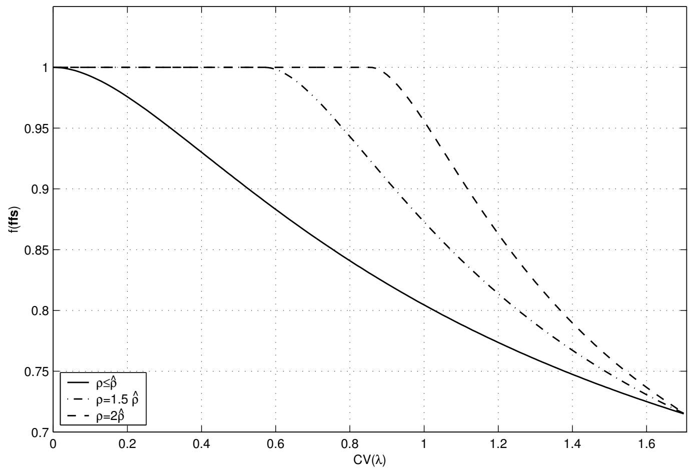


Figure 3.10: *Fairness based on fraction of fair share as a function of the coefficient of variation of the arrival rates for the 1-limited Round Robin poller*

The Predictive Fair Poller and the simplified Predictive Fair Poller poll slaves proportional to their fair share of resources. Hence, the expected fairness based on fraction of fair share approaches one.

3.5.2.5 Simulation results

In this section we discuss the simulation results of the 1-limited Round Robin poller, the Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller in a lowly loaded piconet ($\rho = 0.1$) and in a highly loaded piconet ($\rho = 0.45$).

Lowly loaded piconet ($\rho = 0.1$)
The four polling mechanisms perform equally well with respect to the efficiency (η) as function of the coefficient of variation of the arrival rate. They achieve the maximum efficiency of $\eta = \rho$, which conforms the expectations.

Figure 3.11[5] shows the fairness based on fraction of fair share ($f(ffs)$) for the four polling mechanisms in a lowly loaded piconet. The fairness based on fraction of fair share of the 1-limited Round Robin poller conforms the expectations drawn in Figure 3.10. It can be seen that the fairness of the Fair Exhaustive Poller is approaching the fairness of the 1-limited Round Robin poller. The reason for this is that, because of the low load, the Fair Exhaustive Poller assumes the slaves most of the time to be inactive. Consequently, it polls all the slaves, most of the time, in a 1-limited Round Robin manner. The Predictive Fair Pollers poll each slave at a poll rate proportional to its fair share of resources. Hence, their fairness based on fraction of fair share approaches one.

[5] In this chapter, the figures containing simulation results show the (two-sided) 95% confidence interval. However, as the simulation times are long, the confidence intervals are often very small.

Figure 3.12 shows the total mean response time (MRT) for the four polling mechanisms in a lowly loaded piconet. The total mean response time MRT of the 1-limited Round Robin poller is increasing for higher coefficient of variation of the arrival rates $(\mathrm{CV}(\boldsymbol{\lambda}))$. The reason for this is that the arrival rates at the highly loaded slaves increase for increasing $\mathrm{CV}(\boldsymbol{\lambda})$, while the poll rate to the slaves remains unchanged (similar to increasing utilization in a queueing system). Consequently, the mean response time at the highly loaded slaves increases for increasing $(\mathrm{CV}(\boldsymbol{\lambda}))$. On the other hand, the mean response time of the lowly loaded slaves decreases for increasing values of $\mathrm{CV}(\boldsymbol{\lambda})$. However, a decrease of the load of the lowly loaded slaves means a $\frac{N-n_h}{n_h} = \frac{5}{2}$ times higher increase of the load at the highly loaded slaves. Furthermore, the higher the value of $\mathrm{CV}(\boldsymbol{\lambda})$, the relatively higher the number of packets from the highly loaded slave. This explains the increasing total mean response time, in case of the 1-limited Round Robin poller, for increasing values of $\mathrm{CV}(\boldsymbol{\lambda})$. The Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller adapt their poll rate to the load. Hence, they achieve a total mean response time that is lower than the one achieved by the 1-limited Round Robin poller.

Figure 3.13 shows the fairness based on inverse fraction of reference mean waiting time for the four polling mechanisms in a lowly loaded piconet. According to (3.6) and (3.7), and given the simulation description, the reference mean waiting time for a slave i is

$$W_{\mathrm{R}i} = \frac{\frac{1}{fs_i}\frac{\lambda_i}{fs_i}}{2(1-\frac{\lambda_i}{fs_i})}, \tag{3.130}$$

where fs_i is the fair share of resources that slave i should get. As the system operates in operation area I or II, the fair shares of resources fs_i will be proportional to the arrival rates λ_i. Given the same total arrival rate, the term $\frac{\lambda_i}{fs_i}$ will be constant and independent of the coefficient of variation of the arrival rates. Consequently, the reference mean waiting time of a slave will be inversely proportional to the fair share of that slave and thus to the arrival rate to that slave. However, the 1-limited Round Robin poller polls the slaves independently of their load. Hence, given a particular total load, the lower the load of a slave, the lower its mean waiting time. On the other hand, the higher the load of a slave, the higher its mean waiting time. As the reference mean waiting time of a slave is inversely proportional to the load of that slave, this explains the decreasing fairness based on inverse fraction of reference mean waiting time of the 1-limited Round Robin poller for increasing values of $\mathrm{CV}(\boldsymbol{\lambda})$.

Because of the low load, the Fair Exhaustive Poller is most of the time polling the slaves in a 1-limited Round Robin manner, which explains the low fairness based on inverse fraction of reference mean waiting time for the Fair Exhaustive Poller. The Predictive Fair Poller and the simplified Predictive Fair Poller poll each slave at a poll rate proportional to its fair share. Given the same total load, a higher load of a slave leads to a higher fair share, and thus to a higher poll rate. Consequently, the mean waiting time for each slave will increase for decreasing load and vice-versa. As the reference mean waiting time is inversely proportional to the load, the fairness based on inverse fraction of reference waiting time will be higher than for the 1-limited Round Robin poller and the Fair Exhaustive Poller.

Highly loaded piconet ($\rho = 0.45$)

Figure 3.14 shows the efficiency for the four polling mechanisms in a highly loaded piconet. As expected (see Table 3.3), the 1-limited Round Robin poller becomes inefficient for increasing values of $\mathrm{CV}(\boldsymbol{\lambda})$. The Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller adapt to the different loads, and hence achieve a maximum efficiency of $\eta = \rho = 0.45$.

Figure 3.15 shows the fairness based on fraction of fair share for the four polling mechanisms in a highly loaded piconet. The fairness based on fraction of fair share of the 1-limited Round Robin poller conforms the expectations drawn in Figure 3.10. It can be seen that the fairness based on fraction of fair share of the Fair Exhaustive Poller is now approaching one. The reason for this is that the Fair Exhaustive Poller can now distinguish between active nodes and inactive nodes. Consequently, the Fair Exhaustive Poller does not have to poll all the slaves in a Round Robin manner. As mentioned before, the Predictive Fair Poller and the simplified Predictive Fair Poller poll each slave at a poll rate proportional to its fair share of resources. Hence, their fairness based on fraction of fair share approaches one.

Figure 3.16 shows the total mean response time for the four polling mechanisms in a highly loaded piconet. From (3.128) we know that the considered system served by a 1-limited Round Robin polling mechanism becomes unstable if $\mathrm{CV}(\boldsymbol{\lambda}) \geq 0.076$. This can also be seen in Figure 3.16. Furthermore, we see that the Predictive Fair Pollers achieve a response time that is slightly lower than the one achieved by the Fair Exhaustive Poller. This is caused by the fact that the Fair Exhaustive Poller checks data availability at inactive slaves after fixed intervals, whereas the Predictive Fair Pollers check data availability at slaves depending on their probability of having data available for transmission. The simplified Predictive Fair Poller achieves a response time that is slightly lower than the one achieved by the Predictive Fair Poller. The reason for this is that the simplified Predictive Fair Poller alway assumes availability of data at a slave if the last poll to that slave was a successful poll. Consequently, possibly available packets will experience a lower response time.

Figure 3.17 shows the fairness based on inverse fraction of reference waiting time for the four polling mechanisms in a highly loaded piconet. When the considered system is served by a 1-limited Round Robin polling mechanism, the highly loaded slaves become unstable as soon as $\mathrm{CV}(\boldsymbol{\lambda}) \geq 0.076$. With respect to inverse fraction of reference waiting time, this means that the 1-limited Round Robin polling mechanism then becomes unfair to two of the seven slaves. Hence, a fairness based on inverse fraction of reference waiting time of $f(ifrw) \approx \frac{5}{7}$ is achieved for $\mathrm{CV}(\boldsymbol{\lambda}) \geq 0.076$.

The Fair Exhaustive Poller polls slaves that are assumed to be active. However, the resulting poll rate is not necessarily proportional to the fair shares. As the load at the highly loaded slaves becomes very high for increasing $\mathrm{CV}(\boldsymbol{\lambda})$, a slightly lower fraction of fair share results in a much higher waiting time. Hence, the resulting fairness based on inverse fraction of reference waiting time will also become low for increasing $\mathrm{CV}(\boldsymbol{\lambda})$. The Predictive Fair Poller and the simplified Predictive Fair Poller give each slave a poll rate proportional to their fair share. Consequently, the mean waiting time for each slave will increase for decreasing load and vice-versa. As the mean reference waiting time is inversely proportional to the load, the fairness based on inverse fraction of reference waiting time will be higher than for the 1-limited Round Robin poller and the Fair Exhaustive Poller.

3.5.2.6 Conclusions of scenario I

The simulation results showed that the Round Robin poller is unable to handle traffic that is asymmetrically distributed among the slaves. Furthermore, the results showed that both the Predictive Fair Poller and the Simplified Predictive Fair Poller outperform both the 1-limited Round Robin poller and the Fair Exhaustive Poller. Finally, the simulation results showed that the Predictive Fair Poller and the simplified Predictive Fair Poller achieve a comparable performance, which justifies the simplifications made in Section 3.3.3.

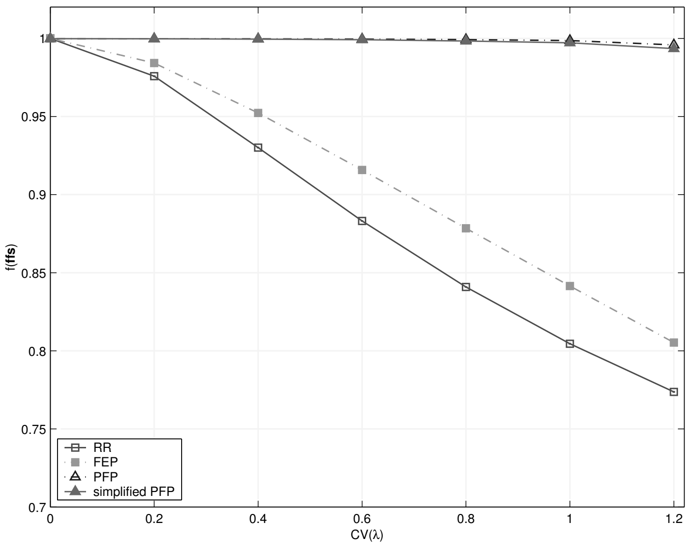


Figure 3.11: *Fairness based on fraction of fair share ($f(ffs)$) in a lowly loaded piconet ($\rho = 0.1$)*

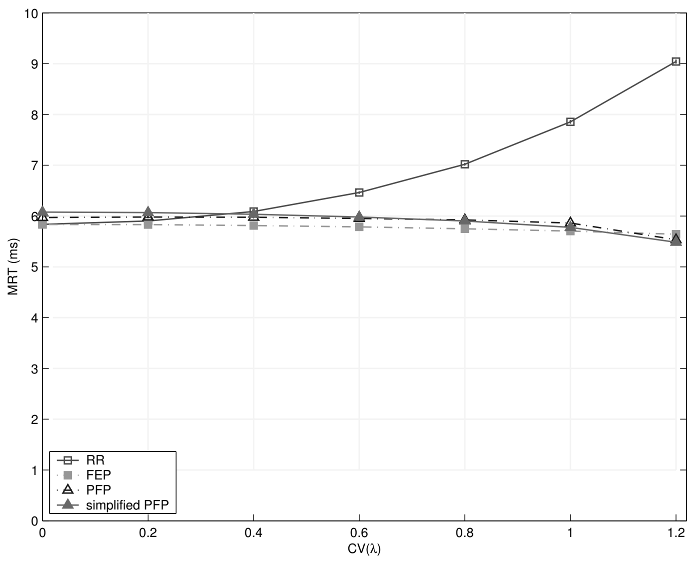


Figure 3.12: *Mean response time (MRT) in a lowly loaded piconet ($\rho = 0.1$)*

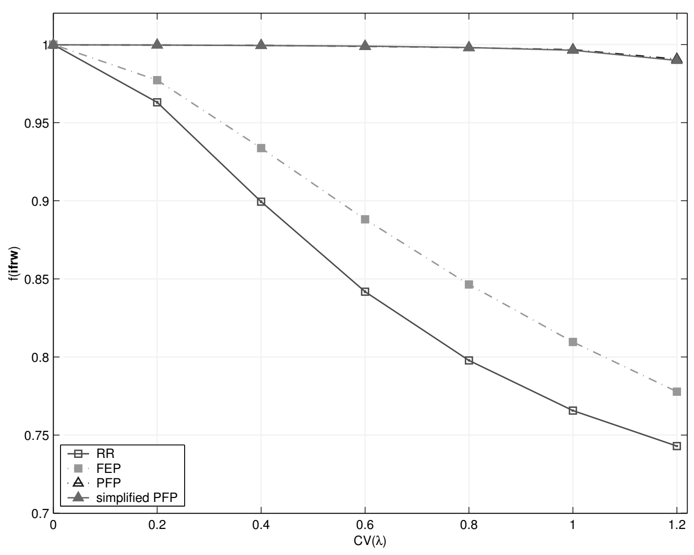


Figure 3.13: *Fairness based on inverse fraction of reference mean waiting time ($f(ifrw)$) in a lowly loaded piconet ($\rho = 0.1$)*

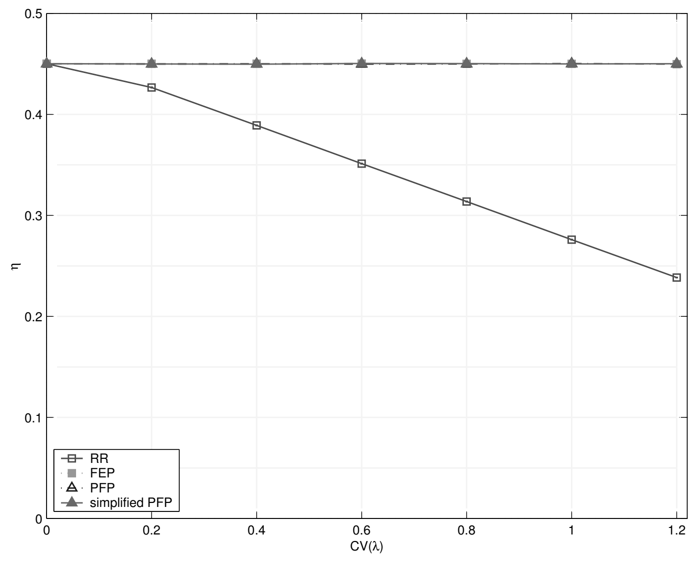


Figure 3.14: *Efficiency (η) in a highly loaded piconet ($\rho = 0.45$)*

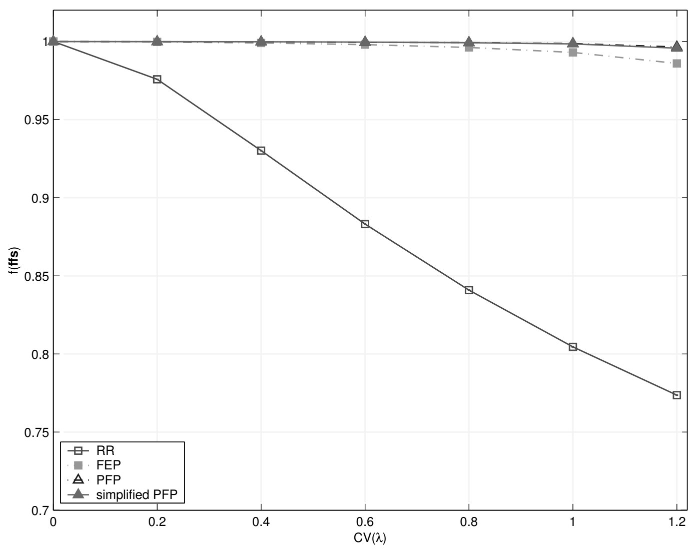


Figure 3.15: *Fairness based on fraction of fair share ($f(ffs)$) in a highly loaded piconet ($\rho = 0.45$)*

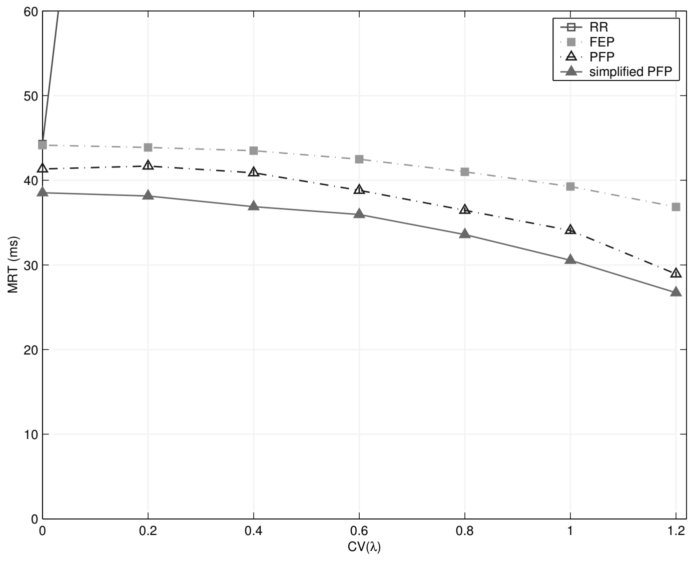


Figure 3.16: *Mean response time (MRT) in a highly loaded piconet ($\rho = 0.45$)*

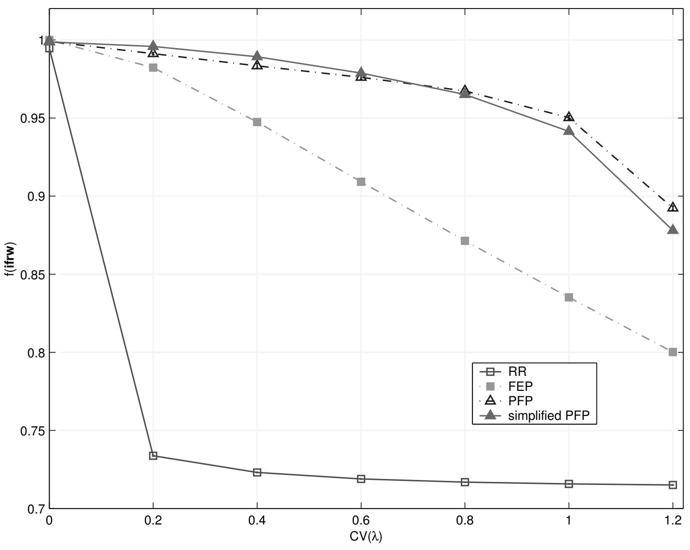


Figure 3.17: *Fairness based on inverse fraction of reference mean waiting time ($f(ifrw)$) in a highly loaded piconet ($\rho = 0.45$)*

3.5.3 The Poisson scenario with variable IP packet sizes

3.5.3.1 Purpose of the simulation

The purpose of simulating in this simulation scenario is to show that the 1-limited Round Robin poller is not always able to handle bursty traffic that is asymmetrically distributed among the slaves. Furthermore, by means of this simulation, we investigate whether the Predictive Fair Pollers perform at least as good as the Fair Exhaustive Poller.

This scenario uses variable-size IP packets that arrive with exponentially distributed inter-arrival times, and which may cover multiple baseband packets. Consequently, the baseband packets arrive in small batches (between 1 and 5 baseband packets) with exponentially distributed inter-arrival times. By means of simulations, the performances of the Predictive Fair Poller and the simplified Predictive Fair Poller are compared in case baseband packets arrive in bursts.

3.5.3.2 Description of the simulation scenario

In this scenario we simulate under the following assumptions (see also Figure 3.8):

- Seven slaves (S1,...,S7) and a master (M) form a piconet, while two of the seven slaves are highly loaded, i.e.,
$$N = 7 \quad \text{and} \quad n_h = 2. \tag{3.131}$$
- There is only upstream traffic, i.e., from the slaves to the master.
- The slaves generate IP packets according to Poisson processes with arrival rates $\lambda_1, ..., \lambda_7$, while
$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = \lambda_l \geq 0, \tag{3.132}$$
and
$$\lambda_6 = \lambda_7 = \lambda_h \geq \lambda_l. \tag{3.133}$$
- The available Bluetooth baseband packet types are DH1, DH3 and DH5 with a maximum payload size of 27 bytes, 183 bytes and 339 bytes, respectively.
- The segmentation policy is such that packets are transmitted using baseband packets that lead to the highest average slot efficiency (bytes per slot).
- The IP packet size distribution used is trimodal [EKW99] with 30% of 40-byte IP packets, 12% of 1500-byte IP packets and 58% of IP packets with a size in the range of 300 to 600 bytes. Given the available baseband packet sizes and the followed segmentation policy, this results in an average of $\overline{d} = 8.30$ data slots per IP packet and an average of $\overline{p} = 1.99$ polls per IP packet, while all the polls are explicit, i.e., $\overline{w} = 1.99$ wasted time slots per IP packet.
- The number of slots per second available for transmission of data or for transmission of a POLL or NULL packet is
$$C_{tdd} = 1600 \text{ slots/sec.} \tag{3.134}$$

- According to (3.42), the load of the piconet is given by

$$\rho = \frac{(5\lambda_l + 2\lambda_h)\overline{d}}{C_{tdd}} = \frac{8.30(5\lambda_l + 2\lambda_h)}{1600}, \tag{3.135}$$

  while, according to (3.71), the minimum load for which no poller can be stable is given by

$$\hat{\rho} = 0.807. \tag{3.136}$$

- According to (3.44), the coefficient of variation of the arrival rates is given by

$$\mathrm{CV}(\boldsymbol{\lambda}) = \frac{\sqrt{\frac{35}{3}}(\lambda_h - \lambda_l)}{5\lambda_l + 2\lambda_h}, \tag{3.137}$$

  while, according to (3.45), the maximum coefficient of variation of the arrival rates for which λ_l is non-negative is

$$\hat{\mathrm{CV}}(\boldsymbol{\lambda}) = \sqrt{\frac{35}{12}}. \tag{3.138}$$

- According to (3.48) and (3.49) the arrival rates are given by

$$\lambda_l = \frac{1600\rho}{8.30 \cdot 7}(1 - \sqrt{\frac{12}{35}}\mathrm{CV}(\boldsymbol{\lambda})), \tag{3.139}$$

  and

$$\lambda_h = \frac{1600\rho}{8.30 \cdot 7}(1 + \frac{5}{2}\sqrt{\frac{12}{35}}\mathrm{CV}(\boldsymbol{\lambda})). \tag{3.140}$$

The simulations are performed in a lowly loaded piconet ($\rho = 0.1$) and in a highly loaded piconet ($\rho = 0.7$). We compare the 1-limited Round Robin poller (RR), the Fair Exhaustive Poller (FEP), the Predictive Fair Poller (PFP), and the simplified Predictive Fair Poller (simplified PFP) taking into account the efficiency (η), the total mean response time (MRT), and the fairness based on fraction of fair share ($f(\mathit{ffs})$). These performance metrics are explained in Section 3.1.2

3.5.3.3 Expectations for the efficiency

According to Section 3.4.2, the efficiency of the 1-limited Round Robin poller will be given by (cf. (3.97))

$$\eta = \begin{cases} \rho, & \text{Operation area I,} \\ 0.410 + 0.492\rho(1 - \sqrt{\frac{12}{35}}\mathrm{CV}(\boldsymbol{\lambda})), & \text{Operation area II and III,} \\ 0.807, & \text{Operation area IV.} \end{cases} \tag{3.141}$$

where the operation areas are shown in Table 3.4 (confer Table 3.1) and Figure 3.18. The system served by a 1-limited Round Robin poller is only stable in operation area I.

Operation area	Conditions	
I	$\rho < 0.807$	$\mathrm{CV}(\boldsymbol{\lambda}) < \frac{2}{5}\sqrt{\frac{35}{12}}2.585(\frac{0.807}{\rho} - 1)$
II		$\mathrm{CV}(\boldsymbol{\lambda}) \geq \frac{2}{5}\sqrt{\frac{35}{12}}2.585(\frac{0.807}{\rho} - 1)$
III	$\rho \geq 0.807$	$\mathrm{CV}(\boldsymbol{\lambda}) \geq \sqrt{\frac{35}{12}}(1 - \frac{0.807}{\rho})$
IV		$\mathrm{CV}(\boldsymbol{\lambda}) < \sqrt{\frac{35}{12}}(1 - \frac{0.807}{\rho})$

Table 3.4: *Operation areas of the 1-limited Round Robin poller in the Poisson scenario with variable IP packet sizes*

As can be seen in Figure 3.18 and be calculated using (3.73), the system served by the 1-limited Round Robin poller is stable regardless of the coefficient of variation of the arrival rates ($\mathrm{CV}(\boldsymbol{\lambda})$) if

$$\rho < \frac{n_h\overline{d}}{n_h(\overline{d}+\overline{w}) + 2(N-n_h)\overline{p}} = 0.410. \tag{3.142}$$

As a result, the system served by the 1-limited Round Robin poller will be stable in the lowly loaded piconet ($\rho = 0.1$) regardless of the coefficient of variation of the arrival rates ($\mathrm{CV}(\boldsymbol{\lambda})$), and a maximum efficiency of $\eta = \rho = 0.1$ is expected to be achieved. However, in the highly loaded piconet ($\rho = 0.7$), the system will be stable as long as (see also operation area I in table 3.4) the following inequality holds:

$$\mathrm{CV}(\boldsymbol{\lambda}) < \frac{2}{5}\sqrt{\frac{35}{12}}2.585\left(\frac{0.807}{\rho} - 1\right) = 0.269. \tag{3.143}$$

Table 3.5 shows the calculated values for the efficiency (η) as a function of the coefficient of variation of the arrival rates ($\mathrm{CV}(\boldsymbol{\lambda})$) in a highly loaded piconet ($\rho = 0.7$).

$\mathrm{CV}(\boldsymbol{\lambda})$	0	0.2	0.4	0.6	0.8	1	1.2
η	ρ	ρ	0.67	0.63	0.59	0.55	0.51
Capacity loss	0%	0%	3.8%	9.5%	15.3%	21%	26.8%

Table 3.5: *Expected efficiency (η) of the 1-limited Round Robin Poller as function of $\mathrm{CV}(\boldsymbol{\lambda})$ in a highly loaded piconet ($\rho = 0.7$)*

The Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller are assumed to be able of handling traffic that is asymmetrically distributed among the slaves. It is expected that these pollers will be stable as long as they operate in operation area I or II, i.e., as long as

$$\rho < \hat{\rho} = \frac{\overline{d}}{\overline{d}+\overline{w}} = 0.807. \tag{3.144}$$

The simulations take place in a lowly loaded piconet ($\rho = 0.1$) and in a highly loaded piconet ($\rho = 0.7$). As (3.144) holds in both cases, we expect the Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller to be stable in both cases. This implies that they achieve the maximum efficiency of $\eta = \rho$.

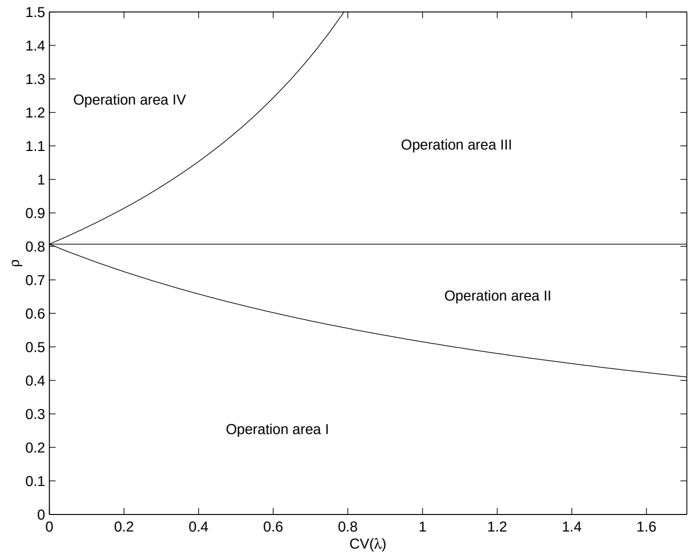


Figure 3.18: *Operation areas of the 1-limited Round Robin poller in the Poisson scenario with variable IP packet sizes*

3.5.3.4 Expectations for the fairness

The simulated system operates in operation area I and II. Consequently, the fairness based on fraction of fair share for the 1-limited Round Robin poller will be given by (3.105), and will be independent of the coefficient of variation of the arrival rates $\mathrm{CV}(\boldsymbol{\lambda})$. Figure 3.10 shows the fairness based on fraction of fair share as a function of the coefficient of variation of the arrival rates.

In this simulation scenario, the load consists of variable-size IP packets that may cover multiple (multi-slot) baseband packets. Consequently, a load equal to the load in the simulation scenario of Section 3.5.2 leads to a much lower ($\approx$4.17 times) required poll rate in this scenario.

In the lowly loaded piconet, the loads will be low to such an extent that the Fair Exhaustive Poller, most of the time, will assume the slaves being inactive. Hence, the Fair Exhaustive Poller will poll the slaves in a 1-limited Round Robin manner, leading to a fairness based on fraction of fair share comparable to that of the 1-limited Round Robin poller. In the highly loaded piconet, the loads will be high enough to make the Fair Exhaustive Poller capable of distinguishing between active nodes and inactive nodes. Consequently, the Fair Exhaustive Poller will achieve a higher fairness in the highly loaded piconet.

The Predictive Fair Poller and the simplified Predictive Fair Poller poll slaves proportional to their fair share of resources. Hence, the expected fairness based on fraction of fair share approaches one.

3.5.3.5 Simulation results

In this section we discuss the simulation results of the 1-limited Round Robin poller, the Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller in a lowly loaded piconet ($\rho = 0.1$) and in a highly loaded piconet ($\rho = 0.7$).

Lowly loaded piconet ($\rho = 0.1$)

The four polling mechanisms perform equally well with respect to the efficiency (η) as function of the coefficient of variation of the arrival rate. They achieve the maximum efficiency of $\eta = \rho$, which conforms the expectation.

Figure 3.19 shows the fairness based on fraction of fair share for the four polling mechanisms in a lowly loaded piconet. The fairness based on fraction of fair share of the 1-limited Round Robin poller conforms the expectations drawn in Figure 3.10. It can be seen that the fairness of the Fair Exhaustive Poller is almost equal to the fairness of the 1-limited Round Robin poller, which conforms the expectations. The Predictive Fair Poller and the simplified Predictive Fair Poller achieve a fairness based on fraction of fair share that approaches unity, except when $\mathrm{CV}(\boldsymbol{\lambda}) > 1$. The reason for this is that the required poll rate of the lowly loaded slaves is then low to such an extent that it is hard to properly estimate the required poll rates.

Figure 3.20 shows the total mean response time for the four polling mechanisms in a lowly loaded piconet. The total mean response time MRT of the 1-limited Round Robin poller is slightly increasing for higher coefficient of variation of the arrival rates $(\mathrm{CV}(\boldsymbol{\lambda}))$. The reason for this is the same as in the first simulation scenario. The Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller adapt their poll rate to the load. Hence, they achieve a total mean response time that is lower than the one achieved by the 1-limited Round Robin poller. Furthermore, it can be seen that, even when the slaves are equally loaded, the response time achieved by these pollers is lower than the response time achieved by the 1-limited Round Robin poller. With respect to the Fair Exhaustive Poller, the reason for this is that whenever a first segment (baseband packet) of an IP packet is received from a slave, that slave will be active until a NULL packet is received. Since there is a high probability that no other slave is active at the same moment, no other slave will be polled before the transmission of the IP packet is completed. With respect to the Predictive Fair Poller, the reason for this low delay is the fact that this poller takes into account whether continuations are pending. With respect to the simplified Predictive Fair Poller, this low delay is caused by the fact that this poller keeps assuming availability of data at a slave until a NULL packet is received.

Highly loaded piconet ($\rho = 0.7$)
Figure 3.21 shows the efficiency for the four polling mechanisms in a highly loaded piconet. As expected (see Table 3.5), the 1-limited Round Robin poller becomes inefficient for increasing values of $\mathrm{CV}(\boldsymbol{\lambda})$. The Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller adapt to the different loads, and hence achieve a maximum efficiency of $\eta = \rho = 0.7$.

Figure 3.22 shows the fairness based on fraction of fair share for the four polling mechanisms in a highly loaded piconet. The fairness based on fraction of fair share of the 1-limited Round Robin poller conforms the expectations drawn in Figure 3.10. With respect to the Fair Exhaustive Poller, and because of the higher total required poll rate, the list of active slaves will be non-empty most of the time. Instead of polling all the slaves in a 1-limited Round Robin manner, the Fair Exhaustive Poller now divides most of its resources between the slaves that need to be polled. Consequently, the Fair Exhaustive Poller achieves a higher fairness based on fraction of fair share than the 1-limited Round Robin poller. As mentioned before, the Predictive Fair Poller and the simplified Predictive Fair Poller poll each slave at a poll rate proportional to its fair share of resources. Hence, their fairness based on fraction of fair share approaches one.

Figure 3.23 shows the total mean response time for the four polling mechanisms in a highly loaded piconet. From (3.143) we know that the considered system served by a 1-limited Round Robin polling mechanism becomes unstable if $\mathrm{CV}(\boldsymbol{\lambda}) \geq 0.269$. This can also be seen in Figure 3.23. Furthermore, as the Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller adapt to the different traffic demands, they achieve lower mean response times than the ones achieved by the 1-limited Round Robin poller, even if the load is asymmetrically distributed among the slaves (i.e., if $\mathrm{CV}(\boldsymbol{\lambda}) = 0$).

3.5.3.6 Conclusions of scenario II

As in scenario I, the simulation results showed that the Round Robin poller is unable to handle traffic that is asymmetrically distributed among the slaves. Furthermore, the simulation results showed that both the Predictive Fair Poller and the simplified Predictive Fair Poller outperform both the 1-limited Round Robin poller and the Fair Exhaustive Poller.

As this scenario uses variable-size IP packets that may cover multiple baseband packets, the baseband packets arrive in small batches (between 1 and 5 baseband packets) with exponentially distributed inter-arrival times. The simplified Predictive Fair Poller always assumes the availability of a baseband packet at a slave if its last poll to that slave was a successful poll. The Predictive Fair Poller, however, does not assume batch arrivals of IP packets. Instead, it detects whether continuation baseband packets are available at a slave or not.

Given the assumptions made by the Predictive Fair Poller and the simplified Predictive Fair Poller, the Predictive Fair Poller is able to achieve a slightly better performance than the simplified Predictive Fair Poller, which is shown by the simulation results.

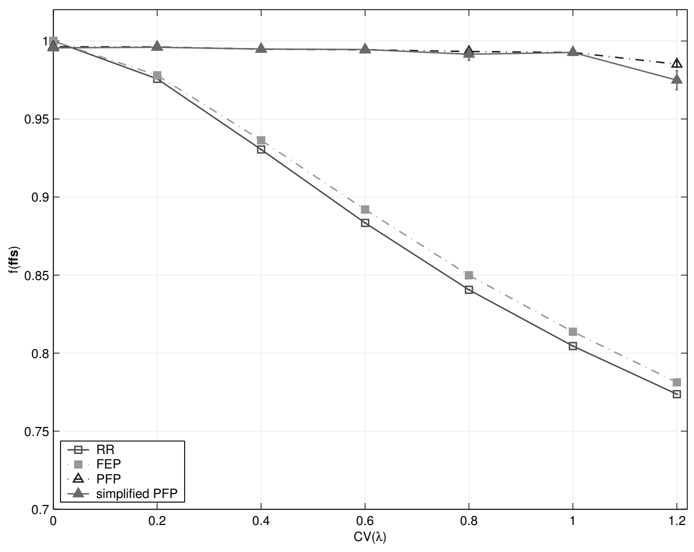


Figure 3.19: *Fairness based on fraction of fair share ($f(\mathit{ffs})$) in a lowly loaded piconet ($\rho = 0.1$)*

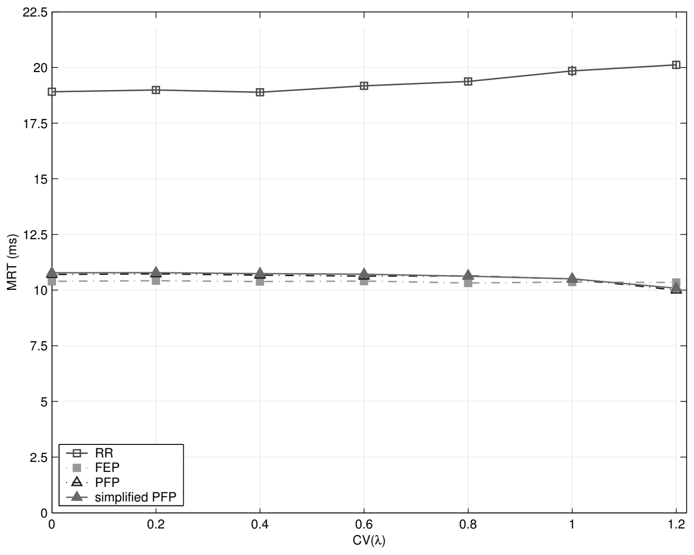


Figure 3.20: *Mean response time (MRT) in a lowly loaded piconet ($\rho = 0.1$)*

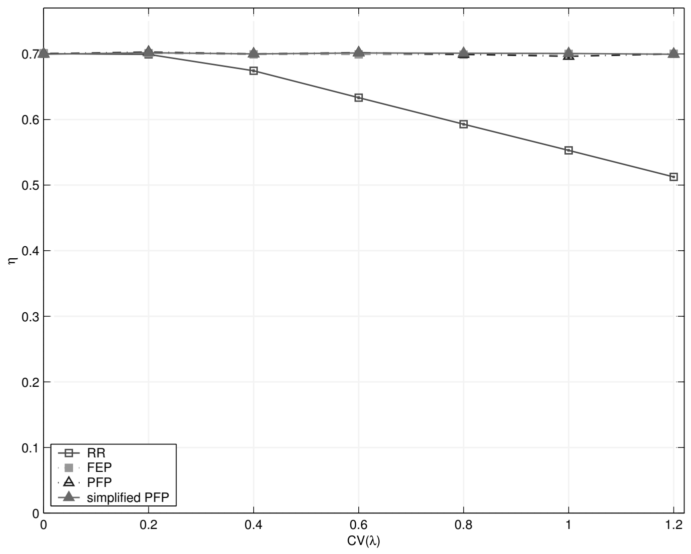


Figure 3.21: *Efficiency (η) in a highly loaded piconet ($\rho = 0.7$)*

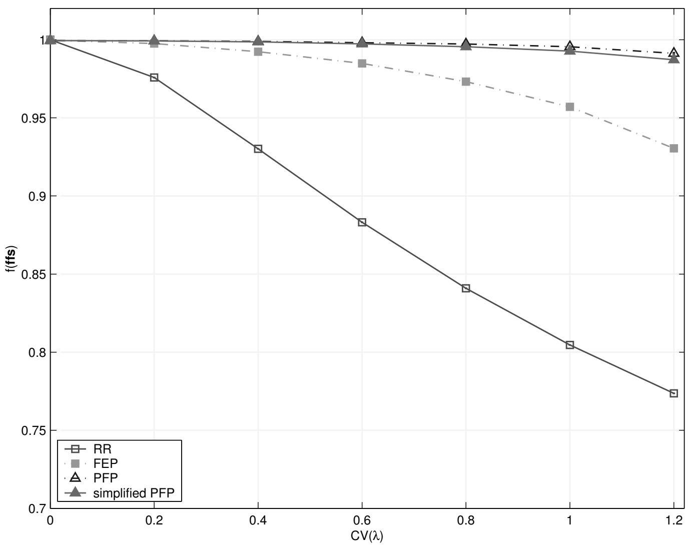


Figure 3.22: *Fairness based on fraction of fair share ($f(ffs)$) in a highly loaded piconet ($\rho = 0.7$)*

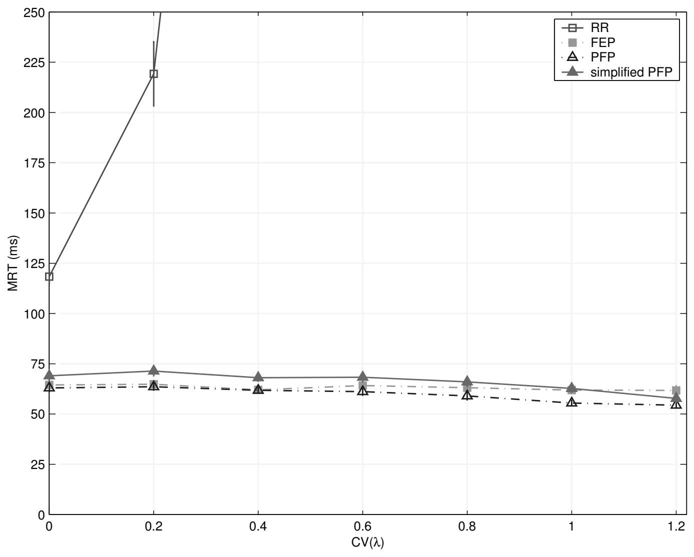


Figure 3.23: *Mean response time (MRT) in a highly loaded piconet ($\rho = 0.7$)*

3.5.4 The FTP/TCP scenario

3.5.4.1 Purpose of the simulation

The purpose of this simulation scenario is to show, in a more realistic traffic load environment, that the 1-limited Round Robin poller is not always able to handle traffic that is asymmetrically distributed among the slaves. Furthermore, by means of this simulation, we investigate whether the Predictive Fair Pollers perform at least as good as the Fair Exhaustive Poller.

This scenario uses MTU-sized IP packets that cover five baseband packets. The Predictive Fair Poller is able to detect the availability of continuation baseband packets, whereas the simplified Predictive Fair Poller always assumes the availability of continuation baseband packets after a successful poll. By means of simulations, the performances of the Predictive Fair Poller and the simplified Predictive Fair Poller are compared, in a more realistic traffic load environment, in case baseband packets arrive in bursts.

3.5.4.2 Description of the simulation scenario

In this scenario we simulated under the following assumptions (see also Figure 3.8):

- Seven slaves (S1,...,S7) and a master (M) form a piconet, while the master is also connected to a wired network, i.e., the master is acting as an accesspoint.
- Each slave Si continuously uploads data to wired node Wi through master M and wired node W0 using FTP/TCP (BSD Tahoe), while capacity C_0 of the wired duplex link L0 is much higher than the total capacity in a piconet.
- The available Bluetooth Baseband packet types are DH1, DH3, and DH5 with a maximum payload size of 27 bytes, 183 bytes and 339 bytes, respectively.
- The segmentation policy requires that the largest allowable baseband packet is used, unless the remainder of the packet fits into a smaller baseband packet.
- The MTU size is 1500 bytes. Because of the available Bluetooth baseband packet types and because of the segmentation policy, each IP packet covers four DH5 and one DH3 baseband packets.
- The capacities $(C_1, ..., C_7)$ of the wired duplex links (L1,...,L7) obey

$$C_1 = C_2 = C_3 = C_4 = C_5 = C_l, \tag{3.145}$$

  and

$$C_6 = C_7 = C_h \geq C_l, \tag{3.146}$$

  while

$$C_{tot} = \sum_{i=1}^{7} C_i. \tag{3.147}$$

- Because of the selection of the link capacities and the use of TCP, two of the seven slaves will belong to the group of highly loaded slaves, i.e.,

$$N = 7 \quad \text{and} \quad n_h = 2. \tag{3.148}$$

- The coefficient of variation of the link capacities is defined as

$$\mathrm{CV}(\boldsymbol{C}) = \frac{\sqrt{\frac{\sum_{i=1}^{7} C_i^2 - \frac{1}{7}(\sum_{i=1}^{7} C_i)^2}{6}}}{\frac{1}{7}\sum_{i=1}^{7} C_i}, \tag{3.149}$$

while the maximum coefficient of variation of the link capacities for which the capacity C_l is non-negative is

$$\hat{\mathrm{CV}}(\boldsymbol{C}) = \sqrt{\frac{35}{12}}. \tag{3.150}$$

Due to the assumption on the link capacities ($C_1, ..., C_7$), each combination of the coefficient of variation of the link capacities and the sum of the link capacities (C_{tot}) gives a unique solution for the link capacities $C_1, ..., C_7$, and thus makes it possible to use $\mathrm{CV}(\boldsymbol{C})$ and C_{tot} as input for the simulations.
Because of the different link capacities and the use of TCP, the slaves will generate data at different rates. Hence, the poller must adapt to these different rates by polling some slaves more often than other slaves in order to maximize the throughput T (total data upload rate) while being fair. We will compare the Predictive Fair Poller, the Fair Exhaustive Poller, and the Round Robin poller taking into account efficiency (η), fairness based on fraction of fair share ($f(ffs)$), and throughput (T). Throughput T is defined as the average number of bits per second received by the master from the slaves.

3.5.4.3 Expectations for the efficiency

An approximation for the efficiency that can be achieved given a total capacity of the wired lines (C_{tot}) and a coefficient of variation of the link capacities can be given making the following assumptions:

- TCP tries to fully utilize the path between the wireless node and the wired node. Assuming that the wired links L1,...,L7 are the bottlenecks, the TCP source at each slave i tries to fully utilize wired link Li, hence generating TCP packets at approximately the following rate

$$\lambda_i \approx \frac{C_i}{8 \cdot \mathrm{MTU}} \text{ TCP packets/sec.} \tag{3.151}$$

- The TCP we use produces one acknowledgment packet (40 bytes) for each TCP packet (MTU). Given the allowed Baseband packets types and the followed segmentation policy, the TCP packet will be transmitted using 4 DH5 Baseband packets and 1 DH3 Baseband packet, whereas the ACK packet is transmitted using one DH3 Baseband packet. Consequently, each TCP packet generation at a slave needs the following number of data slots to be handled

$$\overline{d} = 23 + 3 = 26 \text{ slots/TCP packet}, \tag{3.152}$$

and the following number of polls (implicit and explicit)

$$\overline{p} = 5 \text{ polls/TCP packet.} \tag{3.153}$$

- Although the ACK packet needs a DH3 packet to be transmitted, the actual transmission of this ACK packet lasts less than 3 time slots. We assume the remaining time until expiry of the three slots reserved for that DH3 to be enough for the slave to handle the incoming ACK and generate a new TCP packet. Hence, this DH3 data packet serves as an implicit poll for the first DH5 packet belonging to the newly generated TCP packet. As a result, only four explicit polls (POLL packets) per TCP packet are needed, i.e., four wasted slots per TCP packet

$$\overline{w} = 4 \text{ slots/TCP packet.} \tag{3.154}$$

- Given the assumptions above, an approximation for the load in the piconet can be given by

$$\rho \approx \frac{(\sum_i^N \lambda_i)\overline{d}}{C_{tdd}} = \frac{(\frac{C_{tot}}{8 \cdot \text{MTU}})\overline{d}}{C_{tdd}} = \frac{13}{15}\frac{C_{tot}}{640000}, \tag{3.155}$$

while the minimum load for which Bluetooth will always be a bottleneck is

$$\hat{\rho} = \frac{\overline{d}}{\overline{d} + \overline{w}} = \frac{13}{15}. \tag{3.156}$$

Substituting ρ from (3.155) and $\hat{\rho}$ from (3.156) in (3.97), the expected efficiency for the 1-limited Round Robin poller is given by

$$\eta = \begin{cases} \frac{13}{15}\frac{C_{tot}}{640000}, & \text{Operation area I,} \\ \\ \frac{26}{55} + \frac{13}{33}\frac{C_{tot}}{640000}(1 - \sqrt{\frac{12}{35}}\text{CV}(\boldsymbol{C})), & \text{Operation area II and III,} \\ \\ \frac{13}{15}, & \text{Operation area IV.} \end{cases} \tag{3.157}$$

The operation areas are shown in Table 3.6 and Figure 3.24. Operation area I is the operation area in which the Bluetooth part will not be a bottleneck.

According to (3.73) and (3.155), the Bluetooth part will not be the bottleneck if

$$C_{tot} < \frac{15}{13}640000\frac{n_h\overline{d}}{n_h(\overline{d} + \overline{w}) + 2(N - n_h)\overline{p}} = 349091 \text{ bps.} \tag{3.158}$$

As a result, and given that the system is served by the 1-limited Round Robin poller, the Bluetooth part will never be the bottleneck in the lowly loaded piconet ($C_{tot} = 100000$ bps). However, in the highly loaded piconet ($C_{tot} = 600000$ bps), the Bluetooth part will not be the bottleneck as long as

$$\text{CV}(\boldsymbol{C}) < \frac{6}{5}\sqrt{\frac{35}{12}}(\frac{640000}{C_{tot}} - 1) = 0.137. \tag{3.159}$$

Table 3.7 shows the calculated values for the efficiency (η) and the throughput (T) as a function of the coefficient of variation of the link capacities ($\text{CV}(\boldsymbol{C})$) in a highly loaded piconet ($C_{tot} = 600000$ bps, $\rho \approx 0.8125$).

Operation area	Conditions	
I	$C_{tot} < 640000$	$\text{CV}(\boldsymbol{C}) < \frac{6}{5}\sqrt{\frac{35}{12}}(\frac{640000}{C_{tot}} - 1)$
II		$\text{CV}(\boldsymbol{C}) \geq \frac{6}{5}\sqrt{\frac{35}{12}}(\frac{640000}{C_{tot}} - 1)$
III	$C_{tot} \geq 640000$	$\text{CV}(\boldsymbol{C}) \geq \sqrt{\frac{35}{12}}(1 - \frac{640000}{C_{tot}})$
IV		$\text{CV}(\boldsymbol{C}) < \sqrt{\frac{35}{12}}(1 - \frac{640000}{C_{tot}})$

Table 3.6: *Operation areas of the 1-limited Round Robin poller in the FTP/TCP scenario*

$\text{CV}(\boldsymbol{C})$	0	0.2	0.4	0.6	0.8	1	1.2
η	0.8125	0.799	0.756	0.712	0.669	0.626	0.583
T(bps)	600k	590k	558k	526k	494k	462k	430k
Capacity loss	0%	1.7%	7%	12.3%	17.7%	23%	28.3%

Table 3.7: *Expected efficiency (η) and throughput (T) of the 1-limited Round Robin Poller as function of $\text{CV}(\boldsymbol{C})$ in a highly loaded piconet ($C_{tot} = 600000$ bps, $\rho \approx 0.8125$)*

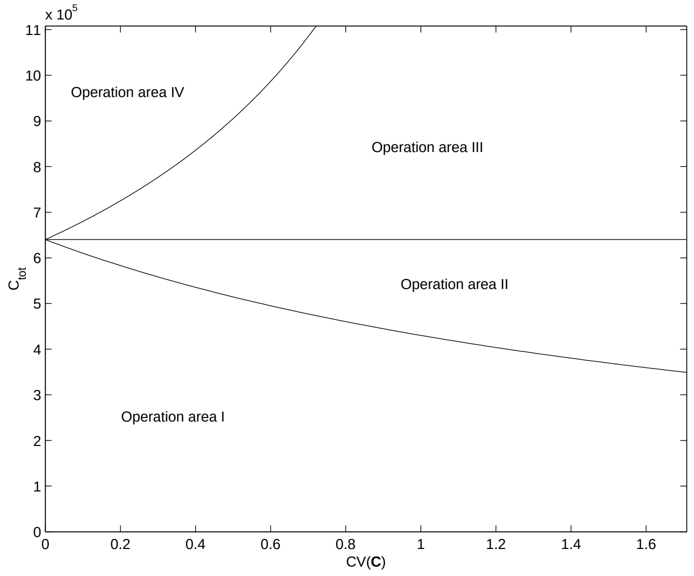


Figure 3.24: *Operation areas of the 1-limited Round Robin poller in the FTP/TCP scenario*

3.5.4.4 Expectations for the fairness

The simulated system operates in operation area I and II. Consequently, the fairness based on the fraction of fair share for the 1-limited Round Robin poller will be given by (3.105), and will be independent of the total capacity of the links C_{tot}. Figure 3.10 shows the fairness based on fraction of fair share as a function of the coefficient of variation of the arrival rates. As we assume a linear relation between a link capacity and the resulting arrival rate, the same figure can be interpreted as fairness based on fraction of fair share as a function of the coefficient of variation of the link capacities ($\mathrm{CV}(\boldsymbol{\lambda}) = \mathrm{CV}(\boldsymbol{C})$)).

In this simulation scenario, the load consists of TCP packets with a packet size of 1500 bytes. Because of the allowed baseband packet types and the followed segmentation policy, these packets will be transmitted using four DH5 packets and one DH3 packet. Consequently, a load equal to the load in the simulation scenario of Section 3.5.2, leads to a much lower (5.2 times) required poll rate in this scenario.

In the lowly loaded piconet, the loads will be low to such an extent that the Fair Exhaustive Poller, most of the time, will assume the slaves being inactive. Hence, all the slaves will be polled in a 1-limited Round Robin manner, leading to a fairness based on fraction of fair share comparable to that of the 1-limited Round Robin poller. In the highly loaded piconet, the load will be high enough to make the Fair Exhaustive Poller be able of distinguishing between active slaves and inactive slaves. Consequently, the Fair Exhaustive Poller will achieve a higher fairness in the highly loaded piconet.

The Predictive Fair Poller and the simplified Predictive Fair Poller poll slaves such that their resulting poll rate is proportional to their fair share of resources. Hence, the expected fairness based on fraction of fair share approaches unity.

3.5.4.5 Simulation results

In this section we discuss the simulation results of the 1-limited Round Robin poller, the Fair Exhaustive Poller, the Predictive Fair Poller, and the simplified Predictive Fair Poller in a lowly loaded piconet ($C_{tot} = 100000$ bps, i.e., $\rho \approx 0.135$) and in a highly loaded piconet ($C_{tot} = 600000$ bps, i.e., $\rho \approx 0.8125$).

Lowly loaded piconet ($C_{tot} = 100000$ bps)

The four pollers perform equally well with respect to the throughput T and efficiency η as function of the coefficient of variation of the link capacities ($\mathrm{CV}(\boldsymbol{C})$), i.e., $T \rightarrow C_{tot}$ and $\eta = \rho$.

Figure 3.25 shows the fairness based on fraction of fair share for the four polling mechanisms in a lowly loaded piconet. The fairness based on fraction of fair share of the 1-limited Round Robin poller conforms the expectations drawn in Figure 3.10. It can be seen that the fairness of the Fair Exhaustive Poller is almost equal to the fairness of the 1-limited Round Robin poller, which also conforms the expectations. The Predictive Fair Poller and the simplified Predictive Fair Poller achieve the maximum fairness based on fraction of fair share.

Highly loaded piconet ($C_{tot} = 600000$ bps)
Figure 3.26 and Figure 3.27 show the throughput and efficiency in a highly loaded piconet, respectively. It can be seen that the 1-limited Round Robin poller cannot handle the load as it becomes asymmetrically distributed among the slaves. The Fair Exhaustive Poller, Predictive Fair Poller and simplified Predictive Fair Poller achieve a maximum throughput and thus a maximum efficiency, i.e., $T \rightarrow C_{tot}$ and $\eta = \rho$, respectively.

Figure 3.28 shows the fairness based on fraction of fair share for the four polling mechanisms in a highly loaded piconet. The fairness based on fraction of fair share of the 1-limited Round Robin poller conforms the expectations drawn in Figure 3.10. With respect to the Fair Exhaustive Poller, and because of the higher total required poll rate, the list of active slaves will be non-empty most of the time. Consequently, the Fair Exhaustive Poller, most of the time, will poll slaves that need to be polled. Hence, the Fair Exhaustive Poller now achieves a higher fairness based on fraction of fair share. The Predictive Fair Poller and the simplified Predictive Fair Poller achieve a fairness that approaches the maximum fairness.

3.5.4.6 Conclusions of scenario III

As in scenario I and II, the simulation results showed that the 1-limited Round Robin poller is not always able to handle traffic that is asymmetrically distributed among the slaves. Furthermore, the simulation results showed that both the Predictive Fair Poller and the simplified Predictive Fair Poller outperform both the 1-limited Round Robin poller and the Fair Exhaustive Poller.

In this scenario, a more realistic traffic load is offered in the piconet. The used IP packets cover five baseband packets, resulting in batch arrivals of baseband packets. The simplified Predictive Fair Poller alway assumes the availability of a baseband packet at a slave if its last poll to that slave was a successful poll. The Predictive Fair Poller, however, does not assume batch arrivals of IP packets. Instead, it detects whether continuation baseband packets are available at a slave or not. Given the assumptions made by the Predictive Fair Poller and the simplified Predictive Fair Poller, the Predictive Fair Poller is able to achieve slightly better performance than the simplified Predictive Fair Poller, which is shown by the simulation results.

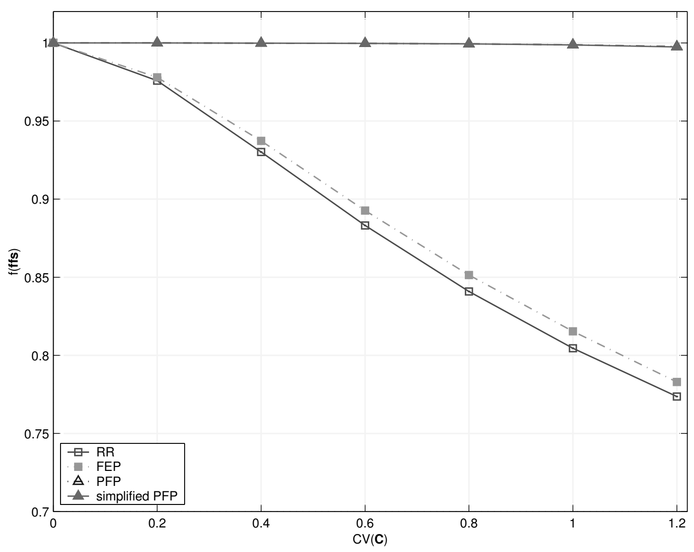


Figure 3.25: *Fairness based on fraction of fair share ($f(ffs)$) in a lowly loaded piconet ($C_{tot} = 100000$ bps)*

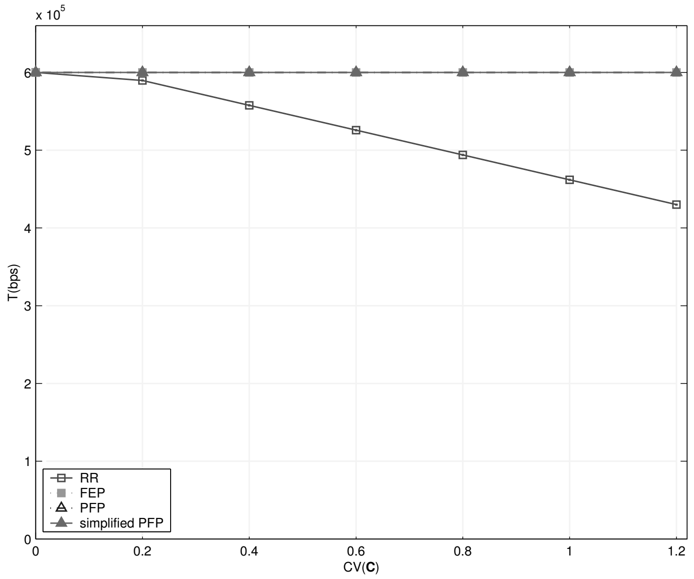


Figure 3.26: *Throughput (T) in a highly loaded piconet ($C_{tot} = 600000$ bps)*

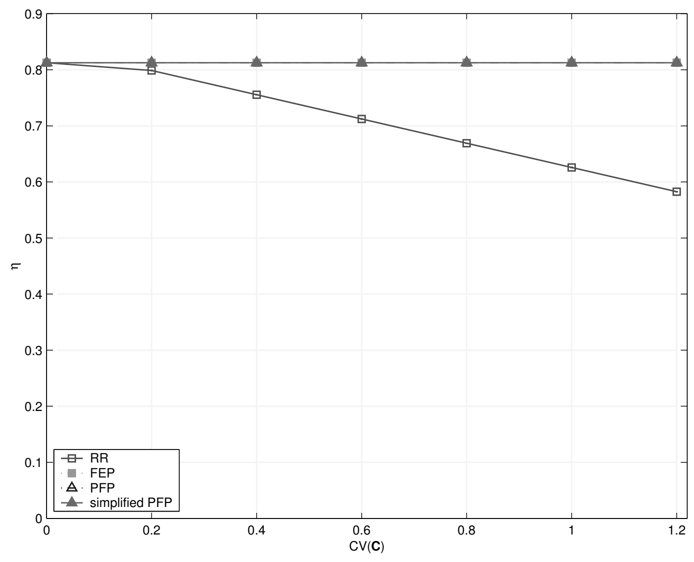


Figure 3.27: *Efficiency (η) in a highly loaded piconet ($C_{tot} = 600000$ bps)*

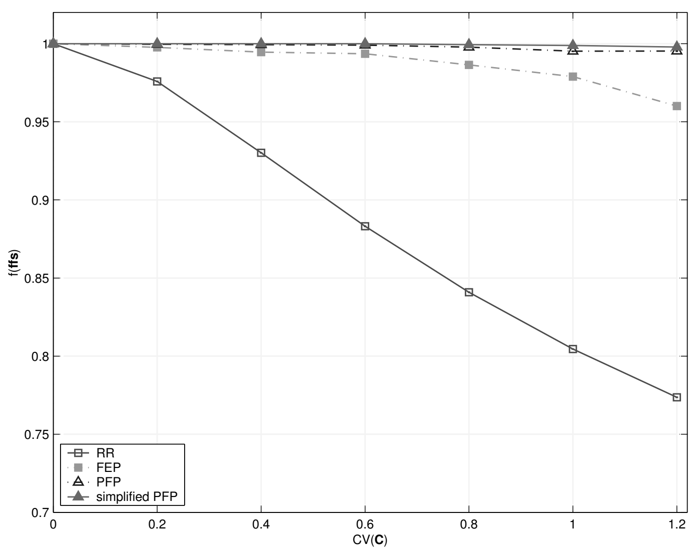


Figure 3.28: *Fairness based on fraction of fair share ($f(ffs)$) in a highly loaded piconet ($C_{tot} = 600000$ bps)*

3.6 Discussion

Bluetooth polling strongly determines the efficiency (and thus throughput) and fairness, especially in a highly loaded piconet with loads that are asymmetrically distributed among the slaves.

In the literature, polling mechanisms can be found that are able to handle loads that are asymmetrically distributed among the slaves. This is the main requirement in a best effort traffic environment in order to efficiently handle the traffic. However, the mentioned polling mechanism have no means for extension to QoS traffic handling.

We developed a polling mechanism, named Predictive Fair Poller, that is able to handle traffic that is asymmetrically distributed among the slaves, while being extendable with QoS capabilities. For that, the Predictive Fair Poller takes both efficiency and fairness into account.

Through simulations we compared the Predictive Fair Pollers with the 1-limited Round Robin poller and the Fair Exhaustive poller in two Poisson scenarios and in an FTP/TCP scenario. Simulation results pointed out that the Predictive Fair Pollers outperform the 1-limited Round Robin poller with respect to all studied performance metrics and that they perform at least as good as and sometimes better than the Fair Exhaustive Poller. Furthermore, the simulation results showed that the impact on the performance of simplifying the Predictive Fair Poller is negligible, which justifies the simplification.

In this chapter, we have explained the operation of the Predictive Fair Poller, and we have analyzed its essentials with respect to Best Effort traffic. In the next chapters we define the QoS capabilities of the Predictive Fair Poller in more detail, and analyze its performance behavior in a QoS environment. Furthermore, we will evaluate the Predictive Fair Poller in a non-ideal radio environment and come up with improvements where needed.

Chapter 4

QoS in Bluetooth: an ideal radio environment

The previous chapter discussed the development of a Bluetooth polling mechanism that is able to poll slaves in a fair and efficient manner. This chapter discusses the development of a Bluetooth polling mechanism that provides QoS in an efficient manner. As opposed to the next chapter, we restrict ourself to an ideal radio environment where no transmission errors occur. The chapter is structured as follows. Section 4.1 presents the problem description. Section 4.2 describes the design of Guaranteed Service support for Bluetooth. Section 4.3 evaluates the proposed scheduling mechanisms, and Section 4.4 concludes this chapter.

4.1 Problem description

Bluetooth is a wireless access technology that was initially developed as a replacement for cables. However, Bluetooth has evolved to a wireless technology that can be used in new areas not envisaged before. We believe that audio and video will be involved in these new areas, and that applications dealing with audio and video will become available. Applications that deal with audio and video require a network that causes small packet delays or at least bounded packet delays. In order for Bluetooth to be useful to such applications, it must ensure that packet delays are low or at least bounded.

Bluetooth uses a polling mechanism to divide bandwidth among the participants. Together with error recovery, paging, and inquiry this polling scheme is highly determining with respect to the packet delay. Bluetooth Polling mechanisms capable of providing QoS have been studied in [ZCKD02, CH02, CSS01, LT02], and [MMG03]. However, these polling mechanisms either do not provide delay and rate guarantees, or they lack an admission control algorithm that would make sure the proposed mechanism can actually handle the QoS traffic. Bluetooth can also use SCO channels to transport some types of traffic (e.g. CBR traffic) that require delay bounds. However, SCO channels cannot transport large packets nor do they have retransmission possibilities.

4.1.1 The Guaranteed Service approach

As mentioned in Chapter 2, the Guaranteed Service approach [SSG97] (GS) makes use of the concept that packet delay in a network is a function of the arrival pattern of packets, the packet sizes, and the way these packets are served throughout the network. It states that if a flow is described using a token bucket [Par93] flow specification, and if each network element in the GS path computes and exports parameters that describe the way it provides a requested fluid model bandwidth R, then a delay bound d^{B} can be computed given a requested fluid model bandwidth R by

$$d^{\mathrm{B}} = \begin{cases} \frac{b-M}{R}\frac{r^{\mathrm{p}}-R}{r^{\mathrm{p}}-r^{\mathrm{t}}} + \frac{M+C_{\mathrm{tot}}}{R} + D_{\mathrm{tot}}, & r^{\mathrm{t}} \leq R < r^{\mathrm{p}}, \\ \frac{M+C_{\mathrm{tot}}}{R} + D_{\mathrm{tot}}, & r^{\mathrm{t}} \leq r^{\mathrm{p}} \leq R, \end{cases} \tag{4.1}$$

where r^{p}, r^{t}, b, M, R, C_{tot}, and D_{tot} are as defined in Section 2.2.1.1. If an application specifies its traffic using a token bucket traffic specification, and if the network elements in the GS path export their deviation from the fluid model, then, provided that $D_{\mathrm{tot}} < d^{\mathrm{B}}$, a fluid model service rate R can be requested such that a desired delay bound d^{B} is achieved.

4.1.2 Problem statement

The main problem addressed in this chapter, is the provisioning of QoS in Bluetooth in case of an ideal radio environment, where transmission errors do not occur. As Guaranteed Service guarantees a delay bound by providing a rate guarantee, the design of Guaranteed Service support for Bluetooth comprises the design of rate guarantees support for Bluetooth. These two types of guarantees are the main QoS types that are needed for audio and video applications.

The provisioning of Guaranteed Service in a network requires the source to provide a traffic specification and a desired delay bound, and it requires the receiver to calculate the proper bandwidth request. Furthermore, it requires the network elements in the GS path to compute and export their deviation from the fluid model, and it requires a mechanism (not necessarily RSVP) that transports all specifications and requests as well as the exported values, between the source, the destination, and the intermediate network elements. Note that a mechanism has been specified for the Bluetooth link, in the so-called Bluetooth logical link control and adaptation protocol (L2CAP), that can be used for exchanging traffic specifications and delay requirements. Finally, the provisioning of Guaranteed Service in a network requires the network elements to perform admission control, and to schedule the Guaranteed Service traffic as promised, i.e., to provide the requested rate, which leads to the requested delay guarantee.

Guaranteed Service is designed such that it can be used in a single-hop connection or in a fixed-route multi-hop connection, as long as each hop can perform the functions mentioned above. Consequently, the Guaranteed Service support to be designed and the accompanying guaranteed rate support can be used in any Bluetooth usage scenario, as long as scatternets are avoided. Note that Guaranteed Service does not scale in networks with large numbers of flows. As the numbers of flows within a Bluetooth piconet is low, the scalability problem of Guaranteed Service is avoided.

In this chapter, we design Guaranteed Service support for Bluetooth. More specifically, we focus on the determination of the C and D error terms, on the admission control, and on the scheduling of the Guaranteed Service traffic. As the C and D error terms and the admission control are directly related to the polling mechanism (i.e., scheduling mechanism), they are studied in the context of a polling mechanism.

In this chapter, we restrict ourselves to an ideal radio environment where no transmission errors occur and where retransmissions are not needed. We assume that no inquiry or paging procedures take place and thus that all the time slots are available for data transmission. Furthermore, we assume the availability of logical channels where a poll for a QoS (e.g. Guaranteed Service) flow cannot result in BE data to be transmitted, and where BE traffic and QoS traffic are queued separately to prevent BE traffic from interfering with QoS traffic within a node.

4.1.3 Related work

Zhu et. al. [ZCKD02] proposed a polling mechanism, which they named Adaptive Power-Conserving service discipline for Bluetooth (APCB). In this mechanism, the idea of the Virtual Clock service model [Zha91] is adopted and the master allocates bandwidth to flows based on their rate specification. Furthermore, APCB uses the hold mode to reduce the power consumptions of less active slaves. The authors present, for the N-th packet in a queue, and as function of N, a bound on the departure time of that packet. However, no bound is presented the number of packets in a queue. The Virtual Clock part of APCB is similar to an earlier work of Ruijs [Rui01], where each slave is associated with a counter that is updated taking the load into account. Furthermore, the counters are decreased by one every two time slots, and a slave is polled as soon as its associated counter reaches zero.

Chen and Hou [CH02] proposed a polling mechanism that provides temporal QoS in Bluetooth networks. The traffic of each flow i is described by the tuple (c_i,d_i), where c_i is the maximum amount of messages (in slots) that can arrive in any time interval d_i, and where d_i is also the relative deadline for a message. The latter means that a message of flow i arriving at time t must be transmitted by time $t + d_i$. Provided that the flows conform their specification, the polling mechanism tries to meet the relative deadlines as much as possible.

Chawla et. al. [CSS01] proposed a polling scheme, which they named latency-based scheduling. In this scheme, they define the QoS request of a flow i in the form of a maximum scheduling delay that each packet of that flow can tolerate. Based on this delay, which they termed as latency of the connection, a relative deadline d_i is defined for each packet belonging to that flow i. Furthermore, the polling scheme applies the Earliest Due Deadline scheme. The authors, however, did not present an admission control algorithm that would make sure the requested latency of the connection can actually be met.

Lapeyrie and Turletti [LT02] proposed a polling mechanism, which they named Fair and efficient Polling algorithm with QoS support (FPQ). This pollers aims at supporting both delay and bandwidth guarantees, while remaining fair and efficient in case the load is asymmetrically distributed among the slaves. Similar to the Predictive Fair Poller, FPQ determines the probability of data being available for a slave. Furthermore, it keeps track, for each slave, of the number of slots since the last poll. Based on these two aspects it decides which slave to poll next. In FPQ, the flows specify their traffic using the average interval between packets and the number of slots required by the transmission of packets. In order to support, for flow i, a maximum delay request of, for instance, $d_{\max_i}$, FPQ tries to make sure that the queue of flow i is emptied at least once every $d_{\max_i}$ time-units. However, as the flows do not specify the minimum interval between packets, it can not be guaranteed that requested maximum delay can actually be met.

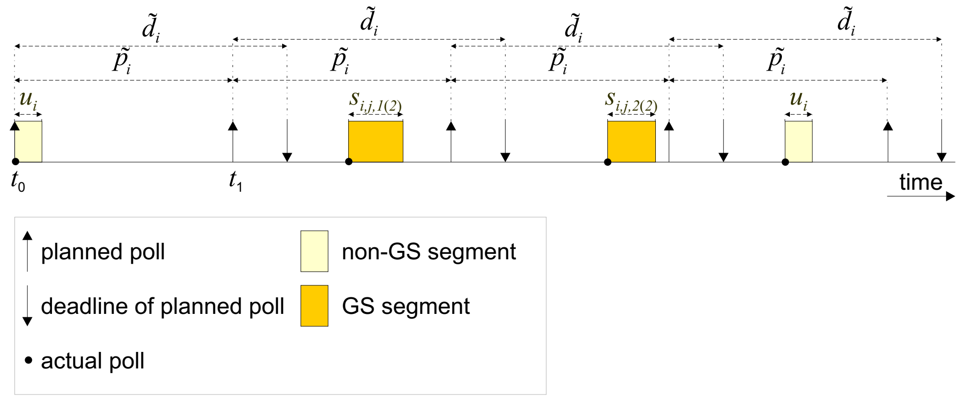


Figure 4.1: *Planning polls with a fixed time interval.*

Recently, Mercier et. al. [MMG03] proposed a service differentiating polling mechanism, which they name Class-Based EDF scheduling (CB-EDF). This mechanism aims at achieving good service differentiation by taking into account two QoS parameters, i.e., the importance of a message, and its relative delivery deadline. CB-EDF applies class-based priority queueing between classes and EDF (Earliest Deadline First) within a class, where a class contains all messages of the same importance degree. The CB-EDF is a service differentiating polling mechanism and provides no delay or rate guarantees.

In this work, we follow the IETF's Guaranteed Service approach, hence providing both a rate guarantee and a delay guarantee. Furthermore, we present an admission control required to ensure that the guarantees can actually be provided. To the author's knowledge, no other work has been reported on this approach.

4.2 Design of QoS support for Bluetooth

In Section 4.2.1, we introduce a polling mechanism that plans polls with a fixed interval in between. In Section 4.2.1.1, we present the admission control routine needed to make sure the fixed-interval poller can handle the GS flows as promised. In Section 4.2.1.2 and Section 4.2.1.3, we presents the determination of the parameters of the fixed-interval poller, and in Section 4.2.1.4 we present the translation of these parameters into the C and D error terms.

In Section 4.2.2, we show the shortcomings of the fixed-interval poller and introduce a variable-interval poller, which is an improved version of the fixed-interval poller. Finally, we present an improvement of the admission control in Section 4.2.3.

4.2.1 Fixed-interval polling

Given the requested bandwidth (R_i) and the token bucket specification (r_i^{t}, b_i, r_i^{p}, m_i, M_i) of a GS flow i, the poll rate that must be supported can be computed. An obvious way to poll at a given poll rate is to calculate the average inter-poll time that results in the given poll rate, and to plan polls with a time spacing $\tilde{p}_i$ (poll period) equal to the calculated average inter-poll time. Each planned poll must complete execution within a relative deadline $\tilde{d}_i$ from its planned time.

Figure 4.1 shows an example of fixed-interval polling, where polls for GS flow i are planned with a fixed interval $\tilde{p}_i$. Furthermore, each planned poll must be completely executed within a relative deadline $\tilde{d}_i$ from its planned time. For instance, the poll planned for t_1 results in a GS segment $(i, j, 1(2))$ to be completely transmitted before before $t_1 + \tilde{d}_i$. In Figure 4.1, $s_{i,j,k(l_{i,j})}$ is the transmission time (duration of both upstream and downstream baseband packet) of the k-th segment out of $l_{i,j}$ segments of the j-th packet that belongs to flow i. Furthermore, u_i is the transmission time following an unsuccessful poll for flow i, where an unsuccessful poll for flow i is a poll, for the node associated with flow i, that did not result in data belonging to flow i. The time at which a poll takes place corresponds to the time at which a master-to-slave transmission starts. In Bluetooth, the slave starts its transmission at least one time slot (0.625 ms) after the master started its transmission, dependent on whether the master transmitted one, three or five slots to the slave. Consequently, for that poll to results in data to be transmitted from the slave to the master, that slave does not necessarily has to have its data available for transmission at the time the master starts its transmission to that slave. However, we require in our study the data to be available at the time the master start its transmission. For instance, in Figure 4.1, data that becomes available at t_0^+ ($= t_0 + \delta$, where $\delta \downarrow 0$) will not be served as a result of the poll at t_0, but has to wait for the next poll.

4.2.1.1 Admission Control

Each traffic specification (r_i^{t}, b_i, r_i^{p}, m_i, M_i) and corresponding requested fluid model bandwidth (R_i) is ultimately converted to a poll period $\tilde{p}_i$, a relative deadline $\tilde{d}_i$ and a maximum segment size $s_i^{\max} = \max_{j,k} s_{i,j,k(l_{i,j})}$. Consequently, a GS flow i can be looked at as a periodic task. A periodic task τ_i that corresponds to GS flow i is represented by the tuple (p_i, e_i, d_i), where

- $p_i = \tilde{p}_i$ is the fixed interval (period) between two consecutive instances of a task τ_i
- $e_i = s_i^{\max}$ is the maximum execution time of an instance of task τ_i
- $d_i = \tilde{d}_i$ is the relative deadline of each instance of task τ_i

The size of the largest possible non-GS segment is given by $s_{\mathrm{BE}}^{\max}$. Hence, the size of the largest possible segment in the piconet is given by $s^{\max} = \max(s_i^{\max}, s_{\mathrm{BE}}^{\max})$, while the corresponding maximum execution time of any task instance (including non-real-time tasks) is given by $e^{\max} = s^{\max}$.

The decision whether a set of n GS flows can be accepted can be done by deciding whether the corresponding task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ can be accepted (feasibility analysis). This decision can only be made if the scheduling policy is also known. The Bluetooth polling mechanism can be modeled as a non-preemptive scheduling policy, which on its turn can be divided into two classes: the class of idling non-preemptive scheduling policies and the class of non-idling non-preemptive scheduling policies. The idling non-preemptive scheduling policies are allowed to insert idle times even if there are task instances waiting for execution. Inserting idle times makes it sometimes possible to schedule task sets that could otherwise not be scheduled under the class of non-idling scheduling policies [GMR95].

As the feasibility analysis of an idling non-preemptive schedule is NP-hard in the strong sense [HV95], we decided to use a non-idling non-preemptive scheduling policy. It is shown in [KN80], [JSM91], and [GMR95] that non-idling non-preemptive Earliest Deadline First (EDF) is optimal among the class of non-idling non-preemptive scheduling policies. This means that if a feasible non-idling non-preemptive scheduling policy exists for a given task set, then non-idling non-preemptive EDF will also be feasible for that task set.

Now that we have decided that the planned polls will be executed according to the non-idling non-preemptive EDF scheduling policy, the admission control of a set of n GS flows can be translated to the feasibility analysis of the corresponding task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ under the non-idling non-preemptive EDF scheduling policy.

Let us introduce *processor utilization* U and *processor demand* $h(t)$. Processor utilization U is the fraction of processor time needed for the execution of task set τ [LL73]. Processor demand $h(t)$ is the amount of execution time requested by all instances whose release times and absolute deadlines are in the interval $[0, t]$ [BMR90][Spu96]. Zheng et al. stated in [ZS94] that in the presence of non real-time tasks, a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is schedulable under the non-idling non-preemptive EDF scheduling policy if and only if

1. Processor utilization condition

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1, \text{ and} \tag{4.2}$$

2. Processor demand condition

$$\forall t \in S, h(t) = \sum_{i: d_i \leq t} \left(\left\lfloor \frac{t - (d_i - p_i)}{p_i} \right\rfloor \right) e_i + e^{\max} \leq t, \tag{4.3}$$

where

$$S = \bigcup_{i=1}^{n} \left\{ d_i + np_i | n = 0, 1, \ldots, \left\lfloor \frac{t_{\max} - d_i}{p_i} \right\rfloor \right\}, \tag{4.4}$$

and

$$t_{\max} = \max \left\{ d_1, \ldots, d_n, \frac{e^{\max} + \sum_{i=1}^{n} (1 - d_i/p_i) e_i}{1 - U} \right\}. \tag{4.5}$$

The first condition ensures that the scheduler can handle the traffic on the long term. The second condition ensures that the deadlines can actually be met. This is done by ensuring that, for time instances t that belong to S, the processor demand $h(t)$ is not higher than the processor time t. Set S denotes, for interval $(0, t_{\max})$, the time instances at which the deadlines occur. The idea behind the restriction to interval $(0, t_{\max})$ is that if the processor demand condition is met for the deadlines in the interval $(0, t_{\max})$, that condition will also be met for deadlines outside that interval.

In the absence of non real-time tasks, the feasibility conditions mentioned above are sufficient but not necessary. For the case in which there are no non real-time tasks, sufficient and necessary conditions can be found in [KN80], [JSM91], [GRS96] and [JL99].

4.2.1.2 Determining poll period $\tilde{p}_i$

The poll period $\tilde{p}_i$ is determined considering the worst case response time a packet can experience. In Figure 4.1, consider an empty packet queue at t_0, and consider packet j of flow i, with a size $L_{i,j}$ (in bytes), which will be broken up into $l_{i,j}$ segments. If this packet becomes available at t_0^+, then it will not be served during the poll at t_0, but it will be served during the next poll, which is planned for t_1. As a result, the worst case service time of a packet is $l_{i,j}\tilde{p}_i + \tilde{d}_i$. In order to let the poll period $\tilde{p}_i$ be inversely proportional to the requested fluid model bandwidth R_i, the relative deadline $\tilde{d}_i$ of a planned poll should be considered as a deviation from the fluid model service time. The remaining part of the worst case service time of a packet should then not be larger than the fluid model service time, i.e.,

$$l_{i,j}\tilde{p}_i \leq \frac{L_{i,j}}{R_i}, \quad m_i \leq L_{i,j} \leq M_i, \tag{4.6}$$

and thus

$$\tilde{p}_i \leq \frac{\frac{L_{i,j}}{l_{i,j}}}{R_i}, \quad m_i \leq L_{i,j} \leq M_i. \tag{4.7}$$

Let us introduce the poll efficiency $\epsilon_{\mathrm{p}_{i,j}}$, which is the average number of bytes per poll that is associated with packet j of flow i. The poll efficiency $\epsilon_{\mathrm{p}_{i,j}}$ is a result of the size $L_{i,j}$ of packet j of flow i, the segmentation policy that is followed, and the set of baseband packet types that is allowed to be used. The minimum poll efficiency of a flow i taken over all possible packet sizes (i.e., for $m_i \leq L_{i,j} \leq M_i$) is

$$\epsilon_{\mathrm{p}_i}^{\min} = \min_{m_i \leq L_{i,j} \leq M_i} \frac{L_{i,j}}{l_{i,j}}. \tag{4.8}$$

Consequently, the maximum poll period that always satisfies (4.7) is

$$\tilde{p}_i = \frac{\epsilon_{\mathrm{p}_i}^{\min}}{R_i}. \tag{4.9}$$

4.2.1.3 Determining relative deadline $\tilde{d}_i$

As mentioned before, it has been decided to consider the relative deadline $\tilde{d}_i$ as the deviation from the fluid model service time. In Guaranteed Service, the deviation from the fluid model is expressed in terms of a rate-dependent deviation (C error term) and a rate-independent deviation (D error term). Hence, relative deadline $\tilde{d}_i$ should consist of a rate-dependent part and a rate-independent part, i.e.,

$$\tilde{d}_i = \alpha_i \tilde{p}_i + \beta_i, \tag{4.10}$$

where α_i and β_i are fixed for flow i, and where $\tilde{p}_i$ is inversely proportional to service rate R_i (cf. (4.9)). While $s^{\max}$ is the size of the largest possible segment in the piconet, we show in step 1 that choosing $\alpha_i \geq 1$ and $\beta_i \geq s^{\max}$ for each GS flow i simplifies the admission control (feasibility analysis) as, in that case, the processor utilization condition becomes a necessary and sufficient condition. In step 2, we introduce the individual processor utilization $\tilde{U}_i$ and the upper bound on the individual processor demand $\tilde{h}_i^{\mathrm{B}}(t)$. In step 3, we determine the derivatives of $\tilde{U}_i$ and $\tilde{h}_i^{\mathrm{B}}(t)$ with respect to α_i and β_i in the two operational areas of a GS flow i. Based on these derivatives, we present some observations in step 4. Based on these observations, we show in step 5 that increasing α_i and β_i beyond $\alpha_i = 1$ and $\beta_i = s^{\max}$ possibly decreases, but certainly does not increase the number of flows that can be accepted. Furthermore, we show that, with respect to the feasibility of a set of GS flows that flow i will be part of, the effect of decreasing α_i and β_i below $\alpha_i = 1$ and $\beta_i = s^{\max}$ cannot be determined a priori.

Step 1

Consider a set of n GS flows where $\alpha_i \geq 1$ and $\beta_i \geq s^{\max}$ for each GS flow i. The processor utilization of a set of n GS flows is given by

$$\tilde{U} = \sum_{i=1}^{n} \frac{s_i^{\max}}{\tilde{p}_i}, \tag{4.11}$$

while the resulting processor demand is then given by

$$\tilde{h}(t) = \sum_{i:\tilde{d}_i \leq t} \left(\left\lfloor \frac{t - (\tilde{d}_i - \tilde{p}_i)}{\tilde{p}_i} \right\rfloor \right) s_i^{\max} + s^{\max}, \tag{4.12}$$

where the summation takes place over the GS flows i for which $\tilde{d}_i \leq t$.
As $\alpha_i \geq 1$ and $\beta_i \geq s^{\max}$, it follows from (4.10) that

$$\tilde{d}_i \geq \tilde{p}_i + s^{\max}. \tag{4.13}$$

Substituting $\tilde{d}_i$ from (4.13) in (4.12) gives

$$\tilde{h}(t) \leq \sum_{i:\tilde{d}_i \leq t} \left(\left\lfloor \frac{t - s^{\max}}{\tilde{p}_i} \right\rfloor \right) s_i^{\max} + s^{\max}. \tag{4.14}$$

Removing the floor function ($\lfloor . \rfloor$) gives

$$\begin{aligned} \tilde{h}(t) &\leq \sum_{i:\tilde{d}_i \leq t} \left(\frac{t - s^{\max}}{\tilde{p}_i} \right) s_i^{\max} + s^{\max} \\ &= (t - s^{\max}) \sum_{i:\tilde{d}_i \leq t} \left(\frac{s_i^{\max}}{\tilde{p}_i} \right) + s^{\max}. \end{aligned} \tag{4.15}$$

It follows from (4.15) that

$$\tilde{h}(t) \leq (t - s^{\max}) \sum_{i=1}^{n} \left(\frac{s_i^{\max}}{\tilde{p}_i} \right) + s^{\max}, \quad t \geq s^{\max}. \tag{4.16}$$

Substituting $\tilde{U}$ from (4.11) in (4.16) gives

$$\tilde{h}(t) \leq \tilde{U}(t - s^{\max}) + s^{\max}, \quad t \geq s^{\max}. \tag{4.17}$$

During admission control, the processor demand function is examined only for $t \geq \min_i \tilde{d}_i \geq s^{\max}$ (see (4.4) and (4.13)). From (4.17), it follows that during admission control $\tilde{h}(t) \leq t$ if $\tilde{U} \leq 1$, which implies that the set of GS flows is then schedulable as both the processor utilization condition and the processor demand condition are met. Furthermore, if $U > 1$ then the set of GS flows is not schedulable as the processor utilization condition is not met in that case. Summarizing, if (4.13) is met, then the set of n GS flow is schedulable if and only if $U \leq 1$.

Step 2

Let us define the individual processor utilization of GS flow i as the contribution of GS flow i to the processor utilization, i.e.,

$$\tilde{U}_i \quad = \quad \frac{s_i^{\max}}{\tilde{p}_i}, \tag{4.18}$$

where summation of the individual processor utilizations over all GS flows gives the total processor utilization, i.e.,

$$\tilde{U} = \sum_{i=1}^{n} \tilde{U}_i. \tag{4.19}$$

Furthermore, let us define the upper bound on the processor demand of set of n GS flows as

$$\tilde{h}^{\mathrm{B}}(t) = \sum_{i:\tilde{d}_i \leq t} \left(\frac{t - (\tilde{d}_i - \tilde{p}_i)}{\tilde{p}_i} \right) s_i^{\max} + s^{\max}. \tag{4.20}$$

Finally, let us define the upper bound on the individual processor demand of GS flow i as the contribution of GS flow i to the upper bound on the processor demand, i.e.,

$$\tilde{h}_i^{\mathrm{B}}(t) \quad = \quad \begin{cases} \frac{t-(\tilde{d}_i - \tilde{p}_i)}{\tilde{p}_i} s_i^{\max}, & t \geq (\tilde{d}_i - \tilde{p}_i), \\ \\ 0, & t < (\tilde{d}_i - \tilde{p}_i), \end{cases} \tag{4.21}$$

where summation of $s^{\max}$ and the upper bounds on the individual processor demands over all GS flows gives the total upper bound on the processor demand, i.e.,

$$\tilde{h}^{\mathrm{B}}(t) = \sum_{i=1}^{n} h_i^{\mathrm{B}}(t) + s^{\max}. \tag{4.22}$$

Step 3

In order to determine the effect of modifying α_i and β_i on the schedulability of a set of GS flows that flow i is (or will be) part of, the derivatives of both the individual processor utilization and the upper bound on the individual processor demand with respect to both α_i and β_i will be determined. For that, the individual processor utilization and the upper bound on the individual processor demand have to be presented as a function of α_i and β_i. This is achieved by first presenting the delay bound d_i^{B} as a function of α_i and β_i. The delay bound d_i^{B} from (4.1) can be rewritten as

$$d_i^{\mathrm{B}} = x_i \frac{r_i^{\mathrm{p}} - R_i}{R_i} + \frac{M_i}{R_i} + \frac{C_{\mathrm{tot}_i}}{R_i} + D_{\mathrm{tot}_i}, \tag{4.23}$$

where

$$x_i = \begin{cases} \frac{b_i - M_i}{r_i^{\mathrm{p}} - r_i^{\mathrm{t}}}, & r_i^{\mathrm{t}} \leq R_i < r_i^{\mathrm{p}}, \\ 0, & r_i^{\mathrm{t}} \leq r_i^{\mathrm{p}} \leq R_i. \end{cases} \tag{4.24}$$

As we already decided to consider the relative deadline $\tilde{d}_i$ as the error term of the Bluetooth node under consideration, substitution of $\tilde{d}_i$ from (4.10) in (4.23) gives

$$d_i^{\mathrm{B}} = x_i \frac{r_i^{\mathrm{p}} - R_i}{R_i} + \frac{M_i}{R_i} + \alpha_i \tilde{p}_i + \beta_i + \frac{C_{\mathrm{rem}_i}}{R_i} + D_{\mathrm{rem}_i}, \tag{4.25}$$

where C_{rem_i} and D_{rem_i} are the error terms of the remaining nodes in the GS path of GS flow i.

The requested fluid model service rate R_i should be at least equal to the token rate. Consequently, two complementary operational areas can be distinguished, depending on whether the requested rate that follows from (4.25) is lower than the token rate r_i^{t}, or not.

Operational area I: In this operational area, the requested service rate that follows from (4.25) is lower than the token rate r_i^{t}. Consequently, the requested service rate is set equal to the token rate, i.e.,

$$R_i = r_i^{\mathrm{t}}. \tag{4.26}$$

By substituting R_i from (4.26) in (4.9) the poll period can be written as

$$\tilde{p}_i = \frac{\epsilon_{\mathrm{p}i}^{\mathrm{min}}}{r_i^{\mathrm{t}}}. \tag{4.27}$$

By substituting $\tilde{p}_i$ from (4.27) in (4.10) the relative deadline can be written as

$$\tilde{d}_i = \alpha_i \tilde{p}_i + \beta_i = \alpha_i \epsilon_{\mathrm{p}i}^{\mathrm{min}} \frac{1}{r_i^{\mathrm{t}}} + \beta_i. \tag{4.28}$$

By substituting $\tilde{p}_i$ from (4.27) in (4.18) the individual processor utilization of GS flow i can be written as

$$\tilde{U}_i = \frac{s_i^{\max} r_i^{\mathrm{t}}}{\epsilon_{\mathrm{p}i}^{\min}}. \tag{4.29}$$

By substituting $\tilde{p}_i$ from (4.27) in (4.21) the upper bound on the individual processor demand of GS flow i can be written as

$$\tilde{h}_i^{\mathrm{B}}(t) = \begin{cases} \frac{s_i^{\max} r_i^{\mathrm{t}}}{\epsilon_{\mathrm{p}i}^{\min}}(t-\beta_i) - (\alpha_i - 1)s_i^{\max}, & t \geq (\tilde{d}_i - \tilde{p}_i), \\ 0, & t < (\tilde{d}_i - \tilde{p}_i). \end{cases} \tag{4.30}$$

It follows that the derivatives of the individual processor utilization with respect to α_i and β_i are given by

$$\frac{\partial \tilde{U}_i}{\partial \alpha_i} = 0, \tag{4.31}$$

and

$$\frac{\partial \tilde{U}_i}{\partial \beta_i} = 0. \tag{4.32}$$

It also follows that the derivatives of the upper bound on the individual processor demand with respect to α_i and β_i are given by

$$\frac{\partial \tilde{h}_i^{\mathrm{B}}(t)}{\partial \alpha_i} = \begin{cases} -s_i^{\max}, & t \geq (\tilde{d}_i - \tilde{p}_i), \\ 0, & t < (\tilde{d}_i - \tilde{p}_i), \end{cases} \tag{4.33}$$

and

$$\frac{\partial \tilde{h}_i^{\mathrm{B}}(t)}{\partial \beta_i} = \begin{cases} -\frac{s_i^{\max} r_i^{\mathrm{t}}}{\epsilon_{\mathrm{p}i}^{\min}}, & t \geq (\tilde{d}_i - \tilde{p}_i), \\ 0, & t < (\tilde{d}_i - \tilde{p}_i). \end{cases} \tag{4.34}$$

Operational area II: In this operational area, the requested rate that follows from (4.25) is not lower than the token rate r_i^{t}. Consequently, the requested service rate is given by

$$R_i = \frac{x_i r_i^{\mathrm{p}} + M_i + \alpha_i \epsilon_{\mathrm{p}i}^{\min} + C_{\mathrm{rem}_i}}{d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i}}. \tag{4.35}$$

By substituting R_i from (4.35) in (4.9) the poll period can be written as

$$\tilde{p}_i = \frac{\epsilon_{\mathrm{p}i}^{\min}(d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i})}{x_i r_i^{\mathrm{p}} + M_i + \alpha_i \epsilon_{\mathrm{p}i}^{\min} + C_{\mathrm{rem}_i}}, \tag{4.36}$$

while by substitution of $\tilde{p}_i$ from (4.36) in (4.10) the relative deadline can be written as

$$\tilde{d}_i = \alpha_i \tilde{p}_i + \beta_i = \frac{\alpha_i \epsilon_{\mathrm{p}i}^{\min}(d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i})}{x_i r_i^{\mathrm{p}} + M_i + \alpha_i \epsilon_{\mathrm{p}i}^{\min} + C_{\mathrm{rem}_i}} + \beta_i. \tag{4.37}$$

By substituting $\tilde{p}_i$ from (4.36) in (4.18) the individual processor utilization of GS flow i can be written as

$$\tilde{U}_i = \frac{s_i^{\max}(x_i r_i^{\mathrm{p}} + M_i + \alpha_i \epsilon_{\mathrm{p}i}^{\min} + C_{\mathrm{rem}_i})}{\epsilon_{\mathrm{p}i}^{\min}(d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i})}. \tag{4.38}$$

By substituting $\tilde{p}_i$ from (4.36) in (4.21) the upper bound on the individual processor demand of GS flow i can be rewritten as

$$\tilde{h}_i^{\mathrm{B}}(t) = \begin{cases} s_i^{\max}\left(\frac{(t-\beta_i)(x_i r_i^{\mathrm{p}} + M_i + \alpha_i \epsilon_{\mathrm{p}i}^{\min} + C_{\mathrm{rem}_i})}{\epsilon_{\mathrm{p}i}^{\min}(d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i})} + 1 - \alpha_i\right), & t \geq (\tilde{d}_i - \tilde{p}_i), \\ 0, & t < (\tilde{d}_i - \tilde{p}_i). \end{cases} \tag{4.39}$$

The derivatives of the individual processor utilization with respect to α_i and β_i are given by

$$\frac{\partial \tilde{U}_i}{\partial \alpha_i} = \frac{s_i^{\max}}{d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i}}, \tag{4.40}$$

and

$$\frac{\partial \tilde{U}_i}{\partial \beta_i} = \frac{s_i^{\max}(x r_i^{\mathrm{p}} + M_i + \alpha_i \epsilon_{\mathrm{p}i}^{\min} + C_{\mathrm{rem}_i})}{\epsilon_{\mathrm{p}i}^{\min}(d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i})^2}. \tag{4.41}$$

The derivatives of the upper bound on the individual processor demand with respect to α_i and β_i are given by

$$\frac{\partial \tilde{h}_i^{\mathrm{B}}(t)}{\partial \alpha_i} = \begin{cases} \frac{s_i^{\max}(t - t_{\mathrm{r}_i})}{d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i}}, & t \geq (\tilde{d}_i - \tilde{p}_i), \\ 0, & t < (\tilde{d}_i - \tilde{p}_i), \end{cases} \tag{4.42}$$

and

$$\frac{\partial \tilde{h}_i^{\mathrm{B}}(t)}{\partial \beta_i} = \begin{cases} \frac{s_i^{\max}(t - t_{\mathrm{r}_i})(x r_i^{\mathrm{p}} + M_i + \alpha_i \epsilon_{\mathrm{p}i}^{\min} + C_{\mathrm{rem}_i})}{(d_i^{\mathrm{B}} + x_i - \beta_i - D_{\mathrm{rem}_i})^2 \epsilon_{\mathrm{p}i}^{\min}}, & t \geq (\tilde{d}_i - \tilde{p}_i), \\ 0, & t < (\tilde{d}_i - \tilde{p}_i), \end{cases} \tag{4.43}$$

where $t_{\mathrm{r}_i} = (d_i^{\mathrm{B}} + x_i - D_{\mathrm{rem}_i})$ is the point t at which $h_i^{\mathrm{B}}(t)$ remains unchanged when α_i and/or β_i are modified. Furthermore, for $t < t_{\mathrm{r}_i}$ and $t > t_{\mathrm{r}_i}$, $h_i^{\mathrm{B}}(t)$ changes in opposite directions when α_i and/or β_i are modified.

Step 4

Based on the above, the following observations can be made:

a. In both operational area I and operational area II, the derivatives of the individual process utilization $\frac{\partial U_i}{\partial \alpha_i}$ and $\frac{\partial U_i}{\partial \beta_i}$ are non-negative.

b. In operational area I, if $t \geq (\tilde{d}_i - \tilde{p}_i)$, then the derivatives of the upper bound on the individual processor demand $\frac{\partial \tilde{h}_i^{\mathrm{B}}}{\partial \alpha_i}(t)$ and $\frac{\partial \tilde{h}_i^{\mathrm{B}}}{\partial \beta_i}(t)$ are strictly negative.

c1. In operational area II, if $t > (d_i^{\mathrm{B}} + x_i - D_{\mathrm{rem}_i})$ and $t \geq (\tilde{d}_i - \tilde{p}_i)$, then the derivatives of the upper bound on the individual processor demand $\frac{\partial \tilde{h}_i^{\mathrm{B}}}{\partial \alpha_i}(t)$ and $\frac{\partial \tilde{h}_i^{\mathrm{B}}}{\partial \beta_i}(t)$ are strictly positive.

c2. In operational area II, if $(\tilde{d}_i - \tilde{p}_i) \leq t < (d_i^{\mathrm{B}} + x_i - D_{\mathrm{rem}_i})$, then the derivatives of the upper bound on the individual processor demand $\frac{\partial \tilde{h}_i^{\mathrm{B}}}{\partial \alpha_i}(t)$ and $\frac{\partial \tilde{h}_i^{\mathrm{B}}}{\partial \beta_i}(t)$ are strictly negative.

d. From (4.3) and (4.20) it follows that an increase of $\tilde{h}^{\mathrm{B}}(t)$ never leads to a decrease of $\tilde{h}(t)$.

Step 5
In order to show that it is not advantageous to increase $\tilde{d}_i$ beyond $\tilde{d}_i = \tilde{p}_i + s^{\max}$ consider a set of n GS flows for which $\alpha_i = 1$ and $\beta_i = s^{\max}$ for each GS flow i and which cannot be admitted. This means that the processor utilization exceeds unity ($U > 1$) as the processor utilization condition is a necessary and sufficient condition when $\alpha_i \geq 1$ and $\beta_i \geq s^{\max}$. According to observation (a), increasing α_i and/or β_i never decreases the processor utilization. In other words, increasing α_i and/or β_i will not help in making the set of n GS flows schedulable.

In order to show that the effect, of modifying α_i and β_i of a GS flow i, on the admission of a set of n GS flows that GS flow i belongs (or will belong) to cannot be determined a priori, consider a set of n GS flows for which the processor utilization condition is met. Furthermore, assume that a single bottleneck exists at t_{b} at which $\tilde{h}^{\mathrm{B}}(t_{\mathrm{b}}) = \tilde{h}(t_{\mathrm{b}}) = \lim_{t \downarrow t_{\mathrm{b}}} t$, and that the set of n GS flows can thus not be admitted.

Now consider a GS flow i that is one of the n GS flows. According to observation (c1), if GS flow i operates in operational area II, and if $t_{\mathrm{b}} > (\tilde{d}_i - \tilde{p}_i)$ and $t_{\mathrm{b}} > (d_i^{\mathrm{B}} + x_i - D_{\mathrm{rem}_i})$, then increasing α_i and/or β_i will increase $\tilde{h}_i^{\mathrm{B}}(t_{\mathrm{b}})$ and thus $\tilde{h}^{\mathrm{B}}(t_{\mathrm{b}})$. According to observation (d), $\tilde{h}(t_{\mathrm{b}})$ will not decrease and the bottleneck at t_{b} remains. On the other hand, decreasing α_i and/or β_i decreases $\tilde{h}_i^{\mathrm{B}}(t_{\mathrm{b}})$ and thus $\tilde{h}^{\mathrm{B}}(t_{\mathrm{b}})$. Since by definition $\tilde{h}^{\mathrm{B}}(t_{\mathrm{b}}) \geq \tilde{h}(t_{\mathrm{b}})$, $\tilde{h}(t_{\mathrm{b}})$ will also decrease. Consequently, decreasing α_i and/or β_i (of GS flow i) is a necessary condition to let the set of n GS flows be schedulable.

Finally, consider a GS flow i that is one of the n GS flows. According to observation (c2), if GS flow i operates in operational area II, and if $(\tilde{d}_i - \tilde{p}_i) \leq t_{\mathrm{b}} < (d_i^{\mathrm{B}} + x_i - D_{\mathrm{rem}_i})$, then decreasing α_i and/or β_i will increase $\tilde{h}_i^{\mathrm{B}}(t_{\mathrm{b}})$ and thus $\tilde{h}^{\mathrm{B}}(t_{\mathrm{b}})$. According to observation (d), $\tilde{h}(t_{\mathrm{b}})$ will not decrease and the bottleneck at t_{b} remains. On the other hand, increasing α_i and/or β_i decreases $\tilde{h}_i^{\mathrm{B}}(t_{\mathrm{b}})$ and thus $\tilde{h}^{\mathrm{B}}(t_{\mathrm{b}})$. Since by definition $\tilde{h}^{\mathrm{B}}(t_{\mathrm{b}}) \geq \tilde{h}(t_{\mathrm{b}})$, $\tilde{h}(t_{\mathrm{b}})$ will also decrease. Consequently, increasing α_i and/or β_i (of GS flow i) is a necessary condition to let the set of n GS flows be schedulable. The same applies if GS flow i operates in operational area I while $t_{\mathrm{b}} \geq (\tilde{d}_i - \tilde{p}_i)$.

Summarizing, in order to determine α_i and β_i for a GS flow i such that a set of GS flows that GS flow i belongs (or will belong) to can be admitted, it must be known whether GS flow i operates (or will operate) in operational area I or II. Furthermore, the place of potential bottlenecks must be known. Finally, the point $t_{\mathrm{r}_i} = (d_i^{\mathrm{B}} + x_i - D_{\mathrm{rem}_i})$ must be known, where D_{rem_i} is the cumulative rate independent deviation from the fluid model of the remaining hops of the path of GS flow i ($D_{\mathrm{rem}_i} = 0$ in case of a single hop GS flow i). As none of these aspects are known a priori, the best value of α_i and β_i cannot be determined a priori.

Based on the previous, it is decided to set the relative deadline of each GS flow i at

$$\tilde{d}_i = \tilde{p}_i + s^{\max}. \tag{4.44}$$

4.2.1.4 Exporting C and D error terms

As mentioned in Section 4.2.1.2, it is decided to consider the relative deadline $\tilde{d}$ as the C and D error terms, i.e.,

$$\frac{C_i}{R_i} + D_i = \tilde{d}_i. \tag{4.45}$$

Substituting $\tilde{d}_i$ from (4.44) in (4.45) gives

$$\frac{C_i}{R_i} + D_i = \tilde{p}_i + s^{\max}. \tag{4.46}$$

Substituting $\tilde{p}_i$ from (4.9) in (4.46) give

$$\frac{C_i}{R_i} + D_i = \frac{\epsilon_{\mathrm{p}_i}^{\min}}{R_i} + s^{\max}. \tag{4.47}$$

The C error term is the rate-dependent deviation from the fluid model, whereas the D error term is the rate-independent deviation from the fluid model. From (4.47), it follows that $C_i = \epsilon_{\mathrm{p}_i}^{\min}$ and $D_i = s^{\max}$.

Note that with respect to multi-hop GS flows, forwarding intermediate nodes should account for packetization as packets are not sent forward to the next node before they are completely received from the previous node. Consequently, if the Bluetooth hop is not the first hop for a multi-hop GS flow i, it should increase C_i by M_i in order to account for packetization.

4.2.2 Variable-interval polling

The fixed-interval poller of Section 4.2.1 plans polls for a GS flow i with a fixed interval $\tilde{p}_i$. The poll interval $\tilde{p}_i$ is determined taking into account the packet size L_{p_i} that is associated with the least number of bytes per poll (minimum poll efficiency). This leads to the following drawbacks:

a. The range of packet sizes may comprise several packet sizes (i.e, if $M_i > m_i$). In that case, interval $\tilde{p}_i$ is too small when packet sizes other than L_{p_i} are used, and GS flow i is then polled more often than necessary.

b. If a planned poll for GS flow i is executed, the next poll for GS flow i will be planned for $\tilde{p}_i$ after the last time a poll for GS flow i was planned for, even if that poll did not result in a GS segment of flow i.

c. Planned polls are executed even if it is known that no GS traffic is available. As the master has only knowledge about the availability of traffic that is directed from the master to a slave, this drawback only applies to GS flows from the master to a slave.

These drawbacks do not adversely affect the performance of the GS flows. On the contrary, polling a GS flow more often than necessary will decrease the average delay of its packets. However, polling the GS flows more often than needed consumes the resources that could otherwise be used for retransmissions (in a non-ideal radio environment) and/or for transmission of best effort traffic. We propose three improvements to eliminate these drawbacks (see Figure 4.2):

a. If a poll for GS flow i resulted in a last segment of a packet j with size $L_{i,j}$, then plan the next poll $\frac{L_{i,j}}{R_i}$ time-units after the planned time of the poll that resulted in the first segment of packet j of GS flow i. Hence, postpone the next poll $g_{i,j}$ time-units, where (see also (4.6))

$$g_{i,j} = \frac{L_{i,j}}{R_i} - l_{i,j}\tilde{p}_i \geq 0. \tag{4.48}$$

For instance, in Figure 4.2, the poll planned for t_1, and which is executed at t_2, resulted in the first GS segment of packet j of flow i. The poll planned for t_3, and which is executed at t_4 resulted in the last GS segment of the same packet. Consequently, the next poll is planned for $t_6 = t_1 + \frac{L_{i,j}}{R_i}$ rather than for $t_5 = t_3 + \tilde{p}_i$. In other words, the poll that was originally to be planned for t_5 has been postponed $g_{i,j} = \frac{L_{i,j}}{R_i} - 2\tilde{p}_i$ time-units.

b. If a poll for GS flow i did not result in a GS segment of flow i, then obviously no GS segment of flow i was available before that actual poll time. As a result, plan the next poll a time period $\tilde{p}_i$ after the actual time of the last poll for flow i rather than a time period $\tilde{p}_i$ after its planned time.

For instance, in Figure 4.2, the poll planned for t_6 is executed at t_7. As it results in a non-GS segment, the next poll is planned for $t_8 = t_7 + \tilde{p}_i$ rather than for $t_6 + \tilde{p}_i$.

c. If at a planned poll time t_n the poller finds that there is no GS traffic to serve, then that poll is skipped, and the next poll is planned for $t_n + \tilde{p}_i$. Since the poller has only knowledge of traffic from the master to the slave, this improvement only applies to GS flows that are directed from the master to the slave (not shown in Figure 4.2).

If the source of GS flow i offers its data using the packet size that leads to the minimum poll efficiency ($\epsilon_{\mathrm{p}_i}^{\min}$), then the next poll after each last segment is planned for exactly $\tilde{p}_i$ time-units after the last time a poll was planned for (i.e., $g_{i,j} = 0$). The determination of $\tilde{p}_i$, $\tilde{d}_i$, C_i and D_i should take this worst case into account, hence they are the same as for the poller presented in Section 4.2.1. Furthermore, the task set corresponding to the set of GS flows becomes a sporadic task set, which is similar to a periodic task, except that the former is parameterized by the minimum interval between consecutive task instances instead of by a fixed interval [GRS96]. The feasibility analysis for such a set is the same as for a periodic task set, except that the minimum interval between consecutive instances of a task is now taken into account. As the minimum interval is the same as the fixed interval determined in Section 4.2.1, the admission control is the same as described in Section 4.2.1.1.

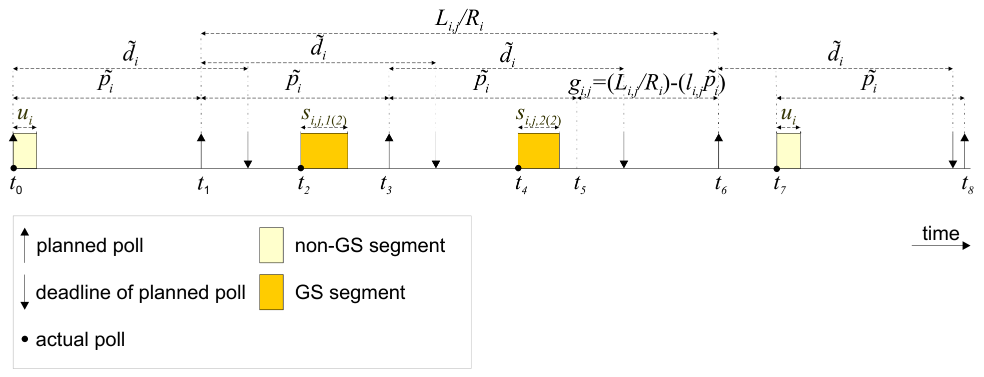


Figure 4.2: *Planning polls with a variable time interval.*

4.2.3 Improvement of the admission control

We assume the availability of logical channels distinguishing between QoS traffic and best effort traffic, and that QoS traffic always has priority over best effort traffic. Consequently, a poll for a GS flow in one direction also gives the opportunity to transmit GS traffic of the same logical channel in the opposite direction. In other words, each GS poll of a slave implies an opportunity to transmit GS traffic of the same logical channel in both directions. Taking this fact into account, we improve the admission control in order to be able to accept more flows.

Consider a GS flow k in one direction and a GS flow l in the opposite direction, which are set up between the master and a particular slave. Furthermore, assume that $\tilde{p}_k \leq \tilde{p}_l$, and that all the relative deadlines are determined according to Section 4.2.1.3. The latter implies that the processor utilization condition (cf. 4.2) is a necessary and sufficient condition.

If the two GS flows use separate logical channels, then GS flow k and GS flow l have a maximum segment size of $s_k^{\max}$ and $s_l^{\max}$ respectively, which are not necessarily equal, and each GS flows is polled independently of the other. Furthermore, each poll results in an empty baseband packet in the opposite direction of the GS flow. However, if we let two oppositely directed flows that involve the same slave share the same logical channel, then a poll for GS flow k implies a poll for GS flow l and vice versa. Furthermore, the resulting maximum segment size will be

$$s_k'^{\max} = s_l'^{\max} = s_k^{\max} + s_l^{\max} - \frac{2}{1600}, \tag{4.49}$$

where the two slots were accounting for the empty baseband packets. Whenever the particular slave is polled, the next poll is planned no earlier than $\tilde{p}_k$ after the planned time of the last poll, i.e., the minimum poll interval is $\tilde{p}_k$. By definition, both flows have the same maximum segment size, i.e., $s_k'^{\max} = s_l'^{\max}$. Knowing that flow l will piggyback on flow k, the admission control has to take into account only the request from flow k with maximum segment size $s_k'^{\max}$ (=$s_l'^{\max}$).

Note that in case of an ideal radio environment, where there are no retransmission that can profit from piggybacking, it is only advantageous to apply piggybacking of GS flows if

$$\frac{s_k^{\max}}{\tilde{p}_k} + \frac{s_l^{\max}}{\tilde{p}_l} > \frac{s_k'^{\max} + s_l'^{\max}}{\min(\tilde{p}_k, \tilde{p}_l)}. \tag{4.50}$$

In that case, piggybacking of GS flow k and GS flow l, which are oppositely directed and involve the same slave, leads to a lower processor utilization.

In other words, if two oppositely directed GS flows exist between the master and a particular slave, and if (4.50) is met, then the real-time task representing the GS flow with the highest value of $\tilde{p}$ should not be included in the feasibility check, and the two GS flows should share the same logical channel.

4.3 Simulation studies

We introduced a poller named Predictive Fair Poller (PFP) in Chapter 3. This poller predicts the availability of data for each slave, and it keeps track of fairness. Based on these two aspects, it decides which slave to poll next. In the BE case, a fair share of resources is determined for each slave, and the fairness is based on the fractions of these fair shares of resources. In the QoS case, this poller applies an EDF scheme to the QoS flows, while planning polls according to the descriptions in Section 4.2.1 and Section 4.2.2. The remaining capacity is used to serve the BE flows according to the BE case.

We evaluate the PFP implementations of the fixed-interval poller and the variable-interval poller by means of simulations in two Guaranteed Service scenarios. In the first scenario, we show the impact of the polling mechanism improvements on the performance of both the guaranteed service flows and the best effort flows. In the second scenario, we compare the PFP implementation of the variable-interval poller with an SCO channel.

The simulation tool we used is Network Simulator (ns2) [ns2] with Bluetooth extensions [Nie00] from Ericsson Switchlab, together with our ns2 implementation of PFP, the fixed-interval poller and the variable-interval poller. The parameters of the PFP with respect to serving the BE flows are the same as in Section 3.5.1.

4.3.1 Scenario I:Comparison between the fixed-interval poller and the variable-interval poller

4.3.1.1 Purpose of the simulation

The purpose of simulating this scenario is to show that the variable-interval poller is able to save more bandwidth than the fixed-interval poller.

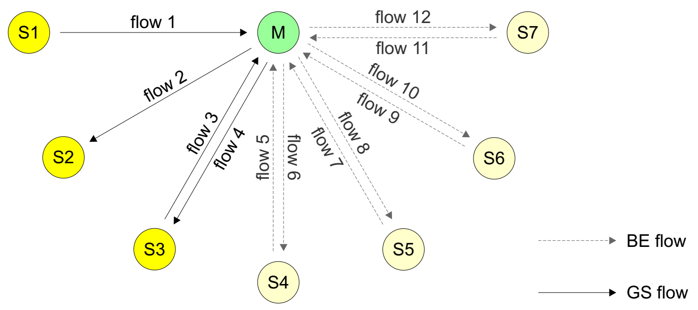


Figure 4.3: *Simulation setup of scenario I*

4.3.1.2 Description of the simulation scenario

The simulations are performed using the setup of Figure 4.3, while making the following assumptions:

- Seven slaves and a master form a piconet, while flows are set up as depicted in the figure.
- Flows 1 to 4 are GS flows, for which the same delay bound is requested, whereas flows 5 to 12 are BE flows (background traffic).
- For the GS flows, the packet sizes are uniformly distributed with a minimum size of 144 bytes, and a maximum size of 176 bytes, i.e., $m_i = 144$ bytes and $M_i = 176$ bytes for each GS flow i. The corresponding average packet size is 160 bytes. In case of a data rate of 64 kbps, this average packet size corresponds to a packet rate of 50 packets/sec, which is a realistic packet rate for audio codecs.
- For the BE flows, the packet sizes are of a fixed size of 176 bytes.
- The time between two consecutive packet generations of the same GS flow equals the size of the first packet divided by a data rate of 8 kbytes/s (64 kbps). The resulting average time interval between two packet generations of the same GS flow is 20 ms.
- The sources of the BE flows generate packets with fixed intervals that depend on the BE load.
- In the first part of scenario I, the delay requirement is set at a fixed value and the sources of the BE flows generate traffic at an equal rate, while simulations are performed at different total BE loads. In the second part of scenario I, the sources of flows 5/6, 7/8, 9/10 and 11/12 generate BE traffic at a data rate of 42.4 kbps, 48 kbps, 53.6 kbps and 59.2 kbps respectively, while simulations are performed at different delay requirements.
- The allowed baseband packet types are DH1 and DH3, with a maximum payload size of 27 bytes and 183 bytes, respectively (including 4 bytes L2CAP header).
- Furthermore, the segmentation policy requires that the DH3 baseband packet is used, unless the remainder of the packet fits in the DH1 baseband packet.

Because of the packet size distribution and the corresponding inter-generation time of packets, the remaining parameters of the token bucket specification are

$$r_i^{\mathrm{p}} = r_i^{\mathrm{t}} = 8 \text{ kbytes/s}, \quad i \in \{1, 2, 3, 4\}, \tag{4.51}$$

and

$$b_i \geq M_i, \quad i \in \{1, 2, 3, 4\}. \tag{4.52}$$

Because of the packet sizes the source of each GS flow i can use, and because of the allowed baseband packet types, the minimum poll efficiency $\epsilon_{\mathrm{p}_i}^{\min}$ is achieved with a packet size of 144 bytes, which is sent using one DH3 baseband packet. Hence, the C error term for these flows is given by $C_i = \epsilon_{\mathrm{p}_i}^{\min} = 144$ bytes for each GS flow i. As all the nodes are allowed to use DH3 baseband packets, the possibility must be taken into account that both the master and the addressed slave transmit a DH3 packet. Consequently, the D error term is given by $D_i = 2\frac{3}{1600} = 3.75$ ms for each GS flow i.

According to Section 4.2, the GS flows 1 to 4 can be looked at as a set of three periodic or sporadic tasks dependent on whether the fixed-interval poller or the variable-interval poller is considered. In both cases, each task i is described by a tuple $(p_i, e_i, d_i) = (\frac{\epsilon_{\mathrm{p}_i}^{\min}}{R_i}, s_i^{\max}, \frac{C_i}{R_i} + D_i)$. All the GS flows are described by equal traffic specifications (token bucket specification), while sharing the same piconet and thus the same maximum possible segment size ($s^{\max}$). As each GS flow is also requesting the same delay bound, the tuple describing each GS flow i can be simplified to $(\frac{144}{R}, \frac{4}{1600}, \frac{144}{R} + \frac{6}{1600})$ for GS flows 1 and 2, and $(\frac{144}{R}, \frac{6}{1600}, \frac{144}{R} + \frac{6}{1600})$ for the pair of GS flows 3 and 4. Considering the feasibility analysis of Section 4.2.1.1, the GS flows can be admitted as long as

$$U = 2\frac{\frac{4}{1600}R}{144} + \frac{\frac{6}{1600}R}{144} \leq 1. \tag{4.53}$$

Consequently, the three GS flows can be admitted as long as $R \leq 16.457$ kbytes/s. This implies that the minimum delay bound that can be requested is $\check{d}^{\mathrm{B}} \approx 23.2$ ms (see (4.1)). On the other hand, the requested fluid model bandwidth R_i of a GS flow i should never be lower than its token rate r_i^{t}. Substituting $R_i = r_i^{\mathrm{t}}$ in (4.1), leads to the delay bound that will never be exceeded, i.e., $\hat{d}_i^{\mathrm{B}} = 43.75$ ms for each GS flow i (see the knee at delay requirement of 43.75 ms in Figure 4.6[1], which is to be discussed below).

4.3.2 Simulation results

As mentioned in Section 4.2.2, the fixed-interval poller plans polls more often than necessary. This has advantageous impact on the mean delay of the GS flows. As can be seen in Figure 4.4, all the GS flows experience the same low mean delay irrespective of the background best effort load. The reason for this is that the fixed-interval poller polls all the GS flows with the same fixed interval. The variable-interval poller polls GS flows only when it assumes that it is needed. For instance, flow 2 is a GS flow that is not involved in piggybacking and that is directed from the master to the slave. This flow will only be polled when GS

[1]The simulation times are chosen such that each (two-sided) 95% confidence interval is less than 2% of its corresponding determined average.

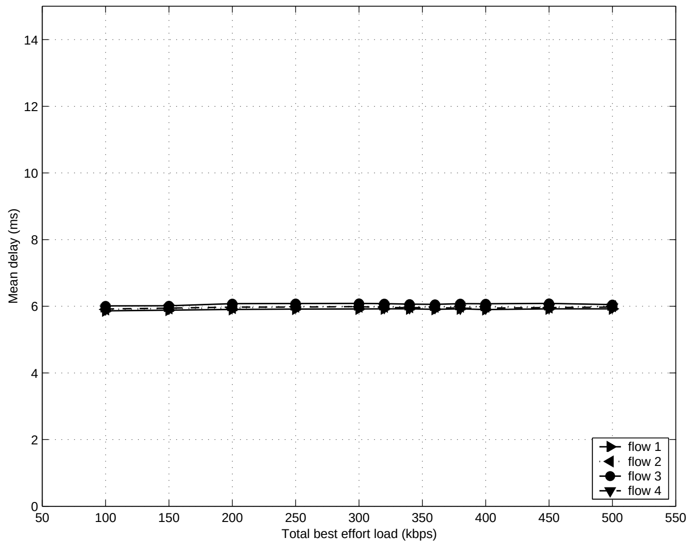


Figure 4.4: *Scenario I: Mean delay as a function of the total best effort load for the fixed-interval poller ($d^{\mathrm{B}} = 23.2$ ms)*

traffic is actually available and when it should be transmitted according to the fluid model. This can be seen in Figure 4.5, where flow 2 experiences a mean delay higher than the ones experienced by the remaining GS flows. Furthermore, these remaining GS flows experience mean delays higher than the one experienced under the fixed-interval poller. The reason for this is that the GS flows are polled less often, depending on the packet sizes they transmit.

Although the mean delay of the GS flows is higher under the variable-interval poller than under the fixed-interval poller, delay bounds are never exceeded. Moreover, the variable-interval poller consumes less resources, saving resources that can be used for retransmissions (in a non-ideal radio environment) or for the transmission of best effort traffic.

Figure 4.6 and Figure 4.7 show the sum of the upstream and downstream throughput of the different slaves as a function of the delay requirement. As could be expected, the GS flows always achieve their maximum throughput as long as they are admitted by the admission control. The throughput of the best effort flows 5 to 12 depends on the requested delay bound of the GS flows as well as on the BE loads (fairness). It can be seen from Figure 4.6 and Figure 4.7 that the best effort flows achieve a higher throughput when served by the variable-interval poller.

4.3.2.1 Conclusions of scenario I

The simulation results showed that a higher BE throughput can be achieved in case a variable-interval poller is used for polling GS flows instead of a fixed-interval poller. Especially with respect to master-to-slave GS flows, this higher BE throughput is achieved at the cost of slightly higher response times for the GS flows.

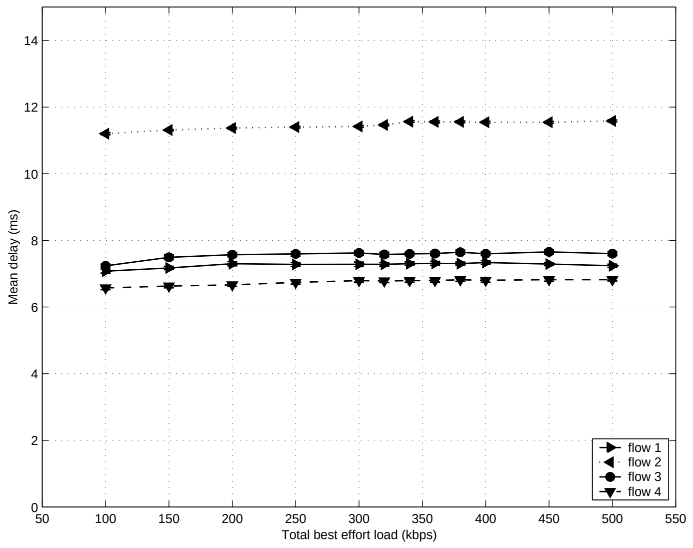


Figure 4.5: *Scenario I: Mean delay as a function of the total best effort load for the variable-interval poller ($d^{\mathrm{B}} = 23.2$ ms)*

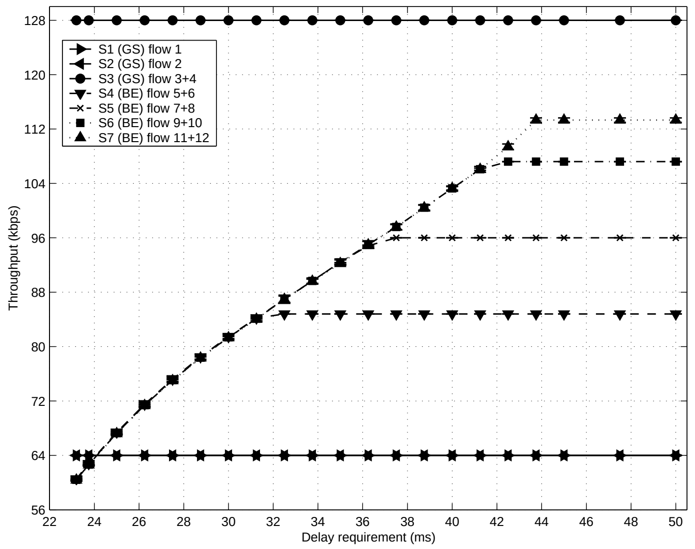


Figure 4.6: *Scenario I: Throughput as a function of the delay requirement for the fixed-interval poller*

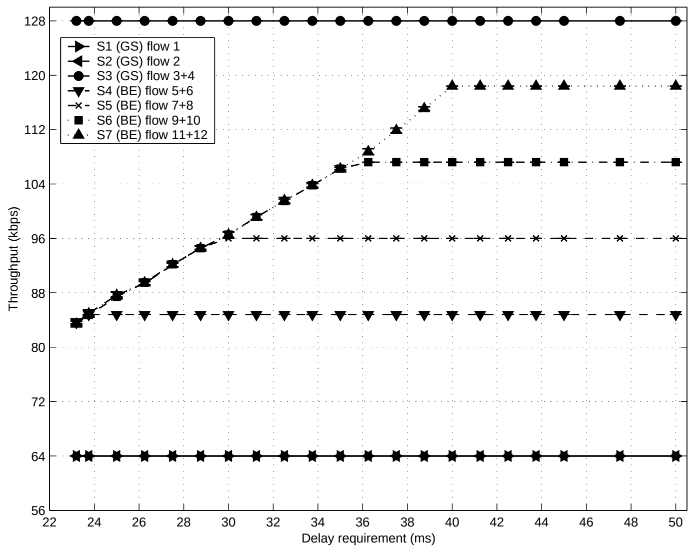


Figure 4.7: *Scenario I: Throughput as a function of the delay requirement for the variable-interval poller*

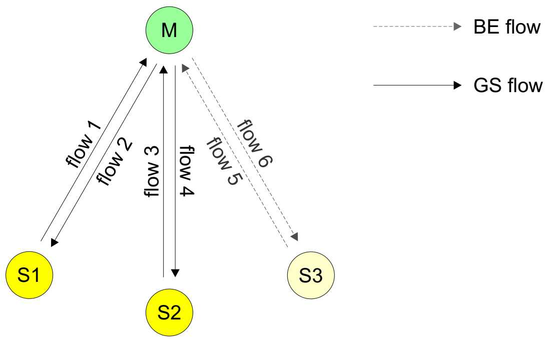


Figure 4.8: *Simulation setup of scenario II*

4.3.3 Scenario II: Comparison between the variable-interval poller and an SCO channel

4.3.3.1 Purpose of the simulation

The purpose of simulating this scenario is to investigate whether the same low total delay requirements can be met by using the variable-interval poller for polling GS flows instead of using an SCO channel. For this purpose, we use a traffic source that SCO channels are very suitable for. Furthermore, this simulation compares the bandwidth saved in case the variable-interval poller is used for serving these traffic flows and in case an SCO channel is used for the same purpose.

4.3.3.2 Description of the simulation scenario

The simulations are performed using the simulation setup of Figure 4.8, while making the following assumptions:

- Three slaves and a master form a piconet
- Flows 1 to 4 are GS flows, which the same delay bound is requested for.
- Flows 5 and 6 are BE flows generating 1 Mbps of background traffic.
- The sources of GS flows 1 to 4 are sample-based codecs that generate samples (1 sample = 1 byte) at a fixed data rate r^{d}.
- The total delay that a packet experiences includes the time needed to collect and packetize samples (packetization delay), the queueing delay, and the transmission delay. Taking into account only packets that fit in a single baseband packet, the larger the packet size, the fewer baseband packets are needed to obtain a certain data rate, but also the higher the packetization delay of a packet. Furthermore, the larger the baseband packet, the higher its transmission delay.

- The sources of the GS flows generate packets of a fixed size L that depends on the total delay requirement. In case of an SCO connection[2] this packet size L is chosen such that the SCO interval, which is the interval between two SCO packets of the same flow, is maximized. In case of an ACL connection, this packet size L is chosen such that the (individual) processor utilization is minimized. Figure 4.9 and Figure 4.10 show the chosen packet size L as a function of the total delay requirement for a data rate of $r^{\mathrm{d}} = 4$ kbytes/s and $r^{\mathrm{d}} = 8$ kbytes/s respectively. For instance, for a data rate of $r^{\mathrm{d}} = 4$ kbytes/s and a total delay requirement of 40 ms, the chosen packet size is 23 bytes for the ACL channel and 30 bytes for the SCO channel.
- The sources of the BE flows generate packets of a fixed size of 179 bytes.
- For the BE slave (S3), the allowed (ACL) baseband packet types are DH1 and DH3, with a maximum payload size of 27 bytes and 183 bytes, respectively (including 4 bytes L2CAP header).
- For the GS slaves (S1 and S2), the allowed ACL baseband packet type is DH1 if the chosen packet size L is not higher than 23 bytes. Otherwise, the DH3 packet type will also be allowed.
- The allowed SCO baseband packet type is HV3 with a maximum payload of 30 bytes.
- The (ACL) segmentation policy requires that the largest allowable baseband packet is used, unless the remainder of the packet fits into a smaller baseband packet.

Note that given a total delay requirement, the GS flows may use a different packet size in the two compared cases (one using an ACL channel with PFP polling, and the other using an SCO channel).

Because of the simulation assumptions, the token bucket specification is given by

$$r_i^{\mathrm{p}} = r_i^{\mathrm{t}} = r_i^{\mathrm{d}}, \quad i \in \{1, 2, 3, 4\}, \tag{4.54}$$

and

$$b_i \geq M_i = m_i = L, \quad i \in \{1, 2, 3, 4\}. \tag{4.55}$$

4.3.3.3 Simulation results

Figure 4.12 and Figure 4.13 show the best effort throughput as a function of the total delay requirement. The PFP line shows the simulated values of the BE throughput that is achieved in case the variable-interval poller is used, whereas the SCO line shows the analytically obtained value of the BE throughput that would have been achieved in case SCO channels would have been used for GS flows 1 to 4.

[2]In the Bluetooth system specification [BT001], each SCO packet type is associated with a fixed payload size and a minimum SCO interval. In this work, we assume that both the payload size and the SCO interval can be chosen without restraint, as long as the associated fixed payload size is not exceeded, and as long as the SCO interval is at least two time slots. Although SCO links are favored by this assumption, it allows for a more fair comparison with ACL links.

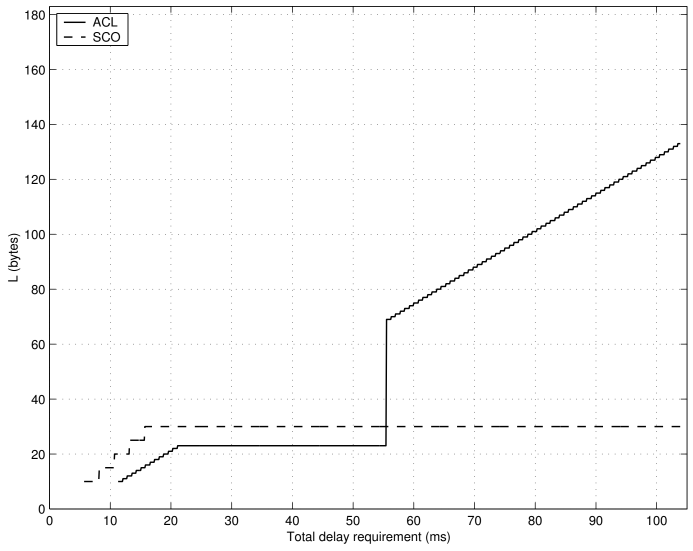


Figure 4.9: *Scenario II: Chosen packet size L as a function of the total delay requirement for $r^{\mathrm{d}} =$ 4 kbytes/s (32 kbps)*

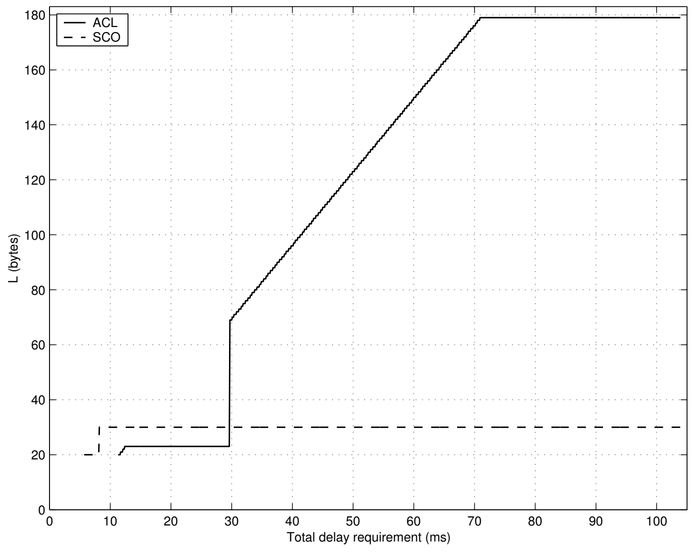


Figure 4.10: *Scenario II: Chosen packet size L as a function of the total delay requirement for $r^{\mathrm{d}} =$ 8 kbytes/s (64 kbps)*

The vertically dashed lines show the minimum total delay that can be guaranteed when PFP is used. As the BE throughput of the PFP at that minimum total delay is zero, that point is not shown in the figures as a simulated mean cannot be obtained.

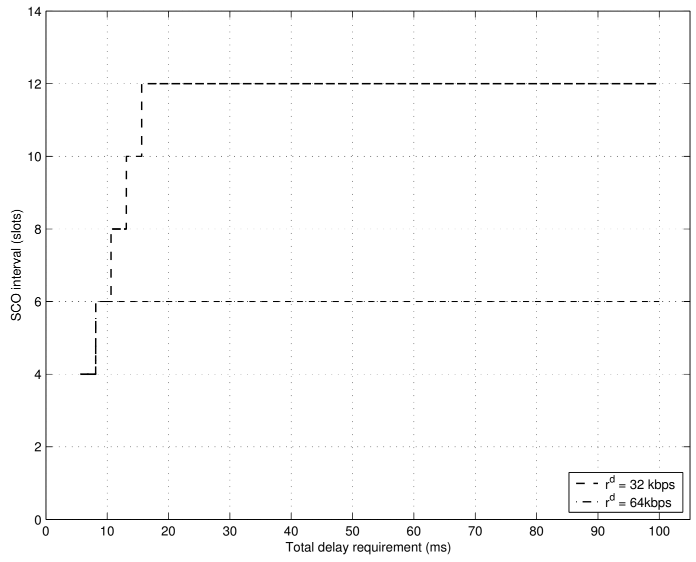


Figure 4.11: *Scenario II: SCO interval as a function of the total delay requirement*

It can be seen that the variable-interval poller can achieve delay bounds that are close to the delay bounds than can be achieved using an SCO channel. Furthermore, it can be seen that the variable-interval poller achieves a higher BE throughput, unless low total delays are required for low data rates. This higher BE throughput is especially prominent for higher GS loads (see Figure 4.13).

With respect to the SCO channel, it can be seen in Figure 4.12 that the BE throughput decreases if the total delay requirement is increased beyond a value of approximately 15 ms. The reason for this is that the packet size is chosen such that the SCO interval is maximized. Increasing the total delay requirement may increase the SCO interval (see Figure 4.11), which means that the number of SCO intervals per seconds decreases. The poller has to take into account that the BE slaves may use the maximum baseband packet size that they are allowed to use (DH3). Consequently, an increase of the SCO interval may not be sufficient for allowing an additional BE transmission within that SCO interval. Hence, the number of BE transmission per second may decrease when the SCO interval is increased.

4.3.3.4 Conclusions of scenario II

The simulation results showed that the variable-interval poller is able to meet total delay requirements that are comparably low to the ones that can be met using an SCO channel. Moreover, the simulation results showed that, for total delay requirements as low as 20 ms, the variable interval poller is able to do so while consuming less resources, which explains the higher total best effort throughput obtained by the variable-interval poller.

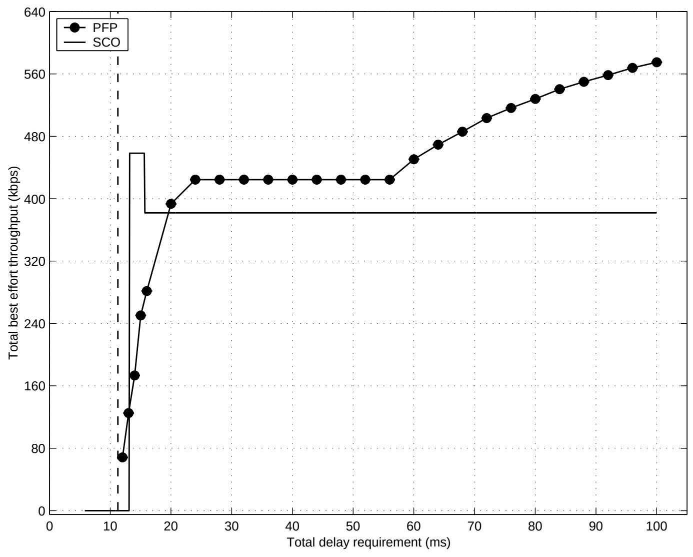


Figure 4.12: *Scenario II: Total best effort throughput as a function of the total delay requirement for $r^{\mathrm{d}} = 4$ kbytes/s (32 kbps) (BE load of 1 Mbps)*

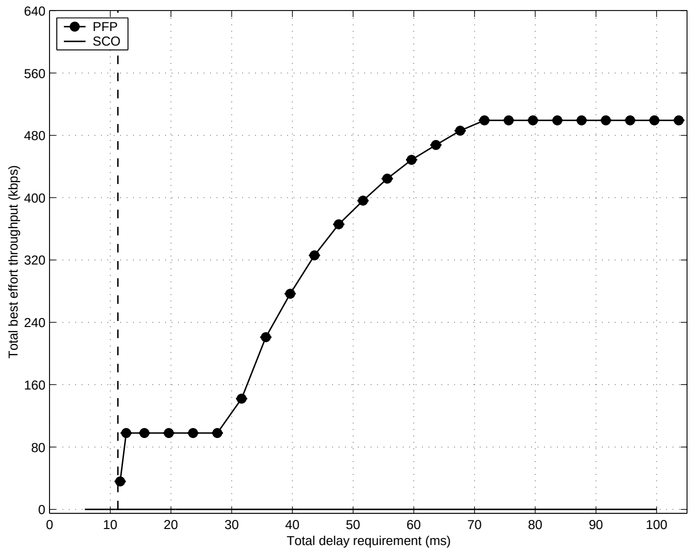


Figure 4.13: *Scenario II: Total best effort throughput as a function of the total delay requirement for $r^{\mathrm{d}} = 8$ kbytes/s (64 kbps) (BE load of 1 Mbps)*

4.4 Discussion

As mentioned in Section 4.1.1, Guaranteed Service guarantees a delay bound by providing a rate guarantee. Consequently, the design of Guaranteed Service support for Bluetooth comprised the design of rate guarantees support for Bluetooth. As a result, this chapter designed the support of two types of QoS for Bluetooth, i.e., guaranteed delays and guaranteed rates.

The Bluetooth polling mechanism determines the delay that packets experience in a piconet. The fixed-interval poller and the variable-interval poller introduced in this chapter divide bandwidth among the slaves such that the delay packets experience is bounded. Moreover, the variable-interval poller polls in such a way that a minimum amount of slots is consumed while polling the GS flows, saving bandwidth that can be used for transmission of BE traffic and/or for retransmission of QoS traffic.

A comparison with an SCO channel showed that the variable-interval poller is able to achieve delay bounds that approach the delay bounds that can be achieved using an SCO channel. For delay requirements as low as 20 ms, the variable-interval poller achieves better BE throughput than when an SCO channel is used for the GS flows. As opposed to an SCO channel, the variable-interval poller can also perform retransmissions.This property can be exploited to avoid the link quality problems of SCO channels in bad radio environments, while keeping up QoS. This will be shown in Chapter 5, when discussing the provisioning of QoS in a non-ideal radio environment.

Chapter 5

QoS in Bluetooth: a non-ideal radio environment

The previous chapter discussed polling mechanisms that are able to provide QoS in an ideal radio environment, where retransmissions are not needed. This chapter discusses the same polling mechanisms while releasing the restriction of the ideal radio environment. This chapter is structured as follows. After the problem description in Section 5.1, Section 5.2 discusses the determination of the so-called flush timeout. Section 5.3 defines methods to perform retransmissions as soon as possible, but without causing scheduling deadlines to be missed. Section 5.4 presents simulation studies of the proposed methods. Finally, Section 5.5 concludes this chapter.

5.1 Problem description

The previous chapter discussed polling mechanisms that are able to provide QoS in an ideal radio environment, where retransmission are not needed. In this chapter, the restriction to such an ideal radio environment is released, i.e., all the flows experience a non-ideal radio environment, where transmission errors may occur. In a non-ideal radio environment, the baseband packets are sometimes erroneously received and consequently discarded. The sender of the discarded baseband packet does not receive an acknowledgment and will retransmit the baseband packet until it receives an acknowledgment or until a flush timeout timer expires. The flush timeout timer is a timer that is used to bound the time which a sender is allowed to (re)transmit a baseband packet in. After expiry of the flush timeout timer, the complete L2CAP packet which the baseband packet belongs to is flushed, and the sender proceeds with the first baseband packet of the next L2CAP packet (if available).

In a non-ideal radio environment, a baseband packet may not be received correctly at the first try. Depending on the time at which retransmission of that baseband packet is performed, and on the flush timeout period, that baseband packet may arrive in time, it may arrive late, or the complete L2CAP which it belongs to may be discarded. In case baseband packets that belong to a GS flow are considered, only L2CAP packets that arrive in time are useful for the receiver of the GS flow, i.e., it makes no difference whether an L2CAP packet is late or discarded. Note that when an L2CAP packet consists of multiple baseband packets, some of the baseband packets arriving late does not necessarily imply that the L2CAP packet is also late.

An obvious way of scheduling retransmissions, is performing retransmissions as soon as no GS polls are pending, i.e., waiting until the EDF scheduler is idle. However, waiting until the EDF scheduler is idle may introduce an additional delay that is unacceptable, and L2CAP packets will more often arrive late or will be discarded depending, among others, on the flush timeout period.

The main problem addressed in this chapter is the scheduling of retransmissions, while meeting deadlines of retransmitted L2CAP packets as much as possible, and while always meeting deadlines of L2CAP packets that need no retransmissions. For this, we first define the largest possible flush timeout that prevents retransmission of L2CAP packets that would cause next L2CAP packets of the same flow to be late. An additional benefit of such a flush timeout is that L2CAP packets that would be late are flushed. Consequently, the deadlines of L2CAP packets that are not flushed are always met. Second, we define methods to identify whether a retransmission can take place before the EDF scheduler is idle, but without causing a poll deadline to be missed. The reason for this is to perform the retransmissions as much a possible within the flush timeout.

5.2 Determining the flush timeout

The flush timeout period, denoted as T_{flush}, is the time period that a particular master or slave is allowed to try to correctly transmit an L2CAP packet in, starting from the time that L2CAP packet arrives at the particular master or slave. On the one hand, a larger timeout value implies that more retransmissions of the same L2CAP packet are allowed to take place. On the other hand, a larger timeout value implies that the next L2CAP can experience a larger initial delay as its service will only start after the preceding L2CAP packet is either correctly transmitted or flushed.

The EDF polling mechanisms introduced in Chapter 4 poll GS flows such that an L2CAP packet j belonging to GS flow i finishes its service at most a time $\frac{L_{i,j}}{R_i} + \tilde{d}_i$ after the moment its service would have started in the fluid model (see also Section 4.2.1.2). This can only be guaranteed if the first poll that can actually be used for servicing L2CAP packet j is planned no later than $\tilde{p}_i$ after its service would have started in the fluid model.

First, let us define an eligible GS L2CAP packet as a GS L2CAP packet whose service would have started in the (theoretical) fluid model at least a time period $\tilde{p}_i$ ago, and which is not (yet) completely served in the actual Bluetooth system. Two conditions must be met in order to guarantee that the first poll that can actually be used for servicing L2CAP packet j is planned no later than $\tilde{p}_i$ after its service would have started in the fluid model. First, a poll must be planned at the right time, which the EDF polling mechanisms of Chapter 4 already take care of. Second, eligible GS L2CAP packets must also be actually available for transmission, which means that a poll for a GS flow that has an eligible GS L2CAP packet should result in a baseband packet belonging to that eligible GS L2CAP packet. In other words, an eligible GS L2CAP packet should not be blocked by a preceding L2CAP packet.

In an ideal radio environment, each baseband packet transmission succeeds, and consequently each GS L2CAP packet is completely served within a time period $\frac{L_{i,j}}{R_i} + \tilde{d}_i$ from the time its service would have started in the fluid model. By definition, the following GS L2CAP packet is not eligible yet, and a GS L2CAP packet will never be blocked by its preceding GS L2CAP packet. In a non-ideal radio environment, baseband packets may be hit by errors and consequently baseband packets may need to be retransmitted. As a result, some GS L2CAP packets will not be successfully served within the time period mentioned above. A direct consequence is that following GS L2CAP packets may be blocked, which makes it impossible to guarantee that GS L2CAP packets will be completely served within a time period $\frac{L_{i,j}}{R_i} + \tilde{d}_i$ from the time its service would have started in the fluid model. The only way to eliminate this problem is to bound the time a master or a slave can try to correctly transmit or retransmit (portions of) a GS L2CAP packet.

In the fluid model, an L2CAP packet j will not be served before L2CAP packet $j-1$ is completely served. Consequently, and according to the definition of an eligible GS L2CAP packet, a packet j of flow i that arrives at time $t_{\mathrm{a}_{i,j}}$ becomes eligible at

$$t_{\mathrm{e}_{i,j}} = \max(t_{\mathrm{a}_{i,j}} + \tilde{p}_i, t_{\mathrm{e}_{i,j-1}} + \frac{L_{i,j-1}}{R_i}), \tag{5.1}$$

where $L_{i,j-1}$ is the size of packet $j-1$ of flow i. In order to guarantee that a GS L2CAP packet is never blocked by (retransmission of) a preceding GS L2CAP packet, the (re)-transmission of packet $j-1$ of flow i should be aborted as soon as packet j of flow i becomes eligible. The flush timeout timer will be used for this purpose.

We assume that a flush timeout can be set for each L2CAP packet and that the flush timeout of a GS L2CAP packet cannot be modified during the lifetime of that GS L2CAP packet. Consequently, the flush timeout of a GS L2CAP packet must be set at the arrival time of corresponding GS L2CAP packet. According to the above, the flush timeout will be given by

$$T_{\mathrm{flush}_{i,j}} = t_{\mathrm{e}_{i,j+1}} - t_{\mathrm{a}_{i,j}}, \tag{5.2}$$

where $t_{\mathrm{a}_{i,j}}$ is the arrival time of packet j of flow i, while $t_{\mathrm{e}_{i,j+1}}$ is the eligibility time of the following GS L2CAP packet. By substituting $t_{\mathrm{e}_{i,j}}$ from (5.1) in (5.2) the flush timeout can be written as

$$T_{\mathrm{flush}_{i,j}} = \max(t_{\mathrm{a}_{i,j+1}} - t_{\mathrm{a}_{i,j}} + \tilde{p}_i, t_{\mathrm{e}_{i,j}} + \frac{L_{i,j}}{R_i} - t_{\mathrm{a}_{i,j}}) \tag{5.3}$$

Unfortunately, the arrival time of the next GS L2CAP packet, i.e., $t_{\mathrm{a}_{i,j+1}}$, is not generally known in advance. As a result the flush timeout must be set to

$$\tilde{T}_{\mathrm{flush}_{i,j}} = t_{\mathrm{e}_{i,j}} + \frac{L_{i,j}}{R_i} - t_{\mathrm{a}_{i,j}} \tag{5.4}$$

By setting the flush time out of an L2CAP packet j of GS flow i to $\tilde{T}_{\mathrm{flush}_{i,j}}$, L2CAP packet $j+1$ of the same GS flow i will never experience an initial delay higher than $\tilde{p}_i$. This is a key requirement for correct functioning of the EDF polling mechanisms introduced in Chapter 4.

5.3 Performing retransmission in slack time

As mentioned in the introduction of this chapter, an obvious way for the polling mechanism to perform retransmission is to wait until there are no pending GS polls, where a pending GS poll is a planned GS poll whose planned time has arrived, but which is not executed yet. In other words, one way of performing retransmissions is to perform retransmissions only if the EDF scheduler is idle. In the literature, this is also referred to as backgrounding.

As the time in which (portions of) a GS L2CAP packet can be retransmitted is bounded by the flush timeout, retransmission of GS baseband packets must be performed as soon as possible. Performing retransmission in idle time may cause the flush timeout to expire. Therefore, the slack of the EDF scheduler will be used for this purpose. The slack of the EDF scheduler is the amount of time that the EDF scheduler can stop polling the GS flows without missing a deadline of a scheduled GS poll.

In this section, the slack determination procedures will be explained. From these procedures, five slack determination/checking policies will be extracted. The first policy is the *offline-determined slack usage* policy, which is concerned with determining the slack during admission control. The second policy is the *online-determined slack usage* policy, which is concerned with determining the slack at the time slack usage is needed. The third one is the *online-checked slack usage* policy, which checks whether a particular amount of slack is available at the time usage of that amount of slack is needed. The fourth one is the *hybrid-determined slack usage* policy, which is a combination of the offline and online slack determination policies. The last one is the *hybrid-checked slack usage* policy, which is a combination of the offline slack determination policy and the online slack checking policy. These five mechanisms will be discussed in Sections 5.3.3-5.3.7. Prior to that, related work is given in Section 5.3.1, and preliminaries are given in Section 5.3.2.

5.3.1 Related work

The problem of performing retransmissions in slack time corresponds to the problem of scheduling a mixed set of hard deadline periodic tasks and aperiodic tasks under the EDF algorithm. Hard deadline periodic tasks are tasks which are activated regularly, and which have a deadline that must be met. Aperiodic tasks are tasks which are activated irregularly, and which have a soft deadline that is desirable to be met, or which have no deadline at all. In other words, aperiodic tasks must be executed as soon as possible.

Using fixed priority methods, this problem has been investigated by Lehoczky et. al. [LSS87], who came up with the Deferrable Server and the Priority Exchange algorithm to enhance aperiodic responsiveness. The Deferrable Server creates a dedicated periodic server task for serving the aperiodic requests. Furthermore, this dedicated periodic server task only consumes execution time if an aperiodic request is actually pending. Allowed execution time that is not consumed is preserved in order to be consumed when an aperiodic request is made. At the beginning of its period, the allowed execution time of the dedicated periodic server task is replenished to its maximum. The Priority Exchange algorithm is similar to the Deferrable Server, except that if no aperiodic request exists, the dedicated periodic server task exchanges its priority with a lower priority periodic task. This exchange is done for the duration of the remaining allowed execution time of the dedicated periodic server task, or until an aperiodic request is made. The allowed execution time of the dedicated periodic server is replenished at the start of its period. Sprunt et. al. [SSL89] came up with another service mechanism which they called the Sporadic Server. The Sporadic Server is similar to the Priority Exchange algorithm, except that the allowed execution time is now replenished in a way that forces the execution time of the dedicated periodic server task to be spread out more evenly [GB95]. Finally, Lehoczky and Ramos-Thuel [LRT92] came up with a service mechanism which they called Slack Stealer. The Slack Stealer is based on the idea of stealing processing time from the periodic tasks, without causing their deadlines to be missed.

Using dynamic priority methods, Ghazalie and Baker proposed dynamic scheduling versions of the Deferrable Server and the Sporadic Server [GB95]. Furthermore, they extended their dynamic version of the Sporadic Server and came up with what they called the Deadline Exchange Server, which can discard remaining allowed execution time of the dedicated periodic server task in exchange for earlier replenishment of the allowed execution time. Spuri and Buttazzo [SB94][SB96] developed five online algorithms for serving soft aperiodic requests in real-time systems, where a set of hard periodic tasks is scheduled using the EDF algorithm. One of these algorithms is the Earliest Deadline Late (EDL) algorithm, which can be considered as a dynamic scheduling version of the Slack Stealing algorithm.

Bosch and Mullender [BM00] presented a scheduler which they called the ΔL-scheduler. It is similar to the EDL scheduler, except that the ΔL-scheduler considers non-preemptively scheduled resources.

Our approach is similar to the ΔL-scheduler, except that we base our slack determination method on the slack determination method used in the ClockWork real-time feasibility analysis tool of Jansen et. al. [JHM02]. The major difference is that we also consider the presence of non-real-time (best effort) tasks. Furthermore, we extended the method to online slack determination in order to be able to exploit the additional idle time generated by our variable-interval poller (see Chapter 4).

5.3.2 Slack determination procedure

Let us define the busy period of an EDF scheduler as a time period in which there is always at least one GS flow for which a poll is pending or being executed. Furthermore, let us define the slack of the EDF scheduler as the amount of time that the EDF scheduler can stop serving the GS flows without causing a deadline to be missed, provided that no transmission errors occur.

Consider a busy period, starting from reference time $t = 0$, of an EDF scheduler that is serving n GS flows. Furthermore, take into account that for each GS flow i, the first poll is planned at $t = t_{\mathrm{f}_i}$, while $\min_i t_{\mathrm{f}_i} = 0$ as the busy period starts at $t = 0$. The processor demand for any time t during this busy period is given by (confer (4.3))

$$\tilde{h}(t) = \sum_{i:t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i\le t} \left(\left\lfloor \frac{t-(\tilde{d}_i-\tilde{p}_i)-t_{\mathrm{f}_i}}{\tilde{p}_i} \right\rfloor \right) s_i^{\max} + s^{\max}. \tag{5.5}$$

As mentioned before, the slack of an EDF scheduler is the amount of time that the EDF scheduler can stop polling the GS flows without missing a deadline of a scheduled GS poll. Hence, the slack S of an EDF scheduler is the minimum distance between the processor demand $\tilde{h}(t)$ and time t starting from the occurrence time t_{h} of the first deadline, i.e., starting from $t_{\mathrm{h}} = \min_i(t_{\mathrm{f}_i} + \tilde{d}_i)$. The slack of the EDF scheduler is

$$S = \lim_{\tau\to\infty} S_{\mathrm{l}}(\tau), \tag{5.6}$$

where $S_{\mathrm{l}}(\tau)$ is the (local) minimum distance between t and $\tilde{h}(t)$ within the interval (t_{h}, τ), which is given by

$$S_{\mathrm{l}}(\tau) = \min_{t_{\mathrm{h}} \leq t \leq \tau} (t - \tilde{h}(t)), \quad \text{for } \tau \geq t_{\mathrm{h}}. \tag{5.7}$$

By substitution of the processor demand $\tilde{h}(t)$ from (5.5) in (5.7), the minimum local distance $S_{\mathrm{l}}(\tau)$ can be written as

$$S_{\mathrm{l}}(\tau) = \min_{t_{\mathrm{h}} \leq t \leq \tau} (t - \sum_{i:t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i \leq t} (\lfloor \frac{t-(\tilde{d}_i-\tilde{p}_i)-t_{\mathrm{f}_i}}{\tilde{p}_i} \rfloor) s_i^{\max} - s^{\max}), \quad \text{for } \tau \geq t_{\mathrm{h}}. \tag{5.8}$$

Determination of the slack using (5.6) is not practical as the time window over which the determination takes place is not bounded. A practical approach would be defining a slack determination time window $(t_{\mathrm{h}}, t_{\mathrm{sd}})$ that is the smallest possible window such that the minimum distance between t and $\tilde{h}(t)$ found in that window can be guaranteed to be the global minimum distance between t and $\tilde{h}(t)$, i.e, such that

$$S_{\mathrm{l}}(t_{\mathrm{sd}}) = \lim_{\tau \to \infty} S_{\mathrm{l}}(\tau) = S. \tag{5.9}$$

Let us first define the distance $\delta_{\tilde{h}^{\mathrm{B}}(t)}$ between the available processor time t and the upper bound on the processor demand $\tilde{h}^{\mathrm{B}}(t)$, i.e.,

$$\delta_{\tilde{h}^{\mathrm{B}}}(t) = t - \tilde{h}^{\mathrm{B}}(t), \tag{5.10}$$

where, for job streams with possibly different start times, the upper bound on the processor demand is defined as

$$\tilde{h}^{\mathrm{B}}(t) = \sum_{i:t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i \leq t} \left(\frac{t-(\tilde{d}_i-\tilde{p}_i)-t_{\mathrm{f}_i}}{\tilde{p}_i} \right) s_i^{\max} + s^{\max}. \tag{5.11}$$

Enforced by the admission control, the processor utilization does not exceed unity, i.e., $\tilde{U} = \sum_i \frac{s_i^{\max}}{\tilde{p}_i} \leq 1$. Consequently, it follows from (5.11) that the upper bound on the processor demand will never increase faster than the available processor time t. As a result, the distance $\delta_{\tilde{h}^{\mathrm{B}}(t)}$ between the available processor time t and the upper bound on the processor demand $\tilde{h}^{\mathrm{B}}(t)$ is a non-decreasing function of t, i.e.,

$$\min_{t \geq \tau} \delta_{\tilde{h}^{\mathrm{B}}}(t) = \delta_{\tilde{h}^{\mathrm{B}}}(\tau). \tag{5.12}$$

By definition, the actual processor demand $\tilde{h}(t)$ is not higher than its upper bound, from which it can be concluded that

$$t - \tilde{h}(t) \geq t - \tilde{h}^{\mathrm{B}}(t) = \delta_{\tilde{h}^{\mathrm{B}}}(t). \tag{5.13}$$

Substitution of $\delta_{\tilde{h}^{\mathrm{B}}}(t)$ from (5.12) in (5.13) gives

$$\min_{t \geq \tau} (t - \tilde{h}(t)) \geq \delta_{\tilde{h}^{\mathrm{B}}}(\tau). \tag{5.14}$$

According to (5.14), $\delta_{\tilde{h}^{\mathrm{B}}}(\tau)$ can be looked at as a lower bound on the distance between t and $\tilde{h}(t)$ for $t \geq \tau$. From (5.14), (5.6) and (5.7) it can be concluded that

$$S = S_{\mathrm{l}}(\tau) \quad \text{if} \quad \delta_{\tilde{h}^{\mathrm{B}}}(\tau) \geq S_{\mathrm{l}}(\tau). \tag{5.15}$$

As $\delta_{\tilde{h}^{\mathrm{B}}}(\tau)$ is a non-decreasing function of τ, the smallest value of τ for which $\delta_{\tilde{h}^{\mathrm{B}}}(\tau) \geq S_{\mathrm{l}}(\tau)$ is $\tau = t_{\mathrm{sd}}$ for which

$$\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{sd}}) = S_{\mathrm{l}}(t_{\mathrm{sd}}) \tag{5.16}$$

Substitution of $\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{sd}})$ from (5.10) in (5.16) gives

$$t_{\mathrm{sd}} - \tilde{h}^{\mathrm{B}}(t_{\mathrm{sd}}) = S_{\mathrm{l}}(t_{\mathrm{sd}}) \tag{5.17}$$

Substitution of $\tilde{h}^{\mathrm{B}}(t_{\mathrm{sd}})$ from (5.11) in (5.17) gives

$$t_{\mathrm{sd}} - \sum_{i:t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i \leq t_{\mathrm{sd}}} \left(\frac{t_{\mathrm{sd}} - (\tilde{d}_i - \tilde{p}_i) - t_{\mathrm{f}_i}}{\tilde{p}_i}\right) s_i^{\max} - s^{\max} = S_{\mathrm{l}}(t_{\mathrm{sd}}) \tag{5.18}$$

As $\delta_{\tilde{h}^{\mathrm{B}}}(t)$ is a non-decreasing function of t, it follows from (5.16) that

$$\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i) \leq S_{\mathrm{l}}(t_{\mathrm{sd}}) \quad \text{for} \quad t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i \leq t_{\mathrm{sd}}. \tag{5.19}$$

Consequently, (5.18) can be rewritten as

$$t_{\mathrm{sd}} - \sum_{i:\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i) \leq S_{\mathrm{l}}(t_{\mathrm{sd}})} \left(\frac{t_{\mathrm{sd}} - (\tilde{d}_i - \tilde{p}_i) - t_{\mathrm{f}_i}}{\tilde{p}_i}\right) s_i^{\max} - s^{\max} = S_{\mathrm{l}}(t_{\mathrm{sd}}) \tag{5.20}$$

Solving t_{sd} from (5.20) gives the right boundary of the slack determination window $(t_{\mathrm{h}}, t_{\mathrm{sd}})$, i.e.,

$$t_{\mathrm{sd}} = \frac{S_{\mathrm{l}}(t_{\mathrm{sd}}) + s^{\max} - \displaystyle\sum_{i:\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i) \leq S_{\mathrm{l}}(t_{\mathrm{sd}})} \frac{t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i}{\tilde{p}_i} s_i^{\max}}{1 - \displaystyle\sum_{i:\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i) \leq S_{\mathrm{l}}(t_{\mathrm{sd}})} \frac{s_i^{\max}}{\tilde{p}_i}} \tag{5.21}$$

It can be seen from the denominator of (5.21) that a solution only exists in one of the following two cases. The first case is if the processor utilization is lower than unity, i.e., $\tilde{U} < 1$ (see also (4.11)). The second case is if the processor utilization equals unity while the job stream with the highest start time starts outside the slack determination window, i.e., $\max_i(t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i) > t_{\mathrm{sd}}$. In that case, the denominator of (5.21) will be larger than zero as the job stream with the highest start time is not included in the summation.

Furthermore, it can be seen from the (5.21) that the exact slack determination window cannot be determined a priori as the local minimum distance $S_{\mathrm{l}}(t_{\mathrm{sd}})$ between the processor demand $\tilde{h}(t)$ and available processor time t within the slack determination window is not known a priori.

However, a priori, the local minimum distance $S_{\mathrm{l}}(t_{\mathrm{h}})$ at the first deadline t_{h} can be determined. From the definition of the local minimum distance (see (5.7)) it can be stated that

$$S_{\mathrm{l}}(t_{\mathrm{h}}) \geq S_{\mathrm{l}}(t_{\mathrm{sd}}), \quad \text{for} \quad t_{\mathrm{h}} \leq t_{\mathrm{sd}}. \tag{5.22}$$

As $\delta_{\tilde{h}^{\mathrm{B}}}(t)$ is a non-decreasing function of t, the smallest t for which $S = S_{\mathrm{l}}(t)$ is a non-decreasing function of $S_{\mathrm{l}}(t)$ (see (5.16)). Consequently, the worst case (largest) slack determination window $(t_{\mathrm{h}}, t_{\mathrm{sd}}^{\mathrm{wc}})$ occurs when $S_{\mathrm{l}}(t_{\mathrm{sd}}) = S_{\mathrm{l}}(t_{\mathrm{h}})$. In that case the slack determination procedure (to be presented later) can be stopped if

$$\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{sd}}^{\mathrm{wc}}) = S_{\mathrm{l}}(t_{\mathrm{h}}) \tag{5.23}$$

Following the same steps as for (5.16), the right boundary of the worst case slack determination window $(t_{\mathrm{h}}, t_{\mathrm{sd}}^{\mathrm{wc}})$ is given by

$$t_{\mathrm{sd}}^{\mathrm{wc}} = \frac{S_{\mathrm{l}}(t_{\mathrm{h}}) + s^{\max} - \displaystyle\sum_{i:\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i) \leq S_{\mathrm{l}}(t_{\mathrm{h}})} \frac{t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i}{\tilde{p}_i} s_i^{\max}}{1 - \displaystyle\sum_{i:\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i) \leq S_{\mathrm{l}}(t_{\mathrm{h}})} \frac{s_i^{\max}}{\tilde{p}_i}} \tag{5.24}$$

Again, a solution for $t_{\mathrm{sd}}^{\mathrm{wc}}$ only exists if either the processor utilization is below unity, i.e., $\tilde{U} < 1$, or if the processor utilization equals unity while the job stream with the highest start time starts outside the worst case slack determination window, i.e., $\max\limits_i (t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i) > t_{\mathrm{sd}}^{\mathrm{wc}}$.

From the above, the slack determination procedure of Figure 5.1 can be extracted. The depicted procedure takes as input a tuple $(\tilde{p}_i, \tilde{d}_i, s_i^{\max}, t_{\mathrm{f}_i})$ for each GS flow i. The output is the amount of slack available, i.e., S. In step 1, the number n_i of deadlines processed for each GS flow i is updated based on the last processed time t. Based on these numbers, the next time t to be processed is determined in step 2. In step 3, the determined slack is set to the minimum of its current value and $t - \tilde{h}(t)$ (see also (5.7)). In step 4, if $S_{\mathrm{l}} > \delta_{\tilde{h}^{\mathrm{B}}}(t)$, the algorithm returns to step 1. Otherwise, the actually available amount of slack S is set to S_{l} and the algorithm stops (see also (5.15)). Given a total number n of GS flows, the procedure needs an initial amount of at most $6n + 2$ operations (summation, multiplication, and their inverse), and at most $14n + 4$ operations for each processed time t.

5.3.3 Offline-determined slack usage policy

The slack of an EDF scheduler varies with time. However, a priori, a minimum amount of slack $S^{\min}$ can be calculated that can be consumed once during each busy period. This minimum amount of slack does not have to be consumed at once, which means that multiple retransmission can be performed during a single busy period as long as the retransmissions fit within the determined slack $S^{\min}$.

- *input*:
  - $(\tilde{p}_i, \tilde{d}_i, s_i^{\max}, t_{\mathrm{f}_i})$ for each GS flow i, where $\tilde{p}_i$ is the poll period, $\tilde{d}_i$ is the relative deadline, $s_i^{\max}$ is the maximum segment size, and where t_{f_i} is the planned time of the first poll for GS flow i within the busy period in consideration.
- *output*:
  - Available slack S.

a. Let t be the reference time, which is initially set to

$$t = t_{\mathrm{h}} = \min_i(\tilde{d}_i + t_{\mathrm{f}_i}). \tag{5.25}$$

b. Let S_{l} be the determined slack, which is initially set to

$$S_{\mathrm{l}} = t_{\mathrm{h}} - \tilde{h}(t_{\mathrm{h}}), \tag{5.26}$$

where the processor demand $\tilde{h}(t)$ is defined as in (5.5).

c. Let n_i be the number of deadlines processed for GS flow i, where initially $\forall i\ n_i = 0$.

1. $\forall i$, if $t \geq (n_i + 1)\tilde{d}_i + t_{\mathrm{f}_i}$ then $n_i = n_i + 1$.
2. Set the reference time at $t = \min_i((n_i + 1)\tilde{d}_i + t_{\mathrm{f}_i})$.
3. Set the determined slack to
$$S_{\mathrm{l}} = \min(S_{\mathrm{l}}, t - \tilde{h}(t)). \tag{5.27}$$
4. With $\delta_{\tilde{h}^{\mathrm{B}}}(t)$ as defined in (5.10), if $S_{\mathrm{l}} > \delta_{\tilde{h}^{\mathrm{B}}}(t)$ then goto step 1.
5. $S = S_{\mathrm{l}}$.
6. End.

Figure 5.1: *Slack determination procedure*

The offline-determined slack usage policy is concerned with determining the minimum amount of slack that can be used once during each busy period. For this, a slack tank with a maximum size equal to the offline-determined slack, and with a current amount S_{T} of available slack is being used. If a retransmission should be performed for a particular GS flow i, then that retransmission is only performed if the maximum segment size $s_i^{\max}$ of GS flow i is not higher than the current amount of available slack S_{T}. Furthermore, the amount of available slack is each time decreased by the amount of time the EDF scheduler is not serving a new segment of a GS flow for the first time (i.e., also decreased after a retransmission). As soon as the EDF scheduler becomes idle, the slack tank is replenished, i.e., $S_{\mathrm{T}} = S^{\min}$.

In order to determine the minimum slack $S^{\min}$, the worst case busy period must be considered. The worst case busy period is the busy period with the lowest local minimum distance between processor time t and processor demand $\tilde{h}(t)$. This implies that the worst case busy period is the period with the highest processor demand $\tilde{h}(t)$ for given t. From (5.5), it can be seen that the worst case busy period is the case in which, for each GS flow i, the first poll within that busy period is planned for reference time $t = 0$, i.e., $t_{\mathrm{f}_i} = 0$ for each GS flow i. In that case reference time $t = 0$ becomes a critical instant, which is defined in [LL73] as a time for which instances of all jobs are simultaneously scheduled.

The maximum size of the slack tank $S^{\min}$ is determined at admission control time using the procedure of Figure 5.1, while inserting, for each GS flow i, a tuple $(\tilde{p}_i, \tilde{d}_i, s_i^{\max}, t_{\mathrm{f}_i})$ with $t_{\mathrm{f}_i} = 0$. The returned amount of slack is the maximum size of the slack tank $S^{\min}$.

Example

Consider a set of four GS flows, of which flow 1 to 4 are described (in slots) by the tuples $(8, 12, 2)$, $(10, 14, 2)$, $(14, 18, 4)$ and $(16, 20, 2)$ respectively, while the tuple of a GS flow i is formatted as $(\tilde{p}_i, \tilde{d}_i, s_i^{\max})$. Furthermore, the maximum segment size is 4 slots, i.e., $s^{\max} = 4$. From the GS flow descriptions, it can be seen that $\tilde{d}_i = \tilde{p}_i + s^{\max}$ for each GS flow i.

It follows from the GS flow descriptions that the processor utilization is given by

$$\tilde{U} = \sum_{i=1}^{4} \frac{s_i^{\max}}{\tilde{p}_i} \quad (5.28)$$

$$= \frac{2}{8} + \frac{2}{10} + \frac{4}{14} + \frac{2}{16} \approx 0.86, \quad (5.29)$$

which is less than unity. As the relative deadline of each GS flow i is given by $\tilde{d}_i = \tilde{p}_i + s^{\max}$, the processor utilization condition of (4.2) is a necessary and sufficient condition and the set of GS flows is schedulable.

This example considers the case in which retransmissions are to be performed in offline-determined slack time, which means that a slack tank is to be used, whose maximum size is given by the offline-determined slack $S^{\min}$. The offline-determined slack is computed at admission control time using the procedure given in Figure 5.1. Using Figure 5.2, Figure 5.3, and Figure 5.4, it will be explained how the available slack is determined.

Figure 5.2 shows the processor demand $\tilde{h}(t)$, the processor time t and the workload $\tilde{w}(t)$, with t expressed in slots. The workload $\tilde{w}(t)$ is the amount of transmission time requested in the period $[0, t)$ by the GS flows in addition to a single maximum segment size $s^{\max}$, i.e.,

$$\tilde{w}(t) = \sum_{i: t_{\mathrm{f}_i} \le t} \left(\left\lceil \frac{t - t_{\mathrm{f}_i}}{\tilde{p}_i} \right\rceil \right) s_i^{\max} + s^{\max}. \quad (5.30)$$

As soon as the processor time t becomes equal to the workload $\tilde{w}(t)$, the EDF scheduler is said to be idle as all the polls that are planned before t are executed. If at the same time new polls are planned, then that idle period is an infinitely small idle period. With respect to the case in which retransmissions are performed in idle time, the first time at which retransmissions can be performed is implementation specific. By example, if newly planned polls always have priority above retransmissions than, in the worst case, retransmissions cannot be performed before $t = 54$. On the other hand, a different implementation can make it possible

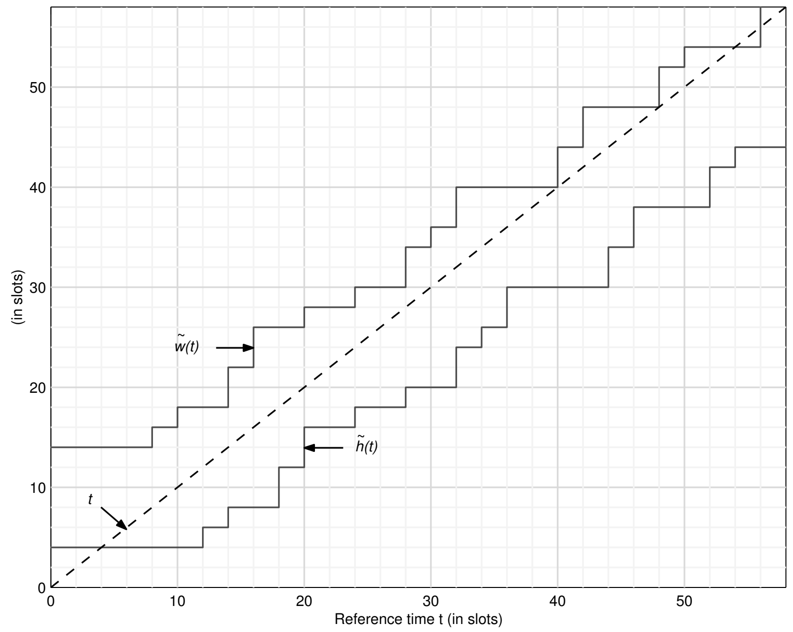


Figure 5.2: *Workload and processor demand as a function of time within the worst case busy period.*

to perform a retransmission at $t = 40$ or $t = 48$. This retransmission will not jeopardize the GS flows as the EDF scheduler already takes into account a maximum size non-GS transmission for each busy period (see Section 4.2.1.1 in Chapter 4). The period $(0, 40)$ is larger than each of the four relative deadlines, which implies that waiting until an idle period in order to perform a retransmission may cause deadlines to be missed.

Figure 5.3 shows the processor demand $\tilde{h}(t)$, the processor time t, and the minimum distance $S_l(t)$ between the processor demand and the processor time in the period (t_h, t), where $t_h = 12$ is the occurrence time of the first deadline. As can be seen in the figure, the local minimum distance $S_l(t)$ does not change outside the slack determination window.

As mentioned before, the right boundary of the slack determination window is the time at which the distance $\delta_{\tilde{h}^{\mathrm{B}}}(t)$ between the processor time t and the upper bound on the processor demand $\tilde{h}^B(t)$ equals the local minimum distance $S_l(t)$. From Figure 5.4, it can be seen that this is the case for $t \approx 32.72$. Moreover, it can be seen that $t - S^{\min} > \tilde{h}^B(t)$ for $t > 32.72$ and thus that $t - S^{\min} > \tilde{h}(t)$, which is the basic idea behind the slack determination window.

Summarizing, eight time instances have to be evaluated for determining $S^{\min}$. The determined available amount of slack is $S^{\min} = 4$, so, depending on the flow, retransmission of one or two packets can be performed at any time during a busy period.

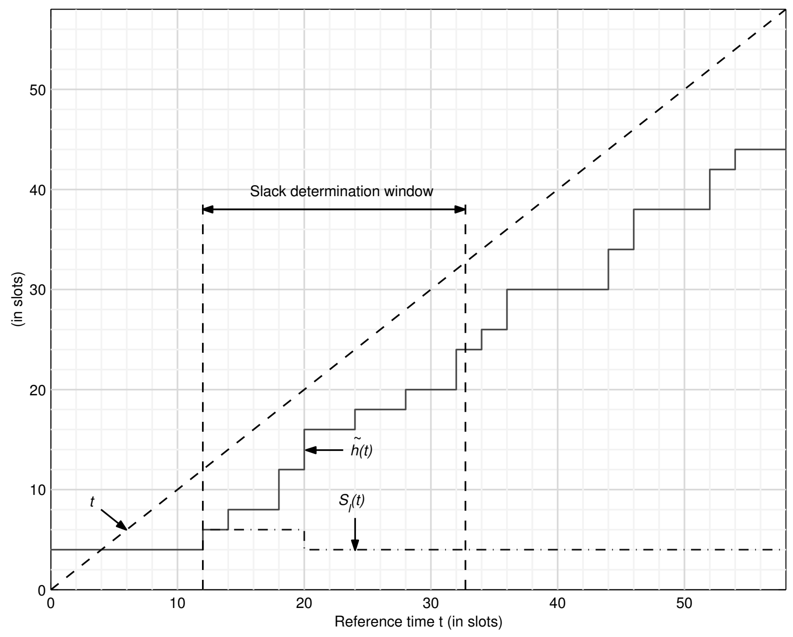


Figure 5.3: *Local minimum distance $S_1(t)$ as a function of time.*

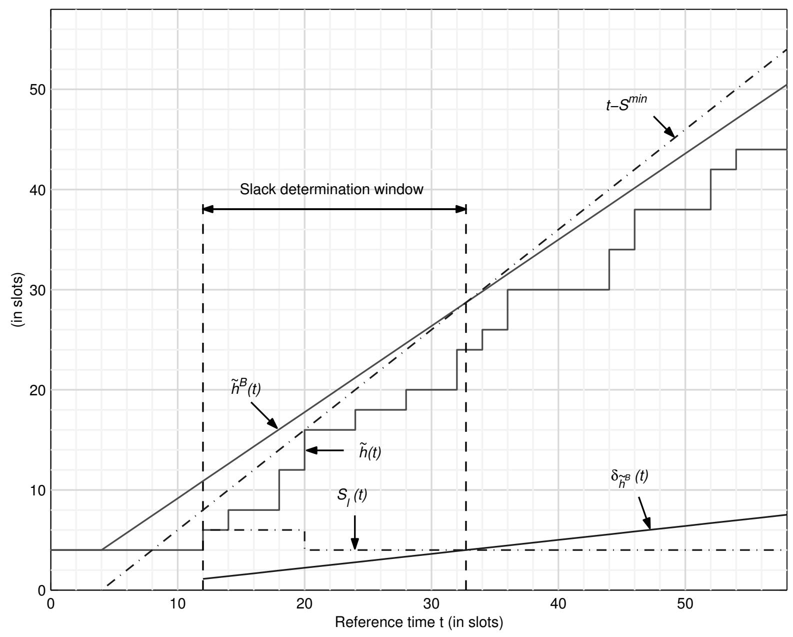


Figure 5.4: *Slack determination window for the worst case busy period.*

5.3.4 Online-determined slack usage policy

The offline-determined slack usage policy described in the previous section is concerned with determining the slack S^{min} that would be available in a worst case busy period. Furthermore, each busy period is treated as a worst case busy period, which means that at most an amount S^{min} of slack can be consumed during each busy period, regardless of the actually available slack.

Fortunately, not every busy period is a worst case busy period, and consequently the actually available slack may be higher than the offline-determined slack. There are several reasons for the majority of busy periods not being a worst case busy period. First, the poll periods of the different GS flows are not necessarily equal, nor are they necessarily synchronized. Hence, the start time of a busy period is not necessarily a critical instant. Second, if the improved poller of Chapter 4 is to be used, then, with respect to a GS flow i, the time between the currently planned poll and the next planned poll is not necessarily equal to the minimum poll period $\tilde{p}_i$. At a particular reference time $t = 0$, these effects translate into different release times of the GS flows with respect to the particular reference time. Note that a GS flow i being released at time t refers to the first poll for GS flow i being planned for time t.

Consider a busy period that starts at reference time $t = 0$, which means that at least one of the GS flows is released at $t = 0$. Clearly, if some of the remaining GS flows is released later than $t = 0$, then, for the same value of t, the processor demand $\tilde{h}(t)$ may be lower than in the case in which $t = 0$ is a critical instant. Consequently, the actually available slack may be higher than S^{min}. Moreover, it will be shown by an example that if at $t = 0$ an amount of slack has already been consumed, it is possible in some cases that the actually available slack at reference time $t = 0$ is still higher than S^{min}.

The online-determined slack usage policy is concerned with determining the actually available slack at the moment slack is needed to be used for retransmission of a GS baseband packet. In order to determine the currently available slack, the procedure depicted in Figure 5.1 can be used. For each GS flow i, a tuple $(\tilde{p}_i, \tilde{d}_i, s_i^{\mathrm{max}}, t_{\mathrm{f}_i})$ is inserted, while t_{f_i} is the release time of flow i relative to reference time $t = 0$, which corresponds to the current time. The value returned by this procedure is the currently available amount of slack S. Note that a negative release time of a GS flow i means that GS flow i has already been released but not yet served, which implies that the busy period started before reference time $t = 0$.

Example

Consider, at a particular time t_{a} (in slots), the same set of GS flows as described in the example of Section 5.3.3, while the GS flows 1 to 4 are released at $t = t_{\mathrm{a}} + 2$, $t = t_{\mathrm{a}} - 2$, $t = t_{\mathrm{a}}$, and $t = t_{\mathrm{a}} + 4$ respectively, with t expressed in slots.

As can be seen from the release times, GS flow 2 is released two slots ago without being served yet. This can be caused by different reasons. For instance, the scheduler could be busy serving a non-GS segment that started before $t = t_{\mathrm{a}} - 2$. An other possible reason is that the EDF scheduler was (or started) serving another GS flow at $t = t_{\mathrm{a}} - 2$. Finally, it may be possible that the server was (or started) performing a retransmission of a GS baseband packet at $t = t_{\mathrm{a}} - 2$. If retransmissions where to be performed in offline-determined slack,

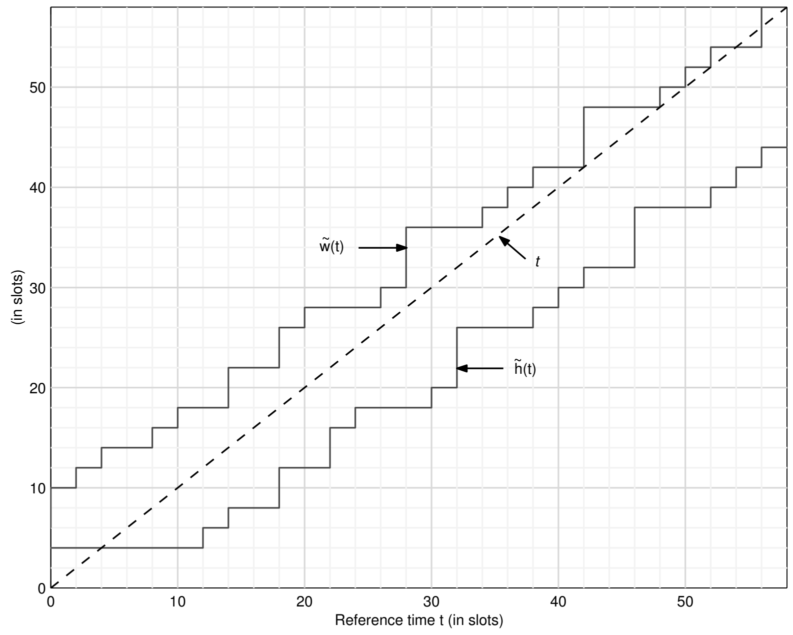


Figure 5.5: *Workload and processor demand as a function of time.*

then the latter reason would have caused the available slack to be decreased with an amount of at least two slots, meaning that the available offline-determined slack would have been at most $S^{\mathrm{min}} - 2 = 2$ at $t = t_{\mathrm{a}}$.

Let us consider the case in which a retransmission of a segment a GS flow has taken place at $t = t_{\mathrm{a}} - 2$, while the size of the retransmitted segment is two slots. Clearly, the offline-determined slack would be at most 2 slots at $t = t_{\mathrm{a}}$. Now assume that retransmission of another GS segment with a maximum size of 4 slots should be performed at $t = t_{\mathrm{a}}$. In case of retransmission if offline-determined slack, this retransmission cannot take place as the remaining offline-determined slack is not sufficient.

As mentioned before, the retransmissions in offline-determined slack policy treats each busy period as a worst case busy period, and consequently assumes, for each busy period, the availability of the slack that would be available in a worst case busy period. The busy period described above is, like much of the busy periods, not a worst case busy period as it does not contain a critical instant. It is expected that the online-determined slack at t_{a} will be higher than the remaining slack in case of the retransmission in offline-determined slack policy.

Figure 5.5 shows the processor demand $\tilde{h}(t)$, the processor time t and the workload $\tilde{w}(t)$. In the worst case, for each GS flow i, a poll will be planned each poll period $\tilde{p}_i$. In that case, the EDF scheduler will not become idle before reference time $t = 42$. In case of the retransmission in idle time policy, the retransmission mentioned above will thus not be performed before $t = 42$. As the period $(0, 42)$ is larger than the maximum relative deadline $\max_i \tilde{d}_i$, this means that the deadline of the segment to be retransmitted will be missed.

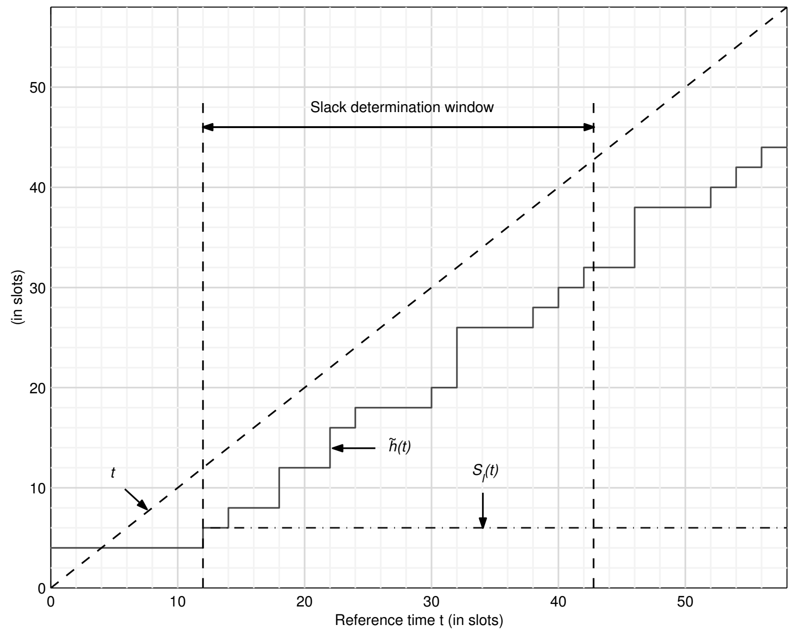


Figure 5.6: *Local minimum distance $S_\mathrm{l}(t)$ as a function of time.*

Figure 5.6 shows the processor demand $\tilde{h}(t)$, the processor time t, and the minimum distance $S_\mathrm{l}(t)$ between the processor demand and the processor time in the period (t_h, t), where $t_h = 12$ is the occurrence time of the first deadline. As can be seen in the figure, the local minimum distance $S_\mathrm{l}(t)$ does not change outside the slack determination window. Furthermore, it can be seen that the actually available slack is $S = 6$, which means that the available slack is enough to perform a retransmission at $t = 0$.

As mentioned before, the right boundary of the slack determination window is the time at which the distance $\delta_{\tilde{h}^\mathrm{B}}(t)$ between the processor time t and the upper bound on the processor demand $\tilde{h}^B(t)$ equals the local minimum distance $S_\mathrm{l}(t)$. From Figure 5.7, it can be seen that this is the case for $t \approx 42.77$. Moreover, it can be seen that $t - S > \tilde{h}^B(t)$ for $t > 42.77$ and thus that $t - S > \tilde{h}(t)$, which is the basic idea behind the slack determination window. Note that the processor demand $\tilde{h}(t)$ must be compared with processor time t for eleven values of reference time t in case the total available amount of slack is to be determined.

We propose three variants of the online-determined slack usage policy that make it possible to perform the same retransmissions that could be performed in the online-determined slack usage policy, but which differ in the average number of steps that is needed to be checked. These variants of the online-determined slack usage policy are the *online-checked slack usage policy*, the *hybrid-determined slack usage policy*, and the *hybrid-checked slack usage policy*.

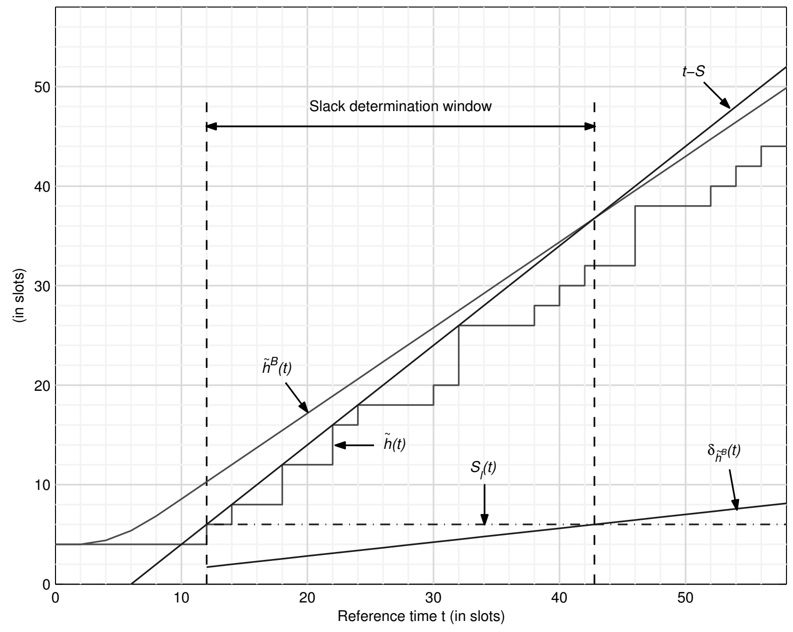


Figure 5.7: *Slack determination window.*

5.3.5 Online-checked slack usage policy

The online-checked slack usage policy is concerned with determining whether at a particular time a particular retransmission will fit into the actually available amount slack without exactly knowing the actually available amount of slack. The advantage of the online-checked slack usage policy is that the number of time instances to be processed is often less than, and at most equal to the number of time instances to be processed in case the total amount of available slack is to be determined. The latter is the case if the required amount of slack equals the actually available amount of slack.

Determining whether a particular retransmission fits in the currently available slack is done using the procedure depicted in Figure 5.8. Besides a tuple $(\tilde{p}_i, \tilde{d}_i, s_i^{\max}, t_{f_i})$ for each GS flow i, a requested amount of slack S_r is also inserted. The procedure returns a boolean A_s, which is true if the particular retransmission fits into the currently available slack and false otherwise.

In step (a), the time t to be processed is set equal to the first deadline. Step (b) initializes the determined slack S by setting it to $S = S_l(t_h) = t_h - \tilde{h}(t_h)$ (see also (5.7)). Step (c) initializes the number of processed deadlines for each GS flow by setting it to zero, i.e., $\forall i$ $n_i = 0$. The slack availability variable A_s is initially set to true in step (d). Step 1 checks whether the amount of slack S found so far is less than the requested amount of slack S_r. If this is the case, the slack availability variable is set to false, i.e., $A_s =$ false, and the algorithm stops. In step 2, the number n_i of deadlines processed for each GS flow i is updated based on the last processed time t. Based on these numbers, the next time t to be processed is deter-

- *input*:
  - $(\tilde{p}_i, \tilde{d}_i, s_i^{\max}, t_{\mathrm{f}_i})$ for each GS flow i, where $\tilde{p}_i$ is the poll period, $\tilde{d}_i$ is the relative deadline, $s_i^{\max}$ is the maximum segment size, and where t_{f_i} is the planned time of the first poll for GS flow i within the busy period in consideration.
  - S_r requested amount of slack.
- *output*:
  - Slack availability A_s, which is true if S_r is available, and which is false otherwise.

a. Let t be the reference time, which is initially set to

$$t = t_\mathrm{h} = \min_i(\tilde{d}_i + t_{\mathrm{f}_i}). \tag{5.31}$$

b. Let S_l be the determined slack, which is initially set to

$$S_\mathrm{l} = t_\mathrm{h} - \tilde{h}(t_\mathrm{h}), \tag{5.32}$$

where the processor demand $\tilde{h}(t)$ is defined as in (5.5).

c. Let n_i be the number of deadlines processed for GS flow i, where initially $\forall i\ n_i = 0$.

d. Let A_s be a boolean variable indicating the availability of slack S_r. Initially $A_\mathrm{s} = \mathrm{true}$.

1. if $S_\mathrm{l} < S_\mathrm{r}$ then $A_\mathrm{s} = \mathrm{false}$ and goto step 6.
2. $\forall i$, if $t \geq (n_i + 1)\tilde{d}_i + t_{\mathrm{f}_i}$ then $n_i = n_i + 1$.
3. Set the reference time at $t = \min((n_i + 1)\tilde{d}_i + t_{\mathrm{f}_i})$.
4. Set the determined slack to
$$S_\mathrm{l} = \min(S_\mathrm{l}, t - \tilde{h}(t)). \tag{5.33}$$
5. With $\delta_{\tilde{h}^\mathrm{B}}(t)$ as defined in (5.10), if $S_\mathrm{r} > \delta_{\tilde{h}^\mathrm{B}}(t)$ then goto step 1.
6. End.

Figure 5.8: *Slack checking procedure*

mined in step 3. In step 4, the determined slack S_l is set to the minimum of it current value and $t - \tilde{h}(t)$ (see also (5.7)). If $S_\mathrm{r} > \delta_{\tilde{h}^\mathrm{B}}(t)$, the algorithm returns to step 1. Otherwise, the algorithm stops while denoting that the requested amount of slack is actually available (i.e., $A_\mathrm{s} = \mathrm{true}$). The procedure needs an initial amount of at most $6n + 2$ operations (summation, multiplication, and their inverse), and at most $14n + 4$ operations for each processed time t.

Example

Consider the same example as in section 5.3.4. Figure 5.9 shows the slack checking window in case it is to be checked whether an amount of slack $S_\mathrm{r} = 4$ is available or not. As soon as $\delta_{\tilde{h}^\mathrm{B}}(t)$ becomes higher than S_r, it can be seen that $t - S_r > \tilde{h}^\mathrm{B}(t)$ and thus that $t - S_r > \tilde{h}(t)$.

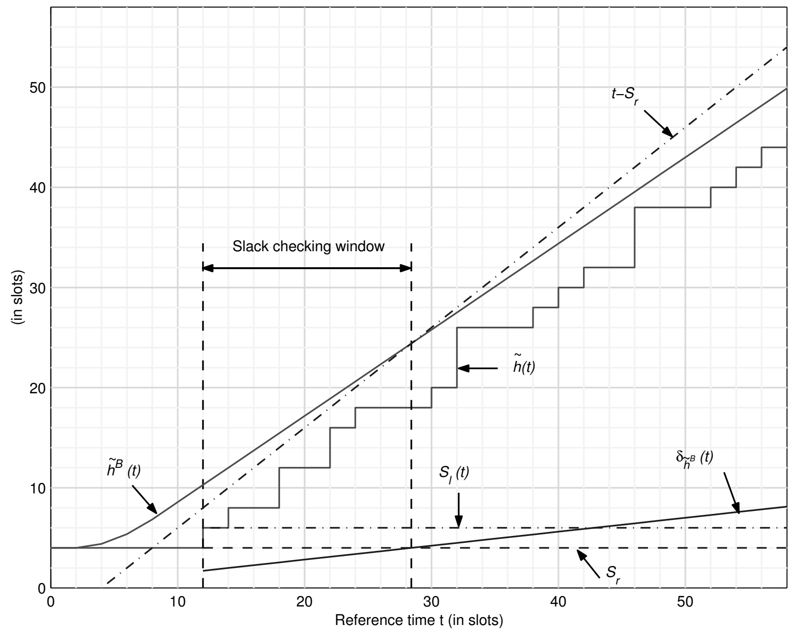


Figure 5.9: *Slack checking window.*

Consequently, the right boundary of the slack checking window is given by the time $t = t_{\mathrm{scw}}$ at which $\delta_{\tilde{h}^{\mathrm{B}}}(t) = S_{\mathrm{r}}$, i.e.,

$$t_{\mathrm{scw}} = \frac{S_{\mathrm{r}} + s^{\max} - \displaystyle\sum_{i:\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i)\leq S_{\mathrm{r}}} \frac{t_{\mathrm{f}_i} + \tilde{d}_i - \tilde{p}_i}{\tilde{p}_i} s_i^{\max}}{1 - \displaystyle\sum_{i:\delta_{\tilde{h}^{\mathrm{B}}}(t_{\mathrm{f}_i}+\tilde{d}_i-\tilde{p}_i)\leq S_{\mathrm{r}}} \frac{s_i^{\max}}{\tilde{p}_i}} \tag{5.34}$$

$$\approx 28.41. \tag{5.35}$$

Note that if it is to be checked whether a retransmission of a GS segment with a maximum size of four slots fits within the available slack, then the processor demand $\tilde{h}(t)$ must be compared with processor time t for only six values of reference time t instead of eleven for the online-determined slack usage policy.

5.3.6 Hybrid-checked slack usage policy

As explained in the previous sections, at least a minimum amount $S^{\min}$ of slack is available at the beginning of each busy period. This amount decreases during the busy period for each retransmission of a GS segment. In terms of calculations, the online-checked slack usage policy can be improved by adding the main idea behind the offline-determined slack usage policy in the following way.

The slack checking procedure of Figure 5.8 can be improved by maintaining a slack tank, which denotes the least available amount of slack during a busy period. During admission control, the minimum amount $S^{\min}$ that is available during each busy period is determined. Furthermore, the hybrid-checked slack usage policy behaves like the offline-determined slack usage policy in the sense that a retransmission can always be performed if the slack tank denotes the availability of the required amount of slack, while the slack tank is replenished each time the EDF scheduler becomes idle. If the amount of slack denoted by the slack tank is insufficient for performing a desired retransmission in, then the hybrid-checked slack usage policy behaves like the online-checked slack usage policy, i.e., it is checked whether the desired retransmission can be performed in the actually available slack.

5.3.7 Hybrid-determined slack usage policy

The hybrid-determined slack usage policy is a combination of the offline-determined slack usage policy and the online-determined slack usage policy. As soon as a retransmission of a particular GS segment is to be performed, it is checked whether that particular GS segment fits into the amount of slack denoted by the slack tank. In case the denoted amount of slack is insufficient for performing the desired retransmission in, the actually available slack is determined and the retransmission takes place if the actually available slack is sufficient for performing the desired retransmission in. Subsequently, the value of the slack tank is adjusted taking into account the determined actually available slack and the performed retransmission if any.

As soon as the EDF scheduler becomes idle again, the slack tank is set to the offline-determined amount of slack $S^{\min}$ if that amount is larger than the amount of slack currently denoted by the slack tank. Furthermore, the slack tank is adjusted (decreased) proportional to the processor time t, unless for new transmission of GS segments. However, during an idle period, the slack tank is never decreased below the offline-determined amount of slack $S^{\min}$.

An overview of the different slack usage policies is given in Table 5.1.

Policy	Description
offline-determined slack usage	the minimum amount of slack per busy period is determined during admission control; this amount of slack can be consumed during each busy period.
online-determined slack usage	whenever an amount of slack is required, the actually available amount of slack is determined
online-checked slack usage	whenever an amount of slack is required, it is checked whether the required amount of slack is not higher than the actually available amount of slack
hybrid-determined slack usage	combination of the offline-determined slack usage policy and the online-determined slack usage policy
hybrid-checked slack usage	combination of the offline-determined slack usage policy and the online-checked slack usage policy

Table 5.1: *Overview of the different slack usage policies.*

5.4 Simulation studies

This section presents simulation studies of the variable-interval poller in a non-ideal radio environment. The first simulation scenario compares the variable-interval poller with the fixed-interval poller with respect to the *residual packet drop ratio* in case retransmissions are performed in idle time. The residual packet drop ratio is the ratio of the number of GS L2CAP packets that are not received by the corresponding receivers to the total number of generated GS L2CAP packets. The second simulation scenario presents, in case of the variable-interval poller, a comparison between different slack usage policies with respect to the residual packet drop ratio and with respect to the number of checked time instances per retransmission attempt. The third simulation scenario also presents, in case of the variable-interval poller, a comparison between different slack usage policies with respect to the residual packet drop ratio. However, the difference with the second scenario is that a high processor utilization is now caused by accepting more flows instead of making the delay requirements tighter. The fourth simulation scenario presents a comparison between an SCO channel and the variable-interval poller in case of the hybrid-checked slack usage policy.

5.4.1 Description of the Simulation environment

The simulation tool used for the evaluations is Network Simulator (ns2) [ns2] with Bluetooth extensions [Nie00] from Ericsson Switchlab, together with our ns2 implementations of the predictive fair poller (see Chapter4), the different slack usage policies defined in section 5.3, and the two-state DTMC Bluetooth bit error model defined in section 5.4.1.1.

5.4.1.1 Bluetooth bit error model

Bluetooth is a wireless access technology, which is inherently subject to transmission errors because of a variety of reasons. Some of them are

- The quality of the link (e.g. bit error rate) between two Bluetooth nodes strongly depends on the distance between these nodes. Furthermore, the quality of the link is adversely affected by shadowing and (multipath) fading.
- Bluetooth uses a frequency hopping mechanism where each transmission takes place in one of the 79 (23 in some countries) available frequencies. The frequency hopping sequence is a pseudo-random one, and is unique for a piconet. Nonetheless, multiple piconets may sometimes perform a transmission using the same frequency as the number of possible frequencies is relatively small. Piconets in the vicinity of each other may therefore suffer from collisions.
- Bluetooth operates in the 2.4 GHz ISM (Industrial Scientific Medical) band, which for instance the IEEE 802.11 access technology also operates in. Hence, Bluetooth nodes suffer from possible interference of IEEE 802.11 nodes.
- Bluetooth is a wireless access technology that has potential applications in an in-home environment. The microwave is an in-home device that transmits energy (noise) in part of the ISM band.

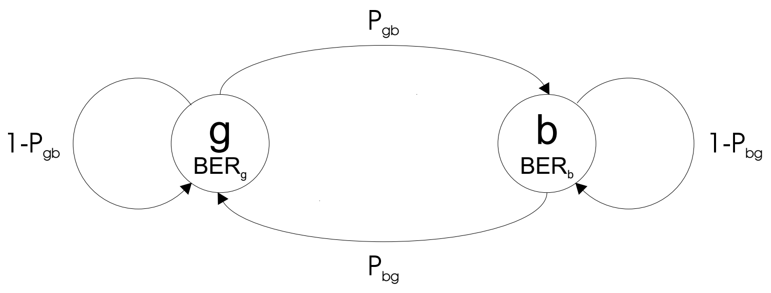


Figure 5.10: *Two-state Discrete-Time Markov Chain (DTMC).*

Determining how the reasons mentioned above translate into the actual error model is a complex task, which also strongly depends on the assumptions made for each of the reasons. However, especially from the last two reasons, it can be concluded that the packet errors will probably be bursty. As the distance between two wireless nodes is determining for the link quality, it is decided to model the link errors independently for each master slave pair.

One of the goals of this section is to investigate the influence of the burstiness of the packet errors on the error recovery mechanisms described in this chapter. For this, it suffices to come up with an error model that can be set at different burstiness levels and at different mean bit error ratios, rather than trying to come up with a model that incorporates the reasons mentioned above in a lot of detail.

The radio link of each master slave pair is modeled as a two-state Discrete-Time Markov Chain (DTMC) [Ros96] as depicted in Figure 5.10. The ”g” state is the good state, which is associated with a low bit error ratio[1] BER_{g}, while the ”b” state is the bad state, which is associated with a high bit error ratio $BER_{\mathrm{b}} > BER_{\mathrm{g}}$. The transition probabilities from the ”g” state to the ”b” state and from the ”b” state to the ”g” state are given by P_{gb} and P_{bg}, respectively. The transitions take place at each slot boundary, while it is assumed that the bit error ratio of a multi-slot packet is given by the bit error ratio in its first slot. The main idea behind this assumption is the fact that the whole baseband packet is transmitted using the RF hop frequency of its first slot.

The different error recovery mechanisms will be evaluated for different average bit error ratios and different burstiness levels. As can be seen in Figure 5.10, there are four degrees of freedom in the DTMC. It is decided to fix the bit error ratios BER_{g} and BER_{b} corresponding to the two states and to express the transition probabilities P_{bg} and P_{gb} as a function of the average bit error ratio BER and the coefficient of autocorrelation ρ_X, which is a measure of burstiness. These will be defined in this section.

Let X_i be the bit error ratio in slot i, i.e., $X_i = BER_{\mathrm{g}}$ if slot i is in the good state, and $X_i = BER_{\mathrm{b}}$ if slot i is in the bad state. Furthermore, let us first introduce P_{g} and P_{b}, which are respectively the probability of a slot being in the good state and the probability of a slot being in the bad state. The probability of a slot being in the good state is defined as [Ros96]

[1]In the literature, the a priori bit error probability of a wireless link is referred to as bit error ratio BER.

$$P_{\mathrm{g}} = P(X_i = BER_{\mathrm{g}}) = \frac{P_{\mathrm{bg}}}{P_{\mathrm{gb}} + P_{\mathrm{bg}}}, \tag{5.36}$$

while the probability of a slot being in the bad state is defined as

$$P_{\mathrm{b}} = P(X_i = BER_{\mathrm{b}}) = \frac{P_{\mathrm{gb}}}{P_{\mathrm{gb}} + P_{\mathrm{bg}}}. \tag{5.37}$$

The probabilities of (5.36) and (5.37) can be interpreted as the steady state probabilities if the Markov chain is irreducible and aperiodic. The two-state discrete-time Markov chain is irreducible if

$$P_{\mathrm{gb}} > 0 \quad \text{and} \quad P_{\mathrm{bg}} > 0, \tag{5.38}$$

while the two-state discrete-time Markov chain is aperiodic if

$$P_{\mathrm{gb}} < 1 \quad \text{and} \quad P_{\mathrm{bg}} < 1. \tag{5.39}$$

As mentioned before, the good state is associated with the lower bit error ratio BER_{g}, while the bad state is associated with the higher bit error ratio BER_{b}. Consequently, the average bit error ratio is given by

$$BER = \mathrm{E}[X_i] = P_{\mathrm{g}} BER_{\mathrm{g}} + P_{\mathrm{b}} BER_{\mathrm{b}}. \tag{5.40}$$

By substitution of P_{g} and P_{b} from (5.36) and (5.37) respectively in (5.40), the average bit error ratio can be written as a function of the transition probabilities, i.e.,

$$BER = \frac{P_{\mathrm{bg}} BER_{\mathrm{g}} + P_{\mathrm{gb}} BER_{\mathrm{b}}}{P_{\mathrm{gb}} + P_{\mathrm{bg}}}. \tag{5.41}$$

Let us now introduce the coefficient of autocorrelation ρ_X of the Markov chain, which corresponds to the correlation between the bit error ratio in two subsequent slots, which can be seen as a measure of burstiness. The coefficient of autocorrelation of the Markov chain is given by [YG99]

$$\rho_X = \frac{\mathrm{Cov}[X_i, X_{i+1}]}{\sqrt{\mathrm{Var}[X_i]\mathrm{Var}[X_{i+1}]}} = \frac{\mathrm{Cov}[X_i, X_{i+1}]}{\mathrm{Var}[X_i]}, \tag{5.42}$$

where

$$\begin{aligned} \mathrm{Var}[X_i] &= \mathrm{E}[X_i^2] - \mathrm{E}[X_i]^2 \\ &= (P_{\mathrm{g}} BER_{\mathrm{g}}^2 + P_{\mathrm{b}} BER_{\mathrm{b}}^2) - (P_{\mathrm{g}} BER_{\mathrm{g}} + P_{\mathrm{b}} BER_{\mathrm{b}})^2, \end{aligned} \tag{5.43}$$

and

$$\begin{aligned} \mathrm{Cov}[X_i, X_{i+1}] &= \mathrm{E}[X_i X_{i+1}] - \mathrm{E}[X_i]\mathrm{E}[X_{i+1}] \\ &= (P_{\mathrm{g}}(1 - P_{\mathrm{gb}}) BER_{\mathrm{g}}^2 + P_{\mathrm{g}} P_{\mathrm{gb}} BER_{\mathrm{g}} BER_{\mathrm{b}} + P_{\mathrm{b}}(1 - P_{\mathrm{bg}}) BER_{\mathrm{b}}^2 + \\ &\quad P_{\mathrm{b}} P_{\mathrm{bg}} BER_{\mathrm{b}} BER_{\mathrm{g}}) - (P_{\mathrm{g}} BER_{\mathrm{g}} + P_{\mathrm{b}} BER_{\mathrm{b}})^2 \end{aligned} \tag{5.44}$$

By substitution of P_{g} and P_{b} from (5.36) and (5.37) respectively in (5.43), the variance of the Markov chain can be written as a function of the transition probabilities, i.e.,

$$\mathrm{Var}[X_i] \quad = \quad \frac{P_{\mathrm{bg}}P_{\mathrm{gb}}}{(P_{\mathrm{gb}}+P_{\mathrm{bg}})^2}(BER_{\mathrm{b}}-BER_{\mathrm{g}})^2. \tag{5.45}$$

By substitution of P_{g} and P_{b} from (5.36) and (5.37) respectively in (5.44), the covariance of the Markov chain can be written as a function of the transition probabilities, i.e.,

$$\mathrm{Cov}[X_i, X_{i+1}] \quad = \quad \frac{P_{\mathrm{gb}}P_{\mathrm{bg}}}{(P_{\mathrm{gb}}+P_{\mathrm{bg}})^2}(1-P_{\mathrm{gb}}-P_{\mathrm{bg}})(BER_{\mathrm{b}}-BER_{\mathrm{g}})^2 \tag{5.46}$$

By substitution of $\mathrm{Cov}[X_i, X_{i+1}]$ from (5.46) and $\mathrm{Var}[X_i]$ from (5.45) in (5.42), the coefficient of autocorrelation of the Markov chain can be written as

$$\rho_X = 1 - P_{\mathrm{gb}} - P_{\mathrm{bg}} \tag{5.47}$$

It can be seen from (5.47) and Figure 5.10, that $P_{\mathrm{gb}} = 1 - P_{\mathrm{bg}}$ in an uncorrelated Markov chain, i.e., when $\rho_X = 0$. Consequently, the probability of moving from one state (e.g. state ”g”) to a different one (e.g. state ”b”) is the same as the probability of being in that different state (e.g. state ”b”) and staying in that different state (e.g. state ”b”).

A positively fully correlated system, i.e., $\rho_X = 1$, implies that $P_{\mathrm{gb}} = P_{\mathrm{bg}} = 0$, which means that the next state is alway the same as the current state. On the other hand, a negatively fully correlated system, i.e., $\rho_X = -1$, implies that $P_{\mathrm{gb}} = P_{\mathrm{bg}} = 1$, which means that the next state is always different from the current state.

The transition probabilities P_{gb} and P_{bg} can be written as a function of the average bit error ratio, the coefficient of autocorrelation of the Markov chain, and the (fixed) bit error ratios of the ”g” state and the ”b” state by solving them from (5.41) and (5.47). In that case, the transition probability from the ”g” state to the ”b” state will be given by

$$P_{\mathrm{gb}} = \frac{(1-\rho_X)(BER - BER_{\mathrm{g}})}{BER_{\mathrm{b}} - BER_{\mathrm{g}}}, \tag{5.48}$$

while the transition probability from the ”b” state to the ”g” state will the be given by

$$P_{\mathrm{bg}} = \frac{(1-\rho_X)(BER_{\mathrm{b}} - BER)}{BER_{\mathrm{b}} - BER_{\mathrm{g}}}. \tag{5.49}$$

In order to comply with (5.38), the average bit error ratio must comply with

$$BER > BER_{\mathrm{g}} \quad \text{and} \quad BER < BER_{\mathrm{b}} \tag{5.50}$$

In order to comply with (5.39), the average bit error ratio must comply with

$$BER < \frac{BER_{\mathrm{b}} - \rho_X BER_{\mathrm{g}}}{1-\rho_X} \quad \text{and} \quad BER > \frac{BER_{\mathrm{g}} - \rho_X BER_{\mathrm{b}}}{1-\rho_X} \tag{5.51}$$

From (5.50) and (5.51) it can be found that the average bit error rate should comply with

$$\max(BER_{\mathrm{g}}, \frac{BER_{\mathrm{g}} - \rho_X BER_{\mathrm{b}}}{1-\rho_X}) < BER < \min(BER_{\mathrm{b}}, \frac{BER_{\mathrm{b}} - \rho_X BER_{\mathrm{g}}}{1-\rho_X}) \quad (5.52)$$

The simulations in this chapter will be performed for different values of the average bit error ratio BER and for different values of the coefficient of autocorrelation ρ_X. Each combination of BER and ρ_X will be checked for compliance to (5.52) and translated to the transition probabilities (using (5.48) and (5.49)) of the discrete-time Markov chain that models the bit errors.

5.4.1.2 Translation of bit errors to packet errors

The bit errors within a received baseband packet are assumed to occur independently of each other and with a probability p (referred to as bit error ratio in the previous section) that depends on the state of the DTMC (described in Section 5.4.1.1) in which the transmission of that packet is started. Using this information, the bit error probability can be translated into a baseband packet error probability, which depends on the baseband packet type and on the size of payload.

A baseband packet is assumed to be correctly received if the sync word, the packet header and the payload are correctly received. The sync word is a 64-bit code word that is part of the access code, which is part of each baseband packet. The sync word is allowed to contain at most 13 bit errors, which implies that, given a bit error probability p , the probability P_{sw} of a sync word being erroneously received is

$$P_{\mathrm{sw}}(p) = \sum_{n=14}^{64} \binom{64}{n} (1-p)^{64-n} p^n. \quad (5.53)$$

The packet header consists of 18 information bits including the header error correction (HEC), and is protected with a rate 1/3 FEC. Consequently, the resulting 54-bit packet header is assumed to be received correctly as long as no more than one bit error occurs in each 3-bit block (1 user data bit plus 2 parity bits). Hence, given a bit error probability p, the probability P_{h} of a packet header being received erroneously is

$$P_{\mathrm{h}}(p) = 1 - \left(\sum_{n=0}^{1} \binom{3}{n} (1-p)^{3-n} p^n \right)^{18}. \quad (5.54)$$

Besides on the bit error probability p, the probability of the payload being received erroneously depends on both the packet type and the size of payload. In an ACL link, a classification of packet types can be made based on the error correction provided. The DH1, DH3, and DH5 packets are the Data - High rate packets, which contain a 16-bit cyclic redundancy check (CRC) code but that do not provide forward error correction (FEC). Besides the 16-bit CRC code, the payload of the DH packets consists of the user data and an 8-bit payload header for the DH1 packet type or a 16-bit payload header for the DH3 and DH5 packets. The payload of the DH packets is discarded as soon as a single bit error occurs. Hence, given an amount of s_{u} (in bits) of user data and a bit error probability p, the probability P_{DH1p} of the payload of a DH1 (at most 216-bit payload) packet being received erroneously is given by

$$P_{\mathrm{DH1p}}(p, s_{\mathrm{u}}) = 1 - (1-p)^{s_{\mathrm{u}}+24}, \quad 1 \leq s_{\mathrm{u}} \leq 216. \tag{5.55}$$

Furthermore the probability P_{DH3p} of the payload of a DH3 (at most 1464-bit payload) packet being received erroneously is given by

$$P_{\mathrm{DH3p}}(p, s_{\mathrm{u}}) = 1 - (1-p)^{s_{\mathrm{u}}+32}, \quad 1 \leq s_{\mathrm{u}} \leq 1464, \tag{5.56}$$

while the probability P_{DH5p} of the payload of a DH5 (at most 2714-bit payload) packet being received erroneously is given by

$$P_{\mathrm{DH5p}}(p, s_{\mathrm{u}}) = 1 - (1-p)^{s_{\mathrm{u}}+32}, \quad 1 \leq s_{\mathrm{u}} \leq 2714. \tag{5.57}$$

The DM1, DM3, and DM5 packets are the Data - Medium rate packets, in which the data plus a 16-bit CRC code are coded with a rate 2/3 FEC, which adds 5 parity bits to each 10 subsequent user data bits[2]. Besides the 16-bit CRC code, the payload of the DM packets consists of the user data and a 8-bit payload header for the DM1 packet type or a 16-bit payload header for the DM3 and DM5 packets. The payload of a DM packet is discarded as soon as more than one bit error occurs in one of the 15-bit blocks (10 user data bits plus 5 parity bits). Hence, given an amount of s_{u} (in bits) of user data and a bit error probability p, the probability P_{DM1p} of the payload of a DM1 (at most 136 bits of user data) packet being received erroneously is given by

$$P_{\mathrm{DM1p}}(p, s_{\mathrm{u}}) = 1 - \left(\sum_{n=0}^{1} \binom{15}{n} (1-p)^{15-n} p^n\right)^{\lceil \frac{s_{\mathrm{u}}+24}{10} \rceil}, \quad 1 \leq s_{\mathrm{u}} \leq 136. \tag{5.58}$$

Furthermore, the probability P_{DM3p} of the payload of a DM3 (at most 968 bits of user data) packet being received erroneously is given by

$$P_{\mathrm{DM3p}}(p, s_{\mathrm{u}}) = 1 - \left(\sum_{n=0}^{1} \binom{15}{n} (1-p)^{15-n} p^n\right)^{\lceil \frac{s_{\mathrm{u}}+32}{10} \rceil}, \quad 1 \leq s_{\mathrm{u}} \leq 968, \tag{5.59}$$

while the probability P_{DM5p} of the payload of a DM5 (at most 1792-bit payload) packet being received erroneously is given by

$$P_{\mathrm{DM5p}}(p, s_{\mathrm{u}}) = 1 - \left(\sum_{n=0}^{1} \binom{15}{n} (1-p)^{15-n} p^n\right)^{\lceil \frac{s_{\mathrm{u}}+32}{10} \rceil}, \quad 1 \leq s_{\mathrm{u}} \leq 1792. \tag{5.60}$$

In an SCO link, and given the class of high quality voice (HV) packets, a classification of packet types based on the error correction provided can be made. While none of the HV

[2]In case the number of user data bits is not a multiple of 10, tail bits will be appended as the forward error encoder operates with information segments of 10 bits

packets contains a CRC code (no retransmission), the HV3 packet type contains at most 30 information bytes (see also footnote on Page 108) and no FEC, and each bit error is passed to the upper layers as is. Given an amount s_u (in bits) of user data and a bit error probability p, the probability P_HV3p of the payload of an HV3 packet (at most 240 bits of user data) being received erroneously is given by

$$P_\mathrm{HV3p}(p, s_\mathrm{u}) = 1 - (1-p)^{s_\mathrm{u}}, \quad 1 \le s_\mathrm{u} \le 240. \tag{5.61}$$

The HV2 packet type contains at most 20 information bytes and is protected with a rate 2/3 FEC. Consequently, the HV2 packet can be received correctly as long as no more than one bit error occurs in each 15-bit block (10 user data bits plus 5 parity bits). Given an amount s_u (in bits) of user data and a bit error probability p, the probability P_HV2p of the payload of an HV2 (at most 160 bits of user data) packet being passed erroneously to the upper layers is given by

$$P_\mathrm{HV2p}(p, s_\mathrm{u}) = 1 - \left(\sum_{n=0}^{1} \binom{15}{n} (1-p)^{15-n} p^n \right)^{\lceil \frac{s_\mathrm{u}}{10} \rceil}, \quad 1 \le s_\mathrm{u} \le 160. \tag{5.62}$$

The HV1 packets contains 10 information bytes, which are protected with a rate 1/3 FEC. Consequently the packet is passed error free as long as there is no more than one bit error in each of the 3-bit blocks (1 user data bit plus 2 parity bits). Given an amount s_u (in bits) of user data and a bit error probability p, the probability P_HV1p of the payload of an HV1 (at most 80 bits of user data) packet being passed erroneously to the upper layers is given by

$$P_\mathrm{HV1p}(p, s_\mathrm{u}) = 1 - \left(\sum_{n=0}^{1} \binom{3}{n} (1-p)^{3-n} p^n \right)^{s_\mathrm{u}}, \quad 1 \le s_\mathrm{u} \le 80. \tag{5.63}$$

In an ACL link, a baseband packet is passed (free of errors) to the upper layers if the sync word, the packet header, and the payload are received correctly. For instance, the probability P_DM3 that a DM3 baseband packet cannot be passed (free of errors) to the upper layers is

$$P_\mathrm{DM3}(p, s_\mathrm{u}) = 1 - ((1 - P_\mathrm{sw}(p))(1 - P_\mathrm{h}(p))(1 - P_\mathrm{DM3p}(p, s_\mathrm{u}))), \quad 1 \le s_\mathrm{u} \le 968. \tag{5.64}$$

A correctly received packet header contains information that is needed to identify the sender, the type of the transmission, information needed to perform flow control, information needed to perform retransmissions (if needed), and information needed to filter out duplicate transmissions. Furthermore, a correctly received packet header serves as an implicit poll (if received by a slave) or as a reply (if received by the master) to a poll. If the packet header or the sync word is erroneously received, then the packet header will be discarded. Consequently, the probability of a packet header not being processed is

$$P_\mathrm{headerNoProc}(p) = 1 - ((1 - P_\mathrm{sw}(p))(1 - P_\mathrm{h}(p))) \,. \tag{5.65}$$

In an SCO link, a baseband packet is passed free of errors to the upper layers only if the sync word, the packet header, and the payload are correctly received. Otherwise, the packet is either passed with errors in case both the sync word and the packet header are correctly received, or the packet is not passed at all in case the sync word and/or the packet header are not correctly received. For instance, the probability that a HV2 baseband packet cannot be passed free of errors to the upper layers is

$$P_{\mathrm{HV2}}(p, s_{\mathrm{u}}) = 1 - ((1 - P_{\mathrm{sw}}(p))(1 - P_{\mathrm{h}}(p))(1 - P_{\mathrm{HV2p}}(p, s_{\mathrm{u}}))), \quad 1 \leq s_{\mathrm{u}} \leq 160. \tag{5.66}$$

5.4.2 Scenario I: Comparison between the fixed-interval poller and the variable-interval poller

5.4.2.1 Purpose of the simulation

According to Chapter 4, delay guarantees are met by both the fixed-interval poller and the variable-interval poller. However, as the variable-interval poller is saving more bandwidth than the fixed-interval poller, it is expected that the variable-interval poller is able to perform more retransmissions, which should lead to a lower residual packet drop ratio.

The purpose of this simulation scenario is to investigate, in case retransmission are performed in idle time, whether the variable-interval poller is able to achieve a lower residual packet drop ratio than the fixed-interval poller. Note that the flush timeout prevents the transmission of packets whose delay guarantee would not be met otherwise. Hence, the delay guarantees of packets that are not dropped are met. For illustration purposes, the residual packet drop ratio in case no retransmission are performed will also be shown.

5.4.2.2 Description of the simulation scenario

In this scenario, simulations are performed using the simulation setup of Section 4.3.1.2 (see also Figure 4.3). In addition to the assumptions of Section 4.3.1.2, the following assumptions are made:

- The channel of each master-slave pair behaves as a two-state DTMC with $BER_{\mathrm{g}} = 10^{-5}$ and $BER_{\mathrm{b}} = 10^{-2}$, whereas ρ_X and BER are varied in the experiments. The chosen values for BER_{g} and BER_{b} make it possible to vary the average bit error ratio between 10^{-5} and 10^{-2}.
- Retransmissions of the GS traffic is only performed if the system is idle, i.e., only if there are no polls (for GS flows) of which the planned time has already arrived but which are not (yet) completely executed.

5.4.2.3 Simulation results

As mentioned in Section 4.2.2, the fixed-interval poller plans polls more often than necessary. These extra polls can be used by the polled GS flow for retransmission of the GS packets that are not correctly received. However, as the assignment of these extra polls to GS flows is independent of the need (following a transmission error) for them, it is expected that the variable-interval poller will achieve a lower residual packet drop ratio than the fixed-interval poller. The reason for this is that the variable-interval poller assigns the polls that it considered as being unnecessary to the GS flows that actually need them.

Figure 5.11 to Figure 5.16 show the total residual packet drop ratio as a function of the average bit error ratio BER for different delay requirements and different burstiness levels. Based on these figures, some observations can be made. Compared with the case in which no retransmissions take place, both the fixed-interval poller and the variable-interval poller achieve a significantly lower residual packet drop ratio.

In case of very tight delay requirements ($d^B = 23.5$ ms, $\tilde{U} \approx 0.985$) or very loose delay requirements ($d^B = 43.75$ ms, $\tilde{U} \approx 0.486$), the fixed-interval poller and the variable-interval poller achieve comparable residual packet drop ratios. For very loose delay requirements, this comparable residual packet drop ratio is caused by the fact that both the fixed-interval poller and the variable-interval poller have enough idle time to perform the retransmissions in. For very tight delay requirements, this is caused by the fact that the variable-interval poller is not able to create much additional space for retransmission. The reason for this is that the requested rate R is much higher than the data rate r^{d}, and thus that most polls do not result in baseband packets containing data, which are needed for postponing next polls (see section 4.2.2). In case of moderate delay requirements ($d^B = 35$ ms, $\tilde{U} \approx 0.622$), the variable-interval poller achieves a lower residual packet drop ratio that the fixed-interval poller, which conforms the expectation.

In case of bursty packet errors, the residual packet drop ratio is up to one order of magnitude higher than in case of independent packet errors. The reason for this is that retransmissions of packets will often experience the same channel conditions as the original packet.

5.4.2.4 Conclusions of scenario I

Except for very tight delay requirement and/or low utilization, the variable-interval poller achieves lower residual packet drop ratios than the fixed-interval poller. Furthermore, it is shown in Chapter 4 that the variable-interval poller makes it possible to achieve higher best effort throughput than with the fixed-interval poller. These two aspects make the variable interval poller superior to the fixed-interval poller.

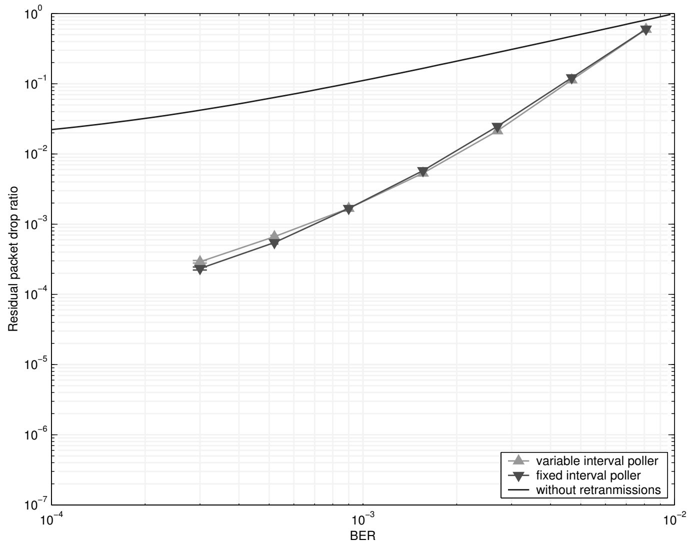


Figure 5.11: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 23.5 ms in case of independent packet errors ($\rho_X = 0$).*

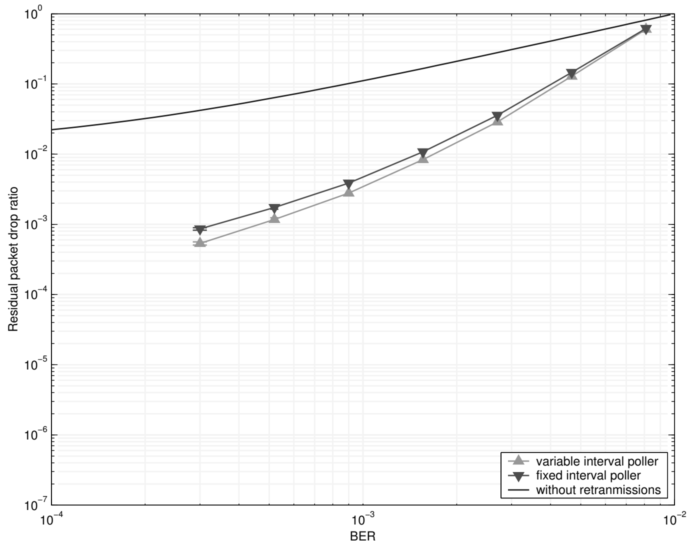


Figure 5.12: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 23.5 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

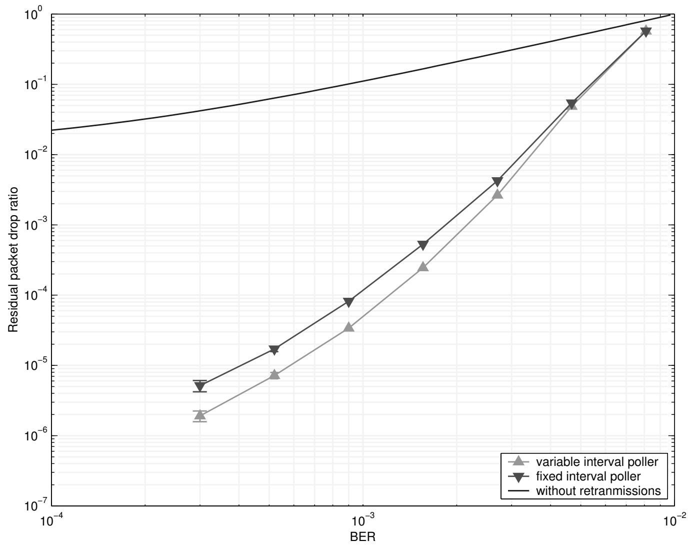


Figure 5.13: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 35 ms in case of independent packet errors ($\rho_X = 0$).*

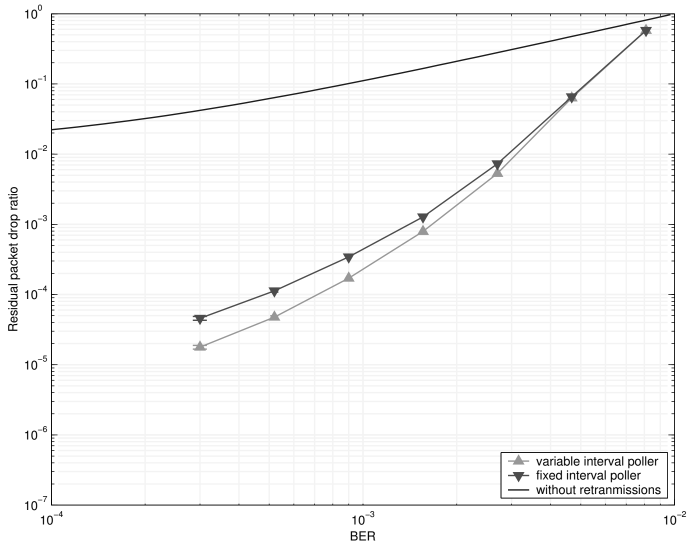


Figure 5.14: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 35 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

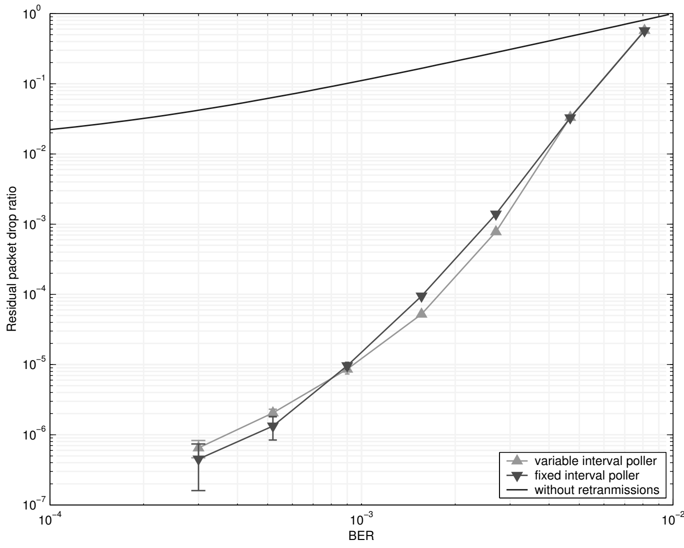


Figure 5.15: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 43.75 ms in case of independent packet errors ($\rho_X = 0$).*

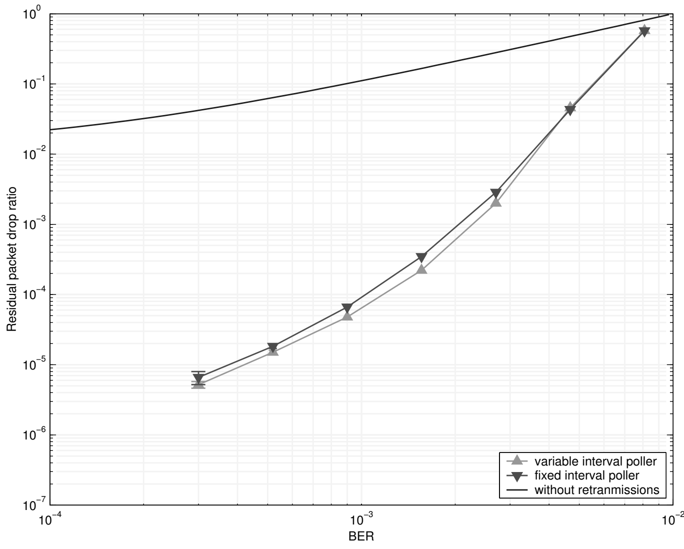


Figure 5.16: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 43.75 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

5.4.3 Scenario II: Comparison of the different retransmission policies

5.4.3.1 Purpose of the simulation

The purpose of this simulation scenario is to evaluate the different retransmission policies (see Table 5.1) if used by the variable-interval poller. First, we investigate whether slack usage leads to a lower residual packet drop ratio. For this, we compare the residual packet drop ratios achieved in case of each of the retransmission policies. Second, we compare the average number of evaluated time instances per retransmission attempt for the online-checked slack usage policy, the online-determined slack usage policy, the hybrid-checked slack usage policy, and the hybrid-determined slack usage policy. These policies differ in the number of evaluated time instance per retransmission attempt, while leading to the same residual packet drop ratio.

5.4.3.2 Description of the simulation scenario

In this scenario, simulations are performed using again the simulation setup of Section 4.3.1.2 (see also Figure 4.3). In addition to the assumptions of Section 4.3.1.2, the following assumptions are also made:

- The channel of each master-slave pair behaves as a two-state DTMC with $BER_\mathrm{g} = 10^{-5}$ and $BER_\mathrm{b} = 10^{-2}$, whereas ρ_X and BER are varied in the experiments.
- For the comparison of the residual packet drop ratio, retransmissions of the GS traffic are performed, depending on the retransmission policy, if the system is idle, in offline-determined slack time, or in hybrid-checked slack time.
- For the comparison of the average number of evaluated time instances per retransmission attempt (Γ), retransmissions of the GS traffic are performed, depending on the retransmission policy, in online-checked slack time, in online-determined slack time, in hybrid-checked slack time, or in hybrid-determined slack time.

5.4.3.3 Simulation results

Figure 5.17 to Figure 5.22 show the total residual packet drop ratio as a function of the average bit error ratio BER for different delay requirements and different burstiness levels. Based on these figures, some observations can be made.

In case of the tight delay requirement of $d^\mathrm{B} = 23.5$ ms, the same total residual packet drop ratio is achieved for the different retransmission policies. The reason for this is twofold. First the utilization is high ($\tilde{U} \approx 0.985$), meaning that the amount of idle time is scarce. Second, the requested rate R is much higher than the data rate r^d, and thus most polls do not result in baseband packets containing data, which are needed for postponing next polls and thus for creating additional idle time. The slack time using policies can be looked at as idle time moving policies. Hence, as idle time is scarce, comparable total residual packet drop ratios are achieved under the different retransmission policies.

For the remaining cases ($d^{\mathrm{B}} = 35$ ms and $d^{\mathrm{B}} = 43.75$ ms), the offline-determined slack usage policy causes a slightly lower residual packet drop ratio to be achieved compared with the case in which retransmission are performed in idle time. Furthermore, the hybrid-checked slack usage policy causes a slightly lower residual packet drop ratio to be achieved compared with the case in which retransmission are performed in offline-determined slack time.

In case of bursty packet errors, the residual packet drop ratio is higher than in case of independent packet errors. This can be seen by comparing Figure 5.17 with Figure 5.18, Figure 5.19 with Figure 5.20, or Figure 5.21 with Figure 5.22. Furthermore, the difference in residual packet drop ratio between the different slack usage policies becomes small in case of bursty packet errors. The reason for this is that retransmissions of packets will often experience the same channel conditions as the original packets.

Figure 5.23 to Figure 5.28 show the average number of evaluated time instances per retransmission attempt (Γ) for the online-checked slack usage policy, the online-determined slack usage policy, the hybrid-checked slack usage policy, and the hybrid-determined slack usage policy. The average number of evaluated time steps increases for increasing utilization (see also (5.24)). This can be seen by comparing Figure 5.23 with Figure 5.25 and Figure 5.27, or Figure 5.24 with Figure 5.26 and Figure 5.28. Furthermore, it can be seen that the average number of evaluated time instances per retransmission attempt needed by the slack checking policies is less than the average number of evaluated time instances per retransmission attempt needed by their slack determining counterparts.

The average number of evaluated time instances per retransmission attempt (Γ) increases for increasing average bit error ratio (BER). The reason for this is that in case of a higher BER, multiple retransmission attempts will be made during a single busy period. Furthermore, the more the retransmissions took place during a busy period, the closer the deadlines of the planned GS polls, and thus the larger the slack determination/checking (see effect of smaller t_{f_i}'s in (5.21) and (5.34)). An exception to the aforementioned increasing Γ occurs when the slack checking policies are used in case of a high utilization (see Figure 5.23 and Figure 5.24). The reason for this is that the slack checking algorithm stops as soon as it finds out that the requested slack is not available (see step 1 in Figure 5.8).

The difference between the average number of evaluated time instances per retransmission attempt in case of bursty packet errors and in case of independent packet errors in negligible. The reason for this is that packet error burstiness has a negligible effect on the number of retransmission attempts per busy period.

5.4.3.4 Conclusions of scenario II

From the simulation results it can be concluded that, w.r.t. residual packet drop ratio, the different retransmission policies perform equally well in case of high utilization. In case of moderate and low utilization, and especially in case of lowly correlated packet errors, a slightly lower residual packet drop ratio is achieved when following the offline-determined slack usage policy instead of performing retransmission in idle-time. A slight additional decrease in the residual packet drop ratio can be achieved by following one of the online/hybrid-determined/checked slack usage policies.

Given the extra cost in terms of computation, the usability of the online/hybrid retransmission policies in this particular context can be argued against. However, if one of the online/hybrid retransmission policies is to be used, the use of the hybrid-checked slack usage policy is recommended because of its lower average number of evaluated time instances per retransmission attempt.

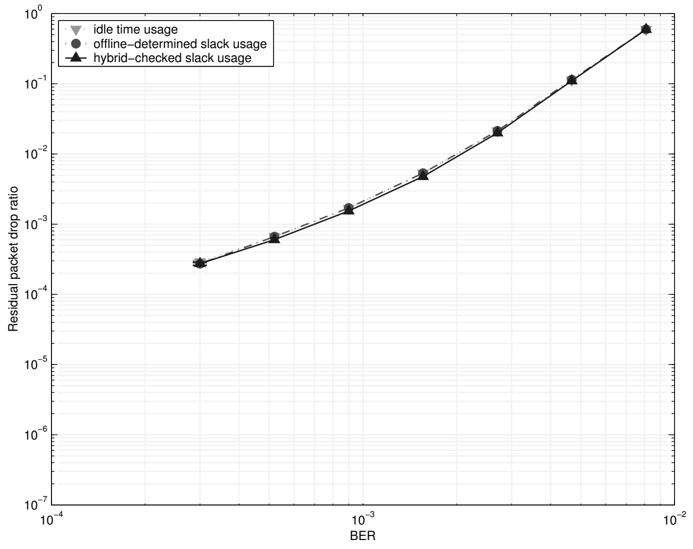


Figure 5.17: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 23.5 ms in case of independent packet errors ($\rho_X = 0$).*

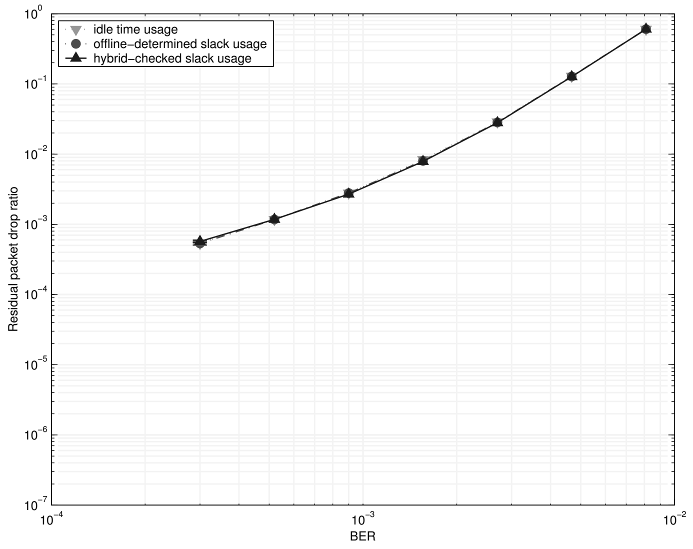


Figure 5.18: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 23.5 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

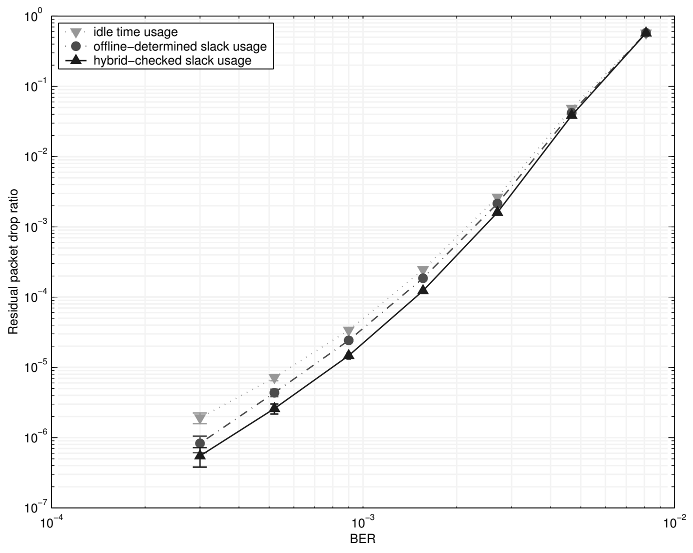


Figure 5.19: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 35 ms in case of independent packet errors ($\rho_X = 0$).*

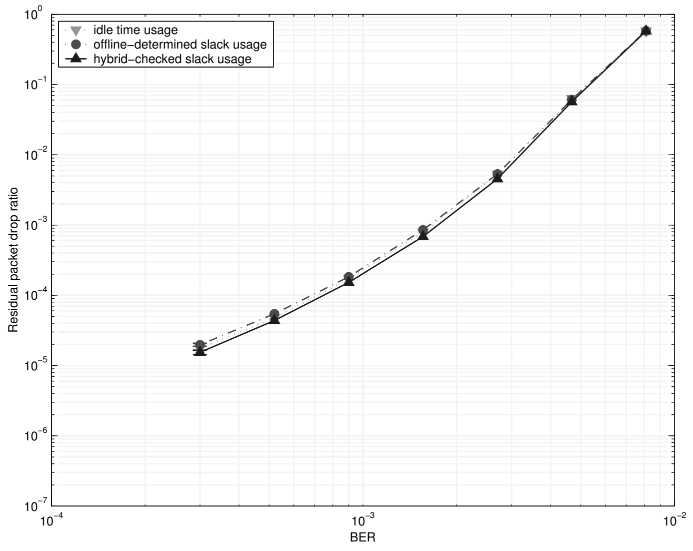


Figure 5.20: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 35 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

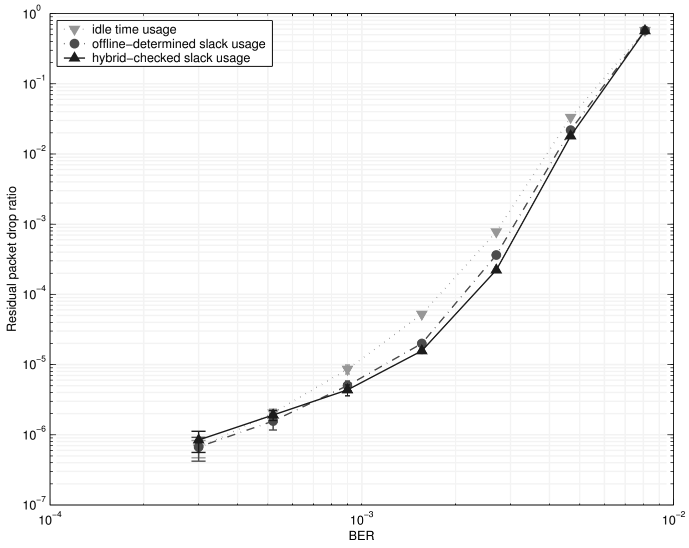


Figure 5.21: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 43.75 ms in case of independent packet errors ($\rho_X = 0$).*

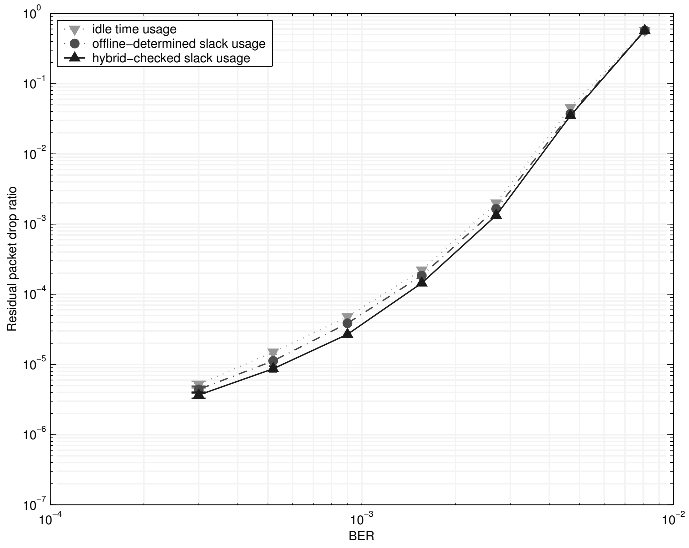


Figure 5.22: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 43.75 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

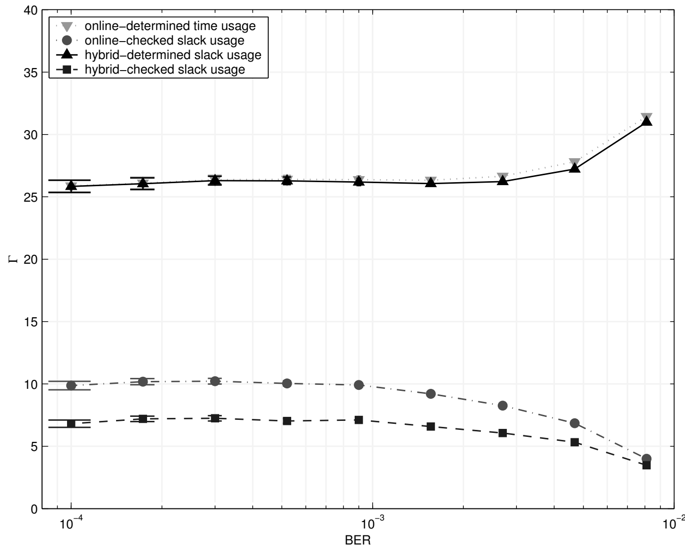


Figure 5.23: *Average number of processed time instances per retransmission attempt (Γ) as a function of average bit error ratio (BER) for a delay requirement of 23.5 ms in case of independent packet errors ($\rho_X = 0$).*

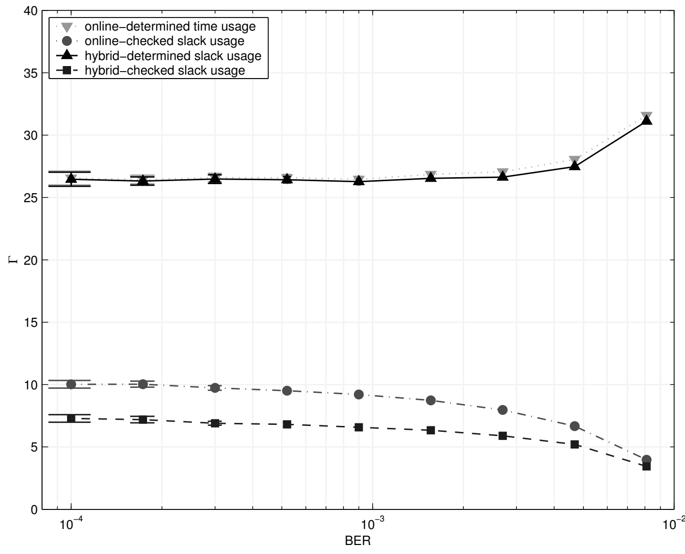


Figure 5.24: *Average number of processed time instances per retransmission attempt (Γ) as a function of average bit error ratio (BER) for a delay requirement of 23.5 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

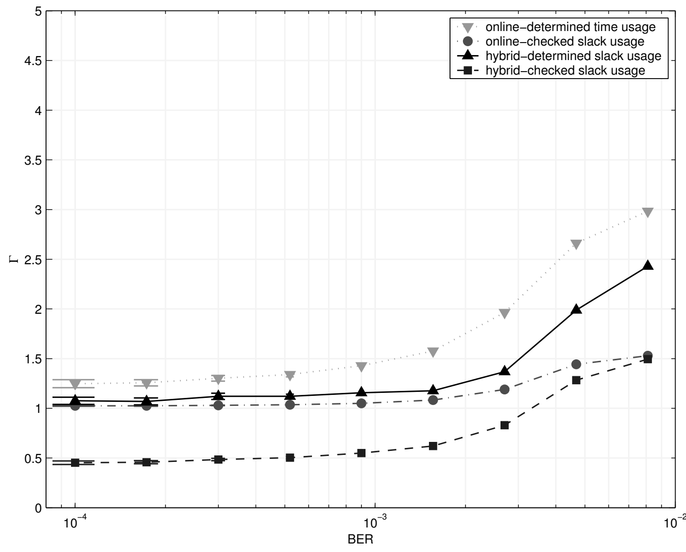


Figure 5.25: *Average number of processed time instances per retransmission attempt (Γ) as a function of average bit error ratio (BER) for a delay requirement of 35 ms in case of independent packet errors ($\rho_X = 0$).*

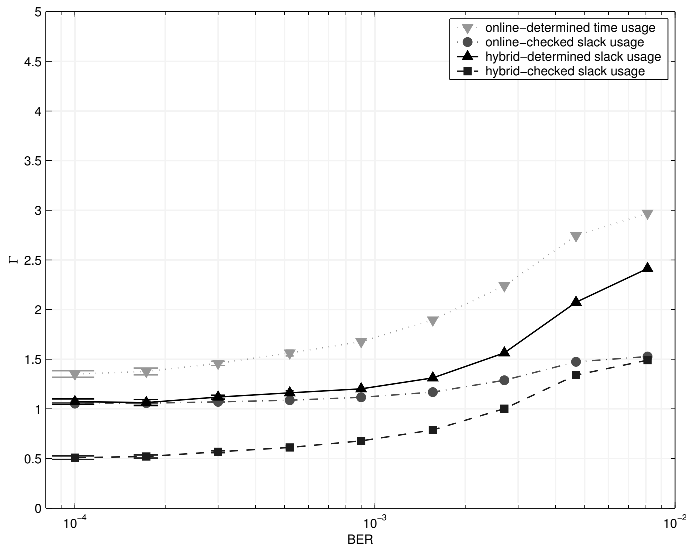


Figure 5.26: *Average number of processed time instances per retransmission attempt (Γ) as a function of average bit error ratio (BER) for a delay requirement of 35 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

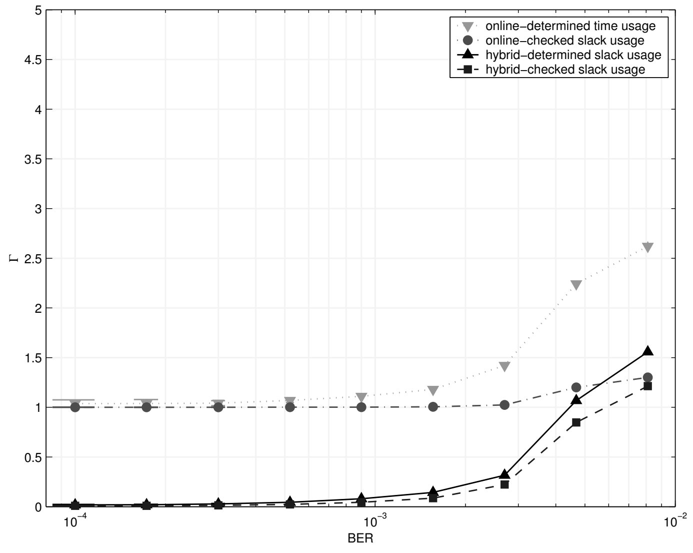


Figure 5.27: *Average number of processed time instances per retransmission attempt (Γ) as a function of average bit error ratio (BER) for a delay requirement of 43.75 ms in case of independent packet errors ($\rho_X = 0$).*

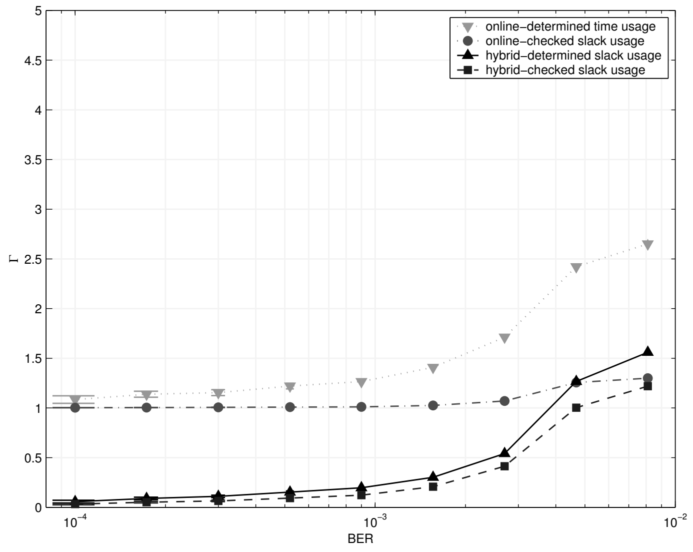


Figure 5.28: *Average number of processed time instances per retransmission attempt (Γ) as a function of average bit error ratio (BER) for a delay requirement of 43.75 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

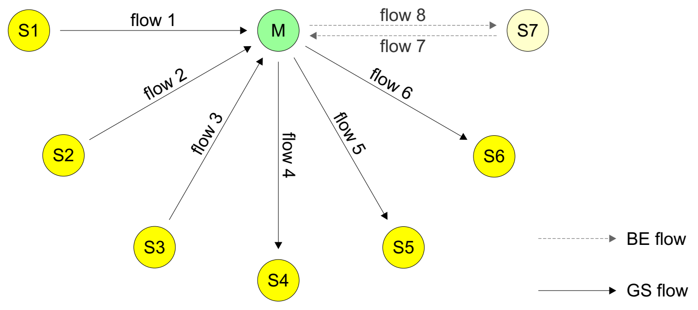


Figure 5.29: *Simulation setup of scenario III*

5.4.4 Scenario III: Comparison of the different retransmission policies for higher number of flows

5.4.4.1 Purpose of the simulation

In scenario I and II, the utilization is increased by making the delay requirement d^{B} tighter (i.e., lower). As a consequence, the resulting requested service rate R, is also higher. In case of a delay requirement of $d^{\mathrm{B}} = 23.5$ ms, the requested service rate is even twice the data rate. Consequently, many GS polls will not result in a GS segment being transmitted, which is needed by the variable-interval poller to create extra idle time.

The purpose of this simulation scenario is to show, in case of high utilization where the requested service rate is not significantly higher than the data rate, that the variable-interval poller is able to create extra idle time, which can be used for retransmissions.

5.4.4.2 Description of the simulation scenario

In this scenario, simulations are performed using the simulation setup of Figure 5.29, while making the following assumptions:

- Seven slaves and a master form a piconet, while flows are set up as depicted in the figure.
- Flows 1 to 6 are GS flows, which the same delay bound of $d^{\mathrm{B}} = 40$ ms is requested for, while flows 7 and 8 are BE flows (background traffic).
- For the GS flows, the packet sizes are uniformly distributed with a minimum size of 144 bytes, and a maximum size of 176 bytes, i.e., $m_i = 144$ bytes and $M_i = 176$ bytes for each GS flow i.
- For the BE flows, the packet sizes are of a fixed size of 176 bytes.
- The time between two consecutive packet generations of the same GS flow equals the size of the first packet divided by a data rate of 8 kbytes/s (64 kbps). The resulting average time interval between two packet generations of the same GS flow is 20 ms.

- The sources of the BE flows generate packets with fixed intervals at a data rate of 64 kbps.
- The allowed baseband packet types are DH1 and DH3, with a maximum payload size of 27 bytes and 183 bytes, respectively (including 4 bytes L2CAP header).
- The segmentation policy requires that the DH3 baseband packet is used, unless the remainder of the packet fits in the DH1 baseband packet.
- The channel of each master-slave pair behaves as a two-state DTMC with $BER_{\mathrm{g}} = 10^{-5}$ and $BER_{\mathrm{b}} = 10^{-2}$, whereas ρ_X and BER are varied in the experiments.
- For the variable-interval poller, retransmissions of the GS traffic are performed, depending on the retransmission policy, if the system is idle, in offline-determined slack time, or in hybrid-checked slack time. For the fixed-interval poller, retransmissions are performed if the system is idle.

Because of the packet size distribution and the corresponding inter-generation time of packets, the remaining parameters of the token bucket specification are

$$r_i^{\mathrm{p}} = r_i^{\mathrm{t}} = 8 \text{ kbytes/s}, \quad i \in \{1, 2, ..., 6\}, \tag{5.67}$$

and

$$b_i \geq M_i, \quad i \in \{1, 2, ..., 6\}. \tag{5.68}$$

Because of the packet sizes the source of each GS flow i can use, and because of the allowed baseband packet types, the minimum poll efficiency $\epsilon_{\mathrm{p}_i}^{\min}$ is achieved by a packet size of 144 bytes, which is sent using one DH3 baseband packet. Hence, the C error term for these flows is given by $C_i = \epsilon_{\mathrm{p}_i}^{\min} = 144$ bytes for each GS flow i. As all the nodes are allowed to use DH3 baseband packets, the possibility must be taken into account that both the master and the addressed slave transmit a DH3 packet. Consequently, the D error term is given by $D_i = 2\frac{3}{1600} = 3.75$ ms for each GS flow i.

According to Section 4.2, the GS flows 1 to 6 can be looked at as a set of six periodic or sporadic tasks dependent on whether the fixed-interval poller or the variable interval poller is considered. In both cases, each task i is described by a tuple $(p_i, e_i, d_i) = (\frac{\epsilon_{\mathrm{p}_i}^{\min}}{R_i}, s_i^{\max}, \frac{C_i}{R_i} + D_i)$. All the GS flows are described by equal traffic specifications (token bucket specification), while sharing the same piconet and thus the same maximum possible segment size ($s^{\max}$). As each GS flow is also requesting the same delay bound, the tuple describing each GS flow i can be simplified to $(\frac{144}{R}, \frac{4}{1600}, \frac{144}{R} + \frac{6}{1600})$. Considering the feasibility analysis of Section 4.2.1.1, the GS flows can be admitted as long as

$$\tilde{U} = 6\frac{\frac{4}{1600}R}{144} \leq 1. \tag{5.69}$$

Consequently, the six GS flows can be admitted as long as $R \leq 9.6$ kbytes/s. This implies that the minimum delay bound that can be requested is $\check{d}^{\mathrm{B}} \approx 37.1$ ms (see (4.1)).

Note that the requested delay bound of $d^{\mathrm{B}} = 40$ ms corresponds to a requested rate of $R \approx 8.8$ kbytes/sec ($\tilde{U} \approx 0.92$), which is slightly higher than the data rate. Consequently, the majority of planned polls will be successful, making it possible for the variable-interval poller to save bandwidth, which can be used for retransmissions.

5.4.4.3 Simulation results

Figure 5.30 and Figure 5.31 show, for a burstiness level of respectively $\rho_X = 0$ and $\rho_X = 0.7$, the total residual packet drop ratio as a function of the average bit error ratio BER in case of a delay requirement of $d^{\mathrm{B}} = 40$ms.

As can be seen in both figures, the variable-interval poller is able to translate its created extra idle time into a considerably lower residual packet drop ratio. Furthermore, the offline-determined slack is insufficient for performing a retransmission in. Consequently, the same residual packet drop ratio is achieved in case retransmission are performed in idle time and in case retransmissions are performed in offline-determined slack.

In case of the variable-interval poller with the hybrid-checked slack usage policy, an even lower residual packet drop ratio is achieved. The reason for this is that with the hybrid-checked slack usage policy the actually available slack is also checked rather than only the offline-determined slack.

Note than in case of bursty packet errors, the residual packet drop ratio increases for all the simulated pollers. This increase is more significant in case of the hybrid-checked slack usage policy. The reason behind this is that, in case of the hybrid-checked slack usage policy, retransmissions are performed as soon as possible. However, because of the high burstiness level, the retransmissions often experience the same channel conditions as the original transmission.

5.4.4.4 Conclusions of scenario III

From the simulation results, it can be concluded that, in case of high utilization and loose delay requirements, significantly lower residual packet drop ratios can be achieved by using the variable interval poller instead of the fixed-interval poller. Moreover, the use of the variable-interval poller with the hybrid-checked slack usage policy further decreases the residual packet drop ratio. Compared with the variable-interval poller with the offline-determined slack usage policy, a more than ten times lower residual packet drop ratio can be achieved by the variable-interval poller with the hybrid-checked slack usage policy.

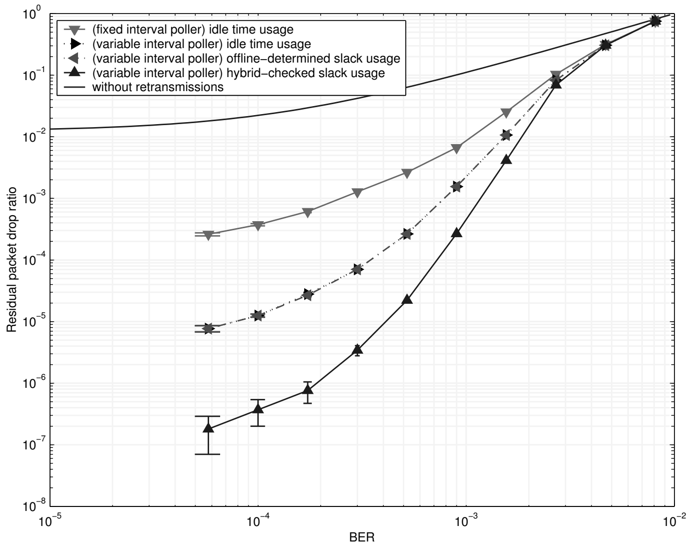


Figure 5.30: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 40 ms in case of independent packet errors ($\rho_X = 0$).*

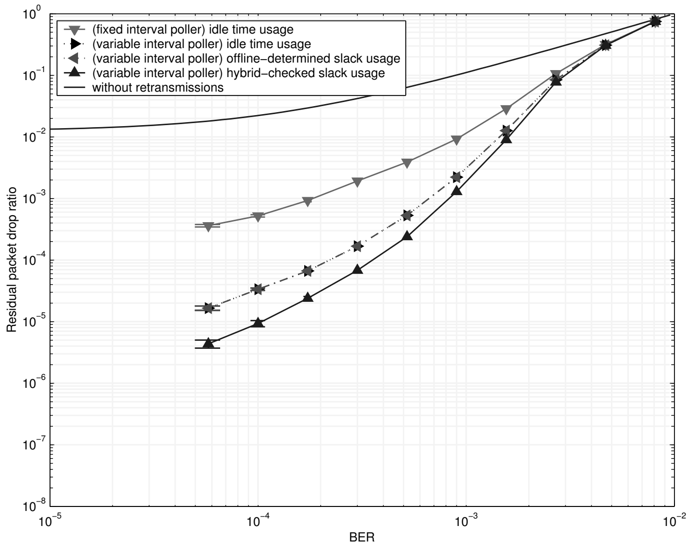


Figure 5.31: *Residual packet drop ratio as a function of average bit error ratio (BER) for a delay requirement of 40 ms in case of highly dependent packet errors ($\rho_X = 0.7$).*

5.4.5 Scenario IV: Comparison with an SCO channel

5.4.5.1 Purpose of the simulation

In an ACL channel, baseband packets that are not correctly received can be retransmitted. In an SCO channel, no retransmissions take place. However, the possibility exists for using the HV2 baseband packet type, which protects the payload with a rate 2/3 FEC, or the HV1 baseband packet type, which protects the payload with a rate 1/3 FEC.

The purpose of this simulation scenario is to compare the achieved total residual packet drop ratio in case of an SCO channel with the achieved total residual packet drop ratio in case of a variable-interval poller with the hybrid-checked slack usage policy. Recall that in the previous scenario, the lowest residual packet drop ratio is achieved using the hybrid-checked slack usage policy.

5.4.5.2 Description of the simulation scenario

In this scenario, simulations are performed using the simulation setup of Section 4.3.3.2 (see also Figure 4.8). In addition to the assumptions of Section 4.3.3.2, the following assumptions are also made:

- The channel of each master-slave pair behaves as a two-state DTMC with $BER_{\mathrm{g}} = 10^{-5}$ and $BER_{\mathrm{b}} = 10^{-2}$, whereas $\rho_X = 0.5$ and $BER = 10^{-3}$ in the experiments.
- In case of an ACL channel, which the variable-interval poller is used in, retransmissions of the GS traffic are performed in hybrid-checked slack time. In case of an SCO channel, no retransmissions are performed.
- In case of an ACL channel, simulations are performed for each of the packet type groups DH and DM. In case of an SCO channel, simulations are performed for each of the packet types HV1, HV2, and HV3.

5.4.5.3 Simulation results

Figure 5.32 and Figure 5.33 show, for a data rate of respectively $r^{\mathrm{d}} = 32\mathrm{kbps}$ and $r^{\mathrm{d}} = 64\mathrm{kbps}$, the total residual packet drop ratio as a function of the total delay requirement in case of an average bit error ratio of $BER = 10^{-3}$ and an error burstiness level of $\rho_X = 0.5$.

Except for the case in which $r^{\mathrm{d}} = 64\mathrm{kbps}$ and in which an ACL channel with DM packets is used, the use of the variable-interval poller leads to supported minimum total delay requirements comparable to those in an SCO channel. Moreover, the use of the DH or DM packet types in an ACL channel with a variable-interval poller that performs retransmissions in hybrid-checked slack time leads to a significantly lower residual packet drop ratio compared to the use of, respectively, the HV3 or HV2 in an SCO channel. In fact, the use of DH or DM packet types in an ACL channel with a variable-interval poller that performs retransmissions in hybrid-checked slack time outperforms even the use of the HV1 packet in an SCO channel. Moreover, by using the HV1 packet in an SCO channel, the data rate of

$r^{\mathrm{d}} = 64\mathrm{kbps}$ cannot be afforded in this scenario.

Note that the discontinuity in the residual packet drop ratio graphs of the ACL channel are caused by the packet size/type selection. Simulations are also performed just before and just after the point at which the discontinuity occurs. As can be seen, the packet size selection does not always lead to the lowest possible residual packet drop ratio. The reason for this is that the packet size selection is based on achieving the lowest possible utilization rather than achieving the lowest residual packet drop ratio.

In case of a data rate of $r^{\mathrm{d}} = 64\mathrm{kbps}$ and loose delay requirements, using the DH packet type group in the ACL channel leads to a lower residual packet drop ratio than when using the DM packet type group, which provides forward error correction. The reason for this is that a lower utilization is achieved when using the DH packet type group, which makes it possible to perform more retransmissions than in case of the DM packet type group.

5.4.5.4 Conclusions of scenario IV

The use of the variable-interval poller showed to be an outstanding alternative for the use of an SCO channel for providing delay guarantees in a non-ideal radio environment. The reason for this is twofold. First, comparably low total delay requirements can be guaranteed in both cases. Second, using the variable-interval poller leads to a much lower residual packet drop ratio compared to the use of an SCO channel.

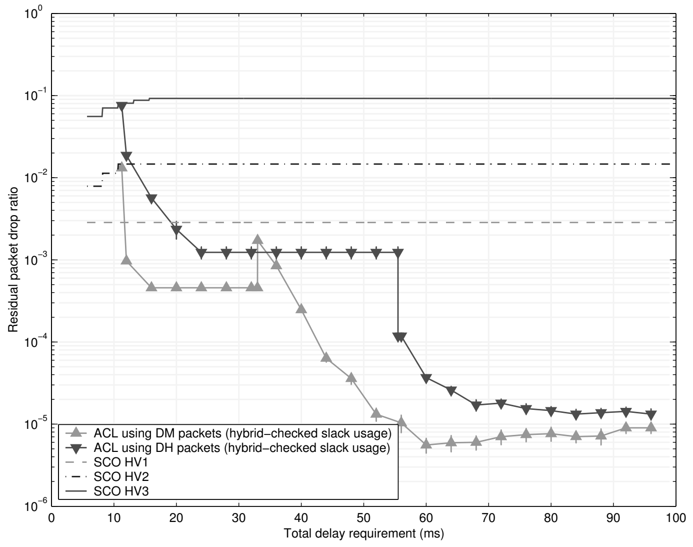


Figure 5.32: *Residual packet drop ratio as a function of the total delay requirement for a data rate of* $r^{\mathrm{d}} = 32\mathrm{kbps}$ *and an average bit error ratio of* $BER = 10^{-3}$ *in case of moderately dependent packet errors* ($\rho_X = 0.5$).

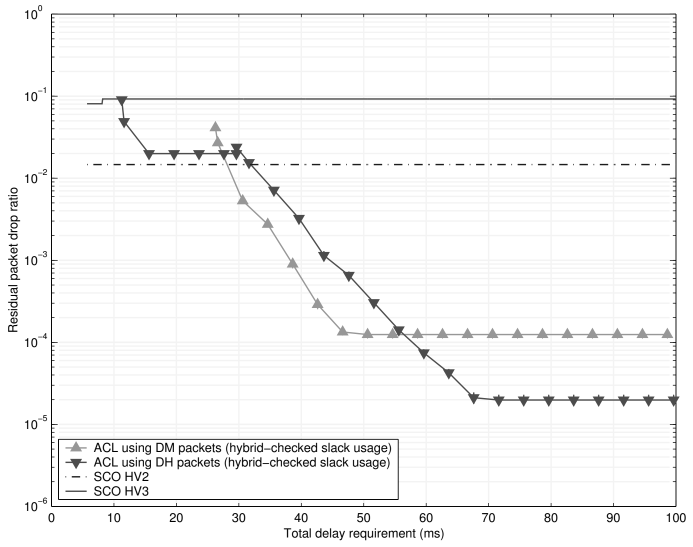


Figure 5.33: *Residual packet drop ratio as a function of the total delay requirement for a data rate of* $r^{\mathrm{d}} = 64\mathrm{kbps}$ *and an average bit error ratio of* $BER = 10^{-3}$ *in case of moderately dependent packet errors* ($\rho_X = 0.5$).

5.5 Discussion

The previous chapter discussed polling mechanisms that are able to provide QoS in an ideal radio environment, where retransmissions are not needed. This chapter discussed the same polling mechanisms while releasing the restriction of the ideal radio environment.

A timeout has been defined in which retransmission can be performed without delaying next packets of the same flow. This timeout also prevents the transmission of packets that would otherwise be late. An obvious way for the polling mechanism to perform retransmissions is to background the retransmissions, i.e., to perform retransmissions when there are no pending QoS polls. However, this may cause the aforementioned timeout to expire and the packet to be dropped. Slack using policies have been defined to perform retransmissions as soon as possible rather than waiting until the poller is idle. These policies are summarized in Table 5.1. Simulation studies showed that among the online/hybrid-determined/checked slack usage policies, the hybrid-checked slack usage policy needs the lowest average number of evaluated time instances per retransmission attempt.

Furthermore, simulation studies showed that the variable-interval poller is able to translate its created extra idle time into a lower residual packet drop ratio for the QoS flows. This is especially the case when the utilization is high and the delay requirements are loose. Moreover, the simulation studies showed that, in that case, the variable-interval poller in combination with the hybrid-checked slack usage policy leads to a significantly lower residual packet drop ratio.

Finally, simulations showed that using the variable-interval poller in combination with the hybrid-checked slack usage policy is an outstanding alternative for using an SCO channel, both in terms of achieved best effort throughput (see Chapter 4) as well in terms of the residual packet drop ratio of the GS flows.

Chapter 6

Conclusions and further work

In this dissertation, the design of new Bluetooth intra-piconet scheduling mechanisms has been addressed. These mechanisms are also referred to as Bluetooth polling mechanisms. The first goal was to design a polling mechanism that, in case of best effort traffic, divides bandwidth among the slaves in a Bluetooth piconet in a fair and efficient manner. The second goal was to design a polling mechanism that is able to provide quality of service. This chapter gives an overview of the major achievements of this work, and points out directions for further work.

6.1 Conclusions and results

Bluetooth uses a polling mechanism, which is applied by the master, to divide the bandwidth among the remaining participants (the slaves) in a piconet. This polling mechanism, also referred to as the Bluetooth intra-piconet scheduling mechanism, highly determines the performance of the traffic flows in a Bluetooth piconet.

In order for the Bluetooth technology to be a successful enabler of personal area networks, the Bluetooth polling mechanism should be efficient in order to get the most out of the scarce bandwidth. At the same time, the polling mechanism should be fair, i.e., it should not ignore lowly loaded slaves. Finally, the polling mechanism must be able to provide quality of service, which is needed to support audio and video applications.

In Chapter 3, we discussed the development of a new polling mechanism, named Predictive Fair Poller, which continuously estimates the fairness and the probability of data being available for transmission at a slave. Based on these two estimates, it decides which slave to poll next. In the best effort case, it estimates the fair share of resources for each slave and keeps track of the fraction of fair share that each slave has been given. Best Effort traffic is polled in a fair and efficient manner by keeping track of both the fairness based on the fractions of fair share and the predictions. In the QoS case, QoS requirements are negotiated with the slaves and translated into fair QoS treatments. The poller keeps track of the fraction of fair QoS treatment that each slave has been given. Similar to the Best Effort case, the Predictive Fair Poller can be used to poll QoS traffic such that the QoS requirements are efficiently met by keeping track of both the fairness based on the fractions of fair QoS treatment and the predictions.

Analytical results showed that the 1-limited Round Robin poller is not able to poll asymmetrically loaded slaves in a fair and efficient manner. By means of simulations, we compared the Predictive Fair Poller with the 1-limited Round Robin poller and the Fair Exhaustive Poller. The simulation results showed that, as opposed to the 1-limited Round Robin poller, the Predictive Fair Poller is able to poll asymmetrically loaded slaves in a fair and efficient manner. Moreover, the Predictive Fair Poller outperforms the Fair Exhaustive Poller, especially with respect to fairness.

The Data Availability Predictor of the initially developed Predictive Fair Poller is complex, as it needs a large amount of calculations for each prediction. Consequently, and based on some assumptions on the traffic, we simplified the Data Availability Predictor and came up with the simplified Predictive Fair Poller, which needs a significantly lower amount of calculations per prediction. Simulation results showed that the simplified Predictive Fair Poller achieves a performance comparable to that of the initially developed Predictive Fair Poller, which justifies the simplifications.

In Chapter 4, we discussed the development of two polling mechanisms that are capable of providing QoS. More specifically, these pollers follow the IETF's Guaranteed Service approach, hence providing both a rate guarantee and a delay guarantee. The QoS pollers of Chapter 4 are the fixed-interval poller, which polls slaves with fixed intervals, and the variable-interval poller, which postpones polls whenever possible without violating deadlines, and consequently polls slaves with variable intervals. The fixed-interval poller and the variable-interval poller make use of the concept of Guaranteed Service. They provide, with some predefined maximum deviation, a rate guarantee, which leads to a delay guarantee, provided that the traffic sources comply to their traffic flow specification. These two types of guarantees are the main QoS types that are needed for audio and video applications. Although the fixed-interval poller and the variable-interval poller can be implemented as a Predictive Fair Poller, they are not required to be so. In other words, they can be executed next to any other best effort polling mechanism, which is executed in their idle time.

Simulation studies have shown that, while providing QoS, the variable-interval poller consumes less bandwidth than the fixed-interval poller. The saved bandwidth can be used for retransmissions of the QoS traffic and/or for achieving a higher best effort throughput. The latter is shown by the simulation results of Chapter 4. A comparison with an SCO channel showed that the variable-interval poller is able to guarantee delay bounds that approach the delay bounds that can be guaranteed using an SCO channel. Moreover, the variable-interval poller is able to do so while consuming less resources. As opposed to an SCO channel, the variable-interval poller can also perform retransmissions. Using the saved bandwidth, this property can be exploited to avoid the link quality problems of SCO channels in bad radio environments, while keeping up QoS.

In Chapter 5, we discussed the same polling mechanisms as in Chapter 4, while releasing the restriction of an ideal radio environment. We defined a time period in which retransmission can be performed without unacceptably delaying next packets of the same flow. An obvious strategy of performing retransmission is to background the retransmissions, i.e., to wait until there are no pending QoS polls. The drawback of this strategy is that busy periods may be long, causing the aforementioned time period to elapse and the packet that needs a retransmission to be dropped. We defined five policies that use slack time in order to perform retransmission as soon as possible rather than to wait until the QoS poller is idle. The first policy is the offline-determined slack usage policy, which, during admission control, determines the worst case (minimum) available amount of slack in a busy period. In each busy period, a total amount of slack, up to this worst case available amount, can be consumed whenever needed within that busy period.

Busy periods that are not worst case may contain more slack than the offline-determined minimum amount of slack. In order to make use of this additional amount of slack, we defined four policies that determine or check the slack online. These policies are the online-determined slack usage policy, the online-checked slack usage policy, the hybrid-determined slack usage policy, and the hybrid-checked slack usage policy. Simulation studies showed that among these slack usage policies, the hybrid-checked slack usage policy needs the lowest average number of evaluated time instances per retransmission attempt.

Simulation studies also showed that the variable-interval poller is able to translate its generated extra idle time into a lower residual packet drop ratio for the QoS flows. This is especially the case when the utilization is high and the delay requirements are loose. Moreover, the simulation studies showed that, in that case, use of the variable-interval poller in combination with a hybrid-checked slack usage policy leads to a significantly lower residual packet drop ratio.

Finally, a comparison with an SCO channel showed that using the variable-interval poller in combination with the hybrid-checked slack usage policy is an outstanding alternative for using an SCO channel, both in terms of achieved best effort throughput and in terms of the residual packet drop ratio of the GS flows.

With the development of the polling mechanisms and techniques presented in this dissertation, the main goal of this work has been met. First, we developed a polling mechanism that is able to divide bandwidth among the slaves in a piconet in a fair and efficient manner (best effort case). Second, we developed polling mechanisms that are able to provide QoS. These polling mechanisms help in making the Bluetooth technology a successful enabler of personal area networks, and thus of personal networks.

6.2 Directions for further research

This dissertation addressed the development of intra-piconet scheduling mechanisms that help in making the Bluetooth technology a successful enabler of personal area networks and personal networks. In order to increase the potential of the Bluetooth technology even further, similar research efforts have to be conducted for *inter*-piconet scheduling. Such efforts should result in efficient communication between PANs, without using an intermediate (infrastructure) network. Furthermore, it should be investigated whether, and how, quality of service can be supported if QoS traffic traverses a scatternet.

The Predictive Fair Poller discussed in Chapter 3 determines at each poll moment which slave to poll next. However, some Bluetooth implementations may require batch poll decisions. For instance, they may require that after every n polls, the next n polls are determined at once. It is a topic of further research to investigate how this will affect the performance of the Predictive Fair Poller. The same investigations must also be done for the variable-interval poller, which generates extra idle time by postponing polls, sometimes just before they would have been executed.

In Chapter 4, we designed QoS support for Bluetooth by making use of the concept behind the IETF's Guaranteed Service. We focused on the scheduling, the determination of the C and D error terms, and on the admission control. Investigating how the various specifications (flow specification, request specification, C and D error terms, etc.) that are used by the Guaranteed Service should be exchanged between the GS sender, the intermediate nodes, and the GS receiver remains a topic for further research. For instance, it should be investigated whether RSVP is suitable for use over Bluetooth, and how the L2CAP quality of service option can be exploited for exchanging parts of the aforementioned specifications.

Bluetooth supports various low power modes, in which the power consumption of the Bluetooth nodes can be reduced at the cost of, for instance, a lower poll frequency. It should be investigated whether and how the developed polling mechanisms can cooperate with these low power modes, while keeping up fairness, efficiency, and quality of service.

For the design of Guaranteed Service support in Bluetooth, we have assumed that no inquiry or paging procedures take place. This means that no new connections can be set up while providing QoS. It is a topic for further research to investigate how inquiry and paging procedures can be performed while keeping up QoS.

In order to cope with a non-ideal radio environment, we defined a flush timeout that is calculated for each L2CAP packet, and various retransmission policies that try to perform retransmission as soon as possible. The current Bluetooth specification defines a flush timeout per ACL connection rather than per L2CAP packet. Finding out how a fixed flush timeout per ACL connection affects the designed retransmission policies remains a topic for further research.

In the recently released specification of the Bluetooth system (version 1.2) [BT003], an extended synchronous connection-oriented (eSCO) link has been introduced. The eSCO link offers a number extensions over SCO links. eSCO links support a more flexible combination of packet types, selectable data contents, and selectable slot periods. Moreover, eSCO links can offer limited retransmission of packets. These properties make eSCO more suitable than SCO for providing QoS. It is a topic for further research to compare the polling mechanisms and techniques developed in Chapter 4 and Chapter 5 with the recently introduced eSCO link.

By using the Bluetooth technology as an enabler for personal networks, the Bluetooth personal area network will interface to various communications technologies such as the Internet, UMTS, and WLAN. Research should be conducted on the interoperability of the QoS supported in each of the potential communication technologies that a Bluetooth PAN will interface to in order to provide end-to-end QoS.

In this work, we focused on scheduling, which is crucial for making the Bluetooth technology a successful enabler of personal area networks. However, there are other crucial PAN-related research issues [NHdG03] such as power consumption, security, context discovery, and cooperation with fixed infrastructures. The success of the Bluetooth technology as an enabler for personal area networks also requires research to be conducted on these issues.

Bibliography

[Alm92] P. Almquist. Type of Service in the Internet Protocol Suite. RFC 1349, IETF, July 1992.

[AYH01a] R. Ait Yaiz and G. Heijenk. Polling Best Effort Traffic in Bluetooth. In *Proceedings of the Fourth International Symposium on Wireless Personal Communications WPMC01*, pages 1381–1386, Aalborg, Denmark, September 2001.

[AYH01b] R. Ait Yaiz and G. Heijenk. Polling in Bluetooth, a Simplified Best Effort Case. In *Proceedings of the 7th annual CTIT Workshop*, pages 91–96, Enschede, the Netherlands, February 2001. Twente University Press.

[AYH02] R. Ait Yaiz and G. Heijenk. Polling Best Effort Traffic in Bluetooth. *Wireless Personal Communications*, 23(1):195–206, October 2002.

[AYH03] R. Ait Yaiz and G. Heijenk. Providing Delay Guarantees in Bluetooth. In *Proceedings of 23rd International Conference on Distributed Computing Systems Workshops ICDCSW'03*, pages 722–727, Providence, Rhode Island, May 2003. IEEE Computer Society Press.

[AYH04] R. Ait Yaiz and G. Heijenk. Providing QoS in Bluetooth. *Cluster Computing*, 2004. Submitted for publication.

[BBC$^+$98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, IETF, December 1998.

[BCG01] R. Bruno, M. Conti, and E. Gregori. Wireless Access to Internet via Bluetooth: Performance Evaluation of the EDC Scheduling Algorithm. In *Proceedings of the First Workshop on Wireless Mobile Internet*, pages 43–49, Rome, Italy, July 2001. ACM Press.

[BCS94] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633, IETF, June 1994.

[BFY$^+$00] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine. A Framework for Integrated Services Operation over Diffserv Networks. RFC 2998, IETF, November 2000.

[BLW91] O.J. Boxma, H. Levy, and J.A. Weststrate. Efficient visit frequencies for polling tables: minimization of waiting cost. *Queueing Systems*, 9:133–162, 1991.

[BM00] H. G. P. Bosch and S. J. Mullender. Real-Time Disk Scheduling in a Mixed-Media File System. Technical Report INS-R0006, CWI, Center for Mathematics and Computer Science, February 2000.

[BMR90] S.K. Baruah, A.K. Mok, and L.E. Rosier. Preemptively Scheduling Hard Real-Time Sporadic Tasks on One Processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, 1990.

[Box91] O.J. Boxma. Analysis and optimization of polling systems. In *Queueing, Performance and Control in ATM (1TC-13)*, pages 173–183, Amsterdam, The Netherlands, 1991. Elsevier Science Publishers B.V. (North-Holland).

[BR87] J.E. Baker and I. Rubin. Polling with a general-service order table. *IEEE Transactions on Communications*, 35(3):283–288, March 1987.

[BT001] Specification of the Bluetooth System; The Bluetooth Consortium, version 1.1. http://www.bluetooth.org, Februari 2001.

[BT003] Specification of the Bluetooth System; The Bluetooth Consortium, version 1.2. http://www.bluetooth.org, November 2003.

[BZB$^+$97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, IETF, September 1997.

[CGK01] A. Capone, M. Gerla, and R. Kapoor. Efficient Polling Schemes for Bluetooth picocells. In *Proceedings of the IEEE International Conference on Communications 2001*, Helsinki, Finland, June 2001. IEEE Computer Society Press.

[CH02] W. Chen and J.C. Hou. Provisioning of Temporal QoS in Bluetooth Networks. In *Proceedings of the 4th IEEE confenrence on Mobile and Wireles Communications Networks MWCN02*, Stockholm, Sweden, September 2002.

[CKK$^+$01] I. Chakraborty, A. Kashyap, A. Kumar, A. Rastogi, H. Saran, and R. Shorey. MAC Scheduling Policies with Reduced Power Consumption and Bounded Packet Delays for Centrally Controlled TDD Wireless Networks. In *Proceedings of the IEEE International Conference on Communications ICC2001*, Helsinki, Finland, June 2001. IEEE.

[CKR$^+$00] I. Chakraborty, A. Kashyap, A. Rastogi, H. Saran, R. Shorey, and A. Kumar. Policies for Increasing Throughput and Decreasing Power Consumption in Bluetooth MAC. In *Proceedings of the IEEE International Conference on Personal Wireless Communications 2000*, pages 90–94, Hyderabad, India, December 2000.

[CMU99] The CMU Monarch Project's Wireless and Mobility Extentions to ns. http://www.monarch.cs.cmu.edu/, August 1999. The CMU Monarch Project, Snapshot Release 1.1.1.

[CSS01] S. Chawla, H. Saran, and M. Singh. QoS Based Scheduling for Incorporating Variable Rate Coded Voice in Bluetooth. In *Proceedings of the IEEE International Conference on Communications ICC2001*, Helsinki, Finland, June 2001.

[DCB$^+$02] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246, IETF, March 2002.

[DGR$^+$01] A. Das, A. Ghose, A. Razdan, H. Saran, and R. Shorey. Enhancing Performance of Asynchronous Data Traffic over the Bluetooth Wireless Ad-hoc Network. In *Proceedings of IEEE Infocom*, volume 1, pages 591–600, Anchorage, Alaska, April 2001. IEEE.

[EKW99] R. Epsilon, J. Ke, and C. Williamson. Analysis of ISP IP/ATM Network Traffic Measurements. *Performance Evaluation Review*, 27(2):15–24, 1999.

[GB95] T.M. Ghazalie and T.P. Baker. Aperiodic Servers in a Deadline Scheduling Environment. *Journal of Real-Time Systems*, 9(1):31–67, July 1995.

[GGPR96] L. Georgiadis, R. Guérin, V. Peris, and R. Rajan. Efficient Support of Delay and Rate Guarantees in an Internet. In *Proceedings of the ACM SIGCOMM'96*, pages 106–116, Stanford, California, August 1996.

[GMR95] L. George, P. Muhlethaler, and N. Rivierre. Optimality and Non-Preemptive Real-Time Scheduling Revisited. Technical Report 2516, Institut National de Researche en Informatique et Automatique, April 1995.

[GRS96] L. George, N. Rivierre, and M. Spuri. Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling. Technical Report 2966, Institut National de Researche en Informatique et Automatique, September 1996.

[Haa98] J. C. Haartsen. Bluetooth - the universal radio interface for ad-hoc, wireless connectivity. *Ericsson Review*, 3:110–117, 1998.

[Haa00] J. C. Haartsen. The Bluetooth Radio System. *IEEE Personal Communications Magazine*, 7(1):28–36, February 2000.

[HAY01] G. Heijenk and R. Ait Yaiz. Predictive Fair Polling in a Wireless Access Scheme, Application for United States Patent, No. USPTO 09/954,780, 2001. Filed September 17, 2001.

[HBWW99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597, IETF, June 1999.

[HV95] R.R. Howell and M.K. Venkatrao. On Non-Preemptive Scheduling of Recurring Tasks Using Inserted Idle Time. *Information and Computation Journal*, 117(1):50–62, February 1995.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, New York, NY, April 1991.

[JCH84] R. K. Jain, D. W. Chiu, and W. R. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared computer systems. Technical Report DEC-TR-301, Digital Equipment Corporation, September 1984.

[JHM02] P.G. Jansen, F.T.Y. Hanssen, and S.J. Mullender. ClockWork: a Real-Time Feasibility Analysis Tool. Technical Report TR-CTIT-02-12, Center for Telematics and Information Technology, University of Twente, June 2002.

[JKJ99] N. J. Johansson, U. Körner, and P. Johansson. Performance Evaluation of Scheduling Algorithms for Bluetooth. In *Proceedings of IFIP TC6 Fifth International Conference on Broadband Communications '99*, pages 139–150, Hong-Kong, November 1999. Kluwer.

[JKKG01] P. Johansson, M. Kazantzidis, R. Kapoor, and M. Gerla. Bluetooth: an enabler for personal area networking. *IEEE Network*, 15(5):28–37, Sept-Oct 2001.

[JL99] P.G. Jansen and R. Laan. The Stack Resource Protocol Based on Real-Time Transactions. *IEE Proceedings-Software*, 146(2):112–119, April 1999.

[JSM91] K. Jeffay, D.F. Stanat, and C.U. Martel. On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. In *Proceedings of the Twelfth IEEE Real-Time Systems Symposium*, pages 129–139, San Antonio, December 1991. IEEE Computer Society Press.

[KBS99] M. Kalia, D. Bansal, and R. Shorey. MAC Scheduling and SAR policies for Bluetooth: A Master Driven TDD Pico-Cellular Wireless System. In *Proceedings of the Sixth International Workshop on Mobile Multimedia Communications*, pages 384–388, San Diego, California, November 1999.

[KBS00] M. Kalia, D. Bansal, and R. Shorey. Data Scheduling and SAR for Bluetooth MAC. In *Proceedings of IEEE Vehicular Technology Conference (VTC)*, Tokyo, Japan, May 2000.

[KN80] K.H. Kim and M. Naghibdadeh. Prevention of Task Overruns in Real-Time Non-Preemptive Multiprogramming Systems. In *Proceedings of Performance*, pages 267–276. ACM Press, March 1980.

[Kue79] P.J. Kuehn. Multiqueue Systems with Nonexhaustive Cyclic Service. *The Bell System Technical Journal*, 58(3):671–698, March 1979.

[LL73] C.L. Lui and W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the Association for Computing Machinery*, 20(1), January 1973.

[LRT92] J. P. Lehoczky and S. Ramos-Thuel. An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Fixed-Priority Preemptive Systems. In *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pages 110–123, Phoenix, Arizona, December 1992. IEEE Computer Society Press.

[LSS87] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In *Proceedings of the 8th IEEE Real-Time Systems Symposium*, pages 261–270, San Jose, California, December 1987. IEEE Computer Society Press.

[LT02] J. Lapeyrie and T. Turletti. Adding QoS Support for Bluetooth Piconet. Technical Report 4514, Institut National de Researche en Informatique et Automatique, July 2002.

[MKM04] V.B. Mišić, E.W.S. Ko, and J. Mišić. Load and QoS-Adaptive Scheduling in Bluetooth Piconets. In *Proceedings of the 37th Hawaii International Conference on System Sciences HICSS04*, Big Island, Hawaii, January 2004.

[MMG03] A. Mercier, P. Minet, and L. George. Introducing QoS support in Bluetooth Piconet with a Class-Based EDF Scheduling. Technical Report 5054, Institut National de Researche en Informatique et Automatique, December 2003.

[MR99] D.C. Montgomery and G.C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, second edition, 1999.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, IETF, December 1998.

[NHdG02] I.G. Niemegeers and S.M. Heemstra de Groot. From Personal Area Networks to Personal Networks: a User Oriented Approach. *Wireless Personal Communications*, 22(2):175–186, August 2002.

[NHdG03] I.G. Niemegeers and S.M. Heemstra de Groot. Research Issues in Ad-Hoc Distributed Personal Networking. *Wireless Personal Communications*, 26(2):149–167, 2003.

[Nie00] J. Nielsen. IP Routing Performance in Bluetooth Scatternets: a Simulation Study. Master's thesis, Department of Computer Systems (DoCS), Uppsala University, Uppsala, 2000.

[ns2] The Network Simulator (ns2). Software and documentation available from http://www.isi.edu/nsnam/ns.

[Par92] C. Partridge. A Proposed Flow Specification. RFC 1363, IETF, September 1992.

[Par93] C. Partridge. *Gigabit Networking*. Addison-Wesley, second edition, December 1993.

[PH03] M. Perillo and W.B. Heinzelman. ASP: An Adaptive Energy-Efficient Polling Algorithm for Bluetooth Piconets. In *Proceedings of the 36th Hawaii International Conference on System Sciences HICSS03*, Big Island, Hawaii, January 2003.

[RBK01] R. Rao, O. Baux, and G. Kesidis. Demand-based Bluetooth Scheduling. In *Web-Based Proceedings of the Third IEEE Workshop on Wireless Local Area Networks*, Boston, Massachusetts, September 2001. http://www.wlan01.wpi.edu/proceedings/index2.html.

[RJC87] K. K. Ramakrishnan, R. K. Jain, and D. W. Chiu. Congestion Avoidance in Computer Networks with a Connectionless Network Layer, Part IV: a Selective Feedback Scheme for General Topologies. Technical Report DEC-TR-510, Digital Equipment Corporation, August 1987.

[Ros96] S. Ross. *Stochastic Processes*. John Wiley & Sons, New York, NY, second edition, 1996.

[Rui01] J.A.L.J. Ruijs. Piconet scheduling. Master's thesis, Faculty of Electrical Engineering, University of Twente, Enschede, 2001.

[SB94] M. Spuri and G. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, pages 2–11, San Juan, Portorico, December 1994. IEEE Computer Society Press.

[SB96] M. Spuri and G. Buttazzo. Scheduling Aperiodic tasks in Dynamic Priority Systems. *Journal of Real-Time Systems*, 10(2):179–210, March 1996.

[Spu96] M. Spuri. Analysis of Deadline Scheduled Real-Time Systems. Technical Report 2772, Institut National de Researche en Informatique et Automatique, January 1996.

[SSG97] C. Partridge S. Shenker and R. Guerin. Specification of Guaranteed Quality of Service. RFC 2212, IETF, September 1997.

[SSL89] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Journal of Real-Time Systems*, 1(1):27–60, June 1989.

[SV96] M. Shreedhar and G. Varghese. Efficient Fair Queueing Using Deficit Round-Robin. *IEEE Transactions on Networking*, 4(3):375–385, 1996.

[Tak86] H. Takagi. Analysis of Polling Systems. *The MIT Press Cambridge, Massachusettes, London, England*, 1986.

[Whi83] W. Whitt. The Queueing Network Analyzer. *The Bell System Technical Journal*, 62(9), November 1983.

[Wro97] J. Wroclawski. Specification of the Controlled-Load Network Element Service. RFC 2211, IETF, September 1997.

[XN99] X. Xiao and L.M. Ni. Internet QoS: A Big Picture. *IEEE Network*, 13(2):8–18, March/April 1999.

[YG99] R.D. Yates and D.J. Goodman. *Probability and Stochastic Processes: a Friendly Introduction for Electrical and Computer Engineers*. John Wiley & Sons, New York, NY, 1999.

[ZCKD02] H. Zhu, G. Cao, G. Kesidis, and C. Das. An Adaptive Power-Conserving Service Discipline for Bluetooth. In *Proceedings of IEEE International Conference on Communications ICC2002*, pages 303–307, New York, April 2002.

[Zha91] L. Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks. *ACM Transactions on Computer Systems*, 9(2):101–124, May 1991.

[ZS94] Q. Zheng and K.G. Shin. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks. *IEEE Transactions on Communications*, 42(2/3/4):1096–1105, 1994.

Acronyms

ACL	- Asynchronous connection-less
ACLS	- Adaptive cycle-limited scheduling
ADS	- Asynchronous dedicated slave
AF	- Assured forwarding
AFP	- Adaptive flow-based polling
AM_ADDR	- Active member address
APCB	- Adaptive power-conserving service discipline for Bluetooth
APPI	- Adaptive probability-based polling interval
APS	- Adaptive share polling
ARQ	- Automatic repeat request
BD_ADDR	- Bluetooth device address
BE	- Best effort
BER	- Bit error ratio
BSD	- Berkeley software distribution
CAC	- Channel access code
CB-EDF	- Class-based EDF
CBR	- Constant bit rate
CID	- Channel ID
CL	- Controlled load
CRC	- Cyclic redundancy check
DAC	- Device access code
DIAC	- Dedicated inquiry access code
DRR	- Deficit Round Robin
DS	- Differentiated services
DTMC	- Discrete-time Markov chain
EDC	- Efficient double cycle

EDD	- Earliest due deadline
EDF	- Earliest deadline first
EDL	- Earliest deadline late
EF	- Expedited forwarding
EPM	- Exhaustive pseudo-cyclic master queue length
eSCO	- Extended synchronous connection-oriented
FEC	- Forward error correction
FEP	- Fair exhaustive poller
FPQ	- Fair and efficient polling algorithm with QoS support
FTP	- File transfer protocol
GFSK	- Gaussian frequency shift keying
GIAC	- General inquiry access code
GS	- Guaranteed service
HEC	- Header error check
HOL	- Head-of-line
HOL-KFP	- HOL K-fairness policy
IAC	- Inquiry access code
ID	- Identity
IETF	- Internet engineering task force
IP	- Internet protocol
IrDA	- Infrared data association
ISM	- Industrial scientific medical
ISP	- Intenet service provider
L2CAP	- Logical link control and adaptation protocol
LAP	- Lower address part
LIBT	- Last inter-burst time
LM	- Link manager
LMP	- Link manager protocol
LWRR	- Limited and Weighted Round Robin
MAC	- Media access control

MTU	- Maximum transfer unit
ns2	- Network simulator
PAN	- Personal area network
PDA	- Personal digital assistant
PFP	- Predictive fair poller
PHB	- Per-hop behavior
PN	- Personal network
PSM	- Protocol/service multiplexer
QoS	- Quality of service
QSPI	- Queue status based polling interval
RF	- Radio frequency
RR	- Round robin
RSVP	- Resource reservation protocol
SAR	- Segmentation and reassembly
SCO	- Synchronous connection-oriented
SDP	- Service discovery protocol
SLA	- Service level agreement
TCP	- Transmission control protocol
TCS	- Transmission convergence sublayer
TOS	- Type of service
UMTS	- Universal mobile telecommunications system
WLAN	- Wireless local area network

Index

Samenvatting

De trend om de persoonlijke apparatuur die mensen met zich mee dragen dynamisch met elkaar te verbinden heeft geleid tot de introductie van persoonlijke omgevingsnetwerken en persoonlijke netwerken. Men gaat ervan uit dat de Bluetooth draadloze toegangstechnologie zal kunnen worden gebruikt om dit soort netwerken op te zetten. In dit proefschrift ligt de nadruk op Bluetooth intra-piconet scheduling (ook bekend als Bluetooth polling) die het mogelijk maakt om de Bluetooth technologie succesvol te gebruiken voor het opzetten van de hiervoor genoemde netwerken.

Om dit te verwezenlijken, moet het Bluetooth polling mechanisme efficiënt zijn. Tegelijkertijd, moet het polling mechanisme ook eerlijk zijn. En als laatste, moet het polling mechanisme in staat zijn om ”quality of service” (QoS) te bieden. Dit laatste is nodig voor de ondersteuning van audio en video applicaties. Conventionele polling mechanismen zijn minder geschikt voor Bluetooth daar zij de Bluetooth specificatie niet in acht nemen. Huidige Bluetooth polling mechanismen zijn niet in staat om op een eerlijke en efficiënte manier te pollen of zij bieden de nodige QoS niet.

In dit proefschrift wordt een nieuw polling mechanisme, genaamd ”Predictive Fair Poller” (PFP), ontwikkeld. Dit polling mechanisme voorspelt de beschikbaarheid van data voor elke slave en houdt de eerlijkheid bij. Gebaseerd op deze twee aspecten bepaalt PFP welke slave gepold moet worden zodanig dat efficiëntie en eerlijkheid worden geoptimaliseerd.

Verder worden twee nieuwe polling mechanismen ontwikkeld die in staat zijn om QoS te bieden, namelijk de vaste interval poller en de variabele interval poller. Deze pollers volgen de Guaranteed Service aanpak van de IETF en bieden dus zowel een rate garantie als een delay garantie. Voor Bluetooth polling is deze aanpak nieuw. De vaste interval poller plant polls voor slaves met een vast interval, terwijl de variabele interval poller, wanneer mogelijk, polls voor slaves uitstelt om bandbreedte te besparen. De vaste interval poller en de variabele interval poller bieden, met een vooraf gedefinieerde maximale afwijking, een rate garantie. Gegeven dat de verkeersbronnen zich houden aan hun verkeersspecificatie, leidt deze rate garantie tot een delay garantie. Deze twee typen garanties zijn de belangrijkste QoS typen die nodig zijn voor audio en video applicaties. Verder worden hertransmissie strategieën ontwikkeld die de invloed van slechte radio omgevingen op het bieden van deze QoS typen minimaliseren.

De mechanismen en technieken die gedurende dit onderzoek zijn ontwikkeld worden geëvalueerd middels simulatiestudies. Deze studies tonen aan dat PFP eerlijk en efficiënt is. In het bijzonder tonen deze studies aan dat PFP minstens zo goed presteert als bestaande Bluetooth polling mechanismen. Verder tonen deze studies aan dat de variabele interval poller beter presteert dan de vaste interval poller en dat deze in staat is delay garanties te bieden vergelijkbaar met de delay garanties die door een zogenaamde ”synchronous connection-oriented” (SCO) kanaal kunnen worden geboden. Bovendien is de variabele interval poller in staat deze garanties te bieden en tegelijkertijd minder capaciteit te consumeren. Aangezien de variabele interval poller in staat is hertransmissies uit te voeren, kan de bespaarde capaciteit worden gebruikt om, met behoud van QoS, de link kwaliteit problemen van SCO kanalen te vermijden.