Application Note 703
Embedded Networking with IPv6

www.maxim-ic.com

OVERVIEW

Address space for Internet Protocol (IP) nodes is getting tight. Although not all of the 2^{32} (roughly 4 billion) IPv4 addresses[1] have been allocated yet (and in 2001 there was a slight dip in the previous exponential growth)[2], it is expected that we will run out of addresses in the course of the next few years. The next generation of IP (IPv6) extends the addressing space to 2^{128} (a number far beyond human imagination, about 6.67×10^{23} addresses per square meter of our planet[3]), making sure all of the many future devices can get a unique address of their own.

Having enough addresses eliminates the need for network address translation (NAT)[4], temporary address leases, and other kludges necessary to conserve the strictly rationed IPv4 addresses. Although there will be significantly more desktop and server computers and other classic network devices, a tremendous increase is expected to happen in a different area—lots and lots of small devices will change internetworks as we know them today. The new network citizens are always-on wireless and mobile devices like GPRS and UMTS cell phones or PDAs, and small embedded devices, monitors, sensors, and smart nodes built into almost anything, from cars to water meters.

IPv6 not only extends the addressing space. By overhauling the IP, it makes configuration easy and automatic (another must for embedded applications). It also makes the IP more robust, extensible, and mobile, and adds security features, quality-of-service support, and faster and simpler routing. A severe problem plaguing IPv4 is the unimpeded growth of the backbone routing tables because of the almost random way IPv4 addresses were originally allocated. IPv6 is a better, reengineered IP and it will gradually replace IPv4—there are just too many advantages to pass it by. Dual IPv4/6 network stacks support mixed environments and allow for gradual adoption of IPv6.

IPv6 is becoming increasingly important, not only for its benefits but also because of government-mandated adoption plans in several countries. Asia, especially Japan, was one of the first to adopt IPv6 because that region was somewhat shortchanged when IPv4 addresses were initially assigned. Yet, India and China have the largest expected internetwork growth, both in relative and absolute numbers. IPv6 has left the prototype stadium and is now a standard part of most operating systems, for example Microsoft Windows® XP, Sun Solaris™ 8/9 etc.[5]

The following discussion briefly introduces IPv6 and describes how to use IPv6 networking with the silicon software resident in a DS80C400 microcontroller. Basic network literacy and a working knowledge of IPv4 are assumed.

[1] The current IP is *Version 4 (IPv4)*.
[2] Due to the way IPv4 addresses are allocated, only about 160 million addresses are actually available.
[3] The total Earth surface is approximately 509,917,870 square kilometers.
[4] Or "IP masquerading."
[5] Visit www.ipv6.org for more information.

IPv6 OVERVIEW

Addresses

An important part of IPv6's auto-configurability is the way addresses are used. A 128-bit IPv6 address is divided into a 64-bit prefix (net bits or subnet) and 64 host bits. The prefix, which also shows the scope of an address, is either assigned by the network provider[6] and broadcast by routers or it can be local to the link or site. On an Ethernet, the host bits are usually derived from the device's unique MAC (media access control) address (in the form of IEEE EUI-64). That means an IPv6 node is operational with a valid IP address as soon as it is plugged in. To communicate globally, the node needs to solicit or listen to the router broadcasts containing the prefix and combine the prefix with the EUI-64. Unlike the DHCP addition to IPv4, all IPv6 nodes can be self-configuring, even in the absence of a server.

IPv6 addresses are written in hexadecimal notation in groups of 16 bits, for example `3ffe:aaaa:bbbb:cccc:260:8ff:fe8d:6ee9`, which is an address of global scope. The same machine would have the "link local" address `fe80::260:8ff:fe8d:6ee9`, where `fe80::/64` is the prefix for link-local addresses; `/64` shows the length of the prefix and `::` is short for 0s. The loopback host (`127.0.0.1` in IPv4 parlance) is simply `::1`. Site local addresses have a prefix of `fec0::/10`. Since there is no direct equivalent to site local addresses in IPv4, these addresses are rarely used now.

From a user's view, these long addresses are, of course, normally hidden behind DNS names like `www.maxim-ic.com`. To serve IPv6 addresses, an IPv6-capable DNS server is required[7]. There are no fundamentally new concepts; an IPv6 address entry in the DNS would be created as `example IN AAAA 3ffe:aaaa:bbbb:cccc:260:8ff:fe8d:6ee9` instead of the `IN A` record used for IPv4. Use of DNS is strongly encouraged, since IPv6 address prefixes are expected to change more frequently. Network renumbering is much easier than with IPv4 and it can even be automatic.

There are both unicast and multicast addresses in IPv6. In addition, a new anycast address destination type was defined. A packet addressed to an anycast IP is delivered to the closest or best host out of several hosts. Anycast helps achieve load balancing through routing[8].

Protocols

Although IPv6 keeps the higher-layer protocols UDP and TCP without any changes, the IP packet header of course had to be modified to accommodate the larger addresses. It was also cleaned up and streamlined to be 64-bit aligned and to always have a fixed length for the benefit of routers; the IP header checksum was removed since the higher-layer protocols already have a checksum that encompasses parts of the IP header.

An interesting modification is the replacement of ARP with the neighbor discovery protocol (NDP), part of the new ICMPv6. Instead of broadcasting address resolution requests all over the campus, IPv6 maps multicast groups and IPv6 addresses in a way that eliminates these broadcasts and ensures that nodes (almost) only receive traffic that is really of interest to them.

Unfortunately, the details of ICMPv6 and multicasting are beyond the scope of this article. Visit www.ipv6.org for more information about IPv6 features.

[6] Usually, 48 prefix bits are assigned by the ISP; 16 are at each site's discretion.
[7] For example, BIND9 from http://www.isc.org/products/BIND/.
[8] On IPv4, DNS load balancing (e.g., round robin) is commonly used, which does not take routing issues into consideration.

TCP/IP on the DS80C400

The on-chip DS80C400 silicon software (ROM) contains the latest revision of the field-proven Dallas TCP/IP stack. The silicon software also includes a small operating system and all utility functions needed to develop small C or assembly language TCP/IP network client and server applications with as little as 128kB of external memory. The DS80C400 can also be used with the TINI® Java™ runtime environment when easier and more rapid application development is desired, or when any of the extended Java features like object serialization are required.

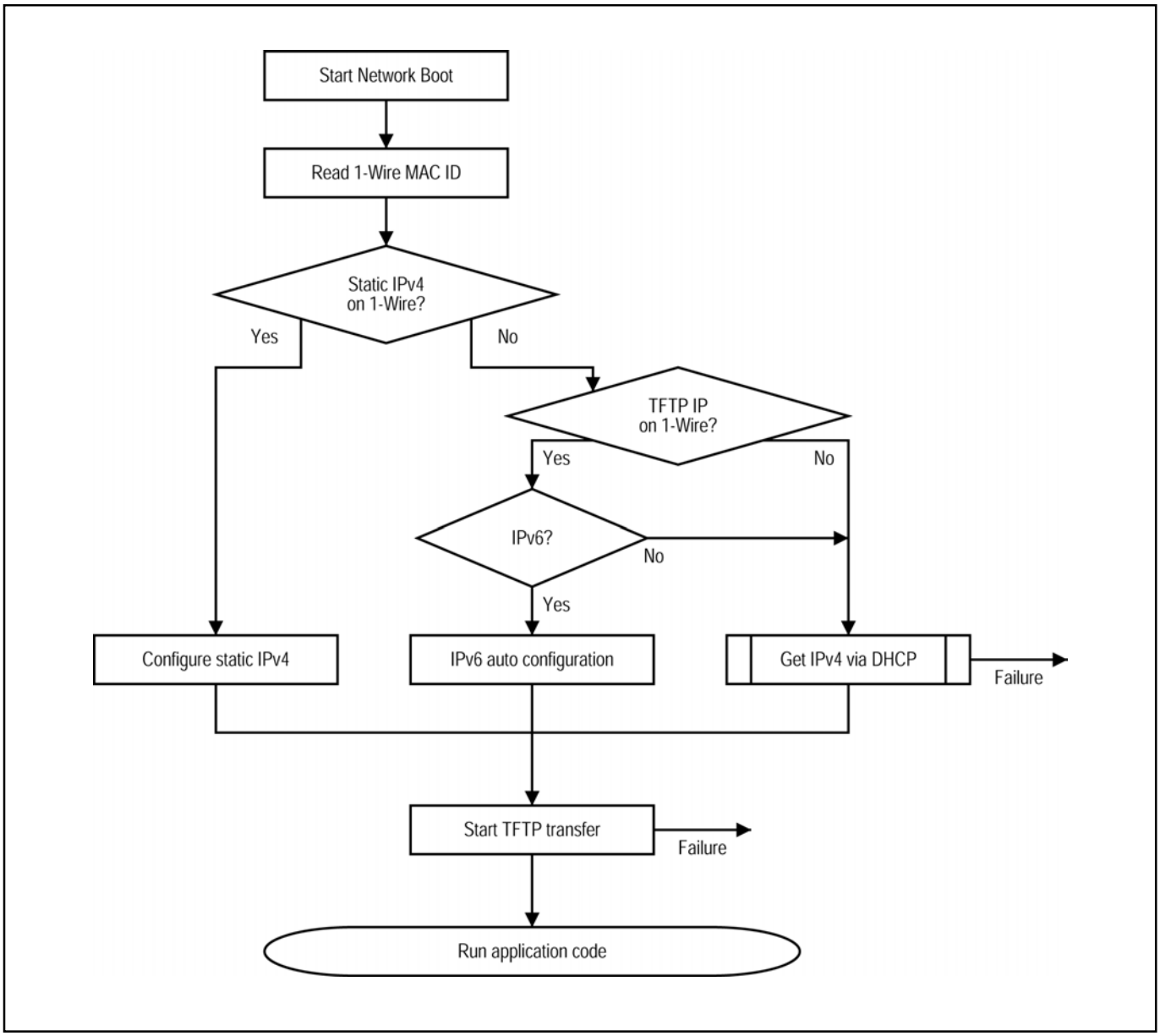
The resident C and assembly language support is implemented in the form of a BSD and industry standard cross-platform socket interface, i.e., functions like `socket()`, `bind()`, `listen()`, `accept()`, `connect()`, `send()` etc.[9].

The TINI Java environment closely follows JDK 1.1.8 and supports the entire `java.net` package; any Java compliant compiler can be used. The TINI executes standard Java programs and byte codes. An overview and detailed documentation for the TINI runtime can be found on the website at www.maxim-ic.com/TINI.

In addition to network application support, the DS80C400 silicon software also implements network boot functionality, which can load applications over TFTP, supporting both DHCP on IPv4 and, even easier, TFTP over self-configuring IPv6. Figure 1 shows the DS80C400 network boot over IPv4 and IPv6, respectively. The network bootloader is invoked either by a hardware pin on the DS80C400 or a user command in the serial bootloader.

[9] These and all other supported functions are documented in the *DS80C400 User's Guide*.

Figure 1. Network Boot



IPv6 on the DS80C400

The DS80C400 silicon software supports the IPv6 features needed[10] to participate on the network and follows the "Minimum Requirements of IPv6 for Low-Cost Network Appliances" draft[11]. Because of their resource-constrained nature, it is not anticipated that embedded devices will implement the full IPv6 feature set including security, mobile IP, and routing.

The adoption of IPv6 will be phased in over several years; the DS80C400 network stack, therefore, is an integrated dual stack for both IPv4 and IPv6. There are ways to tunnel IPv6 over existing IPv4 networks (6over4); since the DS80C400 supports both protocol families, it expects routers to tunnel packets if necessary and does not perform protocol conversions itself.

Example 1 runs on the TINI 1.1 Java environment for the DS80C400 and shows fragments of a simple multithreaded network server handling both IPv4 and IPv6 requests. There is no IPv6 specific code in this example. Applications can usually be ported from IPv4 only to IPv4/6 with zero effort; only the printing of IP addresses has to be checked and possibly replaced by a call to TINI 1.1 utility functions provided for that purpose. The TINI 1.1 Java environment adds the Java 2 SE 1.4 *Inet6Address* class to support IPv6. There were no other user visible changes required, all other changes are behind the scenes.

Example 2 is the core of a network client written in C that solely relies on the DS80C400 silicon software. Again there is no IPv6-specific code, except for the target address. In the DS80C400 network stack implementation, all network addresses are 128 bit long. Internally, IPv4 addresses are right-aligned and the first 96 bits are set to 0. The example assigns an IPv6 target address and a target TCP port, creates a socket and then connects to the target.

[10] The IPv6 portion of the DS80C400 Silicon Software was developed in close collaboration with InternetNode, Inc., a joint venture of the Japanese company Yokogawa and Wide Research Institute Co. Ltd.
[11] See http://www.taca.jp/docs.html

Example 1. TINI Java Network Server

```
// Listen to inbound TCP connections
private class listenTCPThread extends Thread
{
  private ServerSocket serverSock;
  public void run()
  {
    while (running) {
      try {
        // Create new thread for each client
        Thread client = new clientTCPThread(serverSock.accept());
        client.start();
      }
      catch (Exception e) {}
    }
    ...

private class clientTCPThread extends Thread
{
  private Socket sock;
  private InputStream  is; private OutputStream os;
  BufferedReader br;

  public clientTCPThread(Socket s) throws IOException
  {
    sock = s;
    is = s.getInputStream(); os = s.getOutputStream();
    br = new BufferedReader(new InputStreamReader(is));
  }

  public void run()
  {
    // Loop while socket is open
    try {
      while (running) {
       os.write(parseCommand(br.readLine().getBytes(), 0));
      }
    }
    ...
```

Example 2. C Network Client

```
{
  struct sockaddr target;
  unsigned int s;
  ...
  /* Fill sockaddr with valid IPv6 target address and port */
  target.sin_addr[0] = 0x3f;
  target.sin_addr[1] = 0xfe;
  ...
  target.sin_addr[15] = 0xe9;
  target.sin_port = 34000;

  /* Open socket and connect to target address */
  s = socket(0, SOCKET_TYPE_STREAM, 0);
  result = connect(s, &target, sizeof(struct sockaddr));

  ... /* Send/receive data here */

  closesocket(s);
}
```

CONCLUSION

IPv6 provides an unlimited number of IP addresses, auto-configuration, and a general streamlining of the IP protocol. It is thus becoming more important for the success of network-embedded devices. The DS80C400 makes writing an application that supports both IPv4 and IPv6 networking easy. Additionally, the DS80C400's software has all the functions necessary to easily develop small TCP/IP network-client server applications. Together, IPv6 and the DS80C400 offer so many compelling benefits for IP networks, there is every reason to use them for all existing and new applications.