CHAPTER 12

Internet Communication and Control

12.1 Introduction to Internet Communication

Computers and devices connect to the Internet through the Ethernet communications protocol, or through wireless (Wi-Fi) communications. Public data, stored on *servers* as web pages, can be accessed by *client* devices that are connected to the Internet. A client can simply be a home computer that accesses the web pages stored on the server; it could also be an embedded system that accesses and utilizes data stored on the server and responds to control messages sent by the server.

The Internet has transformed worldwide communications over the past 25 years, but rather than being just a network for sharing online information and communicating via email or static web pages, the Internet nowadays facilitates advanced interactive applications. The mbed compiler, for example, is accessed entirely through the Internet, which means that no additional software needs to be installed for a developer to work with the mbed. In this chapter, we look at the Ethernet interface on the mbed and utilize bespoke libraries that allow the mbed to act as a network client or server device.

The Internet has evolved to not only allow people to connect with machines and data, but also for machines and devices to communicate with each other autonomously, i.e., with no (or limited) human interaction. The concept of connecting literally billions of devices and sensors to the Internet is referred to as the *Internet of Things* (IoT), which we shall explore toward the end of this chapter.

12.2 The Ethernet Communication Protocol

12.2.1 Ethernet Overview

Ethernet is a serial protocol which is designed to facilitate network communications, particularly on local area networks (LANs) and wide area networks (WANs), which were introduced in Chapter 11. Any device successfully connected to the Ethernet can potentially communicate with any other device connected to the network. Ethernet communications are defined by the IEEE 802.3 standard (see Ref. [1]) and support fast data rates up to 100 Gbps (Gigabits per second). Ethernet uses differential send (TX) and receive (RX) signals, resulting in 4 wires labeled RX+, RX−, TX+, and TX−.

Ethernet messages are communicated as serial data packets referred to as *frames*. Using frames allows a single message to hold a number of data values including a value defining the length of the data packet as well as the data itself. The Ethernet frame therefore defines its own size. Ethernet communications need to pass a large quantity of data at high rates, so data efficiency is a very important aspect; by defining the data size of each packet, rather than relying on a fixed size for all messages, the number of empty data bytes is minimized. Each frame includes a unique source and destination media access control (MAC) address. The frame is wrapped within a set of *preamble* and *start of frame* (SOF) bytes and a *frame check sequence* (FCS) which enables devices on the network to understand the function of each communicated data element. The standard 802.3 Ethernet frame is constructed as shown in Table 12.1.

The minimum Ethernet frame is 72 bytes; however, the preamble, SOF, and FCS are often discarded once a frame has been successfully received. So a 72-byte message can be reported as having just 60 bytes by some Ethernet communications readers.

The Ethernet data takes the form of the *Manchester encoding* method, which relies on the direction of the edge transition within the timing window, as shown in Fig. 12.1. If the edge transition within the timing frame is high-to-low, the coded bit is a 0; if the transition is low-to-high, then the bit is a 1. The Manchester protocol is very simple to implement in integrated circuit hardware and, as there is always a switch from 0 to 1 or 1 to 0 for every data value, the clock signal is effectively embedded within the data. As shown in Fig. 12.1, even when a stream of zeros (or ones for that matter) is being transmitted, the digital signal still shows transitions between high and low states.

12.2.2 Implementing Simple mbed Ethernet Communications

The mbed Ethernet application programming interface (API) is shown in Table 12.2.

We can set up an mbed Ethernet system to send data and view this on a *logic analyzer* or a fast oscilloscope (you'll need to be able to measure at speeds of around 10 Gbps) to verify the Ethernet frame structure. The logic analyzer is designed to convert an input signal accurately to Logic 0 and Logic 1 data, allowing messages and data streams to be

Table 12.1: Ethernet frame structure.

Preamble	Start of Frame Delimiter	Destination MAC Address	Source MAC Address	Length	Data	Frame Check Sequence	Interframe Gap
7 bytes of 10101010	1 byte of 10101011	6 bytes	6 bytes	2 bytes	46—1500 bytes	4 bytes	

MAC, media access control.

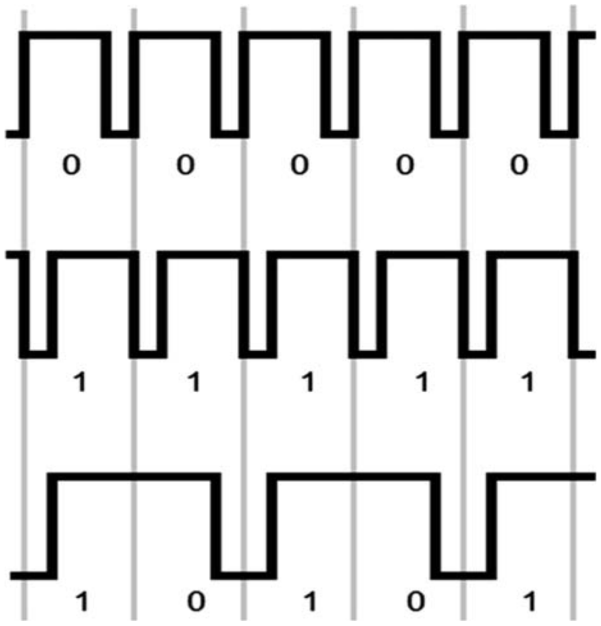


Figure 12.1
Manchester encoding for Ethernet data.

Table 12.2: The mbed Ethernet application programming interface.

Function	Usage
`ethernet`	Create an Ethernet interface
`write`	Writes into an outgoing Ethernet packet
`send`	Send an outgoing Ethernet packet
`receive`	Receives an arrived Ethernet packet
`read`	Read from a received Ethernet packet
`address`	Gives the Ethernet address of the mbed
`link`	Returns the value 1 if an Ethernet link is present and 0 if no link is present
`set_link`	Sets the speed and duplex parameters of an Ethernet link

evaluated. Many logic analyzers also include fast analog oscilloscope features, so it may also be possible to view the raw signal too.

In this example, we are not initially concerned with the specific MAC addresses of sending or receiving devices, we simply want to send some known data and see it appearing on the logic "scope." Program Example 12.1 sends two data bytes every 200 ms from an mbed's Ethernet port. The two byte values are arbitrarily chosen as 0xB9 and 0x46.

```
/* Program Example 12.1: Ethernet write
                                          */
#include "mbed.h"
#include "Ethernet.h"
Ethernet eth;                   // The Ethernet object
char data[]={0xB9,0x46};        // Define the data values
int main() {
    while (1) {
        eth.write(data,0x02);   // Write the package
        eth.send();             // Send the package
        wait(0.2);              // wait 200 ms
    }
}
```

Program Example 12.1: Ethernet write

Figs. 12.2 and 12.3 show details of the data packet transmitted by Program Example 12.1. (To analyze the Ethernet signal, you'll need to connect a *differential* "*scope probe*" to the mbed's Ethernet transmit pins labeled TD+ and TD−). Analog noise can easily be seen on the raw signal, but the logic analyzer accurately converts the Ethernet signal to an idealized digital representation. In Fig. 12.2, the 7-byte preamble and SOF data are identified.

The preamble and SOF delimiter are followed by destination and source MAC addresses, which take up 6 bytes each, and the 2 bytes denoting the data length, which is a minimum of 46 bytes, as described in Table 12.1. Note that even though in this example we have only sent two data bytes, the minimum data size of 46 bytes is sent. The remaining 44 data bytes are made up of empty (0x00) data, being followed by the four FCS bytes. This is not a particularly efficient use of Ethernet, which is usually required to send large data packets. Looking at the end of the data packet, shown in Fig. 12.3, we can see the final zero padded data and the FCS data.

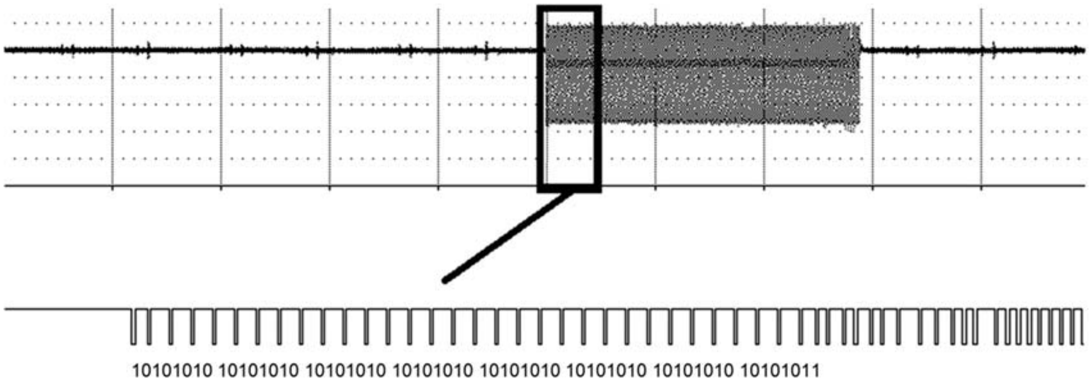


Figure 12.2
Ethernet packet showing preamble and start of frame data.

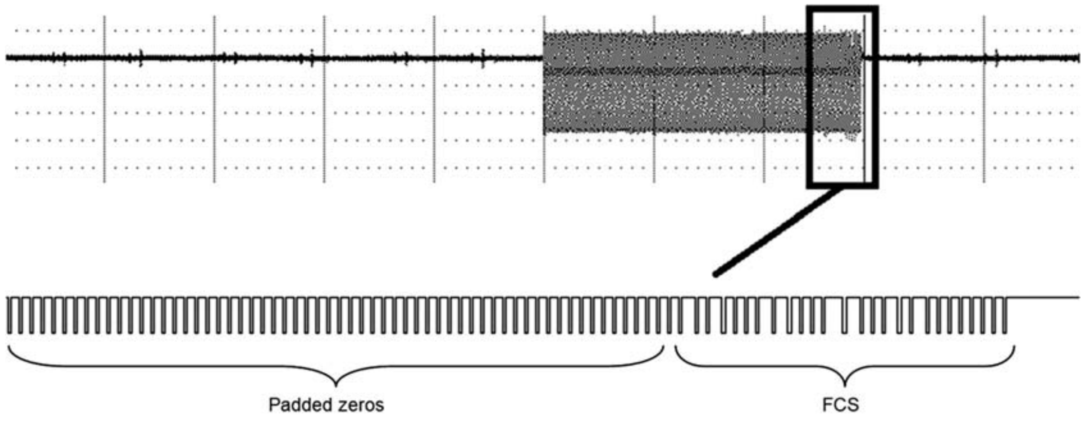


Figure 12.3
Padded Ethernet data and frame check sequence.

Exercise 12.1

Using a logic analyzer or fast oscilloscope, identify the data making up the 0xB9 0x46 data values transmitted in Program Example 12.1. Experiment with different data values and array sizes, ensuring that each time the correct binary data can be observed on the analyzer.

12.2.3 Ethernet Communication Between mbeds

An mbed Ethernet port can be used to read data also. To test this, we can reuse the simple Ethernet write program of Program Example 12.1. We will also need a second mbed system to receive incoming data and display it to the host terminal screen, in order to verify that the correct data is being read.

The following program allows an mbed to read Ethernet data traffic and display the captured data to a host terminal screen (e.g., Tera Term on a Windows PC or CoolTerm on Apple OS X).

```
/* Program Example 12.2: Ethernet read
                                                                      */
#include "mbed.h"
Ethernet eth;                           // Ethernet object
char buf[0xFF];                         // create a large buffer to store data
int main() {
  printf("Ethernet data read and display\n\r");
```

```
    while (1) {
      int size = eth.receive();              // get size of incoming data packet
      if (size > 0) {                        // if packet received
        eth.read(buf, size);                 // read packet to data buffer
        printf("size = %d data = ",size);  // print to screen
        for (int i=0;i<size;i++) {           // loop for each data byte
          pc.printf("%02X ",buf[i]);         // print data to screen
        }
        pc.printf("\n\r");
      }
    }
  }
```

Program Example 12.2: Ethernet read

Program Example 12.2 first defines a large data buffer, as an array labeled **buf**, to store incoming data. During the infinite **while(1)** loop, the program uses the **eth.receive( )** function to determine the size of any Ethernet data packets. If a size greater than zero is reported, then a packet has been received, so the display loop is entered. The size of the data package along with the read data is then displayed to the host terminal. To communicate successfully between the two mbeds, a crossed signal connection is required, as shown in Fig. 12.4 and Table 12.3.

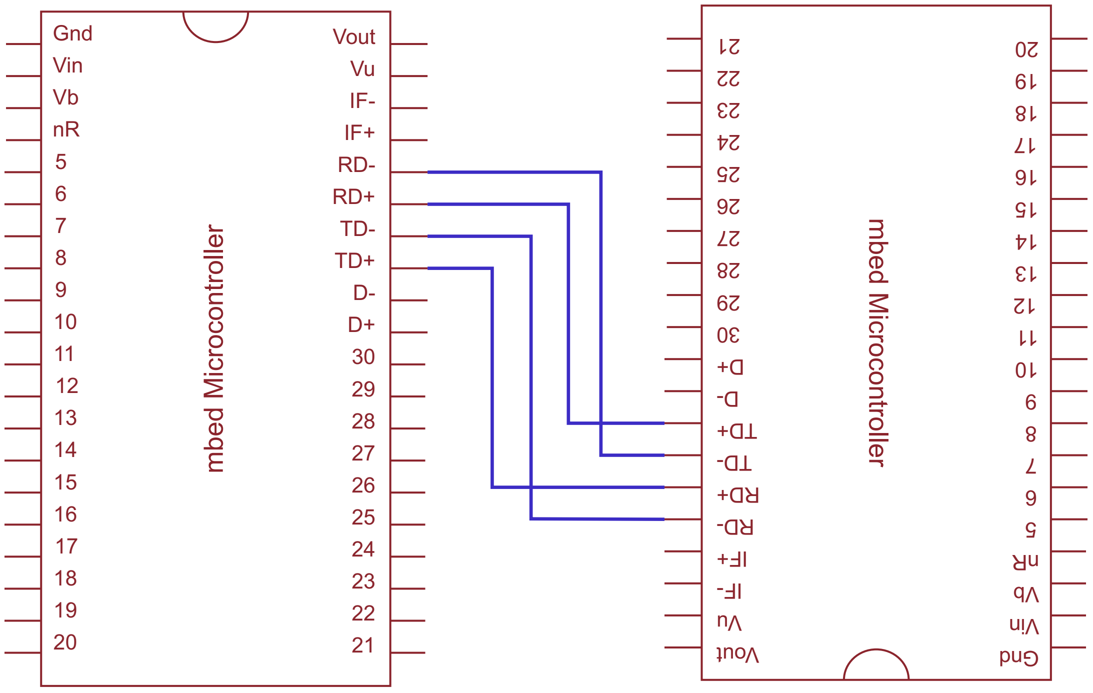
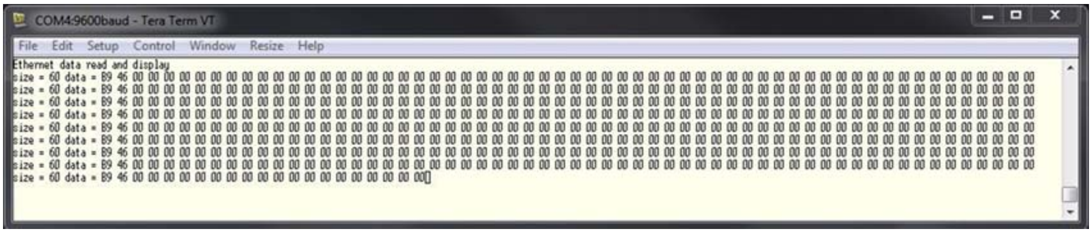


Figure 12.4
mbed-to-mbed Ethernet wiring diagram.

Table 12.3: mbed-to-mbed Ethernet wiring table.

mbed 1	mbed 2
RD−	TD−
RD+	TD+
TD−	RD−
TD+	RD+


COM4:9600baud - Tera Term VT
File Edit Setup Control Window Resize Help

```
Ethernet data read and display
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
size = 60 data = B9 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 12.5
Ethernet data successfully communicated between two mbeds and displayed in Tera Term.

The Ethernet write mbed should run Program Example 12.1, and the data receiving mbed runs 12.2. This is connected to a terminal application running on the host computer. When successfully running, the host PC terminal application should display Ethernet data similar to that shown in Fig. 12.5. It can be seen, as expected, that a 60-byte data package is received, which, as expected, represents the minimum Ethernet data package size (72) minus the preamble, SOF, and FCS bytes.

Exercise 12.2

Experiment with different data values and array sizes, ensuring that each time the correct binary data can be read by the host PC terminal.

12.3 Local Area Network Communications With the mbed

12.3.1 Local Area Network Essentials

With Ethernet connectivity, it is possible to use the mbed as a network-enabled system, for both LAN and WAN communications. When developing Internet (WAN) connected systems, it is often useful to first validate and test designs within a LAN. The LAN is managed by a *router* (or *network hub*) device that processes messages between servers and clients, ensuring that the data is passed on to the intended recipient.

An example LAN with mbed servers and clients is shown in Fig. 12.6. Within the LAN shown, it is possible for the PC or mobile device to remotely access files stored on the

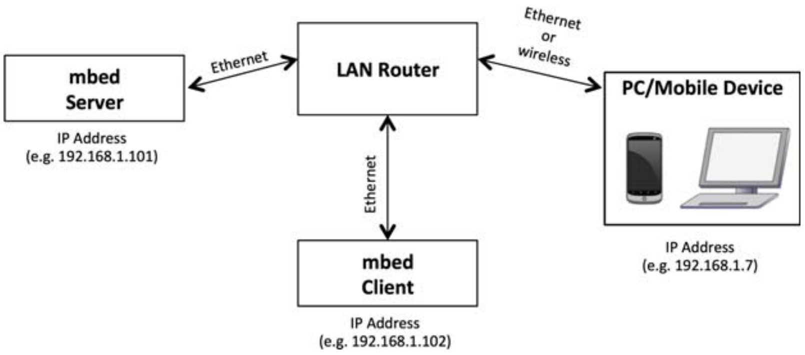

mbed Server
IP Address (e.g. 192.168.1.101)
Ethernet
LAN Router
Ethernet or wireless
PC/Mobile Device
IP Address (e.g. 192.168.1.7)
Ethernet
mbed Client
IP Address (e.g. 192.168.1.102)

Figure 12.6
Example local area network with mbed server and client.

mbed server. Equally, it is possible for the mbed client to access files and data stored on the mbed server, as well as for the mbed server to control the mbed client through remote control protocols which we will explore later in this chapter. The PC client can also request the mbed server to send control messages to the mbed client, hence allowing the PC client to indirectly control the mbed client. This communication can all be implemented as long as the local 32-bit *IP address*—sometimes referred to as the *private IP address*—of each device is known. (The acronym IP stands for *internet protocol* in this context.) The major limitation of the LAN system is that all devices need to be connected to the same router either by a physical Ethernet connection or by a wireless connection that is standard to most LAN routers nowadays. One advantage, however, is that, with a closed LAN featuring no external Internet access, security against hackers and viruses can be more easily managed and controlled.

The 32-bit IP address is usually written as four consecutive 8-bit values separated by decimal points or commas. Predominantly, LANs use a default IP address of 192.168.1.0 (or sometimes 192.168.0.0) and the router on that network will usually take the default address or the next available value above the default address (i.e., 192.168.1.1). The default address or router address is often referred to more formally as the *gateway address* of the LAN. The gateway (or router) is responsible for receiving data messages and forwarding them on (routing) them to the correct devices within the LAN. Additionally, a 32-bit *subnet mask* or *network mask* is often defined, which allows the gateway to divide a LAN network into two or more subnetworks. The network IP address can therefore be split into multiple subnetworks by applying a bitwise AND operation with the subnet mask. The default subnet mask value for a network that does not utilize any subnetworks is 255.255.255.0, which allows up to 255 devices to be routed from a single gateway. This is the value we will always use in this book when a network mask is needed.

It will be seen from examples in this section that it is not always necessary to define the IP addresses of each connected device. Most routers are able to allocate IP addresses to each device on the network using a process called *dynamic host configuration*—defined by the *dynamic host configuration protocol* (DHCP). Using DHCP allows the router, clients, and servers to identify their own private IP addresses without need for a network administrator to set up and configure each connection manually. Nowadays, we all carry mobile Internet devices and many open access Wi-Fi zones exist in public places; DHCP is what allows us to enter new wireless Internet zones, such as a friend's house or an Internet café, and easily be able to join the network with a mobile device, without the need for someone to manually enter any device details or configuration data.

Another important aspect of network communications, which we will explore in the coming sections, is the *hypertext transfer protocol*, commonly referred to as just *HTTP*. HTTP is a fundamental protocol for LAN and WAN communications based on a standard server–client computing model; it is the protocol for communicating hypertext, which is a broad definition for data that represents the design and functionality of web pages. The *Hypertext Markup Language* (HTM or HTML) is a type of hypertext that is regularly used for defining web page content and layout. HTTP clients and servers are therefore specially configured to host and decode hypertext through the HTTP protocol, which make them the fundamental building blocks for sharing data and web pages across LANs and WANs, the Internet, or, as we sometimes call it, the *World Wide Web*.

To implement the examples in this chapter from here onward, you will need to use a standalone router or network hub device that has no security features enabled. For the university lecturer or industrial developer, here is a word of warning; many university and office networks are connected throughout by Ethernet, though unfortunately simply connecting the mbed to an empty socket is unlikely to give you the connectivity you are hoping for. Schools, universities, and offices usually have firewall security settings that mean an mbed server or client—which you will be programming later in this chapter—will most likely be refused a connection. You will need to talk to your IT Network Manager to ensure that access through an infrastructure such as this is possible. Alternatively, you may wish to use a standalone router for LAN applications and stay well away from the network infrastructure of your company or institution. In writing this book, we used a standalone Draytek Vigor model router such as those seen at Ref. [2]; units like this usually ship with a default "plug and play" setup, meaning that no advanced router configuration will be required.

For the home developer or student, most households have a LAN router (often supplied by the Internet Service Provider) for connecting devices and enabling Internet access, so you may be able to use that. Equally, you may prefer to use a separate router to your main home network hub. Many people have a spare one of those lying around since last changing their home Internet supplier!

The router device therefore becomes an important aspect of the examples used, as we will use it to connect mbeds together and, for example, create the LAN setup shown in Fig. 12.6. If you choose to proceed with the network mbed examples in the remainder of this chapter, you will need to be willing to do a little background reading on your particular router and its configuration settings.

12.3.2 Using the mbed for Ethernet Network Communications

A number of bespoke and mbed official libraries exist to facilitate network communications. For example, it is possible to use the mbed official **EthernetInterface** library to connect an mbed to a LAN or WAN. The **EthernetInterface** library is very different to the **Ethernet** library seen in Table 12.2, as it is specifically written for initiating and interfacing LAN and WAN Ethernet network communications, as opposed to simply providing a mechanism for sending and receiving packets of raw data. The **EthernetInterface** API is given in Ref. [3] and is summarized in Table 12.4.

A standard Ethernet socket (RJ45) is required to connect the mbed Ethernet port to a network hub or router. The Sparkfun Ethernet breakout board (detailed in the parts list in Appendix D) is used in this example and can be connected to the mbed as shown in Fig. 12.7. If using the mbed application board, it is simple to use the built-in Ethernet socket.

Table 12.4: EthernetInterface application programming interface.

Function	Usage
`ethernetInterface`	Create an Ethernet Internet interface.
`init()`	Initialize the interface with automatic assigned IP address
`init (const char *ip, const char *mask, const char *gateway)`	Initialize the interface with a static IP address.
`connect`	Connect and open communications
`disconnect ()`	Disconnect Bring the interface down
`getIPAddress ()`	Get the IP address of the Ethernet interface
`getGateway ()`	Get the Gateway address of the Ethernet interface
`getNetworkMask ()`	Get the Network mask of the Ethernet interface

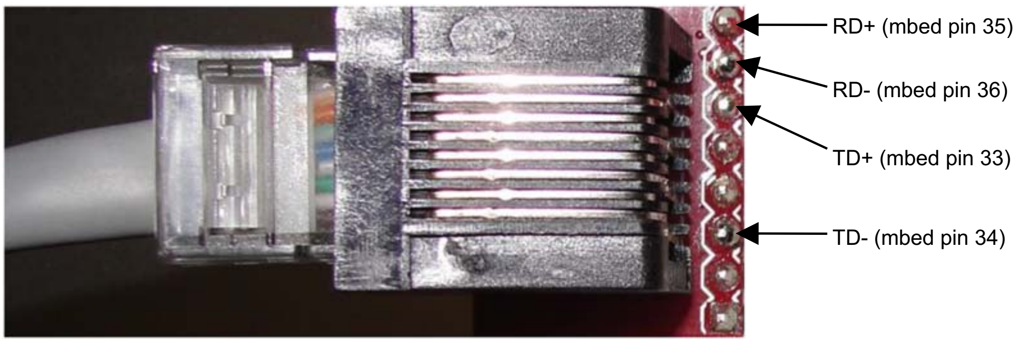


Figure 12.7
RJ45 Ethernet connection for mbed.

Program Example 12.3 creates an Ethernet communications interface and requests the router to automatically assign an IP address to the mbed. The program then prints the assigned IP address to a host terminal.

```
/* Program Example 12.3: Opening an Ethernet network interface
                                                           */
#include "mbed.h"
#include "EthernetInterface.h"
EthernetInterface eth;           // create ethernet interface
int main() {
  eth.init();                    // initialise interface with DCHP
  eth.connect();                 // connect and open communications
  printf("IP Address is %s\n", eth.getIPAddress());  // display IP address
  eth.disconnect();              // disconnect
}
```

Program Example 12.3: Opening an Ethernet network interface

The mbed Real-Time Operating System (or mbed RTOS) was briefly introduced in Section 9.11, here we use it for the first time. The mbed RTOS and its associated **mbed-rtos** library defines functions and classes which are required to run complex and continuous processes such as network communications. For Program Example 12.3 to compile, we therefore need to import the official **mbed-rtos** library, found at Ref. [4]. As described in Section 9.11, the RTOS uses threads (a coding approach sometimes also referred to as *threading* or *multithreading*), which allows the mbed to give the appearance of performing multiple actions at once. So far all of our programs have used single sequential steps of software, however, for network communications we need to allow data transfers to take place continuously in the background (i.e., "in a different thread"), while allowing us to still develop control code that runs in the foreground. It is not the purpose of the book to describe multithreading or the mbed RTOS in detail, though we will see it utilized as an additional library from time to time, particularly where network and Internet communications are concerned.

Exercise 12.3

Create a new program and import the **EthernetInterface** and **mbed-rtos** libraries. You may need to refer back to Section 6.7 where the process of importing libraries is described. Connect the mbed via its Ethernet port to an active network hub or router. Run Program Example 12.3 with a host terminal application open and check that the IP address is identified and output to the screen.

Update the program to also identify and output the interface's gateway address and network mask.

Although the **EthernetInterface init( )** function automatically requests an IP address from the router, it cannot be guaranteed that the assigned IP address will be the same whenever the mbed is powered up, and this can cause problems if another client or server is trying to regularly connect to a device that is not continuously powered on. For this reason, it is often preferable to tell the router directly what value we would like our IP address to be. When we define an IP address directly, this is often referred to as a *static IP address*, as it will always be the same.

Each system on the network needs its own unique IP address based on the default Gateway IP address (usually 192.168.1.1), so if we chose a static IP value of 192.168.1.101, for example, it is unlikely to clash with any other network systems (unless there are more than 100 other systems). Using the **EthernetInterface** API—and using a Network Mask of 255.255.255.0—the following code allows an mbed Ethernet interface to be initialized with a static private IP address:

```
eth.init("192.168.1.101","255.255.255.0","192.168.1.1");
```

Exercise 12.4

Modify the initialization statement in Program Example 12.3 to define a static IP address as shown above. Compile and run the program to verify that the correct IP address is allocated.

12.3.3 Using the mbed as an HTTP File Server

When connected to an Ethernet network, the mbed can be configured to host data files that can be accessed using the HTTP from another PC or device on the LAN. To implement this, we can make use of the bespoke **HTTPServer** library by developer Henry Leinen (available at Ref. [5]). A summary of the **HTTPServer** API is given in Table 12.5.

When using the **HTTPServer** library, it is necessary to specify *request handlers* for the server. Request handlers are specific software definitions that inform the server what types of Ethernet messages to respond to, i.e., requests for the server to do something. As shown in Table 12.5, request handlers are defined by using the **addHandler( )** function and specifying the type of handler that is required; we will see an example of this in Program Example 12.4. The **HTTPServer** library only supports two different types of handler, one of which is a file system handler that uses the handler name **HTTPFsRequestHandler** in program code. We will use the second type of handler, **HTTPRpcRequest Handler**, later in the chapter. Additionally the file system handler needs to be informed where to locate and store files in physical memory; this is managed by defining a **LocalFileSystem** object, as described in Section 10.3.

Once the file server is initiated, it is necessary to inform the server which *transmission control protocol (TCP) port* it should be routed to. Computers and servers generally have many hundreds or thousands of virtual connection ports, which enable simultaneous communication with multiple devices. Many types of TCP port have reserved port numbers by convention; HTTP communications are usually defined to use port 80. In our examples, we will always use port 80 for mbed HTTP server applications. Once the server is started, the **poll( )** function is required to be called regularly, in order to respond quickly to server requests from external clients.

Table 12.5: HTTPServer application programming interface.

Function	Usage
`HTTPServer`	Create an HTTPServer object
`addHandler (const char *path)`	Adds a request handler to the server. Either **HTTPFsRequestHandler** or **HTTPRpcRequestHandler**
`start(int port,*eth)`	Binds the server to a specified port and starts listening
`poll`	Polling of the server to respond to messages

Finally, in setting up an mbed HTTP server, it is necessary to **#include** the **HTTPServer.h** and **FsHandler.h** header files. Program Example 12.4 shows the full code that will run a HTTP file server on the mbed.

```
/* Program Example: 12.4 mbed file server setup
                                                                */
#include "mbed.h"
#include "EthernetInterface.h"
#include "HTTPServer.h"
#include "FsHandler.h"
EthernetInterface eth;           // define Ethernet interface
LocalFileSystem fs("webfs");     // define Local file system
HTTPServer svr;                  // define HHTP server object
int main() {
  eth.init("192.168.1.101","255.255.255.0","192.168.1.1"); // initialise
  eth.connect();                              // connect Ethernet
  HTTPFsRequestHandler::mount("/webfs/", "/"); // mount file server handler
  svr.addHandler<HTTPFsRequestHandler>("/");   // add handler to server object
  svr.start(80, &eth);                   // bind server to port 80
  while(1)
  {
    svr.poll();          // continuously poll for Ethernet messages to server
  }
}
```

Program Example 12.4 mbed file server setup

In order to compile Program Example 12.4 the **HTTPServer** library should be imported from Ref. [5], along with the mbed official **EthernetInterface** and **mbed-rtos** libraries, used previously. Additionally, Program Example 12.4 requires the mbed official **mbed-rpc** library to be imported, as the **HTTPServer** library depends on it. The **mbed-rpc** library can be found at Ref. [6] and will be discussed in more detail in the next section.

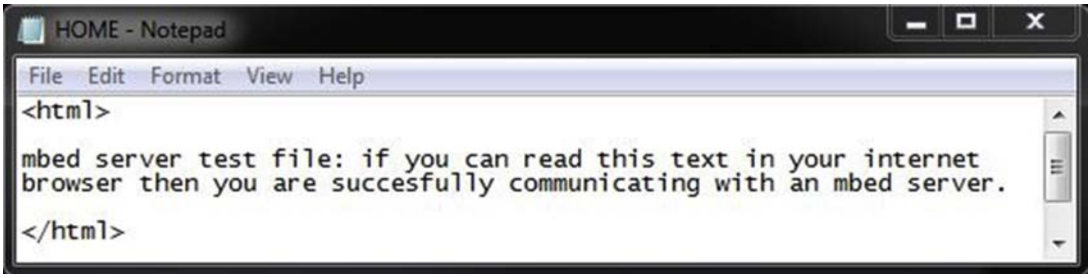
In Program Example 12.4, we see a new C++ feature that uses two consecutive colons in the line of code that mounts the HTTP file server request handler:

```
HTTPFsRequestHandler::mount("/webfs/", "/");
```

C code feature

This code syntax is necessary as it is a fundamental feature of implementing threading in C++ and is therefore a programming approach used by the **mbed-rtos** library, so you will see the **::** operator used from time to time when defining RTOS features.

It is also necessary to create a text file to save onto the mbed, so that we can access it from the remote Internet browser. Using a standard text editor, create an HTML file (which requires a **.htm** filename extension) called, for example, **HOME.HTM**. Enter some example text in the file, so that it is obvious when the Internet browser has correctly



```
<html>

mbed server test file: if you can read this text in your internet
browser then you are succesfully communicating with an mbed server.

</html>
```

Figure 12.8
HOME.HTM file to be stored on the mbed server and remotely accessed.

accessed the file. A simple HTML example is shown in Fig. 12.8. Create this file and save it to the mbed via the standard USB cable connection. Note that the mbed server libraries only support MS-DOS 8.3 type filenames, which means that file names must be of no more than eight characters.

It is now possible to access the **HOME.HTM** file stored on the mbed from any other computer or client device that is successfully connected to the router that manages the LAN. To do this, open a web browser application (such as Firefox, Chrome, or Internet Explorer) on a connected PC, and type in the following navigation address:

http://192.168.1.101/HOME.HTM

Successful navigation to this network address should bring up the HTML text as contained in the **HOME.HTM** file.

Exercise 12.5

Implement Program Example 12.4 and verify that the **HOME.HTM** file can be accessed on the mbed server from a client browser on a PC or any other device that is connected to the LAN router.

When the server starts and whenever server requests are made and managed, the **EthernetInterface** API automatically sends a number of lines of information to a host terminal application, if one is connected. Connect a terminal application and explore the status code that is sent back to the host.

12.4 Using Remote Procedure Calls With the mbed

Remote Procedure Calls (RPCs) are used to enable an action or feature on an Internet-linked device or computer to be controlled from a remote location. Essentially, RPC messages allow functions and variables within one computer to be actioned or manipulated from a remote computer that may be connected through a network or the Internet.

12.4.1 Controlling mbed Outputs With Remote Procedure Calls

As an RPC example for mbed developers, calls can be made from a LAN or WAN connected PC to switch an mbed's LEDs on and off. Indeed, many of the mbed's outputs can be controlled and accessed by RPC messaging from another network connected PC.

The key steps for implementing RPC control, within a program running on an mbed HTTP Server, are:

1. Start a new program and import the **HTTPServer**, **EthernetInterface**, **mbed-rtos**, and **mbed-rpc** libraries.
2. **#include** the **mbed.h**, **EthernetInterface.h**, **HTTPServer.h**, RpcHandler.h and **mbed_rpc.h** header files to the main.cpp program
3. Define mbed interfaces in the *extended RPC format* with the "name" defined within. (Note that mbed interfaces for use with RPC have the letters "Rpc" attached at their start in the object definition, such as **RpcDigitalOut** and **RpcPwmOut**). For example:

```
RpcDigitalOut led1(LED1, "led1");
RpcPwmOut pwm1(p21, "pwm1");
```

4. Make the required interfaces available over RPC by adding the RPC class command, for example:

```
RPC::add_rpc_class<RpcDigitalOut>();
RPC::add_rpc_class<RpcPwmOut>();
```

5. Initiate and connect an mbed Ethernet interface with the LAN router.
6. Add the RPC request handler, for example:

```
svr.addHandler<HTTPRpcRequestHandler>("/rpc");
```

7. Start the mbed server and poll regularly.
8. Manipulate mbed interfaces remotely by using the following browser address format: http://<mbed-ip-address>/rpc/<Object name>/<Method name> <Value>

Applying the required RPC features described above, we arrive at Program Example 12.5. This enables remote control of a **RpcDigitalOut** object assigned to the onboard **LED1**:

```
/* Program Example 12.5 Remote Procedure Calls example
                                                                                    */
#include "mbed.h"
#include "EthernetInterface.h"
#include "HTTPServer.h"
#include "mbed_rpc.h"
#include "RpcHandler.h"
RpcDigitalOut led1(LED1,"led1");      // define RPC digital output object
EthernetInterface eth;                // define Ethernet interface
HTTPServer svr;                       // define HHTP server object
int main() {
  RPC::add_rpc_class<RpcDigitalOut>();
  eth.init("192.168.1.101","255.255.255.0","192.168.1.1"); // initialise
  eth.connect();                                           // connect Ethernet
  svr.addHandler<HTTPRpcRequestHandler>("/rpc");          // add RPC handler
  svr.start(80, &eth);                                  // bind server to port 80
  while(1){
    svr.poll();     // continuously poll for Ethernet messages to server
  }
}
```

Program Example 12.5: Remote procedure calls example

In Program Example 12.5, we set up the mbed as a HTTP server with RPC capability and configured the mbed's digital outputs to be controllable through RPC. In this example, we have defined the mbed's LED1 with an object name **led1**. The value of the **led1** object can now be changed via a web browser on a PC that is also connected to the LAN. This is done by entering into a web browser the RPCs shown in Table 12.6.

Table 12.6: Commands for remote procedure call control of mbed LED1.

Action	Remote Procedure Call
Remotely switch led1 ON	http://192.168.1.101/rpc/led1/write 1
Remotely switch led1 OFF	http://192.168.1.101/rpc/led1/write 0

■ Exercise 12.6

Implement Program Example 12.5 and verify that RPC can be used to control the mbed LED1 from a client browser.

Now implement a similar program to enable manipulation of a pulse width modulation duty cycle by RPC command. You will need to define a PWM object with the extended format, for example:

```
RpcPwmOut pulse(p21, "pulse");
```

You will also need to include the PWM RPC base command as follows:

```
RPC::add_rpc_class<RpcPwmOut>();
```

Set the PWM period at the start of the program and check that the duty cycle, as observed on an oscilloscope, can be manipulated remotely.

Connect the PWM output to a servo motor and show that the servo motor position can be controlled by RPC commands.

■

12.4.2 Using Remote Procedure Call Variables

So far we have used RPC to directly manipulate mbed pins and outputs (LEDs and PWM outputs). However, it is more convenient to exchange data that refers to variable values within the mbed code itself. This approach allows the mbed to receive data from a remote source and to act depending on that data, using its own control algorithm to decide when and which outputs should be modified. For this, it is necessary and possible to create bespoke *RPC Variables* within the mbed program.

To implement and manipulate RPC Variables, we need to create a new program and import the **HTTPServer**, **EthernetInterface**, **mbed-rtos**, and **mbed-rpc** libraries, as before. RPC Variables are created in the variable declaration section of a program, for example:

```
int RemoteVarPercent=0;
RPCVariable<int> RPC_RemoteVarPercent(&RemoteVarPercent,"RemoteVarPercent");
```

The above example defines an integer called **RemoteVarPercent** and an RPC Variable (called **RPC_RemoteVarPercent**), which is defined as a pointer to the integer's memory address. Having defined the RPC Variable in this way, it is possible to use similar RPC control commands as those shown in Table 12.6 in order to manipulate the value of the percentage variable. Program Example 12.6 defines a new RPC Variable and uses this within the main program code to set the percentage duty cycle of a PWM output.

```
/* Program Example 12.6 Using RPC variables for remote mbed control
                                                                       */
#include "mbed.h"
#include "EthernetInterface.h"
#include "HTTPServer.h"
#include "mbed_rpc.h"
#include "RpcHandler.h"
PwmOut led1(LED1);              // define standard PWM output object
EthernetInterface eth;          // define Ethernet interface
HTTPServer svr;                 // define HTTP server object
int RemoteVarPercent=0;
RPCVariable<int> RPC_RemoteVarPercent(&RemoteVarPercent,"RemoteVarPercent");
int main() {
  RPC::add_rpc_class<RpcPwmOut>();
  eth.init("192.168.1.101","255.255.255.0","192.168.1.1"); // initialise
  eth.connect();                                             // connect Ethernet
  svr.addHandler<HTTPRpcRequestHandler>("/rpc");            // add RPC handler
  svr.start(80, &eth);                                 // bind server to port 80
  while(1){
    svr.poll();    // continuously poll for Ethernet messages to server
    led1=float(RemoteVarPercent)/100;   // convert to fraction
  }
}
```

Program Example 12.6 Using remote procedure call variables for remote mbed control

Program Example12.6 allows the user to send a percentage value (i.e., an integer value between 0 and 100) to the mbed via an RPC message. The integer is converted to a fraction within the mbed program and used to set the duty cycle of the PWM output on **LED1**. For example, the following RPC command will set the LED to a PWM duty cycle of 0.5:

```
http://192.186.1.101/rpc/RemoteVarPercent/write 50
```

Having created the mbed program to receive RPC commands, it is also possible to embed these commands within a *graphical user interface* (GUI) built into a web page or mobile application, for example, so that the message doesn't have to be expressly written into a web browser, but can instead be actioned on the press of a button. Building GUIs is outside the scope of this book, but if you have knowledge and skills of web design or mobile application development, it should be quite simple to build interactive applications that control the mbed through buttons, sliders, and other graphical controls. Additional details and examples using RPCs with the mbed can be found in Ref. [7].

Exercise 12.7

Implement Program Example 12.6 and verify that the brightness of **LED1** can be modified by sending RPC percentage values from a network PC to the mbed.

Modify the program to output the PWM signal on pin 21, rather than **LED1**. Add a second RPC variable called **RemoteVarFrequency** and program it such that the PWM frequency can be also modified by RPC commands. Connect an oscilloscope to pin 21 and verify that both the PWM frequency and duty cycle can be controlled over RPC.

12.5 Using the mbed With Wide Area Networks

A more widely distributed system is a WAN that connects devices through the Internet, bringing huge opportunities for remote access and control. Fig. 12.9 shows the same devices as in Fig. 12.6, except configured in an example WAN. It can be seen in the figure that each device is connected to its own LAN, with each LAN connected together through the Internet. Now it is possible for the PC or mobile device to access the data on the mbed server from anywhere in the world as long as it is connected to the Internet (through a LAN or directly through the mobile 4G network, for example). Equally, the two mbed devices can communicate with each other from anywhere in the world as long as they are both connected to the Internet.

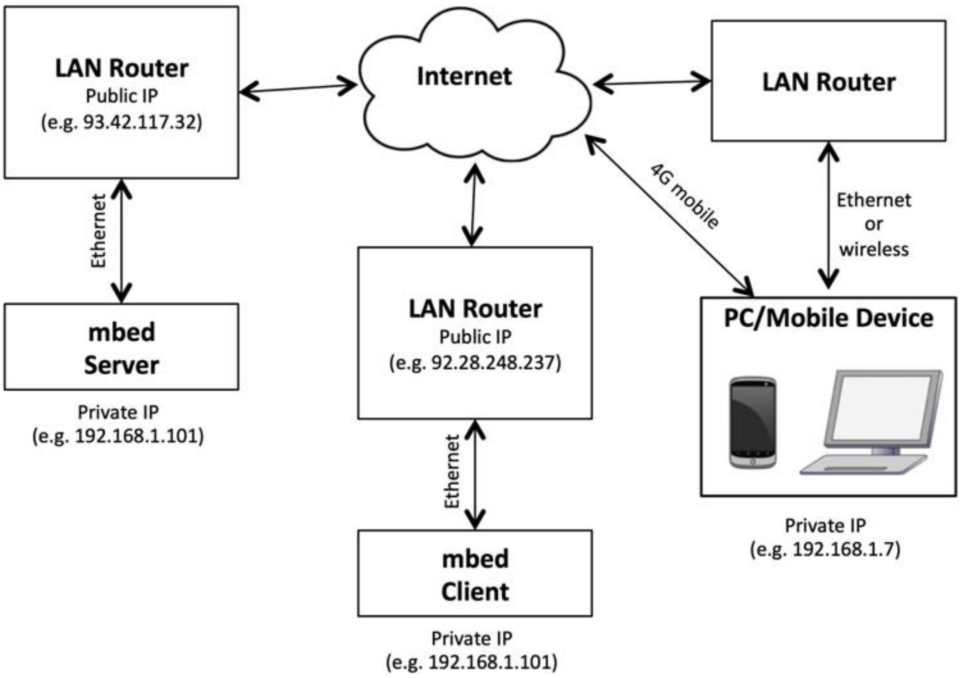


Figure 12.9
Example wide area network with mbed server and client.

The examples described in Section 12.3 are tested specifically on a LAN. However, it is possible to test and verify the functionality of these programs in a WAN setup too. The main obstacle to overcome is to understand the *public IP address* of each LAN. All routers and LANs on the Internet are assigned a unique public IP address, which allows connectivity from anywhere in the world, as long as the unique public IP is known. For any device connected to the Internet it is possible to deduce the public IP address by visiting a website such as Ref. [8].

Each LAN has a unique public IP address, and each device within the LAN has a unique private IP address, but each device in a LAN will still report the same public address. For this reason, routers usually have a *port-forwarding* or *virtual server* setup feature, which allows configuration to define which public messages should be sent to which private devices in a LAN. For example, with respect to Fig. 12.9, it is possible to specify that messages sent to port 80 of public IP address 92.28.248.237 are routed to the device with the private IP address 192.168.1.101, whereas messages sent to port 81 of 92.28.248.237 are routed to device IP 192.168.1.102.

It is possible to use the mbed as a HTTP client in order to access data from the Internet. To do this, we rely on another network interface library called **HTTPClient** by developer Donatien Garnier (Ref. [9]). Program Example 12.7 enables the mbed to connect as an HTTP client to a remote online server and access a text string from within a text file held on the server. The test file we use in this example is stored online at

http://www.rt60.co.uk/mbed/mbedclienttest.txt

You can verify it exists by accessing it through a standard Internet browser.

```
/* Program Example 12.7: mbed HTTP client test
                                                         */
#include "mbed.h"
#include "EthernetInterface.h"
#include "HTTPClient.h"
EthernetInterface eth;
HTTPClient http;
char str[128];
int main() {
  eth.init("192,168,1,101","255,255,255,255","192,168,1,1");
  eth.connect();
  printf("Fetching page data...\n");
  http.get("http://www.rt60.co.uk/mbed/mbedclienttest.txt", str, 128);
  printf("Result: %s\n", str);
  eth.disconnect();
}
```

Program Example 12.7: mbed HTTP client test

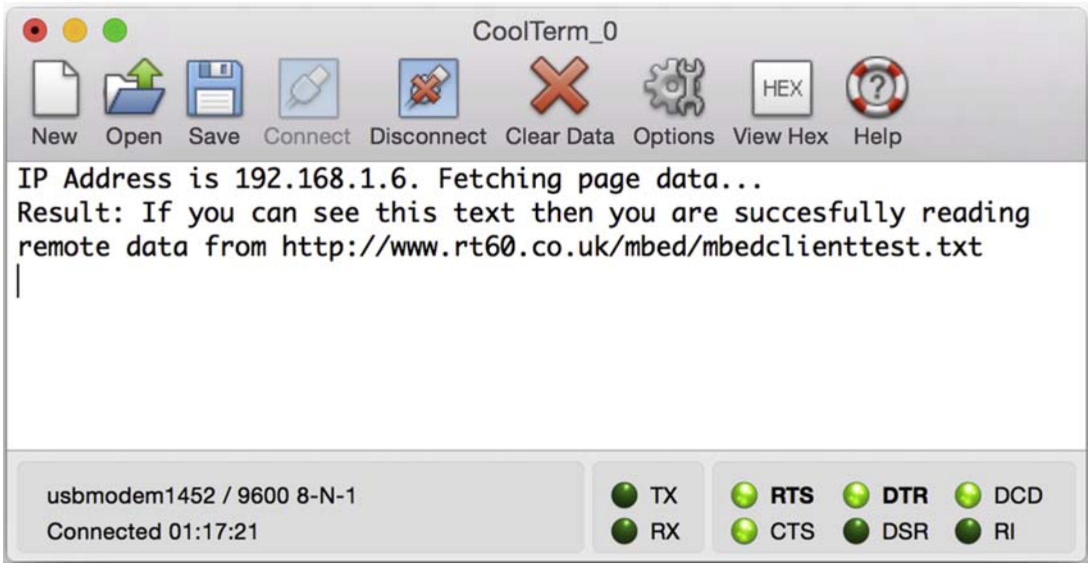
One of the key lines in Program Example 12.7 uses the **HTTPClient** library's **get( )** function, which enables the mbed to retrieve a string of data from a designated website or Internet location. When the mbed is connected to the specified Internet site, the messages shown in Fig. 12.10 should be displayed on a host PC terminal application.

Exercise 12.8

Implement Program Example 12.7 and verify that the mbed HTTP client is able to access the test file stored on a remote Internet server.

If you have access to your own web hosting account, then upload your own text file to the World Wide Web and verify that it too can be accessed from an mbed HTTP client.

Looking back at the RPC examples in Section 12.4, it can be concluded that if the public IP address is known, and correct port-forwarding is employed, it is possible to set up an



```
IP Address is 192.168.1.6. Fetching page data...
Result: If you can see this text then you are succesfully reading
remote data from http://www.rt60.co.uk/mbed/mbedclienttest.txt
```

Figure 12.10
Cool Term displays the output of Program Example 12.7.

mbed with RPC variables and control them from anywhere in the world through the Internet, not just within a LAN.

Exercise 12.9

Implement and test Program Example 12.6 in a LAN and then extend the configuration to verify that the RPC messages can be sent and received from outside the LAN too—i.e., through the Internet. You will need to identify your LAN's public IP address by accessing the website at Ref. [8] and configure your router to forward all messages on port 80 to the private IP address 192.168.1.101. To configure your router you will have to refer to its user manual, as this can be implemented differently for various router makes and models.

You should now be able to connect to another LAN or the 4G mobile network and send the following messages to switch LED1 on and off:

LED1 on: http://xxx.xxx.xxx.xxx:80/rpc/RemoteVarPercent/write 100

LED1 off: http://xxx.xxx.xxx.xxx:80/rpc/RemoteVarPercent/write 0

(where xxx.xxx.xxx.xxx refers to the LAN's public IP address, for example, 92.28.248.237)

12.6 The Internet of Things

12.6.1 The Internet of Things Concept

We have already seen in this chapter that it is possible to connect mbed devices to the Internet and to use mbed devices as both clients and servers. It is therefore possible to make data that the mbed gathers accessible to the outside world, as well as enabling mbed systems to be controlled remotely through the Internet. What we have defined previously in this chapter are the fundamental building blocks for a global *Internet of Things* (IoT). The IoT is a phrase that has evolved to refer to a worldwide network of everyday objects and sensors that are connected to the Internet, allowing remote access to information that can be used to control and enhance everyday activities, while interacting with the mobile Internet devices which people are now carrying regularly, for example, smartphones and tablets. Initially, IoT referred predominantly to global logistical systems, which allow advanced stock control and real-time international tracking of delivery items. The IoT concept revolves around huge quantities of real-time data that can be accessed from anywhere in the world. It is not just physical goods tracking data either, IoT data now includes billions of sensors and databases that are made available for monitoring through the Internet, from travel and transport data, to environmental data, sports results and status data of mechanical devices.

The IoT concept includes reference to the term *cloud computing*, which utilizes servers and data memory storage locations that are only accessed through the Internet. The mbed compiler is itself a simple example of cloud computing, because the mbed programs are stored "in the *cloud*." In reality, the cloud, in this instance, is a data server that is held and managed by ARM, though to the user it is simply an online service that just works as long as a reliable Internet connection is available. Companies such as Google, Microsoft, Apple, and Dropbox all provide their own cloud-based services too and one day it is anticipated that most, if not all, of our personal and professional data (and programs) will be stored in the cloud rather than on local computers and hard drives. The cloud computing approach to mbed development means that it is no longer the user's responsibility to back up data and developers can work on projects from any location in the world without the need to carry local copies of files and programs. This also helps with version control as there is only ever one version of a specific project, so there is less risk of someone working on an outdated version. Developers can of course download and archive their programs locally too.

More recently the IoT concept has expanded to include everyday objects attached to the Internet. Examples include the washing machine that can alert the repair man to an impending fault, the vending machine that can tell the Head Office it is empty, the manufacturer who can download a new version of firmware to an installed burglar alarm, or the home owner who can switch on the oven from the office or check that the garage

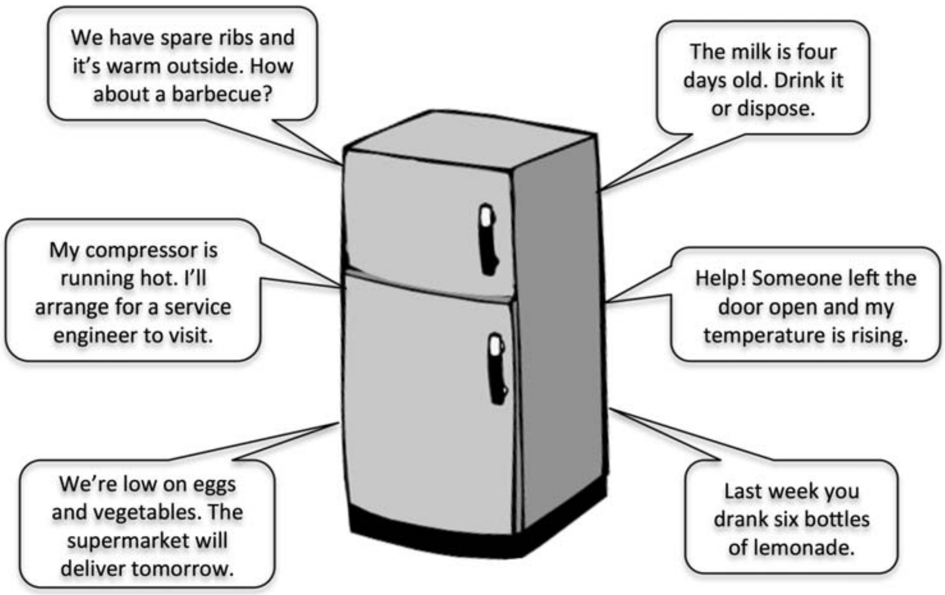

We have spare ribs and it's warm outside. How about a barbecue?
The milk is four days old. Drink it or dispose.
My compressor is running hot. I'll arrange for a service engineer to visit.
Help! Someone left the door open and my temperature is rising.
We're low on eggs and vegetables. The supermarket will deliver tomorrow.
Last week you drank six bottles of lemonade.

Figure 12.11
An Internet connected fridge communicating with its owner.

door is closed. Fig. 12.11 shows an example Internet-connected fridge and highlights some of its unique features that make it stand out from a standard fridge.

The IoT fridge includes intelligent sensors that can read the bar codes and identifying features of products inside, automatically searching the Internet for the necessary product information to identify its own stock levels and track usage. It is *context aware* in that it knows the owner's eating and shopping habits and the state of the weather outside. It is intelligent and can automatically order the shopping from an online grocery store and arrange an engineer visit if it is faulty. It can analyze its own (and its owner's) activity and report statistics and data that may be valuable to the owner, such as monitoring a person's fat or sugar consumption throughout a designated time period. While the IoT fridge is a good example of the functional capabilities of IoT, it doesn't particularly resolve or change any big issues in our society or culture. However, it is anticipated that IoT systems will become ubiquitous over the next 20 years and will change our lives forever, much in the same way that the birth of the World Wide Web did in the 1990s; especially if development systems are made open access (i.e., at no cost) to software developers and network users. The ARM mbed is poised to have a significant role in that revolution.

The potential anticipated power of IoT can be realized when Internet "things" start to interact automatically and intelligently with other Internet "things." Some things such as computers and smartphones can connect directly to the Internet through Ethernet or

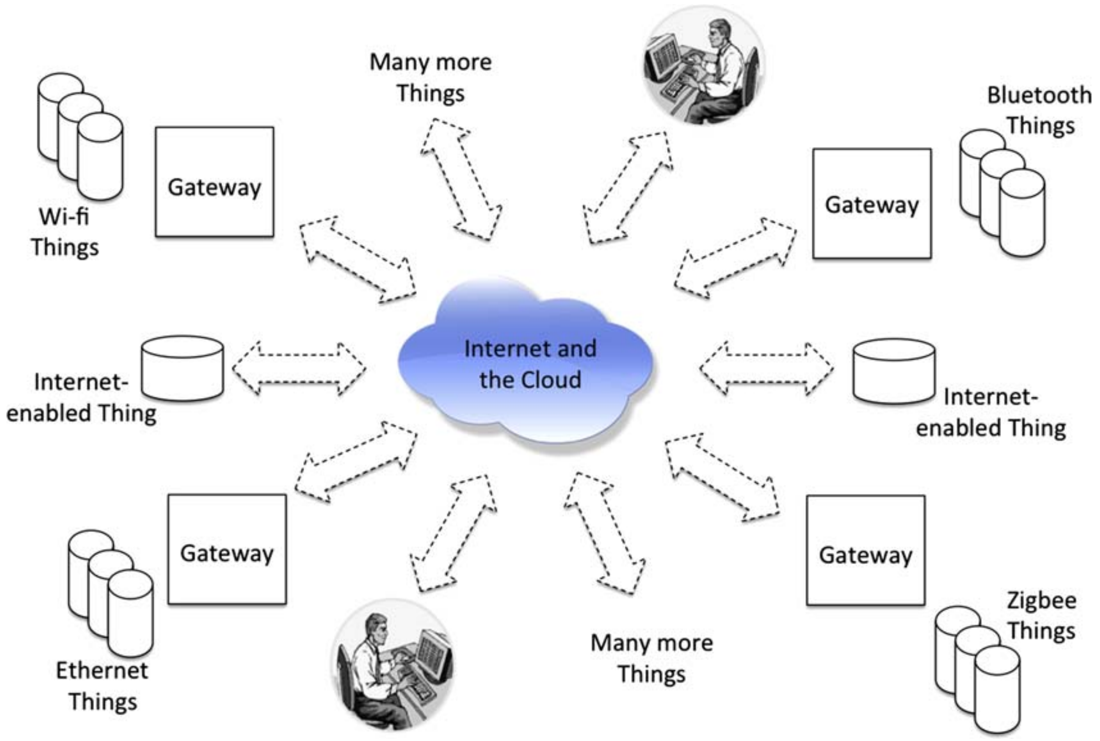


Figure 12.12
The global Internet of Things.

standard mobile communications protocols, whereas other devices will connect to the Internet through a local gateway or router, as shown in Fig. 12.12. Indeed, the gateway could even be a smartphone itself that communicates with other sensors and devices in close proximity. For example, electronic textiles and personal healthcare devices are becoming more widely available and it is not difficult to imagine a person who wears clothes (including wristbands and shoes) that can measure and record data about the person's location, heart rate, body temperature, posture, perspiration levels, exercise routines, and potentially even their emotional state. This data can be measured by small, low-energy devices that are built into the clothes we wear and transmit information over Bluetooth, or another wireless communications protocol, to the user's Internet-connected smartphone or watch. The data is continuously sent to the cloud and can be accessed by the user and other people (or devices) that the data owner wishes to share their data with, such as family members, healthcare organizations, or home automation systems. The mbed is clearly a flexible and capable platform that can be incorporated into designs for both "things" and "gateways" insofar as it is capable of connecting directly to the Internet itself via Ethernet, or, for smaller portable devices, it is capable of connecting with an IoT gateway through Bluetooth or wireless communication.

The IoT concept extends further to represent smart cities, which are made up of houses and communities using home automation and smart transport systems. The smart home has a number of IoT devices—not only the IoT fridge—but also Internet-connected heating systems, lighting, and automatic windows that can be controlled through mobile devices, Internet-connected security systems, and smart renewable energy systems. The owner of a smart home will have a heating system that comes on automatically when a cold weather front is approaching; it can intelligently control lighting to save energy as people move around the house; it can report the status on who has entered and left the building and allow the locks to be controlled remotely; it can even tell you if you left the iron switched on. Smart transport systems have real-time knowledge of traffic movement and congestion, working intelligently with in-car sensors to advise the driver of routes or issues, or the closing time of the hardware store they are approaching. IoT transport systems will be even further enhanced when autonomous vehicles become a commercial reality, enabling vehicles to drive themselves safely and accurately with minimal human interaction.

12.6.2 Opportunities and Challenges for Internet of Things Systems

As seen from the examples given in the previous section, there are many benefits of IoT-enabled systems, while they are also bringing engineering challenges for their efficient and safe implementation. Table 12.7 gives a summary of opportunities and challenges associated with IoT systems.

12.6.3 mbed and the Internet of Things

In the previous network examples, we have always configured the mbed as a HTTP server that manages its own data traffic. The mbed needs to continuously poll (i.e., continuously check) for messages and a browser needs to regularly poll for changes in mbed status data in return. This setup causes demand on the mbed, particularly if data is to be sent and received at the same time, if data is sent rapidly and continuously, and if data connections are initiated by more than one client at once. As a result, the RPC and HTTP server examples shown before are functional, but are prone to fail when overloaded. The IoT concept relies on continuous reliability and fast data transfer rates, so it is clear that a different solution is required for more advanced IoT applications with the mbed. One method is to use a dedicated and robust IoT server that is connected to the Internet and handles all messaging traffic between clients, as shown in Fig. 12.13. This is a particularly advantageous method for using the mbed as an IoT device, as all mbed devices can be configured as clients and avoid the complex processing responsibilities of the server. The server in this setup can be built with substantial processing power and housed in a remote industrial location, where it can be as large and powerful as is necessary to manage all the clients that may wish to communicate with it.

Table 12.7: IoT opportunities and challenges.

IoT Opportunity	IoT Challenge
Remote Control of Hardware Systems enables devices that would normally be controlled by physical switches and potentiometers to be controlled remotely through graphical user interfaces.	**Security** of IoT devices is a major concern. When devices are connected to the Internet, it is feasibly possible for unauthorized persons to access the data and to interfere with control systems.
Real-Time Status Monitoring allows any IoT-enabled device to be monitored remotely and historical charts of performance and functionality to be gathered.	**Reliability** is a challenge because continuous Internet connectivity is required, yet it can be sporadic and unreliable in some locations.
Intelligent Connected Functionality can be realized by the sharing of data between devices; meaning the status of one device can inform the action of another device.	**Robustness** of IoT products is a challenge because they are potentially exposed to more unexpected use, environmental conditions, and accidents.
Remote Diagnostic Analysis allows users and manufacturers to constantly monitor the performance of their products after sale.	**Online Services** are required to enable IoT customers to register their products and access data associated with those products, bringing additional development costs.
Remote Firmware Updates allow functionality to be enhanced or modified remotely, enabling errors to be corrected or new features to be included.	**Graphical Interface Development** skills will be required for organizations to create bespoke user interfaces and mobile applications for accessing and controlling IoT devices.
User Data Gathering enables product developers to evaluate how customers use their products and use those statistics for future product development.	**Bespoke Designs** mean that potentially many different approaches to IoT design will be implemented over the coming years, making some devices compatible with each other and others unable to communicate.
Improved Consumer Experiences through graphical interfaces, plug-and-play products, and improved customer support and education features.	**Standardization and Legislation** is a significant challenge—when billions of devices are connected to the Internet, who will ensure they are all performing correctly and ethically?

IoT, Internet of Things.

In Fig. 12.13, both the mbed client and a mobile or PC-based browser connect directly to the IoT server. Data from the mbed is gathered by the server and sent to the browser to be displayed in real time. Similarly, control messages from the browser or mobile device are managed by the server and forwarded on to the mbed client. This way high-precision control data can be sent to the mbed without the possibility of data overloading, which brings a vast performance and reliability improvement over the WAN arrangement shown in Fig. 12.9. Equally, additional mbed clients could be configured to communicate with each other through the server.

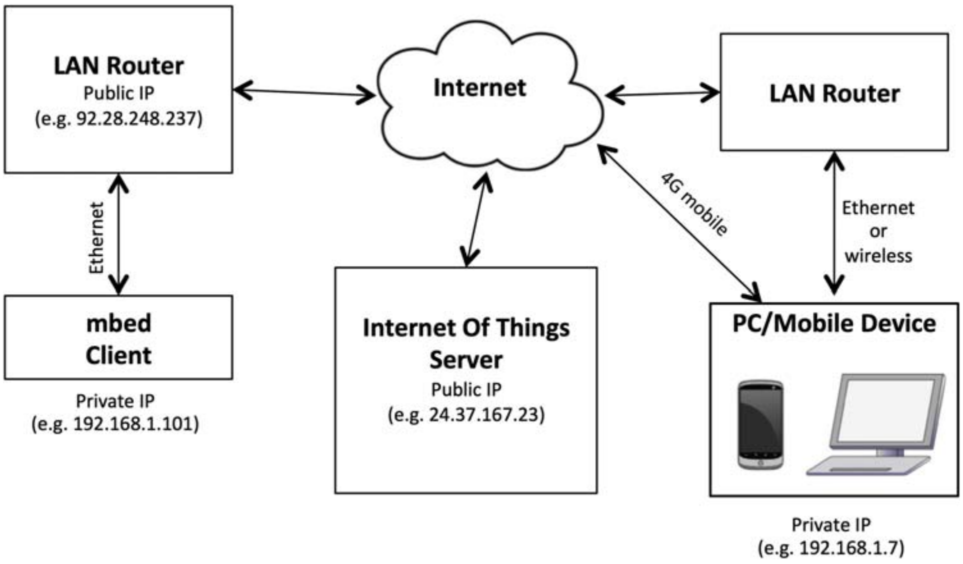
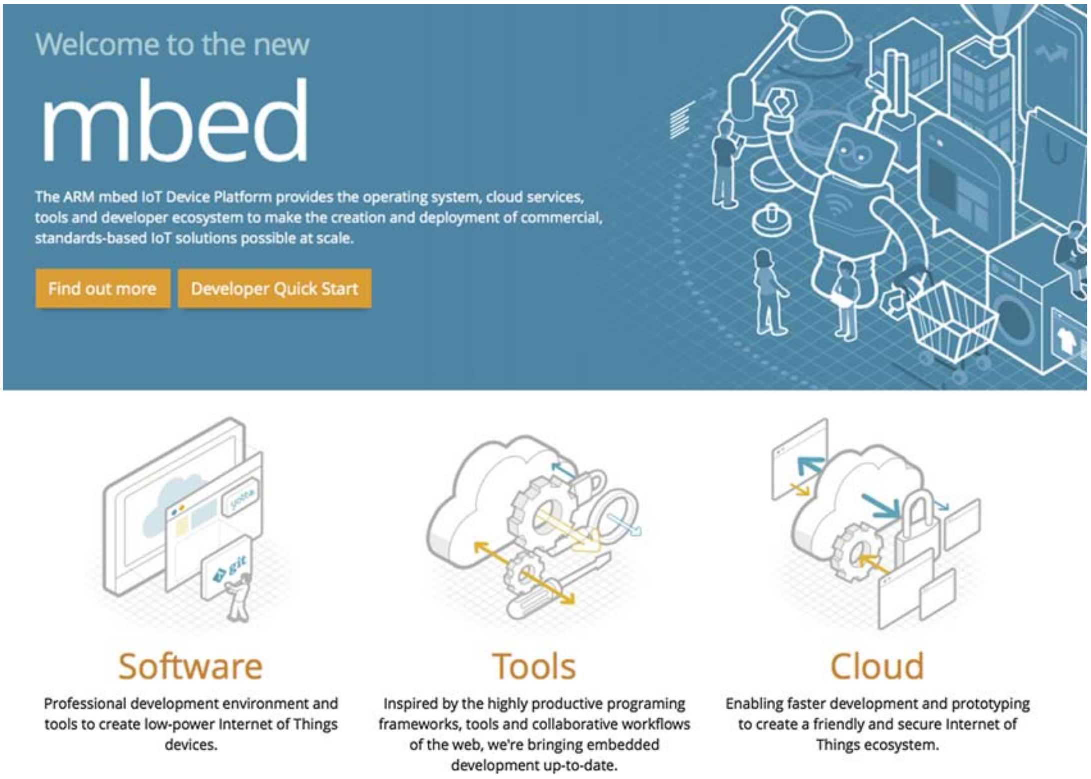


Figure 12.13
Internet of Things design example—a dedicated server and mbed client.

ARM have implemented this application of mbed IoT by using the open-source HTML5 *WebSockets* protocol, which allows bidirectional communications between a server and client devices or browsers (see Ref. [10]). WebSockets allow client devices to send messages to a dedicated server and receive event-driven responses without having to poll the server continuously for a reply. As a result, dynamic data can be exchanged over WebSocket connections that are always on, simultaneously bidirectional and enabling high-speed data transfers. The steps for implementing a WebSocket application with the mbed are outside the scope of the book, and require additional knowledge of network and server configuration, however, some practical examples and additional information using mbed with WebSockets are described on the mbed website at Refs. [11–14].

The WebSocket approach for implementing mbed IoT projects is still susceptible to security breaches, which can cause catastrophic issues. It is important that would-be criminals cannot hijack IoT communications, access confidential information and data, or take control of physical devices that are IoT enabled. One solution is to ensure that all data communication packets over the IoT network are encrypted with advanced algorithms that mean only the intended recipients can access and utilize the contained information. Unfortunately, this needs to be handled by both the server and the client for encryption and decryption, meaning that every IoT device has to be programmed with encryption software within. Given that most embedded systems developers are not encryption experts, this causes a significant headache and is a barrier for innovation around IoT.

Moving forward, ARM have chosen the mbed to be their flagship IoT development platform for the future. They intend to use the mbed RTOS as a low-level software


Welcome to the new

mbed

The ARM mbed IoT Device Platform provides the operating system, cloud services, tools and developer ecosystem to make the creation and deployment of commercial, standards-based IoT solutions possible at scale.

Software

Professional development environment and tools to create low-power Internet of Things devices.

Tools

Inspired by the highly productive programing frameworks, tools and collaborative workflows of the web, we're bringing embedded development up-to-date.

Cloud

Enabling faster development and prototyping to create a friendly and secure Internet of Things ecosystem.

Figure 12.14
The ARM mbed.com web page (at the time of writing).

platform that can manage all encryption and decryption tasks without the programmer needing to become an expert. The mbed RTOS therefore adds this to its list of multithreading responsibilities. In the coming years, they are planning to launch new suites of mbed IoT development frameworks, tools, and services. At the time of writing, these tools are not fully available, so it is not possible to explore them in this edition. The commercial mbed website (www.mbed.com) —the front page of which is shown in Fig. 12.14—explains how they foresee the future of the mbed for IoT applications, ensuring that the platform will have a significant role to play in the "world of connected everything."

Chapter Review

- Ethernet is a high-speed serial protocol which facilitates networked systems and communications between computers, either within a local network or through the World Wide Web.

- LANs and WANS use Ethernet and wireless communications to connect Internet servers and clients through routers, allowing web pages, data and control messages to be accessed between devices.
- The mbed can communicate through Ethernet to access data from files stored on a data server computer.
- The mbed can be configured to act as an Ethernet file server itself, allowing data stored on the mbed to be accessed through a network.
- The mbed RPC interface and libraries allow mbed variables and outputs to be manipulated from an external client through the Internet.
- WebSockets allow robust data communications between a server and mbed clients, meaning that mbed devices can programmed to be controlled remotely through the Internet by accessing web pages and mobile applications.
- The IoT refers to the concept of everyday objects and devices being connected to the Internet, allowing them to be controlled and analyzed from anywhere in the world.
- IoT concepts bring many advantages with respect to massively connected systems, local or global transfer of data, and intelligent systems on a grand scale. IoT does, however, bring challenges with respect to real-time data, security, and reliability.

Quiz

1. Describe the "Manchester" digital communication format for Ethernet signals.
2. Sketch the following Ethernet data streams, as they would appear on an analog oscilloscope, labeling all points of interest:
   a. 0000
   b. 0101
   c. 1110
3. What are the minimum and maximum Ethernet data packet sizes, in bytes?
4. What does the terms "Gateway Address" and "Network Mask" refer to?
5. What are the differences between a public IP address and a private IP address?
6. What does the term RPC refer to? Give a brief explanation of the use of RPC in embedded systems.
7. What does the concept of the Internet of Things (IoT) refer to? Give two examples of IoT devices.
8. Name three opportunities and three challenges associated with Internet of Things systems.
9. Describe three IoT systems that might be found in a "smart home" and explain the benefits that each system brings to the household.
10. Draw a wide area network diagram of an mbed IoT application that serves three remote mbed devices and can be accessed by a mobile device or Internet connected PC.

References

[1] IEEE 802.3 Ethernet Working Group. https://standards.ieee.org/develop/wg/WG802.3.html.
[2] Draytek Products. https://www.draytek.co.uk/products.
[3] EthernetInterface library by mbed official. https://developer.mbed.org/users/mbed_official/code/EthernetInterface/.
[4] mbed-rtos library by mbed official. https://developer.mbed.org/users/mbed_official/code/mbed-rtos/.
[5] HTTPServer library by Henry Leinen. https://developer.mbed.org/users/leihen/code/HTTPServer/.
[6] mbed-rpc library by mbed-official. https://developer.mbed.org/teams/mbed/code/mbed-rpc/.
[7] Interfacing Using RPC. http://mbed.org/cookbook/Interfacing-Using-RPC.
[8] What Is My IP Address. http://www.WhatIsMyIPAddress.com.
[9] HTTPClient library by Doantien Garnier. https://developer.mbed.org/users/donatien/code/HTTPClient/.
[10] HTML5 WebSocket: A Quantum Leap in Scalability for the Web. http://www.websocket.org/quantum.html.
[11] Internet of Things. https://developer.mbed.org/cookbook/IOT.
[12] Internet of Things Demonstration. https://developer.mbed.org/cookbook/Internet-of-Things-Demonstration.
[13] Introduction to WebSockets. https://developer.mbed.org/cookbook/Websockets.
[14] WebSocket Tutorial. https://developer.mbed.org/cookbook/Websockets-Server.