CHAPTER FOURTEEN

Serial Protocol Basics

Whenever there is a need for information to be passed between two computer systems, a communication scheme must be developed to support it. This scheme must include clear definitions of the physical connection, e.g., the wires and voltage levels, the pattern of signals used to synchronize and define the data, methods of encryption and error checking, and so on. Entire books are written on the system of layers that define these communication schemes. This chapter focuses on a single aspect: the pattern of bits used to send data one bit at a time across a single wire.

14.1 OSI Seven-Layer Network Model

The Open Systems Interconnection (OSI) model of networks is a method for classifying the complex components of a communication scheme so that they are easier to describe and understand. There are seven layers to this model ranging from the physical implementation (the wires and such) to the more abstract layer of the applications where the details of the network are hidden. The following is a description of the seven layers:

- ***Application Layer (layer 7)*** – At this layer, one application must format the data so that a second application can properly interpret it. Examples include formatting data so that an e-mail agent or a web browser can properly display data to the user.
- ***Presentation Layer (layer 6)*** – This layer eliminates compatibility problems caused by serving different types of networks. It acts as a generic interface between the application and the details of the network.
- ***Session Layer (layer 5)*** – This layer starts, maintains, and stops the logical connection between two applications.
- ***Transport Layer (layer 4)*** – When a block of data is sent from one application to another, mechanisms are needed to provide error checking, the partitioning of large blocks of data into smaller packets, the reassembly of the data packets, and flow control. The transport layer is responsible for these functions.

- ***Network Layer (layer 3)*** – Once a packet from the transport layer is placed on the network, it needs to find its way from one physical host to another. This is the responsibility of the network layer. It provides routing, forwarding, sequencing, and logical addressing. Since the delivery of a packet may involve several types of networks, this layer must remain independent of the physical implementation of the network.
- ***Datalink Layer (layer 2)*** – The datalink layer handles the bit-level specifications used to deliver a message within a specific type of network. Examples of datalink protocols include the IEEE 802.3 protocol which defines how the bits are organized in an Ethernet frame and the Serial Line Internet Protocol (SLIP) which may be used on a dial-up network. While the network layer uses logical addresses to identify hosts, the datalink layer uses the physical addresses of the hardware.
- ***Physical Layer (layer 1)*** – This layer defines how the bits are transmitted. For example, logic ones and zeros may be identified by different voltage levels, the presence or absence of light, or the frequency of a radio signal. This hardware level description of the network is considered the physical layer.

Since most protocols are described with respect to the layer on which they function, it is important to have a basic understanding of the OSI layer definitions. This chapter discusses three different protocols, one from the datalink layer, one from the network layer, and one from the transport layer.

14.2 Serial versus Parallel Data Transmission

As discussed in Chapter 12, the processor communicates to devices on the motherboard across a group of parallel wires called the bus. A data element can be transferred in an instant by placing each individual bit on a different data line or wire. The data must be held on the lines for a moment so that the inputs of the receiving device stabilize enough for a successful read. This duration, however, is independent of the number of bits of the data element.

There are a few problems with this method of communication. First, wires cost money. At the level of the motherboard, this makes the ICs more expensive due to the greater number of pins and it makes the circuit board larger and more expensive due to the extra room needed for the extra wires. At the system level, i.e., for signals going outside of the computer, each additional data line requires an additional wire in the connecting cable. This results in:

- higher cost for the cable;
- higher cost for the connectors;
- physically larger connectors making miniaturization difficult; and
- decreased reliability of the system.

An increase in the number of connectors is the primary reason for the decrease in reliability of a parallel communication scheme. This is because most failures occur at the connectors. Reliability is also reduced because of a phenomenon known as ***crosstalk***. All wires act like antennas capable of transmitting and receiving electronic signals. If two wires run close together for long distances, they have a tendency to blend their signals, each transmitting their own data and receiving their neighbor's transmitted data.

Another problem with parallel communication schemes that specify the purpose of each wire in the connection is a lack of flexibility. System characteristics such as the number of bits in a data element or the number of devices that can be attached to the bus cannot be modified in a system that uses parallel communication. If, for example, a system has twenty address lines and sixteen data lines, it is restricted to a memory space of 1 Meg and can only transmit unsigned integer values from 0 to 65,535.

The alternative to transmitting data over parallel wires is to send data serially, i.e., one bit at a time, across a single wire. This allows for smaller cables, smaller connectors, fewer pins on ICs, and a more flexible data format. It comes at the expense, however, of speed. If a single wire is used to transmit control, address, and data, then a period of time is used to specify a bit rather than an individual wire. For example, a parallel bus with twenty address lines and sixteen data lines could be converted to a serial connection that took 20 + 16 = 36 "bit periods" to send the same amount of data. Note that this calculation does not take into consideration the control information that would be needed.

The next section presents the basic format of a serial data transmission. It shows how control, addressing, and data information can be sent using a single bit stream.

14.3 Anatomy of a Frame or Packet

A computer system needs a number of pieces of information in order to transmit data from one device to another. This information includes:

- ***Addressing information*** – If more than one device is present on a network, then addressing information for both the sender and receiver is required.
- ***Control information*** – In order to maintain the serial connection, control signals are needed for things such as synchronization (timing), message type, data length, protocol control information such as sequence numbers, and error checking information.
- ***Data*** – While the purpose of some messages can be summed up with the message type, the majority of transmissions are used to send blocks of data.

Serial communication schemes use a single stream of data to send all of this information, each bit's purpose defined by its position with respect to the beginning of the stream. Since a bit's position in the stream of data defines its purpose, strict timing and signal definitions are needed. These definitions, called a ***protocol***, describe a basic unit of communication between serial devices. At the datalink layer, this unit is called a ***frame***. It is called a ***packet*** at higher levels.

In general, a datalink frame is divided into three parts: the ***preamble***, the ***packet***, and the ***trailer***. The preamble is a recognizable bit pattern that is used to represent the start of a frame and provide timing and synchronization for the receiver's circuitry. The timing is necessary because of slight variances in the clock signals of different devices.

The packet of the frame is subsequently divided into two parts: the ***header*** and the data. The header may contain information such as:

- addressing for both the sender and receiver;
- packet numbering;
- message length; and
- message type.

The data may be either the raw data sent from one device to another or it may be an encapsulation of another packet. The datalink layer is not concerned with the pattern of bits within the data portion of the packet. This is to be interpreted at a higher layer in the network model.

The frame's only purpose is to get data successfully from one device to another within a network.

By embedding packets from upper layers of the OSI network model within packets or frames from lower layers, the implementations of the different layers can be swapped as needed. As long as a packet from the network layer gets from one logical address to another, it doesn't matter whether it was carried on a datalink layer implemented with Ethernet, dial-up, or carrier pigeon for that matter. The practice of embedding packets and frames is called a ***protocol stack***.

There are a number of examples of a packet being encapsulated within the packet of another frame. Once again, this is the result of the implementation of the layers of the OSI network model. For example, a typical way to send a web page from a web server to a client is to begin by partitioning the file into smaller packets that are ordered, verified, and acknowledged using the ***transmission control protocol*** (TCP). These TCP packets are then encapsulated into network layer packets used to transport the message from one host to another. A common protocol for this is the ***internet protocol*** (IP). The network layer packets must then be encapsulated into packets used to transfer the data across a specific network such as Ethernet. This places the TCP/IP packets inside the packet of an Ethernet frame. Figure 14-1 shows how the TCP packet is embedded within the IP packet which is in turn embedded into the Ethernet frame.

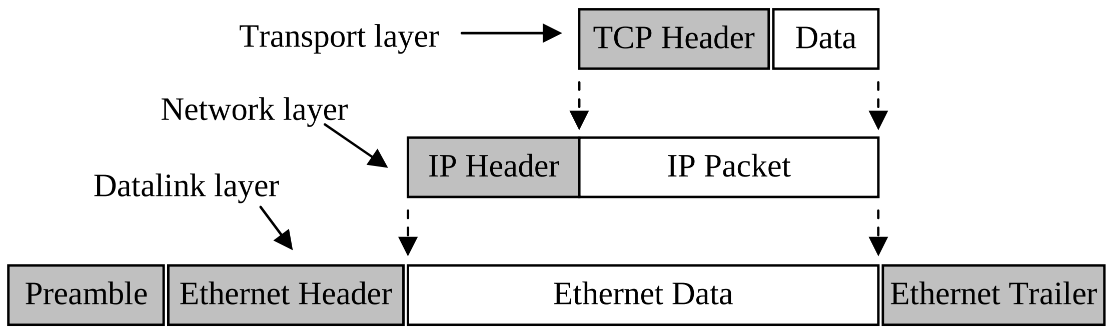

Figure 14-1 Sample Protocol Stack using TCP, IP, and Ethernet

The final part of the frame, the trailer, usually serves two purposes. The first is to provide error detection to verify that a frame has not been corrupted during transmission. Typically, this is a CRC using a predefined polynomial. (See Chapter 9) A second purpose of the trailer may be to include a special bit sequence defining the end of the frame.

14.4 Sample Protocol: IEEE 802.3 Ethernet

One commonly used datalink layer protocol is IEEE 802.3 Ethernet. This protocol is typically sent over a physical layer consisting of special grades of copper wire twisted in pairs to reduce crosstalk. A device on an Ethernet network can see all traffic placed on the network. Therefore, special mechanisms are needed in order to uniquely identify devices (hosts) and to handle messages that are lost when multiple devices attempt to send frames at the same time.

Theoretically, there is an upper limit of over 281 trillion devices on an Ethernet network. Of course this limit is not practical, but it is an indication of the level of addressing used by an Ethernet frame. A single Ethernet frame can transmit up to 1500 bytes, the integrity of which is verified using a 32-bit CRC checksum found in the frame's trailer.

Figure 14-2 presents the basic layout of an Ethernet frame. (Source: IEEE Computer Society. IEEE Std 802.3™-2002. Available on-line at http://standards.ieee.org/getieee802/download/802.3-2002.pdf)

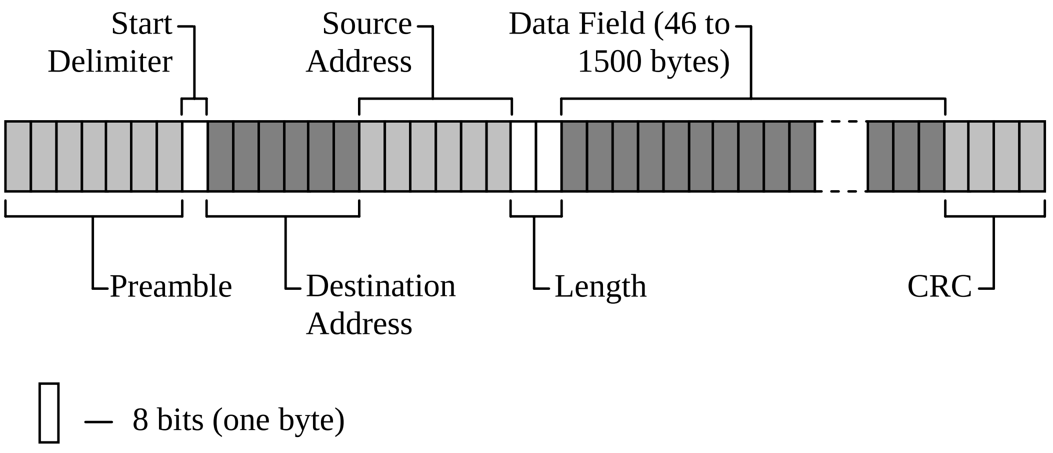

Figure 14-2 Layout of an IEEE 802.3 Ethernet Frame

The frame starts with the preamble, the start delimiter, the destination and source addresses, and the length. The preamble and start delimiter are used to tell the receiving devices when the message is going to start and to provide synchronization for the receive circuitry.

The preamble is 7 bytes (56 bits) of alternating ones and zeros starting with a one. This bit pattern creates a "square wave" which acts like a clock ensuring that all of the receiving devices are synchronized and will read the bits of the frame at the same point in each bit time.

The preamble is immediately followed by a single byte equal to 10101011_2 called the start delimiter. The first seven bits of the start delimiter continue the square wave pattern set up by the preamble. The last bit, which follows a one in the bit sequence, is also equal to a one. This pair of ones indicates to the receiving devices that the next portion of the frame, i.e., the destination address, will start at the next bit time.

The source and destination addresses come next in the frame. These are each 6 bytes long and they identify the hardware involved in the message transaction. It is important not to confuse these addresses with IP addresses which are assigned to a computer by a network administrator. The Ethernet addresses are hardwired into the physical hardware of the network interface card (NIC) and are unique to each device. They are referred to as a Medium Access Control (MAC) addresses and are loaded into the NIC by the manufacturer. They can not be modified. The first three bytes of the MAC address identify the manufacturer. If the destination address is all ones, then the message is meant to be a broadcast and all devices are to receive it.

The next field in the frame is the 2-byte length field. The value in this field represents the number of data bytes in the data field. With two bytes, it is possible to represent a value from 0 to 65,535. The definition of the IEEE 802.3 Ethernet, however, specifies that only values from 0 to 1500 are allowed in this field. Since $1500_{10} = 10111011100_2$, a value which uses only 11 bits, 5 bits are left over for other purposes. Ethernet uses these bits for special features.

The length field is followed by the data field which contains the transmitted data. Although the number of data bytes is identified by the two bytes in the length field, the definition of IEEE 802.3 Ethernet requires that the minimum length of the data field be 46 bytes. This ensures that the shortest Ethernet frame, not including the preamble and start delimiter, is 64 bytes long. If fewer than 46 bytes are being sent, additional bytes are used as padding to expand this field to 46 bytes. These filler bytes are added after the valid data bytes. The value in the length field represents only the valid data bytes, not the padding bytes.

The trailer of the Ethernet frame contains only an error detection checksum. This checksum is a 4-byte CRC. The polynomial for this CRC has bits 32, 26, 23, 22, 16, 12, 11, 10, 8, 7, 5, 4, 2, 1, and 0 set resulting in the 33-bit value $100000100110000010001110110110111_2$.

Remember from Chapter 9 that the polynomial for a CRC is one bit longer than the final CRC checksum.

Since any device can transmit data at any time, the Ethernet network must have a method for resolving when two devices attempt to send data at the same time. Simultaneous transmissions are called ***collisions***, and each device, even the ones that are transmitting data, can detect when a collision occurs. It is assumed that no message survives a collision and both devices are required to retransmit.

In the event of a collision, an algorithm is executed at each of the devices that attempted to transmit data. The outcome of this algorithm is a pseudo-random number specifying how long the device is to wait before attempting to retransmit. The random numbers should be different making it so that one of the devices will begin transmitting before the other forcing the second device to wait until the end of the first device's transmission before sending its own message.

14.5 Sample Protocol: Internet Protocol

IP is a common protocol of the network layer. Remember that the network layer is used to identify logical addresses across a large network regardless of the datalink implementation. IPv4 does this by assigning a unique logical address to every host consisting of four 8-bit numbers. Using these addresses, IP is able to route blocks of data from one host to another.

The blocks of data, called datagrams, are sent either as single blocks or as partitioned fragments. Since error checking occurs at the datalink and transport layers, and since the transport layer provides mechanisms for reassembling packets, IP offers error checking only for its own header and has no mechanism for message-sequencing or flow-control.

In order to be sent across an IP network, an IP header is added to the beginning of a transport layer packet. This header contains information allowing an IP network to route the packet from one host to another. Figure 14-3 presents the basic layout of an IP packet header.

The first field of an IP header is four bits long, and it is used to identify the version of IP used to create the header. Immediately after these four bits is the four-bit length field. The value in this field, when multiplied by four, identifies the number of bytes in the header. This makes it possible to include variable length fields within the IP header.

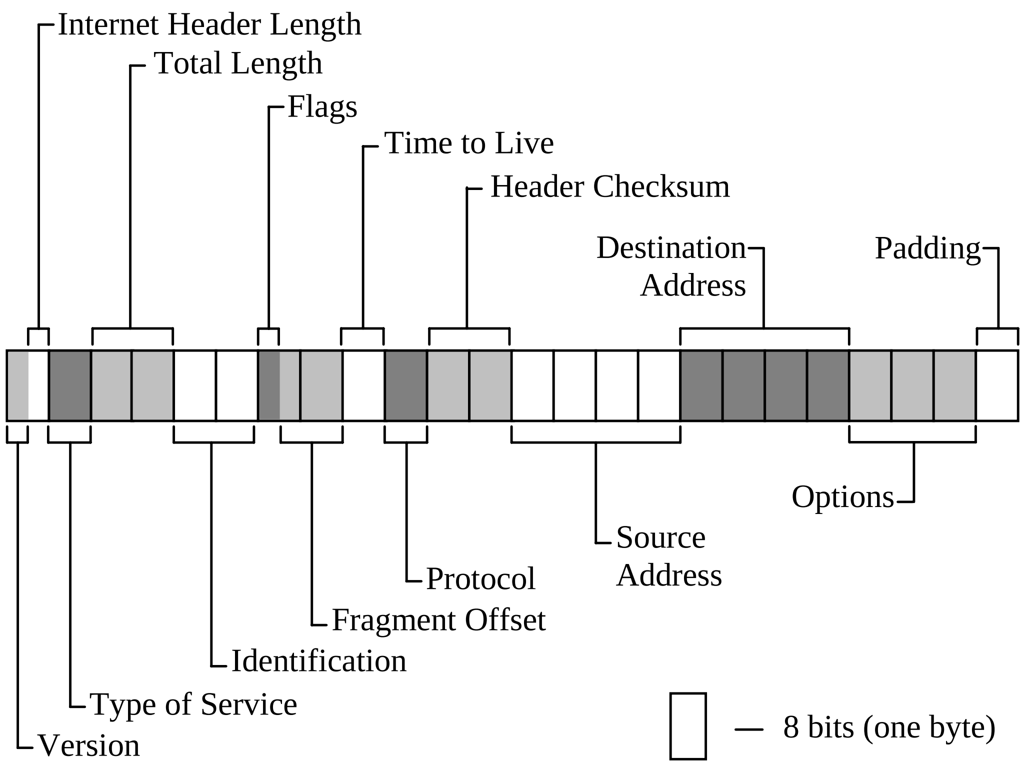

Figure 14-3 Layout of an IP Packet Header

The 8-bit field following the length field identifies the type of service. In large networks, there are usually many paths that a packet can take between two hosts. Each of these paths may have different degrees of speed, reliability, security, and so forth. The type of service field is used to specify the type of link across which a message is to be sent.

The next two bytes define the total length of the message. It is calculated by adding the number of bytes in the IP header to the number of bytes contained in the packet that follows it. This value does not count any bytes from the frame in which the IP packet may be contained.

Sixteen bits would suggest that an Ethernet packet can be up to 65,535 bytes in length. Note, however, that many networks will not allow packets of this size. For example, for an IP packet to be contained within an Ethernet frame, the packet can be at most 1500 bytes long.

The two bytes following the total length field are referred to as the identification field. They are used by the destination host in order to group the fragments contained in multiple IP packets so that they can be reassembled into a single datagram.

The next byte is divided into two parts. The first three bits represent the flag field while the last five bits are coupled with the following byte to create the fragment offset field. The first bit of the flag field is reserved. The second bit indicates whether the datagram may be partitioned into fragments. The last bit is used to identify the current packet as the last fragment.

When a datagram has been partitioned into fragments, a mechanism must be in place to reorder the fragments. This is due to the nature of large networks in which different paths of varying duration can be taken by different packets. This makes it impossible for the receiving device to determine the order in which the packets were sent merely by relying on the order in which they were received.

The fragment offset field contains thirteen bits which are used to identify the starting position of a fragment within the full datagram. Because the partitioning of datagrams must occur on 64-bit boundaries, the value in the fragment offset field is multiplied by eight to determine the fragment's offset in bytes. An offset of zero identifies the fragment as the first fragment within the datagram.

It is possible for a packet to become lost or undeliverable within a large network. Therefore, each packet is sent with a field identifying how long the packet is allowed to remain on the network. This field is referred to as the time to live field. Every time the packet encounters a module on the network, the value in the time to live field is decremented. If the value in this field is decremented to zero before reaching its destination, the packet is discarded.

The next eight bits of the IP header identify the protocol of the packet contained in the data field.

In order to verify the integrity of the IP header, a sixteen bit header checksum is calculated and inserted as the next field. IP uses the one's complement of the one's complement datasum discussed in Chapter 9. Remember that a checksum was identified as being less reliable than a CRC. The one's complement checksum, however, has proven adequate for use with IP headers.

Since the checksum is part of the IP header, this field must be filled with zeros during the checksum calculation to avoid a recursive condition. In addition, since the time to live field changes as the packet is routed across the network, the header checksum will need to be recalculated as it traverses the network.

As was mentioned earlier, IP uses logical addresses to identify both the sending and receiving devices. The next two fields of the IP header contain these addresses. The first four byte field represents the source address while the next four bytes represent the destination address field.

Some IP packets may have special requirements such as security restrictions and routing needs. These requirements are listed next in the IP header in a field referred to as the options field. Depending on the needs of a specific packet, a number of options can be represented. Therefore, this field is variable in length.

Finally, the length of an IP header must be a multiple of 32 bits. To ensure this, a padding field filled with zeros is added to the end of the IP header. Any data contained in the packet immediately follows the padding.

14.6 Sample Protocol: Transmission Control Protocol

A network layer packet such as IP may in turn contain a packet from the transport layer. A common protocol used to define a transport layer packet is the transmission control protocol (TCP). The mechanisms provided by TCP allow large blocks of data to be partitioned, delivered, reassembled, and checked for errors.

TCP uses no host addressing. Instead, TCP uses a set of logical (as opposed to physical) connections called ***ports*** to uniquely identify target and source applications. For example, a TCP port number may identify the data it carries as being part of a web page. Since a TCP packet contains no addressing, it depends on the network layer packet containing it to guarantee it reaches its destination.

Port numbers are identified with 16 bits and therefore can take on values from 0 to 65535. The first 1,024 ports (0 through 1023) are reserved for well-established types of applications such as http (port 80) and version 3 of the post office protocol (POP3) (port 110). Ports 1024 through 49151 are called registered ports, and like the first 1,024 ports, the purpose of these ports is well defined. Unlike the first 1,024 ports, the registered ports typically represent a specific application. For example, port 1433 is reserved for Microsoft SQL Server applications while port 3689 is reserved for Apple iTunes music sharing. The remaining ports, from 49152 through 65535, are dynamically allocated to applications requiring ports not defined by 0 to 49151.

One of the benefits of TCP is its ability to break large blocks of data into smaller packets. This makes it so that one message does not tie up the network for a long period of time. In addition, when an error occurs, only a small portion of the data must be resent rather than the whole data block.

To do this, each byte of the entire data block has a unique integer offset representing the byte's position with respect to the start of the block. This offset is used in each packet for things such as identifying the position of the current packet's data within the completed data block (sequence number) or telling the sending device the position of the data the receiver next expects to receive (acknowledgement number).

Like IP, TCP uses a header inserted before a block of data in order to contain all of the information needed to manage the packet. The layout of this header is presented in Figure 14-4.

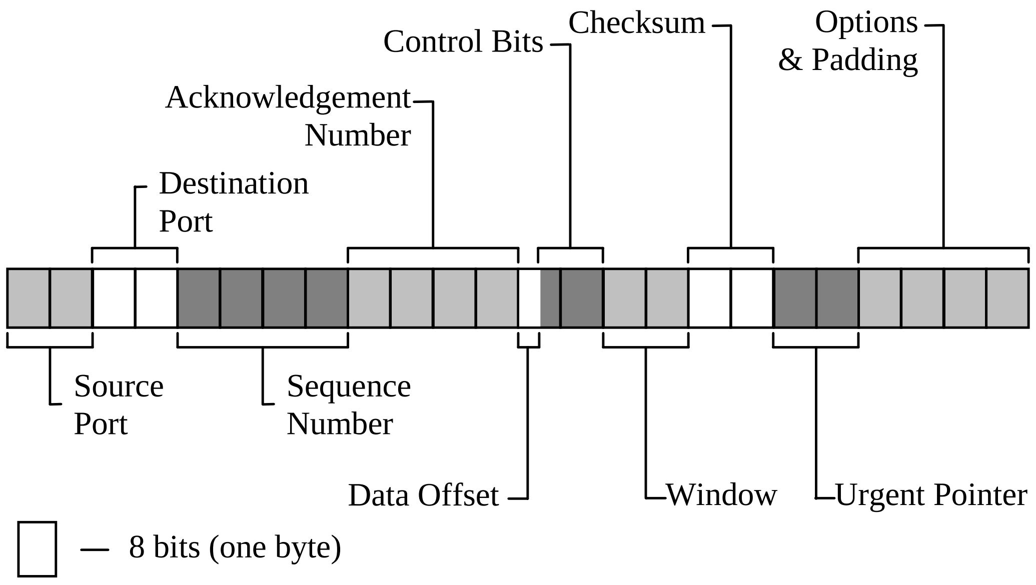

Figure 14-4 Layout of a TCP Packet Header

The first two of the ten fields of the TCP packet header identify the service that sent the message (source port) and the service that is to receive it (destination port) respectively.

The next field is a 32 bit sequence number. The sequence number identifies position of each packet within a block of data using its offset with respect to the beginning of the block. The receiver uses sequence numbers to order the received packets. If packets are received out of order, the sequence number indicates how many bytes to reserve for the missing packet(s). Both the sender and the receiver keep track of the sequence numbers in order to maintain the integrity of the data block.

The next field, the acknowledgement number, is also a 32 bit field. The receiver acknowledges a received message by placing the next expected sequence number in this field.

The next field, the data offset field, is used to identify where the TCP header ends and the data of the packet begins. The value contained in these four bits must be multiplied by four (two left shifts) to calculate the true data offset. This means that a TCP header must have a length equal to an integer multiple of 32 bits (four bytes).

The twelve bits following the data offset field are the control bits of the TCP header. The first six bits are reserved. The last six flag bits are used either to request service such as a connection reset or to identify characteristics of the packet. Figure 14-5 identifies the position and purpose of the flags in the control bits.

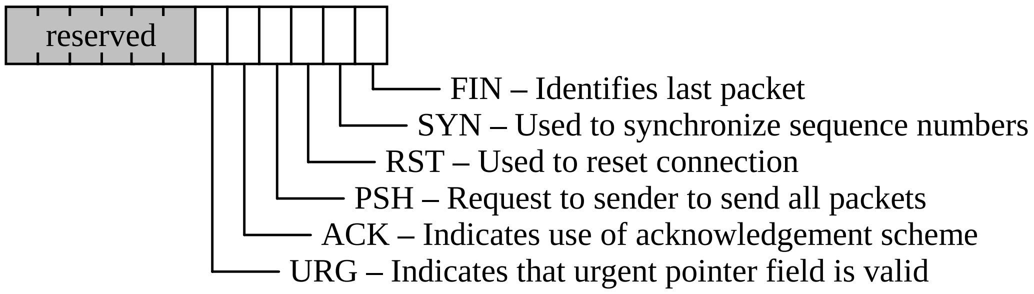
reserved
FIN – Identifies last packet
SYN – Used to synchronize sequence numbers
RST – Used to reset connection
PSH – Request to sender to send all packets
ACK – Indicates use of acknowledgement scheme
URG – Indicates that urgent pointer field is valid

Figure 14-5 Position and Purpose of TCP Control Flags

The next field, the window field, is used by the device requesting data to indicate how much data it or its network has the capacity to receive. A number of things affect the value in this field including the amount of available buffer space the receiver has or the available network bandwidth.

A sixteen bit checksum field is next. Like IP, it contains the one's complement of the one's complement datasum. The difference is that the sum is computed across three groups of bytes:

- the TCP header without the checksum field;
- the data or payload field (if this is an odd number of bytes, pad with a byte of zeros at the end); and
- a pseudo header created using information from the IP header.

The pseudo header consists of the source and destination IP addresses, the protocol field of the IP header, and a count of the number of bytes in both the TCP header and the TCP data or payload field. Figure 14-6 presents the arrangement of the fields in the pseudo header.

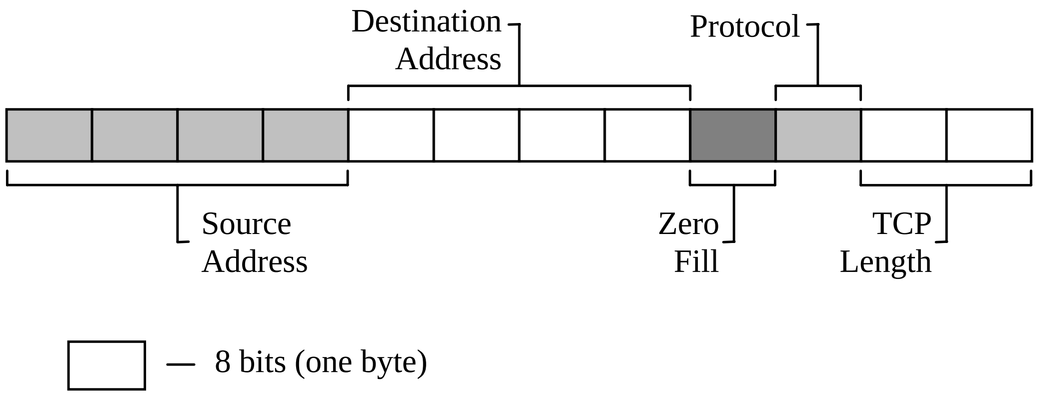

Figure 14-6 Layout of a TCP Pseudo Header

One of the flags in the control block is the urgent flag (URG). By setting this flag to a 1, the sending device is indicating that the data block contains urgent data. When this happens, the sixteen bit field following the checksum is used to identify where that data is contained within the data block. This field is referred to as the urgent pointer field. The value contained in this field is added to the sequence number to identify the position of the urgent data within the packet.

As with the IP protocol, some packets have special requirements. These requirements are identified in a variable length list of options. These options occur in the header immediately after the urgent pointer field. The options identify packet requirements such maximum receive segment size.

Because the option field is variable in length, an additional field referred to as the padding field must be added to ensure the length of the header is a multiple of four bytes. (Remember that the data offset field identifies the length of the header using integer multiples of 32 bits.) The padding field appends zeros to the header after the options field to do this.

14.7 Dissecting a Frame

Remember that the purpose of a protocol is to allow a device receiving a serial data stream to correctly interpret its contents. Figure 14-7 represents one such data stream where a TCP header and data segment is contained within an IP packet which in turn is contained within an Ethernet frame. This section uses the data in this figure to identify the components and data of the frame and its packets.

```
offset                         data

 0000:  00 04 76 48 35 AD 00 B0 D0 C1 6B 31 08 53 45 00
 0010:  00 53 6D F4 40 00 80 06 CC 3C C5 A8 1A 8C C5 A8
 0020:  1A 97 17 0C 0D BE DE B1 57 C5 79 59 3E D4 50 18
 0030:  42 18 B6 3E 00 00 00 B4 00 30 00 22 00 0E 00 00
 0040:  00 05 1A 99 D6 04 DA DE 00 07 FC FF 20 DD 00 00
 0050:  08 00 DA DE 09 04 02 FC FF 0E 00 00 FC FF 01 00
 0060:  0D
```

Figure 14-7 Simulated Raw Data Capture of an Ethernet Frame

Note that the captured data does not include the preamble, start delimiter, or CRC of the Ethernet frame. In general, this information is used by the network interface card for synchronization of the electronics and error checking, but is not made available to the user. Therefore, the frame shown above starts with the destination and source MAC addresses of the Ethernet frame and ends with its data field.

From Figure 14-2, we see that the first six bytes after the start delimiter represent the destination address. Therefore, the MAC address of the destination card is 00:04:76:48:35:AD. Remember that the first three bytes represents the manufacturer. The three bytes 00:04:76 represents 3Com® Corporation.

The next six bytes represent the source address, i.e., the device sending the frame. In this case, the MAC address of the source is 00:B0:D0:C1:6B:31. 00:B0:D0 identifies the card as a NIC from Dell® Computer Corporation.

The next two bytes, 08:53, identifies the both the frame type and the length of the frame. Converting this value to a sixteen-bit binary value gives us $0853_{16} = 0000100001010011_2$. The most significant 5 bits represent the type, and in this case (00001_2) it is an IP version 4 type. The least significant 11 bits represent the length of the data in the frame. In this case, $00001010011_2 = 83_{10}$ indicating the data field of the Ethernet frame contains 83 bytes.

Immediately after the length field of the Ethernet frame is the start of the IP header. By using Figure 14-3 as a reference, the details of the IP packet can also be revealed.

The first four bits identifies the IP version being used. In this case, the first four bits of the byte 45_{16} equal 4, i.e., IP version 4. The next four bits in this byte equal 5. Multiplying this value by four gives us the length of the IP header: 20 bytes.

Next comes one byte identifying the type of service, i.e., the special requirements of this packet. Zeros in this field indicate that this packet has no special needs.

The next two bytes identify the total length of the IP packet, i.e., the number of bytes in the IP header plus the number of bytes of data. A 53_{16} indicates that header and data together total 83 bytes. Subsequent fields are:

- Identification field = $6DF4_{16} = 28148_{10}$
- Flags = first three bits of $40_{16} = 010_2$ (Do not fragment flag is set)
- Fragment offset = last thirteen bits of $4000_{16} = 0000000000000_2$
- Time to live = $80_{16} = 128_{10}$
- Protocol contained in data field = 06_{16} (TCP)
- Header checksum = $CC3C_{16}$
- Source address = C5.A8.1A.8C = 197.168.26.140
- Destination address = C5.A8.1A.97 = 197.168.26.151

To verify the checksum, divide the IP header into words (byte pairs), and add the words together. Doing this for our message give us:

```
 4500
 0053
 6DF4
 4000
 8006
 CC3C
 C5A8
 1A8C
 C5A8
+1A97
-----
3FFFC
```

Remember that IP uses the one's complement datasum which means that the carries must be added to the datasum. This gives us a datasum of $FFFC_{16}$ + 3 = $FFFF_{16}$ which indicates that the checksum is correct.

Immediately after the IP header is the TCP packet. Using Figure 14-4, the components of the TCP header can be identified too.

First two bytes represent the source port. In the sample data, the source port's value is $170C_{16}$ = 5900_{10}. This represents a port for a virtual computing network (VNC). Next two bytes represent the destination port, which in the sample data is $0DBE_{16}$ = 3518_{10}. This represents the Artifact Message Server port.

The next four bytes, $DEB157C5_{16}$, identifies the sequence number. The four-byte acknowledgement number comes next, $79593ED4_{16}$.

The first four bits of the byte following the acknowledgement number is the data offset. This is 0101_2 =5. Multiplying this value by four gives us the length of the TCP header, i.e., twenty bytes.

The last four bits of the same byte joined with the eight bits of the next byte represent the twelve control bits. In binary, this value is 000000011000_2. From Figure 14-5, we see that the first six bits of these twelve are reserved. After that, they come in order URG, ACK, PSH, RST, SYN, and FIN. From our data we see that the ACK and PSH flags are set.

Following the control bits is the two byte value representing the window size. This is 4218_{16} = 16920_{10} in our sample data.

The next two bytes represent a sixteen bit checksum of $B63E_{16}$. To verify the checksum, the one's complement datasum must be calculated from the TCP header (minus the checksum field), the data field (everything after the TCP header not including any Ethernet trailer), and the pseudo header. Note that since the data field contains 43 bytes which is an odd number, an additional byte of zeros must be added to the end to allow for the computation of a sixteen-bit checksum.

To generate the pseudo header, combine the IP source address ($C5A81A8C_{16}$), the IP destination address ($C5A81A97_{16}$), a byte of zeros concatenated with the protocol from the IP header (0006_{16}), and the length of the TCP header and the payload or data field combined (20 bytes for the header plus 43 bytes of data equals 63_{10} = $003F_{16}$).

Adding the pseudo header, the TCP header, and the data field for the sample data results in the following:

```
   C5A8          170C          00B4
   1A8C          0DBE          1030
   C5A8          DEB1          0222
   1A97          57C5          000E
   0006          7959          0000
  +003F          3ED4          0005
  1C0B8          5018          0A99
                 4218          D404
                 B63E          BADE
Sum of bytes    +0000          2017
from pseudo     35BDB          FCEF
header                         20DD
                               0000
                               0800
                               DADE
    Sum of bytes               0904
        from TCP               02FC  Sum of
          header               FF0E  bytes from
                               0000  data field
                               FCFF
                               0100
                              +0D00
                              5E362
```

Adding these three sums together produces the result $1C0B8_{16}$ + $35BDB_{16}$ + $5E362_{16}$ = $AFFF5_{16}$. By adding the carries to the lower sixteen bits, the checksum result is $FFF5_{16}$ + A_{16} = $FFFF_{16}$. Therefore, the checksum is correct.

Following the checksum is the sixteen bit urgent pointer. The sample data has this field set to 0000_{16}. This makes sense since the URG flag was not set which causes the receiver to ignore this field.

The last field of the TCP header is the option and padding field. By using the length of the header calculated earlier, we can see that the option and padding field (which serve to fill the rest of the header) must have been left out. The header is 20 bytes without them.

14.8 Additional Resources

This chapter has provided an overview of serial protocols using examples from Ethernet, IP, and TCP. An entire book could be written on the details of any one of these protocols. This section presents a few additional references that might be helpful to gain further information on these topics.

One of the basic references for Internet protocols can be found in a list of documents referred to as ***requests for comments*** (RFCs). This list of RFCs is maintained by the Internet Engineering Task Force. The two RFCs used for this chapter are RFC 791 (the standard for IP) and RFC 793 (the standard for TCP). These can be found on the web at the following locations:

- Internet Protocol - DARPA Internet Program Protocol Specification (RFC 791) – http://www.faqs.org/rfcs/rfc791.html
- Transmission Control Protocol - DARPA Internet Program Protocol Specification (RFC 793) – http://www.faqs.org/rfcs/rfc793.html

The Institute of Electrical and Electronics Engineers (IEEE) maintains the standard for the Ethernet protocol. It too can be found on the web at:

- IEEE Std 802.3™-2002 – http://standards.ieee.org/getieee802/download/802.3-2002.pdf.

In the discussion of the Ethernet frame, it was shown how the first three bytes of the MAC address identify the manufacturer. A list of these manufacturer codes, called the Organization Unique Identifiers (OUI), is also maintained by the Institute of Electrical and Electronics Engineers (IEEE). It can be found at:

- IEEE OUI and Company_id Assignments – http://standards.ieee.org/regauth/oui/oui.txt

It also might be of help to be able to identify the ports identified in a TCP packet. A list of registered port numbers is maintained by the Internet Corporation for Assigned Names and Numbers (ICANN). There are a number of resources on the web that allow users to search for a specific port using the number found in the TCP header. Once such service is the Internet Ports Database which can be found at:

- The Internet Ports Database – http://www.portsdb.org/

Last of all, there are a number of programs that allow a user to capture and examine data captured by a NIC. These programs are called protocol analyzers or sniffers. Once such program, Packetyzer, is available for the Windows® operating systems under the GNU license agreement. It is available on the web from Network Chemistry at:

- Network Chemistry: Packetyzer Network Packet Analyzer – http://www.packetyzer.com

14.9 What's Next?

This chapter has shown how each bit position of a serial frame has a defined purpose allowing information to be transmitted from one device to another as long as the frame is well-defined. Although serial communication is favored as a long-distance computer system interface due to its reliability, flexibility, and cost effectiveness, there is a drawback. All of the information pertaining to the delivery of the message including addressing and control must be contained within a single stream of bits. This reduces the performance of the network.

Chapter 15 brings us back to the hardware of the computer through an introduction to the architecture of a processor. This takes us from the detailed view of logic gates to the system level view of the computer and its major components. The study of computer architecture will allow us to better understand how the components work together and how they can be used to improve the computer's performance.

Problems

1. List the two primary causes for reduced reliability in a parallel communication scheme.
2. In the IEEE 802.3 Ethernet frame format, what is the purpose of the 7 byte preamble of alternating ones and zeros sent at the beginning of the frame?
3. What is the binary value of the start delimiter of the IEEE 802.3 Ethernet frame?
4. True or false: The two-byte length field of the IEEE 802.3 Ethernet frame contains the length of the entire message including preamble and start delimiter.
5. What are the minimum and maximum values that can be contained in the length field of an IEEE 802.3 Ethernet frame?
6. What are the minimum and maximum lengths of the data field of an IEEE 802.3 Ethernet frame?

7. True or false: If the amount of data being sent using an IEEE 802.3 Ethernet frame is less than the minimum data field length, the data is padded to 46 bytes and the length field is set to 46.
8. True or false: All devices can see all of the messages that pass across their Ethernet network, even the ones not meant for them.
9. True or false: If two Ethernet devices try to transmit at the same time and a collision occurs, this means that only one message got through and one device will have to retransmit.
10. List all of the components of the packet that are summed together to create the IP checksum?
11. List all of the components of the packet that are summed together to create the TCP checksum?
12. Describe the addressing used in an Ethernet frame.
13. Describe the addressing used in an IP packet.
14. Describe the addressing used in a TCP packet.
15. Using the data shown below, identify each of the components of the Ethernet frame and the IP and TCP packets it contains. Be sure to verify the IP and TCP checksums. (Note that the Ethernet preamble and trailer are not shown.)

```
offset                          data

 0000:  00 B0 D0 FE DF 9F 00 07 B3 18 F0 00 08 00 45 60
 0010:  00 B4 00 7B 40 00 2F 06 0C 86 81 2A 3A 8C 97 8D
 0020:  EA 9F 1F 40 0A 49 2F 1B 57 77 91 28 81 88 50 18
 0030:  40 00 E2 14 00 00 CA B1 00 00 00 86 01 00 00 A2
 0040:  4C CB 2D 36 0D 13 7A 00 00 00 00 00 00 00 08 00
 0050:  00 00 0F 00 00 00 2F 00 00 00 0B 02 07 66 4F 02
 0060:  00 04 00 00 00 00 00 04 00 00 00 00 19 24 54 4F
 0070:  50 49 43 2F 77 69 6D 2F 74 65 6E 6E 69 73 2F 73
 0080:  63 6F 72 65 2F 54 07 66 4F 02 00 00 2B 74 00 00
 0090:  00 00 00 00 00 00 00 00 00 00 00 26 1F 8B 08 00
 00A0:  00 00 00 00 00 03 33 35 30 36 AD 71 AE 71 AA 71
 00B0:  76 AC 71 72 AC 31 AC 31 06 00 11 83 33 5B 12 00
 00C0:  00 00
```