2

NETWORK STANDARDS

LEARNING OBJECTIVES

By the end of this chapter, you should be able to:

* Explain how Internet standards are made and why this approach is valuable.
* Provide the definitions of network standards and protocols: message syntax, semantics, order, and reliability.
* Discuss message ordering in general and in HTTP and TCP.
* Discuss message syntax in general and in IP packets, TCP segments, and UDP datagrams.
* Explain how to encode application messages into bits (1s and 0s).
* Explain vertical communication on hosts.

# How Internet Standards Came to Be

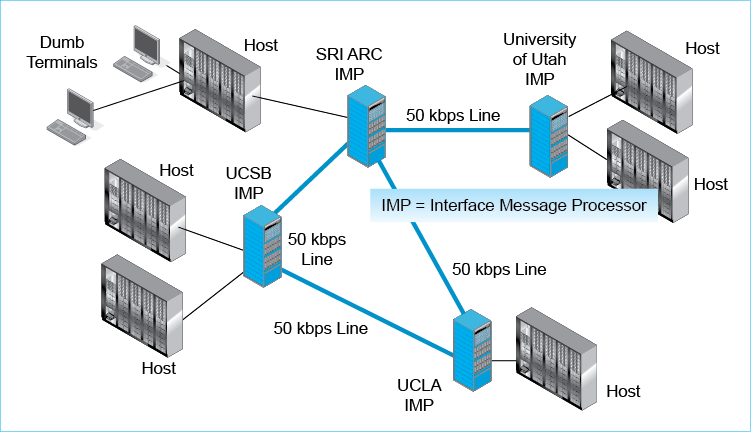
Those who love sausage and revere the law should never see either being made.

Attributed to German Chancellor Otto von Bismarck

Standards are detailed and precise. You might expect standards creation to be logical and rational. For some standards agencies, this is true. The International Telecommunication Union–Telecommunications Standards Sector (the ITU-T in OSI) is part of the ITU, which is an agency of the United Nations. The secretaries of state of individual countries are the nominal participants, although they rarely participate directly. ITU-T has a methodical process for developing new standards. The International Organization for Standards (ISO)[[1]](#footnote-1) also has strict formal processes.

With the Internet, standards setting is different. The Internet grew out of the ARPANET research network funded by the Advanced Research Projects Agency (ARPA).[[2]](#footnote-2) ARPA funded it to explore the then-new technology of packet switching (what we would now call frame switching). Figure 2-1 shows that when the ARPANET began in 1969, it had four sites: UCLA, the Stanford Research Institute Augmentation Research Center, UCSB, and the University of Utah. Each site had a switch called an interface message processor (IMP). IMPs exchanged packets (actually frames) through 50 kbps lines, which seemed blazingly fast at the time.

Figure 2-1: The Early ARPANET



Bolt, Beranek, and Newman (BBN) built the IMPs and designed protocols for IMPs to exchange messages. That was all they did. The first four sites were chosen because they had the technical savvy to get their large host computers to work with the IMPs.

At meetings during the ARPANET’s development phase, researchers from the four sites met with BBN to discuss the network. They realized that the ARPANET would be useless without many additional standards. There had to be standards for hosts to communicate with their IMPs. Far more fundamentally, there had to be application standards if the network was to be useful.

There was nobody to set these standards, so the participants decided to do it themselves. They called their small team the Network Working Group and asked others to join them. When they came up with a standard, they did not call it a standard because they felt that they lacked the authority to do so. Steve Crocker, who led the group and wrote the first document, called it a Request for Comments (RFC). Today, new standards are still RFCs.

Group members developed important application standards. In 1971, Ray Tomlinson saw that e-mail could work across sites. He was already working on e-mail for users of a single host. Mail systems on single hosts used usernames as addresses for delivering mail. Tomlinson saw that an ARPANET address would have to include both a host name and a username on the host. Looking at his keyboard, he saw that the @ sign did not seem to be used very much.[[3]](#footnote-3) He assigned it to separate the username from the host name. It took him a weekend to write the software. E-mail quickly dominated use of the ARPANET.

Born in the late 1960s, the Network Working Group reflected its times. There was a strong focus on egalitarian participation and the recognition of technical merit. A few years later, the Internet Engineering Task Force (IETF) took over Internet standards development. Like the Network Working Group, the IETF has no formal membership. Anyone can participate in the IETF Working Groups that develop individual standards in specific area.

Describing the way the IETF works, Dave Clark wrote, in 1992, “We reject: kings, presidents, and voting. We believe in: rough consensus and running code.”[[4]](#footnote-4) Rejecting kings and presidents refers to the IETF’s strong egalitarian culture. In general, anyone with a good idea stands a fair chance of being heard. By not suppressing new ideas, this culture accounts for much of the rapid development pace of Internet standards. The rejection of voting and going forward if there was rough consensus also made the IETF action-oriented.

The importance of “running code” is not as obvious. Most standards agencies develop full complex standards in committee. When vendors implement these standards, they often find unforeseen ambiguities and even contradictions. When they build their products to these standards, they often find that their products do not interoperate with products from different vendors who supposedly follow the same standard. In addition, committees tend to design standards that are so complex that products take extensive resources to develop and are therefore expensive and slow to develop. In the IETF, almost all standards are created based on running demonstration systems. Experience identifies unforeseen problems and solves them before standards are made.

More subtly, demonstration code is simple. This leads to *simple standards*. Many IETF RFCs even have “simple” in their name; for instance, the Simple Mail Transfer Protocol standardizes communication among mail servers. Simple products emerge quickly, so while OSI development plodded along slowly, simple TCP/IP products appeared fast, at low prices. As something of an insult (although it was not intended to be), the IETF sometimes took bloated OSI standards and created simpler versions of them. These simplified versions often became dominant. Over time, simple IETF standards usually evolve to becoming full-featured, but each step along the way is based on real-world experience.

Test Your Understanding

1. a) Why are Internet standards called RFCs? (Do not just spell out the name.) b) What factors in the Internet’s informal development process lead to rapid standards development and low-cost products?

# Box: April 1 and RFCs

The IETF has a sense of whimsy. In the United States and some other countries, April 1 is April Fool’s Day—a day to play jokes on people by telling them something completely false. A robust tradition in the IETF is the publishing of a facetious RFC or two on April Fool’s Days. One of the most popular is RFC 2549, IP over Avian Carriers. Written in 1990, this RFC describes how to transmit IP packets using carrier pigeons. This RFC was updated twice, in 1999 (to add quality of service) and 2011 (so that the protocol will work with the new IPv6 protocol). Another April 1 RFC warned of a serious authentication problem at IETF meetings. There were so many heavily bearded guys that it was impossible to tell them apart. RFC 3093 introduced the Firewall Enhancement Protocol, which allows all traffic to pass through firewalls while leaving the firewall in place (and useless). An April 1 RFC from 1998, the Hyper Text Coffee Pot Protocol, was created in RFC 2324. This protocol has the promise of growing into a real protocol as the Internet of things unfolds. In justifying the HTCPCP, the RFC said that “there is a strong, dark, rich requirement for a protocol designed espressoly for the brewing of coffee.” One limitation in the protocol is that decaf coffee was explicitly excluded. The explanation was, “What's the point?”

# INTRODUCTION

We looked at network standards briefly in Chapter 1. In this chapter, we will look at standards at a more conceptual level, developing taxonomies of standards types. Much of the rest of this book focuses on specific standards. You will need to understand standards broadly to understand where those specific standards fit into the overall standards picture. This chapter also looks in some detail at the most important standards on the Internet and in corporate networks.

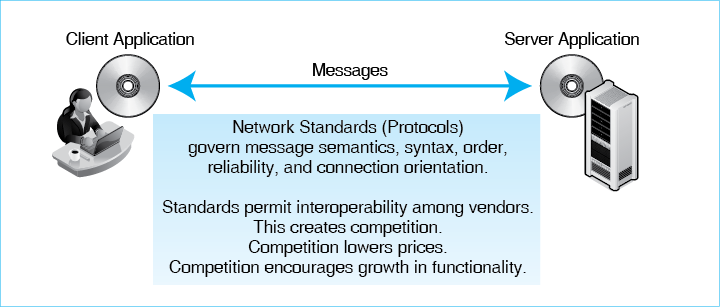
## Standard = Protocol

In this chapter, we use the terms *standard and protocol to mean the same thing*. In fact, standards often have protocol in their names. Examples are the Hypertext Transfer Protocol, the Internet Protocol, the Transmission Control Protocol, and the User Datagram Protocol.

## Network Standards

In Chapter 1, we saw that Standards are detailed rules of operation that specify how two hardware or software processes work together. As Figure 2-2 illustrates, network standards govern the exchange of messages between two hardware or software processes. To give a human analogy, in the authors’ classes, the standard language is American English. Not all of their students are native English speakers, but we are able to communicate because we use a standard language.

Figure 2-2: Network Standards



Standards are important because they allow products from different vendors to interoperate (work together). In Figure 2-2, the client program might be a Chrome browser from Google, and the server program might be Microsoft’s IIS webserver program. Although these companies may actively dislike each other, their products work together because they exchange messages using the Hypertext Transfer Protocol (HTTP) network standard.

With network standards, it is impossible for any company to maintain a monopoly by refusing to allow others to use its proprietary communication protocols. Competition drives down prices. It also spurs companies to add new features so that their products will not be pure commodities that can only compete on price. These new features often appear in the next version of the standard.

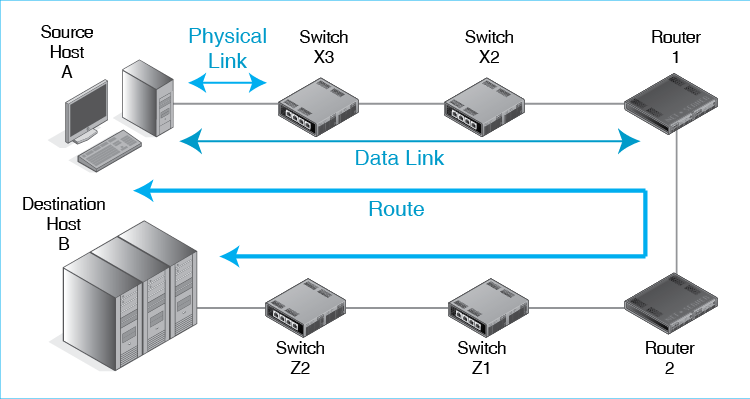
Network standards are not only the key to competition. They are also the key to networking in general. To work in networking, you need to understand individual standards so that you can design networks, set up network components, and troubleshoot problems. Learning networking is heavily about learning standards. In this chapter, we will look broadly at the general characteristics of standards and will also look at some key network standards.

## Recap of Chapter 1 Standards Concepts

In Chapter 1, we saw that standards can be described in terms of their layer of operation.

Delivery Layers.  As Figure 2-3 shows, three layers are involved in the transmission of packets between source hosts and destination hosts.

Figure 2-3: The Physical, Data Link, and Internet Layers



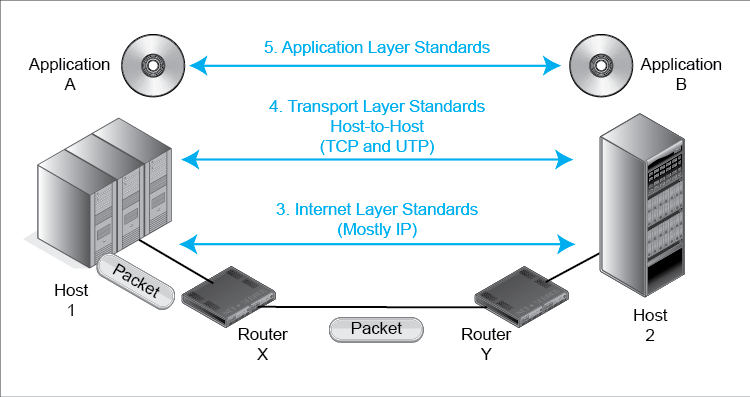
* Physical links are connections between adjacent devices, such as a host and a switch, two switches, two routers, a host and a switch, and so forth. Physical layer standards are not concerned with messages. Their job is to turn the bits of data link layer messages (frames) into signals.
* Data link layer standards govern the transmission of frames between two hosts, two routers, or a host and a router across a single point-to-point, switched, wireless, or hybrid switched/wireless network. The path that a frame takes is called its data link. This layer governs switch operation and frame organization.
* Internet layer standards govern the transmission of packets from the source host to the destination host, across multiple networks in an internet. The path that a packet takes between the two hosts is called its route. This layer governs router operation and packet organization.

A common source of confusion is that concepts are repeated at the data link and internet layers but with different terminology. This occurs because internetworking required the adding of a second layer of standards to those needed for transmission through single networks.

Also, recall that packets are carried inside frames. When a source host sends a packet to a destination host, the packet travels within a frame in each network along the way. If there are 19 single networks on the route between the source and destination hosts, a single packet will travel in 19 different frames.

The Transport and Application Layers.  The physical, data link, and internet layers are for standards that move packets along their way between the source host and the destination host. In contrast, Figure 2-4 shows that transport and application processes govern processes that exist only on the two communicating hosts.

Figure 2-4: Transport and Application Processes



* The transport layer supplements the internet layer. Internet layer operation typically is a best-effort service that does not guarantee that packets will be delivered. The transport layer is a “fix-up” layer that can add reliability and other desirable characteristics to transmission across an internet. In addition, the source host transport layer process fragments application messages. These fragments are sent in individual packets. The destination host transport process reassembles the segments and passes the application message to the application.
* The application layer is for application standards. When two e-mail programs need to work together, they use an e-mail application standard. For webservice, HTTP is an application layer standard. There are more application layer standards than there are standards at all other layers combined because there are so many applications and because different applications usually need different application standards.

The Five Layers.  Figure 2-5 recaps the five layers.

Figure 2-5: Layers Recap

|  |  |  |  |
| --- | --- | --- | --- |
| Broad Function | Layer | Name | Specific Function |
| Interoperability of application programs | 5 | Application | Application layer standards govern how two applications work with each other, even if they are from different vendors. |
| Transmission across an internet | 4 | Transport | Transport layer standards govern aspects of end-to-end communication between two end hosts that are not handled by the internet layer. These standards also allow hosts to work together even if the two computers are from different vendors or have different internal designs. |
| 3 | Internet | Internet link layer standards govern the transmission of packets across an internet—typically by sending them through several routers along the route. Internet layer standards also govern packet organization, timing constraints, and reliability. |
| Transmission across a single network | 2 | Data Link | Data link layer standards govern the transmission of frames across a single switched network—typically by sending them through several switches along the data link. Data link layer standards also govern frame organization, timing constraints, and reliability. |
| 1 | Physical | Physical layer standards govern transmission between adjacent devices connected by a transmission medium. |

* Speaking broadly, the physical and data link layers govern transmission through single networks.
* Also speaking broadly, the internet and transport layers together govern transmission through an internet. The internet layer governs packet organization and raw packet delivery. The transport layer fixes up problems and does fragmentation and assembly.
* Finally, the application layer governs how two applications work together.

Test Your Understanding

2. a) Give the definition of network standards that this chapter introduced. b) In this book, do standards and protocols mean the same thing?

## Network Standard Characteristics

Network standards govern communication. Figure 2-6 notes, more specifically, that standards govern four specific things about message exchanges: semantics, syntax, order, and reliability. In this section, we will focus on message order, semantics, and syntax, but we will also introduce the concept of reliability.

Figure 2-6: Network Standards Concepts

Network Standards

Network standards are rules that govern the exchange of messages between hardware or software processes on different hosts, including messages (ordering, semantics, and syntax), reliability, and connection orientation.

Message Order

Turn taking, order of messages in a complex transaction, who must initiate communication, etc.

In the World Wide Web, the client program sends an HTTP request message

The webserver program sends back an HTTP response message

The client must initiate the interaction

Other network standards have more complex turn-taking; for instance, TCP

Human turn taking is loose and flexible

Message order for network standards must be rigid because computers are not intelligent

Message Semantics

Semantics = the meaning of a message

HTTP request message: “Please give me this file”

HTTP response message: Here is the file. (Or, I could not comply for the following reason)

Network standards normally have a very limited set of possible message meanings

For example, HTTP requests have only a few possible meanings

GET: Please give me a file

PUT: Store this file (not often used)

A few more

Message Syntax (Organization)

Like human grammar, but more rigid

Header, data field, and trailer (Figure 2-9)

Not all messages have all three parts

Field lengths are measured in bits or bytes

Bytes are also called octets

Message Ordering.  In medicine and many other fields, a protocol is a prescribed series of actions to be performed in a particular order. In cooking, if you do not process the ingredients of a cake in the right order, the cake is not likely to turn out very well.

In this same way, network standards govern message ordering. For the Hypertext Transfer Protocol standard that we saw in Chapter 1, message ordering is very simple. The client sends an HTTP request message, and the server sends back an HTTP response message. Many protocols, including the Transmission Control Protocol (TCP) standard, which we will see in this chapter and in Chapter 8, involve many messages being sent in precise order.

Human beings are intelligent, so message ordering in human conversations tends to be informal and even chaotic. Software is not intelligent, so message ordering in protocols has to be very rigid and exact.

Semantics.  To limit the complexity of software, protocols usually define only a few message types, and these types usually have only a few options. Put another way, network protocols greatly limit the semantics (meaning) of their messages. For example, the most common HTTP request message is a GET message, which requests a file. There is also a POST request message, which uploads a file to the webserver. Similarly, the semantics of an HTTP response message are, “Here is the file,” or “Sorry, I can’t deliver the file.”

Semantics is the meaning of a message.

Syntax.  In addition, while human grammar is very flexible, network messages have very rigid syntax, that is, message organization. A little later in this chapter, we will look at the syntax of several important protocol messages.

Syntax is how a message is organized.

Test Your Understanding

3. a) What three aspects of message exchanges did we see in this section? b) Give an example not involving networking in which the order in which you do things can make a big difference. c) Distinguish between syntax and semantics.

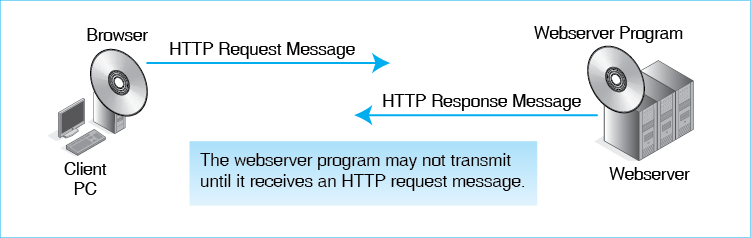
# EXAMPLES OF MESSAGE ORDERING

We will look at two examples of message ordering. We will look first at the very simple message ordering in HTTP. We will then look at the more complex message ordering in TCP.

## Message Ordering in HTTP

Figure 2-7 illustrates an HTTP request–response cycle. As we have just noted, the client sends a request, and the server sends a response. Note that the cycle is always initiated by the client, not by the server. The server cannot transmit unless the client has sent it an HTTP request message. This is a very simple type of message ordering.

Figure 2-7: An HTTP Request–Response Cycle

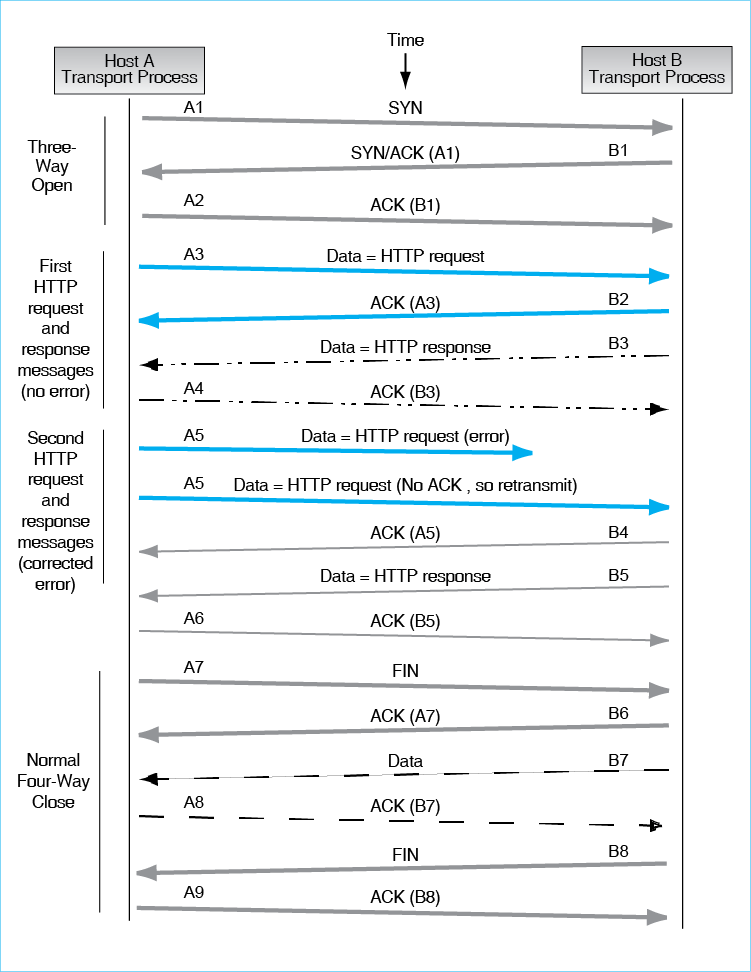


## Message Ordering and Reliability in TCP at the Transport Layer

Many protocols have much more complex rules for message ordering. We will look at the Transmission Control Protocol at the transport layer to see an example of this complexity.

The Situation.  Figure 2-8 shows the transport layer processes on Host A and Host B. They are communicating via HTTP at the application layer. The Hypertext Transfer Protocol requires the use of TCP at the transport layer. The figure shows a sample communication session, which is called a connection.

Figure 2-8: A TCP Session



Segments.  In TCP, messages are called TCP segments because each carries a segment (fragment) of a fragmented application message (or is a control segment that does not carry application data).

The Three-Step Handshake Opening.  The communication begins with a three-step handshake.

* Host A, which is the client in the HTTP exchanges, initiates the communication. It transmits a TCP SYN segment to Host B. This indicates that Host A wishes to communicate.
* Host B sends back a TCP SYN/ACK segment. The SYN indicates that it also is willing to begin the communication. The ACK part is an acknowledgment of Host A’s SYN message. In TCP, all segments are acknowledged, with the primary exception of pure ACKs. (If pure ACKs had to be acknowledged, there would be an endless series of ACKs.)
* Host A sends back a pure TCP ACK segment. This acknowledges Host B’s SYN/ACK.

In TCP, all segments are acknowledged, with the primarily exception of pure ACKs.

Connections.  TCP creates connections with distinct openings and closings. This is like a telephone call, in which you informally make sure that the other person can talk at the start of a call and mutually agree to end the call. In technical jargon, TCP is a connection-oriented protocol.

In contrast, HTTP is a connectionless protocol. There is no need to open a conversation, close it, or send acknowledgement. The client can send a HTTP request any time it wishes. It is like e-mail. You rarely call people to see if it is OK to send them an e-mail message or to stop sending them messages.

Sequence Numbers.  In a connection-oriented protocol, each message is given a sequence number. This allows the receiver to ensure that no message is missing and allows the receiving process to deal with duplicate segments. (It simply discards duplicates.)

Sequence numbers in TCP are important because application message fragments (segments) are delivered in separate packets. Sequence numbers allow the receiver to place the segments in order and reassemble them.

Note in Figure 2-8 that each side numbers its own sequence numbers. For simplicity, we have called Host A’s sequence numbers A1, A2, A3, and so forth. We have done the same with Host B’s messages. So Host A’s SYN segment is A1, while Host B’s SYN/ACK is B1, and Host A’s acknowledgment of the SYN/ACK is A2.

Carrying Application Data.  The next four segments (A3, B2, B3, and A4) constitute a request–response cycle.

* A3 carries an HTTP request.
* B2 is an ACK of A3.
* B3 carries the HTTP response message.
* A4 acknowledges the receipt of B3.

Usually, HTTP request messages are small enough to fit in a single TCP segment. However, most HTTP responses are long and must be sent in a number of TCP segments. This does not change the basic picture, however. There would simply be several more exchanges like B3 and A4.

Reliability.  TCP is a reliable protocol. This means that it corrects errors.

A reliable protocol corrects errors.

* Segment A5 is sent but never reaches Host B.
* Host B does not send an acknowledgment, because ACKs are only sent when a segment is received correctly.
* Host A realizes that A5 has not been acknowledged. After a certain period of time, it retransmits A5.
* This time, the segment arrives correctly at Host B. Host B sends B4, which is an acknowledgment of A5.
* Finally, Host B sends an HTTP response message (B5) and receives an ACK (A6). Again, sending an HTTP response message tends to take several TCP data/acknowledgment cycles.

In this example, Segment A5 never reached the receiving transport process. What would have happened if A5 had reached the transport process but was damaged during transmission? In this case, the receiving transport process would discard the segment. It would not send an ACK. So there is a simple rule for ACKs. Unless a transport process receives a segment correctly, it does not send an acknowledgment.

Unless a transport process receives a segment correctly, it does not send an acknowledgment.

The Four-Step Handshake Closing.  Host A has no more HTTP request messages to send, so it closes the connection. It does so by sending a FIN segment (A7), which Host B acknowledges (B6). This means that Host A will not send new data. However, it will continue to send ACKs to segments sent by Host B.

* Host B has one more data segment to send, B7. When it sends this segment, Host A’s transport process responds with an ACK (A8).
* Now, Host B is finished sending data. It sends its own FIN segment (B8) and receives an acknowledgment (A9).
* The connection is closed.

Perspective.  TCP is a fairly complex protocol. It uses connections so that it can apply sequence numbers to segments. This allows it to fragment long application messages and deliver the segments with an indication of their order. It also uses connections so that it can provide reliable data to the application layer program above it.

We will see that almost all other protocols are unreliable. Many standards check for errors, but if they find an error, they simply discard the message. Discarded messages never get to the transport process on the other host, so they are never acknowledged. Receiving no acknowledgment, the sender resends them.

Why make only TCP reliable? There are two answers. First, TCP sits just below the application layer. This allows it to send clean data to the application program regardless of errors at lower levels, which are corrected by TCP resends.

Second, although all routers along the way do internet layer processing, only the two hosts have transport layer processes, so error correction is done only once, on the two hosts. It is not done at each packet hop between routers or in each frame hop between switches. Error correction is a resource-consuming process, so it should be done as little as possible. Doing error correction at the transport layer processes on the two hosts accomplishes this.

Test Your Understanding

4. a) Describe the simple message ordering in HTTP. b) In HTTP, can the server transmit if it has not received a request message from the client? c) Describe the three-step handshake in TCP connection openings. d) What kind of message does the destination host send if it does not receive a segment during a TCP connection? e) What kind of message does the destination host send if it receives a segment that has an error during a TCP connection? f) Under what conditions will a source host TCP process retransmit a segment? g) Describe the four-step handshake in TCP connection closes. h) After a side initiates the close of a connection by sending a FIN segment, will it send any more segments? Explain. i) In Figure 2-8, suppose Host A had already sent A6 before it realized that it would need to resend A5. When it then resent A5, A6 would arrive before A5. How would Host B be able to put the information in the two segments back in order? j) Why do we say that TCP is connection-oriented but HTTP is not? k) Why do we say that TCP is reliable?

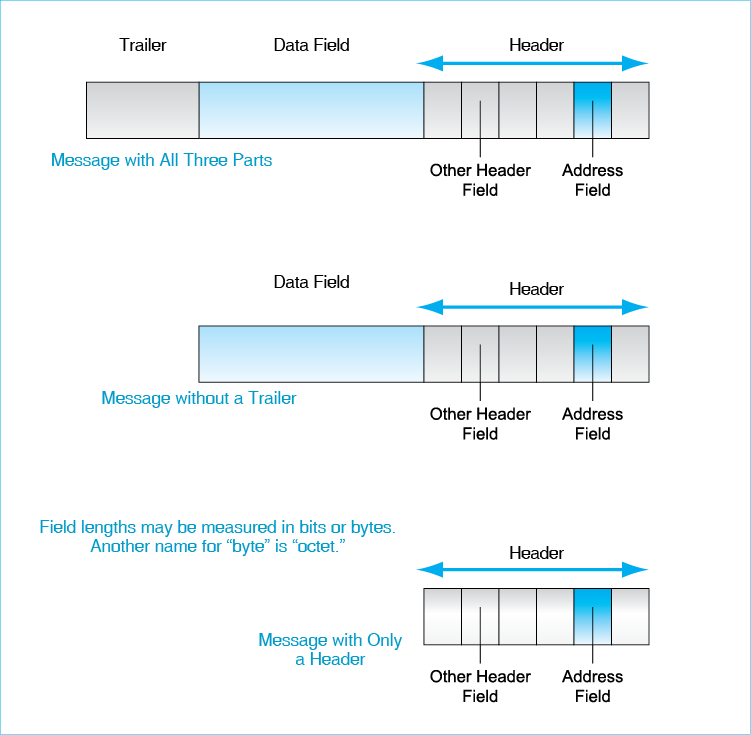
# EXAMPLES OF MESSAGE SYNTAX

We have just looked at message ordering. Now we will turn to message syntax. In this chapter, we will look at the syntax of many different types of messages. To give you a feeling for message syntax, we will look at the syntax of five important message types.

## Syntax: General Message Organization

Figure 2-9 shows that message syntax in general can have three parts—a header, a data field, and a trailer.

Figure 2-9: General Message Organization



Data Field.  The data field is the heart of the message. It contains the content being delivered by the message. In an HTTP response message, the data field contains the file that the response message is delivering.

The data field contains the content being delivered by a message.

Header.  The message header, quite simply, is everything that comes before the data field.

The message header is everything that comes before the data field.

Trailer.  Some messages also have trailers, which consist of everything coming after the data field.

The message trailer is everything that comes after the data field.

Not All Messages Have All Three Parts.  HTTP messages demonstrate that only a header is present in all messages. Data fields are not always present but are very common. Trailers are not common.

Fields in Headers and Trailers.  The header and trailer usually contain smaller syntactic sections called fields. For example, a frame or packet has a destination address header field, which allows switches or routers along the way to pass on the frame or packet they receive. When we look at network standard messages in this chapter and in later chapters, we will be concerned primarily with header fields and trailer fields.

The header and trailer usually contain smaller syntactic sections called fields.

Octets.  Field lengths can be measured in bits. Another common measure for field lengths in networking is the octet. An octet is a group of 8 bits. Hey, isn’t that a byte? Yes, exactly. Octet is just another name for byte. The term is widely used in networking, however, so you need to become familiar with it. Octet actually makes more sense than byte, because *oct* means “eight.” We have octopuses, octagons, and octogenarians.[[5]](#footnote-5)

An octet is a group of 8 bits.

Test Your Understanding

5. a) What are the three general parts of messages? b) What does the data field contain? c) What is the definition of a header? d) Is there always a data field in a message? e) What is the definition of a trailer? f) Are trailers common? g) Distinguish between headers and header fields. h) Distinguish between octets and bytes.

## The Internet Protocol (IP) Packet Syntax

32 Bits per Row.  Figure 2-10 illustrates the syntax of Internet Protocol (IP) version 4 (IPv4) packets. Later, we will look at the syntax of IP version 6 (IPv6) packets.

Figure 2-10: The Internet Protocol (IP) Packet Syntax in IPv4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bit 0 | | | Bit 31 | | |
| Version Number (4 bits) | Header Length (4 bits) | Diff-Serv (8 bits) | Total Length (16 bits) | | |
| Identification (16 bits) | | | Flags  (3 bits) | Fragment Offset (13 bits) | |
| Time to Live (8 bits) | | Protocol (8 bits) | Header Checksum (16 bits) | | |
| Source IP Address (32 bits) | | | | | |
| Destination IP Address (32 bits) | | | | | |
| Options (if any) | | | | | Padding |
| Data Field (dozens, hundreds, or thousands of bits) Often contains a TCP segment or UDP datagram | | | | | |

An IP packet is a long string of bits (1s and 0s). Of course, drawing the packet this way would require a page several meters wide. Instead, Figure 2-10 shows that we usually depict an IP packet as a series of rows with 32 bits per row. In binary counting, the first bit is zero. Consequently, the first row shows bits 0 through 31. The next row shows bits 32 through 63. This is a common way of showing syntax in TCP/IP standards, so you need to be familiar with it.

Source and Destination IP Address Fields.  Each IPv4 packet has source and destination IP addresses. Each is 32 bits long, so each has its own row in the header. Routers use destination IP addresses to decide how to forward packets so that they will get closer to their destination.

Unreliability.  The IPv4 Header Checksum field is used for error detection. The sender computes a number based on all the other bits in the IP header. It places this value in the header checksum field. The receiver redoes the calculation on the bits in the arriving IP packet header. If the numbers match, there have been no errors in transmission. If they do not match, then there has been an error. The receiver discards the packet. Although the receiver checks for errors, there is no retransmission of lost or damaged packets. IP is an unreliable protocol.

Test Your Understanding

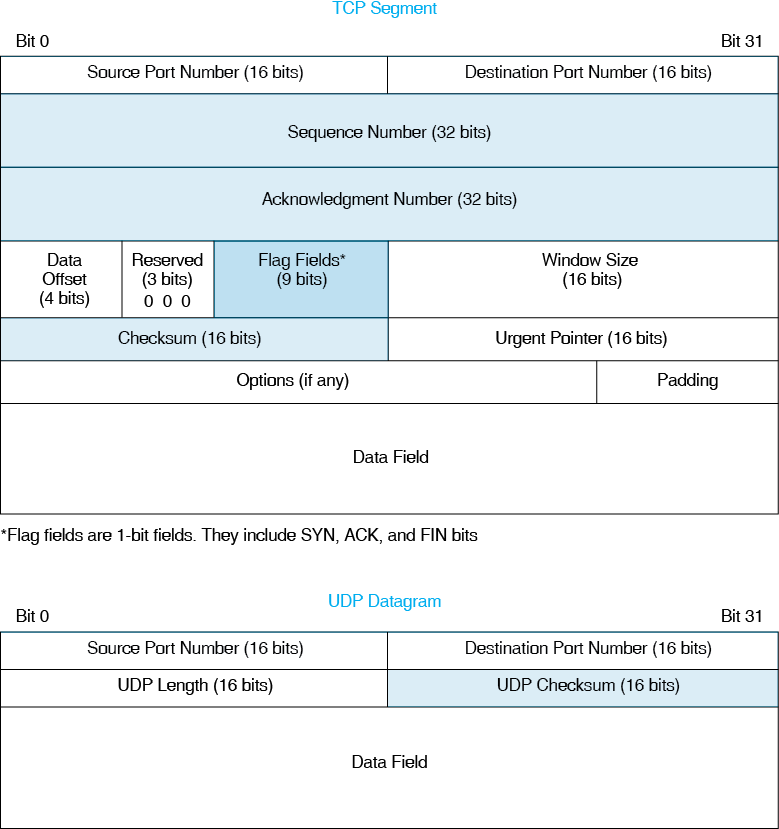
6. a) How many octets long is an IPv4 header if there are no options? (Look at Figure 2-10.) b) List the first bit number on each IPv4 header row in Figure 2-10, not including options. Remember that the first bit in Row 1 is Bit 0. c) What is the bit number of the first bit in the destination address field in IPv4? (Remember that the first bit in binary counting is Bit 0.) d) How long are IPv4 addresses? e) What device besides the destination host reads the destination IP address? f) What is this device’s purpose in doing so? g) Is IP reliable or unreliable? Explain.

## Transmission Control Protocol (TCP) Segment Syntax

Earlier, we saw message ordering in the transmission of TCP segments. Now we will look at the syntax of TCP segments.

Fields in TCP/IP Segments.  When IP was created, it was designed to be a very simple “best effort” protocol (although its routing tables are complex). The IETF left more complex internetwork transmission control tasks to TCP. Consequently, network professionals need to understand TCP very well. Figure 2-11 shows the organization of TCP messages, which are called TCP segments.

Figure 2-11: TCP Segment and UDP Datagram



Flag Fields.  TCP has nine single-bit fields. Single-bit fields in general are called flag fields. If a flag field has the value 1, it is said to be set. (If it has the value 0, it is said to be not set.) In TCP, flag fields allow the receiving transport process to identify the kind of segment it is receiving. We will look at three of these flag bits:

* If the ACK (acknowledgment) bit is set, then the segment acknowledges another segment. When the ACK bit is set, the acknowledgment field also must be filled in to indicate which message is being acknowledged.
* If the SYN (synchronization) bit is set (has the value 1), then the segment requests a connection opening.
* If the FIN (finish) bit is set, then the segment requests a normal connection closing.

Single-bit fields are called flag fields. If a flag field has the value 1, it is said to be set. (If it has the value 0, it is said to be not set.)

Earlier, we talked about TCP SYN segments, ACK segments, and FIN segments. These are simply segments in which the SYN, ACK, or FIN bits are set, respectively.

Sequence Numbers.  Each TCP segment has a unique 32-bit sequence number. This sequence number increases with each segment. Sequence numbers allow the receiving transport process to put arriving TCP segments in order if IP delivers them out of order.

Acknowledgment Numbers.  Earlier in this chapter, we saw that TCP uses acknowledgments (ACKs) to achieve reliability. If a transport process receives a TCP segment correctly, it sends back a TCP segment acknowledging the reception. We saw earlier that if the sending transport process does not receive an acknowledgment, it transmits the TCP segment again.

The acknowledgment number field indicates which segment is being acknowledged. One might expect that if a segment has sequence number X, then the acknowledgment number in the segment that acknowledges it would have acknowledgment number X. The situation is actually more complex.

The acknowledgment number indicates which segment is being acknowledged.

Test Your Understanding

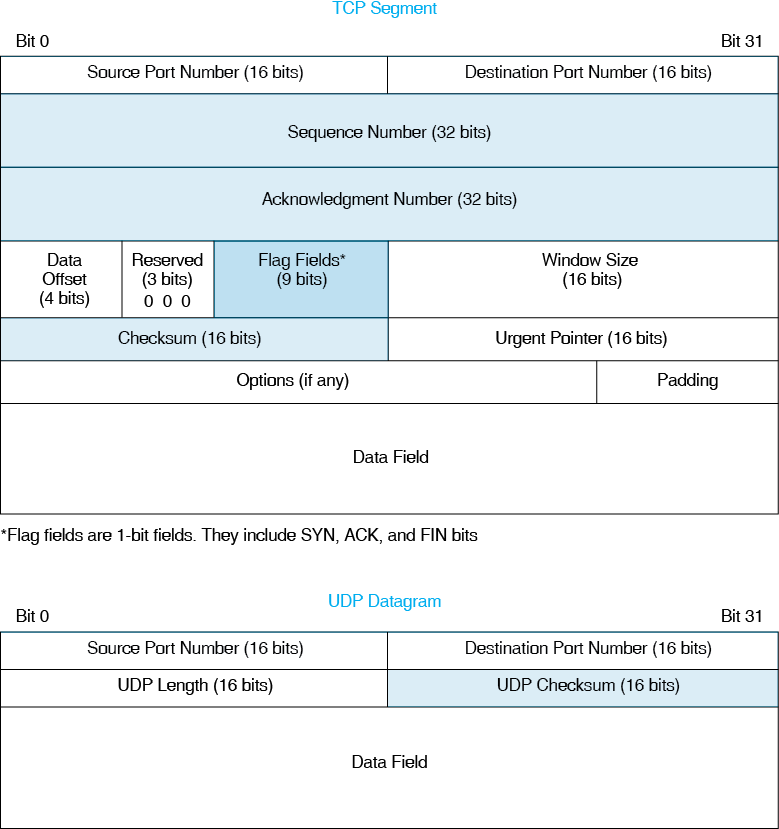
7. a) Why was TCP designed to be complex? b) Why is it important for networking professionals to understand TCP? c) What are TCP messages called?

8. a) Why are sequence numbers good? b) What are 1-bit fields called? c) If someone says that a flag field is set, what does this mean? d) If the ACK bit is set, what other field must have a value? e) What is the purpose of the acknowledgment number field?

## User Datagram Protocol (UDP) Datagram Syntax

Applications that cannot use the high functionality in TCP or that do not need this functionality can use the User Datagram Protocol (UDP) at the transport layer instead of TCP. UDP does not have openings, closings, or acknowledgments, and so it produces substantially less traffic than TCP. UDP messages are called datagrams. Because of UDP’s simple operation, the syntax of the UDP datagram shown in Figure 2-11 is very simple. Besides two port number fields, which we will see next in this chapter, there are only two header fields.

Figure 2-12: UDP Datagram



* There is a UDP length field so that the receiving transport process can know how long the datagram is. The packet in the datagram’s data field has variable length, so the UDP datagram has variable length.
* There also is a UDP checksum field that allows the receiver to check for errors in this UDP datagram.[[6]](#footnote-6) If an error is found, however, the UDP datagram is discarded. In contrast to TCP, UDP has no mechanism for retransmission. It is not reliable.

Test Your Understanding

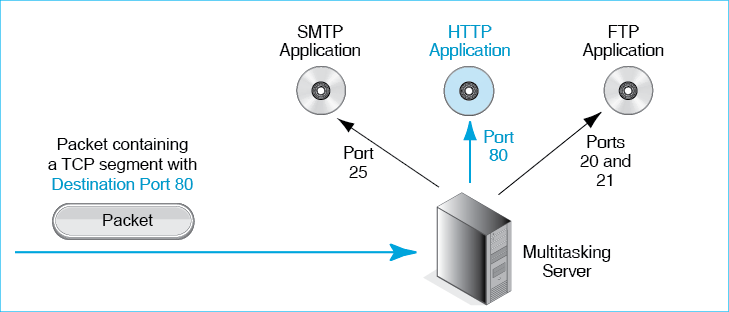
9. a) What are the four fields in a UDP header? b) Describe the third. c) Describe the fourth. d) Is UDP reliable? Explain.

## Port Numbers

Both TCP and UDP headers begin with two port number fields, one specifying the sender’s port number and one specifying the receiver’s port number. Servers and clients use these port number fields differently.

Server Port Numbers.  Computers are multitasking machines, which means that they can run several programs at the same time. Figure 2-13 shows a server running SMTP (the Simple Mail Transfer Protocol), HTTP, and FTP (the File Transfer Protocol) programs. If a packet arrives, how does the TCP or UDP process know which of the application programs running on the server should receive the message?

Figure 2-13: Server Port Numbers



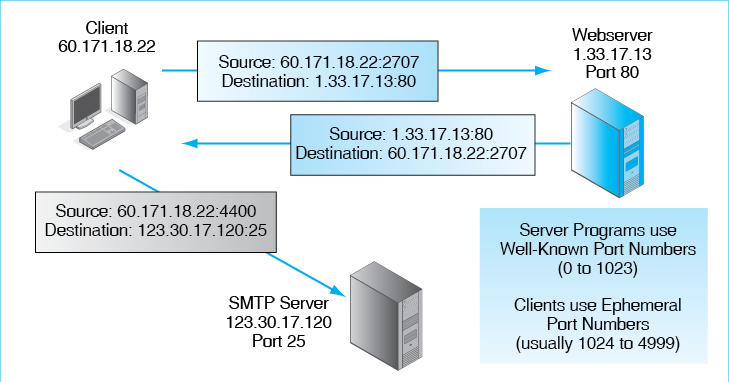
The answer is that the TCP or UDP process uses a port number. A server’s port number specifies a particular application running on the server. Port 20 or 21 specifies the FTP (file transfer protocol) program, while Port 25 specifies the SMTP (e-mail) program, and Port 80 specifies the HTTP (World Wide Web) application.

A server administrator can choose any port number for a program, but there are well-known port numbers that are normally used to specify particular server application programs. Most webservers use this port number for the webserver program. The well-known port numbers have a port number range reserved for their use—port numbers 0 through 1023. To send a TCP or UDP message to the application program on a server, the sender puts the appropriate port number in the destination port number field.

Client Port Numbers.  Clients use port numbers differently. For every conversation a client initiates, it randomly generates an ephemeral port number. On Windows computers, this is the range from port 1024 to Port 4999. These port numbers are ephemeral, in the sense that they are discarded when a conversation between the client and a particular webserver ends. If the client communicates with the same server program later, the client will generate a new ephemeral port number.

Figure 2-14 shows a client host (60.171.18.22) communicating with a blue server host (1.33.17.13). The server port number is Port 80, indicating that the client is communicating with the HTTP program on the server. The client has generated ephemeral Port 2707. When the client transmits to the server, the source port number field has the value 2707 and the destination port number 80. When the server replies, the source port number is 80 and the destination port number is 2707.

Figure 2-14: Client Port Numbers and Sockets



The client is simultaneously connected to an SMTP application on a server (123.30.17.120), which uses the well-known port number 25. For this conversation, the client randomly generates ephemeral Port 4400. When the client transmits, the source port number is 4400 and the destination port number is 25.

Sockets.  Figure 2-14 shows that a conversation always involves a source IP address and a source port number, plus a destination IP address and a destination port number. It is common to represent each IP address and port number as a socket, which is simply the IP address, a colon, and the port number. When the client transmits to the webserver, the source socket is 60.171.18.22:2707 and the destination socket is 1.33.17.13:80. When the webserver replies, the source socket is 1.33.17.13:80, and the destination socket is 60.171.18.22:2707.

Test Your Understanding

10. a) What type of port numbers do servers use for common server programs? b) What type of port numbers do clients use when they communicate with server programs? c) What is the range of port numbers for each type of port? d) How are ephemeral port numbers generated? e) Why are they called ephemeral?

11. a) What is the syntax of a socket? b) In Figure 2-14, when the client transmits to the mail server, what is the source port number? c) What is the destination port number? d) What is the source socket? e) What is the destination socket? f) When the SMTP server transmits to the client host, what is the source port number? g) What is the destination port number? h) What is the source socket? i) What is the destination socket?

# CONVERTING APPLICATION MESSAGES INTO BITS

## Encoding

One function of application layer programs is to convert messages into bits. This conversion is called encoding. At the transport layer and lower layers, all messages consist of bits. Original application layer messages, in contrast, may have text, numbers, graphics images, video clips, and other types of information. It is the application layer’s job to convert all of these into bits before putting them in the application layer message.

Encoding is the conversion of application messages into bits.

Test Your Understanding

12. a) What is encoding? b) At what layer is the encoding of application messages done?

## Encoding Text as ASCII

To convert text data to binary, applications use the ASCII code, whose individual symbols are each 7 bits long. Seven bits give 128 possibilities, as we will see later. This is enough for all keys on the keyboard plus some extra control codes.

Figure 2-15 shows some ASCII codes. It shows that uppercase letters and lowercase letters have different ASCII codes. This is necessary because the receiver may need to know whether a character is an uppercase or lowercase letter. ASCII can also encode the digits from 0 through 9, as well as punctuation and other characters. There are even ASCII control codes that tell the receiver what to do. For example, when we looked at HTTP, we saw carriage returns and line feeds. A carriage return is 0101110, and a line feed is 0100000.

Figure 2-15: Encoding Text as ASCII

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Meaning | 7-Bit ASCII Code | 8th bit in Transmitted Byte |
| Upper-Case Letters | A | 1000001 | Unused |
| Lower-Case Letters | a | 1100001 | Unused |
| Digits (0 through 9) | 3 | 0110011 | Unused |
| Punctuation | Period | 0101110 | Unused |
| Punctuation | Space | 0100000 | Unused |
| Control Codes | Carriage Return | 0001101 | Unused |
| Control Codes | Line Field | 0001010 | Unused |

For transmission, the 7 bits of each ASCII character are placed in a byte. The eighth bit in the byte is not used today.[[7]](#footnote-7)

Test Your Understanding

13. a) Explain how many bytes it will take to transmit “Hello World!” without the quotation marks. (Check Figure: 12.) b) If you go to a search engine, you can easily find converters to represent characters in ASCII. What are the 7-bit ASCII codes for “Hello world” without the quotation marks? (Check: H is 1001000.)

## Converting Integers into Binary Numbers (1s and 0s)

Some application data consists of integers, which are whole numbers (0, 1, 2, 3, … 345, etc.). The sending application program encodes integers as binary numbers. Figure 2-16 shows how this is done.

Figure 2-16: Converting Binary Numbers to Decimal

**Representing Decimal (Base 10) Numbers**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Position Exponent | 104 | 103 | 102 | 101 | 100 |
| Position Value in decimal | 10,000 | 1,000 | 100 | 10 | 1 |
| Decimal Number |  |  | 5 | 0 | 3 |
| Decimal Representation | 503 = 5\*100 + 0\*10+ 3\*1 | | | | |

**Representing Binary (Base 2) Numbers**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Position Exponent | 24 | 23 | 22 | 21 | 20 |
| Position Value in decimal | 16 | 8 | 4 | 2 | 1 |
| Binary Number | 0 | 1 | 0 | 1 | 0 |
| Decimal Equivalents | 0 | 8 | 0 | 2 | 0 |
| Decimal Representation | 1010 = 1\*8 + 1\*2 = 10 | | | | |

Normal arithmetic uses Base 10 representation. A number such as 503 has three decimal positions. Each position has a value that is a power of 10. The position to the farthest right has a value of 100 (1). The next has the value 101 (10). The third has the value 102 (100). Consequently, the number 503 means 5\*100 + 0\*10 +3\*1. This comes so naturally that we do not notice we are treating numbers this way.

Computers use binary (Base 2) arithmetic. Figure 2-16 shows that the positions in binary numbers represent 20, 21, 22, 23, 24, 25, and so forth. These positions have the values 1, 2, 4, 8, 16, 32, and so forth. Consequently, 1010 in binary has the value 1\*8 +0\*4+ 1\*2 + 0\*1 = 10 in decimal. Given this process, you can convert any binary number into its decimal equivalent. In this case, we converted 1010 in binary to 10 in decimal.

Of course, encoding requires conversion in the opposite direction—from decimal to binary.

Figure 2-17 shows how to do this conversion. Here, we wish to convert the decimal number 11 into binary.

Figure 2-17: Encoding a Decimal Number into Binary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Position / Value / Step |  |  |  |  | Remainder |
| Binary Position | 4 | 3 | 2 | 1 |  |
| Binary Position - 1 | 3 | 2 | 1 | 0 |  |
| Binary Value (2Position-1) | 8 | 4 | 2 | 1 |  |
| Step 0: Decimal number = 11 | -- | -- | -- | -- | 11 |
| Step 1: 8 is digit largest that will fit into 11 | 1 | 0 | 0 | 0 | 11-8 = 3 |
| Step 2: 2 is the largest digit that will fit into 3 | 1 | 0 | 1 | 0 | 8=2 = 1 |
| Step 3: 1 is the largest digit that will fit into 1 | 1 | 0 | 1 | 1 | 1-1 = 0 (finished) |
| Final binary number | 1 | 0 | 1 | 1 |  |

* The highest value that will go into 11 is 8, which is 1000. This gives a remainder of 3 (11 – 8).
* In the next step, the highest decimal value that will fit into 3 is 2, or 10 in binary. So now we have 1010 (1000 + 10). The remainder is 1 (3 – 2).
* Finally, the largest decimal value that will fit into 1 is 1. This gives us 1011. The remainder is now 0, so we are finished converting the decimal number 11 into binary.

Test Your Understanding

14. Answer the following without a calculator. a) What is an integer? b) Is 4,307 an integer? c) Is 45.7 an integer? d) Convert the binary number 100 to decimal. (Check Figure: 4.) e) Convert the binary number 1111 to decimal. f) Convert the binary number 10110 to decimal. g) Convert the binary number 100100 to decimal. h) Convert the decimal number 8 to binary. (Check Figure: 1000.) i) Convert 6 to binary (Check Figure: 110.) j) Convert 15 to binary. k) Convert 67 to binary.

## Encoding Alternatives

Some application data can be expressed as alternatives, such as North, South, East, or West. The application layer process will create a field in the application layer message and represent each alternative as a group of bits. For instance, the four cardinal compass points can be represented by a 2-bit field within the application message. North, South, East, and West can be represented as 00, 01, 10, and 11, respectively. (These are the binary numbers for 0, 1, 2, and 3.) There is no order to the alternatives, so any choice can be represented by any pair of bits.

We just saw that having four alternatives requires a 2-bit field. More generally, if a field has *b* bits, it can represent 2*b* alternatives. This gives us the following equation:

Equation 1: *a = 2b*, where a is the number of alternatives and b is the number of bits

We have just seen that a 2-bit field can represent 2*2* alternatives, or 4. Here, *b* is 2, so *a* is 4. What if you need to represent six alternatives? Two bits will not be enough, because 22 is only 4 and we need 6. A three-bit field will give us 23 alternatives, or 8. This gives us enough alternatives. Two alternatives will go unused.

If a field has *b* bits, it can represent *2b* alternatives.

Figure 2-18 illustrates how alternatives encoding is done for fields that have 1, 2, 4, 8, 16, and 32 bits. It shows that with one bit, you can encode yes or no, male or female, or any other dichotomy. Two bits, as we just saw, are good for the four cardinal compass points. With 4 bits, you can have up to 16 alternatives.

Figure 2-18: Binary Encoding to Represent a Certain Number of Alternatives

|  |  |  |  |
| --- | --- | --- | --- |
| Bits in Field | Number of Alternatives that can be Encoded | Possible Bit Sequences | Examples |
| 1 | 21 = 2 | 0, 1 | Yes or No, Male or Female, etc. |
| 2 | 22 = 4 | 00, 01,10, 11 | North, South, East, West; Red, Green, Blue, Black |
| 4 | 24 = 16 | 0000, 0001, 0010, … | Top 10 security threats. 3 bits would give 8 alternatives. Not enough.  4 bits works. 6 values go unused |
| 8 | 28 = 256 | 00000000, 00000001, … | One byte per color gives 256 possible colors levels. |
| 16 | 216 = 65,536 | 0000000000000000, 0000000000000001, ‘… | Two bytes per color gives 65,536 color levels. |
| 32 | 232 = 4,294,967,296 | 000000000000000 0000000000000000, etc. | Number of Internet Protocol Version 4 addresses |

To represent the top 10 security threats, you need 4 bits, which can encode up to 16 alternatives. (Three bits will encode only eight alternatives.) Using 4 bits to represent 10 threats will “waste” six alternatives, but this is necessary.

The *a = 2b* rule is not only used at the application layer. In many layer messages, fields represent alternatives. A one-octet field has 8 bits, so it can represent 28 possible alternatives (256).

You should memorize the number of alternatives that can be represented by 4, 8, and 16 bits, because these are common field sizes. Each added bit doubles the number of possible alternatives, while each bit subtracted cuts the number of possible alternatives in half. So if 8 bits can represent 256 alternatives, 7 bits (one less) can represent 128 alternatives (half as many), while 9 bits (one more) can represent 512 alternatives (twice as many). How many alternatives can 6 and 10 bits represent?

Test Your Understanding

15. a) What does the equation *a = 2b* mean? b) How many alternatives can you represent with a 4-bit field? (Check Figure: 16.) c) For each bit you add to an alternatives field, how many additional alternatives can you represent? d) How many alternatives can you represent with a 10-bit field? (With 8 bits, you can represent 256 alternatives.) e) If you need to represent 128 alternatives in a field, how many bits long must the field be? (Check Figure: 7.) f) If you need to represent 18 alternatives in a field, how many bits long must the field be? g) Come up with three examples of things that can be encoded with 3 bits.

16. a) In TCP, port number fields are 16 bits long. How many possible port numbers are there? b) IPv4 source and destination address are 32 bits long. How many possible IPv4 addresses are there? c) IPv6 addresses are 128 bits long. How many IPv6 addresses are there? d) The IP version number field is four bits long. How many possible versions of IP can there be? e) UDP length fields are 16 bits long. This field gives the number of bytes in the data field. How many bytes long may a UDP data field be? f) ASCII has a 7-bit code. How many keyboard keys can it represent?

# VERTICAL COMMUNICATION ON HOSTS

So far, we have talked about what happens at individual layers. For instance, the transport process on the sending host sends TCP segments or UDP datagrams to the transport process on the receiving host.

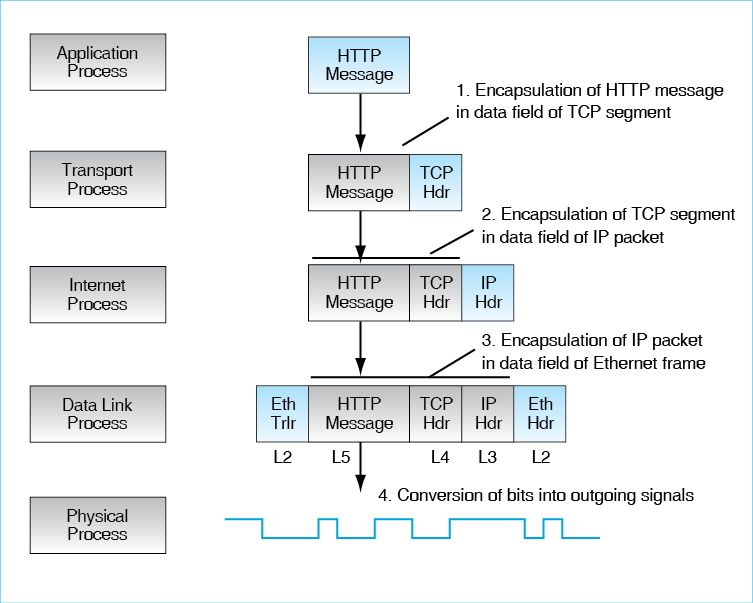
Obviously, however, there is no direct connection between the two hosts at the transport layer. Barring software telepathy, all communication must somehow travel through the physical layer.

In Chapter 1, we saw that Layer 3 packets are carried in the data fields of Layer 2 frames in single networks. Networking people say that the packet is encapsulated (placed) in the data field of the frame. In general, encapsulation is placing a message in the data field of another message.

Encapsulation is placing a message in the data field of another message.

Figure 2-19 shows that encapsulation actually is a process that occurs repeatedly.

Figure 2-19: Layered Communication on the Source Host



* In the figure, the source host’s application layer process sends an HTTP message to the application layer process on the destination host. The source host’s application process cannot deliver the HTTP message, so it passes the HTTP message down to the transport layer process on the source host.
* The transport layer process encapsulates the HTTP message in the data field of a TCP segment. The transport layer then passes the TCP segment down to the internet layer process.
* The Internet layer process encapsulates the TCP segment in the data field of an IP packet. It then passes the packet down to the data link layer.
* The data link layer process encapsulates the IP packet in a data link layer frame. If the single network standard is Ethernet, this is an Ethernet frame. If the single network standard is the Point to Point protocol (PPP), this is a PPP frame. The data link layer process may also add a trailer.

The whole process is like a set of Russian nesting dolls. So the frame consists of the following:

* The data link layer header
* The IP header
* The TCP header
* The application message segment
* The data link trailer (if the single network standard has a trailer in its frame standard).

At the physical layer, something very different occurs. When the physical layer process receives the frame from the data link layer process, it does not do encapsulation. It merely converts the bits of the frame into signals and transmits this signal out to the physical link connecting it to a switch, router, or host.

Test Your Understanding

17. a) What is encapsulation? b) Why is encapsulation necessary for there to be communication between processes operating at the same layer but on different hosts, routers, or switches? c) After the internet layer process in Figure 2-19 receives the TCP segment from the transport layer process, what two things does it do? d) After the data link layer process in Figure 2-19 receives the IP packet from the internet layer process, what two things does it do? e) After the physical layer process receives a frame from the data link layer process, what does the physical layer process do? f) If encapsulation occurs on the source host, what analogous process do you think will occur on the destination host? (The answer is not in the chapter.)

# CONCLUSION

## Synopsis

In this chapter, we looked broadly at standards. Most of this book (and the networking profession in general) focuses on standards, which are also called protocols. Standards govern message exchanges. More specifically, they place constraints on message semantics (meaning), message syntax (format), and message order.

Standards are connection-oriented or connectionless. In connection-oriented protocols, there is a distinct opening before content messages are sent and a distinct closing afterward. There also are sequence numbers, which allow fragmentation and are used in supervisory messages (e.g., acknowledgments) to refer to specific messages. In connectionless protocols, there are no such openings and closings. Connectionless protocols are simpler than connection-oriented protocols, but they lose the advantages of sequence numbers.

In turn, reliable protocols do error correction, while unreliable protocols do not. Although unreliable protocols may do error detection without error correction, this does not make them reliable. In general, standards below the transport layer are unreliable in order to reduce costs. The transport standard usually is reliable; this allows error correction processes on just the two hosts to correct errors at the transport layer and at lower layers, giving the application clean data. Figure 2-20 compares the main protocols we have seen in this chapter in terms of connection orientation and reliability.

Figure 2-20: Protocols in this Chapter

To discuss message ordering in more detail, we looked at HTTP and TCP. Message ordering in HTTP is trivial. The browser must initiate the communication by sending an HTTP request message; afterward the webserver program may transmit. TCP, in contrast, has complex message ordering. Correctly received TCP messages (called TCP segments) are acknowledged by the receiver. If the sender does not receive an acknowledgment promptly, it retransmits the unacknowledged segment. This gives reliability.

To discuss message syntax in more detail, we looked briefly at the syntax of IP packets, TCP segments, and UDP datagrams. We saw that they represent syntax in three different ways. We will be looking at the syntax of many messages in this course, so you should be familiar with all methods for representing syntax. In the discussion, we saw that octet is another name for byte. We also saw that application programs on multitasking servers are usually represented by well-known port numbers, while clients use ephemeral port numbers to represent conversations with server programs. A socket consists of an IP address, a colon, and a port number. It represents a particular program (or conversation) on a particular host.

The application layer must convert text, graphics, video, and other application layer content into bits (1s and 0s). In this chapter, we looked at how application programs encode ASCII text, integers, and number of alternatives into strings of bits.

We looked at how layer processes work together on the source host. After each layer creates its message, it immediately passes the message down to the next-lower-layer process. The data link, internet, and transport processes take every message they are given and encapsulate it in a message suitable for that layer.

## End-of-Chapter Questions

Thought Questions

2-1. How do you think TCP would handle the problem if an acknowledgment were lost, so that the sender retransmitted the unacknowledged TCP segment, therefore causing the receiving transport process to receive the same segment twice?

2-2 a) What is the minimum number of TCP segments required to send an HTTP request and response message? Justify this number. b) Repeat the question, this time if the HTTP response message is damaged during transmission.

2-3. a) In Figure 2-14, what will be the value in the destination port number field if a packet arrives for the e-mail application? b) When the HTTP program on a webserver sends an HTTP response message to a client PC, in what field of what message will it place the value 80?

2-4. Do the following without using a calculator or computer, but check your answers with a calculator or computer. a) Convert 110100 to decimal. (Check Figure: 52.) b). Convert 001100 to decimal. c) Convert 7 to binary. (Check Figure: 111.) d) Convert 47 to binary. e) Convert 100 to binary.

2-5. Do the following without using a calculator or computer, but check your answers with a calculator or computer. You need to represent 1,026 different city names. How many bits will this take if you give each city a different binary number? Explain your answer.

Brainteaser Questions

Think about these questions, but an answer is not required.

2-6. How can you make a connectionless protocol reliable? (You may not be able to answer this question, but try.)

2-7. Spacecraft exploring the outer planets need reliable data transmission. However, the acknowledgments would take hours to arrive. This makes an ACK-based reliability approach unattractive. Can you think of another way to provide more reliable data transmission to spacecraft without using acknowledgments? (You may not be able to answer this question, but try.)

Perspective Questions

2-8. What was the most surprising thing you learned in this chapter?

2-9. What was the most difficult material for you in this chapter?

[Figure 2-1: The Early ARPANET 2](#_Toc440218769)

[Figure 2-2: Network Standards 5](#_Toc440218770)

[Figure 2-3: The Physical, Data Link, and Internet Layers 6](#_Toc440218771)

[Figure 2-4: Transport and Application Processes 7](#_Toc440218772)

[Figure 2-5: Layers Recap 8](#_Toc440218773)

[Figure 2-6: Network Standards Concepts 9](#_Toc440218774)

[Figure 2-7: An HTTP Request–Response Cycle 10](#_Toc440218775)

[Figure 2-8: A TCP Session 11](#_Toc440218776)

[Figure 2-9: General Message Organization 15](#_Toc440218777)

[Figure 2-10: The Internet Protocol (IP) Packet Syntax in IPv4 16](#_Toc440218778)

[Figure 2-11: TCP Segment and UDP Datagram 18](#_Toc440218779)

[Figure 2-12: UDP Datagram 19](#_Toc440218780)

[Figure 2-13: Server Port Numbers 20](#_Toc440218781)

[Figure 2-14: Client Port Numbers and Sockets 21](#_Toc440218782)

[Figure 2-15: Encoding Text as ASCII 23](#_Toc440218783)

[Figure 2-16: Converting Binary Numbers to Decimal 23](#_Toc440218784)

[Figure 2-17: Encoding a Decimal Number into Binary 24](#_Toc440218785)

[Figure 2-18: Binary Encoding to Represent a Certain Number of Alternatives 25](#_Toc440218786)

[Figure 2-19: Layered Communication on the Source Host 27](#_Toc440218787)

[Figure 2-20: Protocols in this Chapter 29](#_Toc440218788)

1. No, the acronyms do not work well with the names, but these are the standard names and acronyms. ISO, by the way, is not an acronym for International Organization for Standardization in any language. It was chosen because *iso* means true in Greek. [↑](#footnote-ref-1)
2. Is it ARPA or DARPA? It depends on the year. It was born ARPA in 1958. In 1972, it became DARPA to emphasize its status as a Department of Defense agency. In 1993, it went back to ARPA. Then it went back to DARPA in 1996. DARPA, “ARPA-DARPA: The Name Chronicles,” undated. <http://www.darpa.gov/arpa-darpa.html>. Last viewed August 2009. [↑](#footnote-ref-2)
3. Personal communication with Ray Tomlinson, May 1986. [↑](#footnote-ref-3)
4. Dave Clark, “A Cloudy Chrystal Ball—Visions of the Future,” *Proceedings of the Twenty-Fourth Internet Engineering Task Force*, Massachusetts Institute of Technology NEARnet, Cambridge, MA, July 13–17, 1992, pp. 539–543. [↑](#footnote-ref-4)
5. What is the eighth month? (Careful!) [↑](#footnote-ref-5)
6. If the UDP checksum field has 16 zeroes, error checking is not to be done at all. [↑](#footnote-ref-6)
7. Early systems used the eighth bit in each byte as a “parity bit” to detect errors in transmission. The total number of bits in a byte was made a whole odd (or even) number by the value of the parity bit. This could detect a change in a single bit in the byte. At today’s high transmission speeds, however, transmission errors normally generate multibit errors rather than single-bit errors. Consequently, parity is useless and is ignored. [↑](#footnote-ref-7)