9
TCP/IP and the Transport Layer

CERTIFICATION OBJECTIVES

9.01 Transport Layer Functions
9.02 Transport Layer Protocols
9.03 TCP and UDP Applications

9.04 Session Establishment
✓ Two-Minute Drill
Q&A Self Test
Chapters 6 and 7 focused on TCP/IP’s Internet layer—comparable to layer 3 of the OSI Reference Model. This chapter moves up one layer and talks about how the transport layer functions at layer 4. It discusses two additional TCP/IP protocols: the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). These two protocols are responsible for moving user data between network components.

CERTIFICATION OBJECTIVE 9.01

Transport Layer Functions

The TCP/IP transport layer is responsible for providing a logical connection between two hosts and can provide these functions:

- Flow control (through the use of windowing)
- Reliable connections (through the use of sequence numbers and acknowledgments)
- Session multiplexing (through the use of port numbers and IP addresses)
- Segmentation (through the use of segment protocol data units, or PDUs)

Flow Control

Flow control, introduced in Chapter 2, is used to ensure that the destination doesn’t become overwhelmed by the source sending too much information at once. Two kinds of flow control exist: ready/not-ready signals and windowing. Recall from Chapter 2 that ready/not-ready signals are not very efficient when a lot of delay is present in the data transmission. For example, if the destination’s receive buffer fills
Transport Layer Functions

up and the destination sends a not-ready signal to the source, the source could still be sending data that the destination would have to drop. And when the destination is ready to start receiving again, the destination sends a ready signal to the source to start sending stuff, introducing a delay before the source can actually begin sending again. This delay can be significant, causing the throughput of the session to drop dramatically.

Windowing is a much more efficient process, since the size of the window determines how many segments can be sent before waiting for an acknowledgment to send the next batch of segments. A good windowing flow control implementation will use a sliding scale, allowing for the size of the window to change based on events occurring at both the source and destination and any congestion or extra bandwidth available between these devices.

Reliability

Reliability is not necessary in all communications. For example, in a voice or video conversation, missing a packet every now and then will probably not be noticeable to the receiver. However, if a file was being transferred, missing even one packet would corrupt the entire file.

When reliability is necessary, it should cover these four items:

■ Recognizing lost packets and having them re-sent
■ Recognizing packets that arrive out of order and reordering them
■ Detecting duplicate packets and dropping the extra ones
■ Avoiding congestion

Most protocols with built-in reliability use sequence and acknowledgment numbers to deal with the first three bullet points. However, how they deal with resending any missed data depends on the protocol’s implementation, as was discussed in Chapter 2. In a best case scenario, the source will resend only those PDUs that were not received by the destination: the destination sends a list of sequence numbers not received and the source resends only those. Most reliable protocols, however, use a simpler but less efficient approach: the destination will send the sequence number of the very first PDU not received and the source will resend that PDU and all subsequent PDUs. A large window size can be efficient if the source is constantly sending a large batch of PDUs, but if the lost PDU is toward the beginning in the sequencing, lots of unnecessary retransmissions result.
Multiplexing

Multiplexing is the ability of a single host to have multiple concurrent sessions open to one or many other hosts. A session occurs when the source opens a connection by sending one or more PDUs and typically, but not always, receives a reply from the destination. A session can be reliable or unreliable and may or may not involve flow control. To handle multiplexing, a transport layer protocol must be able to distinguish between each session to each destination host. Some protocols assign a number to the session, called a session number, to identify the session uniquely. TCP/IP uses a more complicated process that accomplishes basically the same thing.

Segmentation

Segmentation is the process of breaking up data into smaller, identifiable PDUs at the transport layer. In TCP/IP, the transport layer packages application layer data into segments to send to a destination device. The remote destination is responsible for taking the data from these segments and directing it to the correct application. One component of the segment must contain information that will help the destination in the forwarding process, such as specifying the application that is supposed to process the encapsulated data.

CERTIFICATION OBJECTIVE 9.02

Transport Layer Protocols

TCP/IP uses two transport layer protocols: TCP and UDP. The following two sections discuss these protocols in depth and describe their characteristics and the segmentation they use, including the layout of their segment headers.

Transmission Control Protocol

TCP uses a reliable delivery system to deliver layer 4 segments to the destination. This would be analogous to using a certified, priority, or next-day service with the US Postal Service. For example, with a certified letter, the receiver must sign for it, indicating the destination actually received the letter: proof of the delivery is provided. TCP operates under a similar premise: it can detect whether or not the
destination received a sent segment. With the postal example, if the certified letter got lost, it would be up to you to resend it; with TCP, you don’t have to worry about what was or wasn’t received—TCP will take care of all the tracking and any necessary resending of lost data for you.

TCP’s main responsibility is to provide a reliable full-duplex, connection-oriented, logical service between two devices. TCP goes through a three-way handshake to establish a session before data can be sent (discussed later in the “TCP’s Three-Way Handshake” section). Both the source and destination can simultaneously send data across the session. It uses windowing to implement flow control so that a source device doesn’t overwhelm a destination with too many segments. It supports data recovery, where any missed or corrupted information can be re-sent by the source. Any packets that arrive out of order, because the segments traveled different paths to reach the destination, can easily be reordered, since segments use sequence numbers to keep track of the ordering.

TCP transmits information between devices in a data unit called a segment, as mentioned earlier. Recall from Chapter 6 that the IP datagram contains a protocol field, indicating the protocol that is encapsulated in the payload. In the case of TCP, the protocol field contains 6 as a value, indicating that a TCP segment is encapsulated.

Table 9-1 shows the components of a segment. The segment is composed of a header, followed by the application data. Without any options, the TCP header is 20 bytes in length.

TCP provides a reliable connection-oriented, logical service through the use of sequence and acknowledgment numbers, windowing for flow control, error detection and correction (resending bad segments) through checksums, reordering packets, and dropping extra duplicated packets.

Be familiar with the information shown in Table 9-1, especially the fact that a TCP segment contains sequence and acknowledgment numbers as well as a window size.
User Datagram Protocol

UDP uses a best-effort delivery system, similar to how first class and lower postal services of the US Postal Service work. With a first class letter, you place the destination address and return address on the envelope, put it in your mailbox, and hope that it arrives at the destination. With this type of service, nothing guarantees that the letter will actually arrive at the destination, but in most instances, it does. If, however, the letter doesn’t arrive at the destination, it’s up to you, the letter writer, to resend the letter: the post office isn’t going to perform this task for you. UDP operates under the same premise: it does not guarantee the delivery of the transport layer segments.

While TCP provides a reliable connection, UDP provides an unreliable connection. UDP doesn’t go through a three-way handshake to set up a connection—it simply begins sending the data. Likewise, UDP doesn’t check to see whether sent segments were received by a destination; in other words, it doesn’t use an acknowledgment.
process. Typically, if an acknowledgment process is necessary, the transport layer (UDP) won’t provide it; instead, the application itself, at the application layer, will provide this verification.

Given these deficiencies, UDP does have an advantage over TCP: it has less overhead. For example, if you need to send only one segment and receive one segment in reply, and that’s the end of the transmission, it makes no sense to go through a three-way handshake to establish a connection and then send and receive the two segments; this is not efficient. DNS queries are a good example in which the use of UDP makes sense. Voice and video are two other applications that commonly use UDP; assuming that the network path to the destination is fairly reliable and not many packets are dropped, the listener in a phone conversation or the viewer in a video application probably won’t notice that every now and then a packet is lost. Of course, if you are sending a large amount of data to a destination, and you need to verify that all of it was received, TCP would be a better transport mechanism.

When transmitting a UDP segment, an IP header will show 17 as the protocol number in the protocol field. Table 9-2 shows the components of a UDP segment. Notice the many differences between UDP and TCP segments. First, since UDP is connectionless, sequence and acknowledgment numbers are not necessary. Second, since there is no flow control, a window size field is not needed. As you can see, UDP is a lot simpler and more efficient than TCP. Its only reliability component, like TCP, is a checksum field, which allows UDP, at the destination, to detect a bad UDP segment and then drop it. Any control functions or other reliability functions that need to be implemented for the session are not accomplished at the transport layer; instead, these are handled at the application layer.

<table>
<thead>
<tr>
<th>UDP Field Name</th>
<th>Length (in bits)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>16</td>
<td>Identifies the sending application</td>
</tr>
<tr>
<td>Destination Port</td>
<td>16</td>
<td>Identifies the receiving application</td>
</tr>
<tr>
<td>Length</td>
<td>16</td>
<td>Denotes the size of the UDP segment</td>
</tr>
<tr>
<td>Checksum</td>
<td>16</td>
<td>Provides a CRC on the complete UDP segment</td>
</tr>
<tr>
<td>Data</td>
<td>Variable</td>
<td>Application data (not part of the UDP header)</td>
</tr>
</tbody>
</table>

UDP is more efficient than TCP because it has less overhead.
CERTIFICATION OBJECTIVE 9.03

TCP and UDP Applications

One main difference between the OSI Reference Model and TCP/IP’s model is that TCP/IP lumps together the application, presentation, and session layers into one layer, called the application layer, as discussed in Chapter 6. Hundreds and hundreds of TCP/IP applications are available. The most common ones are used to share information, such as file transfers, e-mail communications, and web browsing.

Table 9-3 briefly describes some of the applications and their usage. Here are some common TCP/IP applications, Cisco devices, such as routers and switches, support: domain name service (DNS), HTTP and HTTPS, Simple Network Management Protocol (SNMP), telnet, Secure Shell (SSH), File Transfer Protocol (FTP), and Trivial File Transfer Protocol (TFTP).

<table>
<thead>
<tr>
<th>Application Usage</th>
<th>Applications/Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>File transfers</td>
<td>FTP, TFTP, Network File System (NFS), Remote Procedure Call (RCP)</td>
</tr>
<tr>
<td>Content</td>
<td>HTTP, HTTPS, gopher (HTTPS encrypts traffic)</td>
</tr>
<tr>
<td>E-mail</td>
<td>SMTP, Post Office Protocol 3 (POP)3, Internet Message Access Protocol version 4 (IMAP4)</td>
</tr>
<tr>
<td>Remote login</td>
<td>Telnet, rlogin, RSH, SSH (of these, only SSH encrypts traffic, the rest send traffic in clear text)</td>
</tr>
<tr>
<td>Network management</td>
<td>SNMP</td>
</tr>
<tr>
<td>Name management</td>
<td>DNS</td>
</tr>
<tr>
<td>Voice</td>
<td>Skinny Station Protocol, Session Initiation Protocol (SIP), H.323</td>
</tr>
<tr>
<td>Video</td>
<td>Real Time Streaming Protocol (RTSP), H.323</td>
</tr>
</tbody>
</table>

Some applications encrypt their data, such as SSH and HTTPS (HTTP with SSL).
TCP/IP's transport layer uses port numbers and IP addresses to multiplex sessions between multiple hosts. If you look back at Tables 9-1 and 9-2, you'll see that both the TCP and UDP headers have two port fields: a source port and a destination port. These, as well as the source and destination IP addresses in the IP header, are used to identify each session uniquely between two or more hosts. As you can see from the port number field, the port numbers are 16 bits in length, allowing for port numbers from 0 to 65,535 (a total of 65,536 ports).

Port numbers fall under three types:

- **Well-known** These port numbers range from 0 to 1023 and are assigned by the Internet Assigned Number Authority (IANA) to applications commonly used on the Internet, such as HTTP, DNS, and SMTP.

- **Registered** These port numbers range from 1024 to 49,151 and are assigned by IANA for proprietary applications, such as Microsoft SQL Server, Shockwave, Oracle, and many others.

- **Dynamically assigned** These port numbers range from 49,152 to 65,535 and are dynamically assigned by the operating system to use for a session.

When you want to connect to an application on a destination host, the source port field in the TCP or UDP header will have a dynamically assigned port; the destination port field will have either a well-known or registered port number, depending on the application to which you are connecting. The destination host can use this information to determine what application needs to process the session data.

You can find a list of the port numbers and names at www.iana.org/assignments/port-numbers. Note that some of the applications support TCP, some UDP, and some both, such as DNS. DNS uses UDP for DNS queries and resolutions, but it uses TCP to copy name resolution tables between DNS servers.

**Exam Watch**

Remember a few examples of applications (and their ports) that use TCP: HTTP (80), FTP (21), POP3 (110), SMTP (25), SSH (22), and telnet (23). Remember a few examples of UDP applications, along with their assigned port numbers: DNS queries (53), RIP (520), SNMP (161), and TFTP (69).
Application Mapping

TCP and UDP provide a multiplexing function for simultaneously supporting multiple sessions to one or more hosts. This allows multiple applications to send and receive data to many devices simultaneously. With these protocols, port numbers (at the transport layer) and IP addresses (at the Internet layer) are used to differentiate the sessions.

As shown in Tables 9-1 and 9-2, however, two port numbers are included in the segment: source and destination. When you initiate a connection to a remote application, your operating system should pick a currently unused dynamic port number from 49,152 to 65,535 and assign this number as the source port number in the TCP or UDP header. Based on the application that is running, the application will fill in the destination port number with the well-known or registered port number of the application. When the destination receives this segment, it looks at the destination port number and knows by which application this segment should be processed. This is also true for traffic returning from the destination.

Let’s look at an example, shown in Figure 9-1, that uses TCP for multiplexing sessions. In this example, PC-A has two telnet connections between itself and the server. You can tell these are telnet connections by examining the destination port number (23). When the destination receives the connection setup request, it knows that it should start up the telnet process. Also notice that the source port number is different for each of these connections (50,000 and 50,001). This allows both the PC and the server to differentiate between the two separate telnet sessions. This is a simple example of multiplexing connections.

![Figure 9-1: Multiplexing connections](image-url)
Of course, if more than one device is involved, things become more complicated. In the example shown in Figure 9-1, PC-B also has a session to the server. This connection has a source port number of 50,000 and a destination port number of 23—another telnet connection. This brings up an interesting dilemma. How does the server differentiate between PC-A’s connection that has port numbers 50,000/23 and PC-B’s, which has the same? Actually, the server uses not only the port numbers at the transport layer to multiplex sessions, but also the layer 3 IP addresses of the devices associated with these sessions. In this example, notice that PC-A and PC-B have different layer 3 addresses: 1.1.1.1 and 1.1.1.2, respectively.

Figure 9-2 shows a simple example of using port numbers between two computers. PC-A opens up two telnet sessions to PC-B. Notice that the source port numbers on PC-A are different, which allows PC-A to differentiate between the two telnet sessions. The destination ports are 23 when sent to PC-B, which tells PC-B which application should process the segments. Notice that when PC-B returns data to PC-A, the port numbers are reversed, since PC-A needs to know what application this is from (telnet) and which session is handling the application.

**Exam Watch**

No matter where a session begins, or how many sessions a device encounters, a host can easily differentiate between various sessions by examining the source and destination port numbers as well as the source and destination layer 3 IP addresses.

![Figure 9-2](https://example.com/fig9-2.png)
CERTIFICATION OBJECTIVE 9.04

Session Establishment

TCP and UDP use completely different processes when establishing a session with a remote peer. As you probably already have guessed, UDP uses a fairly simple process. With UDP, one of two situations will occur that indicate that the session is established:

- The source sends a UDP segment to the destination and receives a response
- The source sends a UDP segment to the destination

As to which of the two are used, that depends on the application. And as to when a UDP session is over, that is also application-specific:

- The application can send a message, indicating that the session is now over, which could be part of the data payload
- An idle timeout is used, so if no segments are encountered over a predefined period, the application assumes the session is over

TCP, on the other hand, is much more complicated. It uses what is called a defined state machine. A defined state machine defines the actual mechanics of the beginning of the state (building the TCP session), maintaining the state (maintaining the TCP session), and ending the state (tearing down the TCP session). The following sections cover TCP’s mechanics in much more depth.

TCP’s Three-Way Handshake

With reliable TCP sessions, before a host can send information to another host, a handshake process must take place to establish the connection. Figure 9-3 shows the steps involved.

In Figure 9-3, PC-A wants to send data reliably to PC-B via TCP. Before this can take place, PC-A must establish the session to PC-B. The two hosts go through a three-way handshake to establish the reliable session. The following three steps occur during the three-way handshake:
1. The source sends a synchronization (SYN) segment (where the SYN control flag is set in the TCP header) to the destination, indicating that the source wants to establish a reliable session.

2. The destination responds with both an acknowledgment and synchronization in the same segment. The acknowledgment indicates the successful receipt of the source's SYN segment, and the destination's SYN flag indicates that a session can be set up (it's willing to accept the setup of the session). Together, these two flag settings in the TCP segment header are commonly referred to as SYN/ACK; they are sent together in the same segment header.

3. Upon receiving the SYN/ACK, the source responds with an ACK segment (where the ACK flag is set in the TCP header). This indicates to the destination that its SYN was received by the source and that the session is now fully established.

Once the three-way handshake has occurred, data can be transferred across the session. Because the connection was established first, this type of service is referred to as connection-oriented. Remember that this type of connection always goes through a three-way handshake before one device can start sending and receiving information from another.
An attacker can take advantage of the three-way handshake process to wreak havoc against a host. The attacker spoofs a flood of TCP SYN segments to a victim. In spoofing, the attacker changes his source IP address to something else—valid or invalid. The host receiving the TCP SYNs assumes that each one is a new connection attempt: it places the SYNs in a local table and responds with a TCP SYN/ACK for each connection attempt; then it waits for the third part of the handshake, an ACK reply. The problem with this is that an ACK reply will never arrive for the spoofed sessions. Typically, after 30 to 60 seconds expire, the host will figure out that a problem has occurred and will remove the half-open connection from its local table. However, the local table can fit only so many connections before it begins to deny new ones—both spoofed and valid ones. This is a problem with TCP. A good firewall or intrusion prevention/detection system solution should be able to deal with this problem effectively.

TCP’s Sequencing and Acknowledgments

One of the ways TCP provides a reliable session between devices is by using sequence numbers and acknowledgments. Every TCP segment sent has a sequence number in it. This not only helps the destination reorder any incoming segments that arrived out of order, but it also provides a method of verifying whether all the sent segments were received. The destination responds to the source with an acknowledgment indicating receipt of the sent segments.

Before TCP can provide a reliable session, it has to go through a synchronization phase—the three-way handshake. Let’s expand upon that process by introducing sequence and acknowledgment numbers to the process:

1. The source sends a synchronization frame with the SYN bit marked in the Code field. This segment contains an initial sequence number. This is referred to as a SYN segment.

2. Upon receipt of the SYN segment, the destination responds with its own segment, with its own initial sequence number and the appropriate value in the Acknowledgment field indicating the receipt of the source’s original SYN segment. This notifies the source that the original SYN segment was received. This is referred to as a SYN/ACK segment and the appropriate bits in the Code field are marked.
3. Upon receipt of the SYN/ACK segment, the source will acknowledge receipt of this segment by responding to the destination with an ACK segment, which has the Acknowledgment field set to an appropriate value based on the destination’s sequence number and the appropriate bit set in the Code field.

Here is a simple example of a three-way handshake with sequence and acknowledgment numbers:

1. Source sends a SYN: sequence number = 1
2. Destination responds with a SYN/ACK: sequence number = 10, acknowledgment = 2
3. Source responds with an ACK segment: sequence number = 2, acknowledgment = 11

In this example, the destination’s acknowledgment (step 2) number is one greater than the source’s sequence number, indicating to the source that the next segment expected is 2. In the third step, the source sends the second segment, and, within the same segment in the Acknowledgment field, indicates the receipt of the destination’s segment with an acknowledgment of 11—one greater than the sequence number in the destination’s SYN/ACK segment.

TCP’s Flow Control and Windowing

TCP allows the regulation of the flow of segments, ensuring that one host doesn’t flood another host with too many segments, overflowing its receiving buffer. TCP uses a sliding windowing mechanism to assist with flow control. For example, if the window size is 1, a host can send only one segment and must then wait for a corresponding acknowledgment before sending the next segment. If the window size
is 20, a host can send 20 segments and must wait for the single acknowledgment of the sent 20 segments before sending 20 additional segments. Windowing is discussed in Chapter 2.

TCP employs a positive acknowledgment with retransmission (PAR) mechanism to recover from lost segments. The same segment will be repeatedly re-sent, with a delay between each segment, until an acknowledgment is received from the destination. The acknowledgment contains the sequence number of the segment received and verifies receipt of all segments sent prior to the retransmission process. This eliminates the need for multiple acknowledgments and resending acknowledgments.

The larger the window size for a session, the less number of acknowledgments sent, thus making the session more efficient. Too small a window size can affect throughput, since a host has to send a small number of segments, wait for an acknowledgment, send another bunch of small segments, and wait again. The trick is to figure out an optimal window size that allows for the best efficiency based on the current conditions in the network and on the two hosts' current capabilities.

A nice feature of this TCP windowing process is that the window size can be dynamically changed through the lifetime of the session. This is important because many more sessions may arrive at a host with varying bandwidth needs. Therefore, as a host becomes saturated with segments from many different sessions, it can, assuming that these sessions are using TCP, lower the window size to slow the flow of segments it is receiving. Likewise, a congestion problem might crop up in the network between the source and destination, where segments are being lost; the window size can be lowered to accommodate this problem and, when the network congestion disappears, can be raised to take advantage of the extra bandwidth that now exists in the network path between the two.

What makes this situation even more complicated is that the window sizes on the source and destination hosts can be different for a session. For instance, PC-A might have a window size of 3 for the session, while PC-B has a window size of 10. In this example, PC-A is allowed to send ten segments to PC-B before waiting for an acknowledgment, while PC-B is allowed to send only three segments to PC-A.
CERTIFICATION SUMMARY

TCP/IP has five layers: application, transport, Internet, data link, and physical. This chapter focuses on protocols used at the transport layer: TCP and UDP. TCP provides a reliable connection through the use of sequence numbers and acknowledgments. TCP uses a three-way handshake when establishing a connection: SYN, SYN/ACK, and ACK. TCP uses PAR to recover lost segments, resending segments with a delay between transmissions, until an acknowledgment is received. Applications that use TCP include FTP (21), HTTP (80), SMTP (25), SSH (22), and telnet (23). UDP provides unreliable connections and is more efficient than TCP. Examples of applications that use UDP include DNS (53), RIP (520), SNMP (161), and TFTP (69).
Chapter 9: TCP/IP and the Transport Layer

TWO-MINUTE DRILL

Transport Layer Functions

- The transport layer provides for flow control through windowing and acknowledgments, reliable connections through sequence numbers and acknowledgments, session multiplexing through port numbers and IP addresses, and segmentation through segment PDUs.
- Transport reliability should deal with out-of-order packets, duplicate packets, and congestion avoidance.

Transport Layer Protocols

- TCP provides connection-oriented, reliable connections by using sequence numbers and acknowledgments, windowing, and error detection and correction.
- The TCP header is 20 bytes long and contains two port fields, sequence and acknowledgment number fields, code bit fields, a window size field, a checksum field, and others.
- UDP provides a best effort delivery and is more efficient than TCP because of its lower overhead.
- The UDP header has source and destination port fields, a length field, and a checksum field.

TCP and UDP Applications

- Most TCP and UDP applications/protocols send traffic in clear text, but applications such as HTTPS and SSH encrypt their traffic.
- Well-known (0 to 1023) and registered (1024 to 49,151) port numbers are assigned to applications; dynamic port numbers (49,152 to 65,535) are assigned by the operating system to the source connection of a session.
- Common TCP applications/protocols and their ports are 21 (FTP), 22 (SSH), 23 (telnet), 25 (SMTP), and 80 (HTTP). Common UDP applications/protocols and their ports are 53 (DNS), 69 (TFTP), and 161 (SNMP).
- Multiplexing sessions are achieved through source and destination port numbers and IP addresses.
Session Establishment

❑ TCP goes through a three-way handshake: SYN, SYN/ACK, and ACK.
❑ When acknowledging the receipt of all sequence segments, the destination responds with an acknowledgment number one higher than the last valid sequence number.
❑ If a segment is lost, TCP resends that segment and all proceeding segments up to the last one in the window size by using PAR.
SELF TEST

The following Self Test questions will help you measure your understanding of the material presented in this chapter. Read all the choices carefully, as there may be more than one correct answer. Choose all correct answers for each question.

Transport Layer Functions

1. Flow control is commonly implemented through which of the following mechanisms?
   A. Windowing
   B. Sequence numbers and acknowledgments
   C. Port numbers and IP addresses
   D. Segment PDUs

2. A PDU at the transport layer is called a __________.
   A. frame
   B. datagram
   C. packet
   D. segment

Transport Layer Protocols

3. TCP has all of the following characteristics except which?
   A. Connection-oriented
   B. Windowing
   C. Best-effort delivery
   D. Reordering packets

4. Which of the following is not a common field found in both TCP and UDP headers?
   A. Source Port
   B. Code Bits
   C. Length
   D. Checksum

5. Which of the following is the correct order for a three-way handshake?
   A. SYN, ACK, SYN/ACK
   B. SYN, ACK/SYN, ACK
   C. SYN/ACK, SYN/ACK, ACK
   D. SYN, SYN/ACK, ACK
TCP and UDP Applications

6. Which TCP/IP applications send traffic in clear text? (choose two)
   A. telnet
   B. SSH
   C. HTTPS
   D. FTP

7. Which of the following has an incorrect application-to-well-known-port mapping?
   A. SNMP: 161
   B. TFTP: 21
   C. telnet: 23
   D. SMTP: 25

8. What port number and application does a DNS query use?
   A. 53, TCP
   B. 69, UDP
   C. 520, UDP
   D. 53, UDP

Session Establishment

9. An application is using TCP with a window size of 10. The source sends 10 segments, but segments 5, 6, and 7 are lost. What number does the destination acknowledge with?
   A. 4
   B. 5
   C. 6
   D. 7
   E. 8

10. When a PC opens up two telnet connections to a server, what does the server use to determine that these sessions are different from each other?
    A. Destination IP address
    B. Destination port number
    C. Source port number
    D. Source IP address
SELF TEST ANSWERS

Transport Layer Functions

1. ☑ A. Flow control is commonly implemented through the use of windowing.
   ☑ B refers to reliable connections, C refers to session multiplexing, and D refers to segmentation.

2. ☑ D. A PDU at the transport layer is called a segment.
   ☑ A refers to the data link layer and B and C refer to the network/Internet layer.

Transport Layer Protocols

3. ☑ C. Best-effort delivery is used by UDP.
   ☑ A, B, and D are implemented in TCP.

4. ☑ B. The Code Bits field is found only in TCP and defines control functions, such as synchronization and acknowledgment functions.
   ☑ A, C, and D are found in both TCP and UDP headers.

5. ☑ D. The correct order of the three-way handshake is SYN, SYN/ACK, and ACK.
   ☑ A, B, and C are invalid orders for the code flags in the three-way handshake.

TCP and UDP Applications

6. ☑ A and D. Telnet and FTP send their application data in clear text.
   ☑ B and C encrypt their traffic.

7. ☑ B. TFTP is assigned port 69 and FTP 21.
   ☑ A, C, and D have the correct application-to-port mappings.

8. ☑ D. DNS queries use UDP with port 53.
   ☑ A is used by DNS zone transfers between servers, B is used by TFTP, C is used by the RIP routing protocol.

Session Establishment

9. ☑ B. The destination always acknowledges the next segment expected, which is 5 (the first one lost).
   ☑ Therefore A, C, D, and E are incorrect.

10. ☑ C. Since it’s the same source, the dynamically assigned port numbers will be different numbers.
   ☑ A is incorrect because it’s the same IP address—the server, B would be the same port number—23. D would be the same—the PC’s IP address.