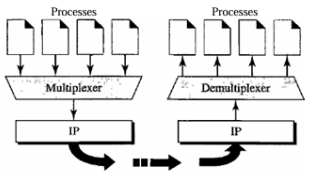
## Multiplexing and Demultiplexing

The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in Figure 23.6.

Figure 23.6 *Multiplexing and demultiplexing*




### *Multiplexing*

At the sender site, there may be several processes that need to send packets. However, there is only one transport layer protocol at any time. This is a many-to-one relationship and requires multiplexing. The protocol accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, the transport layer passes the packet to the network layer.

### *Demultiplexing*

At the receiver site, the relationship is one-to-many and requires demultiplexing. The transport layer receives datagrams from the network layer. After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number.

## Connectionless Versus Connection-Oriented Service

A transport layer protocol can either be connectionless or connection-oriented.

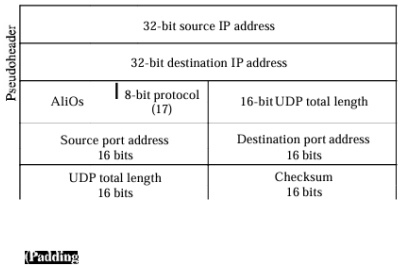
### *Connectionless Service*

In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment either. We will see shortly that one of the transport layer protocols in the Internet model, UDP, is connectionless.

### *Connection-Oriented Service*

In a connection-oriented service, a connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released. We will see shortly that TCP and SCTP are connection-oriented protocols.

Figure 23.10 *Pseudoheader for checksum calculation*




### *Flow and Error Control*

UDP is a very simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages.

There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.

The lack of flow control and error control means that the process using UDP should provide these mechanisms.

*Example 23.2*

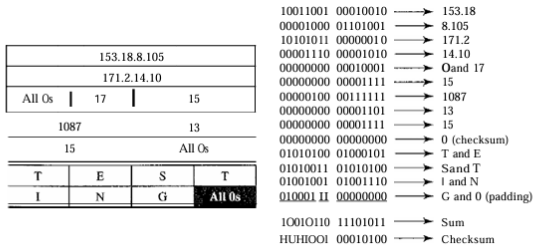
Figure 23.11 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calcUlation. The pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP.

### *Optional Use of the Checksum*

The calculation of the checksum and its inclusion in a user datagram are optional. If the checksum is not calculated, the field is filled with 1s. Note that a calculated checksum can never be allIs because this implies that the sum is all Os, which is impossible because it requires that the value of fields to be Os.



Figure 23.11 *Checksum calculation of a simple UDP user datagram*




## Use of UDP

The following lists some uses of the UDP protocol:

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FrP that needs to send bulk data (see Chapter 26).
- UDP is suitable for a process with internal flow and error control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP (see Chapter 28).
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP) (see Chapter 22).

### *Full-Duplex Communication*

TCP offers full-duplex service, in which data can flow in both directions at the same time. Each TCP then has a sending and receiving buffer, and segments move in both directions.

### *Connection-Oriented Service*

TCP, unlike UDP, is a connection-oriented protocol. When a process at site A wants to send and receive data from another process at site B, the following occurs:

1. The two TCPs establish a connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated.

Note that this is a virtual connection, not a physical connection. The TCP segment is encapsulated in an IP datagram and can be sent out of order, or lost, or corrupted, and then resent. Each may use a different path to reach the destination. There is no physical connection. TCP creates a stream-oriented environment in which it accepts the responsibility of



delivering the bytes in order to the other site. The situation is similar to creating a bridge that spans multiple islands and passing all the bytes from one island to another in one single connection. We will discuss this feature later in the chapter.

*Example 23.3*

Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 1O,00l. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

Solution

The following shows the sequence number for each segment:

| | |
|---|---|
| Segment 1 | Sequence Number: 10,001 (range: 10,001 to 11,(00) |
| Segment 2 | Sequence Number: 11,001 (range: 11,001 to 12,000) |
| Segment 3 | Sequence Number: 12,001 (range: 12,001 to 13,000) |
| Segment 4 | Sequence Number: 13,001 (range: 13,001 to 14,000) |
| Segment 5 | Sequence Number: 14,001 (range: 14,001 to 15,000) |

*Flow Control*

TCP, unlike UDP, provides *flow control.* The receiver of the data controls the amount of data that are to be sent by the sender. This is done to prevent the receiver from being overwhelmed with data. The numbering system allows TCP to use a byte-oriented flow control.

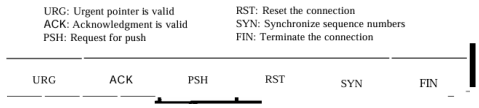
*Error Control*

To provide reliable service, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented, as we will see later.

*Congestion Control*

TCP, unlike UDP, takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also detennined by the level of congestion in the network.

Figure 23.17 *Control field*




A TCP Connection

TCP is connection-oriented. A connection-oriented transport protocol establishes a virtual path between the source and destination. All the segments belonging to a message are then sent over this virtual path. Using a single virtual pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames. You may wonder how TCP, which uses the services of IP, a connectionless protocol, can be connection-oriented. The point is that a TCP connection is virtual, not physical. TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted. Unlike TCP, IP is unaware of this retransmission. If a segment arrives out of order, TCP holds it until the missing segments arrive; IP is unaware of this reordering.

In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

*Connection Establishment*

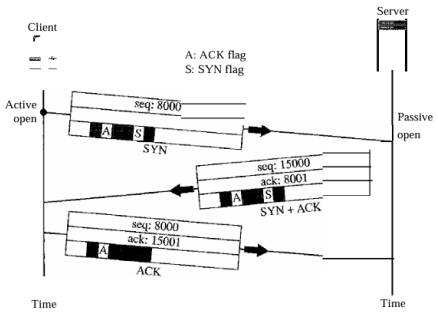
TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously. This implies that each party must initialize communication and get approval from the other party before any data are transferred.

Three-Way Handshaking The connection establishment in TCP is called three-way handshaking. In our example, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport layer protocol.

The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This is called a request for a *passive open.* Although the server TCP is ready to accept any connection from any machine in the world, it cannot make the connection itself.

The client program issues a request for an *active open.* A client that wishes to connect to an open server tells its TCP that it needs to be connected to that particular server. TCP can now start the three-way handshaking process as shown in Figure 23.18. To show the process, we use two time lines: one at each site. Each segment has values for all its header fields and perhaps for some of its option fields, too. However, we show only the few fields necessary to understand each phase. We show the sequence number,

Figure 23.18 *Connection establishment using three-way handshaking*




*Example 23.4*

What is the value of the receiver window *(rwnd)* for host A if the receiver, host B, has a buffer size of 5000 bytes and 1000 bytes of received and unprocessed data?

Solution

The value of *rwnd* = 5000 - 1000 = 4000. Host B can receive only 4000 bytes of data before overflowing its buffer. Host B advertises this value in its next segment to A.

*Example 23.5*

What is the size of the window for host A if the value of *rwnd* is 3000 bytes and the value of *cwnd* is 3500 bytes?

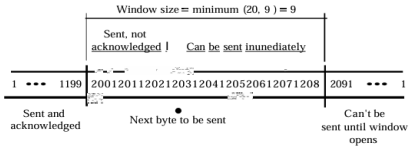
Solution

The size of the window is the smaller of *rwnd* and *cwnd,* which is 3000 bytes.

*Example* 23.6

Figure 23.23 shows an unrealistic example of a sliding window. The sender has sent bytes up to 202. We assume that cwnd is 20 (in reality this value is thousands of bytes). The receiver has sent

Figure 23.23 *Example 23.6*

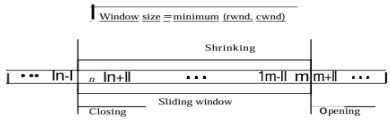



an acknowledgment number of 200 with an *rwnd* of 9 bytes (in reality this value is thousands of bytes). The size of the sender window is the minimum of *rwnd* and *cwnd,* or 9 bytes. Bytes 200 to 202 are sent, but not acknowledged. Bytes 203 to 208 can be sent without worrying about acknowledgment. Bytes 209 and above cannot be sent.

Some points about TCP sliding windows:

- The size of the window is the lesser of *rwnd* and *cwnd.*
- The source does not have to send a full window's worth of data.
- The window can be opened or closed by the receiver, but should not be shrunk.
- The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.
- The receiver can temporarily shut down the window; the sender, however, can always send a segment of 1 byte after the window is shut down.

Figure 23.22 *Sliding window*




need not worry about them anymore. Shrinking the window means moving the right wall to the left. This is strongly discouraged and not allowed in some implementations because it means revoking the eligibility of some bytes for sending. This is a problem if the sender has already sent these bytes. Note that the left wall cannot move to the left because this would revoke some of the previously sent acknowledgments.

A sliding window is used to make transmission more efficient as weD as to control the flow of data so that the destination does not become overwhelmed with data. TCP sliding windows are byte-oriented.

The size of the window at one end is determined by the lesser of two values: *receiver window (rwnd)* or *congestion window (cwnd).* The *receiver window* is the value advertised by the opposite end in a segment containing acknowledgment. It is the number of bytes the other end can accept before its buffer overflows and data are discarded. The congestion window is a value determined by the network to avoid congestion. We will discuss congestion later in the chapter.