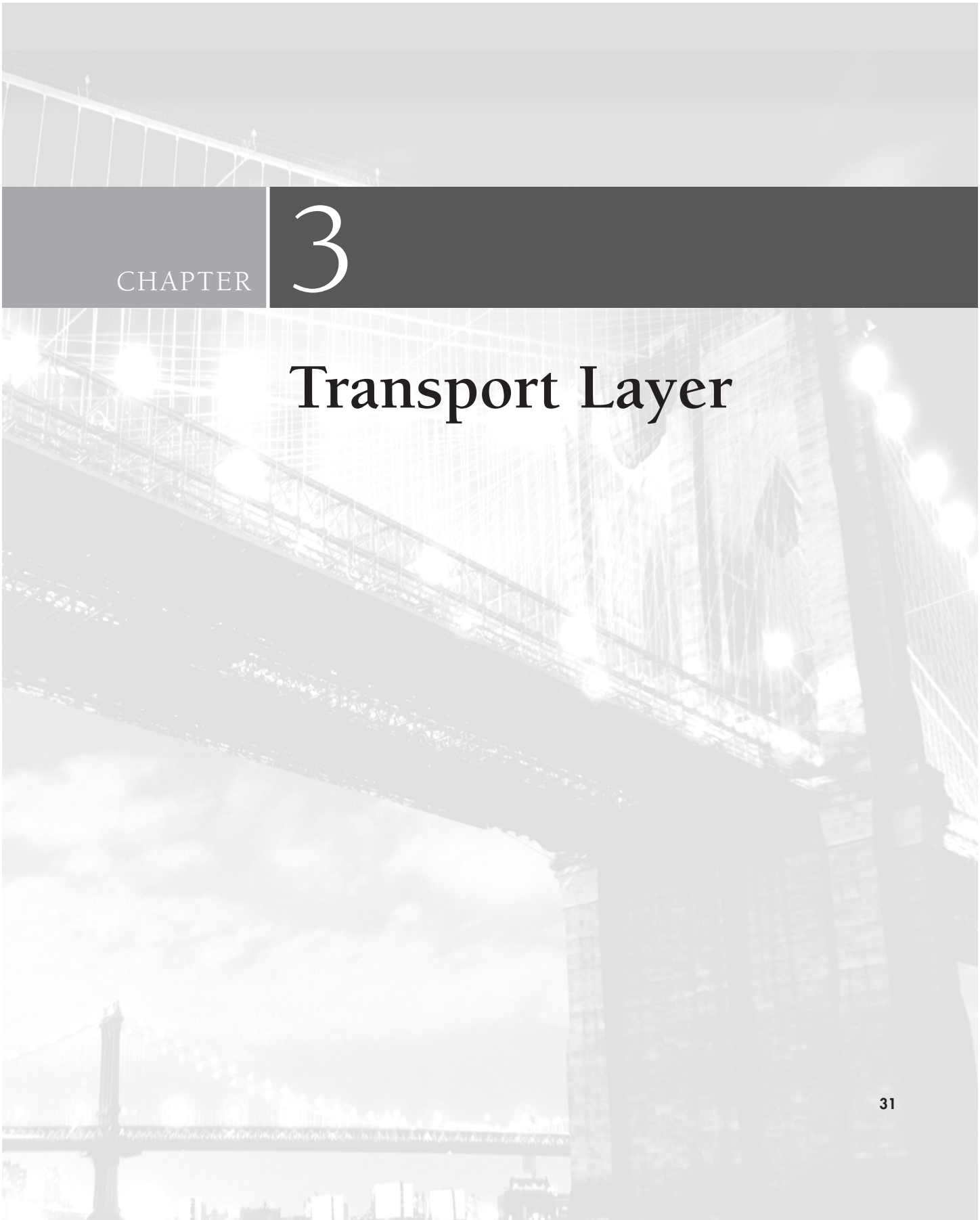


CHAPTER 3

Transport Layer

Most Important Ideas and Concepts from Chapter 3

- **Logical communication between two processes.** Application processes use the logical communication provided by the transport layer to send messages to each other, free from the worries of the details of the network infrastructure used to carry these messages. Whereas a transport-layer protocol provides logical communication between processes, a network-layer protocol provides logical communication between hosts. This distinction is important but subtle; it is explained on page 186 of the textbook with a cousin/house analogy. An application protocol lives only in the end systems and is not present in the network core. A computer network may offer more than one transport protocol to its applications, each providing a different service model. The two transport-layer protocols in the Internet—UDP and TCP—provide two entirely different service models to applications.
- **Multiplexing and demultiplexing.** A receiving host may be running more than one network application process. Thus, when a receiving host receives a packet, it must decide to which of its ongoing processes it is to pass the packet's payload. More precisely, when a transport-layer protocol in a host receives a segment from the network layer, it must decide to which socket it is to pass the segment's payload. The mechanism of passing the payload to the appropriate socket is called *demultiplexing*. At the source host, the job of gathering data chunks from different sockets, adding header information (for demultiplexing at the receiver), and passing the resulting segments to the network layer is called *multiplexing*.
- **Connectionless and connection-oriented demultiplexing.** Every UDP and TCP segment has a field for a source port number and another field for a destination port number. Both UDP (connectionless) and TCP (connection-oriented) use the values in these fields—called port numbers—to perform the multiplexing and demultiplexing functions. However, UDP and TCP have important, but subtle differences in how they do multiplexing and demultiplexing. In UDP, each UDP socket is assigned a port number, and when a segment arrives at a host, the transport layer examines the destination port number in the segment and directs the segment to the corresponding socket. On the other hand, a TCP socket is identified by the four-tuple: (source IP address, source port number, destination IP address, destination port number). When a TCP segment arrives from the network to a host, the host uses all four values to direct (demultiplex) the segment to the appropriate socket.
- **UDP.** The Internet (and more generally TCP networks) makes available two transport-layer protocols to applications: UDP and TCP. UDP is a no-frills, bare-bones protocol, allowing the application to talk almost directly with the network layer. The only services that UDP provides (beyond IP) is multiplexing/demultiplexing and some light error checking. The UDP segment has only four header fields: source port number, destination port number, length of the segment, and checksum. An application may choose UDP for a transport protocol for one or more of the following reasons: it offers finer application control of what data is sent in a segment

and when; it has no connection establishment; it has no connection state at servers; and it has less packet header overhead than TCP. DNS is an example of an application protocol that uses UDP. DNS sends its queries and answers within UDP segments, without any connection establishment between communicating entities.

- **Reliable data transfer.** Recall that, in the Internet, when the transport layer in the source host passes a segment to the network layer, the network layer does not guarantee it will deliver the segment to the transport layer in the destination host. The segment could get lost and never arrive at the destination. For this reason, the network layer is said to provide *unreliable data transfer*. A transport-layer protocol may nevertheless be able to guarantee process-to-process message delivery even when the underlying network layer is unreliable. When a transport-layer protocol provides such a guarantee, it is said to provide *reliable data transfer* (*RDT*). The basic idea behind reliable data transfer is to have the receiver acknowledge the receipt of a packet; and to have the sender retransmit the packet if it does not receive the acknowledgement. Because packets can have bit errors as well as be lost, RDT protocols are surprisingly complicated, requiring acknowledgements, timers, checksums, sequence numbers, and acknowledgement numbers.
- **Pipelined reliable data transfer.** The textbook incrementally develops an RDT *stop-and-wait* protocol in Section 3.4. In a stop-and-wait protocol, the source sends one packet at a time, only sending a new packet once it has received an acknowledgment for the previous packet. Such a protocol has very poor throughput performance, particularly if either the transmission rate, R, or the round-trip time, RTT, is large. In a pipelined protocol, the sender is allowed to send multiple packets without waiting for an acknowledgment. Pipelining requires an increased range in sequence numbers and additional buffering at sender and receiver. The textbook examined two pipelined RDT protocols in some detail: Go-Back-N (GBN) and Selective Repeat (SR). Both protocols limit the number of outstanding unacknowledged packets the sender can have in the pipeline. GBN uses cumulative acknowledgments, only acknowledging up to the first non-received packet. A single-packet error can cause GBN to retransmit a large number of packets. In SR, the receiver individually acknowledges correctly received packets. SR has better performance than GBN, but is more complicated, both at sender and receiver.
- **TCP.** TCP is very different from UDP. Perhaps the most important difference is that TCP is reliable (it employs a RDT protocol) whereas UDP isn't. Another important difference is that TCP is connection oriented. In particular, before one process can send application data to the other process, the two processes must "handshake" with each other by sending to each other (a total of) three empty TCP segments. The process initiating the TCP handshake is called the *client*. The process waiting to be hand-shaken is the *server*. After the 3-packet handshake is complete, a connection is said to be established and the two processes can send application data to each other. A TCP connection has a send buffer and a receive buffer. On the send side, the application sends bytes to the send buffer, and TCP grabs bytes from

the send buffer to form a segment. On the receive side, TCP receives segments from the network layer, deposits the bytes in the segments in the receive buffer, and the application reads bytes from the receive buffer. TCP is a byte-stream protocol in the sense that a segment may not contain a single application-layer message. (It may contain, for example, only a portion of a message or contain multiple messages.) In order to set the timeout in its RDT protocol, TCP uses a dynamic RTT estimation algorithm.

TCP's RDT service ensures that the byte stream that a process reads out of its receive buffer is exactly the byte stream that was sent by the process at the other end of the connection. TCP uses a pipelined RDT with cumulative acknowledgments, sequence numbers, acknowledgment numbers, a timer, and a dynamic timeout interval. Retransmissions at the sender are triggered by two different mechanisms: timer expiration and triple duplicate acknowledgments.

- **Flow control.** Because a connection's receive buffer can hold only a limited amount of data, there is the danger that the buffer overflows if data enters the buffer faster than it is read out of the buffer. Many transport protocols, including TCP, use flow control to prevent the occurrence of buffer overflow. The idea behind flow control is to have the receiver tell the sender how much spare room it has in its receive buffer; and to have the sender restrict the amount of data that it puts in the pipeline to be less than the spare room. Flow control speeds matching: it matches the sender's send rate to the receiver's read rate.
- **Congestion control principles.** Congestion has several costs. Large queuing delays occur as the packet arrival rate nears the link capacity. Unneeded retransmissions by the sender in the face of large delays cause routers to use their link bandwidth to forward unneeded copies of packets. And, when a packet is dropped along a path, the transmission capacity that was used at each of the upstream links to forward that packet (up to point at which it is dropped) is wasted.
- **TCP congestion control.** Because the IP layer provides no explicit feedback to end systems regarding network congestion, TCP uses end-to-end congestion control rather than network-assisted congestion control. The amount of data a TCP connection can put into the pipe is restricted by the sender's congestion window. The congestion window essentially determines the send rate. Unlike the simpler GBN and SR protocols covered in Section 3.4, this window is dynamic. TCP reduces the congestion window during the occurrence of a loss event, where a loss event is either a timeout or the receipt of three duplicate acknowledgements. When loss events are not occurring, TCP increases its congestion window. This gives rise to the sawtooth dynamics for the congestion window, as shown in Figure 3.50 on page 267 of the textbook. The exact rules for how the loss events influence the congestion window are determined by three mechanisms: Additive Increase Multiplicative Decrease (AIMD); slow start; and fast retransmit.

Review Questions

This section provides additional study questions. Answers to each question are provided in the next section.

1. **Logical communication between processes.** Suppose the network layer provides the following service. The source host accepts from the transport layer a segment of maximum size 1,000 bytes and a destination host address. The network layer then guarantees delivery of the segment to the transport layer at the destination host. Suppose many network application processes can be running at the destination host.
   a. Design the simplest possible transport layer protocol that will get application data to the desired process at the destination host. Assume the operating system in the destination host has assigned a two-byte port number to each running application process.
   b. Modify this protocol so that it provides a "return address" to the destination process.
   c. In your protocols, does the transport layer "have to do anything" in the core of the computer network?
2. **Logical communication between families.** Consider a planet where everyone belongs to a family of five, every family lives in its own house, each house has a unique address, and each person in a house has a unique name. Suppose this planet has a mail service that delivers letters from source house to destination house. The mail service requires that (i) the letter be in an envelope and that (ii) the address of the destination house (and nothing more) be clearly written on the envelope. Suppose each family has a delegate family member who collects and distributes letters for the other family members. The letters do not necessarily provide any indication of the recipients of the letters.
   a. Using the solution to Question 1 as inspiration, describe a protocol that the delegates can use to deliver letters from a sending family member to a receiving family member.
   b. In your protocol, does the mail service ever have to open the envelope and examine the letter in order to provide its service?
3. **UDP demultiplexing.** Suppose a process in host C has a UDP socket with port number 787. Suppose host A and host B each send a UDP segment to host C with destination port number 787. Will both of these segments be directed to the same socket at host C? If so, how will the process at host C know that these segments originated from two different hosts?
4. **UDP and TCP checksum.**
   a. Suppose you have the following two bytes: 00110101 and 01101001. What is the 1s complement of these two bytes?

b. Suppose you have the following two bytes: 11110101 and 01101001. What is the 1s complement of these two bytes?

c. For the bytes in part a), give an example where one bit is flipped in each of the two bytes and yet the 1s complement doesn't change.

5. **TCP multiplexing.** Suppose that a Web server runs in host C on port 80. Suppose this Web server uses persistent connections, and is currently receiving requests from two different hosts: A and B. Are all of the requests being sent through the same socket at host C? If they are being passed through different sockets, do both of the sockets have port 80? Discuss and explain.

6. **Why choose UDP?** An application may choose UDP for a transport protocol because UDP offers finer application control (than TCP) of what data is sent in a segment and when it is sent.

   a. Why does an application have more control of what data is sent in a segment?

   b. Why does an application have more control of when the segment is sent?

7. **A simple synchronized message exchange protocol.** Consider two network entities: A and B, which are connected by a perfect bi-directional channel (that is, any message sent will be received correctly; the channel will not corrupt, lose, or re-order packets). A and B are to deliver data messages to each other in an alternating manner: first A must deliver a message to B, then B must deliver a message to A, then A must deliver a message to B, and so on. Draw a FSM specification for this protocol (one FSM for A and one FSM for B). Don't worry about a reliability mechanism here; the main point is to create a FSM specification that reflects the synchronized behavior of the two entities. You should use the following events and actions, which have the same meaning as protocol rdt1.0, shown on page 204 of the textbook:

   ```
   rdt_send(data), packet = make_pkt(data),
   udt_send(packet), rdt_rcv(packet), extract (packet,data),
   deliver_data(data).
   ```

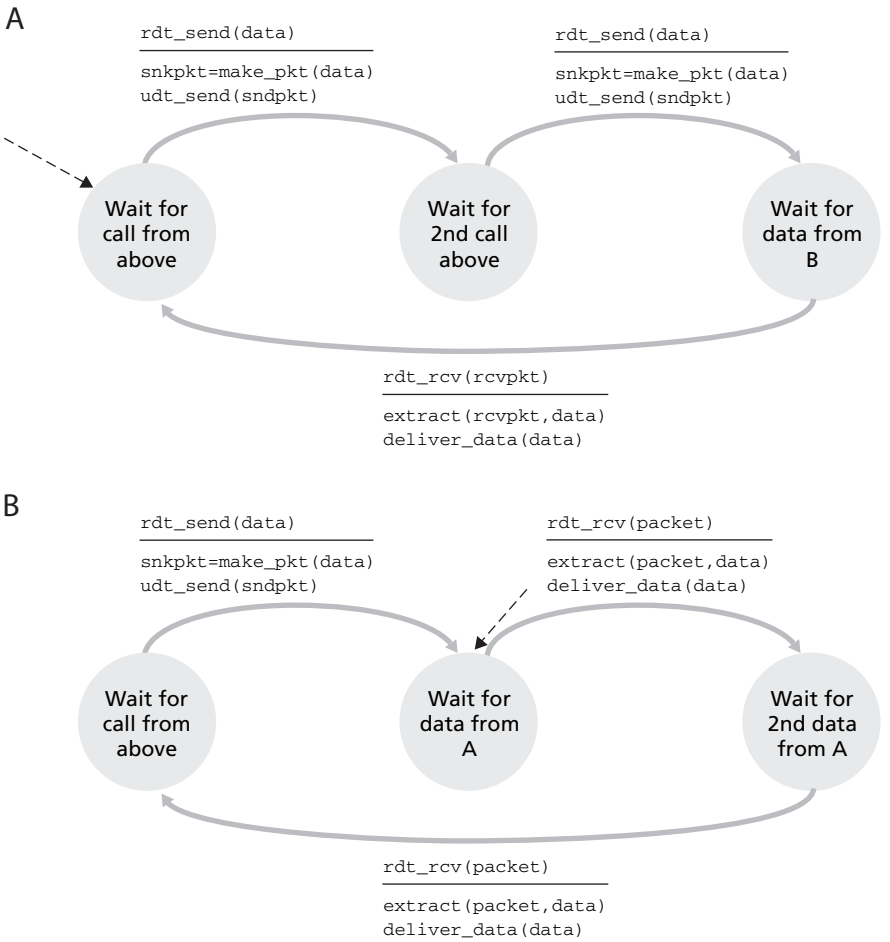
   Make sure that your protocol reflects the strict alternation of sending between A and B. Also, be sure to indicate the initial states for A and B in your FSM description.

8. **Adding ACKs to the simple synchronized message exchange protocol.** Let's modify the protocol in Question 7: After receiving a data message from the other entity, an entity should send an explicit acknowledgement back to the other side. An entity should not send a new data item until after it (i) has received an ACK for its most recently sent message, (ii) has received a data message from the other entity, and (iii) ACKed that message received from the other entity. Draw the FSM specification for this modified protocol. You may use the new function that was introduced in protocol rdt2.0, shown on

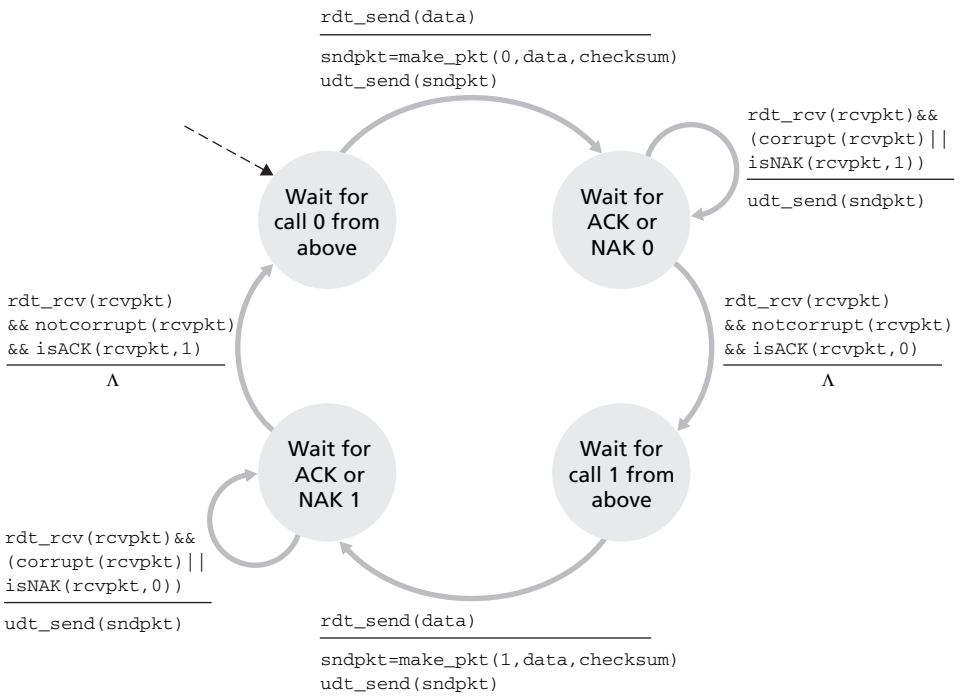
page 206 in the textbook: `udt_send(ACK)`. You may want to use (but do not have to, depending on your solution) the following event as well: `rdt_rcv(rcvpkt) && isACK(rcvpkt)`, which indicates the receipt of an ACK packet (as in rdt2.0 in the textbook), and `rdt_rcv(rcvpkt) && isDATA(rcvpkt)`, which indicates the receipt of a data packet.

9. **A simple three-node synchronized message exchange protocol.** Consider three network entities: A, B, and C, which are connected by perfect point-to-point bi-directional channels (that is, any message sent will be received correctly; the channel will not corrupt, lose, or re-order packets). A, B, and C are to deliver data messages to each other in a rotating manner: first A must deliver a message to B, then B must deliver a message to C, and then C must deliver a message to A, and so on. Draw a FSM specification for this protocol (one FSM for A, one for B, and one for C). You should use the same events as in Questions 7 and 8, except that the `udtsend()` function now includes the name of the recipient. For example, `udt_send(data,B)` is used to send a message to B. Your protocol does *not* have to use ACK messages.

10. **A final variation on the simple synchronized message exchange protocol.** Consider (yet again!) two network entities: A and B, which are connected by a bi-directional channel that is perfect (that is, any message sent will be received correctly; the channel will not corrupt, lose, or re-order packets). A and B are to deliver data messages to each other in the following manner: A is to send *two* messages to B, and then B is to send one message to A. Then the cycle repeats. You should use the same events as in the questions above. Your protocol does *not* have to use ACK messages. (The key idea is to think about how to use states to track how many messages A has sent: one or two.)



11. **Reliable data transfer.** Recall the NAK-free `rdt` protocol in the textbook (figure 3.13) for a channel with bit errors (but without packet loss). The FSM protocol is shown below. How many states does the receiver need: 1, 2, or 4? What are these states? Without looking at the textbook, draw the FSM for the corresponding receiver. (Hint: the receiver must now include the sequence number of the packet being acknowledged.)



12. **Designing a minimalist data transfer protocol.** Chapter 3 shows a number of mechanisms used to provide for reliable data transfer:

    - checksum
    - ACKs
    - timers
    - sequence numbering

    Consider a sender and receiver that are connected by a sender-to-receiver channel that can corrupt and lose packets. The receiver-to-sender channel is perfect (that is, it will not lose or corrupt packets). The delay on each channel is known to always be less than some maximum value, *d*. Neither channel will reorder packets. (Note: re-read the channel properties just described and make sure you understand them!) Design a reliable data transfer protocol for this scenario *using only those mechanisms (among the four listed above) that are absolutely required*. That is, if you use a mechanism that is not required, you will not receive full credit, even if the protocol works. Your protocol should be as simple as possible but have the functionality to deliver data reliably under the stated assumptions. Your solution does not need to be efficient; it must work correctly.

a. Draw the sender and receiver FSMs.

b. For each of the mechanisms (from among the four listed above) that you use in your protocol, explain the role/purpose of the mechanism and why you cannot get by without it. (Note: this does not imply that your protocol will use all four mechanisms above—maybe it does; maybe it does not. However, you must explain why you need the mechanisms that you have chosen to include.)

13. **Pipelined reliable data transfer.** Recall the Go-back-N protocol in Section 3.4.

    a. Does this protocol have a timer for each unacknowledged packet?

    b. When a timer expires, what happens?

    c. Use the interactive applet for Go-Back-N and quickly try to generate seven packets. How many packets did you generate? Just after attempting to generate the seven packets, pause the animation and kill the first packet. What happens when the timeout expires?

14. **Pipelined reliable data transfer with selective repeat.** Recall the selective repeat protocol in Section 3.4.

    a. Does this protocol have a timer for each unacknowledged packet?

    b. When an acknowledgement arrives for the oldest unacknowledged packet, what happens?

15. **Can a window size be too large for the sequence number space?** Consider the Go-Back-N protocol. Suppose that the size of the sequence number space (number of unique sequence numbers) is N, and the window size is N. Show (give a timeline trace showing the sender, receiver, and the messages they exchange over time) that the Go-Back-N protocol will not work correctly in this case.

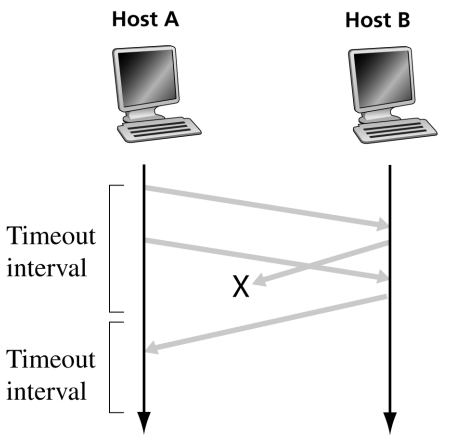
16. **TCP sequence numbers.** Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 144. Suppose that Host A then sends two segments to Host B back-to-back. The first and second segments contain 20 and 40 bytes of data, respectively. In the first segment, the sequence number is 145, source port number is 303, and the destination port number is 80. Host B sends an acknowledgement whenever it receives a segment from Host A.

    a. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?

    b. If the first segment arrives before the second segment, in the acknowledgement of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?

    c. If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment, what is the acknowledgment number?

d. Suppose the two segments sent by A arrive in order at B. The first acknowledgement is lost and the second segment arrives after the first timeout interval, as shown in the figure below. Complete the diagram, showing all other segments and acknowledgements sent. (Assume there is no additional packet loss.) For each segment you add to the diagram, provide the sequence number and number of bytes of data; for each acknowledgement that you add, provide the acknowledgement number.



17. **Round-trip time estimation.** Let $\alpha = 0.2$. Suppose for a given TCP connection three acknowledgments have been returned with RTTs: RTT for first ACK = 80 msec; RTT for second ACK = 60 msec; and RTT for third ACK = 100 msec. Determine the value of EstimatedRTT after each of the three acknowledgments.
18. **Flow control.** Host A and B are directly connected with a 100 Mbps link. There is one TCP connection between the two hosts, and Host A is sending an enormous file to Host B over this connection. Host A can send application data into the link at 50 Mbps but Host B can read out of its TCP receive buffer at a maximum rate of 10 Mbps. Describe the effect of TCP flow control.
19. **TCP connection management.**
    a. A server process in Host B has a welcoming socket at port 977. What will trigger the server process to create a connection socket? What is the source IP address and source port number for this connection socket?
    b. How many bytes is a TCP SYN segment? What flags are set in a TCP SYN segment?
    c. What must happen for Host B to complete this connection?
20. **TCP congestion control.** Consider sending a large file from one host to another over a TCP connection that has no loss.

a. Suppose TCP uses AIMD for its congestion control without slow start. Assuming CongWin increases by 1 MSS every time an ACK is received and assuming approximately constant round-trip times, how long does it take for CongWin to increase from 1 MSS to 5 MSS (assuming no loss events and constant RTT)?

b. What is the average throughput (in terms of MSS and RTT) for this connection up through time = 4 RTT?

21. **More TCP congestion control.** Suppose that in TCP, the sender window is of size N, the base of the window is at sequence number x, and the sender has just sent a complete window's worth of segments. Let RTT be the sender-to-receiver-to-sender round-trip time, and let MSS be the segment size.

a. Is it possible that there are ACK segments in the receiver-to-sender channel for segments with sequence numbers lower than x? Justify your answer.

b. Assuming no loss, what is the throughput (in packets/sec) of the sender-to-receiver connection?

c. Suppose TCP is in its congestion avoidance phase. Assuming no loss, what is the window size after the N segments are ACKed?

22. **TCP Potpourri.**

a. Consider two TCP connections, one between Hosts A (sender) and B (receiver), and another between Hosts C (sender) and D (receiver). The RTT between A and B is half that of the RTT between C and D. Suppose that the senders' (A's and C's) congestion window sizes are identical. Is their throughput (number of segments transmitted per second) the same? Explain.

b. Now suppose that the *average* RTT between A and B, and C and D are identical. The RTT between A and B is constant (never varies), but the RTT between C and D varies considerably. Will the TCP timer values of the two connections differ, and if so, how are they different, and why are they different?

c. Give one reason why TCP uses a three-way (SYN, SYNACK, ACK) handshake rather than a two-way handshake to initiate a connection.

d. It is said that a TCP connection "probes" the network path it uses for available bandwidth. What does this mean?

e. What does it mean when we say that TCP uses "cumulative acknowledgement"? Give two reasons why cumulative acknowledgment is advantageous over selective acknowledgment.

Answers to Review Questions

1. a. Call this protocol Simple Transport Protocol (STP). At the sender side, STP accepts from the sending process a chunk of data not exceeding 998 bytes, a destination host address, and a destination port number. STP adds a two-byte header to each chunk and puts the port number of the destination process in this header. STP then gives the destination host address and the resulting segment to the network layer. The network layer delivers the segment to STP at the destination host. STP then examines the port number in the segment, extracts the data from the segment, and passes the data to the process identified by the port number.

   b. The segment now has two header fields: a source port field and a destination port field. At the sender side, STP accepts a chunk of data not exceeding 996 bytes, a destination host address, a source port number, and a destination port number. STP creates a segment that contains the application data, source port number, and destination port number. Then it gives the segment and the destination host address to the network layer. After receiving the segment, STP at the receiving host gives the application process the application data and the source port number.

   c. No, the transport layer does not have to do anything in the core; the transport layer "lives" in the end systems.

2. a. For sending a letter, the family member is required to give the delegate the letter itself, the address of the destination house, and the name of the recipient. The delegate clearly writes the recipient's name on the top of the letter. Then the delegate puts the letter in an envelope and writes the address of the destination house on the envelope. Then the delegate gives the letter to the planet's mail service. At the receiving side, the delegate receives the letter from the mail service, takes the letter out of the envelope, and notes the recipient name written at the top of the letter. Then the delegate gives the letter to the family member with this name.

   b. No, the mail service does not have to open the envelope; it only examines the address on the envelope.

3. Yes, both segments will be directed to the same socket. For each received segment, at the socket interface, the operating system will provide the process with the IP address of the host that sent the segment. The process can use the supplied IP addresses to determine the origins of the individual segments.

4. a. Adding the two bytes gives 10011110. Taking the 1s complement gives 01100001.

   b. Adding the two bytes gives 01011111. The 1s complement gives 10100000.

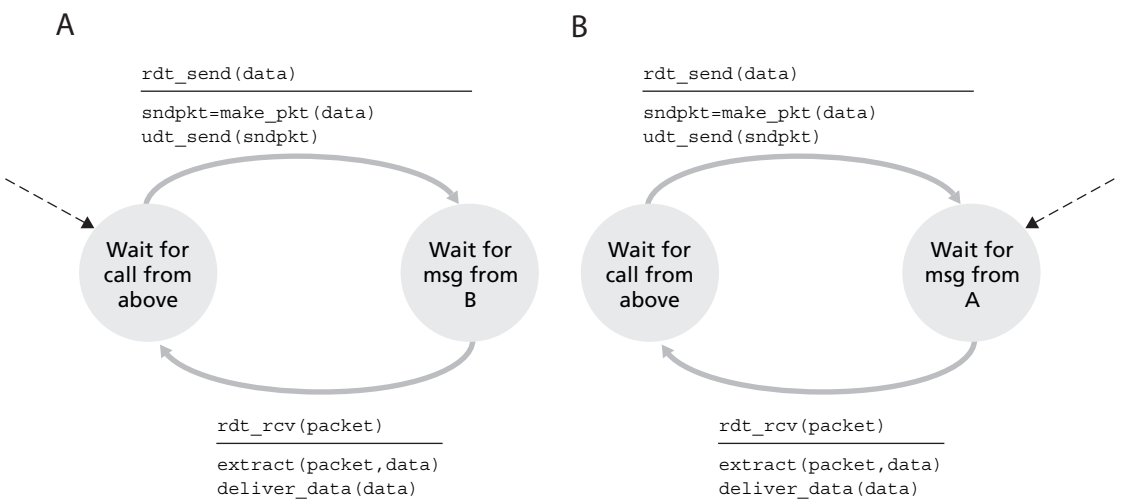
c. First byte = 00110001; second byte = 01101101.

5. For each persistent connection, the Web server creates a separate "connection socket." Each connection socket is identified with a four-tuple: (source IP address, source port number, destination IP address, destination port number). When Host C receives an IP datagram, it examines these four fields in the datagram/segment to determine to which socket it should pass the payload of the TCP segment. Thus, the requests from A and B pass through different sockets. The identifier for both of these sockets has 80 for the destination port; however, the identifiers for these sockets have different values for the source IP addresses. Unlike UDP, when the transport layer passes a TCP segment's payload to the application process, it does not specify the source IP address, as this is implicitly specified by the socket identifier.

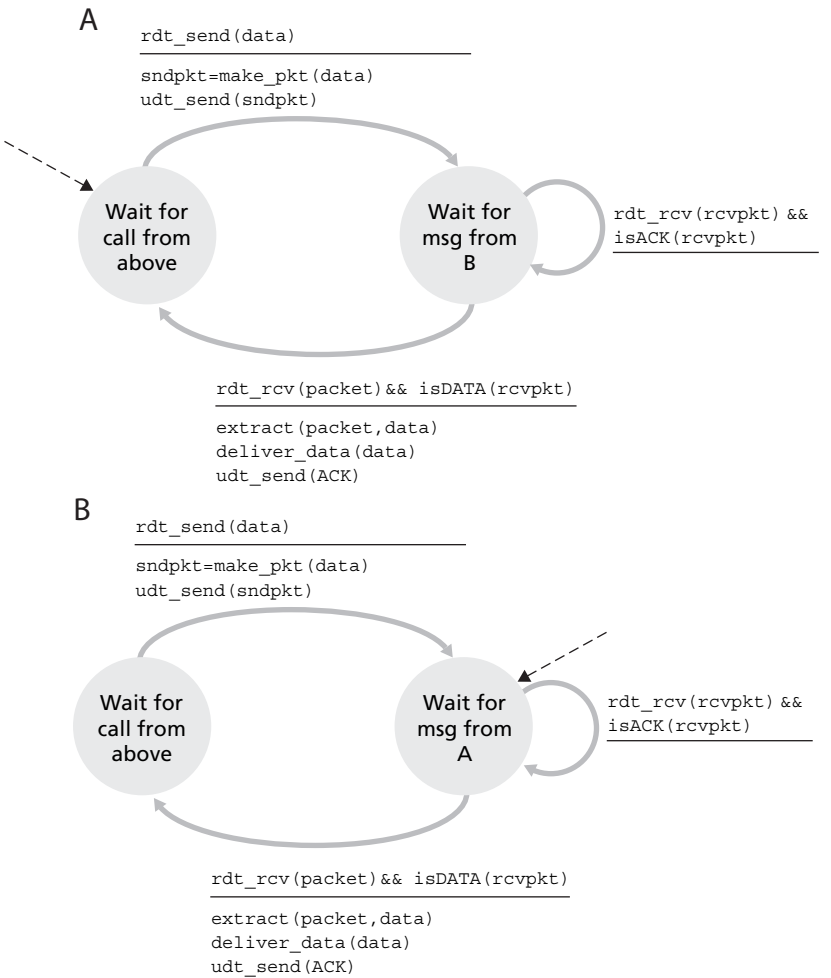
6. a. Consider sending an application message over a transport protocol. With TCP, the application writes data to the connection's send buffer and TCP will grab bytes without necessarily putting a single message in the TCP segment; TCP may put more or less than a single message in a segment. UDP, on the other hand, encapsulates in a segment whatever the application gives it; so that, if the application gives UDP an application message, this message will be the payload of the UDP segment. Thus, with UDP, an application has more control of what data is sent in a segment.

   b. With TCP, due to flow control and congestion control, there may be significant delay from the time when an application writes data to its send buffer until when the data is given to the network layer. UDP does not have delays due to flow control and congestion control.

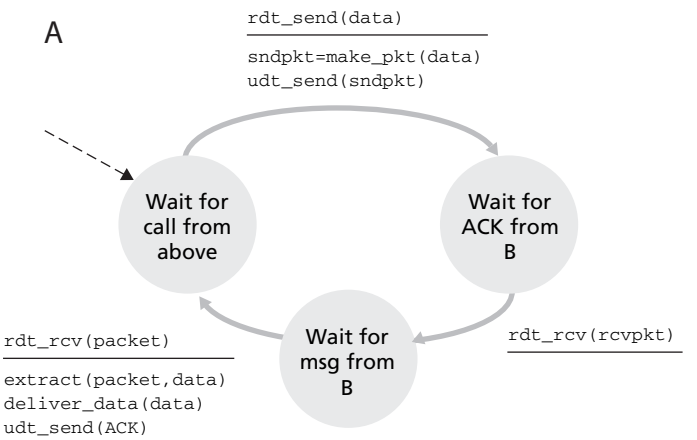
7.



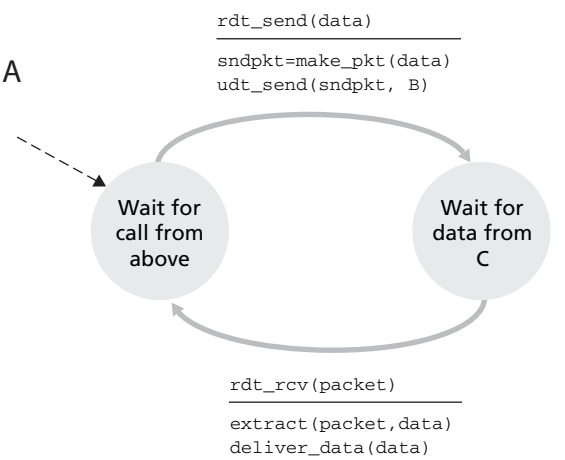
8.



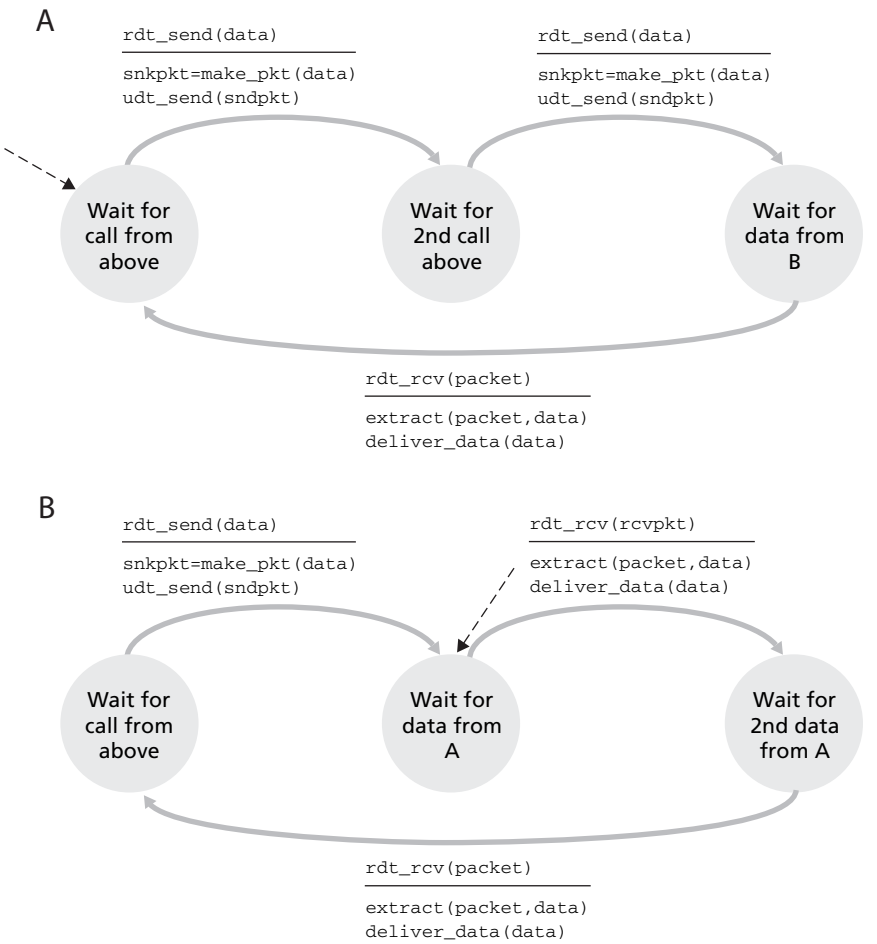
In the solution above, we've used the isACK(rcvpkt) and isDATA(rcvpkt) expressions to indicate whether an ACK or DATA message has been received. If we add another state to each FSM, which is used to reflect whether the entity is waiting for an ACK or waiting for DATA, then we do not need to use these expressions. Below is an alternate solution (for A only; B is similar) that does not use these expressions. The solution above and the solution below are equally good, they differ only in how they represent the handling of the received message. Indeed, they are FSMs for the same protocol!



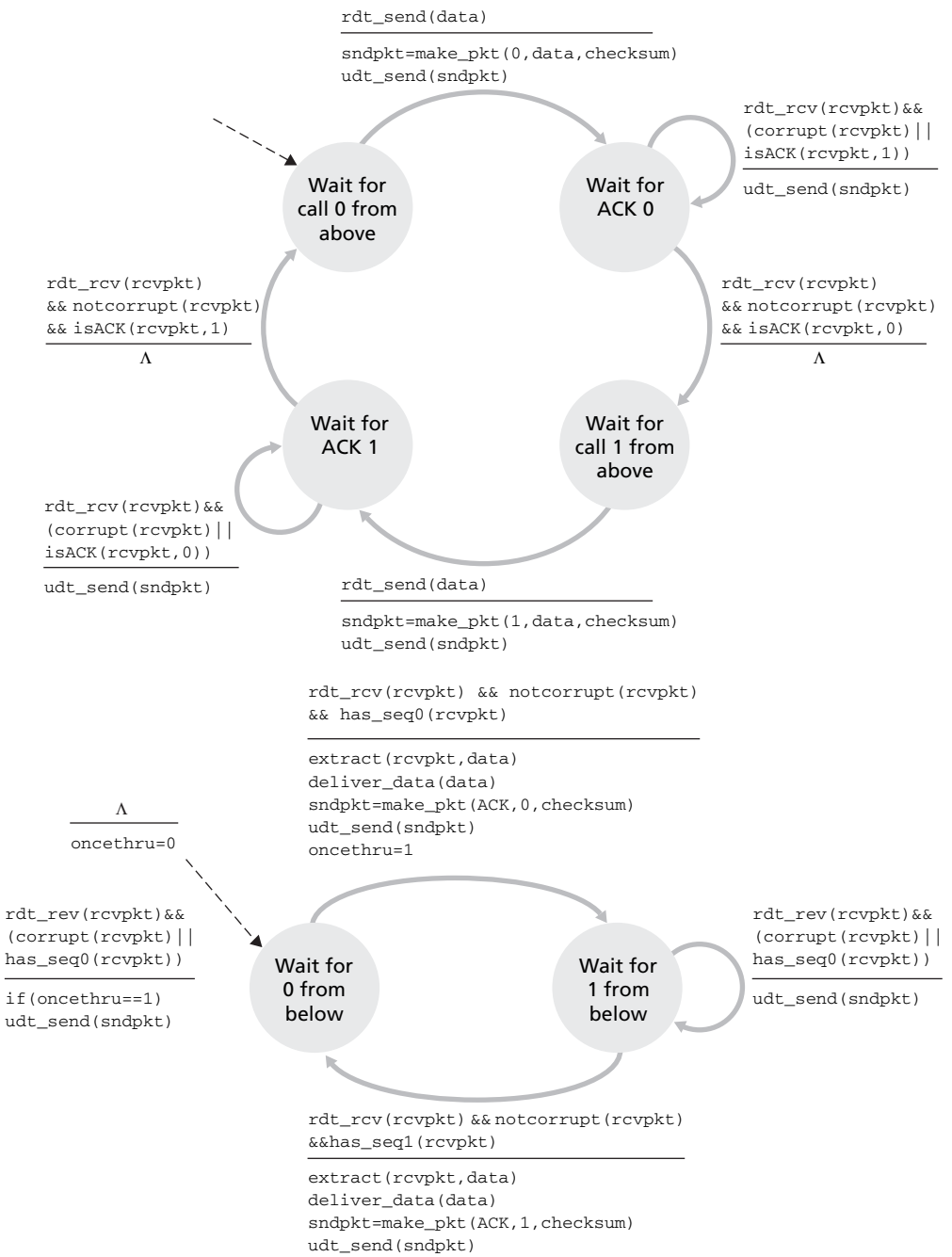
9. The answer to this question is essentially the same as for the first of these FSM questions, except that we need to indicate the address of the outgoing message. Because we have specified that the channels are point-to-point, A does not see any communication between B and C. Thus, A only needs to send its message to B, and then wait for a message from C before sending another message to B. The FSM for A is shown below. (Note that if we had specified that the channel connecting the entities was a broadcast channel, then we would have had to consider the fact that A was receiving messages sent from B to C, which A would receive but then would have to ignore.)



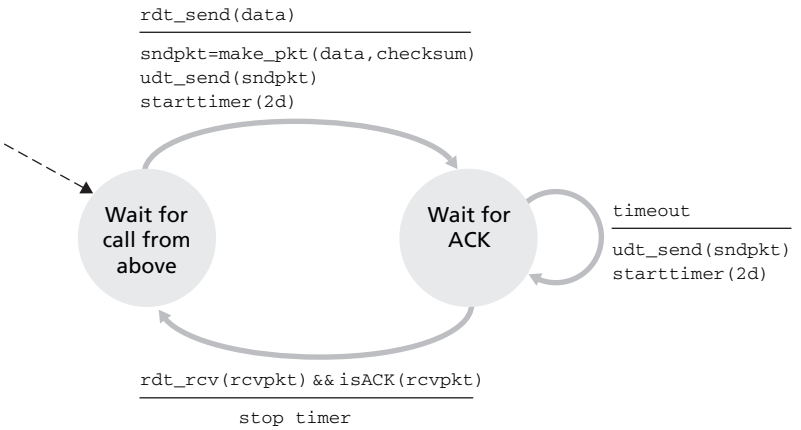
10.


A
rdt_send(data)
snkpkt=make_pkt(data)
udt_send(sndpkt)
rdt_send(data)
snkpkt=make_pkt(data)
udt_send(sndpkt)
Wait for call from above
Wait for 2nd call above
Wait for data from B
rdt_rcv(packet)
extract(packet,data)
deliver_data(data)
B
rdt_send(data)
snkpkt=make_pkt(data)
udt_send(sndpkt)
rdt_rcv(rcvpkt)
extract(packet,data)
deliver_data(data)
Wait for call from above
Wait for data from A
Wait for 2nd data from A
rdt_rcv(packet)
extract(packet,data)
deliver_data(data)

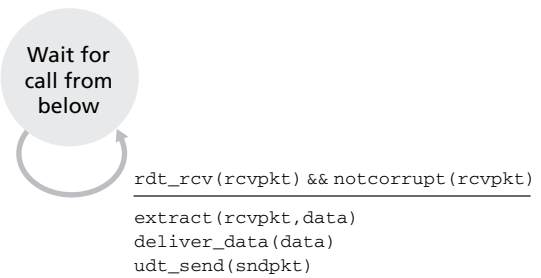
11. This protocol needs two states: wait for zero from below; wait for one from below. The FSM for the receiver is given below.



12. a.



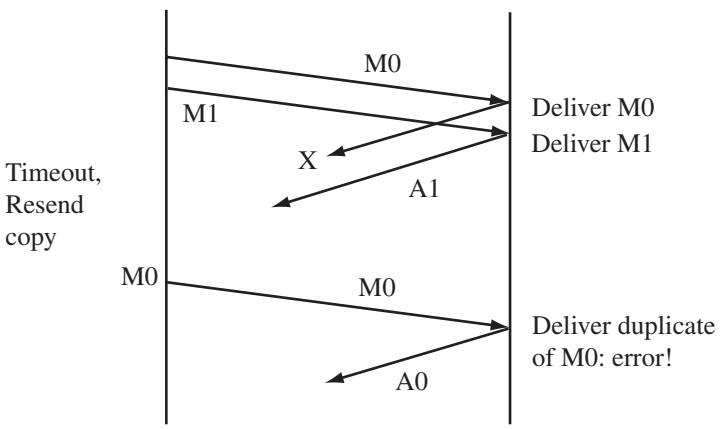
a. sending side



b. receiving side

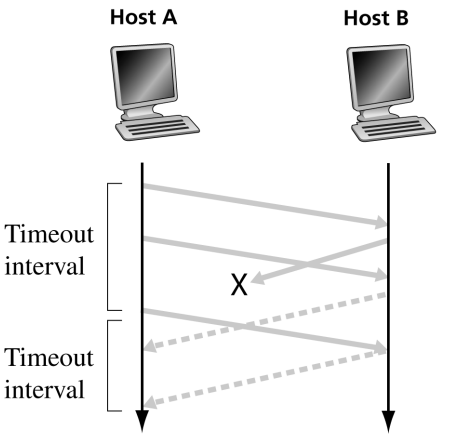
b. Because the sender-to-receiver channel can corrupt packets, the data sent on the sender-to-receiver channel will need a *checksum* to detect bit errors. Because the sender-to-receiver channel can lose packets, we will need to have a *timer* to timeout and retransmit packets that have not been received by the receiver. The receiver will need to indicate which packets it has received by using an ACK *message*; if a packet is not received or is received corrupted, no ACK is sent. Because the maximum delay of the channel is bounded at, d the sender can set its timeout value to $2d$, and therefore only retransmit when it is certain that a retransmission is needed (and expected by the receiver). Thus, there is no need for sequence numbers, since there will be no unneeded (and unexpected at the receiver) retransmissions. .

13. a. No, GBN has only one timer, for the oldest unacknowledged packet.
    b. When the timer expires, the sender resends all packets that have been sent but have not yet been acknowledged.
    c. The applet only generates six packets since the window size is six. The sender doesn't receive an acknowledgment for previously unacknowledged data before the timer expires. When the timer expires, the sender resends all six packets.
14. a. Yes, there is a timer for each unacknowledged packet.
    b. The window advances. If the sender has another packet to send, it sends the packet and starts a timer for the packet.
15. Suppose that the sequence number space is 0,1 and N = 2, that is, that two messages can be transmitted but not yet acknowledged. The timeline below shows an error that can occur.



16. a. The first and second segments contain 20 and 40 bytes of data, respectively. In the second segment sent from A to B, the sequence number is 165, the source port number is 303, and the destination port number is 80.
    b. The first acknowledgment has acknowledgment number 165, source port 80, and destination port 303.
    c. The acknowledgment number will be 145, indicating that it is still waiting for bytes 145 and onward.

d. The sequence number of the retransmission is 145 and it carries 20 bytes of data. The acknowledgment number of the additional acknowledgment is 205.



17. After the first ACK, the EstimatedRTT is equal to the RTT associated with the ACK, namely, 80 msec. After the second ACK, we use the following formula:

$$\text{EstimatedRTT} = (1 - \alpha)\,\text{EstimatedRTT} + \alpha\,\text{SampleRTT}$$

to obtain:

$$\text{EstimatedRTT} = (0.8)(80\text{ msec}) + (0.2)(60\text{ msec}) = 76\text{ msec}$$

Similarly, after third ACK, we get

$$\text{EstimatedRTT} = (0.8)(76\text{ msec}) + (0.2)(100\text{ msec}) = 71.2\text{ msec}$$

18. Host A sends data into the receive buffer faster than Host B can remove data from the buffer. The receive buffer fills up at a rate of roughly 40 Mbps. When the buffer is full, Host B signals to Host A to stop sending data by setting RcvWindow = 0. Host A then stops sending until it receives a TCP segment with RcvWindow > 0. Host A will thus repeatedly stop and start sending as a function of the RcvWindow values it receives from Host B. On average, the long-term rate at which host A sends data to host B as part of this connection is no more than 10 Mbps.

19. a. When Host B receives a TCP SYN segment with destination port number 977, the operating system at Host B will create a (half-open) connection socket. The TCP SYN packet has a source port number, which becomes the source port number of the socket. The TCP SYN segment is also contained in an IP datagram, which has a source IP address, which in turn becomes the source IP address for the socket.

b. A TCP SYN packet contains no data and is thus 20 bytes. In a SYN segment, the SYN flag is set, but not the ACK flag.

c. After receiving the SYN packet, the server sends to the client on Host B a SYNACK segment, which is also 20 bytes, and which has both the SYN and ACK flags set. The client then sends an ACK packet back to the server. Upon receiving this ACK packet, the connection is fully open at both the client and server sides.

20. a. It takes 1 RTT to increase CongWin to 2 MSS; 2 RTTs to increase to 3 MSS; 3 RTTs to increase to 4 MSS; and 4 RTTs to increase to 5 MSS.

b. In the first RTT 1 MSS was sent; in the second RTT 2 MSS were sent; in the third RTT 3 MSS were sent; in the fourth RTT 4 MSS were sent. Thus, up to time 4 RTT, $1 + 2 + 3 + 4 = 10$ MSS were sent (and acknowledged). Thus, one can say that the average throughput up to time 4 RTT was (10 MSS)/(4 RTT) $= 2.5$ MSS/RTT.

21. a. It is possible. Suppose that the window size is $N = 1$. The sender sends packet $x - 1$, which is delayed and so it timeouts and retransmits $x - 1$. There are now two copies of $x - 1$ in the network. The receiver receives the first copy of $x - 1$ and ACKs. The receiver then receives the second copy of $x - 1$ and ACKs. The sender receives the first ACK and sets its window base to x. At this point, there is still an ACK for $x - 1$ propagating back to the sender.

b. Assume that N is measured in segments. The sender can thus send N segments, each of size MSS bytes every RTT sec. The throughput is $N \cdot MSS/RTT$.

c. $N + 1$

22. a. No. The two sessions will transmit the same number of segments per RTT. But since the RTT of the A-B connection is half that of the other session, its throughput will be twice as large.

b. The TCP timer takes the estimate of the RTT and adds on a factor to account for the variation in RTTs. Therefore, the C-D connection timeout value will be larger.

c. Suppose a client transmits multiple SYN messages that take a long time to be received at the server, so the client terminates (thinking the server is dead). The server then accepts these SYN connections (with only a two-way handshake, the server needs to commit as soon as the SYN is received). However, the client side is no longer present, so the server now has multiple connections opened with no client on the other side.

d. TCP's sawtooth behavior results from TCP continuing to increase its transmission rate until it congests some link in the network (that is, until there is no unused bandwidth on that link) at which point a loss occurs. TCP then backs off and continues to increase its bandwidth again.

e. An acknowledgement of X in TCP tells the sender that all data up to X has been correctly received. Cumulative ACKs can decrease the amount of ACK overhead. For example, a TCP receiver will wait a short time before ACKing in the hope that the next in-sequence packet will arrive, and then will just generate a single ACK (for the second packet), which will acknowledge both packets. Also even if the receiver separately ACKs packets X and X + 1, if the ACK of X is lost but the ACK of X + 1 is received, the sender will know that X was received by the receiver.