Homework Solutions: Updated September 2001

Solutions for Chapter 1

Problem 1.

There is no single right answer to this question. Many protocols would do the trick. Here's a simple answer below:

```
Messages from ATM machine to Server
Msg name            purpose
--------            -------
HELO <userid>       Let server know that there is a card in the ATM machine
                    ATM card transmits user ID to Server
PASSWD <passwd>     User enters PIN, which is sent to server
BALANCE             User requests balance
WITHDRAWL <amount>  User asks to withdrawl money
BYE                 user all done

Messages from Server to ATM machine (display)
Msg name            purpose
--------            -------
PASSWD              Ask user for PIN (password)
OK                  last requested operation (PASSWD, WITHDRAWL) OK
ERR                 last requested operation (PASSWD, WITHDRAWL) in ERROR
AMOUNT <amt>        sent in response to BALANCE request
BYE                 user done, display welcome screen at ATM

Correct operation:

client                          server

HELO (userid)    --------------> (check if valid userid)
                 <-------------  PASSWD
PASSWD <passwd>  --------------> (check password)
                 <-------------  OK (password is OK)
BALANCE          -------------->
                 <-------------  AMOUNT <amt>
WITHDRAWL <amt>  --------------> check if enough $ to cover withdrawl
                 <-------------  OK
ATM dispenses $
BYE              -------------->
                 <-------------  BYE

In situation when there's not enough money:

HELO (userid)    --------------> (check if valid userid)
                 <-------------  PASSWD
PASSWD <passwd>  --------------> (check password)
                 <-------------  OK (password is OK)
```

```
BALANCE          -------------->
                 <--------------  AMOUNT <amt>
WITHDRAWL <amt>  --------------> check if enough $ to cover withdrawl
                 <--------------  ERR (not enough funds)
error msg displayed
no $ given out
BYE              -------------->
                 <--------------  BYE
```

Problem 2.

a) A circuit-switched network would be well suited to the application described, because the application involves long sessions with predictable smooth bandwidth requirements. Since the transmission rate is known and not bursty, bandwidth can be reserved for each application session circuit with no significant waste. In addition, we need not worry greatly about the overhead costs of setting up and tearing down a circuit connection, which are amortized over the lengthy duration of a typical application session.

b) Given such generous link capacities, the network needs no congestion control mechanism. In the worst (most potentially congested) case, all the applications simultaneously transmit over one or more particular network links. However, since each link offers sufficient bandwidth to handle the sum of all of the applications' data rates, no congestion (very little queueing) will occur.

Problem 3.

a) The time to transmit one packet onto a link is $(L+h)/R$. The time to deliver the first of the M packets to the destination is $Q(L+h)/R$. Every $(L+h)/R$ seconds a new packet from the $M-1$ remaining packets arrives at the destination. Thus the total latency is

$$t_s + (Q + M - 1)(L + h)/R.$$

b) $(Q + M - 1)(L + 2h)/R$.

c) The time required to transmit the message over one link is $(LM + 2h)/R$. The time required to transmit the message over Q links is $Q(LM + 2h)/R$.

d) Because there is no store-and-forward delays at the links, the total delay is

$$t_s + (h + ML)/R.$$

Problem 5.

First assume that F/S is an integer. The delay is

$$\left(\frac{S+h}{R}\right)\left(\frac{F}{S}+1\right) = \frac{1}{R}\left[F + S + h + \frac{Fh}{S}\right],$$

where $h = 40$.

If F/S is not an integer, then the delay is

$$\left(\frac{S+h}{R}\right)\left(\left\lfloor\frac{F}{S}\right\rfloor + 1\right) + \frac{F - \lfloor\frac{F}{S}\rfloor S + h}{R} = \frac{1}{R}\left[F + S + 2h + h\left\lfloor\frac{F}{S}\right\rfloor\right]$$

In general, the delay is

$$\frac{1}{R}\left[F + h + S + h\left\lceil\frac{F}{S}\right\rceil\right]$$

which is approximately equal to

$$\frac{1}{R}\left[F + h + S + \frac{hF}{S}\right].$$

Taking the derivative of $S + \frac{hF}{S}$ and setting to 0 gives

$$S_{min} = \sqrt{Fh}.$$

Problem 6.

a) $d_{prop} = m/s$ seconds.

b) $d_{trans} = L/R$ seconds.

c) $d_{end-to-end} = (m/s + L/R)$ seconds.

d) The bit is just leaving Host A.

e) The first bit is in the link and has not reached Host B.

f) The first bit has reached Host B.

g) Want

$$m = \frac{L}{R}S = \frac{100}{28 \times 10^3}(2.5 \times 10^8) = 893\text{km}.$$

Problem 7.

Consider the first bit in a packet. Before this bit can be transmitted, all of the bits in the packet must be generated. This requires

$$\frac{48 \cdot 8}{64 \times 10^3}\text{sec} = 6\text{msec}.$$

The time required to transmit the packet is

$$\frac{48 \cdot 8}{1 \times 10^6}\text{sec} = 384\mu\text{sec}.$$

Propagation delay = 2 msec.

The delay until decoding is

$$6\text{msec} + 384\mu\text{sec} + 2\text{msec} = 8.384\text{msec}.$$

A similar analysis shows that all bits experience a delay of 8.384 msec.

Problem 8.

a) 10 users can be supported because each user requires one tenth of the bandwidth.

b) $p = 0.1$.

c) $\binom{40}{n}p^n(1-p)^{40-n}$.

d) $1 - \sum_{n=0}^{9}\binom{40}{n}p^n(1-p)^{40-n}$.

We use the central limit theorem to approximate this probability. Let X_j be independent random variables such that $P(X_j = 1) = p$.

$$P(\text{“11 or more users”}) = 1 - P(\sum_{j=1}^{40} X_j \leqslant 10)$$

$$P(\sum_{j=1}^{40} X_j \leqslant 10) = P\left(\frac{\sum_{j=1}^{40} X_j - 4}{\sqrt{40 \cdot 0.1 \cdot 0.9}} \leqslant \frac{6}{\sqrt{40 \cdot 0.1 \cdot 0.9}}\right)$$

$$\approx P(Z \leqslant \frac{6}{\sqrt{3.6}}) = P(Z \leqslant 3.16)$$

$$= 0.999$$

when Z is a standard normal r.v. Thus $P(\text{“10 or more users”}) \approx 0.001$.

Problem 9.

It takes LN/R seconds to transmit the N packets. Thus, the buffer is empty when a batch of N packets arrive.

The first of the N packets has no queueing delay. The 2nd packet has a queueing delay of L/R seconds. The nth packet has a delay of $(n-1)L/R$ seconds.

The average delay is

$$\frac{1}{N}\sum_{n=1}^{N}(n-1)L/R = \frac{L}{R}\frac{1}{N}\sum_{n=0}^{N-1} n = \frac{L}{R}\frac{1}{N}\frac{(N-1)N}{2} = \frac{L}{R}\frac{N-1}{2}$$

Problem 10.

a) The transmission delay is L/R. The total delay is

$$\frac{IL}{R(1-I)} + \frac{L}{R} = \frac{L/R}{1-I}$$

b) Let $x = L/R$.

$$\text{Total delay} = \frac{x}{1 - ax}$$

Problem 11.

a) There are Q nodes (the source host and the $N-1$ routers). Let d^q_{proc} denote the processing delay at the qth node. Let R^q be the transmission rate of the qth link and let $d^q_{trans} = L/R^q$. Let d^q_{prop} be the propagation delay across the qth link. Then

$$d_{end-to-end} = \sum_{q=1}^{Q} [d^q_{proc} + d^q_{trans} + d^q_{prop}].$$

b) Let d^q_{queue} denote the average queueing delay at node q. Then

$$d_{end-to-end} = \sum_{q=1}^{Q} [d^q_{proc} + d^q_{trans} + d^q_{prop} + d^q_{queue}].$$

Problem 12.

The command:

```
traceroute -q 20 www.eurecom.fr
```

will get 20 delay measurements from the issuing host to the host, www.eurecom.fr. The average and standard deviation of these 20 measurements can then be collected. Do you see any differences in your answers as a function of time of day?

Solutions for Chapter 2

Problem 1.

a) F

b) T

c) F

d) F

Problem 2.

Access control commands:

USER, PASS, ACT, CWD, CDUP, SMNT, REIN, QUIT.

Transfer parameter commands:

PORT, PASV, TYPE STRU, MODE.

Service commands:

RETR, STOR, STOU, APPE, ALLO, REST, RNFR, RNTO, ABOR, DELE, RMD, MRD, PWD, LIST, NLST, SITE, SYST, STAT, HELP, NOOP.

Problem 3.

SFTP: 115, NNTP: 119.

Problem 4.

The total amount of time to get the IP address is

$$RTT_1 + RTT_2 + \cdots + RTT_n.$$

Once the IP address is known, RTT_o elapses to set up the TCP connection and another RTT_o elapses to request and receive the small object. The total response time is

$$2RTT_o + RTT_1 + RTT_2 + \cdots + RTT_n.$$

Problem 5.

a)

$$\begin{aligned} & RTT_1 + \cdots + RTT_n + 2RTT_o + 3 \cdot 2RTT_o \\ = \; & 8RTT_o + RTT_1 + \cdots + RTT_n. \end{aligned}$$

b)

$$\begin{aligned} & RTT_1 + \cdots + RTT_n + 2RTT_o + 2RTT_o \\ = \; & 4RTT_o + RTT_1 + \cdots + RTT_n. \end{aligned}$$

c)

$$\begin{aligned} & RTT_1 + \cdots + RTT_n + 2RTT_o + RTT_o \\ = \; & 3RTT_o + RTT_1 + \cdots + RTT_n. \end{aligned}$$

Problem 6.

HTTP/1.0: GET, POST, HEAD.

HTTP/1.1: GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE, CONNECT.

See RFCs for explanations.

Problem 8.

UIDL abbreviates "unique-ID listing". When a POP3 client issues the UIDL command, the server responds with the unique message ID for all of the messages present in the users mailbox. This command is useful for "download and keep". By keeping a file that lists the messages retrieved in earlier sessions, the client can use the UIDL command to determine which messages on the server have already been seen.

Problem 9.

a) If you run TCPClient first, then the client will attempt to make a TCP connection with a non-existent server process. A TCP connection will not be made.

b) UDPClient doesn't establish a TCP connection with the server. Thus, everything should work fine if you first run UDPClient, then run UDPServer, and then type some input into the keyboard.

c) If you use different port numbers, then the client will attempt to establish a TCP connection with the wrong process or a non-existent process. Errors will occur.

Solutions for Chapter 3

Problem 1.

	source port numbers	destination port numbers
a) A $\rightarrow$ S	467	23
b) B $\rightarrow$ S	513	23
c) S $\rightarrow$ A	23	467
d) S $\rightarrow$ B	23	513

e) Yes.

f) No.

Problem 2.

	0	1	0	1	0	1	0	1
+	0	1	1	1	0	0	0	0
	1	1	0	0	0	1	0	1

	1	1	0	0	0	1	0	1
+	1	1	0	1	0	0	0	1
	1	0	0	1	0	1	1	0

=	1	0	0	1	0	1	1	1

One's complement = 0 1 1 0 1 0 0 0.

To detect errors, the receiver adds the four words (the three original words and the checksum). If the sum contains a zero, the receiver knows there has been an error. All one-bit errors will be detected, but two-bit errors can be undetected (e.g., if the last digit of the first word is converted to a 0 and the last digit of the second word is converted to a 1).

Problem 3.

Suppose the sender is in state "Wait for call 1 from above" and the receiver (the receiver shown in the homework problem) is in state "Wait for 1 from below." The sender sends a packet with sequence number 1, and transitions to "Wait for ACK or NAK 1," waiting for an ACK or NAK. Suppose now the receiver receives the packet with sequence number 1 correctly, sends an ACK, and transitions to state "Wait for 0 from below," waiting for a data packet with sequence number 0. However, the ACK is corrupted. When the rdt2.1 sender gets the corrupted ACK, it resends the packet with sequence number 1. However, the receiver is waiting for a packet with sequence number 0 and (as shown in the home work problem) always sends a NAK when it doesn't get a packet with sequence number 0. Hence the sender will always be sending a packet with sequence number 1, and the receiver will always be NAKing that packet. Neither will progress forward from that state.

Problem 4.

To best answer this question, consider why we needed sequence numbers in the first place. We saw that the sender needs sequence numbers so that the receiver can tell if a data packet is a duplicate of an already received data packet. In the case of ACKs, the sender does not need this info (i.e., a sequence number on an ACK) to tell detect a duplicate ACK. A duplicate ACK is obvious to the rdt3.0 receiver, since when it has received the original ACK it transitioned to the next state. The duplicate ACK is not the ACK that the sender needs and hence is ignored by the rdt3.0 sender.

Problem 5.

The sender side of protocol rdt3.0 differs from the sender side of protocol 2.2 in that timeouts have been added. We have seen that the introduction of timeouts adds the possibility of duplicate packets

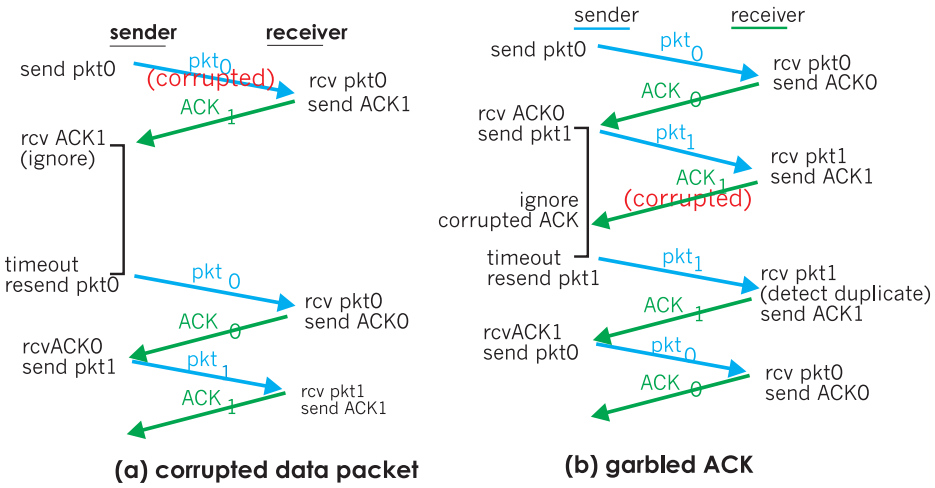


Figure 1: rdt 3.0 scenarios: corrupted data, corrupted ACK

into the sender-to-receiver data stream. However, the receiver in protocol rdt.2.2 can already handle duplicate packets. (Receiver-side duplicates in rdt 2.2 would arise if the receiver sent an ACK that was lost, and the sender then retransmitted the old data). Hence the receiver in protocol rdt2.2 will also work as the receiver in protocol rdt 3.0.

Problem 6.

Suppose the protocol has been in operation for some time. The sender is in state "Wait for call from above" (top left hand corner) and the receiver is in state "Wait for 0 from below". The scenarios for corrupted data and corrupted ACK are shown in Figure 1.

Problem 7.

Here, we add a timer, whose value is greater than the known round-trip propagation delay. We add a timeout event to the "Wait for ACK or NAK0" and "Wait for ACK or NAK1" states. If the timeout event occurs, the most recently transmitted packet is retransmitted. Let us see why this protocol will still work with the rdt2.1 receiver.

- Suppose the timeout is caused by a lost data packet, i.e., a packet on the sender-to-receiver channel. In this case, the receiver never received the previous transmission and, from the receiver's viewpoint, if the timeout retransmission is received, it look *exactly* the same as if the original transmission is being received.
- Suppose now that an ACK is lost. The receiver will eventually retransmit the packet on a timeout. But a retransmission is exactly the same action that is take if an ACK is garbled. Thus the sender's reaction is the same with a loss, as with a garbled ACK. The rdt 2.1 receiver can already handle the case of a garbled ACK.

Problem 8.

The protocol would still work, since a retransmission would be what would happen if the packet received with errors has actually been lost (and from the receiver standpoint, it never knows which of these events, if either, will occur).

To get at the more subtle issue behind this question, one has to allow for premature timeouts to occur (which is not what the question asked, sorry). In this case, if each extra copy of the packet is ACKed and each received extra ACK causes another extra copy of the current packet to be sent, the the number of times packet n is sent will increase without bound as n approaches infinity.

Problem 9.

It takes 8 microseconds (or 0.008 milliseconds) to send a packet. in oder for the sender to be busy 90 percent of the time, we must have

$$util = 0.9 = (0.008n)/30.016$$

or n is approximately 3377 packets.

Problem 10.

In our GBN reliable data transfer protocol, the sender keep sending packets until it receives a NAK. A NAK is generated for packet n only if all packets up to $n-1$ have been correctly received. That is, n is always the smallest sequence number of a packet that is yet to be received. When the receives a NAK for packet n, it simply begins sending again from packet n onwards. This is like the GBN protocol in the text, except that there is no maximum number of unacknowledged packets in the pipeline. Note the sender can't actually know how many packets are unacknowledged. If the current sequence number if k and the last NAK was for packet n, then there may be as many as $k-(n-1)$ unacknowledged packets in the pipeline.

Note also that a receiver does not know that packet n is missing until a packet with a *higher* sequence number is received! (The "gap" in the sequence numbers of received packets tells the receiver that a packet in the gap is lost). Thus, when the data rate is low (i.e., a large amount of time between packet transmissions), it will take longer for a receiver to notice a missing packet than when the data rate is high.

Problem 11.

In our solution, the sender will wait until it receives an ACK for a pair of messages (seqnum and seqnum+1) before moving on to the next pair of messages. Data packets have a data field and carry a two-bit sequence number. That is, the valid sequence numbers are 0, 1, 2, and 3. (Note: you should think about why a 1-bit sequence number space of 0, 1 only would not work in the solution below.) ACK messages carry the sequence number of the data packet they are acknowledging.

The FSM for the sender and receiver are shown in Figure 2. Note that the sender state records whether (i) no ACKs have been received for the current pair, (ii) an ACK for seqnum (only) has been received, or an ACK for seqnum+1 (only) has been received. In this figure, we assume that the seqnum is initially 0, and that the sender has sent the first two data messages (to get things going).

A timeline trace for the sender and receiver recovering from a lost packet is shown below:

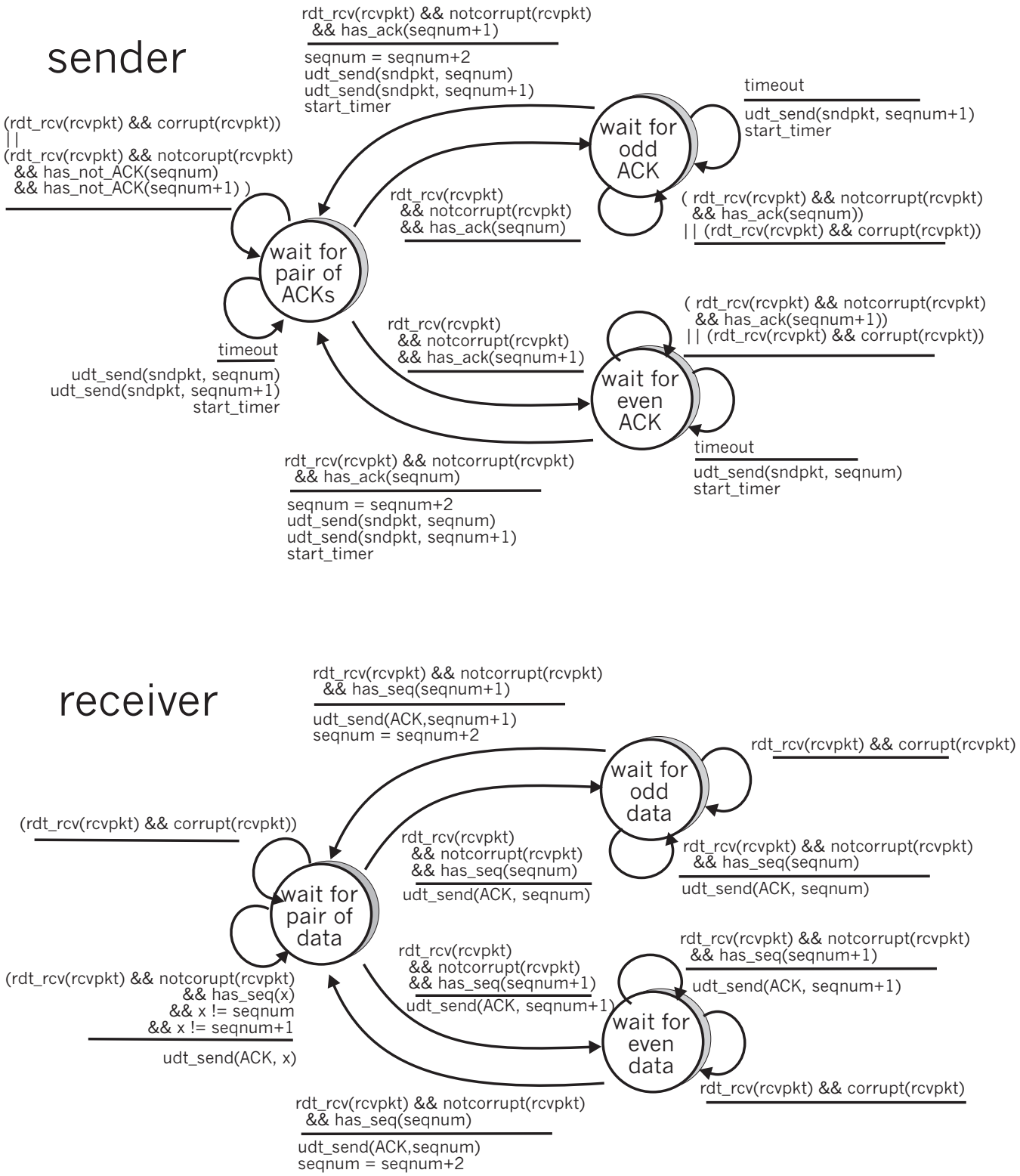


Figure 2: Sender and receiver for Problem 3.11

```
Sender                                              Receiver

make pair (0, 1)
send packet 0
                        packet 0 drops
send packet 1
                                                    receive packet 1
                                                    buffer packet 1
                                                    send ACK 1
receive ACK 1
(timeout)
resend packet 0
                                                    receive packet 0
                                                    deliver pair (0, 1)
                                                    send ACK 0
receive ACK 0
```

Problem 12.

This problem is a variation on the simple stop and wait protocol (rdt3.0). Because the channel may lose messages and because the sender may resend a message that one of the receivers has already received (either because of a premature timeout or because the other receiver has yet to receive the data correctly), sequence numbers are needed. As in rdt3.0, a 0-bit sequence number will suffice here.

The sender and receiver FSM are shown in Figure 3. In this problem, the sender state indicates whether the sender has received an ACK from B (only), from C (only) or from neither C nor B. The receiver state indicates which sequence number the receiver is waiting for.

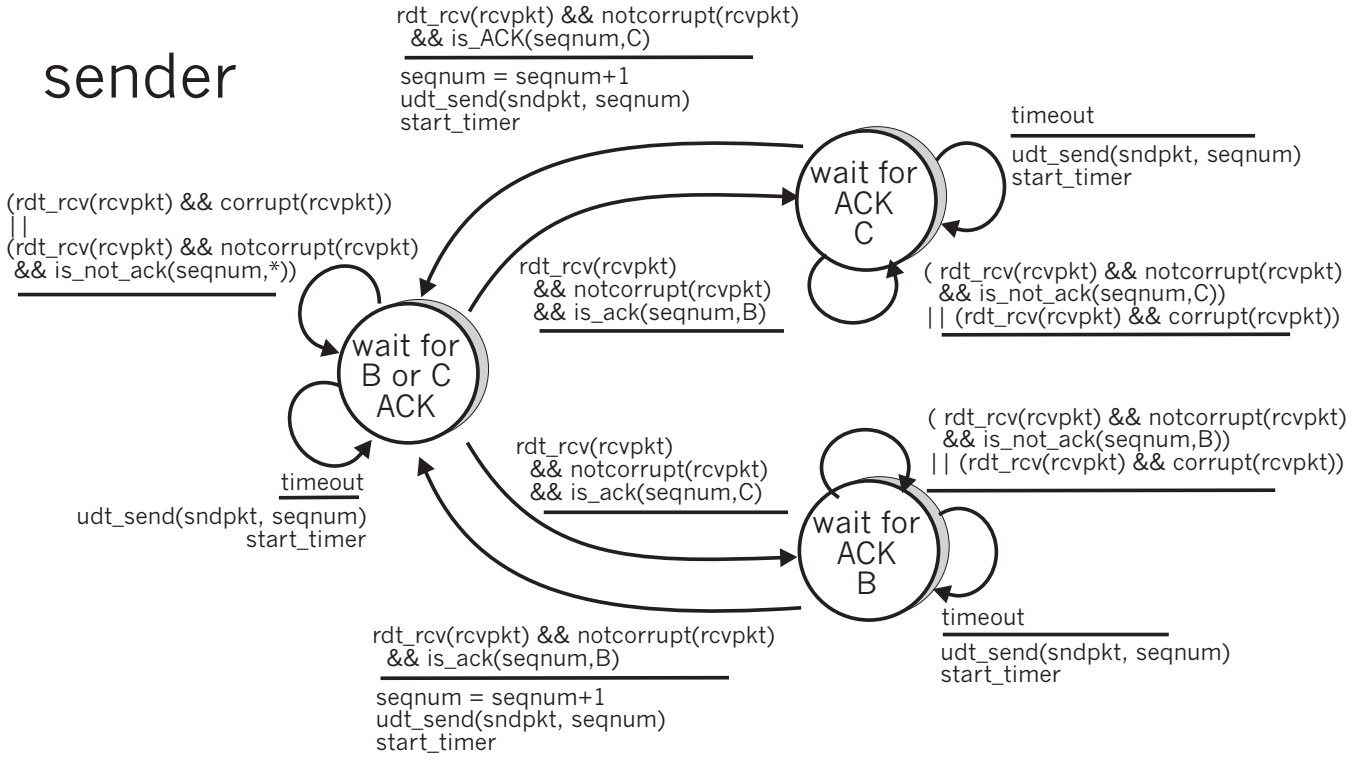
Problem 13.

a) Here we have a window size of N=3. Suppose the receiver has received packet k-1, and has ACKed that and all other preceeding packets. If all of these ACK's have been received by sender, then sender's window is [k, k+N-1]. Suppose next that none of the ACKs have been received at the sender. In this second case, the sender's window contains k-1 and the N packets up to and including k-1. The sender's window is thus [k-N,k-1]. By these arguments, the senders window is of size 3 and begins somewhere in the range [k-N,k].

b) If the receiver is waiting for packet k, then it has received (and ACKed) packet k-1 and the N-1 packets before that. If none of those N ACKs have been yet received by the sender, then ACK messages with values of [k-N,k-1] may still be propagating back. Because the sender has sent packets [k-N, k-1], it must be the case that the sender has already received an ACK for k-N-1. Once the receiver has sent an ACK for k-N-1 it will never send an ACK that is less that k-N-1. Thus the range of in-flight ACK values can range from k-N-1 to k-1.

Problem 14.

Because the A-to-B channel can lose request messages, A will need to timeout and retransmit its request messages (to be able to recover from loss). Because the channel delays are variable and unknown, it is possible that A will send duplicate requests (i.e., resend a request message that has



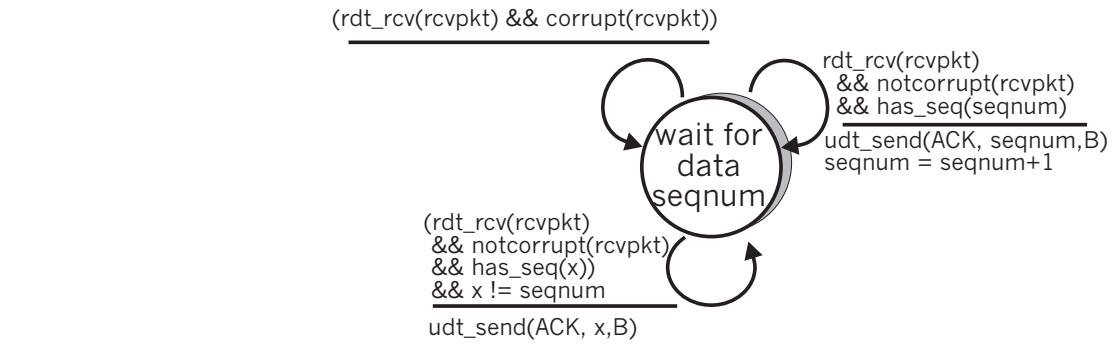


Figure 3: Sender and receiver for Problem 3.12

already been received by B). To be able to detect duplicate request messages, the protocol will use sequence numbers. A 1-bit sequence number will suffice for a stop-and-wait type of request/response protocol.

A (the requestor) has 4 states:

- **"Wait for Request 0 from above."** Here the requestor is waiting for a call from above to request a unit of data. When it receives a request from above, it sends a request message, R0, to B, starts a timer and makes a transition to the "Wait for D0" state. When in the "Wait for Request 0 from above" state, A ignores anything it receives from B.
- **"Wait for D0"**. Here the requestor is waiting for a D0 data message from B. A timer is always running in this state. If the timer expires, A sends another R0 message, restarts the timer and remains in this state. If a D0 message is received from B, A stops the time and transits to the "Wait for Request 1 from above" state. If A receives a D1 data message while in this state, it is ignored.
- **"Wait for Request 1 from above."** Here the requestor is again waiting for a call from above to request a unit of data. When it receives a request from above, it sends a request message, R1, to B, starts a timer and makes a transition to the "Wait for D1" state. When in the "Wait for Request 1 from above" state, A ignores anything it receives from B.
- **"Wait for D1"**. Here the requestor is waiting for a D1 data message from B. A timer is always running in this state. If the timer expires, A sends another R1 message, restarts the timer and remains in this state. If a D1 message is received from B, A stops the timer and transits to the "Wait for Request 0 from above" state. If A receives a D0 data message while in this state, it is ignored.

The data supplier (B) has only two states:

- **"Send D0."** In this state, B continues to respond to received R0 messages by sending D0, and then remaining in this state. If B receives a R1 message, then it knows its D0 message has been received correctly. It thus discards this D0 data (since it has been received at the other side) and then transits to the "Send D1" state, where it will use D1 to send the next requested piece of data.
- **"Send D1."** In this state, B continues to respond to received R1 messages by sending D1, and then remaining in this state. If B receives a R1 message, then it knows its D1 message has been received correctly and thus transits to the "Send D1" state.

Problem 15.

In order to avoid the scenario of Figure 3.26, we want to avoid having the leading edge of the receiver's window (i.e., the one with the "highest" sequence number) wrap around in the sequence number space and overlap with the trailing edge (the one with the "lowest" sequence number in the sender's window). That is, the sequence number space must be large enough to fit the entire receiver window and the entire sender window without this overlap condition. So - we need to determine how large a range of sequence numbers can be covered at any given time by the receiver and sender windows (see Problem 13).

Suppose that the lowest-sequence number that the receiver is waiting for is packet m. In this case, it's window is [m,m+w-1] and it has received (and ACKed) packet m-1 and the w-1 packets before that, where w is the size of the window. If none of those w ACKs have been yet received by the sender, then ACK messages with values of [m-w,m-1] may still be propagating back. If no ACKs with these ACK numbers have been received by the sender, then the sender's window would be [m-w,m-1].

Thus, the lower edge of the sender's window is m-w, and the leading edge of the receivers window is m+w-1. In order for the leading edge of the receiver's window to not overlap with the trailing edge of the sender's window, the sequence number space must thus be big enough to accommodate 2w sequence numbers. That is, the sequence number space must be at least twice as large as the window size, $k \geq 2w$.

Problem 16.

a) True. Suppose the sender has a window size of 3 and sends packets 1, 2, 3 at $t0$. At $t1$ ($t1 > t0$) the receiver ACKS 1, 2, 3. At $t2$ ($t2 > t1$) the sender times out and resends 1, 2, 3. At $t3$ the receiver receives the duplicates and re-acknowledges 1, 2, 3. At $t4$ the sender receives the ACKs that the receiver sent at $t1$ and advances its window to 4, 5, 6. At $t5$ the sender receives the ACKs 1, 2, 3 the receiver sent at $t2$. These ACKs are outside its window.

b) True. By essentially the same scenario as in (a).

c) True.

d) True. Note that with a window size of 1, SR, GBN, and the alternating bit protocol are functionally equivalent. The window size of 1 precludes the possibility of out-of-order packets (within the window). A cumulative ACK is just an ordinary ACK in this situation, since it can only refer to the single packet within the window.

Problem 17.

There are $2^{32} = 4,294,967,296$ possible sequence numbers.

a) The sequence number does not increment by one with each segment. Rather, is increments by the number of bytes of data sent. So the size of the MSS is irrelevant – the maximum size file that can be sent from A to B is simply the number of bytes representable by $2^{32} \approx 4.19$ Gbytes.

b) The number of segments is $\lceil \frac{2^{32}}{1460} \rceil = 2,941,758$. 66 bytes of header get added to each segment giving a total of 194,156,028 bytes of header. The total number of bytes transmitted is $2^{32} + 194,156,028 = 3,591 \times 10^7$ bits.

Thus it would take 3,591 seconds = 59 minutes to transmit the file over a 10 Mbps link.

Problem 18.

Suppose packets n, n+1, and n+2 are sent, and that packet n is received and ACKed. If packets n+1 and n+2 are reordered along the end-to-end-path (i.e., are received in the order n+2, n+1) then the receipt of packet n+2 will generate a duplicate ack for n and would trigger a retransmission under a policy of waiting only for second duplicate ACK for retransmission. By waiting for a triple duplicate ACK, it must be the case that *two* packets after packet n are correctly received, while n+1 was not received. The designers of the triple duplicate ACK scheme probably felt that waiting for

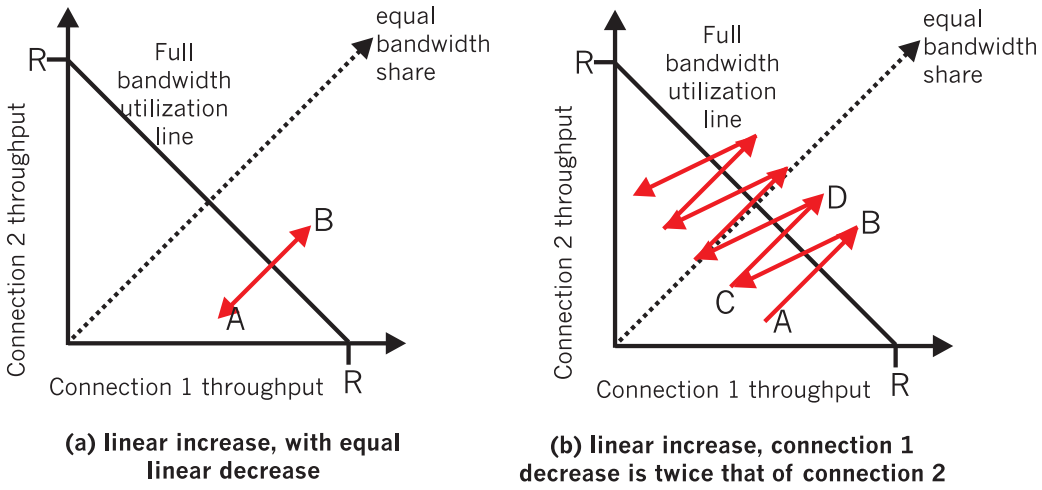


Figure 4: Lack of TCP convergence with linear increase, linear decrease

two packets (rather than 1) was the right tradeoff between triggering a quick retransmission when needed, but not retransmitting prematurely in the face of packet reordering.

Problem 19. Denote $EstimatedRTT^{(n)}$ for the estimate after the nth sample.

$$EstimatedRTT^{(1)} = SampleRTT_1$$

$$EstimatedRTT^{(2)} = xSampleRTT_1 + (1-x)SampleRTT_2$$

$$\begin{aligned} EstimatedRTT^{(3)} &= xSampleRTT_1 \\ &\quad + (1-x)[xSampleRTT_2 + (1-x)SampleRTT_3] \\ &= xSampleRTT_1 + (1-x)xSampleRTT_2 \\ &\quad + (1-x)^2SampleRTT_3 \end{aligned}$$

$$\begin{aligned} EstimatedRTT^{(4)} &= xSampleRTT_1 + (1-x)EstimatedRTT^{(3)} \\ &= xSampleRTT_1 + (1-x)xSampleRTT_2 \\ &\quad + (1-x)^2xSampleRTT_3 + (1-x)^3SampleRTT_4 \end{aligned}$$

b)

$$EstimatedRTT^{(n)} = x\sum_{j=1}^{n-1}(1-x)^j SampleRTT_j$$

$$+ (1-x)^n SampleRTT_n$$

c)

$$EstimatedRTT^{(\infty)} = \frac{x}{1-x}\sum_{j=1}^{\infty}(1-x)^j SampleRTT_j$$

$$= \frac{1}{9}\sum_{j=1}^{\infty}.9^j SampleRTT_j$$

The weight given to past samples decays exponentially.

Problem 20.

Refer to Figure 4. In Figure 4(a), the ratio of the linear decrease on loss between connection 1 and connection 2 is the same - as ratio of the the linear increases: unity. In this case, the throughputs never move off of the AB line segment. In Figure 4(a), the ratio of the linear decrease on loss between connection 1 and connection 2 is 2:1. That is, whenever there is a loss, connection 1 decreases its window by twice the amount of connection 2. We see that eventually, after enough losses, and subsequent increases, that connection 1's throughput will go to 0, and the full link bandwidth will be allocated to connection 2.

Problem 21.

a) The loss rate, L, is the ratio of the number of packets lost over the number of packets sent. In a

cycle, 1 packet is lost. The number of packets sent in a cycle is

$$\frac{W}{2} + \left(\frac{W}{2} + 1\right) + \cdots + W = \sum_{n=0}^{W/2} \left(\frac{W}{2} + n\right)$$

$$= \left(\frac{W}{2} + 1\right)\frac{W}{2} + \sum_{n=0}^{W/2} n$$

$$= \left(\frac{W}{2} + 1\right)\frac{W}{2} + \frac{W/2(W/2+1)}{2}$$

$$= \frac{W^2}{4} + \frac{W}{2} + \frac{W^2}{8} + \frac{W}{4}$$

$$= \frac{3}{8}W^2 + \frac{3}{4}W$$

Thus the loss rate is

$$L = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

b) For W large, $\frac{3}{8}W^2 \gg \frac{3}{4}W$. Thus $L \approx 8/3W^2$ or $W \approx \sqrt{\frac{8}{3L}}$. From the text, we therefore have

$$\text{average throughput} = \frac{3}{4}\sqrt{\frac{3}{8L}} \cdot \frac{\text{MSS}}{\text{RTT}}$$

$$= \frac{1.22 \cdot \text{MSS}}{\text{RTT} \cdot \sqrt{L}}$$

Problem 22.

The minimum latency is $2\text{RTT} + O/R$. The minimum W that achieves this latency is

$$W = \min\left\{w : w \geq \frac{\text{RTT} + S/R}{S/R}\right\}$$

$$= \left\lceil \frac{\text{RTT}}{S/R} \right\rceil + 1.$$

R	min latency	W
28 Kbps	28.77 sec	2
100 Kbps	8.2 sec	4
1 Mbps	1 sec	25
10 Mbps	0.28 sec	235

Problem 23.

a)

$$K = \text{number of windows that cover the object}$$

$$= \min\{k : 3^0 + 3^1 + \cdots + 3^{k-1} \geq O/S\}$$

$$= \min\left\{k : \frac{1-3^k}{1-3} \geq O/S\right\}$$

$$= \min\{k : 3^k \geq 1 + 2O/S\}$$
$$= \lceil \log_3(1 + 2O/S) \rceil$$

b) Q is the number of times the server would stall for an object of infinite size.

$$Q = \max\left\{k : \text{RTT} + \frac{S}{R} - \frac{S}{R}3^{k-1} \geq 0\right\}$$

$$= \left\lfloor 1 + \log_3\left(1 + \frac{\text{RTT}}{S/R}\right)\right\rfloor$$

c)

$$\text{latency} = \frac{O}{R} + 2\text{RTT} + \sum_{k=1}^{P} \text{stall}_k$$

$$= \frac{O}{R} + 2\text{RTT} + \sum_{k=1}^{P}\left(\text{RTT} + \frac{S}{R} - \frac{S}{R}3^{k-1}\right)$$

$$= \frac{O}{R} + 2\text{RTT} + P(\text{RTT} + S/R) - \frac{(3^P - 1)}{2}\frac{S}{R}$$

Problem 24.

R	O/R	P	Min latency	Latency with slow start
28 Kbps	29.25 s	3	31.25 sec	33.18 sec
100 Kbps	8.19 s	5	10.19 sec	13.86 sec
1 Mbps	819 msec	7	2.81 sec	9.26 sec
10 Mbps	82 msec	7	2 sec	9 sec

Problem 25.

a) T

b) T

c) F

d) F

Problem 26.

When the server sends a segment, it has to wait a time of $TS/R + RTT$ for the acknowledgement to arrive. The transmission time of the kth window is $(S/R)2^{k-1}$. The stall time for the kth window is

$$\left[\frac{TS}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^+$$

The number of stalls Q is

$$Q = \max\left\{k : RTT + \frac{TS}{R} - \frac{S}{R}2^{k-1} \geq 0\right\}$$

$$= \max\left\{k : 2^{k-1} \leq T + \frac{RTT}{S/R}\right\}$$

$$= \max\left\{k : k \leq \log_2\left(T + \frac{RTT}{S/R}\right) + 1\right\}$$

$$= \left\lfloor \log_2\left(T + \frac{RTT}{S/R}\right)\right\rfloor + 1$$

The number of times the server stalls is $P = \min(Q, K-1)$. The latency is

$$\text{latency} = 2RTT + \frac{O}{R} + \sum_{k=1}^{P}\left(RTT + \frac{TS}{R} - \frac{S}{R}2^{k-1}\right)$$

which simplifies to

$$\text{latency} = 2RTT + \frac{O}{R} + P\left(RTT + \frac{TS}{R}\right) - (2^P - 1)\frac{S}{R}$$

Problem 27.

$$\frac{P\left(\text{RTT} + \frac{S}{R}\right) - (2^P - 1)\frac{S}{R}}{2\text{RTT} + \frac{O}{R} + P\left(\text{RTT} + \frac{S}{R}\right) - (2^P - 1)\frac{S}{R}}$$

Problem 28.

a)

$$\text{Transfer time of all } M+1 \text{ objects}: \quad (M+1)\frac{O}{R}$$

$$\text{TCP connection setup}: \quad 2 \cdot RTT$$
$$\text{Request for images}: \quad RTT$$

To this we have to add the latency due to slow-start.

Comparing the contribution of RTTs with that in non-persistent HTTP we see that the only RTTs that affect the response time are the two RTTs needed for setting up the TCP connection and

sending the initial request and one RTT for requesting the images. In non-persistent HTTP we have a separate TCP connection setup for each object.

b)

Because the last window of the initial object is full and the server does not need to wait for a request for the images, the situation is identical to the server sending one large object of $(M+1)O$ bytes. The response time is

$$2 \cdot RTT + \frac{(M+1)O}{R} + P'\left(RTT + \frac{S}{R}\right) - (2^{P'} - 1)\frac{S}{R}$$

c)

In the previous question, the server did not have to wait for a request for the images. The stall time after sending the initial object in that case is $[S/R + RTT - 2^{K-1}(S/R)]^+$. In reality, the stall time is RTT because the server has to wait for an explicit request for the images from the client. Substituting this into the response time equation gives

$$3 \cdot RTT + \frac{(M+1)O}{R} + P'\left(RTT + \frac{S}{R}\right) - (2^{P'} - 1)\frac{S}{R} - [\frac{S}{R} + RTT - 2^{K-1}\frac{S}{R}]^+$$

Problem 29.

Rate	Persistent	Non-persistent
28 Kbps	16.2 sec	20.4 sec
100 Kbps	5.1 sec	10.6 sec
1 Mbps	1.3 sec	8.9 sec
10 Mbps	1.1 sec	8.8 sec

Problem 30.

Rate	Persistent	Non-persistent
28 Kbps	23.6 sec	91.7 sec
100 Kbps	13.4 sec	89.0 sec
1 Mbps	11.2 sec	88.1 sec
10 Mbps	11.0 sec	88.0 sec

Problem 31.

$$\begin{aligned} \text{Time to transfer all objects over the link :} \quad & (M+1)O/R \\ \text{TCP connection setup for first request :} \quad & 2RTT \\ \text{TCP connection setup for } M/X \text{ sets of image requests :} \quad & M/X \cdot 2RTT \end{aligned}$$

Adding the latency due to slow-start gives

$$(M+1)\frac{O}{R} + 2\left(\frac{M}{X}+1\right)RTT + \text{latency due to slow-start stalling.}$$

The contribution of the term involving RTT is larger than that of persistent connections but smaller than that of non-persistent, non-parallel connections. This is because we open more connections, M/X, than with persistent connections (only one connection in that case) but less than with non-parallel connections in which case we open M connections.

Solutions for Chapter 4

Problem 1.

a) With a connection-oriented network, every router failure will involve the routing of that connection. At a minimum, this will require the router that is"upstream" from the failed router to establish a new downstream part of the path to the destination node, with all of the requisite signaling involved in setting up a path. Moreover, all of the router on the initial path that are downstream from the failed node must take down the failed connection, with all of the requisite signaling involved to do this.

With a connectionless datagram network, no signaling is required to either set up a new downstream path or take down the old downstream path. We have seen, however, that routing tables will need to be updated (e.g., either via a distance vector algorithm or a link state algorithm) to take the failed router into account. We have seen that with distance vector algorithms, this routing table change

can sometimes be localized to the area near the failed router. Thus, a datagram network would be preferable. Interesting, the design criteria that the initial ARPAnet be able to function under stressful conditions was one of the reasons that a datagram architecture was chosen for this Internet ancestor.

b) In order for a router to determine the delay (or a bound on delay) along an outgoing link, it would need to know the characteristics of the traffic from all sessions passing through that link. That is, the router must have per-session state in the router. This is possible in a connection-oriented network, but not with a connectionless network. Thus, a connection-oriented network would be preferable.

Problem 2.

ABCDEF
ABCEF
ABCF
ABDCEF
ABDEF
ABDECF
ADBCF
ADBCEF
ADCEF
ADCF
ADEF
ADECF
ACF
ACBDF
ACDEF
ACEF

Problem 3.

N	a	b	c	d	e	g	h
	inf	inf	inf	3,f	1,f	6,f	inf
e	inf	inf	4,e	2,e		g,f	inf
ed	inf	11,d	3,d			3,d	inf
edc	7,c	5,c				3,d	inf
edcg	7,c	5,c					17,g
edcgb	6,b						7,b
edcgba							7,b

Problem 4.

The distance table in E is:

		via	
	b	c	d
to a	6	13	4
b	5	14	5
c	9	10	3
d	8	11	2

Problem 5.

The wording of this question was a bit ambiguous. We meant this to mean, "the number of iterations from when the algorithm is run for the first time"(that is, assuming the only information the nodes initially have is the cost to their nearest neighbors). We assume that the algorithm runs synchronously (that is, in one step, all nodes compute their distance tables at the same time and then exchange tables).

At each iteration, a node exchanges distance tables with its neighbors. Thus, if you are node A, and your neighbor is B, all of B's neighbors (which will all be one or two hops from you) will know the shortest cost path of one or two hops to you after one iteration (i.e., after B tells them its cost to you).

Let d be the "diameter" of the network - the length of the longest path without loops between any two nodes in the network. Using the reasoning above, after d-1 iterations, all nodes will know the shortest path cost of d or fewer hops to all other nodes. Since any path with greater than d hops will have loops (and thus have a greater cost than that path with the loops removed), the algorithm will converge in at most d-1 iterations.

ASIDE: if the DV algorithm is run as a result of a change in link costs, there is no a priori bound on the number of iterations required until convergence unless one also specifies a bound on link costs.

Problem 6.

The distance table in X is:

		via	
	W	Y	
to W	1	?	
Y	?	4	
A	6	10	

Note that there is not enough information given in the problem (purposefully!) to determine the distance table entries D(W,Y) and D(Y,W). To know these values, we would need to know Y's minimum cost to W, and vice versa.

Since X's least cost path to A goes through W, a change in the link cost c(X,W) will cause X to inform its neighbors of a new minimum cost path to A, Since X's least cost path to A does not go through Y, a change in the link cost c(X,Y) will not cause X to inform its neighbors of a new minimum cost path to A.

Problem 7.

The distance table in X, Y, Z are:

```
DX    via          DY    via          DZ    via
to    Y    Z       to    X    Z       to    X    Y
  Y   2    8        X    2    4        X    7    3
  Z   3    7        Z    5    1        y    9    1
```

Although Y's distance table has changed as a result of this iteration, its minimum cost paths to various destinations have *not* changed and hence Y will not send out its distance tables. X's and Z's tables have not changed as a result of their iteration and hence they will not send out their tables. Thus, the algorithm has converged after this iteration.

Problem 8.

```
        via                  via
   DX  Y   Z            DX   Y   Z
to Y   5  inf     to Y    5   8
   Z   inf 2          Z    11  2

        via                  via
   DY  X   Z            DY  X    Z
to X   5  inf     to  X   5    8
   Z  inf  6          Z   7    6

        via                  via
   DZ  Y   X            DZ  Y   X
to Y   6  inf     to  Y   6   7
   X  inf  2          X   11  2
```

Problem 9.

The minimal spanning tree has G connected to D at a cost of 1, E connected to D at a cost of 1; C connected to D at a cost of 1; C connected to B at a a cost of 2; B connected to A at a cost of 1.

We can argue informally that the tree cost is minimal as follows. C, D, E, and G can be connected to each other at a cost of 3. One can't do any better than that. In order to connect A in to any of C, D, E, G, the minimum cost is 3 (via B).

Problem 10.

Since there is no network layer protocol to identify hosts participating in a group, this must be done at a higher layer. In practice, this is done using an application-level protocol over IP multicast. See Problem 23.

Problem 11.

The cost of sending one packet to all 32 receivers via multicast is 62. With multicast, the packet traverses a link exactly once. There are 62 links in a binary tree of depth 5. If the packet is unicast to all receivers, then each of the 32 receivers is sent an individual packet. Each packet must travel across five links (recall the tree depth is 5) from the source to any receiver. Hence the cost is 32*5=160. We see there is a significant savings using multicast in this case.

Problem 12.

Here's a rough sketch of how an application level group-participant-identification (GPI) protocol would work. We'll assume the application entities use a *separate multicast* channel in the GPI protocol. There are two key components:

- Each application periodically sends an "I am in this group" message, giving it's host identity, into the multicast control channel. This message is then received by all other participants.
- Each application maintains a timer for each of the other participants. If it doesn't hear an "I am in this group" message from a group participant within the timeout interval, it removes the participant from its participant list. An alternative approach would have been for a participant to explicitly remove itself from the group. In this latter case, if the participant died, it would remain in everyone's group list forever. For this reason, we prefer the "soft-state" mechanism in the former approach. But either answer is OK; it's a matter of taste.

Problem 13.

With the new link cost, the Steiner tree connects A to C; B to C; C to E; and F to E. In order to prove this is a minimal cost tree, we could generate all possible spanning trees connecting these 4 nodes and verify that the tree described above indeed has minimum cost.

Problem 14.

The center-based tree for the topology shown in the original figure connects A to D; B to C; E to C; and F to C (all directly). This center-based tree is different from the minimal spanning tree shown in the figure.

Problem 15.

The least cost path tree connects F directly to E. B's path to E is BDE. A's path to E is ACE.

Problem 16.

See Figure 5.

Problem 17.

Reverse path forwarding would still work. RPF uses the unicast cost only to determine from which incoming link it will receive and forward (downstream) multicast packets. Packets arriving from other links are dropped. RPF would also work (arguably not as efficiently) if one forwarded only the packet arriving on the most costly path to the source.

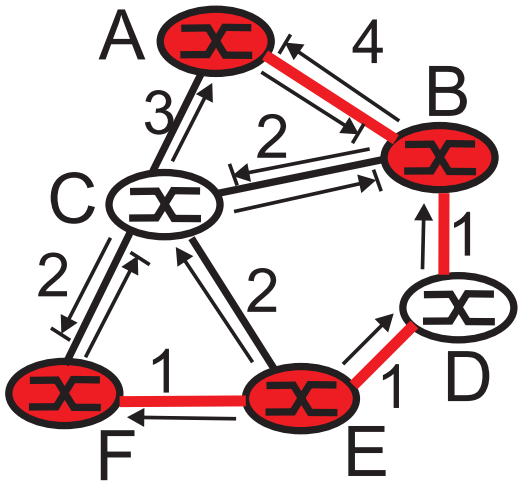


Figure 5: Solution to problem 16

Problem 18.

In the center-based tree the total traffic intensity (upstream + downstream traffic) is 4 on links AC, BC, EC, and FC. There is one unit of traffic emanating from nodes A,B,E,F and three units of traffic coming downstream from C that was sent by the other three nodes.

The source tree rooted at A has one unit of traffic on AB, AC, CE, and CF. The source tree rooted at B has 1 unit of traffic on AB, BD, DE, EF. The source tree rooted at E has one unit of traffic on EF, ED. DB, EC and CA. The source tree rooted at F has one unit of traffic on FE, ED, DB, FC, and CA. Summing these up, the union of the source rooted trees generates: 3 units on CA; 2 units on CF; 2 units on AB; 3 units on BD; 3 units on ED; 3 units on EF; and 2 units on CE. Note that no link has 4 units of traffic, while four of the links in the center-based tree have 4 units of traffic on them. Thus, in this example, we can say that the center-based tree tends to concentrate traffic.

Problem 19.

Since MOSP builds source-specific trees, its state complexity is S*G. PIM SM builds a single center-based tree for the group and hence has state complexity G. PIM DM is like DVMRP and thus has a complexity of S*G.

Problem 20.

$32 - 4 = 28$ bits are available for multicast addresses. Thus, the size of the multicast address space is $N = 2^{28}$.

The probability that two groups choose the same address is

$$\frac{1}{N} = 2^{-28} = 3.73 \cdot 10^{-9}$$

The probability that 1000 groups all have different addresses is

$$\frac{N \cdot (N-1) \cdot (N-2) \cdots (N-999)}{N^{1000}} = \left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right) \cdots \left(1 - \frac{999}{N}\right)$$

Ignoring cross product diterms, this is approximately equal to

$$1 - \left(\frac{1 + 2 + \cdots + 999}{N}\right) = 1 - \frac{999 \cdot 1000}{2N} = 0.998$$

Problem 21.

The IP router at the end of the tunnel will use the protocol number in the "Upper Layer Protocol" field of the IP packet to determine which upper layer protocol to pass the IP packet. Thus, IP doesn't really know that the IP packet contains a multicast datagram. This is only discovered when the upper layer protocol (which will perform the multicast copy and routing) "opens" the IP datagram it is handed by the IP layer.

Solutions for Chapter 5

Problem 1.

The rightmost column and bottom row are for parity bits.

```
1 0 1 0 0
1 0 1 0 0
1 0 1 0 0
1 0 1 1 1
0 0 0 1 1
```

Problem 2.

Suppose we begin with the initial two-dimensional parity matrix:

```
0 0 0 0
1 1 1 1
0 1 0 1
1 0 1 0
```

With a bit error in row 2, column 3, the parity of row 2 and column 3 is now wrong in the matrix below:

```
0 0 0 0
1 1 0 1
0 1 0 1
1 0 1 0
```

Now suppose there is a bit error in row 2, column 2 and column 3. The parity of row 2 is now correct! The parity of columns 2 and 3 is wrong, but we can't detect in which rows the error occurred!

```
0 0 0 0
1 0 0 1
0 1 0 1
1 0 1 0
```

The above example shows that a double bit error can be detected (if not corrected)

Problem 3.

To compute the Internet checksum, we add up the values at 16-bit quantities:

```
00000000  00000001
00000010  00000011
00000100  00000101
00000110  00000111
00001000  00001001
------------------
00010100  00011001
```

The one's complement of the sum is 11101011 11100110.

Problem 4.

If we divide 1001 into 10101010000 we get 10010111, with a remainder of R = 001.

Problem 5.

The output is

```
-1,+1,-1,+1,-1,+1,-1,+1  +1,-1,+1,-1,+1,-1,+1,-1
```

Problem 6.

Sender 2's output before the adder is

```
+1,-1,+1,+1,+1,-1,+1,+1     +1,-1,+1,+1,+1,-1,+1,+1
```

Problem 7.

$$d_0^2 = (2,0,2,0,2,-2,0,0) * (1,-1,1,1,1,-1,1,1)/8 = (2+2+2+2)/8 = 1$$

$$d_1^2 = (0,-2,0,2,0,0,2,2) * (1,-1,1,1,1,-1,1,1)/8 = (2+2+2+2)/8 = 1$$

Problem 8.

a)

$$E(p) = Np(1-p)^{N-1}$$

$$E'(p) = N(1-p)^{N-1} - Np(N-1)(1-p)^{N-2}$$

$$= N(1-p)^{N-2}((1-p) - p(N-1))$$

$$E'(p) = 0 \quad \Rightarrow \quad p^* = \frac{1}{N}$$

b)

$$E(p^*) = N\frac{1}{N}(1-\frac{1}{N})^{N-1} = (1-\frac{1}{N})^{N-1} = \frac{(1-\frac{1}{N})^N}{1-\frac{1}{N}}$$

$$\lim_{N\to\infty}(1-\frac{1}{N}) = 1 \qquad \lim_{N\to\infty}(1-\frac{1}{N})^N = \frac{1}{e}$$

Thus

$$\lim_{N\to\infty} E(p^*) = \frac{1}{e}$$

Problem 9.

$$E(p) = Np(1-p)^{2(N-1)}$$

$$E'(p) = N(1-p)^{2(N-2)} - Np2(N-1)(1-p)^{2(N-3)}$$

$$= N(1-p)^{2(N-3)}((1-p) - p2(N-1))$$

$$E'(p) = 0 \quad \Rightarrow \quad p^* = \frac{1}{2N-1}$$

$$E(p^*) = \frac{N}{2N-1}(1 - \frac{1}{2N-1})^{2(N-1)}$$

$$\lim_{N\to\infty} E(p^*) = \frac{1}{2} \cdot \frac{1}{e} = \frac{1}{2e}$$

Problem 10.

Pure Aloha has an efficiency of zero for almost all values of p. See Figure 6

Problem 11.

The length of a polling round is

$$N(Q/R + t_{poll})$$

The number of bits transmitted in a polling round is NQ. The maximum throughput therefore is

$$\frac{NQ}{N(Q/R + t_{poll})} = \frac{R}{1 + \frac{t_{poll}R}{Q}}$$

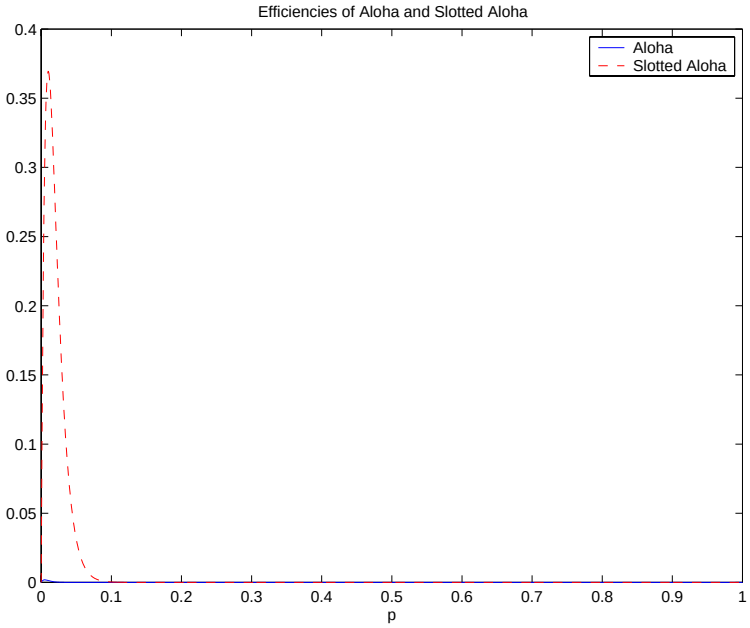


Figure 6: Efficiency of Aloha and Slotted Aloha

Problem 12.

a), b), c) See Figure 7

d)

1. Routing table in A determines that the datagram should be routed to interface 111.111.111.002.
2. Host A uses ARP to determine the LAN address for 111.111.111.002, namely 22-22-22-22-22-22.
3. The adapter in A creates and Ethernet packet with Ethernet destination address 22-22-22-22-22-22.
4. The first router receives the packet and extracts the datagram. The routing table in this router indicates that the datagram is to be routed to 122.222.222.003.

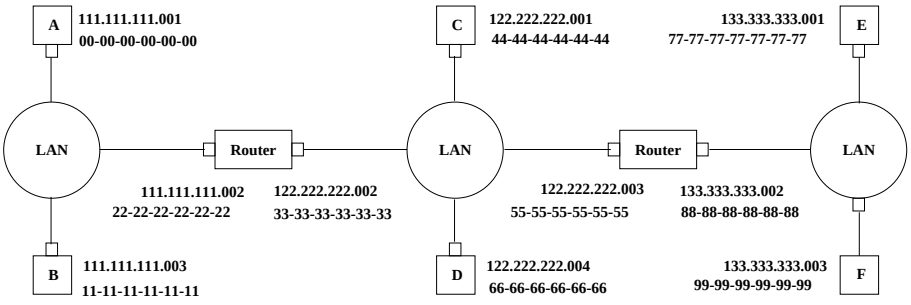


Figure 7: Solution to problem 12

5. The first router then uses ARP to obtain the associated Ethernet address, namely 55-55-55-55-55-55.

6. The process continues until the packet has reached Host F.

e)

ARP in A must now determine the LAN address of 111.111.111.002. Host A sends out an ARP query packet within a broadcast Ethernet frame. The first router receives the query packet and sends to Host A an ARP response packet. This ARP response packet is carried by an Ethernet frame with Ethernet destination address 00-00-00-00-00-00.

Problem 13.

Wait for 51,200 bit times. For 10 Mbps, this wait is

$$\frac{51.2 \times 10^3 \text{ bits}}{10 \times 10^6 \text{ bps}} = 5.12 \text{ msec}$$

For 100 Mbps, the wait is 512 μsec.

Problem 14.

At $t = 0$ A transmits. At $t = 576$, A would finish transmitting. In the worst case, B begins transmitting at time $t = 224$. At time $t = 224 + 225 = 449$ B's first bit arrives at A. Because $449 < 576$, A aborts before completing the transmission of the packet, as it is supposed to do.

Thus A cannot finish transmitting before it detects that B transmitted. This implies that if A does not detect the presence of a host, then no other host begins transmitting while A is transmitting.

Problem 15.

Time, t	Event
0	A and B begin transmission
225	A and B detect collision
273	A and B finish transmitting jam signal
273+225 = 498	B's last bit arrives at A; A detects an idle channel
498	A starts transmitting. B schedules transmission at 273 + 512 = 785
498 + 225 = 723	A's retransmission reaches B

Because A's retransmission reaches B before B's scheduled retransmission time, B refrains from transmitting while A retransmits. Thus A and B do not collide. Thus the factor 512 appearing in the exponential backoff algorithm is sufficiently large.

Problem 16.

We want $1/(1+5a) = .5$ or, equivalently, $a = .2 = t_{prop}/t_{trans}$. $t_{prop} = d/(1.8 \times 10^8)$ m/sec and $t_{trans} = (576 \text{ bits})/(10^8 \text{ bits/sec}) = 5.76\mu$sec. Solving for d we obtain $d = 265$ meters. For the 100 Mbps Ethernet standard, the maximum distance between two hosts is 200 m.

For transmitting station A to detect whether any other station transmitted during A's interval, t_{trans} must be greater than $2t_{prop} = 2 \cdot 265 \text{ m}/1.8 \times 10^8 \text{ m/sec} = 2.94\mu$sec. Because $2.94 < 5.76$, A will detect B's signal before the end of its transmission.

Problem 17.

a)

Let Y be a random variable denoting the number of slots until a success:

$$P(Y = m) = \beta(1-\beta)^{m-1},$$

where β is the probability of a success.

This is a geometric distribution, which has mean $1/\beta$. The number of consecutive wasted slots is $X = Y - 1$ that

$$x = E[X] = E[Y] - 1 = \frac{1-\beta}{\beta}$$

$$\beta = Np(1-p)^{N-1}$$

$$x = \frac{1 - Np(1-p)^{N-1}}{Np(1-p)^{N-1}}$$

$$\text{efficiency} = \frac{k}{k+x} = \frac{k}{k + \frac{1-Np(1-p)^{N-1}}{Np(1-p)^{N-1}}}$$

b)

Maximizing efficiency is equivalent to minimizing x, which is equivalent to maximizing β. We know from the text that β is maximized at $p = \frac{1}{N}$.

c)

$$\text{efficiency} = \frac{k}{k + \frac{1-(1-\frac{1}{N})^{N-1}}{(1-\frac{1}{N})^{N-1}}}$$

$$\lim_{N\to\infty} \text{efficiency} = \frac{k}{k + \frac{1-e}{e}} = \frac{ke}{1 + (k-1)e}$$

d) Clearly, $\frac{ke}{1+(k-1)e}$ approaches 1 as $k \to \infty$.

Problem 18.

a)

$$\frac{900 \text{ m}}{2 \cdot 10^8 \text{ m/sec}} + 4 \cdot \frac{20 \text{ bits}}{10 \times 10^6 \text{ bps}}$$

$$\begin{aligned} &= (4.5 \times 10^{-6} + 8 \times 10^{-6}) \text{ sec} \\ &= 12.5\mu\text{sec} \end{aligned}$$

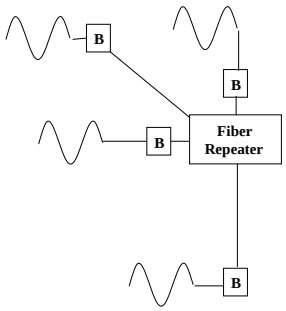
b)

- At time $t = 0$, both A and B transmit.
- At time $t = 12.5\mu$sec, A detects a collision.
- At time $t = 25\mu$sec last bit of B's aborted transmission arrives at A.
- At time $t = 37.5\mu$sec first bit of A's retransmission arrives at B.
- At time $t = 37.5\mu\text{sec} + \frac{1000 \text{ bits}}{10\times10^6 \text{ bps}} = 137.5\mu$sec A's packet is completely delivered at B.

c) $12.5\mu\text{sec} + 5 \cdot 100\mu\text{sec} = 512.5\mu\text{sec}$.

Problem 19.

a) The key to the design is to build a network with a backbone. This is done by putting a multiport fiber repeater in WC2, and connecting four ports of this repeater to bridges:



Each bridge has one fiber port and one coax port. We put the marketing bridge in WC1, the support and manufacturing bridges in WC2, and the engineering bridge in WC3.

b) Here we use a design similar to part (a) except we replace the thin coax cables with UTP hubs, and we use bridges that have one fiber port and one UTP port. Note that the maximum length of a UTP connection is 100 m. For this reason, the design just outlined may not suffice for the manufacturing, which may have hosts more than 100 m away from WC2. To solve this problem we put a second bridge for manufacturing in WC1. This bridge is connected to the multiport fiber repeater by fiber.

Problem 20.

a) $CIR \cdot T_c$ "high-priority bits" can be sent in a measurement interval. Thus

$$\left\lfloor \frac{CIR \cdot T_c}{L} \right\rfloor$$

packets can be sent in a measurement interval.

b)

$$\left\lfloor \frac{R \cdot T_c}{L} \right\rfloor - \left\lfloor \frac{CIR \cdot T_c}{L} \right\rfloor$$

Problem 21.

The maximum number of high-priority packets that can pass in a measurement interval is

$$N = \left\lfloor \frac{CIR \cdot T_c}{L} \right\rfloor$$

Each packet has L bits, so that maximum size of object is

$$O = L \cdot N = L \left\lfloor \frac{CIR \cdot T_c}{L} \right\rfloor \approx CIR \cdot T_c$$

Problem 22.

a) The time required to fill $L \cdot 8$ bits is

$$\frac{L \cdot 8}{64 \times 10^3} \text{sec} = \frac{L}{8} \text{msec}.$$

b) For $L = 1,500$, the packetization delay is

$$\frac{1500}{8} \text{msec} = 187.5 \text{msec}.$$

For $L = 48$, the packetization delay is

$$\frac{48}{8} \text{msec} = 6 \text{msec}.$$

c)

$$\text{Store-and-forward delay} = \frac{L \cdot 8 + 40}{R}$$

For $L = 1,500$, the delay is

$$\frac{1500 \cdot 8 + 40}{155 \times 10^6}\text{sec} \approx \frac{12}{155}\text{msec} \approx 77\mu\text{sec}$$

For $L = 48$, store-and-forward delay $< 1\mu$sec.

d) Store-and-forward delay is small for both cases for typical ATM link speeds. However, packetization delay for $L = 1500$ is too large for real-time voice applications.

Solutions for Chapter 6

Problem 3.

$x(t)$ will continue to grow until the client buffer becomes full. Once the client buffer becomes full, the client application will drain the receive TCP buffer at rate d. TCP flow control will then throttle the sender's transmission rate so that the average of $x(t)$ after the client buffer becomes full is approximately d.

Problem 4.

No, they are not the same thing. The client application reads data from the TCP receive buffer and puts it in the client buffer. If the client buffer becomes full, then application will stop reading from the TCP receive buffer until some room opens up in the client buffer.

Problem 5.

a) $160 + h$ bytes are sent every 20 msec. Thus the transmission rate is

$$\frac{(160 + h) \cdot 8}{20}\text{Kbps} = (64 + .4h)\text{Kbps}$$

b)

IP header : 20bytes
UDP header : 8bytes
RTP header : 12bytes

$h = 40$ bytes (a 25% increase in the transmission rate!)

Problem 6.

a) Denote $d^{(n)}$ for the estimate after the nth sample.

$$d^{(1)} = r_4 - t_4$$

$$d^{(2)} = u(r_3 - t_3) + (1-u)(r_4 - t_4)$$

$$d^{(3)} = u(r_2 - t_2) + (1-u)[u(r_3 - t_3) + (1-u)(r_4 - t_4)]$$

$$= u(r_2 - t_2) + (1-u)u(r_3 - t_3) + (1-u)^2(r_4 - t_4)$$

$$d^{(4)} = u(r_1 - t_1) + (1-u)d^{(3)}$$

$$= u(r_1 - t_1) + (1-u)u(r_2 - t_2) + (1-u)^2u(r_3 - t_3) + (1-u)^3(r_4 - t_4)$$

b)

$$d^{(n)} = u\sum_{j=1}^{n-1}(1-u)^j(r_j - t_j) + (1-u)^n(r_n - t_n)$$

c)

$$d^{(\infty)} = \frac{u}{1-u}\sum_{j=1}^{\infty}(1-u)^j(r_j - t_j)$$

$$= \frac{1}{9}\sum_{j=1}^{\infty}.9^j(r_j - t_j)$$

The weight given to past samples decays exponentially.

Problem 7.

a) Denote $v^{(n)}$ for the estimate after the nth sample. Let $\Delta_j = r_j - t_j$.

$$v^{(1)} = |\Delta_4 - d^{(1)}| \qquad (= 0)$$

$$v^{(2)} = u|\Delta_3 - d^{(2)}| + (1-u)|\Delta_4 - d^{(1)}|$$

$$v^{(3)} = u|\Delta_2 - d^{(3)}| + (1-u)v^{(2)}$$

$$= u|\Delta_2 - d^{(3)}| + u(1-u)|\Delta_3 - d^{(2)}| + (1-u)^2|\Delta_4 - d^{(1)}|$$

$$v^{(4)} = u|\Delta_1 - d^{(4)}| + (1-u)v^{(3)}$$

$$= u|\Delta_1 - d^{(4)}| + (1-u)u|\Delta_2 - d^{(3)}| + u(1-u)^2|\Delta_3 - d^{(2)}|$$

$$+ (1-u)^3|\Delta_4 - d^{(1)}|$$

$$= u\big[|\Delta_1 - d^{(4)}| + (1-u)|\Delta_2 - d^{(3)}| + (1-u)^2|\Delta_3 - d^{(2)}|\big]$$

$$+ (1-u)^3|\Delta_4 - d^{(1)}|$$

b)

$$v^{(n)} = u\sum_{j=1}^{n-1}(1-u)^{j-1}|\Delta_j - d^{(n-j+1)}| + (1-u)^n|\Delta_n - d^{(1)}|$$

Problem 8.

The two procedures are very similar. They both use the same formula, thereby resulting in exponentially decreasing weights for past samples.

One difference is that for estimating average RTT, the time when the data is sent and when the acknowledgement is received is recorded on the same machine. For the delay estimate, the two values are recorded on different machines. Thus the sample delay can actually be negative.

Problem 9.

a) If there is packet lost, then other packets may have been transmitted inbetween the two received packets.

b) Let S_i denote the sequence number of the ith received packet. If

$$t_i - t_{i-1} > 20\text{msec}$$

and

$$S_i = S_{i-1} + 1$$

then packet i begins a new talkspurt.

Problem 10.

a) Both schemes require 25% more bandwidth. The first scheme has a playback delay of 5 packets. The second scheme has a delay of 2 packets.

b) The first scheme will be able to reconstruct the original high-quality audio encoding. The second scheme will use the low quality audio encoding for the lost packets and will therefore have lower overall quality.

c) For the first scheme, many of the original packets will be lost and audio quality will be very poor. For the second scheme, every audio chunk will be available at the receiver, although only the low quality version will be available for every other chunk. Audio quality will be acceptable.

Problem 11.

The interarrival jitter J is defined to be the mean deviation (smoothed absolute value) of the difference D in packet spacing at the receiver compared to the sender for a pair of packets. As shown in the equation below, this is equivalent to the "relative transit time" for the two packets; the relative transit time is the difference between a packet's RTP timestamp and the receiver's clock at the time of arrival. If T_i is the RTP timestamp for packet i and r_i is the time of arrival in RTP timestamp units for packet i, then for two packets i and j, D is defined as

$$D(i,j) = (r_i - r_j) - (s_i - s_j) = (r_j - s_j) - (r_i - s_i)$$

The interarrival jitter is calculated continuously as each data packet i is received, using this difference D for that packet and the previous packet $i-1$ in order of arrival (not necessarily in sequence), according to the formula

$$J = J + \frac{|D(i,j)| - J}{16}$$

Whenever a reception report is issued, the current value of J is sampled.

Problem 12.

Let x denote the average size of an RTCP packet. The rate at which one sender sends RTCP packets is $1/T$ or

$$\frac{.25 \cdot .05 \cdot \text{session bandwidth}}{S \cdot x}$$

The aggregate rate across all S senders is

$$\frac{.25 \cdot .05 \cdot \text{session bandwidth}}{x}$$

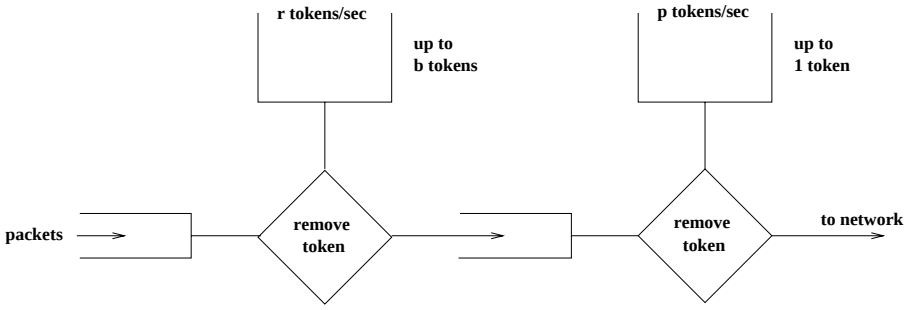


Figure 8: Solution to problem 16

Since the average RTCP packet is x bits, the aggregate rate at which senders send bits is

$$.25 \cdot .05 \cdot \text{session bandwidth}$$

Similarly, the aggregate rate at which receivers send bits is

$$.75 \cdot .05 \cdot \text{session bandwidth}$$

The total aggregate rate is therefore

$$.05 \cdot \text{session bandwidth}$$

Problem 15.

a) One possible sequence is 1 2 1 3 1 2 1 3 1 2 1 3 ...

Another possible sequence is 1 1 2 1 1 3 1 1 2 1 1 3 1 1 2 1 1 3 ...

b) 1 1 3 1 1 3 1 1 3 1 1 3 ...

Problem 16.

See Figure 8.

For the second leaky bucket, $r = p$, $b = 1$.

Problem 18.

Let τ be a time at which flow 1 traffic starts to accumulate in the queue. We refer to τ as the beginning of a flow-1 busy period. Let $t > \tau$ be another time in the same flow-1 busy period. Let $T_1(\tau, t)$ be the amount of flow-1 traffic transmitted in the interval $[\tau, t]$. Clearly,

$$T_1(\tau, t) \geq \frac{W_1}{\sum W_j} R(t - \tau)$$

Let $Q_1(t)$ be the amount of flow-1 traffic in the queue at time t. Clearly,

$$Q_1(t) = b_1 + r_1(t-\tau) - T_1(\tau, t)$$

$$\leq b_1 + r_1(t-\tau) + \frac{W_1}{\sum W_j} R(t-\tau)$$

$$= b_1 + (t-\tau)\left[r_1 - \frac{W_1}{\sum W_j} R\right]$$

Since $r_1 < \frac{W_1}{\sum W_j} R$, $Q_1(t) \leq b_1$. Thus the maximum amount of flow-1 traffic in the queue is b_1. The minimal rate at which this traffic is served is $\frac{W_1 R}{\sum W_j}$.

Thus, the maximum delay for a flow-1 bit is

$$\frac{b_1}{W_1 R / \sum W_j} = d_{max}.$$

Solutions for Chapter 7

Problem 1.

The encoding of "This is an easy problem" is "uasi si my cmiw lokngch".

The decoding of "rmij'u uamu xyj" is "wasn't that fun"

Problem 2.

If Trudy knew that the words "bob" and "alice" appeared in the text, then she would know the ciphertext for b,o,a,l,i,c,e (since "bob" is the only palindrome in the message, and "alice" is the only 5-letter word. If Trudy knows the ciphertext for 7 of the letters, then she only needs to try 19!, rather than 26!, plaintext-ciphertext pairs. The difference between 19! and 26! is 26*25*24...*20, which is 3315312000, or approximately 10^9.

Problem 3.

Every letter in the alphabet appears in the phrase "The quick fox jumps over the lazy brown dog." Given this phrase in a chosen plaintext attack (where the attacker has both the plain text, and the ciphertext), the Caesar cipher would be broken - the intruder would know the ciphertext character for every plaintext character. However, the Vigenere cipher does not alway translate a given plaintext character to the same ciphertext character each time, and hence a Vigenere cipher would not be immediately broken by this chosen plaintext attack.

Problem 4.

We are given $p = 3$ and $q = 11$. We thus have $n = 33$ and $q = 11$. Choose $e = 9$ (it might be a good idea to give students a hint that 9 is a good value to choose, since the resulting calculations are less likely to run into numerical stability problems than other choices for e.) since 3 and $(p-1)*(q-1) = 20$ have no common factors. Choose $d = 9$ also so that $e * d = 81$ and thus $e * d - 1 = 80$ is exactly divisible by 20. We can now perform the RSA encryption and decryption using $n = 33$, $e = 9$ and $d = 9$.

```
letter  m      m**e          ciphertext = m**e mod 33
  h     8      134217728     29
  e     5      1953125       20
  l     12     5159780352    12
  l     12     5159780352    12
  o     15     38443359375   3


ciphertext    c**d              m = c**d mod n   letter
29           14507145975869     8                h
20           512000000000       5                e
12           5159780352         12               l
12           5159780352         12               l
3            19683              15               o
```

Problem 5.

This wouldn't really solve the problem. Just as Bob thinks (incorrectly) that he is authenticating Alice in the first half of Figure 7.14, so too can Trudy fool Alice into thinking (incorrectly) that she is authenticating Bob. The root of the problem that neither Bob nor Alice can tell is the public key they are getting is indeed the public key of Alice of Bob.

Problem 6.

As discussed in section 7.4.2, full blown encryption is more computationally complex than a message digest such as MD5.

Problem 7.

The message

```
I  O  U  2
```

```
0  0  .  8
9  B  O  B
```

has the same checksum.

Problem 8.

If Alice wants to ensure that the KDC is live (that is, the message she will be receiving back from the KDC are not part of a playback attack), she can include a nonce, R0 in the initial message ($K_{A-KDC}(A, B, R0)$) to the KDC. The KDC would then include R) in the reply back to Alice, thus proving the KDC is indeed live. Note that it is already assumed that only the KDC and Alice know the key to decrypt $K_{A-KDC}(A, B, R0)$.

Problem 9.

Alice has received a one-time session key, R1, from the trusted KDC, as well as the KDC-encoded version of R1 to send to Bob. If Alice trusts the KDC then she knows (i) R1 is indeed a one time session key, i.e., like a nonce, and (ii) the person who communicates back to Alice using R1 must be Bob, since the KDC has encoded R1 so only Bob can unencrypt it.

Problem 10.

The message from Alice is encoded using a key that is only known to Alice and the KDC. Therefore the KDC knows (by definition) that anyone using the key must be Alice. It is interesting to think about what damage Trudy could do if she obtains Alice's key. In this case, she can impersonate Alice to anyone; see also the answer to question 11.

Problem 11.

Bob would want to authenticate that Alice is indeed live by sending her a nonce value encoded using R1. This would at least let Bob know he is not being subjected to a playback attack. However, how does Bob know that it is indeed Alice to whom he is talking? Note that Bob obtains the identity of the person with whom he is talking (Alice) from the message that was encoded by the KDC (and routed to Bob via Alice). Thus at least Bob knows that the KDC thinks that Alice is indeed Alice.

Problem 12.

If the KDC goes down, no one can communicate securely, as a first step in communication (see Figure 7.20) is to get the one-time session key from the KDC. If the CA goes down, then as along as the CA's public key is known, one can still communicate securely using previously-issued certificates (recall that once a certificate is issued, the CA is not explicitly involved in any later communication among parties using the CA's certificate. Of course, if the CA goes down, no new certificates can be issued.

Solutions for Chapter 8

Problem 1.

Request response mode will generally have more overhead (measured in terms of the number of messages exchanged) for several reasons. First, each piece of information received by the manager requires two messages: the poll and the response. Trapping generates only a single message to the sender. If the manager really only wants to be notified when a condition occurs, polling has more overhead, since many of the polling messages may indicate that the waited-for condition has not yet occurred. Trapping generates a message only when the condition occurs.

Trapping will also immediately notify the manager when an event occurs. With polling, the manager needs will need to wait for half a polling cycle (on average) between when the event occurs and the manager discovers (via its poll message) that the event has occurred.

If a trap message is lost, the managed device will not send another copy. If a poll message, or its response, is lost the manager would know there has been a lost message (since the reply never arrives). Hence the manager could repoll, if needed.

Problem 2.

Often, the time when network management is most needed is in times of stress, when the network may be severely congested and packets are being lost. With SNMP running over TCP, TCP's congestion control would cause SNMP to back-off and stop sending messages at precisely the time when the network manager needs to send SNMP messages.

Problem 3.

1.3.6.1.2.1.5

Problem 4.

```
3  7  'J' 'a' 'c' 'k' 's' 'o' 'n' 2  2  1  15
```