*Computer Networking: A Top-Down Approach,*

*7th Edition*

Solutions to Review Questions and Problems

Version Date: December 2016

This document contains the solutions to review questions and problems for the 7th edition of *Computer Networking: A Top-Down Approach* by Jim Kurose and Keith Ross. These solutions are being made available to instructors ONLY. Please do NOT copy or distribute this document to others (even other instructors). Please do not post any solutions on a publicly-available Web site. We’ll be happy to provide a copy (up-to-date) of this solution manual ourselves to anyone who asks.

*Acknowledgments:* Over the years, several students and colleagues have helped us prepare this solutions manual. Special thanks goes to Honggang Zhang, Rakesh Kumar, Prithula Dhungel, and Vijay Annapureddy. Also thanks to all the readers who have made suggestions and corrected errors.

All material © copyright 1996-2016 by J.F. Kurose and K.W. Ross. All rights reserved

# Chapter 1 Review Questions

1. There is no difference. Throughout this text, the words “host” and “end system” are used interchangeably. End systems include PCs, workstations, Web servers, mail servers, PDAs, Internet-connected game consoles, etc.
2. From Wikipedia: Diplomatic protocol is commonly described as a set of international courtesy rules. These well-established and time-honored rules have made it easier for nations and people to live and work together. Part of protocol has always been the acknowledgment of the hierarchical standing of all present. Protocol rules are based on the principles of civility.
3. Standards are important for protocols so that people can create networking systems and products that interoperate.
4. 1. Dial-up modem over telephone line: home; 2. DSL over telephone line: home or small office; 3. Cable to HFC: home; 4. 100 Mbps switched Ethernet: enterprise; 5. Wifi (802.11): home and enterprise: 6. 3G and 4G: wide-area wireless.
5. HFC bandwidth is shared among the users. On the downstream channel, all packets emanate from a single source, namely, the head end. Thus, there are no collisions in the downstream channel.
6. In most American cities, the current possibilities include: dial-up; DSL; cable modem; fiber-to-the-home.

7. Ethernet LANs have transmission rates of 10 Mbps, 100 Mbps, 1 Gbps and 10 Gbps.

8. Today, Ethernet most commonly runs over twisted-pair copper wire. It also can run over fibers optic links.

9. Dial up modems: up to 56 Kbps, bandwidth is dedicated; ADSL: up to 24 Mbps downstream and 2.5 Mbps upstream, bandwidth is dedicated; HFC, rates up to 42.8 Mbps and upstream rates of up to 30.7 Mbps, bandwidth is shared. FTTH: 2-10Mbps upload; 10-20 Mbps download; bandwidth is not shared.

10. There are two popular wireless Internet access technologies today:

1. Wifi (802.11) In a wireless LAN, wireless users transmit/receive packets to/from an base station (i.e., wireless access point) within a radius of few tens of meters. The base station is typically connected to the wired Internet and thus serves to connect wireless users to the wired network.
2. 3G and 4G wide-area wireless access networks. In these systems, packets are transmitted over the same wireless infrastructure used for cellular telephony, with the

base station thus being managed by a telecommunications provider. This provides wireless access to users within a radius of tens of kilometers of the base station.

11. At time t0 the sending host begins to transmit. At time *t1 = L/R1*, the sending host completes transmission and the entire packet is received at the router (no propagation delay). Because the router has the entire packet at time *t1*, it can begin to transmit the packet to the receiving host at time *t1*. At time *t2 = t1 + L/R2*, the router completes transmission and the entire packet is received at the receiving host (again, no propagation delay). Thus, the end-to-end delay is *L/R1 + L/R2*.

12. A circuit-switched network can guarantee a certain amount of end-to-end bandwidth for the duration of a call. Most packet-switched networks today (including the Internet) cannot make any end-to-end guarantees for bandwidth. FDM requires sophisticated analog hardware to shift signal into appropriate frequency bands.

13. a) 2 users can be supported because each user requires half of the link bandwidth.

b) Since each user requires 1Mbps when transmitting, if two or fewer users transmit simultaneously, a maximum of 2Mbps will be required. Since the available bandwidth of the shared link is 2Mbps, there will be no queuing delay before the link. Whereas, if three users transmit simultaneously, the bandwidth required will be 3Mbps which is more than the available bandwidth of the shared link. In this case, there will be queuing delay before the link.

c) Probability that a given user is transmitting = 0.2

d) Probability that all three users are transmitting simultaneously = 

=(0.2)3 = 0.008*.* Since the queue grows when all the users are transmitting, the fraction of time during which the queue grows (which is equal to the probability that all three users are transmitting simultaneously) is 0.008.

14. If the two ISPs do not peer with each other, then when they send traffic to each other they have to send the traffic through a provider ISP (intermediary), to which they have to pay for carrying the traffic. By peering with each other directly, the two ISPs can reduce their payments to their provider ISPs. An Internet Exchange Points (IXP) (typically in a standalone building with its own switches) is a meeting point where multiple ISPs can connect and/or peer together. An ISP earns its money by charging each of the the ISPs that connect to the IXP a relatively small fee, which may depend on the amount of traffic sent to or received from the IXP.

15. Google's private network connects together all its data centers, big and small. Traffic between the Google data centers passes over its private network rather than over the public Internet. Many of these data centers are located in, or close to, lower tier ISPs. Therefore, when Google delivers content to a user, it often can bypass higher tier ISPs. What motivates content providers to create these networks? First, the content

provider has more control over the user experience, since it has to use few intermediary ISPs. Second, it can save money by sending less traffic into provider networks. Third, if ISPs decide to charge more money to highly profitable content providers (in countries where net neutrality doesn't apply), the content providers can avoid these extra payments.

16. The delay components are processing delays, transmission delays, propagation delays, and queuing delays. All of these delays are fixed, except for the queuing delays, which are variable.

17. a) 1000 km, 1 Mbps, 100 bytes

b) 100 km, 1 Mbps, 100 bytes

18. 10msec; d/s; no; no

19. a) 500 kbps

b) 64 seconds

c) 100kbps; 320 seconds

20. End system A breaks the large file into chunks. It adds header to each chunk, thereby generating multiple packets from the file. The header in each packet includes the IP address of the destination (end system B). The packet switch uses the destination IP address in the packet to determine the outgoing link. Asking which road to take is analogous to a packet asking which outgoing link it should be forwarded on, given the packet’s destination address.

21. The maximum emission rate is 500 packets/sec and the maximum transmission rate is

350 packets/sec. The corresponding traffic intensity is 500/350 =1.43 > 1. Loss will eventually occur for each experiment; but the time when loss first occurs will be different from one experiment to the next due to the randomness in the emission process.

22. Five generic tasks are error control, flow control, segmentation and reassembly, multiplexing, and connection setup. Yes, these tasks can be duplicated at different layers. For example, error control is often provided at more than one layer.

23. The five layers in the Internet protocol stack are – from top to bottom – the application layer, the transport layer, the network layer, the link layer, and the physical layer. The principal responsibilities are outlined in Section 1.5.1.

24. Application-layer message: data which an application wants to send and passed onto the transport layer; transport-layer segment: generated by the transport layer and encapsulates application-layer message with transport layer header; network-layer datagram: encapsulates transport-layer segment with a network-layer header; link-layer frame: encapsulates network-layer datagram with a link-layer header.

25. Routers process network, link and physical layers (layers 1 through 3). (This is a little bit of a white lie, as modern routers sometimes act as firewalls or caching components, and process Transport layer as well.) Link layer switches process link and physical layers (layers 1 through2). Hosts process all five layers.

26. a) Virus

Requires some form of human interaction to spread. Classic example: E-mail viruses.

b) Worms

No user replication needed. Worm in infected host scans IP addresses and port numbers, looking for vulnerable processes to infect.

27. Creation of a botnet requires an attacker to find vulnerability in some application or system (e.g. exploiting the buffer overflow vulnerability that might exist in an application). After finding the vulnerability, the attacker needs to scan for hosts that are vulnerable. The target is basically to compromise a series of systems by exploiting that particular vulnerability. Any system that is part of the botnet can automatically scan its environment and propagate by exploiting the vulnerability. An important property of such botnets is that the originator of the botnet can remotely control and issue commands to all the nodes in the botnet. Hence, it becomes possible for the attacker to issue a command to all the nodes, that target a single node (for example, all nodes in the botnet might be commanded by the attacker to send a TCP SYN message to the target, which might result in a TCP SYN flood attack at the target).

28. Trudy can pretend to be Bob to Alice (and vice-versa) and partially or completely modify the message(s) being sent from Bob to Alice. For example, she can easily change the phrase “Alice, I owe you $1000” to “Alice, I owe you $10,000”. Furthermore, Trudy can even drop the packets that are being sent by Bob to Alice (and vise-versa), even if the packets from Bob to Alice are encrypted.

# Chapter 1 Problems

### 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 withdraw 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:

# Chapter 1 Problems

### 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 withdraw 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

### At time N\*(L/R) the first packet has reached the destination, the second packet is stored in the last router, the third packet is stored in the next-to-last router, etc. At time N\*(L/R) + L/R, the second packet has reached the destination, the third packet is stored in the last router, etc. Continuing with this logic, we see that at time N\*(L/R) + (P-1)\*(L/R) = (N+P-1)\*(L/R) all packets have reached the destination.

### Problem 3

a) A circuit-switched network would be well suited to the application, 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 without significant waste. In addition, the overhead costs of setting up and tearing down connections are amortized over the lengthy duration of a typical application session.

b) In the worst case, all the applications simultaneously transmit over one or more network links. However, since each link has sufficient bandwidth to handle the sum of all of the applications' data rates, no congestion (very little queuing) will occur. Given such generous link capacities, the network does not need congestion control mechanisms.

### Problem 4

1. Between the switch in the upper left and the switch in the upper right we can have 4 connections. Similarly we can have four connections between each of the 3 other pairs of adjacent switches. Thus, this network can support up to 16 connections.
2. We can *4* connections passing through the switch in the upper-right-hand corner and another *4* connections passing through the switch in the lower-left-hand corner, giving a total of *8* connections.
3. Yes. For the connections between A and C, we route two connections through B and two connections through D. For the connections between B and D, we route two connections through A and two connections through C. In this manner, there are at most 4 connections passing through any link.

### Problem 5

Tollbooths are 75 km apart, and the cars propagate at 100km/hr. A tollbooth services a car at a rate of one car every 12 seconds.

a) There are ten cars. It takes 120 seconds, or 2 minutes, for the first tollbooth to service the 10 cars. Each of these cars has a propagation delay of 45 minutes (travel 75 km) before arriving at the second tollbooth. Thus, all the cars are lined up before the second tollbooth after 47 minutes. The whole process repeats itself for traveling between the second and third tollbooths. It also takes 2 minutes for the third tollbooth to service the 10 cars. Thus the total delay is 96 minutes.

b) Delay between tollbooths is 8\*12 seconds plus 45 minutes, i.e., 46 minutes and 36 seconds. The total delay is twice this amount plus 8\*12 seconds, i.e., 94 minutes and 48 seconds.

### Problem 6

a)  seconds.

b)  seconds.

c)  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

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

sec=7msec.

The time required to transmit the packet is

sec=sec.

Propagation delay = 10 msec.

The delay until decoding is

7msec +sec + 10msec = 17.224msec

A similar analysis shows that all bits experience a delay of 17.224 msec.

### Problem 8

a) 20 users can be supported.

b) .

c) .

d) .

We use the central limit theorem to approximate this probability. Let  be independent random variables such that .

“21 or more users”







when  is a standard normal r.v. Thus “21 or more users”.

### Problem 9

1. 10,000
2. 

### Problem 10

The first end system requires *L/R1* to transmit the packet onto the first link; the packet propagates over the first link in *d1/s1*; the packet switch adds a processing delay of *dproc*; after receiving the entire packet, the packet switch connecting the first and the second link requires *L/R2* to transmit the packet onto the second link; the packet propagates over the second link in *d2/s2*. Similarly, we can find the delay caused by the second switch and the third link: *L/R3*, *dproc*, and *d3/s3*.

Adding these five delays gives

*dend-end = L/R1 + L/R2 + L/R3 + d1/s1 + d2/s2 + d3/s3+ dproc+ dproc*

To answer the second question, we simply plug the values into the equation to get 6 + 6 + 6 + 20+16 + 4 + 3 + 3 = 64 msec.

### Problem 11

Because bits are immediately transmitted, the packet switch does not introduce any delay; in particular, it does not introduce a transmission delay. Thus,

*dend-end = L/R + d1/s1 + d2/s2+ d3/s3*

For the values in Problem 10, we get 6 + 20 + 16 + 4 = 46 msec.

### Problem 12

The arriving packet must first wait for the link to transmit 4.5 \*1,500 bytes = 6,750 bytes or 54,000 bits. Since these bits are transmitted at 2 Mbps, the queuing delay is 27 msec. Generally, the queuing delay is (*nL* + (*L* - *x*))/*R*.

### Problem 13

1. The queuing delay is 0 for the first transmitted packet, *L/R* for the second transmitted packet, and generally, *(n-1)L/R* for the *nth* transmitted packet. Thus, the average delay for the *N* packets is:

*(L/R + 2L/R + ....... + (N-1)L/R)/N*

*= L/(RN) \* (1 + 2 + ..... + (N-1))*

*= L/(RN) \* N(N-1)/2*

*= LN(N-1)/(2RN)*

*= (N-1)L/(2R)*

Note that here we used the well-known fact:

*1 + 2 + ....... + N = N(N+1)/2*

1. It takes  seconds to transmit the  packets. Thus, the buffer is empty when a each batch of  packets arrive. Thus, the average delay of a packet across all batches is the average delay within one batch, i.e., (*N-*1)*L*/*2R*.

### Problem 14

1. The transmission delay is . The total delay is



1. Let .

Total delay = 

For x=0, the total delay =0; as we increase x, total delay increases, approaching infinity as x approaches 1/a.

### Problem 15

Total delay .

### Problem 16

The total number of packets in the system includes those in the buffer and the packet that is being transmitted. So, N=10+1.

Because , so (10+1)=a\*(queuing delay + transmission delay). That is,

11=a\*(0.01+1/100)=a\*(0.01+0.01). Thus, a=550 packets/sec.

### Problem 17

1. There are  nodes (the source host and the  routers). Let denote the processing delay at the th node. Let  be the transmission rate of the th link and let

. Let  be the propagation delay across the th link. Then

.

1. Let  denote the average queuing delay at node . Then

.

### Problem 18

On linux you can use the command

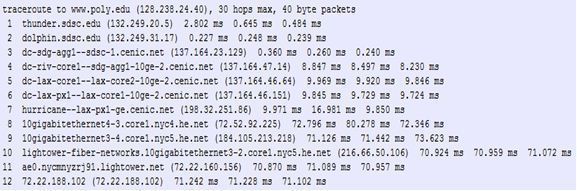
traceroute www.targethost.com

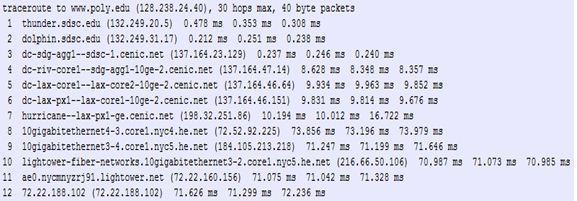
and in the Windows command prompt you can use

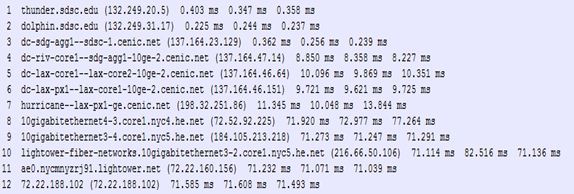
tracert www.targethost.com

In either case, you will get three delay measurements. For those three measurements you can calculate the mean and standard deviation. Repeat the experiment at different times of the day and comment on any changes.

Here is an example solution:

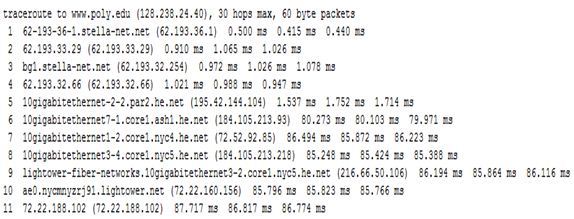


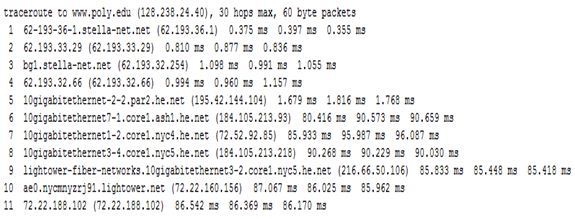


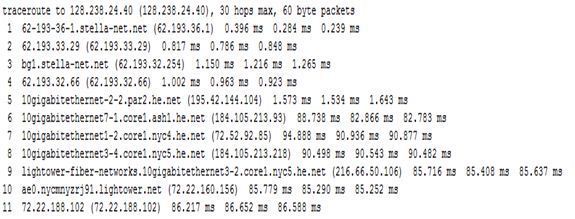


Traceroutes between San Diego Super Computer Center and [www.poly.edu](http://www.poly.edu)

1. The average (mean) of the round-trip delays at each of the three hours is 71.18 ms, 71.38 ms and 71.55 ms, respectively. The standard deviations are 0.075 ms, 0.21 ms, 0.05 ms, respectively.
2. In this example, the traceroutes have 12 routers in the path at each of the three hours. No, the paths didn’t change during any of the hours.
3. Traceroute packets passed through four ISP networks from source to destination. Yes, in this experiment the largest delays occurred at peering interfaces between adjacent ISPs.







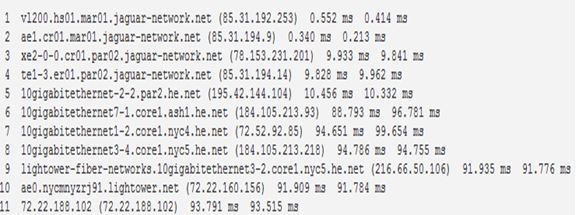
Traceroutes from [www.stella-net.net](http://www.stella-net.net) (France) to [www.poly.edu](http://www.poly.edu) (USA).

1. The average round-trip delays at each of the three hours are 87.09 ms, 86.35 ms and 86.48 ms, respectively. The standard deviations are 0.53 ms, 0.18 ms, 0.23 ms, respectively. In this example, there are 11 routers in the path at each of the three hours. No, the paths didn’t change during any of the hours. Traceroute packets passed three ISP networks from source to destination. Yes, in this experiment the largest delays occurred at peering interfaces between adjacent ISPs.

### Problem 19

An example solution:





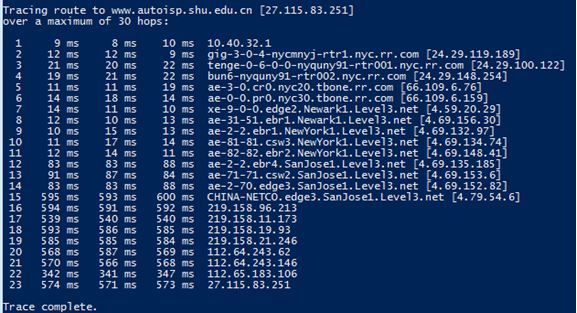
Traceroutes from two different cities in France to New York City in United States

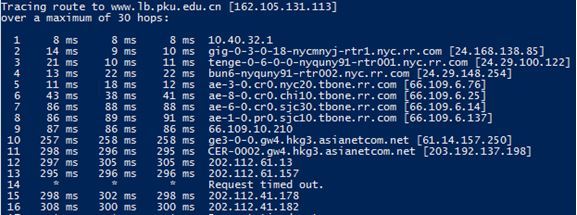
1. In these traceroutes from two different cities in France to the same destination host in United States, seven links are in common including the transatlantic link.





1. In this example of traceroutes from one city in France and from another city in Germany to the same host in United States, three links are in common including the transatlantic link.





Traceroutes to two different cities in China from same host in United States

1. Five links are common in the two traceroutes. The two traceroutes diverge before reaching China

### Problem 20

Throughput = *min{Rs, Rc, R/M}*

### Problem 21

If only use one path, the max throughput is given by:

.

If use all paths, the max throughput is given by .

### Problem 22

Probability of successfully receiving a packet is: ps= (1-p)N.

The number of transmissions needed to be performed until the packet is successfully received by the client is a geometric random variable with success probability ps. Thus, the average number of transmissions needed is given by: 1/ps . Then, the average number of re-transmissions needed is given by: 1/ps -1.

### Problem 23

Let’s call the first packet A and call the second packet B.

1. If the bottleneck link is the first link, then packet B is queued at the first link waiting for the transmission of packet A. So the packet inter-arrival time at the destination is simply *L/Rs*.
2. If the second link is the bottleneck link and both packets are sent back to back, it must be true that the second packet arrives at the input queue of the second link before the second link finishes the transmission of the first packet. That is,

*L/Rs + L/Rs + dprop < L/Rs + dprop + L/Rc*

The left hand side of the above inequality represents the time needed by the second packet to *arrive at* the input queue of the second link (the second link has not started

transmitting the second packet yet). The right hand side represents the time needed by the first packet to finish its transmission onto the second link.

If we send the second packet *T* seconds later, we will ensure that there is no queuing delay for the second packet at the second link if we have:

*L/Rs + L/Rs + dprop + T >= L/Rs + dprop + L/Rc*

Thus, the minimum value of T is *L/Rc* − *L/Rs* .

### Problem 24

40 terabytes = 40 \* 1012 \* 8 bits. So, if using the dedicated link, it will take 40 \* 1012 \* 8 / (100 \*106 ) =3200000 seconds = 37 days. But with FedEx overnight delivery, you can guarantee the data arrives in one day, and it should cost less than $100.

### Problem 25

1. 160,000 bits
2. 160,000 bits
3. The bandwidth-delay product of a link is the maximum number of bits that can be in the link.
4. the width of a bit = length of link / bandwidth-delay product, so 1 bit is 125 meters long, which is longer than a football field
5. s/R

### Problem 26

*s*/*R*=20000km, then *R*=*s*/20000km= 2.5\*108/(2\*107)= 12.5 bps

### Problem 27

1. 80,000,000 bits
2. 800,000 bits, this is because that the maximum number of bits that will be in the link at any given time = min(bandwidth delay product, packet size) = 800,000 bits.
3. .25 meters

### Problem 28

1. *ttrans + tprop* = 400 msec + 80 msec = 480 msec.
2. 20 \* (*ttrans + 2 tprop*) = 20\*(20 msec + 80 msec) = 2 sec.
3. Breaking up a file takes longer to transmit because each data packet and its corresponding acknowledgement packet add their own propagation delays.

### Problem 29

Recall geostationary satellite is 36,000 kilometers away from earth surface.

1. 150 msec
2. 1,500,000 bits
3. 600,000,000 bits

### Problem 30

Let’s suppose the passenger and his/her bags correspond to the data unit arriving to the top of the protocol stack. When the passenger checks in, his/her bags are checked, and a tag is attached to the bags and ticket. This is additional information added in the Baggage layer if Figure 1.20 that allows the Baggage layer to implement the service or separating the passengers and baggage on the sending side, and then reuniting them (hopefully!) on the destination side. When a passenger then passes through security and additional stamp is often added to his/her ticket, indicating that the passenger has passed through a security check. This information is used to ensure (e.g., by later checks for the security information) secure transfer of people.

### Problem 31

1. Time to send message from source host to first packet switch = With store-and-forward switching, the total time to move message from source host to destination host = 
2. Time to send 1st packet from source host to first packet switch = . . Time at which 2nd packet is received at the first switch = time at which 1st packet is received at the second switch = 
3. Time at which 1st packet is received at the destination host = . After this, every 5msec one packet will be received; thus time at which last (800th) packet is received = . It can be seen that delay in using message segmentation is significantly less (almost 1/3rd).
4. Without message segmentation, if bit errors are not tolerated, if there is a single bit error, the whole message has to be retransmitted (rather than a single packet).
5. Without message segmentation, huge packets (containing HD videos, for example) are sent into the network. Routers have to accommodate these huge

packets. Smaller packets have to queue behind enormous packets and suffer unfair delays.

1. Packets have to be put in sequence at the destination.
2. Message segmentation results in many smaller packets. Since header size is usually the same for all packets regardless of their size, with message segmentation the total amount of header bytes is more.

### Problem 32

Yes, the delays in the applet correspond to the delays in the Problem 31.The propagation delays affect the overall end-to-end delays both for packet switching and message switching equally.

### 

### Problem 33

There are *F*/*S* packets. Each packet is S=80 bits. Time at which the last packet is received at the first router is sec. At this time, the first F/S-2 packets are at the destination, and the F/S-1 packet is at the second router. The last packet must then be transmitted by the first router and the second router, with each transmission taking sec. Thus delay in sending the whole file is

To calculate the value of S which leads to the minimum delay,



**Problem 34**

The circuit-switched telephone networks and the Internet are connected together at "gateways". When a Skype user (connected to the Internet) calls an ordinary telephone, a circuit is established between a gateway and the telephone user over the circuit switched network. The skype user's voice is sent in packets over the Internet to the gateway. At the gateway, the voice signal is reconstructed and then sent over the circuit. In the other direction, the voice signal is sent over the circuit switched network to the gateway. The gateway packetizes the voice signal and sends the voice packets to the Skype user.

# Chapter 2 Review Questions

1. The Web: HTTP; file transfer: FTP; remote login: Telnet; e-mail: SMTP; BitTorrent file sharing: BitTorrent protocol
2. Network architecture refers to the organization of the communication process into layers (e.g., the five-layer Internet architecture). Application architecture, on the other hand, is designed by an application developer and dictates the broad structure of the application (e.g., client-server or P2P).
3. The process which initiates the communication is the client; the process that waits to be contacted is the server.
4. No. In a P2P file-sharing application, the peer that is receiving a file is typically the client and the peer that is sending the file is typically the server.
5. The IP address of the destination host and the port number of the socket in the destination process.
6. You would use UDP. With UDP, the transaction can be completed in one roundtrip time (RTT) - the client sends the transaction request into a UDP socket, and the server sends the reply back to the client's UDP socket. With TCP, a minimum of two RTTs are needed - one to set-up the TCP connection, and another for the client to send the request, and for the server to send back the reply.
7. One such example is remote word processing, for example, with Google docs. However, because Google docs runs over the Internet (using TCP), timing guarantees are not provided.
8. a) Reliable data transfer

TCP provides a reliable byte-stream between client and server but UDP does not.

b) A guarantee that a certain value for throughput will be maintained

Neither

c) A guarantee that data will be delivered within a specified amount of time

Neither

d) Confidentiality (via encryption)

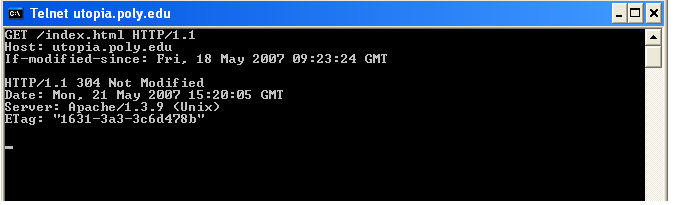
Neither

1. SSL operates at the application layer. The SSL socket takes unencrypted data from the application layer, encrypts it and then passes it to the TCP socket. If the application developer wants TCP to be enhanced with SSL, she has to include the SSL code in the application.
2. A protocol uses handshaking if the two communicating entities first exchange control packets before sending data to each other. SMTP uses handshaking at the application layer whereas HTTP does not.
3. The applications associated with those protocols require that all application data be received in the correct order and without gaps. TCP provides this service whereas UDP does not.
4. When the user first visits the site, the server creates a unique identification number, creates an entry in its back-end database, and returns this identification number as a cookie number. This cookie number is stored on the user’s host and is managed by the browser. During each subsequent visit (and purchase), the browser sends the cookie number back to the site. Thus the site knows when this user (more precisely, this browser) is visiting the site.

1. Web caching can bring the desired content “closer” to the user, possibly to the same LAN to which the user’s host is connected. Web caching can reduce the delay for all objects, even objects that are not cached, since caching reduces the traffic on links.
2. Telnet is not available in Windows 7 by default. to make it available, go to Control Panel, Programs and Features, Turn Windows Features On or Off, Check Telnet client. To start Telnet, in Windows command prompt, issue the following command

> telnet webserverver 80

where "webserver" is some webserver. After issuing the command, you have established a TCP connection between your client telnet program and the web server. Then type in an HTTP GET message. An example is given below:



Since the index.html page in this web server was not modified since Fri, 18 May 2007 09:23:34 GMT, and the above commands were issued on Sat, 19 May 2007, the server returned "304 Not Modified". Note that the first 4 lines are the GET message and header lines inputed by the user, and the next 4 lines (starting from HTTP/1.1 304 Not Modified) is the response from the web server.

1. A list of several popular messaging apps: WhatsApp, Facebook Messenger, WeChat, and Snapchat. These apps use the different protocols than SMS.
2. The message is first sent from Alice’s host to her mail server over HTTP. Alice’s mail server then sends the message to Bob’s mail server over SMTP. Bob then transfers the message from his mail server to his host over POP3.

17.

|  |  |
| --- | --- |
| Received: | from 65.54.246.203 (EHLO bay0-omc3-s3.bay0.hotmail.com) (65.54.246.203) by mta419.mail.mud.yahoo.com with SMTP; Sat, 19 May 2007 16:53:51 -0700 |
| Received: | from hotmail.com ([65.55.135.106]) by bay0-omc3-s3.bay0.hotmail.com with Microsoft SMTPSVC(6.0.3790.2668); Sat, 19 May 2007 16:52:42 -0700 |
| Received: | from mail pickup service by hotmail.com with Microsoft SMTPSVC; Sat, 19 May 2007 16:52:41 -0700 |
| Message-ID: | <BAY130-F26D9E35BF59E0D18A819AFB9310@phx.gbl> |
| Received: | from 65.55.135.123 by by130fd.bay130.hotmail.msn.com with HTTP; Sat, 19 May 2007 23:52:36 GMT |
| From: | "prithula dhungel" <prithuladhungel@hotmail.com> |
| To: | [prithula@yahoo.com](mailto:prithula@yahoo.com) |
| Bcc: |  |
| Subject: | Test mail |
| Date: | Sat, 19 May 2007 23:52:36 +0000 |
| Mime-Version: | 1.0 |
| Content-Type: | Text/html; format=flowed |
| Return-Path: | [prithuladhungel@hotmail.com](mailto:prithuladhungel@hotmail.com) |

**Figure: A sample mail message header**

Received: This header field indicates the sequence in which the SMTP servers send and receive the mail message including the respective timestamps.

In this example there are 4 “Received:” header lines. This means the mail message passed through 5 different SMTP servers before being delivered to the receiver’s mail box. The last (forth) “Received:” header indicates the mail message flow from the SMTP server of the sender to the second SMTP server in the chain of servers. The sender’s SMTP server is at address 65.55.135.123 and the second SMTP server in the chain is by130fd.bay130.hotmail.msn.com.

The third “Received:” header indicates the mail message flow from the second SMTP server in the chain to the third server, and so on.

Finally, the first “Received:” header indicates the flow of the mail messages from the forth SMTP server to the last SMTP server (i.e. the receiver’s mail server) in the chain.

Message-id: The message has been given this number BAY130-F26D9E35BF59E0D18A819AFB9310@phx.gbl (by bay0-omc3-s3.bay0.hotmail.com. Message-id is a unique string assigned by the mail system when the message is first created.

From: This indicates the email address of the sender of the mail. In the given example, the sender is “prithuladhungel@hotmail.com”

To: This field indicates the email address of the receiver of the mail. In the example, the receiver is “prithula@yahoo.com”

Subject: This gives the subject of the mail (if any specified by the sender). In the example, the subject specified by the sender is “Test mail”

Date: The date and time when the mail was sent by the sender. In the example, the sender sent the mail on 19th May 2007, at time 23:52:36 GMT.

Mime-version: MIME version used for the mail. In the example, it is 1.0.

Content-type: The type of content in the body of the mail message. In the example, it is “text/html”.

Return-Path: This specifies the email address to which the mail will be sent if the receiver of this mail wants to reply to the sender. This is also used by the sender’s mail server for bouncing back undeliverable mail messages of mailer-daemon error messages. In the example, the return path is “prithuladhungel@hotmail.com”.

1. With download and delete, after a user retrieves its messages from a POP server, the messages are deleted. This poses a problem for the nomadic user, who may want to access the messages from many different machines (office PC, home PC, etc.). In the download and keep configuration, messages are not deleted after the user retrieves the messages. This can also be inconvenient, as each time the user retrieves the stored messages from a new machine, all of non-deleted messages will be transferred to the new machine (including very old messages).
2. Yes an organization’s mail server and Web server can have the same alias for a host name. The MX record is used to map the mail server’s host name to its IP address.
3. You should be able to see the sender's IP address for a user with an .edu email address. But you will not be able to see the sender's IP address if the user uses a gmail account.
4. It is not necessary that Bob will also provide chunks to Alice. Alice has to be in the top 4 neighbors of Bob for Bob to send out chunks to her; this might not occur even if Alice provides chunks to Bob throughout a 30-second interval.
5. Recall that in BitTorrent, a peer picks a random peer and optimistically unchokes the peer for a short period of time. Therefore, Alice will eventually be optimistically unchoked by one of her neighbors, during which time she will receive chunks from that neighbor.
6. The overlay network in a P2P file sharing system consists of the nodes participating in the file sharing system and the logical links between the nodes. There is a logical link (an “edge” in graph theory terms) from node A to node B if there is a semi-permanent TCP connection between A and B. An overlay network does not include routers.
7. One server placement philosophy is called Enter Deep, which enter deep into the access networks of Internet Service Providers, by deploying server clusters in access ISPs all over the world. The goal is to reduce delays and increase throughput between end users and the CDN servers. Another philosophy is Bring Home, which bring the ISPs home by building large CDN server clusters at a smaller number of sites and typically placing these server clusters in IXPs (Internet Exchange Points). This Bring Home design typically results in lower maintenance and management cost, compared with the enter-deep design philosophy.
8. Other than network-related factors, there are some important factors to consider, such as load-balancing (clients should not be directed to overload clusters), diurnal effects, variations across DNS servers within a network, limited availability of rarely accessed video, and the need to alleviate hot-spots that may arise due to popular video content.

Reference paper:

Torres, Ruben, et al. "Dissecting video server selection strategies in the YouTube CDN." The 31st IEEE International Conference on. Distributed Computing Systems (ICDCS), 2011.

Another factor to consider is ISP delivery cost – the clusters may be chosen so that specific ISPs are used to carry CDN-to-client traffic, taking into account the different cost structures in the contractual relationships between ISPs and cluster operators.

1. With the UDP server, there is no welcoming socket, and all data from different clients enters the server through this one socket. With the TCP server, there is a welcoming socket, and each time a client initiates a connection to the server, a new socket is created. Thus, to support n simultaneous connections, the server would need *n+1* sockets.
2. For the TCP application, as soon as the client is executed, it attempts to initiate a TCP connection with the server. If the TCP server is not running, then the client will fail to make a connection. For the UDP application, the client does not initiate connections (or attempt to communicate with the UDP server) immediately upon execution

# Chapter 2 Problems

### Problem 1

a) F

b) T

c) F

d) F

e) F

### Problem 2

SMS (Short Message Service) is a technology that allows the sending and receiving of text messages between mobile phones over cellular networks. One SMS message can contain data of 140 bytes and it supports languages internationally. The maximum size of a message can be 160 7-bit characters, 140 8-bit characters, or 70 16-bit characters. SMS is realized through the Mobile Application Part (MAP) of the SS#7 protocol, and the Short Message protocol is defined by 3GPP TS 23.040 and 3GPP TS 23.041. In addition, MMS (Multimedia Messaging Service) extends the capability of original text messages, and support sending photos, longer text messages, and other content.

iMessage is an instant messenger service developed by Apple. iMessage supports texts, photos, audios or videos that we send to iOS devices and Macs over cellular data network or WiFi. Apple’s iMessage is based on a proprietary, binary protocol APNs (Apple Push Notification Service).

WhatsApp Messenger is an instant messenger service that supports many mobile platforms such as iOS, Android, Mobile Phone, and Blackberry. WhatsApp users can send each other unlimited images, texts, audios, or videos over cellular data network or WiFi. WhatsApp uses the XMPP protocol (Extensible Messaging and Presence Protocol).

iMessage and WhatsApp are different than SMS because they use data plan to send messages and they work on TCP/IP networks, but SMS use the text messaging plan we purchase from our wireless carrier. Moreover, iMessage and WhatsApp support sending photos, videos, files, etc., while the original SMS can only send text message. Finally, iMessage and WhatsApp can work via WiFi, but SMS cannot.

### Problem 3

Application layer protocols: DNS and HTTP

Transport layer protocols: UDP for DNS; TCP for HTTP

### Problem 4

1. The document request was http://gaia.cs.umass.edu/cs453/index.html. The Host : field indicates the server's name and /cs453/index.html indicates the file name.
2. The browser is running HTTP version 1.1, as indicated just before the first <cr><lf> pair.
3. The browser is requesting a persistent connection, as indicated by the Connection: keep-alive.
4. This is a trick question. This information is not contained in an HTTP message anywhere. So there is no way to tell this from looking at the exchange of HTTP messages alone. One would need information from the IP datagrams (that carried the TCP segment that carried the HTTP GET request) to answer this question.
5. Mozilla/5.0. The browser type information is needed by the server to send different versions of the same object to different types of browsers.

### Problem 5

1. The status code of 200 and the phrase OK indicate that the server was able to locate the document successfully. The reply was provided on Tuesday, 07 Mar 2008 12:39:45 Greenwich Mean Time.
2. The document index.html was last modified on Saturday 10 Dec 2005 18:27:46 GMT.
3. There are 3874 bytes in the document being returned.
4. The first five bytes of the returned document are : <!doc. The server agreed to a persistent connection, as indicated by the Connection: Keep-Alive field

### Problem 6

1. Persistent connections are discussed in section 8 of RFC 2616 (the real goal of this question was to get you to retrieve and read an RFC). Sections 8.1.2 and 8.1.2.1 of the RFC indicate that either the client or the server can indicate to the other that it is going to close the persistent connection. It does so by including the connection-token "close" in the Connection-header field of the http request/reply.
2. HTTP does not provide any encryption services.
3. (From RFC 2616) “Clients that use persistent connections should limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy.”
4. Yes. (From RFC 2616) “A client might have started to send a new request at the same time that the server has decided to close the "idle" connection. From the server's point of view, the connection is being closed while it was idle, but from the client's point of view, a request is in progress.”

### Problem 7

The total amount of time to get the IP address is

.

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



### Problem 8



.





1. Persistent connection with pipelining. This is the default mode of HTTP.



.

Persistent connection without pipelining, without parallel connections.



.

### Problem 9

1. The time to transmit an object of size L over a link or rate R is L/R. The average time is the average size of the object divided by R:

Δ = (850,000 bits)/(15,000,000 bits/sec) = .0567 sec

The traffic intensity on the link is given by βΔ=(16 requests/sec)(.0567 sec/request) = 0.907. Thus, the average access delay is (.0567 sec)/(1 - .907) ≈ .6 seconds. The total average response time is therefore .6 sec + 3 sec = 3.6 sec.

1. The traffic intensity on the access link is reduced by 60% since the 60% of the requests are satisfied within the institutional network. Thus the average access delay is (.0567 sec)/[1 – (.4)(.907)] = .089 seconds. The response time is approximately zero if the request is satisfied by the cache (which happens with probability .6); the average response time is .089 sec + 3 sec = 3.089 sec for cache misses (which happens 40% of the time). So the average response time is (.6)(0 sec) + (.4)(3.089 sec) = 1.24 seconds. Thus the average response time is reduced from 3.6 sec to 1.24 sec.

### Problem 10

Note that each downloaded object can be completely put into one data packet. Let Tp denote the one-way propagation delay between the client and the server.

First consider parallel downloads using non-persistent connections. Parallel downloads would allow 10 connections to share the 150 bits/sec bandwidth, giving each just 15 bits/sec. Thus, the total time needed to receive all objects is given by:

(200/150+*T*p + 200/150 +*T*p + 200/150+*T*p + 100,000/150+ *T*p )

+ (200/(150/10)+*T*p + 200/(150/10) +*T*p + 200/(150/10)+*T*p + 100,000/(150/10)+ *T*p )

= 7377 + 8\**T*p (seconds)

Now consider a persistent HTTP connection. The total time needed is given by:

(200/150+*T*p + 200/150 +*T*p + 200/150+*T*p + 100,000/150+ *T*p )

+ 10\*(200/150+*T*p + 100,000/150+ *T*p )

=7351 + 24\**T*p (seconds)

Assuming the speed of light is 300\*106 m/sec, then Tp=10/(300\*106)=0.03 microsec. Tp is therefore negligible compared with transmission delay.

Thus, we see that persistent HTTP is not significantly faster (less than 1 percent) than the non-persistent case with parallel download.

### Problem 11

1. Yes, because Bob has more connections, he can get a larger share of the link bandwidth.
2. Yes, Bob still needs to perform parallel downloads; otherwise he will get less bandwidth than the other four users.

### Problem 12

Server.py

from socket import \*

serverPort=12000

serverSocket=socket(AF\_INET,SOCK\_STREAM)

serverSocket.bind(('',serverPort))

serverSocket.listen(1)

connectionSocket, addr = serverSocket.accept()

while 1:

sentence = connectionSocket.recv(1024)

print 'From Server:', sentence, '\n' serverSocket.close()

### Problem 13

The MAIL FROM: in SMTP is a message from the SMTP client that identifies the sender of the mail message to the SMTP server. The From: on the mail message itself is NOT an SMTP message, but rather is just a line in the body of the mail message.

### 

### Problem 14

SMTP uses a line containing only a period to mark the end of a message body.

HTTP uses “Content-Length header field” to indicate the length of a message body.

No, HTTP cannot use the method used by SMTP, because HTTP message could be binary data, whereas in SMTP, the message body must be in 7-bit ASCII format.

### Problem 15

MTA stands for Mail Transfer Agent. A host sends the message to an MTA. The message then follows a sequence of MTAs to reach the receiver’s mail reader. We see that this spam message follows a chain of MTAs. An honest MTA should report where it receives the message. Notice that in this message, “asusus-4b96 ([58.88.21.177])” does not report from where it received the email. Since we assume only the originator is dishonest, so “asusus-4b96 ([58.88.21.177])” must be the originator.

### Problem 16

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 user's mailbox. This command is useful for “download and keep”. By maintaining a file that lists the messages retrieved during earlier sessions, the client can use the UIDL command to determine which messages on the server have already been seen.

### Problem 17

a) C: dele 1

C: retr 2

S: (blah blah …

S: ………..blah)

S: .

C: dele 2

C: quit

S: +OK POP3 server signing off

b) C: retr 2

S: blah blah …

S: ………..blah

S: .

C: quit

S: +OK POP3 server signing off

1. C: list

S: 1 498

S: 2 912

S: .

C: retr 1

S: blah …..

S: ….blah

S: .

C: retr 2

S: blah blah …

S: ………..blah

S: .

C: quit

S: +OK POP3 server signing off

### Problem 18

* 1. For a given input of domain name (such as ccn.com), IP address or network administrator name, the *whois* database can be used to locate the corresponding registrar, whois server, DNS server, and so on.
  2. NS4.YAHOO.COM from www.register.com; NS1.MSFT.NET from ww.register.com
  3. *Local Domain:* *www.mindspring.com*

Web servers : www.mindspring.com

207.69.189.21, 207.69.189.22,

207.69.189.23, 207.69.189.24,

207.69.189.25, 207.69.189.26, 207.69.189.27,

207.69.189.28

Mail Servers : mx1.mindspring.com (207.69.189.217)

mx2.mindspring.com (207.69.189.218)

mx3.mindspring.com (207.69.189.219)

mx4.mindspring.com (207.69.189.220)

Name Servers: itchy.earthlink.net (207.69.188.196)

scratchy.earthlink.net (207.69.188.197)

*www.yahoo.com*

Web Servers: www.yahoo.com (216.109.112.135, 66.94.234.13)

Mail Servers: a.mx.mail.yahoo.com (209.191.118.103)

b.mx.mail.yahoo.com (66.196.97.250)

c.mx.mail.yahoo.com (68.142.237.182, 216.39.53.3)

d.mx.mail.yahoo.com (216.39.53.2)

e.mx.mail.yahoo.com (216.39.53.1)

f.mx.mail.yahoo.com (209.191.88.247, 68.142.202.247)

g.mx.mail.yahoo.com (209.191.88.239, 206.190.53.191)

Name Servers: ns1.yahoo.com (66.218.71.63)

ns2.yahoo.com (68.142.255.16)

ns3.yahoo.com (217.12.4.104)

ns4.yahoo.com (68.142.196.63)

ns5.yahoo.com (216.109.116.17)

ns8.yahoo.com (202.165.104.22)

ns9.yahoo.com (202.160.176.146)

*www.hotmail.com*

Web Servers: www.hotmail.com (64.4.33.7, 64.4.32.7)

Mail Servers: mx1.hotmail.com (65.54.245.8, 65.54.244.8, 65.54.244.136)

mx2.hotmail.com (65.54.244.40, 65.54.244.168, 65.54.245.40)

mx3.hotmail.com (65.54.244.72, 65.54.244.200, 65.54.245.72)

mx4.hotmail.com (65.54.244.232, 65.54.245.104, 65.54.244.104)

Name Servers: ns1.msft.net (207.68.160.190)

ns2.msft.net (65.54.240.126)

ns3.msft.net (213.199.161.77)

ns4.msft.net (207.46.66.126)

ns5.msft.net (65.55.238.126)

d) The yahoo web server has multiple IP addresses

www.yahoo.com (216.109.112.135, 66.94.234.13)

e) The address range for Polytechnic University: 128.238.0.0 – 128.238.255.255

f) An attacker can use the *whois* database and nslookup tool to determine the IP address ranges, DNS server addresses, etc., for the target institution.

* 1. By analyzing the source address of attack packets, the victim can use whois to obtain information about domain from which the attack is coming and possibly inform the administrators of the origin domain.

### Problem 19

1. The following delegation chain is used for gaia.cs.umass.edu

a.root-servers.net

E.GTLD-SERVERS.NET

ns1.umass.edu(authoritative)

First command:

dig +norecurse @a.root-servers.net any gaia.cs.umass.edu

;; AUTHORITY SECTION:

edu. 172800 IN NS E.GTLD-SERVERS.NET.

edu. 172800 IN NS A.GTLD-SERVERS.NET.

edu. 172800 IN NS G3.NSTLD.COM.

edu. 172800 IN NS D.GTLD-SERVERS.NET.

edu. 172800 IN NS H3.NSTLD.COM.

edu. 172800 IN NS L3.NSTLD.COM.

edu. 172800 IN NS M3.NSTLD.COM.

edu. 172800 IN NS C.GTLD-SERVERS.NET.

Among all returned edu DNS servers, we send a query to the first one.

dig +norecurse @E.GTLD-SERVERS.NET any gaia.cs.umass.edu

umass.edu. 172800 IN NS ns1.umass.edu.

umass.edu. 172800 IN NS ns2.umass.edu.

umass.edu. 172800 IN NS ns3.umass.edu.

Among all three returned authoritative DNS servers, we send a query to the first one.

dig +norecurse @ns1.umass.edu any gaia.cs.umass.edu

gaia.cs.umass.edu. 21600 IN A 128.119.245.12

1. The answer for google.com could be:

a.root-servers.net

E.GTLD-SERVERS.NET

ns1.google.com(authoritative)

### Problem 20

We can periodically take a snapshot of the DNS caches in the local DNS servers. The Web server that appears most frequently in the DNS caches is the most popular server. This is because if more users are interested in a Web server, then DNS requests for that server are more frequently sent by users. Thus, that Web server will appear in the DNS caches more frequently.

For a complete measurement study, see:

Craig E. Wills, Mikhail Mikhailov, Hao Shang

“Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches”, in IMC'03, October 27­29, 2003, Miami Beach, Florida, USA

### Problem 21

Yes, we can use dig to query that Web site in the local DNS server.

For example, “dig cnn.com” will return the query time for finding cnn.com. If cnn.com was just accessed a couple of seconds ago, an entry for cnn.com is cached in the local DNS cache, so the query time is 0 msec. Otherwise, the query time is large.

### Problem 22

For calculating the minimum distribution time for client-server distribution, we use the following formula:

*Dcs = max {NF/us, F/dmin}*

Similarly, for calculating the minimum distribution time for P2P distribution, we use the following formula:



Where, *F* = 15 Gbits = 15 \* 1024 Mbits

*us* = 30 Mbps

*dmin* = *di*= 2 Mbps

Note, 300Kbps = 300/1024 Mbps.

Client Server

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | N | | |
| 10 | 100 | 1000 |
| u | 300 Kbps | 7680 | 51200 | 512000 |
| 700 Kbps | 7680 | 51200 | 512000 |
| 2 Mbps | 7680 | 51200 | 512000 |

Peer to Peer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | N | | |
| 10 | 100 | 1000 |
| u | 300 Kbps | 7680 | 25904 | 47559 |
| 700 Kbps | 7680 | 15616 | 21525 |
| 2 Mbps | 7680 | 7680 | 7680 |

### Problem 23

1. Consider a distribution scheme in which the server sends the file to each client, in parallel, at a rate of a rate of *us*/*N*. Note that this rate is less than each of the client’s download rate, since by assumption *us*/*N* ≤ *d*min. Thus each client can also receive at rate *us*/*N*. Since each client receives at rate *us*/*N*, the time for each client to receive the entire file is F/( *us*/*N*) = *NF/ us.* Since all the clients receive the file in *NF/ us*, the overall distribution time is also *NF/ us.*
2. Consider a distribution scheme in which the server sends the file to each client, in parallel, at a rate of *d*min. Note that the aggregate rate, *N* *d*min, is less than the server’s link rate *us*, since by assumption *us*/*N* ≥ *d*min. Since each client receives at rate *d*min, the time for each client to receive the entire file is F/ *d*min*.* Since all the clients receive the file in this time, the overall distribution time is also F/ *d*min*.*
3. From Section 2.6 we know that

*DCS* ≥ max {*NF/us, F/d*min} (Equation 1)

Suppose that *us*/*N* ≤ *d*min. Then from Equation 1 we have *DCS* ≥ *NF/us .* But from (a) we have *DCS* ≤ *NF/us .* Combining these two gives:

*DCS* = *NF/us* when *us*/*N* ≤ *d*min. (Equation 2)

We can similarly show that:

*DCS* =*F/d*min when *us*/*N* ≥ *d*min (Equation 3).

Combining Equation 2 and Equation 3 gives the desired result.

### Problem 24

1. Define u = u1 + u2 + ….. + uN. By assumption

us <= (us + u)/N Equation 1

Divide the file into N parts, with the ith part having size (ui/u)F. The server transmits the ith part to peer i at rate *r*i = (*u*i/*u*)*u*s. Note that *r*1 + *r*2 + ….. + *r*N = *u*s, so that the aggregate server rate does not exceed the link rate of the server. Also have each peer i forward the bits it receives to each of the *N-1* peers at rate *r*i. The aggregate forwarding rate by peer i is (N-1)ri. We have

(*N-1*)*r*i = (*N-1*)(*u*s*u*i)/*u* <= *u*i,

where the last inequality follows from Equation 1. Thus the aggregate forwarding rate of peer i is less than its link rate ui.

In this distribution scheme, peer i receives bits at an aggregate rate of



Thus each peer receives the file in F/us.

1. Again define *u* = *u*1 + *u*2 + ….. + *u*N. By assumption

*u*s >= (*u*s + *u*)/*N* Equation 2

Let *r*i = *u*i/(*N-1*) and

*r*N+1 = (*u*s – *u*/(*N-1*))/*N*

In this distribution scheme, the file is broken into *N+1* parts. The server sends bits from the ith part to the ith peer (i = *1, …., N*) at rate ri. Each peer i forwards the bits arriving at rate ri to each of the other N-1 peers. Additionally, the server sends bits from the (*N+1*) st part at rate rN+1 to each of the *N* peers. The peers do not forward the bits from the (*N+1*)st part.

The aggregate send rate of the server is

*r*1+ …. + *r*N + *N* *r*N+1 = *u*/(*N-1*) + *u*s – *u*/(*N-1*) = *u*s

Thus, the server’s send rate does not exceed its link rate. The aggregate send rate of peer i is

(*N-1*)*r*i = *u*i

Thus, each peer’s send rate does not exceed its link rate.

In this distribution scheme, peer i receives bits at an aggregate rate of



Thus each peer receives the file in NF/(us+u).

(For simplicity, we neglected to specify the size of the file part for i = 1, …., N+1. We now provide that here. Let Δ = (us+u)/N be the distribution time. For i = 1, …, N, the ith file part is Fi = ri Δ bits. The (N+1)st file part is FN+1 = rN+1 Δ bits. It is straightforward to show that F1+ ….. + FN+1 = F.)

1. The solution to this part is similar to that of 17 (c). We know from section 2.6 that



Combining this with a) and b) gives the desired result.

### Problem 25

There are *N* nodes in the overlay network. There are *N(N-1)/2* edges.

### Problem 26

Yes. His first claim is possible, as long as there are enough peers staying in the swarm for a long enough time. Bob can always receive data through optimistic unchoking by other peers.

His second claim is also true. He can run a client on each host, let each client “free-ride,” and combine the collected chunks from the different hosts into a single file. He can even write a small scheduling program to make the different hosts ask for different chunks of the file. This is actually a kind of Sybil attack in P2P networks.

### **Problem 27**

### a. N files, under the assumption that we do a one-to-one matching by pairing video versions with audio versions in a decreasing order of quality and rate.

### b. 2N files.

### Problem 28

1. 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.
2. 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.
3. 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.

### Problem 29

In the original program, UDPClient does not specify a port number when it creates the socket. In this case, the code lets the underlying operating system choose a port number. With the additional line, when UDPClient is executed, a UDP socket is created with port number 5432 .

UDPServer needs to know the client port number so that it can send packets back to the correct client socket. Glancing at UDPServer, we see that the client port number is not “hard-wired” into the server code; instead, UDPServer determines the client port number by unraveling the datagram it receives from the client. Thus UDP server will work with any client port number, including 5432. UDPServer therefore does not need to be modified.

Before:

Client socket = x (chosen by OS)

Server socket = 9876

After:

Client socket = 5432

**Problem 30**

Yes, you can configure many browsers to open multiple simultaneous connections to a Web site. The advantage is that you will you potentially download the file faster. The disadvantage is that you may be hogging the bandwidth, thereby significantly slowing down the downloads of other users who are sharing the same physical links.

**Problem 31**

For an application such as remote login (telnet and ssh), a byte-stream oriented protocol is very natural since there is no notion of message boundaries in the application. When a user types a character, we simply drop the character into the TCP connection.

In other applications, we may be sending a series of messages that have inherent boundaries between them. For example, when one SMTP mail server sends another SMTP mail server several email messages back to back. Since TCP does not have a mechanism to indicate the boundaries, the application must add the indications itself, so that receiving side of the application can distinguish one message from the next. If each message were instead put into a distinct UDP segment, the receiving end would be able to

distinguish the various messages without any indications added by the sending side of the application.

**Problem 32**

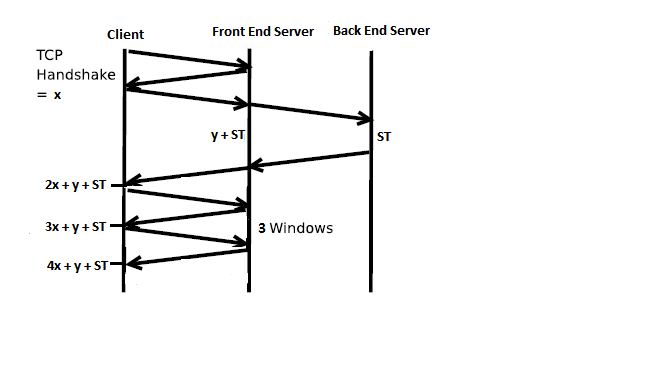
To create a web server, we need to run web server software on a host. Many vendors sell web server software. However, the most popular web server software today is Apache, which is open source and free. Over the years it has been highly optimized by the open-source community.

# Chapter 3 Review Questions

1. Call this protocol Simple Transport Protocol (STP). At the sender side, STP accepts from the sending process a chunk of data not exceeding 1196 bytes, a destination host address, and a destination port number. STP adds a four-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.
2. The segment now has two header fields: a source port field and destination port field. At the sender side, STP accepts a chunk of data not exceeding 1192 bytes, a destination host address, a source port number, and a destination port number. STP creates a segment which contains the application data, source port number, and destination port number. It then 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.
3. No, the transport layer does not have to do anything in the core; the transport layer “lives” in the end systems.
4. 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. The delegate then puts the letter in an envelope and writes the address of the destination house on the envelope. The delegate then 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 takes note of the recipient name written at the top of the letter. The delegate then gives the letter to the family member with this name.
5. No, the mail service does not have to open the envelope; it only examines the address on the envelope.
6. Source port number y and destination port number x.
7. An application developer may not want its application to use TCP’s congestion control, which can throttle the application’s sending rate at times of congestion. Often, designers of IP telephony and IP videoconference applications choose to run their applications over UDP because they want to avoid TCP’s congestion control. Also, some applications do not need the reliable data transfer provided by TCP.
8. Since most firewalls are configured to block UDP traffic, using TCP for video and voice traffic lets the traffic though the firewalls.
9. Yes. The application developer can put reliable data transfer into the application layer protocol. This would require a significant amount of work and debugging, however.
10. 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 addresses to determine the origins of the individual segments.
11. 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 and 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 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.
12. Sequence numbers are required for a receiver to find out whether an arriving packet contains new data or is a retransmission.
13. To handle losses in the channel. If the ACK for a transmitted packet is not received within the duration of the timer for the packet, the packet (or its ACK or NACK) is assumed to have been lost. Hence, the packet is retransmitted.
14. A timer would still be necessary in the protocol rdt 3.0. If the round trip time is known then the only advantage will be that, the sender knows for sure that either the packet or the ACK (or NACK) for the packet has been lost, as compared to the real scenario, where the ACK (or NACK) might still be on the way to the sender, after the timer expires. However, to detect the loss, for each packet, a timer of constant duration will still be necessary at the sender.
16. The packet loss caused a time out after which all the five packets were retransmitted.
17. Loss of an ACK didn’t trigger any retransmission as Go-Back-N uses cumulative acknowledgements.
18. The sender was unable to send sixth packet as the send window size is fixed to 5.
19. When the packet was lost, the received four packets were buffered the receiver. After the timeout, sender retransmitted the lost packet and receiver delivered the buffered packets to application in correct order.
20. Duplicate ACK was sent by the receiver for the lost ACK.
21. The sender was unable to send sixth packet as the send window size is fixed to 5

When a packet was lost, GO-Back-N retransmitted all the packets whereas Selective Repeat retransmitted the lost packet only. In case of lost acknowledgement, selective repeat sent a duplicate ACK and as GO-Back-N used cumulative acknowledgment, so that duplicate ACK was unnecessary.

1. a) false b) false c) true d) false e) true f) false g) false
2. a) 20 bytes b) ack number = 90
3. 3 segments. First segment: seq = 43, ack =80; Second segment: seq = 80, ack = 44; Third segment; seq = 44, ack = 81
4. R/2
5. False, it is set to half of the current value of the congestion window.
6. Let X = RTTFE, Y = RTTBE and ST = Search time. Consider the following timing diagram.



TCP packet exchange diagram between a client and a server (Back End) with a proxy (Front End) between them.

From this diagram we see that the total time is 4X + Y+ ST = 4\*RTTFE + RTTBE + Search time

# **Chapter 3 Problems**

### Problem 1

|  |  |  |
| --- | --- | --- |
|  | source port  numbers | destination port  numbers |
| a) AS | 467 | 23 |
| b) BS | 513 | 23 |
| c) SA | 23 | 467 |
| d) SB | 23 | 513 |

e) Yes.

f) No.

### Problem 2

Suppose the IP addresses of the hosts A, B, and C are a, b, c, respectively. (Note that a, b, c are distinct.)

To host A: Source port =80, source IP address = b, dest port = 26145, dest IP address = a

To host C, left process: Source port =80, source IP address = b, dest port = 7532, dest IP address = c

To host C, right process: Source port =80, source IP address = b, dest port = 26145, dest IP address = c

### Problem 3

Note, wrap around if overflow.





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

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 4

a) Adding the two bytes gives 11000001. Taking the one’s complement gives 00111110.

b) Adding the two bytes gives 01000000; the one’s complement gives 10111111.

c) First byte = 01010100; second byte = 01101101.

### Problem 5

No, the receiver cannot be absolutely certain that no bit errors have occurred. This is because of the manner in which the checksum for the packet is calculated. If the corresponding bits (that would be added together) of two 16-bit words in the packet were 0 and 1 then even if these get flipped to 1 and 0 respectively, the sum still remains the same. Hence, the 1s complement the receiver calculates will also be the same. This means the checksum will verify even if there was transmission error.

### Problem 6

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 7

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 8

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 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 9

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.



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

### Problem 10

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 looks *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 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 11

If the sending of this message were removed, the sending and receiving sides would deadlock, waiting for an event that would never occur. Here’s a scenario:

* Sender sends pkt0, enter the “Wait for ACK0 state”, and waits for a packet back from the receiver
* Receiver is in the “Wait for 0 from below” state, and receives a corrupted packet from the sender. Suppose it does not send anything back, and simply re-enters the ‘wait for 0 from below” state.

Now, the ender is awaiting an ACK of some sort from the receiver, and the receiver is waiting for a data packet form the sender – a deadlock!

### Problem 12

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. 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 number of times packet is sent will increase without bound as approaches infinity.

### Problem 13



### 

### Problem 14

In a NAK only protocol, the loss of packet *x* is only detected by the receiver when packet *x+1* is received. That is, the receivers receives *x-1* and then *x+1,* only when *x+1 is* received does the receiver realize that *x* was missed. If there is a long delay between the transmission of x and the transmission of *x+1,* then it will be a long time until *x* can be recovered, under a NAK only protocol.

On the other hand, if data is being sent often, then recovery under a NAK-only scheme could happen quickly. Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent – a significant reduction in feedback in the NAK-only case over the ACK-only case.

### Problem 15

It takes 12 microseconds (or 0.012 milliseconds) to send a packet, as 1500\*8/109=12 microseconds. In order for the sender to be busy 98 percent of the time, we must have



or  approximately 2451 packets.

### Problem 16

Yes. This actually causes the sender to send a number of pipelined data into the channel.

Yes. Here is one potential problem. If data segments are lost in the channel, then the sender of rdt 3.0 won’t re-send those segments, unless there are some additional mechanism in the application to recover from loss.

**Problem 17**

Wait: send

to B

Wait: receive

from B

rdt\_send(data)

packet=make\_pkt(data)

udt\_send(packet)

rdt\_receive(packet)

extract(packet,data)

deliver\_data(data)

rdt\_send(data)

rdt\_unable\_to\_send(data)

rdt\_send(data)

Rdt\_unable\_to\_send(data)

A

Wait: send

to A

Wait: receive

from A

rdt\_send(data)

packet=make\_pkt(data)

udt\_send(packet)

rdt\_receive(packet)

extract(packet,data)

deliver\_data(data)

rdt\_send(data)

rdt\_unable\_to\_send(data)

rdt\_send(data)

Rdt\_unable\_to\_send(data)

B

### Problem 18

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:

hw11

Figure 2: Sender and receiver for Problem (3.18)

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 19

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.

hw12

Figure 3. Sender and receiver for Problem 3.19(Problem 19)

### Problem 20

rdt\_rcv(rcvpkt)&&from\_B(rcvpkt)

Λ

Figure 4: Receiver side FSM for 3.18

rdt\_rcv(rcvpkt)&&(corrupt(rcvpkt) ||has\_seq0(rcvpkt))&&from\_A(rcvpkt)

sndpkt=make\_pkt(ACK, 0, checksum)

udt\_send(A,sndpkt)

rdt\_rcv(rcvpkt)&&(corrupt(rcvpkt) ||has\_seq1(rcvpkt))&&from\_B(rcvpkt)

sndpkt=make\_pkt(ACK, 1, checksum)

udt\_send(B,sndpkt)

rdt\_rcv(rcvpkt)&&(corrupt(rcvpkt) ||has\_seq1(rcvpkt))&&from\_A(rcvpkt)

sndpkt=make\_pkt(ACK, 1, checksum)

udt\_send(A,sndpkt)

rdt\_rcv(rcvpkt)&&from\_A(rcvpkt)

Λ



rdt\_rcv(rcvpkt)&&(corrupt(rcvpkt) ||has\_seq0(rcvpkt))&&from\_B(rcvpkt)

sndpkt=make\_pkt(ACK, 0, checksum)

udt\_send(B,sndpkt)

rdt\_rcv(rcvpkt)&&not\_corrupt(rcvpkt)&&has\_seq1(rcvpkt)&&from\_A(rcvpkt)

extract(rcvpkt,data)

deliver\_data(data)

sndpkt=make\_pkt(ACK,1,checksum)

udt\_send(A,sndpkt)

rdt\_rcv(rcvpkt)&&from\_A(rcvpkt)

Λ

rdt\_rcv(rcvpkt)&&not\_corrupt(rcvpkt)&&has\_seq0(rcvpkt)&&from\_B(rcvpkt)

extract(rcvpkt,data)

deliver\_data(data)

sndpkt=make\_pkt(ACK,0,checksum)

udt\_send(B,sndpkt)

rdt\_rcv(rcvpkt)&&from\_B(rcvpkt)

Λ

rdt\_rcv(rcvpkt)&&not\_corrupt(rcvpkt) &&has\_seq1(rcvpkt)&&from\_B(rcvpkt)

extract(rcvpkt,data)

deliver\_data(data)

sndpkt=make\_pkt(ACK,1,checksum)

udt\_send(B,sndpkt)

*Sender*

The sender side FSM is exactly same as given in Figure 3.15 in text

### Problem 21

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 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 22

1. Here we have a window size of N=3. Suppose the receiver has received packet k-1, and has ACKed that and all other preceding 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].
2. 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 23

In order to avoid the scenario of Figure 3.27, 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.

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, .

### Problem 24

1. True. Suppose the sender has a window size of 3 and sends packets 1, 2, 3 at . At  the receiver ACKS 1, 2, 3. At   the sender times out and resends 1, 2, 3. At  the receiver receives the duplicates and re-acknowledges 1, 2, 3. At  the sender receives the ACKs that the receiver sent at  and advances its window to 4, 5, 6. At  the sender receives the ACKs 1, 2, 3 the receiver sent at . These ACKs are outside its window.
2. True. By essentially the same scenario as in (a).
3. True.
4. 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 25

1. Consider sending an application message over a transport protocol. With TCP, the application writes data to the connection 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.
2. 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.

### Problem 26

There are  possible sequence numbers.

1. The sequence number does not increment by one with each segment. Rather, it 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. The number of segments is. 66 bytes of header get added to each segment giving a total of 528,857,934 bytes of header. The total number of bytes transmitted is  bytes.

Thus it would take 249 seconds to transmit the file over a 155~Mbps link.

### Problem 27

1. In the second segment from Host A to B, the sequence number is 207, source port number is 302 and destination port number is 80.
2. If the first segment arrives before the second, in the acknowledgement of the first arriving segment, the acknowledgement number is 207, the source port number is 80 and the destination port number is 302.
3. If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment, the acknowledgement number is 127, indicating that it is still waiting for bytes 127 and onwards.

Host B

Host A

Seq = 127, 80 bytes

Seq = 207, 40 bytes

Ack = 207

Timeout interval

Ack = 247

Seq = 127, 80 bytes

Ack = 247

Timeout interval

### Problem 28

Since the link capacity is only 100 Mbps, so host A’s sending rate can be at most 100Mbps. Still, 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 40Mbps. 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 60Mbps.

### Problem 29

1. The server uses special initial sequence number (that is obtained from the hash of source and destination IPs and ports) in order to defend itself against SYN FLOOD attack.
2. No, the attacker cannot create half-open or fully open connections by simply sending and ACK packet to the target. Half-open connections are not possible since a server using SYN cookies does not maintain connection variables and buffers for any connection before full connections are established. For establishing fully open connections, an attacker should know the special initial sequence number corresponding to the (spoofed) source IP address from the attacker. This sequence number requires the "secret" number that each server uses. Since the attacker does not know this secret number, she cannot guess the initial sequence number.
3. No, the sever can simply add in a time stamp in computing those initial sequence numbers and choose a time to live value for those sequence numbers, and discard expired initial sequence numbers even if the attacker replay them.

### Problem 30

1. If timeout values are fixed, then the senders may timeout prematurely. Thus, some packets are re-transmitted even they are not lost.
2. If timeout values are estimated (like what TCP does), then increasing the buffer size certainly helps to increase the throughput of that router. But there might be one potential problem. Queuing delay might be very large, similar to what is shown in Scenario 1.

**Problem 31**

DevRTT = (1- beta) \* DevRTT + beta \* | SampleRTT - EstimatedRTT |

EstimatedRTT = (1-alpha) \* EstimatedRTT + alpha \* SampleRTT

TimeoutInterval = EstimatedRTT + 4 \* DevRTT

After obtaining first SampleRTT 106ms:

DevRTT = 0.75\*5 + 0.25 \* | 106 - 100 | = 5.25ms

EstimatedRTT = 0.875 \* 100 + 0.125 \* 106 = 100.75 ms

TimeoutInterval = 100.75+4\*5.25 = 121.75 ms

After obtaining 120ms:

DevRTT = 0.75\*5.25 + 0.25 \* | 120 – 100.75 | = 8.75 ms

EstimatedRTT = 0.875 \* 100.75 + 0.125 \* 120 = 103.16 ms

TimeoutInterval = 103.16+4\*8.75 = 138.16 ms

After obtaining 140ms:

DevRTT = 0.75\*8.75 + 0.25 \* | 140 – 103.16 | = 15.77 ms

EstimatedRTT = 0.875 \* 103.16 + 0.125 \* 140 = 107.76 ms

TimeoutInterval = 107.76+4\*15.77 = 170.84 ms

After obtaining 90ms:

DevRTT = 0.75\*15.77 + 0.25 \* | 90 – 107.76 | = 16.27 ms

EstimatedRTT = 0.875 \* 107.76 + 0.125 \* 90 = 105.54 ms

TimeoutInterval = 105.54+4\*16.27 =170.62 ms

After obtaining 115ms:

DevRTT = 0.75\*16.27 + 0.25 \* | 115 – 105.54 | = 14.57 ms

EstimatedRTT = 0.875 \* 105.54 + 0.125 \* 115 = 106.72 ms

TimeoutInterval = 106.72+4\*14.57 =165 ms

### Problem 32

a)

Denote  for the estimate after the *n*th sample.











b)





c)





The weight given to past samples decays exponentially.

### Problem 33

Let’s look at what could wrong if TCP measures SampleRTT for a retransmitted segment. Suppose the source sends packet P1, the timer for P1 expires, and the source then sends P2, a new copy of the same packet. Further suppose the source measures SampleRTT for P2 (the retransmitted packet). Finally suppose that shortly after transmitting P2 an acknowledgment for P1 arrives. The source will mistakenly take this acknowledgment as an acknowledgment for P2 and calculate an incorrect value of SampleRTT.

Let’s look at what could be wrong if TCP measures SampleRTT for a retransmitted segment. Suppose the source sends packet P1, the timer for P1 expires, and the source then sends P2, a new copy of the same packet. Further suppose the source measures SampleRTT for P2 (the retransmitted packet). Finally suppose that shortly after transmitting P2 an acknowledgment for P1 arrives. The source will mistakenly take this acknowledgment as an acknowledgment for P2 and calculate an incorrect value of SampleRTT.

### Problem 34

At any given time *t*, SendBase – 1 is the sequence number of the last byte that the sender knows has been received correctly, and in order, at the receiver. The actually last byte received (correctly and in order) at the receiver at time *t* may be greater if there are acknowledgements in the pipe. Thus

SendBase–1 ≤ LastByteRcvd

### Problem 35

When, at time *t*, the sender receives an acknowledgement with value *y*, the sender knows for sure that the receiver has received everything up through *y*-1. The actual last byte received (correctly and in order) at the receiver at time *t* may be greater if y ≤ SendBase or if there are other acknowledgements in the pipe. Thus

y-1 ≤ LastByteRvcd

### Problem 36

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  are correctly received, while n+1 was not received. The designers of the triple duplicate ACK scheme probably felt that waiting for 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 37

1. GoBackN:

A sends 9 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later re-sent segments 2, 3, 4, and 5.

B sends 8 ACKs. They are 4 ACKS with sequence number 1, and 4 ACKS with sequence numbers 2, 3, 4, and 5.

Selective Repeat:

A sends 6 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later re-sent segments 2.

B sends 5 ACKs. They are 4 ACKS with sequence number 1, 3, 4, 5. And there is one ACK with sequence number 2.

TCP:

A sends 6 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later re-sent segments 2.

B sends 5 ACKs. They are 4 ACKS with sequence number 2. There is one ACK with sequence numbers 6. Note that TCP always send an ACK with expected sequence number.

1. TCP. This is because TCP uses fast retransmit without waiting until time out.

### Problem 38

Yes, the sending rate is always roughly cwnd/RTT.

### Problem 39

If the arrival rate increases beyond R/2 in Figure 3.46(b), then the total arrival rate to the queue exceeds the queue’s capacity, resulting in increasing loss as the arrival rate increases. When the arrival rate equals R/2, 1 out of every three packets that leaves the queue is a retransmission. With increased loss, even a larger fraction of the packets leaving the queue will be retransmissions. Given that the maximum departure rate from the queue for one of the sessions is R/2, and given that a third or more will be transmissions as the arrival rate increases, the throughput of successfully deliver data can not increase beyond λout. Following similar reasoning, if half of the packets leaving the queue are retransmissions, and the maximum rate of output packets per session is R/2, then the maximum value of λout is (R/2)/2 or R/4.

### Problem 40

1. TCP slowstart is operating in the intervals [1,6] and [23,26]
2. TCP congestion avoidance is operating in the intervals [6,16] and [17,22]
3. After the 16th transmission round, packet loss is recognized by a triple duplicate ACK. If there was a timeout, the congestion window size would have dropped to 1.
4. After the 22nd transmission round, segment loss is detected due to timeout, and hence the congestion window size is set to 1.
5. The threshold is initially 32, since it is at this window size that slow start stops and congestion avoidance begins.
6. The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 16, the congestion windows size is 42. Hence the threshold is 21 during the 18th transmission round.
7. The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 22, the congestion windows size is 29. Hence the threshold is 14 (taking lower floor of 14.5) during the 24th transmission round.
8. During the 1st transmission round, packet 1 is sent; packet 2-3 are sent in the 2nd transmission round; packets 4-7 are sent in the 3rd transmission round; packets 8-15 are sent in the 4th transmission round; packets 16-31 are sent in the 5th transmission round; packets 32-63 are sent in the 6th transmission round; packets 64 – 96 are sent in the 7th transmission round. Thus packet 70 is sent in the 7th transmission round.
9. The threshold will be set to half the current value of the congestion window (8) when the loss occurred and congestion window will be set to the new threshold value + 3 MSS . Thus the new values of the threshold and window will be 4 and 7 respectively.
10. threshold is 21, and congestion window size is 1.
11. round 17, 1 packet; round 18, 2 packets; round 19, 4 packets; round 20, 8 packets; round 21, 16 packets; round 22, 21 packets. So, the total number is 52.

### Problem 41

Refer to Figure 5. In Figure 5(a), the ratio of the linear decrease on loss between connection 1 and connection 2 is the same - as ratio of the linear increases: unity. In this case, the throughputs never move off of the AB line segment. In Figure 5(b), 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.

tcpFair

Figure 5: Lack of TCP convergence with linear increase, linear decrease

### Problem 42

If TCP were a stop-and-wait protocol, then the doubling of the time out interval would suffice as a congestion control mechanism. However, TCP uses pipelining (and is therefore not a stop-and-wait protocol), which allows the sender to have multiple outstanding unacknowledged segments. The doubling of the timeout interval does not prevent a TCP sender from sending a large number of first-time-transmitted packets into the network, even when the end-to-end path is highly congested. Therefore a congestion-control mechanism is needed to stem the flow of “data received from the application above” when there are signs of network congestion.

### Problem 43

In this problem, there is no danger in overflowing the receiver since the receiver’s receive buffer can hold the entire file. Also, because there is no loss and acknowledgements are returned before timers expire, TCP congestion control does not throttle the sender. However, the process in host A will not continuously pass data to the socket because the send buffer will quickly fill up. Once the send buffer becomes full, the process will pass data at an average rate or R << S.

### Problem 44

a)  It takes 1 RTT to increase CongWin to 7 MSS; 2 RTTs to increase to 8 MSS;  3 RTTs to increase to 9 MSS; 4 RTTs to increase to 10 MSS; 5 RTTs to increase to 11 MSS; 6 RTTs to increase to 12 MSS.

b)    In the first RTT 6 MSS was sent; in the second RTT 7 MSS was sent; in the third RTT 8 MSS was sent; in the fourth RTT 9 MSS was sent; in the fifth RTT, 10 MSS was sent; and in the sixth RTT, 11 MSS was sent. Thus, up to time 6 RTT,

6+7+8+9+10+11 = 51 MSS were sent.  Thus, we can say that the average throughput up to time 6 RTT was (51 MSS)/(6 RTT) = 8.5 MSS/RTT.

**Problem 45**

1. The loss rate,, 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











Thus the loss rate is



b) For  large, . Thus  or . From the text, we therefore have

average throughput 



### Problem 46

1. Let W denote the max window size measured in segments. Then, W\*MSS/RTT = 10Mbps, as packets will be dropped if the maximum sending rate exceeds link capacity. Thus, we have W\*1500\*8/0.15=10\*10^6, then W is about 125 segments.
2. As congestion window size varies from W/2 to W, then the average window size is 0.75W=94 (ceiling of 93.75) segments. Average throughput is 94\*1500\*8/0.15 =7.52Mbps.
3. When there is a packet loss, W becomes W/2, i.e., 125/2=62.  
     (125 - 62) \*0.15 = 9.45 seconds, as the number of RTTs (that this TCP connections needs in order to increase its window size from 62 to 125) is 63. Recall the window size increases by one in each RTT.

### Problem 47

Let W denote max window size. Let S denote the buffer size. For simplicity, suppose TCP sender sends data packets in a round by round fashion, with each round corresponding to a RTT. If the window size reaches W, then a loss occurs. Then the sender will cut its congestion window size by half, and waits for the ACKs for W/2 outstanding packets before it starts sending data segments again. In order to make sure the link always busying sending data, we need to let the link busy sending data in the period *W*/(2\**C*) (this is the time interval where the sender is waiting for the ACKs for the W/2 outstanding packets). Thus, S/C must be no less than *W*/(2\**C*), that is, *S*>=*W*/2.

Let Tp denote the one-way propagation delay between the sender and the receiver.

When the window size reaches the minimum W/2 and the buffer is empty, we need to make sure the link is also busy sending data. Thus, we must have W/2/(2Tp)>=C, thus, *W*/2>=*C*\*2*T*p.

Thus, *S*>=*C*\*2*T*p.

### Problem 48

1. Let W denote the max window size. Then, W\*MSS/RTT = 10Gbps, as packets will be dropped if maximum sending rate reaches link capacity. Thus, we have W\*1500\*8/0.15=10\*10^9, then W= 125000 segments.
2. As congestion window size varies from W/2 to W, then the average window size is 0.75W=93750 segments. Average throughput is 93750\*1500\*8/0.1=7.5Gbps.
3. 93750/2 \*0.15 /60= 117 minutes. In order to speed up the window increase process, we can increase the window size by a much larger value, instead of increasing window size only by one in each RTT. Some protocols are proposed to solve this problem, such as ScalableTCP or HighSpeed TCP.

### Problem 49

As TCP’s average throughput B is given by, so we know that,

*L*= (1.22\**MSS* / (B\**RTT*) ) 2

Since between two consecutive packet losses, there are 1/*L* packets sent by the TCP sender, thus, *T*=(1/*L*)\**MSS*/*B*. Thus, we find that *T*=*B*\**RTT*2/(1.222\**MSS*), that is, T is a function of B.

### Problem 50

1. The key difference between C1 and C2 is that C1’s RTT is only half of that of C2. Thus C1 adjusts its window size after 50 msec, but C2 adjusts its window size after 100 msec. Assume that whenever a loss event happens, C1 receives it after 50msec and C2 receives it after 100msec. We further have the following simplified model of TCP. After each RTT, a connection determines if it should increase window size or not. For C1, we compute the average total sending rate in the link in the previous 50 msec. If that rate exceeds the link capacity, then we assume that C1 detects loss and reduces its window size. But for C2, we compute the average total sending rate in the link in the previous 100msec. If that rate exceeds the link capacity, then we assume that C2 detects loss and reduces its window size. Note that it is possible that the average sending rate in last 50msec is higher than the link capacity, but the average sending rate in last 100msec is smaller than or equal to the link capacity, then in this case, we assume that C1 will experience loss event but C2 will not.

The following table describes the evolution of window sizes and sending rates based on the above assumptions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C1 | | C2 | |
| Time (msec) | Window Size  (num. of segments sent in next 50msec) | Average data sending rate (segments per second, =Window/0.05) | Window Size(num. of segments sent in next 100msec) | Average data sending rate (segments per second, =Window/0.1) |
| 0 | 10 | 200 (in [0-50]msec] | 10 | 100 (in [0-50]msec) |
| 50 | 5  (decreases window size as the avg. total sending rate to the link in **last 50msec** is 300= 200+100) | 100 (in [50-100]msec] |  | 100 (in [50-100]msec) |
| 100 | 2  (decreases window size as the avg. total sending rate to the link in **last 50msec** is 200= 100+100) | 40 | 5  (decreases window size as the avg. total sending rate to the link in **last 100msec** is 250= (200+100)/2 +  (100+100)/2) | 50 |
| 150 | 1  (decreases window size as the avg. total sending rate to the link in last 50msec is 90= (40+50) | 20 |  | 50 |
| 200 | 1  (no further decrease, as window size is already 1) | 20 | 2  (decreases window size as the avg. total sending rate to the link in **last 100msec** is 80= (40+20)/2 + (50+50)/2) | 20 |
| 250 | 1  (no further decrease, as window size is already 1) | 20 |  | 20 |
| 300 | 1  (no further decrease, as window size is already 1) | 20 | 1  (decreases window size as the avg. total sending rate to the link in **last 100msec** is 40= (20+20)/2 + (20+20)/2) | 10 |
| 350 | 2 | 40 |  | 10 |
| 400 | 1 | 20 | 1 | 10 |
| 450 | 2 | 40 |  | 10 |
| 500 | 1  (decreases window size as the avg. total sending rate to the  link in last 50msec is 50= (40+10) | 20 | 1 | 10 |
| 550 | 2 | 40 |  | 10 |
| 600 | 1 | 20 | 1 | 10 |
| 650 | 2 | 40 |  | 10 |
| 700 | 1 | 20 | 1 | 10 |
| 750 | 2 | 40 |  | 10 |
| 800 | 1 | 20 | 1 | 10 |
| 850 | 2 | 40 |  | 10 |
| 900 | 1 | 20 | 1 | 10 |
| 950 | 2 | 40 |  | 10 |
| 1000 | 1 | 20 | 1 | 10 |

Based on the above table, we find that after 1000 msec, C1’s and C2’s window sizes are 1 segment each.

1. No. In the long run, C1’s bandwidth share is roughly twice as that of C2’s, because C1 has shorter RTT, only half of that of C2, so C1 can adjust its window size twice as fast as C2. If we look at the above table, we can see a cycle every 200msec, e.g. from 850msec to 1000msec, inclusive. Within a cycle, the sending rate of C1 is (40+20+40+20) = 120, which is thrice as large as the sending of C2 given by (10+10+10+10) = 40.

### Problem 51

1. Similarly as in last problem, we can compute their window sizes over time in the following table. Both C1 and C2 have the same window size 2 after 2200msec.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C1 | | C2 | |
| Time (msec) | Window Size  (num. of segments sent in next 100msec) | Data sending speed (segments per second, =Window/0.1) | Window Size(num. of segments sent in next 100msec) | Data sending speed (segments per second, =Window/0.1) |
| 0 | 15 | 150 (in [0-100]msec] | 10 | 100 (in [0-100]msec) |
| 100 | 7 | 70 | 5 | 50 |
| 200 | 3 | 30 | 2 | 20 |
| 300 | 1 | 10 | 1 | 10 |
| 400 | 2 | 20 | 2 | 20 |
| 500 | 1 | 10 | 1 | 10 |
| 600 | 2 | 20 | 2 | 20 |
| 700 | 1 | 10 | 1 | 10 |
| 800 | 2 | 20 | 2 | 20 |
| 900 | 1 | 10 | 1 | 10 |
| 1000 | 2 | 20 | 2 | 20 |
| 1100 | 1 | 10 | 1 | 10 |
| 1200 | 2 | 20 | 2 | 20 |
| 1300 | 1 | 10 | 1 | 10 |
| 1400 | 2 | 20 | 2 | 20 |
| 1500 | 1 | 10 | 1 | 10 |
| 1600 | 2 | 20 | 2 | 20 |
| 1700 | 1 | 10 | 1 | 10 |
| 1800 | 2 | 20 | 2 | 20 |
| 1900 | 1 | 10 | 1 | 10 |
| 2000 | 2 | 20 | 2 | 20 |
| 2100 | 1 | 10 | 1 | 10 |
| 2200 | 2 | 20 | 2 | 20 |

1. Yes, this is due to the AIMD algorithm of TCP and that both connections have the same RTT.
2. Yes, this can be seen clearly from the above table. Their max window size is 2.
3. No, this synchronization won’t help to improve link utilization, as these two connections act as a single connection oscillating between min and max window size. Thus, the link is not fully utilized (recall we assume this link has no buffer). One possible way to break the synchronization is to add a finite buffer to the link and randomly drop packets in the buffer before buffer overflow. This will cause different connections cut their window sizes at different times. There are many AQM (Active Queue Management) techniques to do that, such as RED (Random Early Detect), PI (Proportional and Integral AQM), AVQ (Adaptive Virtual Queue), and REM (Random Exponential Marking), etc.

### Problem 52

Note that W represents the maximum window size.

First we can find the total number of segments sent out during the interval when TCP changes its window size from W/2 up to and include W. This is given by:

*S*= *W*/*2* + (*W*/*2*)\*(*1*+*α*) + (*W/2*)\*(*1+α)*2 + (*W/2*)\*(*1+α*)3 + … + (*W/2*)\*(*1+α*)k

We find k=log(1+α)2, then *S*=*W*\*(*2α+1*)/(*2α*).

Loss rate L is given by:

*L*= *1/S* = (*2α*) / (*W*\*(*2α+1*) ).

The time that TCP takes to increase its window size from W/2 to W is given by:

k\**RTT*= (log(1+α)2) \* *RTT*,

which is clearly independent of TCP’s average throughput.

Note, TCP’s average throughput is given by:

*B*=*MSS* \* *S*/((*k+1*)\**RTT*) = *MSS* / (*L*\*(*k+1*)\**RTT*).

Note that this is different from TCP which has average throughput:  , where the square root of L appears in the denominator.

### Problem 53

Let’s assume 1500-byte packets and a 100 ms round-trip time. From the TCP throughput equation , we have

10 Gbps = 1.22 \* (1500\*8 bits) / (.1 sec \* srqt(L)), or

sqrt(L) = 14640 bits / (10^9 bits) = 0.00001464, or

L = 2.14 \* 10^(-10)

### Problem 54

An advantage of using the earlier values of cwnd and ssthresh at t2 is that TCP would not have to go through slow start and congestion avoidance to ramp up to the throughput value obtained at t1. A disadvantage of using these values is that they may be no longer accurate. In particular, if the path has become more congested between t1 and t2, the sender will send a large window’s worth of segments into an already (more) congested path.

### Problem 55

1. The server will send its response to Y.
2. The server can be certain that the client is indeed at Y. If it were at some other address spoofing Y, the SYNACK would have been sent to the address Y, and the TCP in that host would not send the TCP ACK segment back. Even if the attacker were to send an appropriately timed TCP ACK segment, it would not know the correct server sequence number (since the server uses random initial sequence numbers.)

### Problem 56

1. Referring to the figure below, we see that the total delay is