Final Exam — Dec. 18, 2007

EE122: Introduction to Communication Networks

Fall 2007

SOLUTIONS

Prof. Paxson
Department of Electrical Engineering and Computer Sciences
College of Engineering
University of California, Berkeley

Problem	1	2	3	4	5	6	7	8	9	10	11	12	Total
Mean Score	18.4	17.6	11.6	16.4	15.1	13.9	13.7	15.7	15.5	13.7	15.4	18.7	182.8
Max Points	20	20	20	20	20	20	20	20	20	20	20	30	250

High score: 240
90th percentile: 220
75th percentile: 208
Median score: 193.5
25th percentile: 172
Low score: 89

1. **Alphabet soup.**

   For each of the concepts given below, list which of the following acronyms apply:

   802.11, ACK, AES, AIMD, ARP, BGP, CA, CTS, CWND, DDoS, DES, DHCP, DoS, ECN, FIN, ICMP, MACA, MD5, MSS, NAK, NAT, P2P, PMTU, QoS, RED, RSA, RST, RTS, SACK, SHA-1, SSL, SSTHRESH, SYN, TCP, TLS

   Not all acronyms are used. If there is more than one acronym that applies to a given concept, you only need to list one of them. (1 point per item)

   (a) A mechanism by which a router can mark a packet in response to congestion without having to drop the packet: **ECN**

(b) The size of the largest IP packet that can be sent end-to-end without requiring fragmentation: **PMTU**.

It's important to understand the distinction between PMTU and MSS. MSS concerns the largest-sized data segment a TCP sends. It often *relates* to the PMTU (being a bit smaller to account for IP/TCP headers), but this problem asks about IP packets in general.

(c) The entity in a Public Key Infrastructure that signs certificates of identity: **CA**.

Some answers listed TLS or SSL here instead. Those are protocols that *use* PKI certs, but they themselves are not entities that sign such certs; rather, the entities are Certificate Authorities.

(d) An IEEE standard for wireless networking: **802.11**

(e) An in-line network device that rewrites IP addresses (and often transport ports): **NAT**.

Some answers listed DHCP instead. DHCP *allocates* IP addresses, but does not itself rewrite them. In addition, it does not do any allocation or rewriting of transport ports.

(f) The global routing protocol used in the Internet between Autonomous Systems: **BGP**

(g) The TCP congestion-control state variable that holds the value that determines when a sender transitions from Slow Start to Congestion Avoidance: **SSTHRESH**.

Some answers instead gave **CWND**. That's not as precise of an answer, since CWND is used for things other than just determining the Slow Start / Congestion Avoidance transition. However, I allowed it, since CWND is a key state variable used in determining when the transition occurs.

(h) The largest amount of data that TCP will send in a single packet: **MSS**.

Some answers instead gave **CWND**. However, the problem specifically asks "*in a single packet*".

(i) A form of Internet attack in which the attacker tries to make it impossible for the victim to use the network or a service provided over the network: **DoS, DDoS**

(j) A scheme that manages queues in a router by proactively dropping packets rather than only doing so when the queue completely fills up: **RED**

(k) A TCP control flag used to terminate a connection: **RST, FIN**

(l) A protocol that provides secure communication layered on top of TCP: **SSL, TLS**

(m) A protocol used to find the MAC address associated with a given IP address: **ARP**

(n) A control message used in some wireless networks to implement Collision Avoidance: **RTS, CTS**.

Some answers instead gave **MACA**. MACA is a Collision Avoidance *protocol* (indeed, that's what the "**CA**" stands for in the MACA acronym). However, the problem asks for a *control message*.

(o) A form of congestion control that makes increases the sending rate in a linear fashion in the absence of congestion, and divides the rate in the presence of congestion: **AIMD**.

Some answers gave **TCP**. While TCP uses a form of AIMD for its congestion control, TCP is much more than just a congestion control algorithm.

(p) The notion of attaining a degree of performance from the network that is better than "Best Effort": **QoS**.

Some answers instead gave **RED** or **ECN**, both of which are *mechanisms* for modifying the packet-drop process. Applying them to Best Effort traffic does not make the traffic better-than-Best-Effort, it just gives a way to improve the queue management for the Best Effort traffic.

Some answers instead gave **TCP**. However, TCP is a protocol that attempts to provide a widely useful, well-performing protocol *using* an underlying Best Effort delivery service, rather than changing the basic underlying service.

(q) A protocol used to communicate network status and control information, such as packets discarded due to expired Time-to-Live or unreachable destinations: **ICMP**

(r) A widely used cryptographic hash function: **MD5, SHA-1**

(s) A widely used symmetric-key encryption algorithm: **DES, AES**

(t) A widely used public-key encryption algorithm: **RSA**

2. **Networking Terminology.**

For each of the concepts given below, list the term that best applies:

Ack-Splitting	Input-queued router	Revocation
Admission Control	IntServ	Selective Acknowledgment
Autonomous System	Link State Routing	Sequence Number
Certificate	Middlebox	Session Key
Congestion Avoidance	Negative Acknowledgment	Slow Start
Count-to-Infinity	Network Neutrality	Symmetric Key Cryptography
Distance Vector Routing	Non-repudiation	TCP Throughput Equation
Distributed Hash Table	Nonce	Three-Way Handshake
Exponential Backoff	Output-queued router	Token Bucket
Fast Retransmission	PMTU Discovery	Tunneling
Head-of-Line Blocking	Poisoned Reverse	Window Scaling
Hidden Terminal	Random Early Detection	Worm

Not all terms are used. (1 point per item)

(a) A form of Internet QoS that can provide guarantees to individual flows: **IntServ**

(b) A service provided by some peer-to-peer systems for storing content: **Distributed Hash Table**

(c) A notion from cryptography that is supported by the use of public-key digital signatures: **Non-repudiation** or **Certificate**.

Some answers instead gave **Session Key**. However, the question asks about public-key *signatures*, which do not relate to session keys.

(d) A technique of encoding one protocol inside another, which can be used to construct overlay networks or to evade policy restrictions imposed by firewalls: **Tunneling**

(e) The notion that ISPs should provide network service that does not discriminate against certain types of traffic: **Network neutrality**

(f) The mechanism used by TCP to open up the congestion window if the connection is not operating in Congestion Avoidance: **Slow Start**

(g) The problem in wireless networking of a sender not being able to detect if its transmissions collide with those of another nodes because the sender cannot itself hear the other node's transmissions, even though the receiver can: **Hidden Terminal**

(h) The concept of being able to cancel a PKI certificate if the corresponding private key has been stolen: **Revocation**

(i) A malicious network process that repeatedly replicates and self-propagates: **Worm**

(j) A value, usually randomly generated, put in a message in order to make it difficult for an attacker to guess or replay the message contents, or to enable a recipient to demonstrate that they could successfully read the message: **Nonce**

A number of answers instead gave **Session Key**. While such keys are randomly generated, their role differs, and in particularly they do not *by themselves* prevent replay of messages or enable a recipient to demonstrate they could successfully read a message.

(k) The decision made by the network regarding whether to allow a request for QoS: **Admission Control**

(l) A TCP option that overcomes performance limits that come about due to the size of the field used for TCP's flow control: **Window Scaling**

(m) A class of encryption algorithms that require use of a single, secret key: **Symmetric Key Cryptography**

A number of answers instead gave **Session Key**. Session keys are used in symmetric key cryptography, but do not reflect the notion of a *class* of encryption algorithms.

(n) Part of the Karn-Partridge algorithm for managing TCP's retransmission timeout: **Exponential Backoff**

(o) A technique a TCP receiver can use in order to spur the TCP sender to transmit at a rate faster than allowed by congestion control: **Ack-Splitting**

(p) A problem that can occur in input-buffered routers that can cause some packets to unnecessarily wait: **Head-of-Line Blocking**

(q) A mechanism used by TCP to reliably establish a connection:

(r) A routing algorithm based on adjacent routers repeatedly exchanging information about what destinations they know how to reach and with what cost: **Distance Vector Routing**

(s) A routing algorithm based on each router having complete information about the network topology: **Link State Routing**

(t) A network element that alters the semantics of traffic passing through it, often in a transparent fashion: **Middlebox**

3. **Network System Design.**

Consider designing a peer-to-peer system for distributing files.

Note: many students found this problem difficult.

(a) What problem arises if you assume that many of the peers will reside on home machines behind NATs? (4 pts)

Answer: the most important problem is that hosts behind NATs cannot receive incoming connections (that is, they cannot act as servers) without manual configuration of the NAT, because the NAT will not have an address/port mapping for incoming traffic to those hosts. In a peer-to-peer system, hosts act as both clients and servers, so all of the hosts behind NATs cannot carry out their full role.

Other answers listed the difficulty of having *multiple* peers behind a NAT; disambiguating messages coming from a single public address; or NATs timing out state associated with established connections. These (and the subsequent discussions of design modifications to accommodate them, and the performance implications of these changes) received partial credit based on the significance of the problem addressed, the degree to which the design modification ameliorated the problem, and the thoroughness with which the performance implications were assessed. However, most such answers did not come close to receiving full credit, since the problems the dealt with were modest compared to the main one of peers not being able to carry out their server roles.

One common error was to equate NAT functionality with *firewall* functionality. It is important to understand that the two are quite distinct (even though in practice they might be combined into a single hardware device). Specifically, NATs do not block particular ports.

(b) Describe a way by which a peer-to-peer design can overcome this problem. (8 pts)

Answer: one solution to the inability of peers behind NATs to act as servers is to change the system so that end-system peers connect to intermediary nodes (which have public addresses) and relay their requests/responses through those nodes.

Another approach, which is discussed in the textbook, is to use an intermediary just to serve as a point of *rendezvous*. That is, NAT'd peers connect to the intermediary, thus establishing an address/port mapping in the NAT, and the intermediary then sends that information to other peers that wish to contact the first peer.

In fact, this second approach doesn't always work (despite the text stating that it's the standard means for "NAT traversal"), because some NATs do not use a simple mapping of a single address/port pair, but instead map full connection 4-tuples (i.e., taking into account both the local public address/port pair *and* the remote one). However, we didn't go into this difficulty in lecture, so solutions based on this approach received full credit.

Some answers had designs with "super-peers" located *behind* the NAT, in order to disseminate requests among multiple hosts all located behind the same NAT. Such schemes only address the issue of disambiguating multiple peers behind a NAT, but not the problem of external access to such servers in the first place.

Answers that stated one could use an intermediary, but without describing how the intermediary would be used, received partial credit.

Designs that required significant networking changes, such as new header bits, new router actions, or abandonment of NAT entirely (e.g., by mandating IPv6) received limited points, since such broad assumptions have very constrained applicability.

Likewise, designs that required end users to configured their NATs to allow particular ports through have significantly less utility than designs that work without requiring users to take actions beyond the expertise of most of them.

(c) What are the performance implications of incorporating this mechanism into the design? (8 pts)

Answer: for the first solution given above, there are three key performance concerns. First, the public intermediaries can serve as bottlenecks if many peers use them and/or they have limited bandwidth. Second, if all messages are relayed then they incur increased latency and consume increased network resources. Third, the

intermediary introduces an additional point of failure.

For the second solution given above, the performance concerns are the increased connection setup time, the additional point of failure, and the need to maintain the NAT mapping to ensure that the NAT doesn't time it out and render it stale.

(Note that the problem asks for the performance implications, i.e., not just one performance issue but mention of each significant one.)

4. **Queueing.**

You arrive at the airport and need to clear security. You observe that there are two lines, one after the other: one for identification checks followed by one for X-ray of carry-on baggage. You do some quick measurements and observe:

(a) On average, 5 people arrive at the first line every minute

(b) There are on average 30 people in line for the identification check

(c) After people get their identification checked, they immediately go into the X-ray line

(d) There are on average 10 people in line for the baggage X-ray.

How long would you estimate your total wait in line will be? (10 pts)
Show your work. (10 pts)

Answer: this problem can be solved by an application of Little's Theorem, which gives the relationship between N, the number of customers in a queueing system, d, the mean delay experienced by a customer, and λ, the arrival rates of customers as:

$$N = \lambda \cdot d$$

A key point is the third item above, that once people clear the identification check, they then immediately go into the X-ray line. This allows us to consider both lines as a single queue, in which case we have $\lambda = 5/min$, $N = 10 + 30 = 40$, and therefore:

$$d = \frac{N}{\lambda} = \frac{40}{5/\,\mathrm{min}} = 8\,\mathrm{min}$$

Alternatively, we could solve it as two separate systems:

$$\begin{aligned} d_1 = \frac{N_1}{\lambda_1} &= \frac{30}{5/\,\mathrm{min}} = 6\,\mathrm{min} \\ d_2 = \frac{N_2}{\lambda_2} &= \frac{10}{5/\,\mathrm{min}} = 2\,\mathrm{min} \end{aligned}$$

which then gives $d = d_1 + d_2 = 8\,\mathrm{min}$. (Here, we again use the coupling between the two queues, in this case to observe that $\lambda_2 = \lambda_1$.)

While the intent was for the problem to be solved using Little's Theorem, solutions that did so using other forms of reasoning received full credit if correct and the work involved was adequately shown.

5. **Quality of Service.**

   (a) Suppose the capacity C of a link is 21. Assume that 5 sources— S_1, S_2, S_3, S_4, and S_5— are trying to send over the link at rates of $R_1 = 20$, $R_2 = 2$, $R_3 = 3$, $R_4 = 5$, and $R_5 = 11$, respectively. What is the max-min fairness allocation? (8 pts)

   Answer: When we begin, we have $N = 5$ sources and an available capacity of $C = 21$. The fair share is $C/N = 4.2$. Therefore we can satisfy S_2 and S_3 completely, with allocations of 2 and 3 respectively.

   We then remove them, giving us $N \leftarrow N - 2 = 3$, $C \leftarrow C - (2+3) = 16$. The new fair share is $16/3 = 5\frac{1}{3}$. We can satisfy S_4 completely with an allocation of 5.

   We remove S_4, giving us $N \leftarrow N - 1 = 2$, $C \leftarrow C - 5 = 11$. The new fair share is $11/2 = 5\frac{1}{2}$. We cannot satisfy either S_1 nor S_5, so each gets the remaining fair share of $5\frac{1}{2}$.

   Some common mistakes were: (1) rounding down the final shares to 5 rather than $5\frac{1}{2}$, (2) an arithmetic error, (3) not indicating the actual allocations but only the capacity remaining at the end (i.e., $C = 11$), (4) splitting the final fair share 6:5 between S_1 and S_5, or (5) not iterating the process after first satisfying S_2 and S_3.

   (b) Circle one of the following best suited for providing quality of service to individual connections: (2 pts)

   IntServ.

   Neither RED nor DiffServ tracks individual connections, so they aren't appropriate choices.

   (c) Circle one of the following that is best suited for enabling an ISP to provide a form of quality of service without requiring per-flow state in its routers: (2 pts)

   DiffServ.

   IntServ requires per-flow state, and RED does not provide QoS (it instead provides queue management).

   (d) Circle one of the following that best provides a means for ISPs to help keep the router queues for Best Effort traffic from growing to large: (2 pts)

   RED with ECN.

IntServ and DiffServ are both forms of QoS that provide service *beyond* Best Effort, so are immediately inapplicable.

(e) If a router uses only input buffering, discuss a performance problem that can arise. (3 pts)

Answer: One problem that arises is *Head-of-Line Blocking*, in which the packet at the front of one of the input interface queues can't be transferred to its output interface because a packet from another input interface is being transferred there instead. This causes other packets in the queue behind the first that are destined for *other*, free output interfaces to be unnecessarily delayed waiting behind the blocked first packet.

A related problem concerns *contention*: how the router determines which input queues get to use the cross-connect to transfer to a given output interface.

Full credit was given for describing either problem. Simply mentioning Head-of-Line Blocking without defining it was worth partial credit.

Some answers simply noted that if queues fill up, packets get dropped; or that for more advanced types of queueing, it can be difficult to determine which packets to queue and which to discard. These problems however are not particular to routers that only use input buffering.

(f) If a router uses only output buffering, discuss a performance problem that can arise. (3 pts)

Answer: if multiple input interfaces need to transfer packets to the same output interface at the same time, then the interconnect bandwidth to the output queue needs to be N times larger than the bandwidth of the links, where N is the number of possible simultaneous transfers.

Many answers expressed this as $R = N \cdot C$, using the notation from the lecture slides. To receive full credit, the answer needed to explain the meaning of N (which most did).

Again, some answers were in terms of queues filling up and packets being dropped, or difficulties in determining how to prioritize packets. These however are not particular to routers that only use output buffering.

6. **TCP Throughput.**

For each of the following circumstances, state what the TCP Throughput equation would qualitatively give for the expected throughput (for example, "throughput would go to zero"), and explain the corresponding intuition behind why TCP's congestion control would yield that behavior (5 pts each):

(a) The loss rate p becomes vanishingly small ($\rightarrow 0$)

Answer: throughput will go to infinity. If no packets are lost, then the connection will stay in the Additive Increase part of AIMD, linearly increasing the window size, and thus the throughput, without bound.

Some answers stated that TCP would remain in Slow Start. While this is true of an operational TCP implementation, the TCP Throughput equation captures TCP's *AIMD* behavior, and the question was asked in the context of the equation.

Some answers only discussed the throughput gains that come about due to not having to resend data or wait for timeouts. These are much more modest than the fundamental gain due to staying in the Additive Increase phase, and thus only received partial credit.

For this and the other parts of the problem, simply stating what would happen to throughput without explanation was worth 2 pts. A correct explanation was required to earn the full 5 pts for each part.

(b) The RTT becomes vanishingly small ($\rightarrow 0$)

Answer: throughput will go to infinity. As RTT decreases, CWND will open more quickly, allowing more data per RTT.

Many answers were in terms of the window sliding more quickly, which is true, but is not what is captured by the TCP Throughput equation, so these received partial credit.

Some answers were in terms of the effects on RTO. However, the TCP Throughput equation concerns AIMD behavior—it does *not* include the effects of timeout.

(c) The RTT becomes huge ($\rightarrow \infty$)

Answer: throughput will go to zero. As RTT increases, CWND opens more slowly.

Again, answers in terms of the effects on RTO miss that the TCP Throughput equation concerns AIMD behavior and not the effects of timeout.

In addition, in abstract terms RTO should adapt to the larger RTT, and hence not timeout unnecessarily.

Some answers stated that packets would never arrive. That's true when RTT = ∞, but not as it approaches ∞, and misses the central point about how AIMD behaves with respect to the equation's variables.

(d) The MSS becomes huge ($\rightarrow \infty$)

Answer: throughput will go to infinity. As the amount of data sent in packets increases, we need fewer packets to transmit a given amount of data, and therefore

fewer RTTs, since TCP's AIMD congestion control is in terms of full-sized packets rather than in terms of bytes.

Some answers stated that as packets become larger, they are more likely to be lost. There's some (minor) truth to this, but that effect is *opposite* the effect as given by the equation.

Some answers stated that larger packets would take more time to transmit, and thus throughput would go down. While it's true that individual packets would take longer to transmit, it's also true that there would be fewer of them; and since TCP congestion control is managed in units of MSS, it remains the case that increasing MSS will increase throughput. Furthermore, for a given amount of data the best one can possibly do is send it in a single packet, and therefore the fact that individual packets take longer to transmit does not alter that the time required to transmit a given amount of data decreases.

Some answers stated that MSS does not appear in the equation, and therefore has no effect. However, it is the same as the B term (size of packets in bytes).

7. **Wireless.**

   (a) A big problem in wireless is signals attenuating as they propagate through the physical environment. One solution for this would be to boost the strength of signal sent by the transmitter. State two problems with doing so. (8 pts)

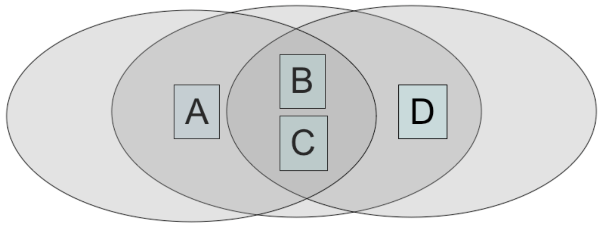
   Answer: (1) Doing so would require a greater expenditure of power, and many wireless nodes have limited power available to them. (2) Doing so would cause the signal to propagate further, thereby increasing interference with other wireless nodes.

   Some answers stated that self-interference due to multipath reflections would increase, or more generally that the signal-to-noise ratio would increase. This isn't clear, however. While there would be more reflections, the base signal would also be stronger, so there might not be any additional self-interference. Similarly, while noise might increase, signal also is certainly increasing, and therefore the ratio might not change.

   Some answers stated that the signal would be more broadly available to malicious nodes. While true, this is a minor consideration compared to the other two effects. For partial credit, the answer needed to frame this in terms of specifics (such as potentially compromising privacy).

   Note that increasing interference is essentially the same as creating more hidden or exposed terminals, so listing both did not suffice as stating "two problems."

(b) Consider wireless nodes communicating as indicated in the diagram below. That is, B and C can hear A, but D cannot; B and C can hear D, but A cannot; and every node can hear B and C.



Suppose that A and B are using MACA (Multiple Access with Collision Avoidance), and A is sending a large data item to B requiring many packets (but B does not need to send back acknowledgments). C wishes to transmit data to D (again, D does not need to send back acknowledgments).

i. To what degree can C improve its performance by "cheating" and ignoring the CTS messages exchanged between A and B. (Here "ignoring" means not taking the action that C is supposed to take when it hears the CTS messages.) Explain why. (3 pts)

Answer: the key insight here is that D cannot hear messages sent by A, and therefore C can freely send to D. Even though locally at C there will be collisions with the messages sent by A, D will (successfully) receive only C's messages. Therefore C can significantly improve its performance by transmitting all the time.

Some answers observed that there would be collisions during the transmission by B of a CTS itself, since D could hear that. This is correct, but pointing this out was not required for full credit.

There were many other possible answers based on different assumptions about how the different nodes react and what sort of messages overlap. For example, some answers stated that A would not transmit to B at all, because at A, B's CTS would collide with C's transmission to D, so A would never successfully receive the CTS. Other answers assumed that C and D would also use MACA, and therefore since D hears B's CTS, D would not itself send a CTS to C at that time, so C would still have to wait a while. Answers such as these that were internally consistent and matched a reasonable interpretation of how the messages might interact received full credit.

ii. To what degree can C improve its performance by "cheating" and ignoring the RTS messages exchanged between A and B. Explain why. (3 pts)

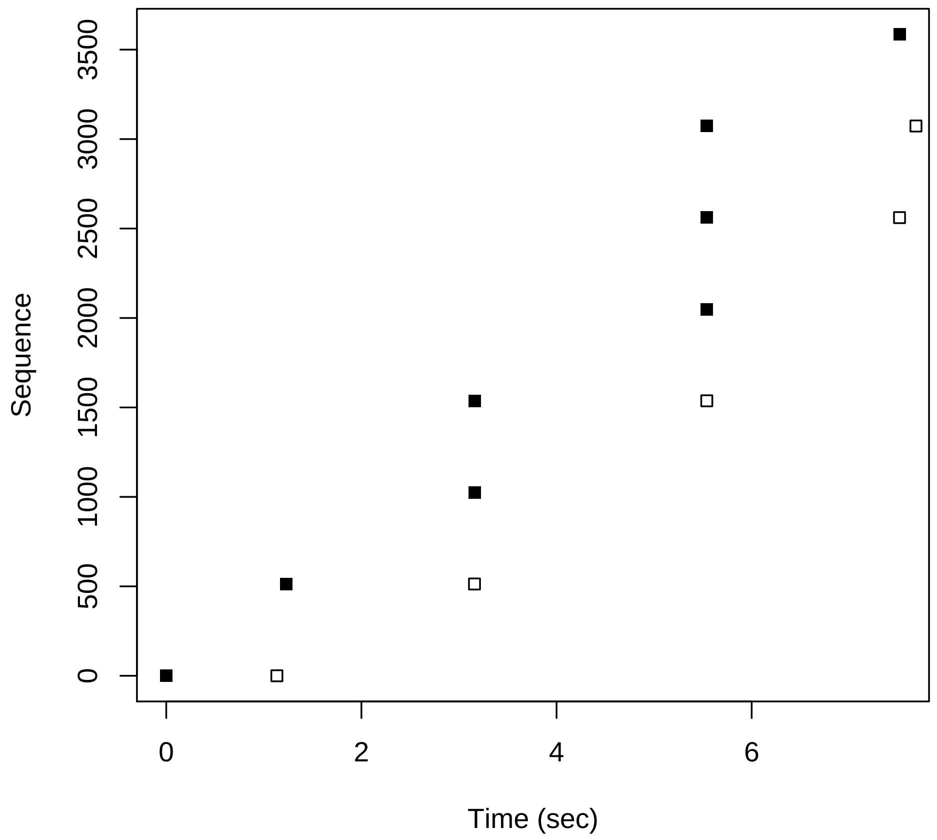
Answer: again, D can hear C whenever C chooses to transmit (except during the brief periods when B sends CTS's), so C can transmit essentially continuously.
Some answers assumed that C still would honor CTS's it heard from B. This was a valid alternative interpretation.

(c) Suppose now that the data transfers are instead from B to A, and that D wishes to transmit data to C.

i. To what degree can D improve its performance by "cheating" and ignoring the CTS messages exchanged between A and B. Explain why. (3 pts)
Answer: these CTS messages are transmitted by A rather than B, and hence D cannot hear them. Thus, there's nothing to ignore, and D does not have an opportunity to improve its performance.
In addition, there's a more fundamental problem, which is that whenever B sends to A, C hears it, and hence D cannot fruitfully transmit to C at those times due to collisions. Some answers considered actions by D (such as sending out its own RTS or CTS) that could cause B to not transmit to A, and hence would avoid these collisions. Such answers if worked out in a consistent fashion received full credit.
Some answers missed the stipulation that now B transmits to A rather than the other way around, and thus assumed that D *would* hear the CTS messages. These answers received partial credit if consistently worked out.

ii. To what degree can D improve its performance by "cheating" and ignoring the RTS messages exchanged between A and B. Explain why. (3 pts)
Answer: the same problem arises as above, namely that C hears transmission from B to A, and therefore D's transmissions will collide with these. Thus, D cannot improve its performance.
Again, many variants were possible, assuming that D's actions could affect B's decisions of when to transmit, and these received full credit when consistently worked out (including some that argued that performance would decrease due to more collisions!).

8. **Time-Sequence Plots**.



Consider the time-sequence plot shown above, which (as usual) is shown from the perspective of the data sender.

(a) What approximately is the MSS used by the sender? (2 pts)

Answer: the MSS can be read off from the sequence number of the first data packet sent, or the difference in sequence numbers between subsequent data packets. The exact value in this trace is 512 bytes, and a reasonable approximate value is 500 bytes.

Those who got this wrong confused MSS with window size.

(b) What is the approximate RTT for the first exchange of the connection establishment handshake? (2 pts)

Answer: the units along the X-axis are seconds, so the RTT for the first exchange is somewhere between 1.0 and 1.5 seconds.

(c) What is the approximate RTT for data packets once the connection is established? (2 pts)

Answer: the RTTs for subsequent exchanges vary between about 1.5 seconds and a bit over 2.0 seconds. Full credit was allowed for answers in the range [1.5, 2.5] providing that the value was larger than that given for the first exchange, which is clearly smaller.

Note: the RTT increases because this Internet path has a very low bottleneck bandwidth. Hence, because data packets are larger than SYN packets, they take significantly longer to transmit over the bottleneck, and thus have higher RTTs.

(d) What, in **number of packets**, is the value of CWND just before $T = 6$ sec? (2 pts)

Answer: 3 packets. This can be seen by the fact that, at that point, three data packets (solid squares) are in flight and not yet acknowledged.

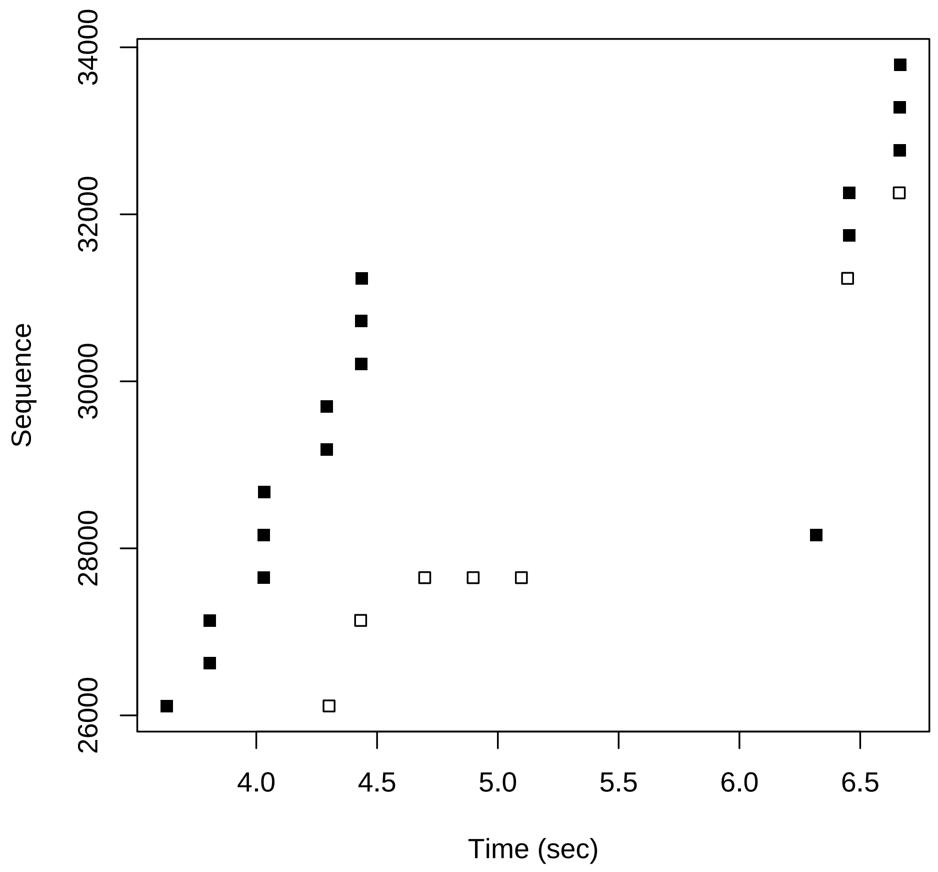
A common error was to state that the window is 4 packets. Presumably these were due to treating the ACK (hollow square) as a data packet.

(e) Does the receiver use delayed acknowledgments? Support your answer by specifying an element of the plot that reflects your conclusion. (2 pts)

Answer: yes, it does. This can be seen by the frequent arrival of ACKs that acknowledge two packets rather than one (e.g., the ACK for data sequence $\approx$ 1500); as well as by the ACKs that acknowledge a single packet often having somewhat larger RTTs.

Some answers incorrectly stated that delayed acknowledgments were contra-indicated by the presence of ack-every-other. Rather, the two go hand-in-hand: the point of the delay is to await the possible arrival of additional packets. Ack-every-other means that such a delay must end, and an ACK then sent, any time a second full-sized packet arrives.

Some answers only cited the longer RTTs for some of the ACKs as indicating the presence of delayed ACKs. While delayed ACKs will generally have longer RTTs, this is not definitive evidence, since queueing delays in the network can vary, causing some ACKs to take longer than others.



(f) For the above time-sequence plot (again recorded at the sender), what, in MSS-sized packets, is the value of CWND after the arrival of the ACK that comes just before $T = 4.5$ sec? (2 pts)

Answer: 8 packets. The corresponding ACK is for sequence number $\approx$ 27,000. Shortly after its arrival, 8 packets are in flight (not yet acknowledged), ranging from 27,500 (which was sent earlier, around time $T = 4.0$ sec, along with two more packets) to 31,500 (sent as the last of 3 data packets shortly after the arrival of the ACK).

A number of answers gave instead 7 packets. Perhaps these were based on the window effective after the arrival of the ACK for $\approx$ 26,000, which does indeed correspond to a window of 7 packets.

(g) The plot on the previous page reflects a retransmitted packet:

i. What, approximately, is the highest sequence number of the retransmitted packet? (2 pts)
Answer: the packet is retransmitted at about time $T = 6.2$ sec, as indicated by the same data sequence number reappearing (it originally appeared at around time $T = 4.0$ sec). This packet has a sequence number of a bit over 28,000.
Quite a few answers were off by an order of magnitude, listing values near 2,800. This presumably reflects errors in reading the plot.

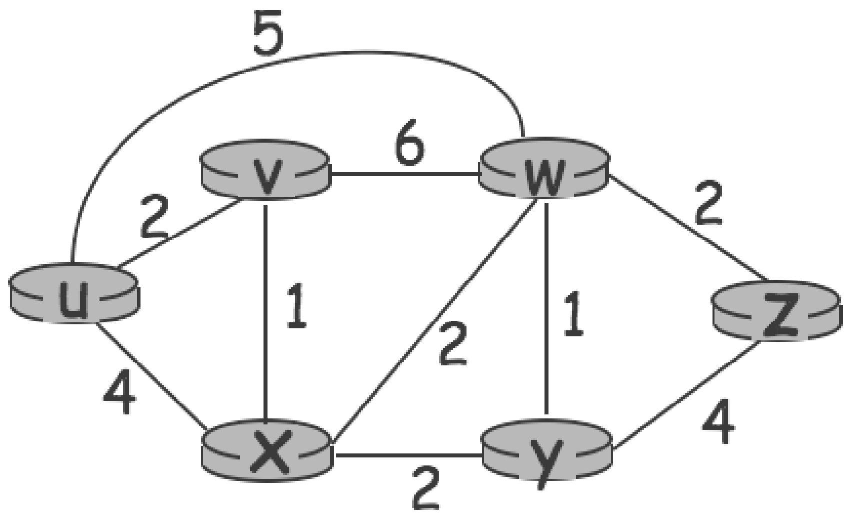
ii. Was the packet retransmitted via Timeout or via Fast Retransmission? (2 pts)
Answer: It was retransmitted via Timeout. There are two ways to tell. First, only *two* duplicate ACKs arrive. (Note, there are three ACKs in total for the preceding segment, the first is *not* a "duplicate"—rather, the others are duplicates of it!) Second, a considerable delay (more than 1 sec) elapses between the arrival of the last dup and the retransmission.
Answers that stated there were 3 dups but also observed that the lengthy delay after them indicated a Timeout received full credit, since it's an easy mistake to read three ACKs in a row as reflecting 3 dups rather than 2 dups; but valuable to then realize it wasn't Fast Retransmission due to the subsequent delay prior to retransmitting.

iii. Was only that one packet lost, or others also from its flight? How can you tell? (2 pts)
Answer: only one packet was lost, which is indicated by the fact that when shortly after it is transmitted an ACK arrives, the cumulative acknowledgment covers all of the data packets that had been sent after the retransmitted packet; so these must have made it to the receiver in the original flight.

iv. Once the retransmitted packet is acknowledged, what in **number of packets** is the value of CWND? (2 pts)
Answer: the window is 2 packets. This can be seen since shortly after the arrival at time $T = 6.3$ sec of the ACK in response to the retransmitted packet, two new data packets are sent, and they are the only ones in flight at that point, indicating the window is 2 packets.

9. **Routing**.

(a) For the topology below, the number on top of each link represents the cost of the link:



(E.g., the weight of the link between V and W is 6.) Node U runs Dijkstra's algorithm to compute the shortest paths to all other nodes in the network. The following table captures the algorithm state at each step. Fill in the values for each step of the algorithm. You can break any ties however you wish. (10 pts)

One solution is:

Step	S	D(V),p(V)	D(W),p(W)	D(X),p(X)	D(Y),p(Y)	D(Z),p(Z)
0 (Initialization)	U	2,U	5,U	4,U	∞	∞
1	U,V			3,V		
2	U,V,X				5,X	
3	U,V,X,W					7,W
4	U,V,X,W,Y					
5	U,V,X,W,Y,Z					

Another solution, based on resolving the tie differently:

Step	S	D(V),p(V)	D(W),p(W)	D(X),p(X)	D(Y),p(Y)	D(Z),p(Z)
0 (Initialization)	U	2,U	5,U	4,U	∞	∞
1	U,V			3,V		
2	U,V,X				5,X	
3	U,V,X,Y					9,Y
4	U,V,X,Y,W					7,W
5	U,V,X,Y,W,Z					

The notation used in the above table is:

S : Set of nodes whose least cost path is definitively known

D(v) : Current value of cost of path from source to v

p(v) : Predecessor node along path from source to v, that is next to v

(b) If the network instead used a distance-vector routing algorithm, what path would be found from node U to node Y? (2 pts)

Answer: both distance-vector and link-state routing algorithms find shortest paths, and there is only one shortest path from U to Y, which is $U \rightarrow V \rightarrow X \rightarrow Y$.

(c) Suppose nodes in the topology are AS's, and AS U wants to avoid going through AS X.

i. What information does BGP provide that allows U to do so? (2 pts)
Answer: BGP is a *path-vector* protocol, meaning that routes carry with them not just their cost but also the AS's traversed by the route. This allows U to inspect different routes and reject any that would go through X.
Some answers stated that BGP supports policy decisions, but the point of this question was *how* does it do so.

ii. What path would BGP find from U to Y, given this constraint? (2 pts)
Answer: the shortest route that avoids X is $U \rightarrow W \rightarrow Y$.

(d) Rank BGP, Link-State and Distance-Vector in order of least scalable to most, and briefly explain why. (4 pts)

i. Link-State is the least scalable, because it requires *global knowledge* of the network topology (and link costs), which it attains by *flooding*. In addition, Dijkstra's algorithm has a running time that is superlinear with the number of nodes in the network.

ii. Distance-Vector is more scalable because it does not require global knowledge. Nodes exchange information only with their neighbors. They do this iteratively, but still not reflecting the same total volume as needed for flooding.

iii. BGP provides routing among AS's rather than individual routers. Since these are much coarser-grained entities, there (1) are not (nearly) as many of them in the topology, and (2) changes inside an AS do not require updates to propagate external to the AS.

Some answers considered the problem simply in terms of how each scheme would perform on the same-sized network (so, no benefit to BGP due to its operating on more aggregated information). This received nearly-full credit if correct otherwise, but does miss the key notion that BGP attains scalability by hiding certain types of change.

Some answers framed BGP as less scalable than Distance-Vector since the routing information includes more state (in particular, the path vector rather than the distance vector). This is a valid consideration, but a minor one compared to the gains for BGP from aggregating information and avoiding loops and count-to-infinity.

Some answers were along the lines of "The Internet primarily uses BGP, so it clearly scales." This claim misses the key point that the Internet necessarily uses *both* BGP and other (Distance-Vector and Link-State) algorithms.

Some answers used the term "*flooding*" for the iterative exchange of information between neighbors in BGP and Distance-Vector. That's not a correct use of the term, which only applies to disseminating data in a *global* or *broadcast* fashion.

Some answers were confused about the meaning of "*scaling*." It always refers to how well some system or approach continues to work as it is applied to increasingly larger problem domains.

10. **Securing communication**.

One of the main protocols that secures ecommerce in the Internet today is HTTPS. It uses both symmetric (secret key) and asymmetric (public key) cryptography.

What would be the implications for using HTTPS if:

(a) You could only use symmetric cryptography. (10 pts)

Answer: it becomes much harder to securely distribute cryptography keys. Ecommerce sites can't have a single key to secure their HTTPS communication, since the end systems will know it once they interact with the site, and therefore can compromise any future communication anyone else conducts with the site.

Because a single key will not suffice, sites will require a separate key for each user. This clearly introduces major scaling problems.

Alternatively, a trusted-third-party scheme such as the KDC design in one of the homework problems could be used. (This is how the *Kerberos* system, in fairly common use today, works.)

Answers received partial credit if they simply stated that certificates couldn't be used, or that attackers could eavesdrop on key exchanges, or that servers would need to store keys but without mentioning scalability issues.

(b) You could only use asymmetric cryptography. (10 pts)

Answer: a fundamental problem with asymmetric cryptography is that it is *much slower* than symmetric cryptography.

Some answers stated that we could no longer have session keys and thus attackers would have more opportunities to observe encryptions made using the public keys and hence possibly to break the key. However, one could still have session keys, they would just themselves be newly generated public/private key pairs, which would first be exchanged using the main public/private keys, in a fashion directly analogous to how session keys are exchanged in HTTPS already.

Some answers discussed the difficulty of requiring clients to have public keys. They actually wouldn't necessarily need to. A client could generate a fresh public/private key pair and send the public half to the server when initially contacting it.

Some answers discussed the inability to use cryptographic hashes for providing non-repudiation. This however misses that use of such hashes is a *performance optimization*, allowing the signer to sign a small data item (the hash) rather than a large one (the full document).

11. **Attacks and Defenses**.

Consider the observation that in a buffer overflow attack on a Windows machine, the attacker must transmit data across the network that includes executable x86 instructions. Your roommate proposes starting a company to build a hardware firewall device that will inspect individual packets and block any that include valid x86 instructions.

(a) To what degree would such a device work for preventing buffer overflow attacks? (8 pts)

Answer: such a device can block all Windows buffer overflow attacks for which the executable instructions are not split across multiple packets. However, because x86 instructions are (in general) multi-byte, an attacker could split a single instruction across multiple TCP segments or multiple packets, in which case inspecting individual packets would miss it.

Another limitation is that the attacker can *still* overflow the buffer, they just can't then inject executable code on the stack. However, they *can* inject invalid instructions on the stack and branch to them, which will cause the server to crash, resulting in a Denial-of-Service. Or they might locate a code sequence already present in the program that does what they want (for example, an invocation of the Unix `system()` library call that runs a shell command) and branch to that.

Some answers stated that the approach would not work for encrypted traffic. This is true, but is not as fundamental of a limitation as those above, since many services that might be attacked do not encrypt their traffic.

Some answers stated that the approach would not work for *tunneled* traffic. While this is true (depending on how the tunneling is encoded), tunneling requires that *both* ends of a transfer agree on the encoding. For an attack that has not yet succeeded, this isn't possible, since the victim has no means (or reason) to understand or agree to the tunneled encoding.

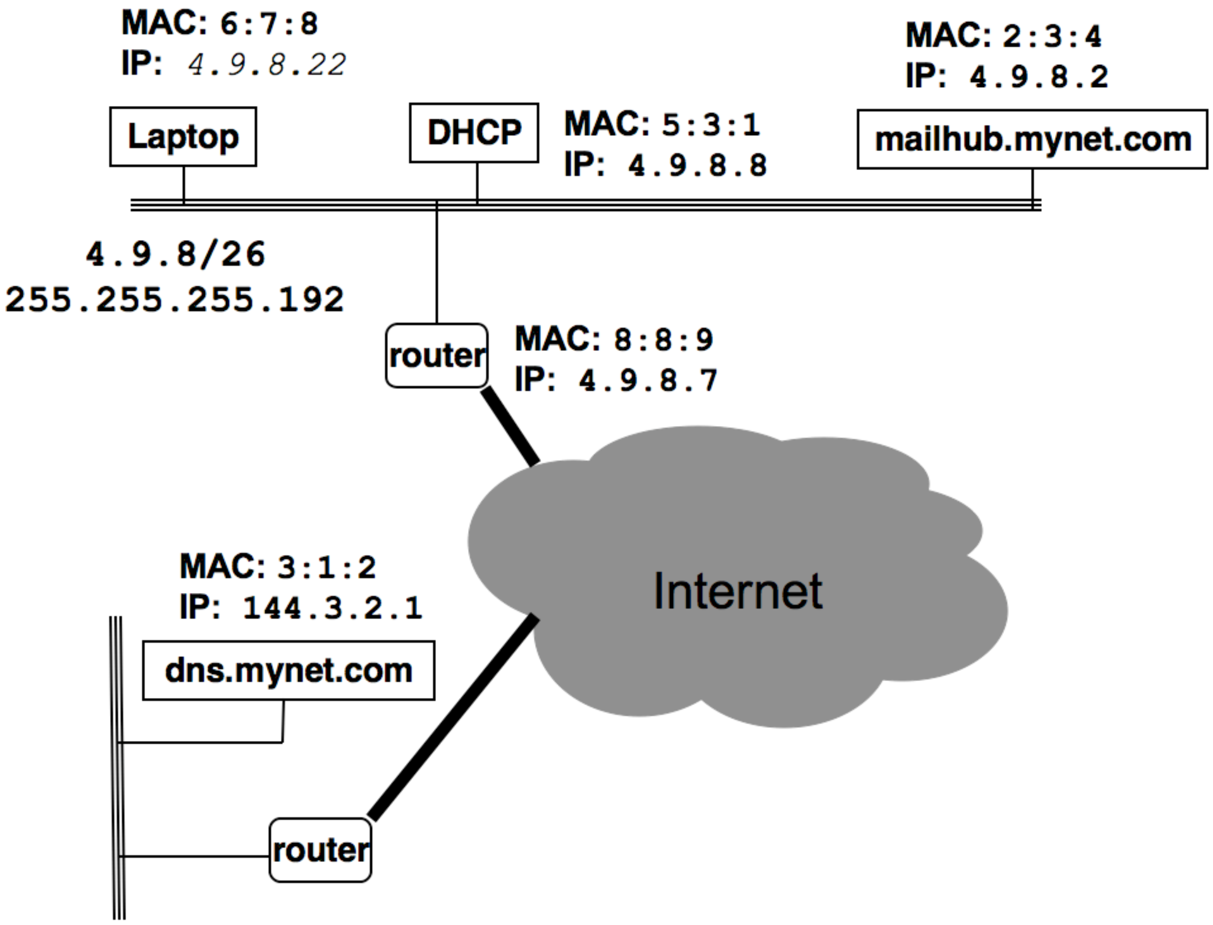
(b) Name two drawbacks with your roommate's proposed approach. (12 pts)

There are a number of drawbacks. A very large on is the problem of *false positives*; that is, blocking traffic that matches but does not reflect a buffer overflow attack. This can come in two forms. The first is the presence of executable x86 instructions due to a legitimate transfer of a binary (such as downloading a program). The second is the presence of *bit patterns* that *happen* to represent legitimate instructions, but are being interpreted in a different context and do not in fact constitute any code that will be executed.

Another drawback concerns the need to perform *stateful* processing in order to deal with the problems of instructions split across fragments or TCP segments, as discussed above.

A third problem concerns the extra processing required, since now the firewall must inspect entire packets rather than just packet headers. Actually, numerous techniques have been developed for efficiently implementing such matching in hardware; however, we did not discuss these, and the concern about performance is in general a reasonable one, so listing this received full credit as a drawback.

12. **Putting It All Together**.



In the above figure, you connect your laptop via Ethernet to a local area network. Hosts on the LAN have IP addresses out of the 4.9.8/26 block, with a corresponding netmask of 255.255.255.192. For each host, the diagram shows its MAC address and its IP address. (We use shortened MAC addresses of just 3 digits to make them easier to write down.) Your laptop's MAC address is `6:7:8`. It initially does *not* have an IP address, though once it acquires one, it will be 4.9.8.22.

On your laptop, your mail client wishes to transfer an email message to the server at `mailhub.mynet.com`. For the purposes of this problem, we approximate the actual sending of an email message as a one-way transfer from your laptop (acting as a client) to the mail server associated with `mailhub.mynet.com`.

Each half of this problem is worth **15 points**.

(a) Describe *all* of the messages (IP packets, and link-layer Ethernet frames for messages that are not transmitted using IP) sent and received by your laptop up until the point at which your laptop has fully established a TCP connection to the mail server. List the messages in order of transmission (number the messages if you wish).

For each message, give the relevant source and destination addresses (IP and/or MAC), the highest-layer protocol present in the packet (e.g., DNS rather than UDP or IP or Ethernet), any control flags used by the transport protocol (such as SYN or ACK for TCP), and the meaning of the message (e.g., for an HTTP response this might be "status code 200 - OK - and headers").

You only need to describe the messages as they are transmitted or received by the laptop. (For example, you don't need to describe a transmitted packet again as it travels through one of the routers.)

Assume:

- No packets are lost.
- Your laptop's DNS cache is empty.
- Your laptop's ARP cache is empty.
- Your laptop initially does not have an IP address.
- For any other host your laptop communicates with, its caches are already populated with any necessary information. (For example, any request you make to the DNS server can be answered without the server making any further requests.)
- DNS requests are made using UDP.

Here is the set of packets generated:

```
# First, laptop configures using DHCP to get an IP address, the
# address of a DNS server, the address of its local router, and
# its local network/netmask so it can tell which IP addresses
# are directly connected.

1. laptop -> broadcast
    src MAC 6:7:8, IP none ; dst MAC ff:ff:ff (broadcast), IP none
    DHCP discover
2. DHCP -> laptop
    src MAC 5:3:1, IP 4.9.8.88 ; dst MAC 6:7:8, IP none
    DHCP offer
     includes client IP address 4.9.8.22
      DNS server IP address 144.3.2.1
      router IP address 4.9.8.7,
      local network/netmask 4.9.8/26, 255.255.255.192
3. laptop -> broadcast (same)
```

```
    DHCP request (or "accept")
4. DHCP -> laptop (same)
    DHCP ack

# laptop is now going to determine the IP address associated
# with mailhub.mynet.com.  To do so, it needs to contact its DNS
# server.  Because it determines that the DNS server is not on the
# local network, it needs to send its request through the router
# that DHCP configured it to use.  However, it only has the
# IP address of the router, so it needs to first ARP to determine
# the MAC address to use to contact it.

5. laptop -> broadcast (same)
    ARP who has 4.9.8.7
6. router -> laptop
    src MAC 8:8:9, IP none ; dst MAC 6:7:8, IP none
    ARP 4.9.8.7's MAC address is 8:8:9

# Now laptop can send an IP packet to the DNS server at 144.3.2.1.
# but with a destination *MAC* address of 8:8:9, since the packet needs to
# be forwarded by its local router.  Similarly, the replies will have
# a source *MAC* address of 8:8:9, since locally they come from the
# router.
#
# Note, strictly speaking laptop should first look up the MX record
# associated with mailhub.mynet.com, and then the A record for the
# name returned in the MX reply.  It was okay to skip that step, however,
# and we leave it out below.

7. laptop -> dns.mynet.com (via router)
    src MAC 6:7:8, IP 4.9.8.22 ; dst MAC 8:8:9, IP 144.3.2.1
    UDP payload: DNS lookup, A record for mailhub.mynet.com
8. dns.mynet.com (via router) -> laptop
    src MAC 8:8:9, IP 144.3.2.1 ; dst MAC 6:7:8, IP 4.9.8.22
    UDP payload: DNS reply, mailhub.mynet.com's A record is 4.9.8.2

# laptop wants to connect to mail server at 4.9.8.22.  Due to the netmask,
# it knows that this host is on the local layer-2 network.  However, it only
# knows the server's IP address, not its MAC address, so it again gets
# the latter via ARP.

9. laptop -> broadcast (same)
```

```
    ARP who has 4.9.8.2
10. mailhub -> laptop
     src MAC 2:3:4 IP none ; dst MAC 6:7:8, IP none
     ARP 4.9.8.2's MAC address is 2:3:4

# Now, laptop is ready to establish a TCP connection to 4.9.8.2.

11. laptop -> mailhub
     src MAC 6:7:8, IP 4.9.8.22 ; dst MAC 2:3:4, IP 4.9.8.2
     TCP SYN to port 25
12. mailhub -> laptop
     src MAC 2:3:4, IP 4.9.8.2 ; dst MAC 6:7:8, IP 4.9.8.22
     TCP SYN ACK
13. laptop -> mailhub (same)
     TCP ACK of SYN ACK, connection now established
```

Common difficulties:

- DHCP configuration information does *not* include MAC addresses, just IP addresses, so you need to then use ARP to resolve those to MAC addresses for any hosts local to the LAN (in particular, for this problem this means the mail server and the router).
- DHCP configuration *does* include the address of the DNS server, and *does not* include the address of the mail server. Thus, you need to use DNS to look up the mail server's address.
- ARP is a link-level protocol and as such does *not* use IP addresses.
- A DHCP "Request" (also termed "Accept") sent by the laptop to the DHCP server is *broadcast*, so that any other DHCP servers that replied to the initial "Discover" can see that they have not been selected.
- With DHCP, the "Offer" first sent back by the server contains the configuration information, not the later "Ack" sent in response to a Request/Accept.
- `dns.mynet.com`'s MAC address shouldn't show up in any of the packets, unless you included not only packets directly sent/received by the laptop but also elsewhere in the network. Packets sent by the laptop to the server have a destination *MAC address* of the local router, not of the server; likewise, packets returned from the server to the laptop have this address as their source, from the perspective of the client.
- Similarly, the router will *not* ARP to resolve the MAC address of `dns.mynet.com`, since it's not directly connected to it.
- The router does *not* ARP for the mail server on behalf of the laptop. (In addition, the laptop knows to ARP itself for the mail server since they are

directly connected, which the laptop can tell from the netmask.)

- Packets sent to/from `dns.mynet.com` will never have an IP address corresponding to the router; they will always be that of the laptop and that of `dns.mynet.com`.
- For those who included lookup of an `MX` record, note that what is returned for an `MX` record is a hostname, not an IP address, so a second lookup to resolve the hostname via its `A` record would be required. However, since this step was optional, there was no penalty for answers that assumed the `MX` lookup returned an address.

(b) Once the connection is established, list in order *all* of the packets used to transfer the email from your laptop to the mail server. For each packet, note:

- Whether it is sent by the client (which you can abbreviate C) or the server (S).
- The sequence number it carries. Express sequence numbers in terms of full-sized packets.
- Whether it is a data packet or an acknowledgment.
- For data packets, the value of the sender's CWND.

So for example "`C -> S data packet 5`" would indicate that the client sent to the server the 5th data packet.

If sending the packet was delayed for some reason, note this fact and briefly explain why. If it was lost (see below), also note that fact.

Assume:

- The email message you want to send takes 10 full-sized packets.
- The RTT is on the order of 1 msec.
- The server does not send any data to the client.
- All data packets are full-sized, and you can specify their sequence numbers simply with the number of the packet.
- **The 3rd, 4th and 5th acknowledgments are lost.** (Note, this doesn't necessarily mean that they are the acks for the 3rd-5th data segments; rather, it means that of the acks for data segments transmitted by the receiver, the 3rd, 4th and 5th are lost.)
- No other packets are lost.
- The TCPs initialize CWND to its usual starting value.
- The TCP endpoints both advertise windows of 8 packets.
- The client initiates shutdown of the connection after transferring the message, and the server immediately agrees to close the connection.
- The TCPs use the usual form of delayed acknowledgments.

- The TCPs initialized SSTHRESH to 10 packets.

Here is the set of packets generated:

```
 1. C -> S data packet 1, CWND=1
 2. S -> C delayed ack for 1  (seq # = 2)
 3. C -> S data packet 2, CWND=2
 4. C -> S data packet 3, CWND=2
 5. S -> C ack for 3
 6. C -> S data packet 4, CWND=3
 7. C -> S data packet 5, CWND=3
 8. C -> S data packet 6, CWND=3
 9. S -> C ack for 5  LOST
10. S -> C delayed ack for 6  LOST
11. C -> S data packet 4 timeout retransmission, CWND=1
13. S -> C delayed ack for 6  LOST
14. C -> S data packet 4 timeout retransmission,
           exp. backoff, CWND=1
15. S -> C delayed ack for 6
16. C -> S data packet 7, CWND=2
17. C -> S data packet 8, CWND=2
18. S -> C ack for 8
           # At this point, the window may or may not
           # increase by MSS depending on the
           # implementation of Congestion Avoidance.
19. C -> S data packet 9, CWND=3
20. C -> S data packet 10, CWND=3
21. C -> S FIN
22. S -> C ack for 10
23. S -> C FIN, ack for FIN
           # Above could be combined with 22
24. C -> S ack of FIN
```

Common difficulties:

- Slow-start growth is 1 MSS per ACK. Some solutions had it being 1 MSS per MSS being ack'd (so an ACK that covers two full-sized packets increased CWND by 2 MSS).
- Ack-every-other means that every 2nd segment will generate an ACK; *or* if a single segment is awaiting an ACK, this will occur after the delayed-ACK timer

expires (typically 200 ms). So, for example, a flight of three data packets 6,7,8 will generate two ACKs, one for 7 and one for 8, where the latter will be a delayed-ACK. If the sender is in Slow Start, the receipt of both of these will increase the window by $2 \cdot$ MSS, *not* $3 \cdot$ MSS.

- Acknowledgments are always *cumulative*, and thus the sequence number being acknowledged will never decrease.
- Because no data segments are lost, the ACKs sent once the retransmitted packet gets through will be for the highest data segment sent so far (packet 6 in the above solution) and *not* for the retransmitted segment (packet 4 above).
- RTT $\neq$ RTO! Just because RTT is 1 msec does *not* mean that RTO is 1 msec.
- CWND $\neq$ SSTHRESH! The sender's CWND starts off as $1 \cdot$ MSS as usual, not 8 segments. (Some solutions had it starting as 2 segments on the belief that the ACK in the connection establishment handshake would increase it. This might be true for the server, which is the recipient of the final handshake ACK, but not of the client.)
- A number of solutions left out either indications of delays (including delayed ACKs), sequence numbers, or CWND.