Solutions to Final Exam

EE122: Introduction to Communication Networks

Fall 2006

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

1. **Router messages and feedback.** (Total: 20 points)

   (a) When routers generate ICMP messages, to where do they send them? Along with the ICMP header at the beginning, what additional contextual information do routers include in the messages? (5 pts)

   Answer: the messages are sent to the source address specified in the IP header. For context, the ICMP message includes the IP header of the packet that triggered the ICMP, along with at least 8 bytes of that packet's payload, which is enough to include the ports used in the transport header if the packet was carrying UDP or TCP.
   Common difficulties:

   - Some answers limited the discussion of information in the ICMP message to the error code. However, the problem asks for what *contextual* information the messages include *in addition* to the ICMP header (which is where the error code is given).
   - Other answers stated that the ICMP includes "the first 8+ bytes of the original packet." This misses that its the IP header of the original packet *plus* 8+ bytes of its payload. (-1 point)

   (b) Are ICMP messages delivered reliably? If so, briefly explain the mechanism. If not, give a reason why not. (5 pts)

   Answer: they are *not* reliably delivered. To deliver them would require that the router maintain state for every ICMP message it delivers, which would be very expensive.

Common difficulties:

- There was some confusion over the fact that some routers will not generate certain types of ICMP messages, or will rate-limit their generation. This is true, but such routers will still *forward* ICMP messages sent by other routers (as far as the forwarding router is concerned, it's just another IP packet). Also, the term "reliably delivered" presumes that a decision is made to send data in the first place.

(c) Name a circumstance under which an end-host (not a router) will *send* an ICMP message. (5 pts)

Answer: Three possibilities are

- When an end-host sends an ICMP *ping* ("Echo Request") message.
- When an end-host receives a *ping* and sends back an ICMP "Echo Reply" message.
- When a packet arrives at an end-host destined for a UDP port for which no socket is associated (no server is listening on that port). This causes the host to generate an ICMP "port unreachable" in response.
  Note that the same will not occur if the packet is destined for a *TCP* port that lacks a receiving socket. However, the principle is similar, so this answer received nearly full credit.

Common difficulties:

- Several answers included endhosts sending an ICMP Source Quench to tell other hosts transmitting to them to slow down. Endhosts do not generate Source Quenches; only routers do.
  However, this point was not made clear in lecture, so this answer was allowed full credit.
- Other answers mentioned that a packet might arrive at the endhost that is too large for its NIC, which would cause the endhost to send back a "Needs Fragmentation" ICMP. In actuality, this should not happen: the endhost should always be able to process a full-sized packet that is sent *to* it; "Needs Fragmentation" is generated by routers when they need to *forward* a packet onto another link (beyond the one upon which it arrived) for which the packet's size exceeds the MTU.
  However, again this point was not made clear in lecture, so the answer was allowed full credit.
- Another answer was that when executing `traceroute`, the endhost *receiving* a `traceroute` probe (i.e., the target of the `traceroute` measurement) would send a "TTL Expired" ICMP when one of the probes finally

reaches it. In practice, endhosts do not check TTLs on incoming packets because they are not *forwarding* packets; rather, they send back to the `traceroute` host an ICMP "Port Unreachable" (because the usual form of `traceroute` sends UDP packets to high-numbered ports for which it's unlikely a server is listening).
However, again this was not clear from lecture, and so was allowed full credit.

- Other answers mentioned ICMP "redirects." However, these are only generated by routers, to inform endhosts to use a different router to more efficiently reach a given destination.
- If an endhost receives a corrupted packet, it does *not* send an ICMP message in response, for two reasons. First, no such ICMP message has been standardized. Second, when a packet arrives that's corrupted, it's not certain even which host sent it (perhaps the source IP address is part of what got corrupted; recall that the IP checksum only covers the IP header).
- Path MTU discovery does not *send* ICMP messages; it works by receiving them ("Needs Fragmentation").
- Similarly, `traceroute` does not send ICMP messages; it works by receiving them ("TTL Expired").

(d) Briefly describe how the `traceroute` tool works (i.e., what does it do in order to identify the routers that make up an Internet path).

Answer: `traceroute` sends a series of packets for which it sets the IP "Time To Live" (TTL) hop-count field to different values. Setting the TTL to N will result in the Nth router generating a "Time Exceeded" ICMP message which it sends back to the originating host. Because ICMP messages are sent using IP packets, `traceroute` can extract from their source address the IP address associated with the router at the Nth hop. (Strictly speaking it's not *the* IP address associated with the router, since routers have one IP address for each of their interfaces.)

Common difficulties:

- There was some confusion over the IP "Record Route" option as being one way to determine the route a packet takes. While the option does provide a (not very good, due to size limits) mechanism for doing so, it is quite distinct from the `traceroute` tool, and it's important to understand how in particular `traceroute` works, since it's what is widely used in practice.

2. **Attacks.** (Total: 20 points)

Suppose we could deploy a mechanism that would ensure IP source addresses correspond to the actual sender of a packet (i.e., it's impossible to "spoof" source addresses). For each of the following threats, explain whether (and briefly why) the mechanism would: *(i)* completely eliminate the threat, *(ii)* eliminate some instances of the threat, but not all of them, or *(iii)* have no impact on the threat. (Each is worth 4 pts.)

(a) Buffer overflow attacks

Answer: a buffer overflow attack works by sending an over-long message to an endpoint (usually a server) that has not allocated enough buffer space to store the message. As such, the attack does not rely on disguising the attacker's source address. Furthermore, for TCP traffic, the source address *cannot* be spoofed, since that will prevent the attacker from establishing the connection in the first place.

Acceptable answers for this problem were either "it has no impact on the threat" or "it only reduces the threat for UDP (and other non-TCP) attacks that can be launched using spoofed source addresses." An FYI regarding this latter possibility: some notable worms ("Slammer" and "Witty") exploited UDP-based services; however, they did not in fact bother to spoof their source addresses when doing so.

Common difficulties:

- Some answers confused whether the IP packet itself was somehow being overflowed versus the buffer memory inside an end-host.

(b) TCP SYN flooding

Answer: a SYN flood works by sending a large number of SYN packets to a server in order to tie up all of its available state, preventing it from accepting any new connections from legitimate clients. This attack is especially effective when the attacker can use spoofed source addresses in their SYN packets, as this makes it more difficult for the victim to install filters to remove the attack traffic. However, the attack still works if source addresses are not spoofed, provided that the attacker has enough zombies available to make it impractical for the victim to install filters for all of them. Related to this, we looked at the "SYN cookie" defense, which prevents the attack from succeeding *if* the attacker uses spoofed source addresses; but we also discussed how in that case the attacker can still try to launch the attack using legitimate source addresses.

Thus, the answer is "reduces the threat."

(c) TCP "ack splitting" to open up the congestion window quickly

Answer: ack-splitting only works for established TCP connections. As these generally can't be made using spoofed source addresses, the answer is "has no impact on the threat."

Common difficulties:

- Some answers discussed launching this attack as a man-in-the-middle, in order to force the sender to transmit excessively fast. This is not the basic nature of the attack (which instead is how a selfish receiver can improve the performance of its downloads), but was worth partial credit.

(d) Reflector DDOS attacks

Answer: in a Distributed Denial-of-Service attack that uses reflectors, the attacker sends packets to intermediaries for which the purported source address of the packet corresponds to the victim. When the intermediary replies to the incoming packet, it actually sends its reply to the victim. Therefore, eliminating the attacker's ability to spoof source addresses will completely remove this threat.

Actually, that's not the full story. For some protocols, reflector attacks can actually be launched using non-spoofed source addresses. For example, the attacker's zombies send to the intermediaries a DNS lookup request for the victim's domain (using the zombie's actual source address for this request). When doing so, they request "recursion" for the lookup, i.e., that the intermediary do the lookup itself if it doesn't already have the information. This results in the intermediary sending a request to the victim's DNS server. Because of this possibility, an answer of "reduces the threat" was also acceptable *if* you explained how sometimes reflector attacks can be launched using non-spoofed source addresses.

Note: the use of such non-spoofed DNS reflector attacks became widespread enough about a year ago that there has been a large-scale campaign to configure DNS servers to not support recursive queries (other than for specific clients that are meant to be the customers of the server). Today, most no longer do.

Common difficulties:

- Some were confused by the fact that the final traffic sent from the reflector host to the victim does not contain any spoofed addresses. However, to *get* the reflector to send this traffic requires spoofing of source addresses by the slave.

(e) DNS cache poisoning

Answer: one way to poison a DNS cache is to spoof a reply to a request that the victim makes. However, *another* way to poison a cache is to have your DNS server include bogus Additional records when replying to a lookup request that the client makes to your server. Therefore, eliminating spoofed source addresses only "reduces the threat."

Answers that included one of these attacks but not the other were worth half credit.

3. **Securing communication with cryptography.** (Total: 20 points)

(a) Identify two different errors Alice has made in her use of cryptographic methods, and describe how to correct each of these.

Answer: two basic errors Alice has made:

- She encrypted the session key with *her own* public key rather than with Bob's. To decrypt this message to recover the session key requires access to the corresponding private key. Only Alice has the private key that matches her public key, so Bob won't be able to recover the session key.
- If she wants to provide non-repudiation, then she needs to *sign* the digest by encrypting it with her private key. This allows Bob to demonstrate that Alice signed the message by showing that Alice's public key correctly transforms the signature into the SHA-1 digest that matches the message.

Some students spotted a third error, which is that Alice sent the digest unencrypted. If she encrypted it using the session key, then she could prevent the attack discussed below in the next part of the problem.

Common difficulties:

- Several answers stated that Alice needs to encrypt the message with her *private* key rather than her *public* key. That is not a correct fix, because if she does so, than anyone can read the message by decrypting it using Alice's well-known public key. (In particular, Eve can then read the message.)
- Alice's private key might be known to Eve, or the public key encryption might be breakable by factoring large numbers: this is a *risk* Alice takes, but not an *error* Alice makes, because there's no *a priori* reason to expect either of these problems to occur.

- Some answers stated that Alice should send over her certificate. This is not an error she has made because the problem states her key is already "well known." In addition, Eve could send over Alice's certificate too, so such an answer had to explain the additional steps (such as those undertaken by HTTPS) to prevent such impersonation.
- Some answers stated that for Alice to provide non-repudiation, she should encrypt the *session key* with her private key rather than the digest. This doesn't work, as follows:
  i. Eve generates a random number R.
  ii. Eve encrypts R with Alice's public key (which Eve knows) to get K_R.
  iii. Eve then constructs whatever message M' she wishes, encrypts it with K_R, and sends to Bob $E_{K_R}(M')$, R (supposedly, the session key signed by Alice), and the Digest.
  iv. Bob applies Alice's public key to R to recover the session key. He obtains K_R, which indeed decrypts the message correctly.
- Some answers stated that Alice should send a digest of the encrypted message rather than the original. This doesn't provide non-repudiation, however. *Anyone* can compute a SHA-1 hash of a given message; without incorporating encryption using Alice's private key, there's no way to tie a hash back to Alice.
- Similarly, some answers stated that Alice should encrypt the digest with the session key. Again, this doesn't provide non-repudiation.
- Some answers assumed that at this point that Eve could generate hash collisions. To get credit for this required identifying that SHA-1 is today still difficult to break in terms of generating collisions (unlike the assumption in the next part of the problem), and not reusing this attack in the next part of the problem.
- Some answers observed that Alice did not convey to Bob which cryptographic protocols (e.g., AES vs. DES, or SHA-1 vs. MD5 for hashing) she was using. The intent of the problem was that this was well-known beforehand. However, since the potential for the specific protocols to not be clearly identified beforehand is a legitimate problem that can arise when arranging cryptographic exchanges (witness how TLS/SSL explicitly negotiates a suite of crypto protocols when starting up), discussion of this possibility merited partial credit.

(b) Assuming that these errors are corrected, but that Eve figures out how to "break" SHA-1 such that she can generate hash collisions, explain an attack that Eve can then conduct. (8 pts)

Answer: if Eve can see the SHA-1 digest that Alice computes for the message, then (given the assumption that she can break SHA-1) she can generate other messages with the same SHA-1 digest. Call this message M'. She picks a new session key, K', encrypts M' with it, and sends the following to Bob:

- $E_{K'}(M')$ (the new message encrypted with the new key)
- $E_{\mathsf{Pub}_{\mathsf{Bob}}}(K')$ (the session key encrypted using Bob's public key)
- The original SHA-1 digest, which also matches M'
- Alice's signature of the original digest

Bob will have no way of determining that M' is not in fact a message that Alice transmitted, since her signature for its digest does indeed match its digest.

As noted above, some students observed that Alice could send the digest to Bob only in an encrypted form (either using the session key, or using Bob's public key) and not in clear text. If so, then Eve doesn't know for what digest she needs to find a collision. This observation is correct, and worth full credit. (Note though that if Eve can guess what the original message was, then she can still launch the substitution attack.)

Common difficulties:

- Some answers stated that it's enough for Alice to encrypt the digest with her private key to prevent Eve from conducting this attack. However, if that's all that Alice does, then Eve simply applies Alice's public key to the encrypted digest to recover the original digest. She then generates a collision and the attack proceeds as discussed above.
- Some answers stated that breaking the hash allows Eve to see (or modify) the original message. It doesn't, however, because Eve still can't recover the session key. All she can do is generate an additional, fake message that still matches the signature.
  A variant on this was thinking that one could try all the possible messages that have hashes colliding with the original to see which one looks like a plausible message. However, there are infinitely many such collision messages (since the hash takes *any* string and maps it to a fixed-size value).
- Some answers mixed this use of hashing with the use of SHA-1 for encoding SYN "cookies." However, the two are quite different. SYN cookies defend against spoofed TCP SYNs; if the attacker launching a SYN flooding attack using spoofed source addresses can generate collisions, that doesn't help them with spoofing the ACKs for the SYNs (to complete the TCP 3-way handshake), since they don't see the cookie values in the first place.

- Minor error: specifying that Eve generates a false *digest* rather than a false *message*.

4. **QoS.** (Total: 20 points)

   (a) Suppose the capacity C of a link is 18. Assume that 4 sources—S1, S2, S3, and S4—are trying to send over the link at rates of r1=2, r2=4, r3=5, and r4=8, respectively. What is the max-min fairness allocation? (8 pts)

   Answer: in the first round, we have $N = 4$ sources needing allocations, so the fair share is $18/4 = 4.5$. This suffices for sources 1 and 2 (giving them their full requested allocation of 2 and 4, respectively), so we remove them and repeat the process.

   For the second round, $N' = 2$ and we have already given out $2 + 4 = 6$ of the total capacity of 18. Therefore, 12 remains, and the fair share is $12/2 = 6$. This suffices for source 3, so it gets its full requested allocation of 5. This leaves us with a remaining capacity of 7, with $N'' = 1$. The fair share is 7. All remaining sources (just source 4) want more than that, so each is allocated the remaining fair share.

   Thus, the allocations are: A1=2, A2=4, A3=5, A4=7.

   Common difficulties:

   - A number of answers didn't highlight the allocation for each source. When it could be *completely* inferred from the calculations performed, the answers received full credit (since the question wasn't explicit about what to show), but in general it's always good to be clear about the elements that go into a particular answer.
   - Similarly, some answers didn't show any work and just gave the allocation. Since the question wasn't explicit about showing work, these were allowed full credit, but again it's much better to always be clear about how one arrives at an answer.

   (b) For each of the following, annotate it with "IS" if it applies to Integrated Services (IntServ), "DS" if it applies to Differentiated Services (DiffServ), and "BE" if it applies to Best Effort. (A given statement can apply to more than just one type of service.)

      i. The service is provided end-to-end (3 pts):

      Answer: IS and BE. DiffServ operates only between domains and not end-to-end, while IntServ and Best Effort both are provided as end-to-end services.

Common pitfalls: it's easy to not think of Best Effort as providing any actual sort of service, and therefore not considering that it provides a service end-to-end. The scoring was thus 2 points for IS, 1 point for BE.

ii. Among the three, requires the most state in routers (3 pts):

Answer: IS. IntServ requires the most state, since it needs to track individual flows or connections. DiffServ only needs to maintain per-class state, of which there are not many classes. Best Effort doesn't maintain any state.

iii. Is widely available in the Internet today (3 pts):

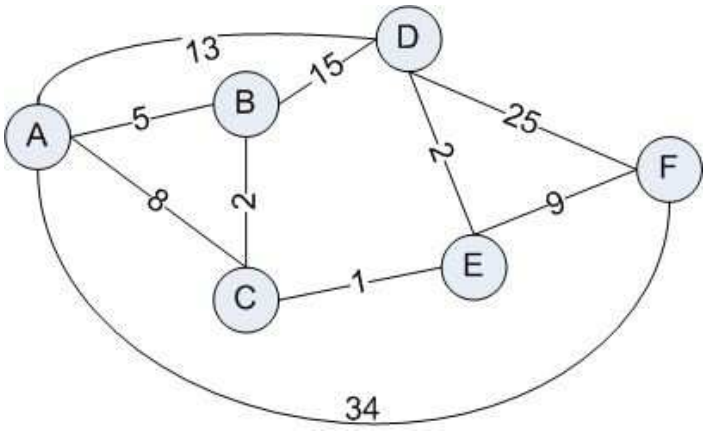
Answer: BE (with DS also being allowed in addition).
Best Effort is the only end-to-end service widely available today. We also discussed how DiffServ is frequently available within individual domains, though usually not between domains. Because the question wasn't clear on just what constitutes "widely available," answers that included DiffServ too were allowed.

iv. Provides isolation and guarantees among aggregated flows but not individual connections (3 pts):

Answer: DS. DiffServ operates on large aggregates. IntServ provides fine-grained isolation and guarantees (which makes it more difficult to deploy, since it requires more state). Best Effort doesn't provide any isolation or guarantees, period.

5. **Routing.** (Total: 20 points)

(a) Consider the following network with nodes A through F:



Step	S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0 (Initialization)	A	5,A	8,A	13,A	∞	34,A
1	AB		7,B			
2	ABC				8,C	
3	ABCE			10,E		17,E
4	ABCED					
5	ABCEDF					

(b) In the above example, all the link costs are positive. Explain why Dijkstra's algorithm does not work if some of the link costs are negative. (5 pts)

Answer: At each step of Dijkstra's algorithm, we choose the node, say X, with the smallest cost path from the source. If all link costs are positive, then any path through any other node will always have a higher cost, so we can be sure that we have found the shortest path to X. This no longer holds if some link costs are negative, however, as the cost of a path can decrease any time. If there exists a cycle with negative cost in the network graph, the shortest cost of a path can be made infinitely small, by repeatedly going around the cycle.

Common difficulties:

- Some answers did not include the previous hops in the table, just the weight.

- Some answers miscomputed some of the previous hops (including marking the previous hop of a node X as being X itself).
- Some answers missed the ABCE (cost 5+2+1) path to E and instead used ACE (cost 8+1) as the shorted to E. (This then led to a similar mistake for the path to D and F.)

(c) Name two problems that would arise if all routing in the Internet was done using Link-State and Dijkstra's algorithm. Does Distance-Vector routing suffer from these? How about Path-Vector? (5 pts)

Answer: here are some problems that would arise:

- Link-State (LS) routing requires flooding of link information to all nodes in the network. For a very large network such as the Internet, this would result in a huge number of messages.
  Distance-Vector (DV) and Path-Vector (PV) do not need to flood information to the same degree.
- Dijkstra's algorithm runs in time $O(N^2)$ for N links. Thus, the computational cost at each node would become prohibitive.
  DV and PV have better computation cost, although each can take a long time to *converge*. DV is prone to forming temporary *routing loops*; PV avoids many of these.
- LS routing exposes an ISP's precise connectivity, which some ISPs prefer to keep private for competitive reasons.
  DV and PV both do not advertise *how* an ISP gets to a given location, so they are better in this regard.
- LS routing does not allow ISPs to express policies regarding what traffic they are willing to carry.
  DV also is not able to express such policies. PV can to a limited degree, by allowing a router to inspect the AS path associated with an update to determine whether the AS's represented in the path fit with the ISP's policy.

Common difficulties:

- Some answers stated only two closely related problems with Link State: for example, that it consumes too much memory, and that it also consumes to much processing.
  Full credit required stating two problems with Link State that were sufficiently different.

6. **TCP mechanisms.** (Total: 20 points) You decide to modify the TCP stack on your desktop so you experience better performance (higher throughput when either sending, receiving, or both). Note that you only get to change your desktop's TCP—you can't change that of the other endpoint with which you'll be communicating.

   (a) Suppose your stack originally supports both timeout-driven retransmission and fast retransmission. Among the following, **circle** which one would gain you the greatest benefit to your performance:

      i. disable timeout retransmissions (instead only retransmit using fast retransmission)
      ii. **disable exponential backoff of timeouts**
      iii. disable fast retransmission (instead only retransmit using timeouts)
      iv. disable RTT estimation / RTO adaptation (use the initial values set for RTT and RTO)

   and **explain** why it would offer an improvement (8 pts):

   Answer: disabling exponential backoff of timeouts will enable your TCP to recover from repeated loss (i.e., loss of retransmitted packets) more quickly. For the others:

   - If you disable timeout retransmissions, then you will *lose* performance any time a lost packet is not followed by enough duplicate acknowledgments to trigger fast retransmission (you will never recover from the loss).
   - If you disable fast retransmission, then you will *lose* performance any time you could have detected a loss quickly by observing 3 duplicate acknowledgments (you will only recover from the loss later, when the retransmission timeout finally expires).
   - If you do not perform RTT estimation and do not adapt RTO based on it, then you're stuck with a fixed value for RTO. This may be too high, in which case you lose performance since you could have recovered from losses more quickly; or too low, in which case you will lose performance by retransmitting unnecessarily.

   Common difficulties:

   - A number of answers confused *collisions*, which occur in protocols like Ethernet when multiple senders try to transmit simultaneously, with *congestion*, which refers to excessive contention for capacity along a network path, but does not have an element of simultaneous transmission to it.
   - A number of answers confused exponential timer backoff with *cutting CWND* for *multiplicative decrease*. When timeouts occur, CWND has already been cut all the way down to a single MSS. The sender's effective

transmission rate is cut still further, however, by its waiting increasingly long amounts of time before transmitting.

- Similarly, disabling exponential backoff will not alter the TCP throughput "sawtooth."
- Some answers confused RTT (estimated round-trip time) with RTO (time to wait before retransmitting). RTO needs to be larger than RTT, or else we risk retransmitting every single packet.

(b) What would happen if everyone's TCP stack in the Internet did this? (6 pts)

Answer: if everyone's TCP failed to use exponential backoff for their retransmission timeouts, then during times of congestion there would be no mechanism in the network to drain the load presented to the network. This in turn could lead to *congestion collapse*.

Note, full credit was allowed here when the answer in the previous part of the question differed from the correct one, but the explanation for this part of the question was consistent with that previous answer.

(c) If you could pick something *else* to modify about your TCP stack so that you experience better performance, what would it be, and why? (6 pts)

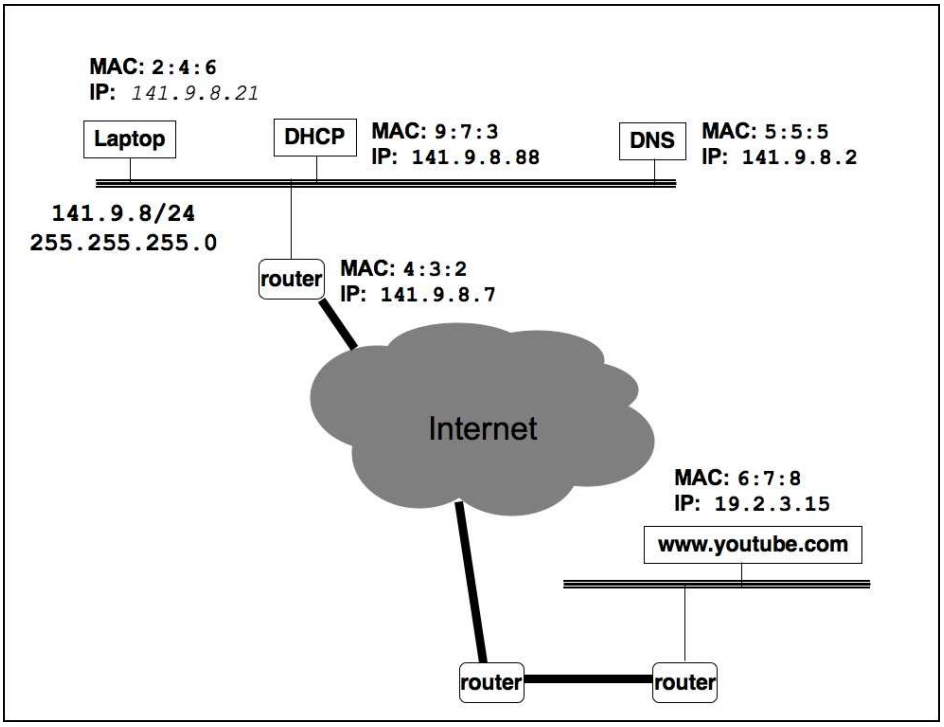
Answer: there are a number of possibilities

- Disable AIMD congestion control. When your TCP suffers a loss, leave the window alone. This allows you to transmit more data, potentially improving performance.
- Skip congestion avoidance. Never leave Slow Start, or always just send the entire window offered by the receiver.
- Change your TCP to send multiple ACKs for each incoming packet (the "ack splitting" attack).
- Increase your send and/or receive buffers (this one doesn't violate congestion control in any fashion; it's really just tuning).

Common difficulties:

- Some people misinterpreted the term "*else*" in this question as meaning they should pick another one of the four options given in the first part of the question. Even though this was clarified on the board during the exam, enough answers had this problem that clearly the problem was ambiguous in this regard, so solid explanations about the effects of one of the other mechanisms were acceptable here for full credit.

- While in general increasing the congestion window or either the receiver's advertised window or the sending window would improve your performance, this is *not* necessarily the case if you modify your TCP to ignore the receivers advertised window. Doing so risks sending data across the network that the receiver must discard because they lack buffer to hold it.



7. **Putting it all together.** (Total: **40 points**)

   Here are 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 2:4:6, IP none ; dst MAC ff:ff:ff (broadcast), IP none
    DHCP discover
2. DHCP -> laptop
    src MAC 9:7:3, IP 141.9.8.88 ; dst MAC 2:4:6, IP none
    DHCP offer
     includes client IP address 141.9.8.21,
      DNS server IP address 141.9.8.2,
      router IP address 141.9.8.7,
      local network/netmask 141.9.8/24, 255.255.255.0
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 www.youtube.com.  To do so, it needs to contact its DNS
# server.  Because it determines that the DNS server is on the
# local network, it needs to use ARP to determine the MAC address
# to use to contact it.

5. laptop -> broadcast (same)
    ARP who has 141.9.8.2
6. DNS -> laptop
    src MAC 5:5:5, IP none ; dst MAC 2:4:6, IP none
    ARP 141.9.8.2's MAC address is 5:5:5
7. laptop -> DNS
    src MAC 2:4:6, IP 141.9.8.21 ; dst MAC 5:5:5, IP 141.9.8.2
    UDP payload: DNS lookup, A record for www.youtube.com
8. DNS -> laptop
    src MAC 5:5:5, IP 141.9.8.2 ; dst MAC 2:4:6, IP 141.9.8.21
    UDP payload: DNS reply, www.youtube.com's A record is 19.2.3.15

# laptop wants to connect to HTTP server at 19.2.3.15.  This is
# not a local address, so it needs to address it via its router.
# However, it only knows the router's IP address, not its MAC address,
# so it gets the latter via ARP.

9. laptop -> broadcast (same)
    ARP who has 141.9.8.7
10. DNS -> laptop
     src MAC 4:3:2, IP none ; dst MAC 2:4:6, IP none
     ARP 141.9.8.7's MAC address is 4:3:2

# Now, laptop is ready to establish a TCP connection to 19.2.3.15.
# To do so, it sends a SYN with a destination IP address of 19.2.3.15,
# but a destination *MAC* address of 4:3:2, since the packet needs to
# be forwarded by its local router.  Similarly, the replies will have
# a source *MAC* address of 4:3:2, since locally they come from the
# router.
```

```
11. laptop -> www.youtube.com
     src MAC 2:4:6, IP 141.9.8.21 ; dst MAC 4:3:2, IP 19.2.3.15
     TCP SYN to port 80
12. www.youtube.com -> laptop
     src MAC 4:3:2, IP 19.2.3.15 ; dst MAC 2:4:6, IP 141.9.8.21
     TCP SYN ACK
13. laptop -> www.youtube.com (same)
     TCP ACK of SYN ACK, connection now established
14. laptop -> www.youtube.com (same)
     TCP data "HTTP GET", CWND = 1
15. www.youtube.com -> laptop
     TCP data "HTTP REPLY 1" + ack of "HTTP GET", CWND = 1
# Note 1: it's possible that www.youtube.com would send a
#         separate ACK for the HTTP GET request before
#         transmitting the reply
#
# Note 2: it's okay if you interpret the ACK of the SYN-ACK
#         as having already opened CWND to 2; some TCP's
#         do in fact behave that way
16. laptop -> www.youtube.com (same)
     TCP ACK of "HTTP REPLY 1"
17. www.youtube.com -> laptop (same)
     TCP data "HTTP REPLY 2", CWND = 2
18. www.youtube.com -> laptop (same)
     TCP data "HTTP REPLY 3", CWND = 2
19. laptop -> www.youtube.com (same)
     TCP ACK of "HTTP REPLY 3"
# Note: the laptop does not generate a separate ACK for
# "HTTP REPLY 2" since it uses ack-every-other.
20. www.youtube.com -> laptop (same)
     TCP data "HTTP REPLY 4", CWND = 3
21. www.youtube.com -> laptop (same)
     TCP FIN, beginning of close handshake
# Note 1: the FIN could have been bundled with the previous
#         reply
# Note 2: the client might initiate the connection termination
#         rather than the server
22. laptop -> www.youtube.com (same)
     TCP FIN + ack of FIN, both sides closed
23. www.youtube.com -> laptop (same)
```

```
TCP ACK of FIN, termination handshake complete
```

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 DNS server and the router).
- Link-level protocols such as DHCP and ARP do *not* have IP addresses in them, since they're framed at a lower layer.
- 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.
- *www.youtube.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 *www.youtube.com*, since it's not directly connected to it. It likewise will not ARP for the DNS server on behalf of the laptop, since the laptop *is* directly connected to the DNS server (which it can tell from the netmask).
- Packets sent to/from *www.youtube.com* will never have an IP address corresponding to the router; it will always be that of the laptop.
- Some solutions had the laptop including a FIN with its HTTP request, or just after it. It is unlikely that an app would be written in a style to do this (it takes an explicit system call to perform a half-close); much more likely that the app (browser) would close the connection only upon receiving the reply. However, since we didn't delve into this distinction in lecture, these solutions received full credit.
- TCP "delayed acknowledgments" can acknowledge a single packet (they do so after a delay, hence the term). Thus, the single packet sent as the HTTP request will indeed be ack'd; there won't be a timeout due to a failure to ack it.

- Slow-start growth is 1 MSS per ACK. Some solutions had it being 1 MSS per MSS being ack'd (so a delayed ack that covers two full-sized packets increased CWND by 2 MSS).
- A number of solutions had the *www.youtube.com* server operating in *congestion avoidance* rather than *slow start*. Connections always start in slow start.

8. (20 pts) What different or alternative packets are generated in the previous problem in the following situations:

   (a) The browser crashes just before the last data packet sent by www.youtube.com arrives. (5 pts)

   Answer: if the browser crashes, then the TCP connection no longer has a socket open on your laptop to receive incoming packets. In this case, when the next TCP packet arrives from *www.youtube.com*, your laptop's TCP stack will reply with a TCP **RST** packet (a packet with the "RST" flag bit set in the header). It will continue to do so for any additional packets the arrive.

   Some solutions had the kernel sending the RST spontaneously upon the browser crashing. In general, this won't occur, but as we didn't cover this point in lecture, and it's a natural expectation, this solution was allowed.

   (b) The URL being fetched is instead https://www.youtube.com. No need to give TCP-level details, just sketch the additional higher-layer interactions. (5 pts)

   Answer: HTTPS uses TLS (or the very similar SSL) protocol on top of TCP. Doing so leads to the following messages being exchanged once your laptop establishes a connection to *www.youtube.com*:

   i. *Laptop* $\rightarrow$ *www.youtube.com*: sends a list of suites of cryptography protocols that the laptop's implementation of SSL/TLS supports.
   ii. *www.youtube.com*$\rightarrow$ *Laptop*: sends the server's selection from the list
   iii. *www.youtube.com*$\rightarrow$ *Laptop*: sends *www.youtube.com*'s certificate
   iv. (*Laptop* validates the certificate by looking for a signature corresponding to one of the CAs configured into your browser)
   v. *Laptop* $\rightarrow$ *www.youtube.com*: sends a newly-generated session key encrypted using the server's public key
   vi. *www.youtube.com*$\rightarrow$ *Laptop*: confirms use of that key
   vii. (subsequent communication between the two endhosts is now all encrypted using the session key)

(c) The router at 141.9.8.7 is also a NAT box, though the subnet continues to use the public address block 141.9.8/24. (5 pts)

Answer: the messages exchanged between your laptop and hosts on the other side of the NAT (only the *www.youtube.com* server, in this example) only contain IP addresses in their IP headers. They don't carry any in their payloads. Therefore, *from the perspective of your laptop* (i.e., the packets it immediately sends and receives), nothing changes. On the other side of the NAT, the Laptop's IP address and the ephemeral port associated with the connection will be remapped.

Some solutions discussed the translation performed by the router. As long as this was correct (mapping of both addresses and ports, and both for incoming and outgoing packets), this earned full credit.

(d) You connect your laptop to the LAN using 802.11 (any version) rather than Ethernet. You need only state which different or additional link-layer messages will be sent, and in what order, for the first packet transmitted by the laptop. (5 pts)

Answer: 802.11 is a wireless link-layer. As such, it cannot use *collision detection* (such as Ethernet's CSMA/CD) but instead must use *collision avoidance* (CSMA/CA). To do so, for each packet your laptop transmits the following steps occur:

i. Your laptop transmits a *Request to Send* (RTS) control message to the local host with which it wishes to communicate.
ii. The local host replies with a *Clear to Send* (CTS) control message.
iii. Your laptop transmits the packet it is trying to send.
iv. The local receiver responds with an *Acknowledgment* (ACK) control message.

Common difficulties:

- Nearly everyone left off the final ACK, but it's a crucial part of the protocol, since the sender needs to hear it in order to know that no collision occurred.
- Some answers focused on associating with base stations. We didn't discuss how this works in lecture, and these answers didn't give "additional link-layer messages" as called for by the problem, so they received limited partial credit.