Homework Assignment #3

Due: Friday Nov. 9th @ 3:50PM

EE122: Introduction to Communication Networks

(Fall 2007)

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

Vern Paxson / Jorge Ortiz / Lisa Fowler / Daniel Killebrew

Turn in as **hardcopy** to the drop box in **240 Cory**.

1. Kurose & Ross, Chapter 3, pp. 301-303:

   (a) P33.

   (a) TCP Slow Start is operating in the interval $[1, 6]$ and $[23, 26]$.

   (b) TCP Congestion Avoidance is operating in the intervals $[6, 16]$ and $[17, 22]$.

   (c) After the 16^{th} transmission round, packet loss is recognized by a triple duplicate ACK. If there was a timeout, the congestion window size would have dropped to 1.

   (d) After the 22^{nd} transmission round, segment loss is detected due to timeout, and hence the congestion window size is set to 1.

   (e) The threshold is initially 32, since it is at this window size that Slow Start stops and Congestion Avoidance begins.

   (f) The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 16, the congestion window size is 42. Hence the threshold is 21 during the 18^{th} transmission round.

   (g) Again, the threshold is set to half the value of the congestion window when packet loss is detected. Here it is detected by timeout. When loss is detected during transmission round 22, the congestion window size is 26. Hence the threshold is set to 13.

   (h) During the first transmission round, packet 1 is sent; packet 2–3 are sent in the second transmission round; packet 4–7 are sent in the third transmission round; packet 8–15 are sent in the fourth transmission round; packet 15–31 are sent in the fifth transmission round; packets 32–63 are sent in the sixth transmission round; packets 64–96 are sent in the seventh transmission round. Thus, packet 70 is sent in the seventh transmission round.

(i) The congestion window and threshold will be set to half the current value of the congestion window (8) when the loss occurred. Thus, the new values of the threshold and window will be 4.

(b) P38.

(a) The loss rate, L, is the ratio of the number of packets lost over the number of packets sent per cycle. In addition, in a cycle exactly 1 packet is lost, so L is simply the inverse of the number of packets sent per cycle, which is:

$$\begin{aligned}
& \frac{W}{2} + \left(\frac{W}{2}+1\right) + \left(\frac{W}{2}+2\right) + \ldots + \left(\frac{W}{2}+\frac{W}{2}\right) \\
= & \sum_{i=0}^{W/2} \left(\frac{W}{2}+i\right) \\
= & \left(\frac{W}{2}+1\right)\frac{W}{2} + \sum_{i=0}^{W/2} i \\
= & \left(\frac{W}{2}+1\right)\frac{W}{2} + \frac{W/2(W/2+1)}{2} \\
= & \frac{W^2}{4} + \frac{W}{2} + \frac{W^2}{8} + \frac{W}{4} \\
= & \frac{3}{8}W^2 + \frac{3}{4}W
\end{aligned}$$

Therefore the loss rate is:

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

(b) For large values of W, $\frac{3}{8}W^2 \gg \frac{3}{4}W$. Therefore $L \approx \frac{8}{3}W^2$ or $W \approx \sqrt{\frac{8}{3L}}$. From the text (page 287), we have an average throughput shown below:

$$\begin{aligned}
\text{average throughput} &= \frac{3}{4}\sqrt{\frac{8}{3L}} \cdot \frac{MSS}{RTT} \\
&= \frac{1.22 \cdot MSS}{RTT \cdot \sqrt{L}}
\end{aligned}$$

2. Kurose & Ross, Chapter 4, pp. 424-426:

(a) P22.

Step	N'	D(s),p(s)	D(t),p(t)	D(u),p(u)	D(v),p(v)	D(w),p(w)	D(y),p(y)	D(z),p(z)
0	x	∞	∞	∞	3,x	6,x	6,x	∞
1	xv	∞	7,v	6,v	3,x	6,x	4,x	∞
2	xvy	∞	7,v	6,v	3,x	6,x	4,x	18,y
3	xvyu	10,u	7,v	6,v	3,x	6,x	4,x	18,y
4	xvyuw	10,u	7,v	6,v	3,x	6,x	4,x	18,y
5	xvyuwt	8,t	7,v	6,v	3,x	6,x	4,x	12,t
6	xvyuwt	8,t	7,v	6,v	3,x	6,x	4,x	12,t
7	xvyuwt	8,t	7,v	6,v	3,x	6,x	4,x	12,t

(b) P23, part (a).

Step	N'	D(s),p(s)	D(t),p(t)	D(u),p(u)	D(v),p(v)	D(w),p(w)	D(y),p(y)	D(z),p(z)
0	st	∞	1,s	4,s	∞	∞	∞	∞
1	st	∞	1,s	3,t	5,t	∞	8,t	6,t
2	stu	∞	1,s	3,t	5,t	6,u	8,t	6,t
3	stuv	8,v	1,s	3,t	5,t	6,u	6,v	6,t
4	stuvy	8,v	1,s	3,t	5,t	6,u	6,v	6,t
5	stuvyz	8,v	1,s	3,t	5,t	6,u	6,v	6,t
6	stuvyzw	8,v	1,s	3,t	5,t	6,u	6,v	6,t
7	stuvyzwx	8,v	1,s	3,t	5,t	6,u	6,v	6,t

(c) P24. The problem states "Consider the distance vector algorithm . . ." This means: compute each step of the distance vector algorithm, i.e., show z's full distance-vector table at each iteration until the algorithm terminates.

The rows are FROM and the columns are TO:

	u	v	x	y	z
v	∞	∞	∞	∞	∞
x	∞	∞	∞	∞	∞
y	∞	∞	∞	∞	∞
z	∞	5	2	10	0

	u	v	x	y	z
v	4	0	∞	7	5
x	12	∞	0	1	2
y	∞	7	1	0	10
z	9	5	2	3	0

	u	v	x	y	z
v	4	0	7	7	5
x	11	7	0	1	2
y	11	7	1	0	3
z	9	5	2	3	0

(d) P27.

The rows are FROM and the columns are TO:

Node x table

	x	y	z
x	0	5	2
y	∞	∞	∞
z	∞	∞	∞

	x	y	z
x	0	5	2
y	5	0	6
z	2	6	0

Node y table

	x	y	z
x	∞	∞	∞
y	5	0	6
z	∞	∞	∞

	x	y	z
x	0	5	2
y	5	0	6
z	2	6	0

Node z table

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	2	6	0

	x	y	z
x	0	5	2
y	5	0	6
z	2	6	0

(e) P29. Information learned directly from other autonomous systems comes via eBGP ('e' = "external"). Information about other autonomous systems propagates to an AS's other border routers via iBGP ('i' = "internal"). Such information would propagate to interior routers via the AS's internal routing protocol, such as RIP or OSPF, but that case does not arise in what's asked for in this question.

(a) eBGP

(b) iBGP

(c) eBGP

(d) iBGP

3. **CSMA/CD** Let A and B be two stations attempting to transmit on an Ethernet. Each has a steady queue of frames ready to send; A's frames will be numbered A_1, A_2, and so on, and B's similarly. Let T= 51.2μs be the exponential backoff base unit. Suppose A and B simultaneously attempt to send frame 1, collide, and happen to choose backoff times of 0 x T and 1 x T, respectively, meaning A wins the race and transmits A_1 while B waits. At the end of this transmission, B will attempt to retransmit B_1 while A will attempt to transmit A_2. These first attempts will collide, but now A backs off for either 0 x T or 1 x T, while B backs off for time equal to one of 0 x T, . . ., 3 x T.

(a) Give the probability that A wins this second backoff race immediately after this first collision; that is A's first choice of backoff time k x 51.2 is less that B's.

Answer. For the second backoff race, A picks $k_A(2)$ to be either 0 or 1 with equal probability, so 1/2 for each. B picks $k_B(2)$ from $(0, 1, 2, 3)$ with probability 1/4 for each choice. A wins the second backoff race if $k_A(2) < k_B(2)$.

$$\begin{aligned} P[A\ wins] &= P[k_A(2) < k_B(2)] \\ &= P[k_A(2) = 0] \times P[k_B(2) > 0] + P[k_A(2) = 1] \times P[k_B(2) > 1] \\ &= \frac{1}{2} \times \frac{3}{4} + \frac{1}{2} \times \frac{2}{4} \\ &= \frac{5}{8} \end{aligned}$$

(b) Suppose A wins this second backoff race. A transmits A_3, and when it is finished, A and B collide again as A tries to transmit A_4 and B tries once more to transmit B_1. Give the probability that A wins this third backoff race immediately after the first collision.

Answer. In this case, again A picks $k_A(3)$ to be either 0 or 1 with probability 1/2 each, while B picks $k_B(3)$ from $(0, 1, 2, 3, 4, 5, 6, 7)$, each with probability 1/8:

$$\begin{aligned} P[A\ wins] &= P[k_A(3) < k_B(3)] \\ &= P[k_A(3) = 0] \times P[k_B(3) > 0] + P[k_A(3) = 1] \times P[k_B(3) > 1] \\ &= \frac{1}{2} \times \frac{7}{8} + \frac{1}{2} \times \frac{6}{8} \\ &= \frac{13}{16} \end{aligned}$$

(c) Give a reasonable lower bound for the probability that A wins all the remaining backoff races.

Answer. We assume that B will retry 16 times (the typical value), after which it gives up. Furthermore, when choosing k between 0 and $2^n - 1$ in the *exponential backoff*, n is capped at 10.

The probability that A wins all 13 remaining backoff races is:

$$P[A\ wins\ remaining\ races] \;=\; \prod_{i=4}^{16} P[A\ wins\ i | A\ wins\ i-1]$$

Let $k_A(i)$ be the k value A picks for ith backoff race. Given that A wins that race, ($k_A(i) < k_B(i)$), the probability of A winning backoff race #$(i+1)$ is 1 if $k_A(i) + 1 < k_B(i)$. (Here we assume that the unit of waiting is equal to the transmission time of the frame. Other interpretations are also possible.)

Otherwise (in case $k_A(i) + 1 \geq k_B(i)$), A and B collide when A is done with the ith frame, and the probability becomes $P[k_A(i+1) < k_B(i+1)]$.

$$\begin{aligned}
P[A\ wins\ i+1 | A\ wins\ i] &= P[k_A(i)+1 < k_B(i)] \cdot 1 \\
&\quad + P[k_A(i)+1 \geq k_B(i)] \cdot P[k_A(i+1) < k_B(i+1)] \\
&\geq P[k_A(i)+1 < k_B(i)] \cdot P[k_A(i+1) < k_B(i+1)] \\
&\quad + P[k_A(i)+1 \geq k_B(i)] \cdot P[k_A(i+1) < k_B(i+1)] \\
&= (P[k_A(i)+1 < k_B(i)] + P[k_A(i)+1 \geq k_B(i)]) \\
&\quad \times P[k_A(i+1) < k_B(i+1)] \\
&= P[k_A(i+1) < k_B(i+1)]
\end{aligned}$$

Since A won the previous backoff, $k_A(i)$ is either 0 or 1, each with probability 1/2. On the other hand, $k_B(i)$ is in the range $0 \ldots 2^i - 1$, each with probability 2^{-i}, unless $i \geq 10$, in which case the range is $0 \ldots 1023$, each with probability $\frac{1}{1024}$.

For $1 \leq i \leq 9$,

$$\begin{aligned}
P[k_A(i) < k_B(i)] &= P[k_A(i)=0] \times P[k_B(i) > 0] + P[k_A(i)=1] \times P[k_B(i) > 1] \\
&= \frac{1}{2} \times \frac{2^i - 1}{2^i} + \frac{1}{2} \times \frac{2^i - 2}{2^i} \\
&= \frac{2^{i+1} - 3}{2^{i+1}}
\end{aligned}$$

For $10 \leq i \leq 16$,

$$\begin{aligned}
P[k_A(i) < k_B(i)] &= P[k_A(i)=0] \times P[k_B(i) > 0] + P[k_A(i)=1] \times P[k_B(i) > 1] \\
&= \frac{1}{2} \times \frac{2^{10} - 1}{2^{10}} + \frac{1}{2} \times \frac{2^{10} - 2}{2^{10}} \\
&= \frac{2045}{2048}
\end{aligned}$$

Then, the formula above becomes

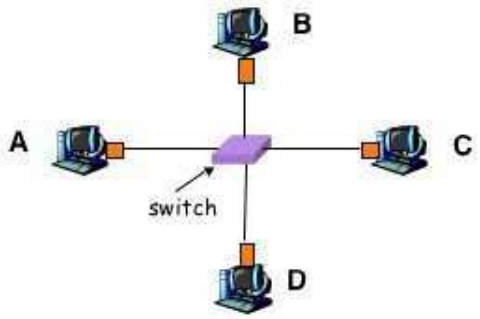
$$\begin{aligned} P[A\ wins\ remaining\ races] &= \prod_{i=4}^{16} P[A\ wins\ i | A\ wins\ i-1] \\ &> \prod_{i=4}^{16} P[k_A(i) < k_B(i)] \\ &= \prod_{i=4}^{9} P[k_A(i) < k_B(i)] \cdot \prod_{i=10}^{16} P[k_A(i) < k_B(i)] \\ &= \prod_{i=4}^{9} \frac{2^{i+1}-3}{2^{i+1}} \cdot \prod_{i=10}^{16} \frac{2045}{2048} \\ &\approx 0.82 \end{aligned}$$

Therefore, 0.82 is an approximate lower bound (but it is not tight).

(d) What then happens to the frame B_1?
Answer. B_1 will be dropped, and B will try the next frame B_2.

This scenario is known as the Ethernet *capture effect*.

4. **Switches and forwarding** The figure shows four nodes – A, B, C, and D – connected via a switch.



Suppose the switch has a forwarding table that can hold 10 entries, and when it needs to add a new entry it evicts the oldest entry to make room. Assume A has been compromised by an attacker, after which B starts communicating with C. Suppose B sends 10 frames to C every second at a steady rate, and whenever C receives two of these frames, it sends a single frame in response (so it sends 5 frames every second, at a steady rate). No frames are lost or corrupted.

(a) On how much of B's communication with C can A eavesdrop (i.e., is it able to see the traffic)? Explain why. Consider both directions of the communication, and assume the switch starts with an empty forwarding table.

Answer. A can eavesdrop on the first 2 frames sent from B to C, but not on any of the replies sent in the other direction.

The switch starts with an empty forwarding table. When the first frame from B destined to C arrives at the switch, the switch does not know to which interface C is connected. Therefore, the switch floods the frame, and A can read a copy (assuming that A's NIC is operating in *promiscuous* mode).

At that point, the switch learns to which interface B is connected. But when the second frame from B arrives, the switch still does not know where to forward it. The frame is flooded again, and A hears the frame also.

After C receives two frames, it sends a single frame in response. This frame is destined to B, which the switch already knows how to directly forward. Since the frame is not flooded, A cannot eavesdrop on it. In addition, at this point the switch learns to which interface C is connected, so A no longer sees transmissions destined for it, either.

(b) If A is able to reprogram the MAC address of its NIC, how can it use this to eavesdrop on more traffic? What must it do to maximize the amount of traffic it can see, and what proportion of the total communication between B and C does that amount to?

Answer. By continuously sending frames with different MAC addresses to the switch, A can evict entries in the forwarding table of the switch. The switch will then lose information about which host is connected to which interface, and will flood subsequent traffic destined for the evicted hosts.

To maximize the amount of traffic A eavesdrops, A needs to evict every new legitimate entry entered into the switch's forwarding table before the entry is used for forwarding. Doing so requires A to send 10 frames with 10 different spoofed MAC addresses in between any time when one of B or C transmits a frame, and when they next are the destination of a frame.

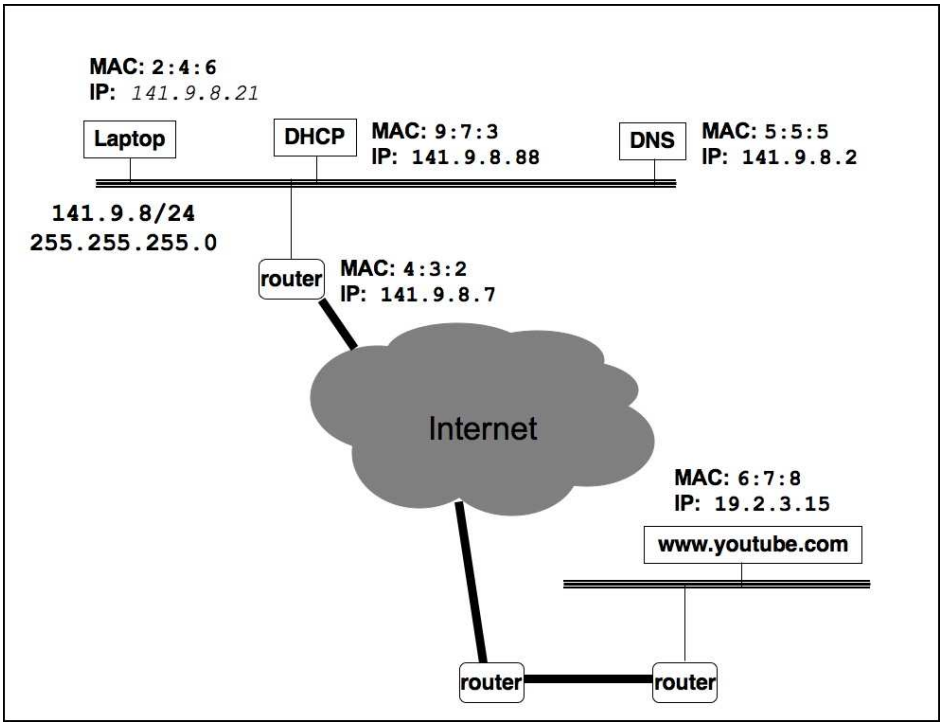
If the spacing between transmissions is large enough to always allow A to slip in these 10 frames, then A can eavesdrop on the entire traffic. As B and C only send 15 frames in total each second, this requires A to transmit 150 frames each second (though in bursts). (In fact, it can send fewer, since it need only evict B's entries just before C transmits to B, not after every one of B's transmissions.)

If, on the other hand, C responds quickly each time it receives a second frame from B, or if B transmits its next frame very shortly after receiving a frame from C, then A might not be able to flood the table quickly enough to force an eviction. In this case, it misses either C's traffic to B, or half of B's traffic to C, depending on which of the two hosts transmits just after receiving a frame from the other. As a result, A will see 10 out of 15 frames, or two-thirds of all of the traffic.

(c) Suppose instead A wishes to disrupt the communication between B and C. If it can reprogram the NIC's MAC address, how could A do this? How many frames must it send?

Answer. If A sends a frame using as its source MAC address the address belonging to some other host, the switch will associate A with that MAC address rather than the proper host. The switch will forward frames destined to that host to A, until the rightful host sends another frame.

Consider the first frame sent from B to C. It is flooded, so because of this A learns both B's and C's MAC addresses.
A then spoofs a frame with a fake source of C's MAC address. This means that B's second packet to C will go to A, rather than to C. A can simply discard this frame and any subsequent frames, and no further progress will occur, since C only generates frames in response to receiving them from B, and in this case it will never receive a second frame from B. (C received the first frame because it was flooded.)
Thus, it takes just one spoofed frame, sent during the 100 msec between B's first two frames. Even if A starts the attack later, it can quickly and completely disrupt C's ability to receive from B using just one spoofed frame.



5. **The Complete Life of a Web Page Fetch.**

   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 141.9.8/24 block, with a corresponding netmask of 255.255.255.0. 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 `2:4:6`. It initially does *not* have an IP address, though once it acquires one, it will be 141.9.8.21.

   You type `http://www.youtube.com` into your browser's URL line and hit return. Back comes the YouTube home page.

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. Number the packets in the order they are sent or received, and for each give:

- Source MAC and IP addresses, if any.
- Destination MAC and IP address, if any.
- Transport protocol and flag bits (e.g., TCP SYN), if any.
- Description of the message carried in the packet.
- For data packets, the sender's CWND (in terms of MSS-sized packets), as applicable.

For example, here is how we might describe the tail end of an SMTP session during which your laptop sends mail to a server running on the same machine as the DNS server in the diagram (your laptop has already acquired an IP address and both hosts know the MAC address of the other):

```
1. laptop -> DNS
     src MAC 2:4:6, IP 141.9.8.21 ; dst MAC 5:5:5, IP 141.9.8.2
     TCP data, "QUIT" + ack of prev. msg, CWND=6
2. DNS -> laptop
     src 5:5:5, 141.9.8.2 ; dst 2:4:6, 141.9.8.21
     TCP data, "221 closing" + ack of prev. msg, CWND=5
3. laptop -> DNS (same addresses)
     TCP FIN + ack of prev. message, beginning of close handshake
4. DNS -> laptop (same)
     TCP FIN + ack of FIN, both sides closed
5. laptop -> DNS (same)
     TCP ACK of FIN, termination handshake complete
```

In addition to needing to acquire an IP address, assume:

- No packets are lost.
- Your laptop's DNS cache is empty.
- Your laptop's ARP cache is empty.
- 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.)
- The HTTP request fits in a single packet, but the reply requires four packets. You can describe the request as just "HTTP GET" (no need to describe details or headers) and the replies as "HTTP REPLY 1" . . . "HTTP REPLY 4".
- All TCPs use delayed acknowledgments (ack-every-other) and standard congestion control (including the MSS*MSS/CWND version of Congestion Avoidance, as applicable).

- DNS requests are made using UDP.

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).
- ARP is a link-level protocol and as such does *not* use IP addresses, since its messages are framed at a lower layer. For DHCP, the story is more complicated (see RFC 2131), but it runs over UDP, and thus uses IP addresses. However, this point was not clear in lecture, so we did not require IP addresses for these messages.
- 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.