Solutions for Homework #3

EE122: Introduction to Communication Networks

(Fall 2006)

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

Vern Paxson / Sukun Kim / Dilip Antony Joseph

1. Problems from Peterson & Davie.

   (a) **Exercise 2.4** (6 points)

   To have at most one leading 0, the first two bits should be one of 01, 10, 11. Likewise, to have at most one trailing 0, the last two bits should be one of 01, 10, 11. One bit in the middle can be either 0 or 1. Therefore, any 5-bit sequence of the form $(01|10|11)(0|1)(01|10|11)$ is good.

   There can be 3 possible cases for the first two bits. There can be 2 possible cases for the third bit in the middle. Finally, there can be 3 possible cases for the last two bits. The total number of possible sequences is $3 \times 2 \times 3 = 18$. All 16 4-bit sequences can be mapped.

   (b) **Exercise 2.6** (6 points)

   To distinguish sentinels, bit stuffing is used at the sender. Bit stuffing is inserting a 0 after 5 consecutive 1s. If when receiving we observe more than 5 consecutive 1s, this should be due to the sentinel, which has the form 01111110. If instead we observe 01111111, then an error has occurred.

   After removing any 0 following 5 consecutive 1s from the sequence of bits we received over the link, we obtain the following bit sequence:

   11010111111011111010111111110

   Since when unpacking the bit sequence we never encountered six 1s in a row, then it contained neither the sentinel nor an errored version of the sentinel (01111111), so we do not see evidence of any error.

   (c) **Exercise 2.21** (6 points)

      (a) (3 points)

Let M be the set of all 8-bit messages with a single 1 bit.

$$\begin{aligned} M &= \{00000001, 00000010, 00000100, 00001000, \\ & \quad 00010000, 00100000, 01000000, 10000000\} \end{aligned}$$

Any message $m_1 \in M$ can be transmuted into any other message $m_2 \in M (m_1 \neq m_2)$ with a 2-bit error. For example, assume 00000001 had bit flips at the last two bits, causing it to become 00000010. To detect a 2-bit error, each message in M must have a different error code. However, with 2 bits for error detection we can only express $2^2 = 4$ error codes. Thererfore, there must exist $m_1, m_2 \in M, m_1 \neq m_2$, such that m_1 and m_2 share the same error code. If m_1 is transmuted to m_2 due to a 2-bit error, since their error codes are the same, this 2-bit error cannot be detected.

Therefore, there is no error detection code of size 2 bits that can detect all 2-bit errors for 8-bit messages.

(b) (3 points)

Let M be the set of all N-bit messages with a single 1 bit. There are N elements in M. Any message $m_1 \in M$ can be transmuted into any other message $m_2 \in M (m_1 \neq m_2)$ with a 2-bit error. For example, assume 0000....0001 had bit flips at the last two bits, then it becomes 0000....0010. Since this is a 2-bit error, any code that detects up to 8-bit errors must detect it.

To detect it, however, each message in M needs to have a different error code (same as in the previous problem). However, with 32 bits we can only generate 2^{32} different error codes. Therefore, if $N > 2^{32}$, there must exist $m_1, m_2 \in M (m_1 \neq m_2)$ such that m_1 and m_2 share the same error code. When m_1 is transmuted to m_2 with a 2-bit error, since they have the same error code, we cannot detect the 2-bit error.

This then shows that if $N = 2^{32} + 1$, there is no error detection code of size 32 bits that can detect all errors altering up to 8 bits, since such codes cannot even detect all 2-bit errors.

(d) **Exercise 2.44** (8 points)

(a) (2 points)

For the second backoff race, A picks either $k_A(2)$ 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) (2 points)

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) (2 points)

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 $1 \leq i \leq 9$,

$$P[k_A(i) < k_B(i)] = P[k_A(i)=0] \times P[k_B(i)>0] + P[k_A(i)=1] \times P[k_B(i)>1]$$

$$\begin{aligned} &= \frac{1}{2} \times \frac{2^i - 1}{2^i} + \frac{1}{2} \times \frac{2^i - 2}{2^i} \\ &= \frac{2^{i+1} - 3}{2^{i+1}} \end{aligned}$$

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) (2 points)
B_1 will be dropped, and B will try the next frame B_2.

(e) **Exercise 3.13** (6 points)

The description of this problem and its companion (3.14) worked out in the back of the book have ambiguities. Accordingly, the following two answers are both considered correct. The first one reflects an interpretation "over which links will the given bridge not forward flooded traffic?" while the second reflects an interpretation "for which links is the given bridge not selected as the designated bridge?".

The following ports are not selected by the spanning tree algorithm:

Bridge	Port
Bridge 1	
Bridge 2	A
Bridge 3	
Bridge 4	
Bridge 5	B, F
Bridge 6	I
Bridge 7	

Bridge	Port
Bridge 1	
Bridge 2	A,D
Bridge 3	E
Bridge 4	G
Bridge 5	B, F
Bridge 6	H, I
Bridge 7	C

(f) **Exercise 4.13** (6 points)

Until some host ARPs for the address shared by A and B, A's existing connections will proceed without any impairment. B will not even hear the traffic since packets will not be destined for its MAC address, and thus B's NIC will discard them at the link layer.

However, once a host ARPs for the address, then a race will occur in terms of which of A's or B's replies winds up occupying the cache of the requestor (and perhaps the cache of others hosts if the opportunistically listen in on ARP replies that they can hear). If B's MAC address winds up in the cache, then A's existing connections will stall; in fact, if they are using TCP, then B's TCP stack will RST the connections as soon as it receives packets for them, because it has no associated state.

With self-ARP, B can notice that A is already using the IP address. It can then notify its user(s) of the problem and the need to change to a different address, and refrain from polluting the ARP tables of others hosts and preventing them from communicating with A.

(g) **Exercise 4.14** (7.5 points)

(a) (2.5 points)

The biggest problem is that if D does not exist, the host will send an ARP request for each packet; all but the first of these will be unnecessary, and will result in further unnecessary work as other hosts reply.

In addition, if D does not exist then we will fill up queue space with packets destined to it, with no way to reclaim this space.

(b) (2.5 points)

First, check whether an ARP is already pending for D, which can be done either by inspecting the destinations of the queued packets or by maintaining an additional table that lists pending ARPs.

Second, to recover queue space for packets sent to non-existing destinations, add a time-out when waiting for an ARP response. If the timeout expires, indicating D does not exist, remove any packets in the queue destined to D.

```
if (<Ethernet address for D is in ARP cache>)
  <send P>
else
  if (D not already present as destination of queued packets)
```

```
    <send ARP request to resolve D>
  if (<timer is not running>)
    <start timer>
  <put P into a queue>
---------------------------------------------
on <timeout>
  <remove packets in queue destined to D>
```

(c) (2.5 points)

Every time the host begins communicating with a new destination, it will suffer a timeout delay (assuming its communication uses a reliable service such as TCP).

The one benefit is that we do not need to manage the additional queue of pending packets.

(h) **Exercise 4.15** (7.5 points)

(a) (2.5 points)

Information	Distance to Reach Node					
Stored at Node	A	B	C	D	E	F
A	0	∞	3	8	∞	∞
B	∞	0	∞	∞	2	∞
C	3	∞	0	∞	1	6
D	8	∞	∞	0	2	∞
E	∞	2	1	2	0	∞
F	∞	∞	6	∞	∞	0

(b) (2.5 points)

Information	Distance to Reach Node					
Stored at Node	A	B	C	D	E	F
A	0	∞	3	8	4	9
B	∞	0	3	4	2	∞
C	3	3	0	3	1	6
D	8	4	3	0	2	∞
E	4	2	1	2	0	7
F	9	∞	6	∞	7	0

(c) (2.5 points)

Information	Distance to Reach Node					
Stored at Node	A	B	C	D	E	F
A	0	6	3	6	4	9
B	6	0	3	4	2	9
C	3	3	0	3	1	6
D	6	4	3	0	2	9
E	4	2	1	2	0	7
F	9	9	6	9	7	0

(i) **Exercise 4.17** (6 points)

(Node, Cost, Predecessor) is used.

Step	Confirmed	Tentative
1	(D,0,-)	
2	(D,0,-)	(A,8,D)
		(E,2,D)
3	(D,0,-)	(A,8,D)
	(E,2,D)	
4	(D,0,-)	(A,8,D)
	(E,2,D)	(B,4,E)
		(C,3,E)
5	(D,0,-)	(A,8,D)
	(E,2,D)	(B,4,E)
	(C,3,E)	
6	(D,0,-)	(A,6,C)
	(E,2,D)	(B,4,E)
	(C,3,E)	(F,9,C)
7	(D,0,-)	(A,6,C)
	(E,2,D)	(F,9,C)
	(C,3,E)	
	(B,4,E)	
8	(D,0,-)	(F,9,C)
	(E,2,D)	
	(C,3,E)	
	(B,4,E)	
	(A,6,C)	
9	(D,0,-)	
	(E,2,D)	
	(C,3,E)	
	(B,4,E)	
	(A,6,C)	
	(F,9,C)	

(j) **Exercise 4.20** (7.5 points)

Routing table before the C-E link fails:

Node A

Node	NextHop	
	C	D
B	**6**	12
C	**3**	11
D	**6**	8
E	**4**	10
F	**9**	17

Node B

Node	NextHop
	E
A	**6**
C	**3**
D	**4**
E	**2**
F	**9**

Node C

Node	NextHop		
	A	E	F
A	**3**	5	15
B	9	**3**	15
D	9	**3**	15
E	7	**1**	13
F	12	8	**6**

Node D

Node	NextHop	
	A	E
A	8	**6**
B	14	**4**
C	11	**3**
E	12	**2**
F	17	**9**

Node E

Node	NextHop		
	B	C	D
A	8	**4**	8
B	**2**	4	6
C	5	**1**	5
D	6	4	**2**
F	11	**7**	11

Node F

Node	NextHop
	C
A	**9**
B	**9**
C	**6**
D	**9**
E	**7**

Advertised routing table:

Node A

Node	Cost	NextHop
B	6	C
C	3	C
D	6	C
E	4	C
F	9	C

Node B

Node	Cost	NextHop
A	6	E
C	3	E
D	4	E
E	2	E
F	9	E

Node C

Node	Cost	NextHop
A	3	A
B	3	E
D	3	E
E	1	E
F	6	F

Node D

Node	Cost	NextHop
A	6	E
B	4	E
C	3	E
E	2	E
F	9	E

Node E

Node	Cost	NextHop
A	4	C
B	2	B
C	1	C
D	2	D
F	7	C

Node F

Node	Cost	NextHop
A	9	C
B	9	C
C	6	C
D	9	C
E	7	C

Routing table after the C-E link fails:

Node C

Node	NextHop		
	A	E	F
A	**3**	∞	15
B	**9**	∞	15
D	**9**	∞	15
E	**7**	∞	13
F	12	∞	**6**

Node E

Node	NextHop		
	B	C	D
A	**8**	∞	8
B	**2**	∞	6
C	**5**	∞	5
D	6	∞	**2**
F	**11**	∞	11

Advertised routing table:

Node C

Node	Cost	NextHop
A	3	A
B	9	A
D	9	A
E	7	A
F	6	F

Node E

Node	Cost	NextHop
A	8	B
B	2	B
C	5	B
D	2	D
F	11	B

(a) (2.5 points)

Routing table:

Node A

Node	NextHop	
	C	D
B	**12**	12
C	**3**	11
D	12	**8**
E	**10**	10
F	**9**	17

Node B

Node	NextHop
	E
A	**10**
C	**7**
D	**4**
E	**2**
F	**13**

Node D

Node	NextHop	
	A	E
A	**8**	10
B	14	**4**
C	11	**7**
E	12	**2**
F	17	**13**

Node F

Node	NextHop
	C
A	**9**
B	**15**
C	**6**
D	**15**
E	**13**

Advertised routing table:

Node A

Node	Cost	NextHop
B	12	C
C	3	C
D	8	D
E	10	C
F	9	C

Node B

Node	Cost	NextHop
A	10	E
C	7	E
D	4	E
E	2	E
F	13	E

Node D

Node	Cost	NextHop
A	8	A
B	4	E
C	7	E
E	2	E
F	13	E

Node F

Node	Cost	NextHop
A	9	C
B	15	C
C	6	C
D	15	C
E	13	C

(b) (2.5 points)

Routing table:

Node A

Node	NextHop	
	C	D
B	**12**	12
C	**3**	15
D	12	**8**
E	**10**	10
F	**9**	21

Node D

Node	NextHop	
	A	E
A	**8**	10
B	20	**4**
C	11	**7**
E	18	**2**
F	17	**13**

Advertised routing table:

Node A

Node	Cost	NextHop
B	12	C
C	3	C
D	8	D
E	10	C
F	9	C

Node D

Node	Cost	NextHop
A	8	A
B	4	E
C	7	E
E	2	E
F	13	E

(c) (2.5 points)

Routing table:

Node C

Node	NextHop		
	A	E	F
A	**3**	∞	15
B	**15**	∞	15
D	**11**	∞	15
E	**13**	∞	13
F	12	∞	**6**

Advertised routing table:

Node C

Node	Cost	NextHop
A	3	A
B	15	A
D	11	A
E	13	A
F	6	F

(k) **Exercise 4.31** (8 points)

(a) (2 points)
The path can be any route without loop, as long as policy does not prevent it. There can be up to 3 routes: <Q,P> through link 1, <Q,P> through link 2, and <Q,R,P>.

(b) (2 points)
The provider of host A is P, and the provider of B is Q. Traffic from A to B will be routed via the closest link to B's provider (namely, Q), which is Link 1. In the same way, traffic from B to A will travel via Link 2.

(c) (2 points)
In a selection policy, give a higher preference to a route with a shorter distance.

(d) (2 points)
In a selection policy, give a higher preference to a route whose next AS is the provider R than a route whose next AS is the provider P.

2. (7.5 points)

(a) (2.5 points)
Total propagation delay is the sum of propogation time through the wire and delays in all repeaters.

$$\begin{aligned}(Propagation\ time) &= \frac{(Distance)}{(Propagation\ Speed)} \\ &= \frac{900m}{2 \cdot 10^8 m/s} = 4.5\mu s\end{aligned}$$

$$\begin{aligned}(Delay\ in\ Repeater) &= \frac{(Delay\ in\ bits)}{(Transmission\ Rate)} \\ &= \frac{20bits}{10Mbps} = 2\mu s\end{aligned}$$

Since there are 4 repeaters

$$\begin{aligned}(Propagation\ Delay) &= (Propagation\ time) + 4 \times (Delay\ in\ Repeater) \\ &= 4.5\mu s + 4 \times 2\mu s \\ &= 12.5\mu s\end{aligned}$$

(b) (2.5 points)
Propagation delay of $12.5\mu s$ between A and B is smaller than the backoff interval of $51.2\mu s$ $(= 512 \text{ bits}/10 \text{ Mbps})$. With CSMA/CD, when the start of the frame from A arrives at B, B will not send a packet. Therefore, A's packet will be delivered to B without collision or retry. Then

$$(Latency) = (Propagation\ Delay) + (Transmit\ Time)$$

$$\begin{aligned}
&= 12.5\mu s + \frac{(Size)}{(Bandwidth)} \\
&= 12.5\mu s + \frac{\text{125 bytes}}{\text{10 Mbps}} \\
&= 12.5\mu s + 100\mu s \\
&= 112.5\mu s
\end{aligned}$$

Therefore, A's packet is completely delivered to be at a time $112.5\mu s$ after it begins transmitting it.

We could be more precise by computing the time since the start of the entire tranmission. To do so, we include the time required for the first collision to be detected, which is 512 bit times, or $51.2\mu s$. Thus, the packet is completely delivered $164\mu s$ after A began, ignoring the inter-frame delay.

(c) (2.5 points)

Total propagation delay is the sum of propagation time through the wire and delays in all bridges. In each bridge, there is a store-and-forward delay, which is equal to the transmit time, and a 20-bit processing delay.

$$\begin{aligned}
(Propagation\ time) &= \frac{(Distance)}{(Propagation\ Speed)} \\
&= \frac{900m}{2 \cdot 10^8 m/s} = 4.5\mu s
\end{aligned}$$

$$\begin{aligned}
(Delay\ in\ Bridge) &= (\textit{Store-and-forward Delay}) + (Processing\ Delay) \\
&= (Transmit\ Time) + (Processing\ Delay) \\
&= \frac{(Size)}{(Bandwidth)} + \frac{(Delay\ in\ bits)}{(Transmission\ Rate)} \\
&= \frac{\text{125 bytes}}{\text{10 Mbps}} + \frac{\text{20 bits}}{\text{10 Mbps}} \\
&= 100\mu s + 2\mu s \\
&= 102\mu s
\end{aligned}$$

There are 4 bridges.

$$\begin{aligned}
(Propagation\ Delay) &= (Propagation\ time) + 4 \times (Delay\ in\ Bridge) \\
&= 4.5\mu s + 4 \times 102\mu s \\
&= 412.5\mu s
\end{aligned}$$

Total latency becomes

$$(Latency) = (Propagation\ Delay) + (Transmit\ Time)$$

$$\begin{aligned}
&= 412.5\mu s + \frac{(Size)}{(Bandwidth)} \\
&= 412.5\mu s + \frac{125 \text{ bytes}}{10 \text{ Mbps}} \\
&= 412.5\mu s + 100\mu s \\
&= 512.5\mu s
\end{aligned}$$

Therefore, at time $t = 512.5\mu s$, A's packet is completely delivered to B.

3. (10.5 points)

   (a) (1.5 points)

   Headers: **Date, Server, Set-Cookie, x-amz-id-1, x-amz-id-2, Vary, Content-Type, Connection**. If you fetched using HTTP 1.1, you also might have seen **Transfer-Encoding** and a bizarre **nnCoection**.

   Cookies set: **skin, session-id, session-id-time**, each with attributes of **path, domain, expires**.

   (b) (1.5 points)

   101,434 bytes

   (c) (1.5 points)

   Described in RFC 1945:

   Date the date and time at which the message was originated

   Server information about the software used by the origin server to handle the request

   Content-Type the media type of the *Entity Body* sent to the recipient

   (d) (1.5 points)

   Described in RFC 2616 and also in RFC 1945: **Date**, **Server**, **Content-Type**.

   Described only in RFC 2616:

   Transfer-Encoding specifies how the entity returned is encoded. For HTTP 1.1, the value we observed as *chunked*, indicating the entity was returned in several pieces, each of which starts with the size of the piece (expressed in hexadecimal).

   Vary indicates the set of *request-header* fields that fully determines, while the response is fresh, whether a cache is permitted to use the response to reply to a subsequent request without revalidation. In this case, the server is indicating that the version of the page it returned was determined by the setting of *Accept-Encoding* and *User-Agent*, so if the result is cached, it should only be used from the cache if those fields have the same values as they did in this request.

   Connection specifies options that are desired for that particular connection

   (e) (1.5 points)

   Set-Cookie carries state information between participating origin servers and user agents. See RFC 2109.

x-amz-id-1, x-amz-id-2 Headers for Amazon’s internal use. See `http://developer.amazonwebservices.com/connect/thread.jspa?messageID=46114`

nnCoection Appears to be due to a weird hack in a hardware load-balancer. See `http://www.nextthing.org/archives/2005/08/07/fun-with-http-headers`

(f) (1.5 points)

All headers were found and are discussed above.

(g) (1.5 points)

When fetching using HTTP 1.1, the transfer is encoded using *chunking*, as discussed above, and there is the additional bizarre **nnCoection** header.

Note, if you access the server manually using Telnet, then the following:

```
GET / HTTP/1.0
<<empty line>>
```

works fine, but:

```
GET / HTTP/1.1
<<empty line>
```

produces an error message:

```
400 Bad Request
Bad Request
Your browser sent a request that this server could not understand.
```

because with HTTP 1.1 you must include a **Host** header in your request. The following:

```
GET / HTTP/1.1
Host: www.amazon.com
<<empty line>
```

will work.

4. (7.5 points)

(a) (2.5 points)

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. 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) (2.5 points)

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) (2.5 points)

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.