ECE 4380/6380: Computer Communications **Spring 2017**
Problem Set 5 Solutions

All problems carry equal weight.

1. **Datagram forwarding.** Chapter 3, number 4. Notice that table entries with the same output port as the default entry are removed from the table. This improves the performance of the router as it has fewer table entries to search when forwarding a packet.

S1	
Destination	port
A	1
B	2
Default	3

S3	
Destination	port
C	2
D	3
Default	1

S2	
Destination	port
A	1
B	1
C	3
D	3
Default	2

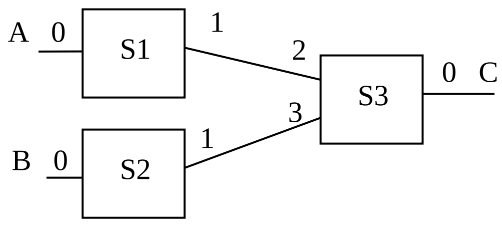
S4	
Destination	Port
D	2
Default	1

2. **Virtual Circuit forwarding**. Chapter 3, number 5. Let, S*i*[*j*] denote the *j*-th entry in the table for switch S*i* (where we are counting from 1 at the top of the table).
A connects to D with VC S1[1] – S2[1] – S3[1]
A connects to B with VC S1[2]
B connects to D with VC S1[3] – S2[2] – S3[2]
3. **Source routing.** Chapter 3, number 6. We provide space in the packet header for a second address list, in which we build the return address. Each time the packet traverses a switch, the switch must add the *inbound* port number to this return-address list, in addition to forwarding the packet out the output port listed in the "forward" address. For example, as the packet traverses Switch 1 in Figure 3.7, towards forward address "port 1", the switch writes "port 2" into the return address. Similarly, Switch 2 must write "port 3" in the next position of the return address. The return address in complete once the packet arrives at its destination.

Another possible solution is to assign each switch a locally unique name; that is, a name not shared by any of its directly connected neighbors. Forwarding switches (or the originating host) would then fill in the sequence of these names. When a packet was sent in reverse, switches would use these names to look up the previous hop. We might reject locally unique names, however, on the grounds that if interconnections can be added later it is hard to see how to permanently allocate such names without requiring global uniqueness.

Note that switches <u>cannot</u> figure out the reverse route from the far end, given just the original forward address. The problem is that multiple senders might use the same forward address to reach a given destination; no reversal mechanism could determine to which sender the response was to be delivered. As an example, suppose Host A connects to port 0 of Switch 1, Host B connects to port 0 of Switch 2, and Host C connects to port 0 of Switch 3. Furthermore, suppose port 1 of Switch 1 connects to port 2 of

Switch 3, and port 1 of Switch 2 connects to port 3 of Switch 3. The source-routing path from A to C *and* from B to C is (1, 0); the reverse path from C to A is (2, 0) and from C to B is (3, 0).



4. **Forwarding Tables**.

(a) Because least-cost paths are used the forwarding table for switch 2 is:

Destination	Output port	Distance
A	0	1
B	1	0
C	0	10
D	0	7
OUT	0	10

(b) (i) The circuit beginning with (port, VCI) = (0, 0) at switch S1 is from A – S1 – S4 – S2 – B and the following rows are deleted: the first row at S1, the third row at S4, and the fourth row at S2. (ii) The new circuit D to B uses S4 – S1 – S2 and the port and virtual circuit Ids are listed in the tables. Note that the lowest ID number that does not conflict with an existing number is selected.

S1			
port$_{in}$	*VCI*$_{in}$	*port*$_{out}$	*VCI*$_{out}$
~~0~~	~~0~~	~~2~~	~~0~~
0	1	1	0
1	0	0	0
2	0	1	1

S3			
port$_{in}$	*VCI*$_{in}$	*port*$_{out}$	*VCI*$_{out}$
0	0	3	0
0	1	3	1
3	0	0	0

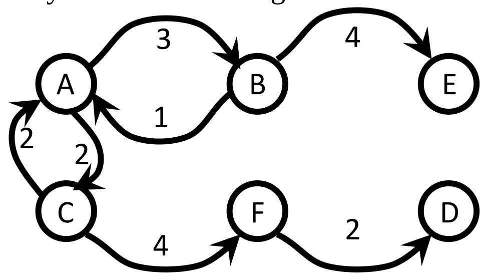
S2			
port$_{in}$	*VCI*$_{in}$	*port*$_{out}$	*VCI*$_{out}$
0	0	2	0
2	0	3	0
2	1	1	2
~~2~~	~~2~~	~~1~~	~~1~~
3	0	1	0
0	1	1	1

S4			
port$_{in}$	*VCI*$_{in}$	*port*$_{out}$	*VCI*$_{out}$
0	0	2	0
0	1	2	1
~~1~~	~~0~~	~~2~~	~~2~~
2	0	0	0
0	2	1	0

(c) For source routing only the *output* port numbers need to be listed in the packet header. The route is B – S2 – S1 – S4 – S3 – C, and the port numbers that should be in the packet header are (0, 2, 3, 3).

5. **Forwarding Tables**.

(a) The network connectivity is shown in the figure



Note that we are able to determine the cost of the link from F to D. The cost of the link from C to F is known to be 4 from C's forwarding table. Since the cost of the route from C to D is 6, and the first link on the route is the link C to F, this implies the cost of the route from F to D is 2. Since link costs are non-negative integers, the route from F to D can only have one or two links. Furthermore, we can rule out any of the possible two-link routes from F to D because of the other entries in the forwarding tables. For example, it is not possible for the route to be F to B to D, because this would imply that the link from B to D has cost of 1. But, B's minimum cost route to D has cost 9, thus the link from B to D must have cost 9 or greater. Likewise, the link from A to D or E to D cannot have cost 1. So, we can conclude that the F to D link has cost 2.

(b) The C to B link now has cost 1, since the next hop to B is now B. Previously, the cost of this link must have been 3 or greater.
Also, notice the C to D route is now via B with cost of 11. This implies that the cost of the F to D link has increased and is now 7 or larger. This also implies that the route from B to D must have a cost of 10, but we don't know the details of this route. It is possible B has a new cost for the link to D, or there is now a route through another node.

(c) The problem is that to reach destination D, C forwards to B, B to A, and A to C. A loop has (temporarily) formed because C has updated its forwarding table, but nodes A and B have not yet updated theirs. This illustrates a fundamental challenge in updating forwarding tables, since loops can form until all nodes have completed their changes to the forwarding tables.

6. **Ethernet Bridges**. Consider host 1. A packet it submits to the network is broadcast on the locally attached Ethernet cable. Also, if the destination is on one of the other Ethernet cables, only then will the bridge forward the packet. The destinations are equally likely, and on average host 1 will place a load of *R* on the locally attached cable, and the first bridge will forward 2/3 of the frames. So, the load placed on the middle LAN is 2*R*/3 and the load placed on the right LAN is *R/3*. Consider the total load placed on the left LAN. The *N* hosts attached to the LAN place a load of *R*. Hosts on the middle LAN will place a load of *R*/3 on the left LAN, as will the hosts on the right LAN. So, the total load for the left LAN is *R N* + (*R*/3) *N*+ (*R*/3) *N* = 5 *R N*/ 3. Because the Ethernet is assumed to be 80% efficient the average load it can support is 8 Mbps, and (8 Mbps) (3/5) / (100 Kbps) = 48. However, consider the load on the center LAN. The hosts from the left LAN place a load of 2*R*/3 on the center LAN, as do the hosts from the right LAN. So, the total load for the middle LAN is *R N* + (2*R*/3) *N*+ (2*R*/3) *N* = 7 *R N*/ 3. The load on the middle LAN is also limited to 8 Mbps, and (8 Mbps) (3/7) / (100 Kbps) = 34.28. The load on the middle LAN is greater than the other two LANs, so the maximum number of hosts is limited by the level of traffic the middle LAN can support. The maximum number of stations that can be support on average is 34.

Part (b). In this scenario, each of the lower LANs forwards 2*R*/3 of its frames to the backbone LAN. So, the total load on the backbone LAN is 3 × (2*R*/3), and the maximum number of hosts is limited to (8 Mbps) (1/2) / (100 Kbps) = 40. The load on each of the lower LANs is $R\,N + (R/3)\,N + (R/3)\,N = 5\,R\,N/3$. The lower LANs are not as heavily loaded as the backbone LAN, so the backbone LAN limits the maximum number of stations to 40.

Part (c). The lower LANs now limited the number of hosts. The load is the same as calculated in part (b) as is $5\,R\,N/3$ and the maximum number of hosts is 48.