CSE 123: Computer Networks

Homework 2 solutions

Out: 10/14, Due: 10/21

Instructions:

1. Turn in a physical copy at the beginning of the class on 10/21 .
2. Ensure the HW cover page has the following information clearly written:
    a. Name
    b. UCSD email
    c. PID
3. Please contact the TAs or post on Piazza to seek any clarification.
4. The homework is to be done individually.

1) **2-D Parity**

Consider the following set of bits sent with 2-D even parity; the data bits are in the 4x4 upper-left block and the parity bits are in the rightmost column and bottom row. The given data has a 1-bit corruption.

1	1	0	1	1
0	1	0	0	1
1	1	1	1	1
1	0	0	1	0
1	1	1	0	1

a) Which is the corrupted bit in the data given above?

`3Rd row and 4th column have incorrect parities hence, the corrupt bit is on their intersection.`

1	1	0	1	1
0	1	0	0	1
1	1	1	**1**	1
1	0	0	1	0
1	1	1	0	1

b) Can 2-D parity be used to **correct** all 2 bit errors? Show examples or a counter-example for the answer.

`No.`

`Example: -`

`Original Data`

1	1	0	1	1
0	1	0	0	1
1	1	1	0	1
1	0	0	1	0
1	1	1	0	1

`Corrupted Data`

0	0	0	1	1
0	1	0	0	1
1	1	1	0	1
1	0	0	1	0
1	1	1	0	1

The parities of the first two columns are incorrect but all the rows have correct parities so, there is no way to know which cells are corrupted.

c) What is the efficiency of this scheme?
   Total bits sent = (16 (data) + 9 (parity)) = 25
   Efficiency = 16/25 = 0.64

2) **CRC**

Suppose we wish to transmit the following data : **1101 1001 1110**. We are using x^4+x^2+1 as the CRC generator.

a) What is the transmitted bit sequence (show your work)?
   1101 1001 1110 1010
b) How large a burst of errors can be detected in this case?
   K+1 bit long generator can detect k bit burst. So, 4 bit large burst errors can be detected.
c) Can this CRC be used detect all single-bit errors? What about double-bit errors?
   - Yes, it can detect all single-bit errors since it has non-zero x^0 and x^k terms (k = 4, 2).
   - Yes, it can detect double-bit errors. Since it has three non zero terms.

3) **Sliding Window**

Suppose that we run the sliding window algorithm with SWS(Sender Window Size) = 6 and RWS(Receiver Window Size) = 4, and no out-of-order arrivals.

a) Find the smallest value for the "maximum sequence number"(N) to be used for successful transmission via Sliding Window algorithm.
   N = SWS + RWS – 1 = 6+4-1 = 9
b) Give an example showing that N – 1 is not sufficient for SWP, where N is the maximum sequence number calculated in part (a).
   max seq number = 9 -1 = 8.
   Sender sends first 6 frames 0, 1, 2, 3, 4, 5. The receiver receives all the frames but all the ACKs are lost. Now the receiver is expecting 6, 7, 8, 0 (since seq number wraps around, instead of 9 we have seq number 0). After timeout, the sender sends 0, 1, 2, 3, 4, 5 again. Receiver accepts frame 0 but it was the old incarnation instead of the new frame with seq number 0.
c) State a general rule for smallest maximum sequence number (N) in terms of SWS and RWS.
   N = SWS + RWS - 1

Assume that the sequence numbers are 0 based. So a max. seq. number = 4 means sequence numbers 0, 1, 2, 3 and 4 can be used.

4) **Sliding Window again**

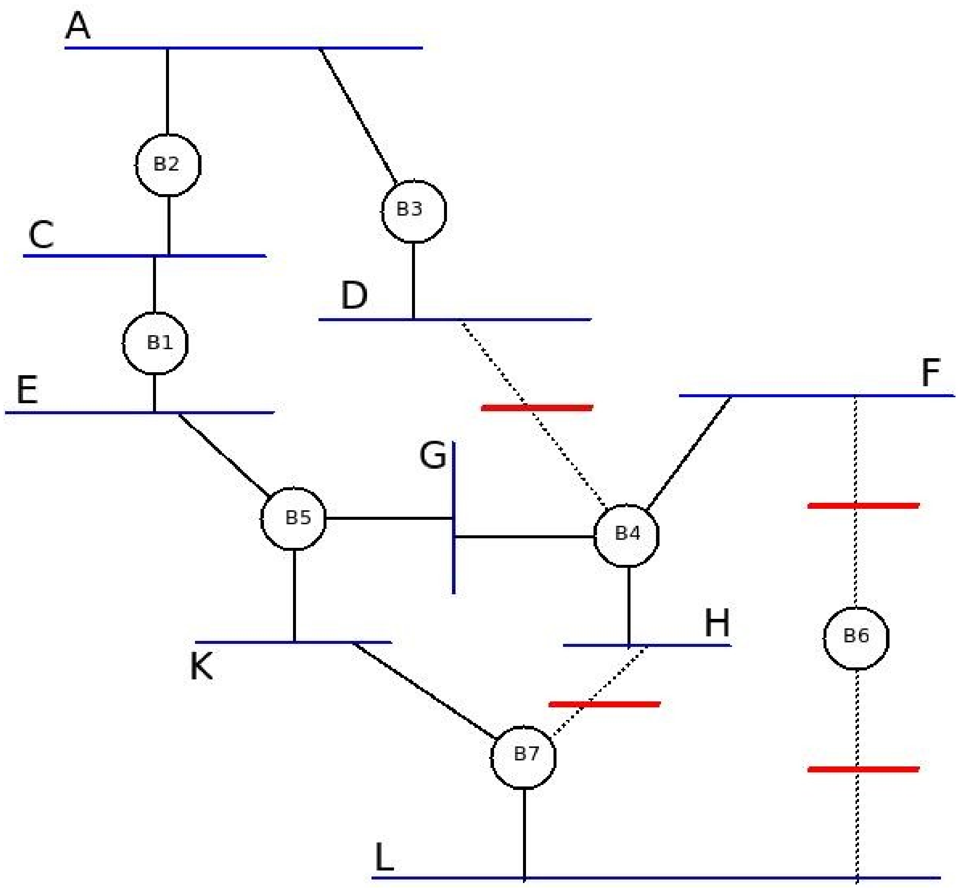
Suppose SWS=RWS=4. The sender has 9 frames to send with sequence numbers 0, 1, 2, 3, 4, 5, 6, 7, 8 respectively with timeout = 4*RTT. Provide a comma separated list showing the order of packets sent by the sender, by sequence number (include packets lost in transit and any retransmissions) for the following cases:-

a) Frame with sequence number 3 is lost.
`0, 1, 2, 3, 4, 5, 6, 3, 7, 8`

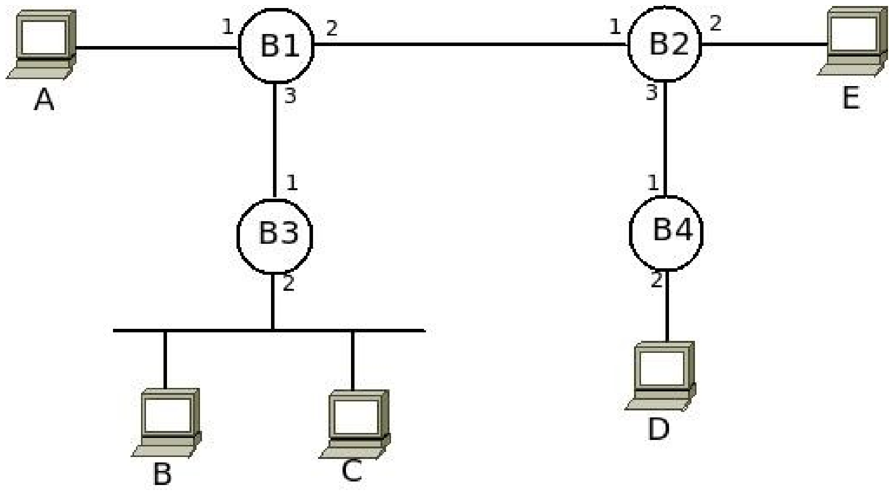
b) Frame with sequence numbers 3, 4, 5 and 7 are lost.
`0, 1, 2, 3, 4, 5, 6, 3, 4, 5, 6, 7, 8, 7, 8`

Assume that ONLY the frames mentioned in each part are lost and ONLY during their first transmission. No other frame, ACK or retransmission is lost.

5) **Spanning tree**
Given the extended LAN shown below, indicate which ports are not selected by the spanning tree algorithm.



6) **Learning bridge**
Consider the below network with bridges B1, B2, B3 and B4 with learning capability. All the nodes in the system have just booted up and all their forwarding tables are initially empty. Write down the forwarding tables of each bridge for the following sequence of events. Assume the events are in sequence and the tables from the previous step are retained.



a) B sends to C

Bridge B1	
Destination	**Port**
B	3

Bridge B2	
Destination	**Port**
B	1

Bridge B3	
Destination	**Port**
B	2

Bridge B4	
Destination	**Port**
B	1

b) A sends to B

Bridge B1	
Destination	**Port**
B	3
A	1

Bridge B2	
Destination	**Port**
B	1

Bridge B3	
Destination	**Port**
B	2
A	1

Bridge B4	
Destination	**Port**
B	1

c) E sends to A

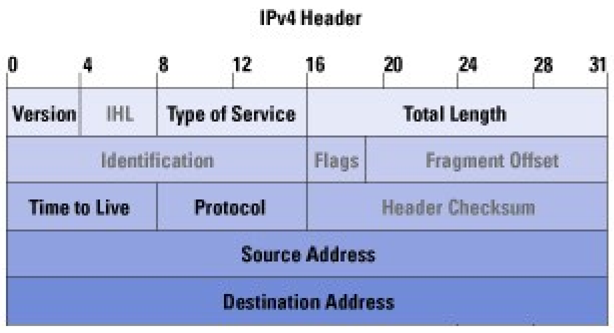
Bridge B1	
Destination	**Port**
B	3
A	1
E	2

Bridge B2	
Destination	**Port**
B	1
E	2

Bridge B3	
Destination	**Port**
B	2
A	1

Bridge B4	
Destination	**Port**
B	1
E	1

7) **IPv4 header**
Below is a diagram of the IPv4 header with its fields specified. As shown below, the width of each row is 32 bits (0-31).



An IPv4 packet is received at the destination. The IPv4 header of that packet has been mentioned below in network byte order. (The representation is in hexadecimal):
4500 0054 0000 4000 4001 7066 5014 010a 7620 0305

a) What is the IP header checksum value sent in this packet?
```
0x7066
```
b) Compute the header checksum for the packet using Internet checksum algorithm (show your work). Is there an error present in the header of the packet?
```
 4500 + 0054 + 0000 + 4000 + 4001 + 7066 + 5014 + 010a + 7620 +
0305 = 1fffe
Adding overflow back to least significant 16-bits = fffe + 1 = ffff
Checksum = ~ffff = 0000

No error.
```
c) Assuming the total length and header length fields to be error free, compute the length of the payload in this case? Show your work.
```
Total length = 0x0054 = 84 bytes
Header length = 0x5 32-bit words = 5 * 32 bits = 20 bytes
Payload length = total length – header length
               = 84 – 20 = 64 bytes
```
d) Find the TTL. Compute the new checksum after decrementing the TTL by 1.
```
TTL = 0x40 = 64
new TTL = 63 = 0x3f

4500 + 0054 + 0000 + 4000 + 3f01 + 0000 + 5014 + 010a + 7620 +
0305 = 18e98

Add the overflow back to the least significant 16 bits
= 8e98 + 1 = 8e99
New checksum = ~ 8e99 = 7166
```