**Objectives**

* To practice networking algorithms.
* To understand the sliding window algorithm.
* To build a foundation for understanding routing

You may complete this lab individually or with one colleague from this class using pair programming rules (see the course Canvas page). If you work in a pair, state clearly both lab members at the top of source/submission files.

For this lab, you may print and draw on this document by hand or illustrate it using word (or any other technology of your choice).

**1 Reliable Transmission (15 pts)**

1) Consider a variation of the Sliding Window algorithm. Assume the receiver sends a duplicate acknowledgment if it does not receive the expected frame. For example, it sends DUP\_ACK[2] when it expects to see Frame[2] but receives Frame[3] instead. Also, the receiver sends a cumulative acknowledgment after it receives all the outstanding frames. For example, it sends ACK[5] when it receives the lost frame Frame[2] after it already received Frame[3], Frame[4], and Frame[5]. Use a timeout interval of about 2×RTT.

Draw a timeline diagram for the sliding window algorithm with SWS = RWS = 3 frames in the following two situations. Label all frames, acknowledgements, and round trip times for the dropped frame.

1. The sender wishes to transmit frames 0-5. Frame 1 is lost. Retransmission takes place upon timeout (as usual). **(5 pts)**

Frame[0]

Ack[5]

Ack[4]

Frame[5]

Frame[4]

Frame[3]

Ack[3]

Ack[2]

2 RTT

Ack[1]

Frame[1]

1 RTT

4 RTT

3 RTT

Ack[0]

Frame[2]

Frame[1]

1. The sender wishes to transmit frames 0-5. Frame 1 is lost. Retransmission takes place either upon receipt of the ﬁrst DUP\_ACK or upon timeout. Does this scheme reduce the transaction time? (Note that some end-to-end protocols, such as variants of TCP, use similar schemes for fast retransmission.) **(5 pts)**

Frame[1]

Frame[0]

Ack[0]

Frame[2]

Ack[5]

Ack[4]

Frame[5]

Ack[2]

Ack[3]

Dup\_Ack[0]

Frame[4]

Ack[1]

Frame[1]

Frame[3]

2) Suppose that we run the sliding window algorithm with SWS = 5 and RWS = 3, and no out-of-order arrivals. As usual, each frame is marked with its sequence number, in order to let the receiver detect undelivered frames, and also order them. The sender maintains a counter variable that is used to keep sequence number. As you can imagine, this variable is not unbounded. It will eventually reach its maximum value and will have to wrap around. But what is the smallest value for MaxSeqNum that could possibly work? (You may assume that it sufﬁces to ﬁnd the smallest MaxSeqNum such that if Frame[MaxSeqNum] is in the receive window, then Frame[0] can no longer arrive. You can provide your number in a form of a number or a formula involving SWS and RWS. Justify your answer in a couple of sentences. **(5 pts)**

MaxSeqNum >= SWS + RWS

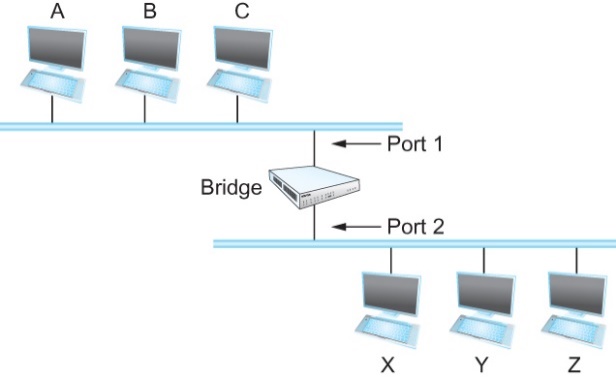
In this case that would mean that MaxSeqNum would be 8. This means there can’t be any mix-ups when the SeqNum resets to 0 again. If it were 7 then a situation such as this could happen which would lead to issues:

The sender sends frame[0] – frame[4] normally. The receiver sends Ack[4] but it arrives slowly. The receiver’s window is now frame[5] - frame[7]. The sender times out and then resends frame[0] but the receiver accepts it as frame[7].

**2 Bridging (10 pts)**

A**local area network (LAN)** supplies networking capability to a group of computers in close proximity to each other such as in an office building, a school, or a home. Most LANs these days are supported by Ethernet technology. We will focus our attention on the earlier versions of Ethernet, which were based on broadcasting transmissions. A network would be a set of hosts connected to a common cable.  Any signal placed on the Ethernet by a host would be broadcasted over the entire network, similarly how all radio stations broadcast the news and music over the shared medium (air). In that type of network, all hosts would be able to hear transmissions by all other hosts. (Imagine a lecture session with instructor and students. When instructor (host) transmits a frame “Luke Skywalker, where is your homework?” everyone can hear it but only Luke will act upon it).

That network would have physical size limitation, so in order to grow them, there would be a need to connect several of such networks together using bridges. (A bridge is essentially a LAN switch that forwards packets between shared-media LANs (Ethernets) that would forward the frames from one Ethernet to the other). For example, on the picture below, two LANs are connected with a bridge.



**Learning Bridges**

Observe that there is no need to forward all the frames that a bridge receives. For example, on the previous figure, when a frame from host A that is addressed to host B arrives on port 1, there is no need for the bridge to forward the frame out over port 2. How does a bridge come to learn on which port the various hosts reside? Can the bridge learn this information by itself? The answer is yes. Bridge inspects source address of frames. If frames from A come in Port 1, Port 1 should be used to forward to A. When a bridge first boots, this table is empty. Entries are added over time. A timeout is associated with each entry to protect handle situations in which hosts are moved from one network to another. If the bridge receives a frame that is addressed to host not currently in the table, forward the frame out on all other ports. Strategy works fine if the extended LAN does not have a loop in it. If there is a loop, frames potentially loop through the extended LAN forever.

3) Consider the following arrangement of learning bridges:

A close up of a clock

Description automatically generated

Assuming all are initially empty, give the forwarding tables for each of the bridges B1 to B4 after the following transmissions:

A sends to C. C sends to A. D sends to C.

Each forwarding table will maintain **<destination\_address, interface>** pairs. Mark ports/interfaces with the unique neighbor reached directly from that port; that is, the ports for B1 are to be labeled “A” and “B2” and use these values in the forwarding table. **(5 pts)**

B1

|  |  |
| --- | --- |
| Host | Port |
| A | left |
| B2 | right |

B2

|  |  |
| --- | --- |
| Host | Port |
| B1 | left |
| B3 | up |
| B4 | down |

B3

|  |  |
| --- | --- |
| Host | Port |
| C | right |
| B2 | down |

B4

|  |  |
| --- | --- |
| Host | Port |
| B2 | up |
| D | right |

4) Suppose learning bridges B1 and B2 form a loop:

A close up of a clock

Description automatically generated

Also assume that they do **not** implement the loop detection algorithm. Each bridge maintains a single table of **<address, interface>** pairs. Provide a short explanation describing what will happen if M sends to L? **(5 pts)**

If M sends to L, then once B1 receives the packet, it will forward it to L and B2. B2 will also forward it back to B1. This cycle repeats where B1 and B2 are constantly sending each other packet without ever reaching a destination.

**Submission**

Upload your completed version of this lab to canvas.