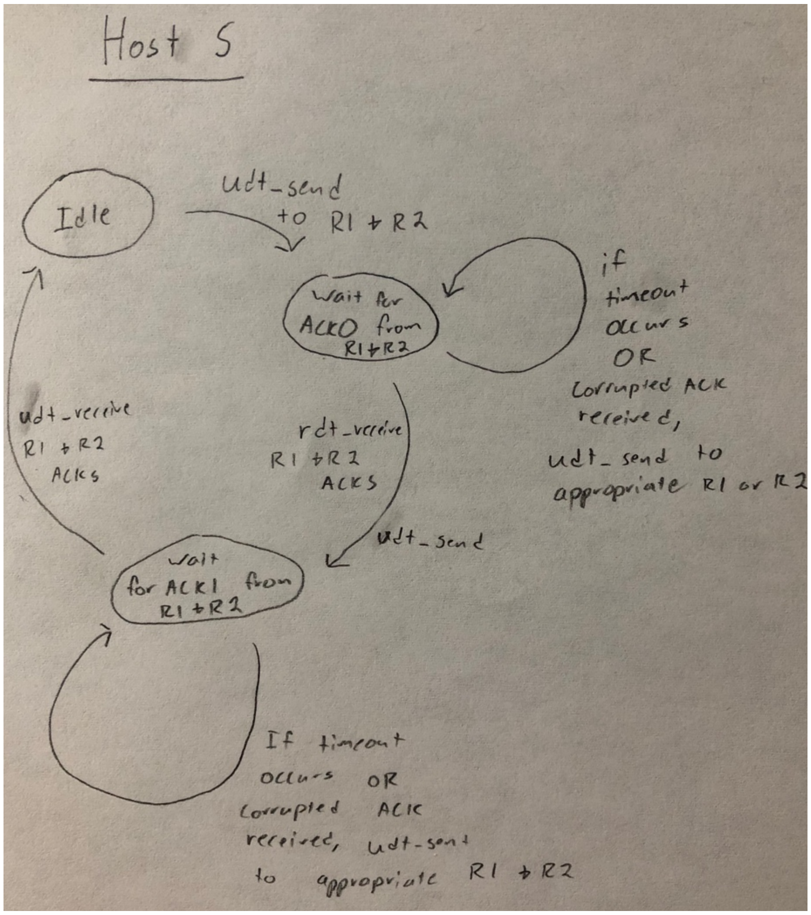
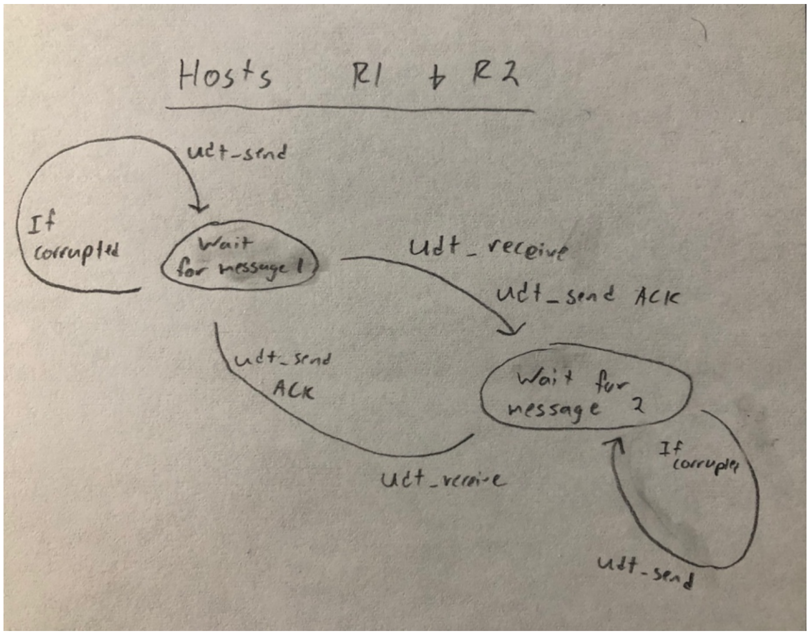
**Homework 2: Lower Network Layers**

**Question 1 -**


Host S
Idle
udt-send to R1 + R2
Wait for ACK0 from R1+R2
if timeout occurs OR Corrupted ACK received, udt_send to appropriate R1 or R2
udt-receive R1 + R2 ACKs
rdt_receive R1 + R2 ACKS
udt_send
Wait for ACK1 from R1 + R2
If timeout occurs OR Corrupted ACK received, udt-send to appropriate R1 + R2

S sends a message to both R1 & R2 and then waits a set amount of time for a valid acknowledgement from each of them. If one isn't received (either because of corruption or timeout) then it will resend the message. On a successful receipt I will send a second message to both R1 & R2 and again wait for set time to receive an acknowledgement.



R1 & R2 both wait indefinitely for a message from S and when one is received (if it is correctly formed) they will send an acknowledgement back to S and then begin waiting for the next message. If the message was corrupted R1 & R2 will still respond, but with a "NACK" instead indicating message failure. When the second message is received it will be handled in a similar fashion.

**Question 2 -**

Flow control keeps the sender from overloading the receiver by sending more information than the receiver can handle at a time. The receive window send in the response indicates how many bytes can currently be held in the receiver's buffer and the sender scales their message to keep from exceeding that buffer. The sender will never slow below sending one byte at a time to maintain the communication stream.

Congestion Control tries to keep the network from being overloaded by modulating the sender's speed. This is achieved by assuming that when a packet is dropped the network is too busy. When the sender has a sent packet timeout without receiving an acknowledgement that signals the sender to slow down. As a whole, the sender originally starts sending messages very slowly but rapidly ramps up in speed, once a packet timeout is received the sender returns to sending slowly and growing rapidly, but will instead slow down the speed growth rate when it approaches the speed at which the packet was previously dropped.

**Question 3 -**

| | |
|---|---|
| From A to X behind the NAT - | |
| Source: 10.0.0.1, port 8080 | Destination: 1.2.3.4, port 80 |
| From B to X behind the NAT - | |
| Source: 10.0.0.2, port 8080 | Destination: 1.2.3.4, port 80 |
| From A to X between X and the NAT - | |
| Source: 5.6.7.8, port 80 | Destination: 1.2.3.4, port 80 |
| From B to X between X and the NAT - | |
| Source: 5.6.7.8, port 81 | Destination: 1.2.3.4, port 80 |
| From X to A between X and the NAT - | |
| Source: 1.2.3.4, port 80 | Destination: 5.6.7.8, port 80 |
| From X to A between the NAT and A - | |
| Source: 5.6.7.8, port 8080 | Destination: 10.0.0.1, port 8080 |

Translation Table -

5.6.7.8:80 => 10.0.0.1 (A)
5.6.7.8:81 => 10.0.0.2 (B)

**Question 4 -**

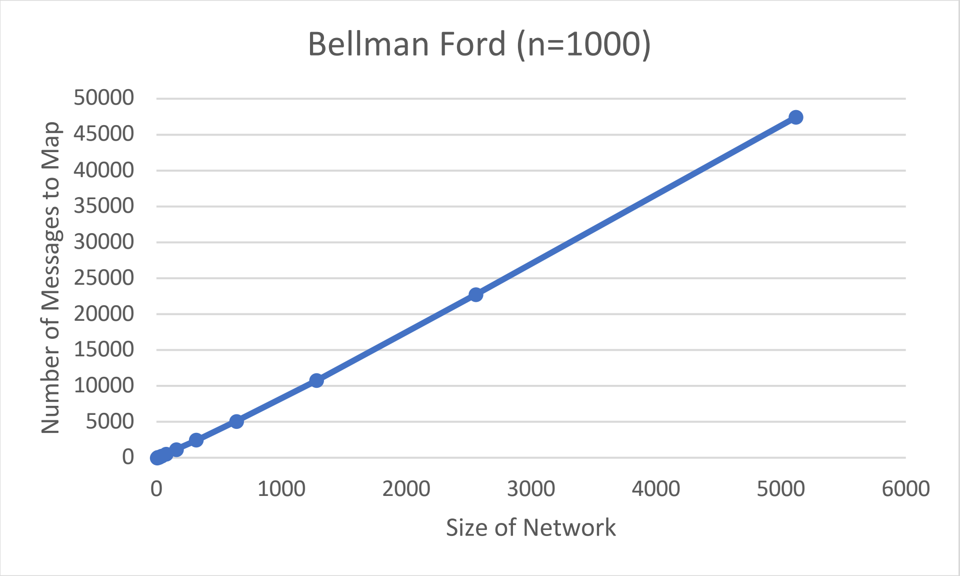
There are six subnets on the network and each one only needs a single bit to be correctly represented (since each subnet has no more than two links). Therefore 31 out of 32 bits can be fixed.

The smallest number of addresses necessary to buy require at least one complete byte (for the last value) and 3 free bits in the second to last byte to allow values from 1-6. That means that 21 of the bits 32 bits can be fixed.

Forwarding Table -

| **Port** | **Destination** | **Mask** |
|---|---|---|
| 1 (Group Subnet) | 127.0.0.1 | 255.0.0.0 |
| 2 (Router B) | 1.1.4.0 | 255.255.255.0 |
| 3 (Router C) | 1.1.5.0 | 255.255.255.0 |
| D (ISP) | ? | 255.255.255.255 |

**Question 5 -**



Based on a sample size of 1000 the cost in messages to map the network seems to grow linearly with the size of the Network (O(N)). This is impressive given that Dijkstra's is O(N*log(N)) and other map traversal algorithms are even slower!