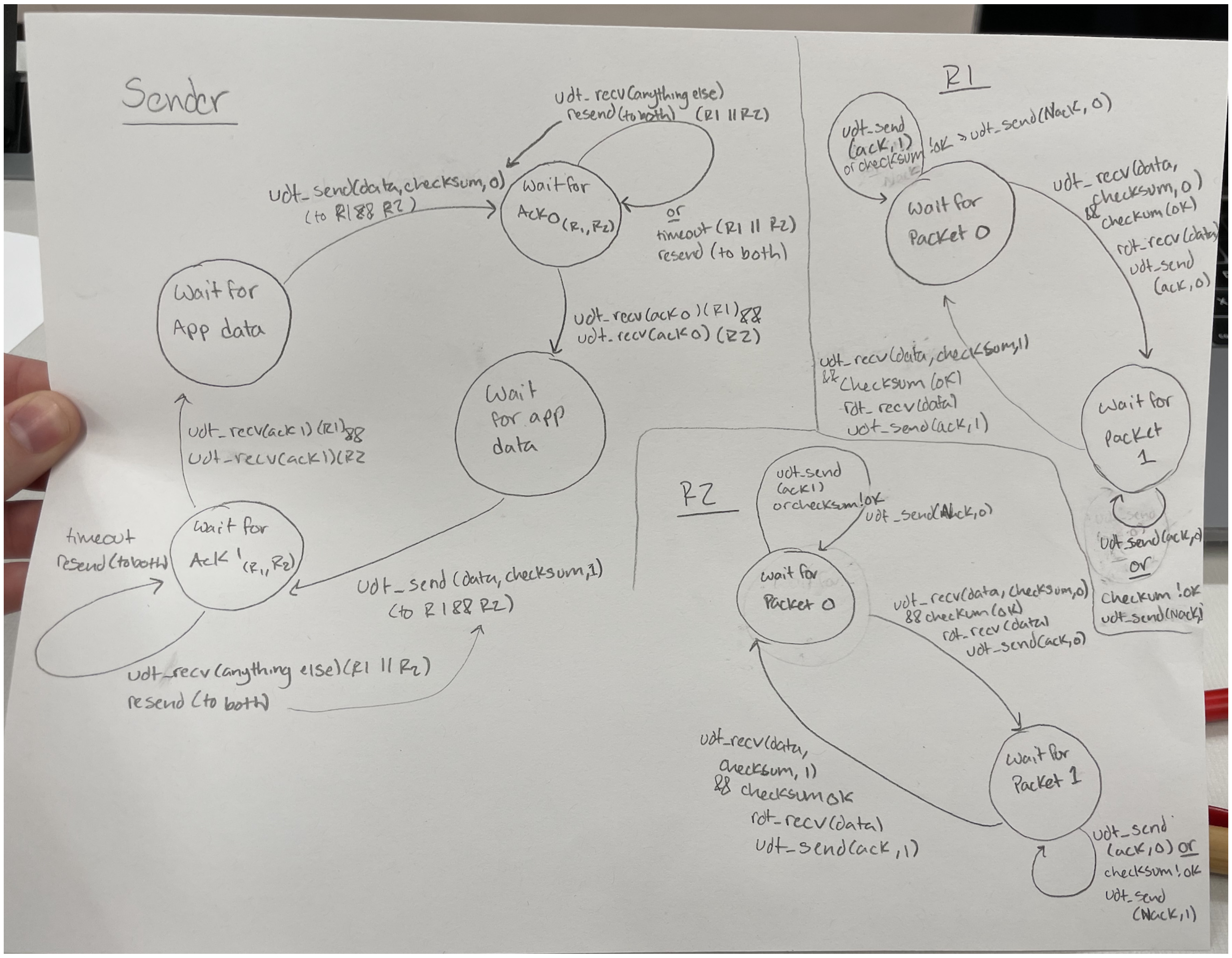
Question1:
(R1 & R2 are the same, just drawn independently)


Sender
Wait for App data
udt_send(data,checksum,0) (to R1&& R2)
Wait for Ack0 (R1,R2)
udt_recv(anything else) resend(to both) (R1 || R2)
or timeout (R1 || R2) resend (to both)
udt_recv(ack0)(R1)&& udt_recv(ack0) (R2)
Wait for app data
udt_send (data, checksum,1) (to R1&& R2)
Wait for Ack 1 (R1, R2)
timeout resend (to both)
udt_recv(anything else)(R1 || R2) resend (to both)
udt_recv(ack1) (R1)&& udt_recv(ack1)(R2
R1
udt_send (ack,1) or checksum !ok > udt_send(Nack, 0)
Wait for Packet 0
udt_recv(data, checksum, 0) && checkum (OK) rdt_recv(data) udt_send (ack,0)
Wait for Packet 1
udt_send(ack,0) or checkum !ok udt_send(Nack)
udt_recv(data, checksum,1) && checksum (OK) rdt_recv(data) udt_send(ack,1)
R2
udt_send (ack1) or checksum !ok udt_send(Nack,0)
wait for Packet 0
udt_recv(data, checksum,0) && checkum (ok) rdt_recv(data) udt_send(ack,0)
Wait for Packet 1
udt_send (ack,0) or checksum !ok udt_send (Nack,1)
udt_recv(data, checksum, 1) && checksum ok rdt_recv(data) udt_send(ack,1)

Question 2:
Flow control and congestion control both work by throttling the sender but they do so for very different reasons.

Flow control: each side of a TCP connection has a receive buffer for that connection. Data received through the TCP connection that is in the right order and is uncorrupted is placed in the receive buffer. Without flow control the sender could be sending data far faster than the receiver is reading it which could result in overflowing the connection's receive buffer. Flow control is what prevents/eliminates the risk of overflowing the receiver's buffer. Flow control works by matching the rate of the sender sending data to the rate of the receiving application reading the data. A "receive window" is maintained by the sender as a way for the sender to have a general idea of how much space is available in the receiver's buffer.
The last received byte (the number of the byte most recently received from the network and placed into the buffer) minus the last read byte (the number of the last byte in the data stream

that has been read by the receiver) must be less than (or equal to) the total size of the receive buffer to prevent overflowing the buffer.:
Last byte received - last byte read <= receive buffer

The space available in the buffer is called the receive window and it is equal to the receive buffer minus (last byte received minus last byte read):

Receive window: receive buffer - (last byte received - last byte read)

The size of the receive window is communicated from the receiver to the sender by placing the value of the current receive window into a specific field for the receive window in every segment it sends in response to the sender. Keeping track and sending this value is the responsibility of the receiver. The sender on the other hand needs to keep track of the last byte sent and the last byte acked. The last byte sent minus the last bite acked must be less than the size of the receive window to prevent overflowing.
Even when a host's receive window is at 0, the sender will still send 1 byte segments that will be acknowledged by the receiver. This is so that a host isn't unintentionally blocked and not updated if the receive window changes from 0.
Note: this flow control process is ONLY done in TCP, UDP does not care about overflowing the receive buffer

Congestion control: this deals with congestion on the network layer which is caused by too many sources trying to send data too frequently. Congestion control works by throttling the senders when this happens. In TCP, the network notices it's in trouble and that packets are being dropped based on duplicate acks and timeouts. There are 3 states within TCP congestion control:

1. Slow start: this is a state for when a connection is first being established and we don't know what the network can handle. The congestion window starts at 1 mss (maximum segment size). Each time we get an ack we increase the congestion window by doubling it. This grows exponentially! This will eventually lead to congestion avoidance mode if all goes well or fast recovery if we receive 3 duplicate acks.
2. Congestion avoidance mode: when things seem to be going well we switch to this state. Ideally this is where we want to spend the most time because this is where we can send data the fastest because the window is the biggest. We are in this state when we are near the network limit, so instead of doubling, we will increase the congestion window by 1 mss per round trip time, so this increases linearly. If we get duplicate acks in this state, that indicated dropped packets and we will switch to fast recovery. If there is a timeout, we return to snow start.
3. Fast recovery: in this state we cut the congestion window in half. Once we get a new ack we switch back to congestion avoidance mode. If there is a timeout, we return to slow start

Question 3:

Note: coloring of port text indicated if it is the same port
from A to X behind the NAT:

  Source: 10.0.0.1, some assigned port #*
  Dest: 1.2.3.4 port 80

from B to X behind the NAT

  Source: 10.0.0.2, some assigned port #**
  Dest: 1.2.3.4 port 80

from A to X between the NAT and X

  Source: 5.6.7.8 some assigned port #***
  Dest: 1.2.3.4 port 80

from B to X between the NAT and X

  Source: 5.6.7.8 some assigned port #****
  Dest: 1.2.3.4 port 80

from X to A between X and the NAT

  Source: 1.2.3.4 port 80
  Dest: 5.6.7.8 some assigned port # ***

from X to A between the NAT and A

  Source: 1.2.3.4 port 80
  Dest: 10.0.0.1, some assigned port #*

What there corresponding contents of the router's NAT translation table?

Nat Translation Table			
LAN side		**WAN side**	
Private IP	**Port #**	**Public IP**	**Port #**
10.0.0.1	Randomly assigned	5.6.7.8	Randomly assigned
10.0.0.2	Randomly assigned	5.6.7.8	Randomly assigned

NOTE: the randomly assigned port # is different on the LAN side and WAN side but it does remain constant
Question 4:

Question 4: Routers

How many subnets are a part of this network, and what is the smallest IP prefix (i.e. most fixed bits) that can be used to describe each one?

6 subnets. The Smallest IP prefix that can be used is 3 bytes (24 bits) for each one. The first byte for all of them is 0000 0001. The second byte for all of them is 0000 0001. The third byte varies for each one

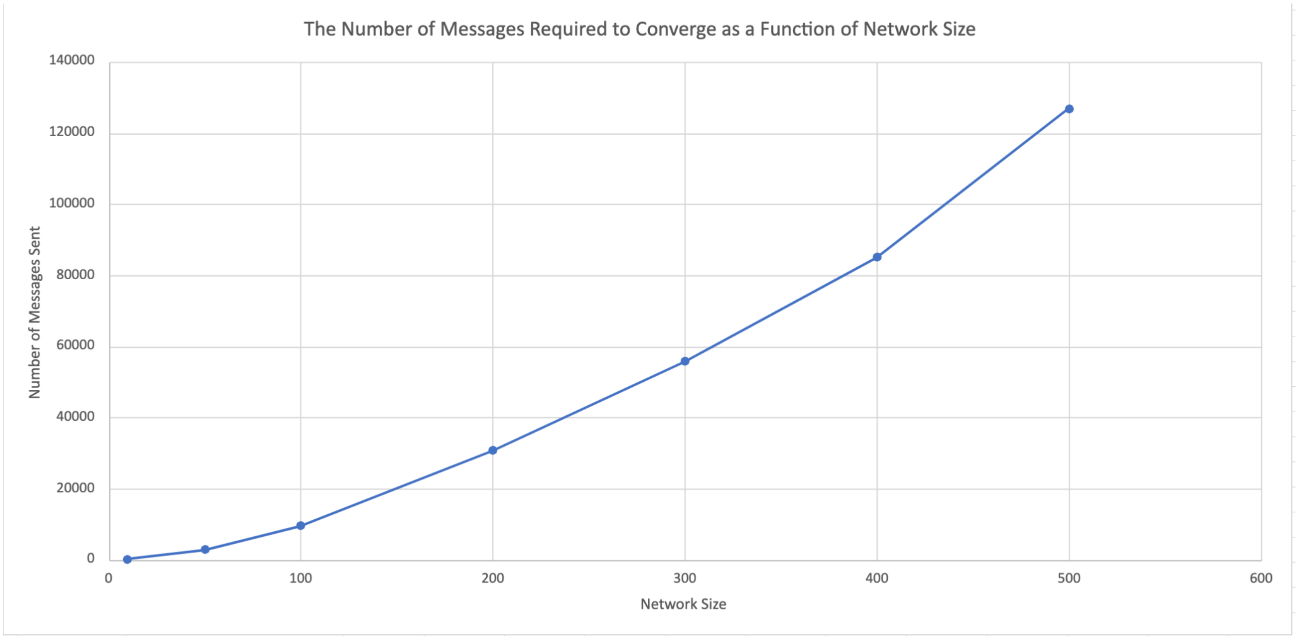
If this network is somehow connected to the internet, what is the cheapest (i.e. smallest number of address) IP prefix the company could have purchased (without using NAT)?

2 bytes + 5 bits AKA:
0000 0001. 0000 0001. 0000 0~

Assume the router for group A has 4 ports: port 1 is connected to the group subnet, port 2 is connected to router B, port 3 is connected to router C, and port D is connected to the ISP. Write out router A's forwarding table.

Router A’s forwarding table	
Prefix	Output link
1.1.1.0/24	Port 1
1.1.2.0/24	Port 2
1.1.3.0/24	Port 3
1.1.4.1 (A-B)	Port 2
1.1.5.1 (A-C)	Port 3
1.1.6.0 (B-C)	Port 2
1.1.6.1 (C-B)	Port 3
ISP (whatever destination is)	Port 4

Question 5: Routing



The Number of Messages Required to Converge as a Function of Network Size
Number of Messages Sent
Network Size
0
20000
40000
60000
80000
100000
120000
140000
0
100
200
300
400
500
600