NETID:________

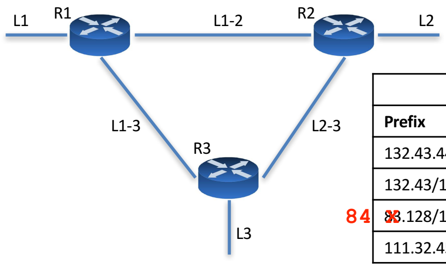
[20pts] 5. Network Routing

Depicted is a tiny part of the Internet with three routers R1, R2, and R3, and six bi-directional links, L1, L2, L3, L1-2, L2-3, L1-3. Also shown are the routing tables of each of the three routers. Each row shows an IP address prefix and the corresponding outgoing link. Recall that /16 stands for a netmask 255.255.0.0 (i.e., the only the first 16 bits are significant), and /24 stands for a netmask 255.255.255.0. Also recall that each IP datagram (aka packet) has a source address, a destination address, and a TTL (Time-To-Live, usually used as a maximum hop count). For simplicity, assume there are four types of IP datagram: TCP, UDP, TIME_EXCEEDED, and UNREACHABLE.

When a datagram arrives at a router, the router first checks if the destination address matches an entry in the routing table. If so and the TTL > 0, then the router decrements the TTL and forwards the datagram to the outgoing link in the entry. If not, there are two cases. If the datagram type is not UDP or TCP, then the router simply drops the datagram. Otherwise the router swaps the source and destination address in the datagram and sets the TTL to 100. It sets the type to UNREACHABLE if there was no match in the routing table, or to TIME_EXCEEDED otherwise. It then treats the datagram as if it had just arrived.

Router R1	
Prefix	**Outgoing Link**
132.43/16	L1-3
84.128/16	L1
84.128.33/24	L1-2

Router R2	
Prefix	**Outgoing Link**
84.128/16	L1-2
132.43/16	L2-3
84.128.33/24	L2-3
132.43.44/24	L2



Router R3	
Prefix	**Outgoing Link**
132.43.44/24	L2-3
132.43/16	L3
84 ~~83~~.128/16	L1-3
111.32.45/24	L2-3

Below we will present several examples of IP datagrams arriving at a particular router. We will give you the router and IP datagram header upon entry at the router. You are to describe the path it will take, and the contents of the datagram "exit" header on the last link it travels in this diagram. In case a router drops the datagram, that would be the contents of the datagram just before it got dropped. For example, if router R1 receives a UDP datagram for destination 111.32.45.68, R1 turns the type into UNREACHABLE and sets the TTL to 100 (because there is no entry for the destination address in the routing table) and forwards the datagram to R3 over link L1-3 (because the destination address is now 132.43.22.33). Finally, R3 forwards the datagram to link L3:

Router R1	**Source**	**Destination**	**TTL**	**TYPE**
Entry:	**132.43.22.33**	**111.32.45.68**	**10**	**UDP**
Path:	R1 $\rightarrow$ L1–3 $\rightarrow$ R3 $\rightarrow$ L3			
Exit:	111.32.45.68	132.43.22.33	98	UNREACHABLE

[5 pts]

Router R2	**Source**	**Destination**	**TTL**	**TYPE**
Entry:	**142.23.33.19**	**132.43.44.2**	**5**	**TCP**
Path:				
Exit:				

[5 pts]

Router R1	**Source**	**Destination**	**TTL**	**TYPE**
Entry:	**142.23.33.19**	**132.43.44.2**	**5**	**UDP**
Path:				
Exit:				

[5 pts]

Router R3	**Source**	**Destination**	**TTL**	**TYPE**
Entry:	**84.128.22.3**	**132.43.44.2**	**1**	**TCP**
Path:				
Exit:				

[5 pts]

Router R1	**Source**	**Destination**	**TTL**	**TYPE**
Entry:	**132.43.44.33**	**84.128.33.100**	**4**	**UDP**
Path:				
Exit:				

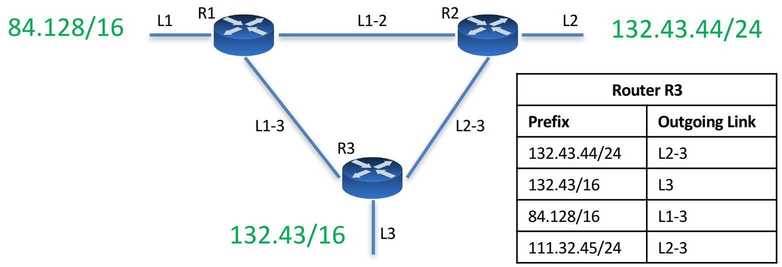
[20pts] 5. Network Routing

Depicted is a tiny part of the Internet with three routers R1, R2, and R3, and six bi-directional links, L1, L2, L3, L1-2, L2-3, L1-3. Also shown are the routing tables of each of the three routers. Each row shows an IP address prefix and the corresponding outgoing link. Recall that /16 stands for a netmask 255.255.0.0 (i.e., the only the first 16 bits are significant), and /24 stands for a netmask 255.255.255.0. Also recall that each IP datagram (aka packet) has a source address, a destination address, and a TTL (Time-To-Live, usually used as a maximum hop count). For simplicity, assume there are four types of IP datagram: TCP, UDP, TIME_EXCEEDED, and UNREACHABLE.

When a datagram arrives at a router, the router first checks if the destination address matches an entry in the routing table. If so and the TTL > 0, then the router decrements the TTL and forwards the datagram to the outgoing link in the entry. If not, there are two cases. If the datagram type is not UDP or TCP, then the router simply drops the datagram. Otherwise the router swaps the source and destination address in the datagram and sets the TTL to 100. It sets the type to UNREACHABLE if there was no match in the routing table, or to TIME_EXCEEDED otherwise. It then treats the datagram as if it had just arrived.

Router R1	
Prefix	**Outgoing Link**
132.43/16	L1-3
84.128/16	L1
84.128.33/24	L1-2

Router R2	
Prefix	**Outgoing Link**
84.128/16	L1-2
132.43/16	L2-3
84.128.33/24	L2-3
132.43.44/24	L2



Router R3	
Prefix	**Outgoing Link**
132.43.44/24	L2-3
132.43/16	L3
84.128/16	L1-3
111.32.45/24	L2-3

Below we will present several examples of IP datagrams arriving at a particular router. We will give you the router and IP datagram header upon entry at the router. You are to describe the path it will take, and the contents of the datagram "exit" header on the last link it travels in this diagram. In case a router drops the datagram, that would be the contents of the datagram just before it got dropped. For example, if router R1 receives a UDP datagram for destination 111.32.45.68, R1 turns the type into UNREACHABLE and sets the TTL to 100 (because there is no entry for the destination address in the routing table) and forwards the datagram to R3 over link L1-3 (because the destination address is now 132.43.22.33). Finally, R3 forwards the datagram to link L3:

Router R1	Source	Destination	TTL	TYPE
Entry:	**132.43.22.33**	**111.32.45.68**	**10**	**UDP**
Path:	R1 ➔ L1−3 ➔ R3 ➔ L3			
Exit:	111.32.45.68	132.43.22.33	98	UNREACHABLE

[5 pts]

Router R2	Source	Destination	TTL	TYPE
Entry:	**142.23.33.19**	**132.43.44.2**	**5**	**TCP**
Path:	R2 ➔ L2			
Exit:	142.23.33.19	132.43.44.2	4	TCP

[5 pts]

Router R1	Source	Destination	TTL	TYPE
Entry:	**142.23.33.19**	**132.43.44.2**	**5**	**UDP**
Path:	R1 ➔ L1-3 ➔ R3 ➔ L2-3 ➔ R2 ➔ L2			
Exit:	142.23.33.19	132.43.44.2	2	UDP

[5 pts]

Router R3	Source	Destination	TTL	TYPE
Entry:	**84.128.22.3**	**132.43.44.2**	**1**	**TCP**
Path:	R3 ➔ L2-3 ➔ R2 ➔ L1-2 ➔ R1 ➔ L1			
Exit:	132.43.44.2	84.128.22.3	98	TIME_EXCEEDED

[5 pts]

Router R1	Source	Destination	TTL	TYPE
Entry:	**132.43.44.33**	**84.128.33.100**	**4**	**UDP**
Path:	R1 ➔ L1-2 ➔ R2 ➔ L2-3 ➔ R3 ➔ L1-3 ➔ R1 ➔ L1-2 ➔ R2 ➔ L2 There was a typo in the original exam and the R3 routing table had an entry for 83.128/16 instead of 84.128/16. In that case, the route is: R1 ➔ L1-2 ➔ R2 ➔ L2-3 ➔ R3 ➔ L2-3 ➔ R2 ➔ L2			
Exit:	84.128.33.100	132.43.44.33	99	TIME_EXCEEDED
ALT. Exit	84.128.33.100	132.43.44.33	98	UNREACHABLE

NETID:______

[20pts] 2. Alternating Bit Protocol

This question is to test your understanding of retransmission protocols such as TCP. Suppose there are two computers, X and Y, connected by a single physical network link. Packets can flow in both directions. Packets can get lost, but they can't get re-ordered or damaged. While unreliable, if one computer keeps retransmitting the same packet (with the same contents), eventually at least one copy will arrive at the other computer. The minimum latency on the link (the time between sending a packet and receiving it) is 1 millisecond and the maximum packet size is 101 bytes. The bandwidth is unlimited. Note that because of the set-up, packets do not need addresses: a packet sent on one end of the link is automatically destined for the other. The length of a packet *p* is given by function length(*p*).

Pat designs an "alternating bit protocol" for reliable communication from X to Y: X and Y both maintain a sequence number that counts the number of packets sent and received, respectively. A packet has two fields: a 1 byte header and a payload of at most 100 bytes. Having only limited size, the header cannot store the entire sequence number. In this case, the 1-byte header stores the sequence number **mod 2**, that is, the header only contains the least significant bit of the sequence number. Packets from X to Y are data packets, and packets from Y to X are acknowledgment packets.

The send function on X is as below:

```
var send_seq initially 0;

fun reliable_send(payload):
  if length(payload) > MTU – 1:
    return ERROR(“payload too large”)

  # Keep trying until an acknowledgment is received
  for ever:
    # Send a data packet
    var data = new Packet()
    data.seq = send_seq mod 2
    data.payload = payload
    link.send(data)

    # Wait for an ack packet with the same sequence
    # number, timing out after 5 seconds. If successful
    # increment send_seq and return SUCCESS.
    var ack = link.receive(5)
    if ack != TIMEOUT:
      if ack.seq == send_seq mod 2:
        send_seq += 1
        return SUCCESS
```

The corresponding receive function on Y is:

```
var recv_seq initially 0;

fun reliable_receive():
  # Keep receiving packets until a packet
  # arrives with the expected sequence number
  for ever:
    # Wait for data packet and prepare ACK
    var data = link.receive(∞)
    var ack = new Packet()
    ack.seq = data.seq
    ack.payload = None

    # If the data packet has the right sequence
    # number, increment recv_seq,
    # send the ack, and return the payload
    if data.seq == recv_seq mod 2:
      recv_seq += 1
      link.send(ack)
      return data.payload

    # send acknowledgment in any case
    link.send(ack)
```

Basically, the sender sends even packets (0, 2, 4, …) with a header containing 0 and odd packets (1, 3, 5, …) with a header containing 1. For each packet, the sender keeps sending the same packet until it gets an acknowledgment with the same bit in the header. The receiver acknowledges all packets it receives. It delivers the first packet with a 0 header, then the first packet with a 1 header, and then it goes back to 0 and so on, alternating between 0 and 1.

Answer the following questions:

a) [3] True or False: It is an invariant that ((*send_seq* == *recv_seq*) or (*send_seq* + 1 == *recv_seq*)).

a) [2] What layer protocol is this: Data Link, Network, Transport, or Application?

c) [3] What is the maximum payload transmission rate in bytes / second from the sender's perspective? (Think about the best case in which no packets get lost.)

Briefly explain (or provide the work for) your answer:

d) [3] True or False: if packets could be re-ordered on the link, the protocol still works.

d) [3] True or False: if the protocol used all 8 bits in the header and used an 8-bit sequence number (0 … 255) instead of a 1-bit sequence number (i.e., replacing **mod** 2 with **mod** 256), the protocol would work even if packets could get arbitrarily re-ordered?

f) [3] Suppose the protocol used an 8-bit sequence number and windows of at most 10 packets so that up to 10 packets could be sent before an acknowledgment was required, what would the maximum payload transmission rate be (in bytes/sec)? (Recall that the bandwidth is unlimited, but the end-to-end latency is not.)

Briefly explain (or provide the work for) your answer:

g) [3] Which of the following statements are consistent with the end-to-end design principle? Check either True or False (no points if you check both):

	True	False
• If the probabilities of packet loss on the links that connect a source and destination are independent of one another, then it is not strictly necessary to implement per-link reliability: end-to-end retransmission is sufficient to provide reliability. For some applications an end-to-end acknowledgment is even necessary, for example in the case of reliable file transfer between hosts that may crash.	☐	☐
• Implementing reliability on intermediate links is useless and one should never do it.	☐	☐
• Implementing reliability on intermediate links induces overhead (for example, buffering for retransmission or computing checksums) even for end-hosts that don't need it.	☐	☐

[20pts] 2. Alternating Bit Protocol

NETID:_______

This question is to test your understanding of retransmission protocols such as TCP. Suppose there are two computers, X and Y, connected by a single physical network link. Packets can flow in both directions. Packets can get lost, but they can't get re-ordered or damaged. While unreliable, if one computer keeps retransmitting the same packet (with the same contents), eventually at least one copy will arrive at the other computer. The minimum latency on the link (the time between sending a packet and receiving it) is 1 millisecond and the maximum packet size is 101 bytes. The bandwidth is unlimited. Note that because of the set-up, packets do not need addresses: a packet sent on one end of the link is automatically destined for the other. The length of a packet *p* is given by function length(*p*).

Pat designs an "alternating bit protocol" for reliable communication from X to Y: X and Y both maintain a sequence number that counts the number of packets sent and received, respectively. A packet has two fields: a 1 byte header and a payload of at most 100 bytes. Having only limited size, the header cannot store the entire sequence number. In this case, the 1-byte header stores the sequence number **mod 2**, that is, the header only contains the least significant bit of the sequence number. Packets from X to Y are data packets, and packets from Y to X are acknowledgement packets.

The send function on X is as below:

```
var send_seq initially 0;

fun reliable_send(payload):
  if length(payload) > MTU – 1:
    return ERROR(“payload too large”)

  # Keep trying until an acknowledgment is received
  for ever:
    # Send a data packet
    var data = new Packet()
    data.seq = send_seq mod 2
    data.payload = payload
    link.send(data)

    # Wait for an ack packet with the same sequence
    # number, timing out after 5 seconds. If successful
    # increment send_seq and return SUCCESS.
    var ack = link.receive(5)
    if ack != TIMEOUT:
      if ack.seq == send_seq mod 2:
        send_seq += 1
        return SUCCESS
```

The corresponding receive function on Y is:

```
var recv_seq initially 0;

fun reliable_receive():
  # Keep receiving packets until a packet
  # arrives with the expected sequence number
  for ever:
    # Wait for data packet and prepare ACK
    var data = link.receive(∞)
    var ack = new Packet()
    ack.seq = data.seq
    ack.payload = None

    # If the data packet has the right sequence
    # number, increment recv_seq,
    # send the ack, and return the payload
    if data.seq == recv_seq mod 2:
      recv_seq += 1
      link.send(ack)
      return data.payload

    # send acknowledgment in any case
    link.send(ack)
```

Basically, the sender sends even packets (0, 2, 4, ...) with a header containing 0 and odd packets (1, 3, 5, ...) with a header containing 1. For each packet, the sender keeps sending the same packet until it gets an acknowledgment with the same bit in the header. The receiver acknowledges all packets it receives. It delivers the first packet with a 0 header, then the first packet with a 1 header, and then it goes back to 0 and so on, alternating between 0 and 1.

Answer the following questions:

a) [3] True or False: It is an invariant that ((*send_seq* == *recv_seq*) or (*send_seq* + 1 == *recv_seq*)).

True

a) [2] What layer protocol is this: Data Link, Network, Transport, or Application?

Transport

c) [3] What is the maximum payload transmission rate in bytes / second from the sender's perspective? (Think about the best case in which no packets get lost.)

50,000

Briefly explain (or provide the work for) your answer:

100 / (2 x .001)

d) [3] True or False: if packets could be re-ordered on the link, the protocol still works.

False

d) [3] True or False: if the protocol used all 8 bits in the header and used an 8-bit sequence number (0 ... 255) instead of a 1-bit sequence number (i.e., replacing **mod** 2 with **mod** 256), the protocol would work even if packets could get arbitrarily re-ordered?

False

f) [3] Suppose the protocol used an 8-bit sequence number and windows of at most 10 packets so that up to 10 packets could be sent before an acknowledgment was required, what would the maximum payload transmission rate be (in bytes/sec)? (Recall that the bandwidth is unlimited, but the end-to-end latency is not.)

500,000

Briefly explain (or provide the work for) your answer:

10 x 100 / (2 x .001)

g) [3] Which of the following statements are consistent with the end-to-end design principle? Check either True or False (no points if you check both):

	True	False
• If the probabilities of packet loss on the links that connect a source and destination are independent of one another, then it is not strictly necessary to implement per-link reliability: end-to-end retransmission is sufficient to provide reliability. For some applications an end-to-end acknowledgment is even necessary, for example in the case of reliable file transfer between hosts that may crash.	✓	
• Implementing reliability on intermediate links is useless and one should never do it.		✓
• Implementing reliability on intermediate links induces overhead (for example, buffering for retransmission or computing checksums) even for end-hosts that don't need it.	✓	