Lower Network Layers

Design the protocol state machines for S and R (both R1 and R2 should use the same protocol).

Use the primitives we discussed in the notes (`udt_send` and `udt_receive`, etc). Don't consider pipelining. The RDT protocol we developed with sequence numbers 0 or 1 and with timeouts is a good starting point.

My Answer:

Since we have two receivers we want to create two variables:

- gotACK0_R1, gotACK0_R2 (because we have two receivers
- gotACK1_R1, gotACK1_R2 we need to declare two)

STATE: This waits for application 0.
EVENT: rdt_send(msg)
ACTION:

- pkt = make_pkt_w_chksum(msg, 0) (use checksum)
- udt_send (pkt)
- gotACK0_R1 = false; gotACK0_R2 = false.
- start_timer()

NEXT: wait for ACK 0

STATE: Wait for ACK 0
EVENT: udt_recv(r_pkt) && !corrupt(r_pkt) && is_ack(r_pkt, 0, R1)
ACTIONS: gotACK0_R1 = true
Next: wait for ACK 0

EVENT: udt_recv(r_pkt) && !corrupt(r_pkt) && is_ack(r_pkt, 0, R2)
ACTIONS: gotACK0_R2 = true
Next: wait for ACK 0

CONDITION: gotACK0_R1 && gotACK0_R2
ACTIONS:

- stop_timer( )
- Then wait for application 1

EVENT: udt_recv(r_pkt) && corrupt(r_pkt)
Actions: ignore wait for timeout
NEXT: stay

EVENT: udt_recv(r_pkt) && !corrupt(r_pkt) && !is_ack(r_pkt, 0, R1) && !is_ack(r_pkt, 0, R2)
Action: Ignore
Next: stay

EVENT: timeout
ACTIONS:

- udt_send(pkt)
- start_timer( )

NEXT: stay in wait for ACK 0

STATE: Wait for application 1
EVENT: rdt_send(msg)
ACTIONS:

- pkt = make_pkt_w_chksum(msg, 1)
- udt_send(pkt)
- gotAKC1_R1 = false; gotACK1_R2 = false;
- start_timer( )

NEXT: wait for ACK1

STATE: Wait for ACK 1
This is the same as ACK 0 but the different here is seq = 1
Then condition to check gotACK1_R1 && gotACK1_R2 to stop timer and go back to wait for application 0.

STATE: Wait for packet 0
ACTIONS:

- **Msg = extract(pkt)**
- **rdt_recv(msg)**
- **udt_send(ack_pkt_0_with_reciver_id)**

Next: wait for packet 1

EVENT: udt_recv(pkt) && corrupt(pkt, 0)
Actions:

- **udt_send(ack_pkt_1_with_receiver_id)**

NEXT: wait for packet 0
EVENT : udt_recv(pkt) && !corrupt(pkt) && has_seq(pkt,1)
ACTION : udt_send(ack_pkt_1_with_receiver_id)
NEXT: stay

STATE : Wait for packet 1
EVENT: udt_recv(pkt) && !corrupt(pkt, 1)
Actions:

- **msg = extract(pkt)**
- **rdt_recv(msg)**
- **udt_send(ack_pkt_1_with_receiver_id)**

Next: Wait for packet 0

Event: udt_recv(pkt) && corrupt(pkt, 1)
Actions: udt_send(ack_pkt_0_with_receiver_id)
Next: stay

Event: udt_recv(pkt) && !corrupt(pkt) && has_seq(pkt,0) (duplicate/out-of-order)
Actions: udt_send(ack_pkt_0_with_receiver_id)
Next: stay

CHecksum detects corruption/collision => corrupt
Seq 0/1 prevents duplicates from being delivered twice.
Timeout, handles lots data or lots collided ACKs
Sender moves to next seq after receiving ACK from R1 and R2

Question 2 (5pts): Throttling

What is the difference between flow control and congestion control? Describe the way TCP implements each of these features.

Control flow : relies on rwnd and window ≤ rwnd

- TCP uses the receive window.
- The way it works receiver has a buffer when incoming data is stored
- This determines how much free space is left in my buffer.
- The sender must not send more data than this amount.
- For example rwnd = 500 bytes window has to be 500 or less.
- Main goal: is preventing receiver overflow

Congestion control: uses cwnd and window ≤ cwnd

- TCP asks if the network is overloaded?
- Using cwnd congestion window
- This is controlled by the sender only based on network condition.
- TCP uses congestion when packet loss, timeout, duplicate ACK, and delay.
- Main goal: prevent network collapse

Describe the way TCP implements each of these features:

- TCP uses both features together
- Does not choose one
- So, TCP uses these features to check what the receiver can handle, and what network can handle it.

Question 3: (8pts) Routers

A company has 3 groups that each hava a subnet on the corporate network. Group A uses subnet 1.1.1.0/24. Group B uses 1.1.2.0/24. Group C uses subnet 1.1.3.0/24.

Each group has a router. There is a link between each pair of routers. A and B have a link: 1.1.4.0 (on A) to 1.1.4.1 (on B) A and C have a link: 1.1.5.0 (on A) to 1.1.5.1 (on C) B and C have a link: 1.1.6.0 (on B) to 1.1.6.1 (on C).

- **What subnet prefix (the smallest one) describes the entire collection of addresses, including the routers as identified in the network where they are linked?**

1) Subnet prefix

1.1.1.0/24
1.1.2.0/24
1.1.3.0/24
1.1.4.0 (A–B link)
1.1.5.0 (A–C link)
1.1.6.0 (B–C link)

1.1.1.0 -> 1 how many pieces of address stay the same we already have /8 /16
1.1.2.0 -> 2 but we want to find a way to combine the third arguments /?
1.1.3.0 -> 3 third number are 1-6 we want to cover 1- 6
1.1.4.0 -> 4 (NEED HELP WITH THIS) **Smallest prefix that covers everything : 1.1.0.0/21**
1.1.5.0 -> 5 because 21 covers from 0-7 this will indicate 1-6
1.1.6.0 -> 6

- **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 4 is connected to the ISP (i.e., the rest of the internet). Write out router A's forwarding table.**

2) Routers A forwarding table

Since souter A is the connection of the three

- Souter A Porst;
  1- Port 1 => Group A 1.1.1.0/24
    - Destination 1.1.1.0/24 group A (direct)
  2- Port 2 => link to Router B (A:1.1.4.0, B:1.1.4.1)
    - 1.1.2.0/24 (group B) destination next hop => 1.1.4.1 router B
  3 - Port 3 => link to Router C (A:1.1.5.0, C:1.1.5.1)
    - 1.1.3.0/24 group C destination next hop => 1.1.5.1 router C
  4 - Port 4 => ISP (default Router)
    - 0.0.0.0/0 default

This is how data will be forwarded in routers.