CS 6014

Wednesday, February 12, 2025 14:21

Assignment 3: Lower Network Layers

- Question 1: Reliable Data Transfer
  - we want to send data from one node to two other nodes using a single broadcast signal
  - specifically we want to design a protocol for reliably sending data from host S to hosts R1 and R2 over this channel
  - channel can disrupt or lose packets
    - packet sent by S received by R1 but not R2
  - Unlike them are collisions on the broadcast channel, you can assume receiving hosts will detect them as corrupt packets
  - If data needs to be resent, you can ignore random backoff etc. and assume colliding hosts will be able to resend their data w/o interference

Design the protocol state machines for S and R (R1 and R2 = same protocol)

- use primitives udt_send udt_receive etc.
- don't consider pipelining
- RDT protocol with sequence numbers 0 and 1 → good starting point

- RDT protocol for single receiver
  - uses sequence numbers (0 and 1)
  - to handle duplicates and acknowledgments
  - to sender sends a packet with a seq. num.
    - waits for an Ack
    - if it doesn't receive it within timeout, it resends
- receiver sends Acks for received packets, ignores duplicates

- Two receivers
  - main challenge: ensure both R1 and R2 receive data correctly
  - since channel is a broadcast
    - when S sends a packet both R1 and R2 can receive it but they might experience different outcomes (one gets it, the other doesn't; both receive corrupted versions)

- How does sender know when to move to the next packet?
  - needs Acks from both R1 and R2
  - How do receivers send Acks w/o causing collisions?
  - Collisions can happen but when resending we can assume they eventually get through
  - So protocol could handle cases where Acks from R1 and R2 maybe collide (leading to corruption), sender would detect them retransmit
    - but we are ignoring random backoffs so sender can retransmit until it gets necessary Acks

- Can receivers coordinate Acks?
  - receivers a separate but have same protocol so maybe each receiver sends Acks independently and sender must collect Acks from both

- Senders state machine needs to track whether both R1 and R2 have ACKed the current packet
- for each data packet, S sends it, then waits for ACK0 (for seq 0) from both R1 and R2
- Once both ACKs are recieved, it can send the next packet with seq 1 and so on

- How does S track which reciever has ACKed which sequence number?
  - maybe we need 2 separate state variables for each receiver
    - For example, for the current sequence number, S needs to know if R1 has ACKed and if R2 has ACKed
      Only when both have ACKed does S increment the sequence number and send the next data

- Approach
  1. Sender S sends packet with sequence number 'seq'
  2. Both R1 and R2, upon receiving the packet correctly (checksum passes) send an ACK with 'seq'
  3. Sender S must receive an ACK from R1 and R2 for the current 'seq' before moving to the next packet
  4. If S doesn't receive ACKs from both within a timeout, it resends the packet
  5. For handling sequence numbers, since were using 0 and 1, after sending a packet with seq 0, S waits for ACK0 from R1 and R2
     Once both are recieved, it sends the next packet with seq 1, and waits for ACK1 from both

- How does S track the ACKs?
  - Channel is a broadcast, so when S sends a packet both R1 and R2 recieve it (if not lost)
  - When S recieves ACK, could be from R1 or R2
  - ACK packets must include identifier

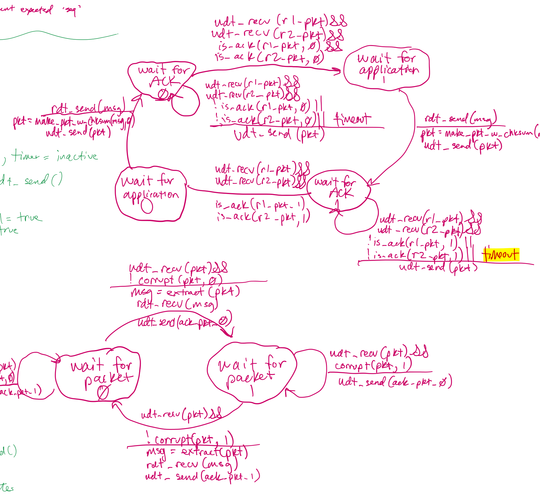
- Sender S
  - S maintains 2 flags: acked_R1 and acked_R2 for current 'seq'
  - When S sends packet, it starts a timer
  - If S recieves ACK(seq) from R1, set acked_R1 = true
  - Same for R2
  - If both are true before timeout, send next packet
  - If one or both = false when timeout occurs, S resend current packet reset flag and restart timer
- Recievers R1 and R2
  - Wait for packet from S
  - When packet is recieved correctly (no corruption), send ACK with own ID and 'seq'
  - If packet is corrupted (detected via checksum or collision), they ignore it
  - Check sequence number to avoid duplicates

- Recievers need to keep track of expected sequence number
- So each reciver (R1 and R2) has their own state for the next expected 'seq'
- recieved packet w/ 'seq' = expected sequence number
  - ↳ deliver data to upper layer, send ACK with their ID and 'seq', increment expected 'seq'
  - ↳ if ≠ (ie duplicate) resend ACK for last expected 'seq'

QUESTION 1 ANSWER

Sender S State Machine

1. States
   - Current sequence number (0 or 1)
   - Acked_R1 (True / False)
   - Acked_R2 (True / False)
   - Timer (Active / Inactive)
2. Transitions
   - Initial State: seq=0, Acked_R1=false, Acked_R2= False, timer = inactive
   - Send Data
     - When data is available, send packet with current sequence via udt_send()
     - Start timer
   - Recieve ACK
     - if ACK is for current sequence and from R1: Set Acked_R1 = true
     - if ACK is for current sequence and from R2: Set Acked_R2 true
     - if both Acked_R1 and Acked_R2 are True:
       - toggle current sequence (0↔1)
       - Reset Acked_R1 and Acked_R2 to false
       - Stop Timer. If more data, send next packet
   - Timeout
     - Resend packet with current sequence
     - Restart timer



Reciever R (R1 and R2) State Machines

1. States
   - Expected Sequence Numbers (0 or 1)
2. Transitions
   - Initial state: expected_seq = 0
   - Recieve packet
     - if packet is corrupt: ignore
     - if packet is valid and sequence # = expected_seq:
       - deliver data to application via deliver_data()
       - Send ACK with (reciever_id, recieved seq) via udt_send()
       - toggle expected_seq (0↔1)
     - if packet is valid but seq ≠ expected seq:
       - Send ACK with (reciever id, received seq) → handles duplicates

QUESTION 2 ANSWER

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

- Flow control prevents sender from overwhelming receiver
  - ↳ by ensuring sender does not transmit data faster than receiver can process it
- Congestion control prevents senders from overwhelming network
  - ↳ by dynamically adjusting transmission rates to avoid congestion and packet loss

- flow control TCP implementation
  - Uses a receiver-advertised window (rwnd) sent in TCP header fields
    - ↳ rwnd updated dynamically to indicated available buffer space
  - Sender limits data transmission to the size of rwnd
    - ↳ so it never exceeds receivers capacity
  - If receiver's buffer fills (rwnd = 0) sender pauses transmission until receiver sends new rwnd > 0 via ACK

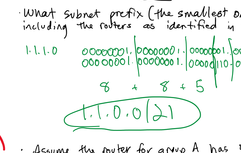
- Congestion Control TCP implementation
  - Uses congestion window (cwnd) to limit amount of unacknowledged data in flight
  - Employs algorithms:
    - Slow Start
    - Congestion Avoidance
    - Fast Retransmit + Recovery
    - Timeout Handling

- TCPs combined Approach
  - Sender's effective window is the minimum of rwnd and cwnd
    - ↳ ensures both receiver capacity and network conditions are respected
  - This dual mechanism allows TCP to balance efficiency with fairness in shared networks

QUESTION 3 ANSWER'S

Company → 3 groups, each have subnet on the corporate network

Groups
- Group A 1.1.1.0/24
- Group B 1.1.2.0/24
- Group C 1.1.3.0/24



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

1.1.1.0 00000001.00000001.00000001.00000000
000 00001 00000001.00000001.00000110.00000001

8 + 8 + 5

1.1.0.0/21

- 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, (ie rest of the internet) Write out router A's forwarding table.

- Port 1: connected to Group A's subnet (1.1.1.0/24)
- Port 2: connected to router B (IP 1.1.4.0 on A's side)
- Port 3: connected to router C (IP 1.1.5.0 on A's side)
- Port 4: connected to ISP (rest of the internet)

Forwarding table based on destination IP addresses
for each destination, router decides which interface to send the packet out

- Group A's subnet (1.1.1.0/24) → send directly out Port 1
- Group B's subnet (1.1.2.0/24) → Port 2, next Hop 1.1.4.1
- Group C's subnet (1.1.3.0/24) → Port 3 → router C's IP 1.1.5.1

Router A's directly connected networks
- Port 1: 1.1.1.0/24 (direct)
- Port 2: 1.1.4.0/30 (direct)
- Port 3: 1.1.5.0/30 (direct)
- Port 4: ISP connection (default route)

other subnets
- 1.1.2.0/24 (Group B) reachable via Port 2, Next hop 1.1.4.1
- 1.1.3.0/24 (Group C) reachable via Port 3 (next hop 1.1.5.1)
- 1.1.6.0/30 (B-C link) reachable via Port 2 (next hop 1.1.4.1)

Router A's Forwarding Table

Destination	Next Hop	Interface	Notes
1.1.1.0/24	Direct	Port 1	Local subnet (Group A)
1.1.4.0/30	Direct	Port 2	direct link to router B (A-B link)
1.1.5.0/30	Direct	Port 3	direct link to router C (A-C link)
1.1.2.0/24	1.1.4.1	Port 2	Group B's subnet via router B
1.1.3.0/24	1.1.5.1	Port 3	Group C's subnet via router C
1.1.6.0/30	1.1.4.1	Port 2	B-C link via router B (shortest path)
0.0.0.0/0	ISP Gateway	Port 4	default route for rest of the internet