Transport Layer

**rdt3.0 sender**

- `sndpkt = make_pkt(0, data, checksum)`
- `udt_send(sndpkt)`
- `start_timer`
- `rdt_send(data)`
- `wait for ACK`
- `rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt, 0)`
- `stop_timer`
- `udt_send(sndpkt)`
- `start_timer` (timeout)

**rdt3.0 in action**

- (a) Synchronous operation
- (b) Lost packet
- (c) Lost ACK
- (d) Premature timeout

**Exercise (Kurose & Ross, 5th ed.)**

- `rdt 3.0 is correct only under a FIFO channel assumption.`
  - Correct = guarantees reliable transmission. Data sent by sender is exactly the data reconstructed in the receiver side.
- `Show a case where a non-FIFO channel (i.e., one that can cause packet reordering) causes rdt 3.0 to deliver incorrect data.`
Exercise (Kurose & Ross, 5th ed.)

• The sender of rdt 3.0 simply ignores all received packets that are either in error or have the wrong value in the acknum field of an ack packet.
• Suppose that in such circumstances, rdt 3.0 were simply to transmit the current data packet.
• Would the protocol still work?
• Would it be more or less efficient than before?

Exercise (Kurose & Ross, 5th ed.)

• Would the protocol still work?
  ▫ Yes. A retransmission is exactly what would happen if the sender’s timeout expired (for instance, because an ack was completely lost instead of garbled).
  ▫ The receiver can’t even distinguish between the two events.

Exercise (Kurose & Ross, 5th ed.)

• Would it be more or less efficient than before?
  ▫ Depends on the length of the sender timeout, compared to the expected channel delay.
  ▫ If the timeout is very long, then the immediate retransmit can save us the long wait until the timeout expires.
  ▫ However, premature timeouts can cause a pathologies.

Exercise (Kurose & Ross, 5th ed.)

• Would it be more or less efficient than before?
  ▫ We will show a scenario in which one premature timeout causes duplication of all the packets in the session from a certain time point.
  ▫ This is the “Sorcerer’s Apprentice Syndrome”

Performance of rdt3.0

• rdt 3.0 works, but performance stinks
• ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:
  \[ d_{\text{trans}} = \frac{L}{R} = \frac{8000\text{bits}}{10^7\text{bps}} = 8\text{microseconds} \]
  \[ U_{\text{sender}} = \frac{L / R}{\text{RTT} + L / R} = \frac{0.008}{30.008} = 0.00027 \]
  \[ \text{if RTT}=30 \text{msec}, 1KB pkt every 30 msec \rightarrow 33kB/sec throughput over 1 Gbps link \]
  • network protocol limits use of physical resources!
**RDT 3.0: stop-and-wait operation**

- First packet bit transmitted, \( t = 0 \)
- Last packet bit transmitted, \( t = L / R \)
- First packet bit arrives, \( t = RTT \)
- Last packet bit arrives, send ACK

\[
U_{\text{sender}} = \frac{L}{R} \cdot \frac{RTT + L}{L} = 0.00027
\]

**Pipelining**

- Sender allows multiple, "in-flight", yet-to-be-acknowledged packets
- Range of sequence numbers must be increased
- Buffering at sender and/or receiver

\[
U_{\text{sender}} = \frac{3 \cdot L}{R} \cdot \frac{RTT + L}{L} = 0.0008
\]

**Pipelined Protocols**

- Two generic forms of pipelined protocols: **Go-Back-N**, **Selective Repeat**

---

**Go-Back-N**

- **Sender:**
  - \( k \)-bit seq # in pkt header
  - "window" of up to \( N \), consecutive unack'ed pkts allowed

  \[
  \text{send}_{\text{base}} \quad \text{nextseqnum}
  \]

- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
- May receive duplicate ACKs (see receiver)
- Timer for each in-flight pkt
- Timeout(n): retransmit pkt n and all higher seq # pkts in window

**Selective Repeat**

- **Sender:**
  - Can have up to \( N \) unack'ed pkts in pipeline
  - Receiver sends individual ack for each packet
  - Sender maintains timer for each unack'ed packet
  - When timer expires, retransmit only unack'ed packet

---

**GBN: sender extended FSM**

- `rdt_send(data)`
  - If `nextrnum < getacknum()`:
    - `sndpkt[nextrnum] = make_pkt(nextrnum,data,checksum)`
    - `udt_send(sndpkt[nextrnum])`
    - If `nextrnum == base`
      - `start_timer`
      - `nextrnum++`
  - Else
    - `refuse_data(data)`
    - `base = getacknum()`
    - `nextseqnum = 1`

- `rdt_rcv(rcvpkt)`
  - If `corrupt(rcvpkt)`
    - `base = getacknum()`
    - `maxnum = nextrnum+1`
    - `if base <= maxnum`
      - `start_timer`
    - `else`
      - `start_timer`

---

**Transport Layer**

- 3-13
- 3-14
- 3-15
- 3-16
- 3-17
- 3-18
GBN: receiver extended FSM

ACK-only: always send ACK for correctly-received pkt with highest in-order seq #

- may generate duplicate ACKs
- need only remember expectedseqnum

out-of-order pkt:
- discard (don’t buffer) → no receiver buffering!
- Re-ACK pkt with highest in-order seq #

Selective Repeat

- receiver individually acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
- sender timer for each unACKed pkt
- sender window
  - N consecutive seq #'s
  - again limits seq #'s of sent, unACK'ed pkts

Selective repeat in action

Selective repeat: sender, receiver windows

sender

data from above:
- if next available seq # in window, send pkt
timeout(n):
- resend pkt n, restart timer
ACK(n) in [sendbase,sendbase+N):
- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt
pkt n in [rcvbase-N,rcvbase-1):
- ACK(n)
- otherwise:
  - ignore

receiver

pkt n in [rcvbase, rcvbase+N-1]
- send ACK(n)
- out-of-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt
pkt n in [rcvbase-N,rcvbase-1]
- ACK(n)
otherwise:
- ignore
Selective repeat: dilemma

Example:
- seq #'s: 0, 1, 2, 3
- window size = 3
- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?

Minimal sequence range

- Assume we want to use a sender window of size N.
- What is the minimal number of unique sequence numbers we should allow to prevent such errors?
- The cyclic sequence number should never cause the sender and receiver's window to ambiguously overlap
- In FIFO channels:
  - GBN: N + 1
  - SR: 2N
  - Proof: on-board

Exercise (Kurose & Ross, 5th ed.)

- Are the following statements true or false?
  - With SR, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.
  - True. Suppose sender has a window size of 3.
    - Time t₀: It sends packets 1, 2, 3.
    - Time t₁ > t₀: Receiver sends packet 1, 2, 3.
    - Time t₂ > t₁: Sender times out and retransmits 1, 2, 3.
    - Time t₃ > t₂: Receiver sends the ack sent at t₁, advances its window to 4, 5, 6.
    - Time t₄ > t₃: Receiver receives the acks sent at t₁, that fall outside of its current window.
  - With GBN, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.
  - True, with the same scenario as described above. Only need to replace the selective acks with cumulative acks.

Exercise (Kurose & Ross, 5th ed.)

- Recall the GBN receiver: assume it is waiting for packet m (i.e., it received correctly all the packets up to m − 1 inclusive).
  - When a data packet with sequence n = m is received, the receiver accepts it and advances its window.
  - Whenever a data packet with sequence n = m is received, the receiver discards it and resends ack m (“I am still waiting for m”).
  - Assume a FIFO channel and an infinite sequence number. Does the protocol remain correct if we perform the following changes?
    - If n < m the receiver discards the packet and does not send an ack.
      - Incorrect. Let the sender send packets 1, ..., m — 1. All received correctly, but all acks are lost.
      - The receiver waits for packet m.
      - But whenever the sender times-out expires, it resends packets 1, ..., m — 1.
      - Receiver discards them and does not ack.
      - Deadlock.

Exercise (Kurose & Ross, 5th ed.)

- Are the following statements true or false?
  - rdt 3.0 is the same as SR with a sender and receiver window size of 1.
    - Both are true. With a window size of 1, SR, GBN, and the rtd 3.0 are functionally equivalent.
  - rdt 3.0 is the same as GBN with a sender and receiver window size of 1.
  - Both are true. With a window size of 1, SR, GBN, and the rtd 3.0 are functionally equivalent.
    - The window size of 1 precludes the possibility of out-of-order packets (within the window).
    - A cumulative ACK is just an ordinary ACK in this situation, since it can only refer to the single packet within the window.
Exercise

- If $n > m$, the receiver discards the packet and does not send an ack. Otherwise, operate as before.
- Correct. If $n > m$ was received, but the receiver is waiting for $m$, it means we have a gap. The sender will eventually timeout for $m$, and resend packet $n$ then.