#4: Transport Layer. Reliable transmission

- resides on end hosts
- handles process-to-process communication

two internet transport layer protocol: TCP and UDP

Brief Responsibilities

Sender:
- breaks message into segments
- forwards to network layer

Receiver
- reassembles message
- pass to app layer

Router
- check dest IP address
- decide routing

application layer "message" → transport layer "segment"
- adds src and dest port

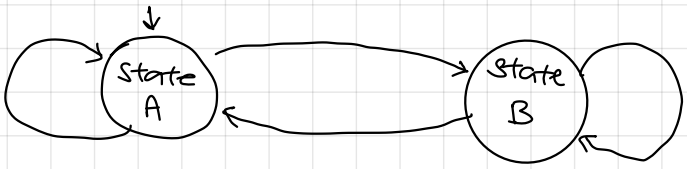
→ network layer "datagram"
- adds src and dest IP address

In transport layer, we are required to build a reliable communication service over a unreliable channel (network layer).

Network may:

- corrupt packets
- drop packets
- re-order packets
- deliver packets after too long

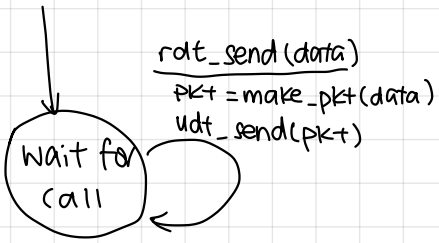
Finite State Machine



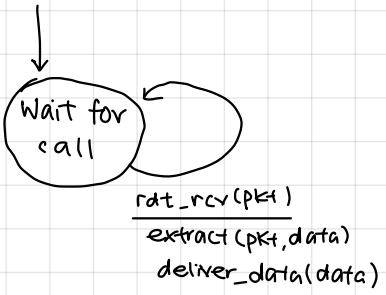
Event causing transition
Action taken on transition

rdt 1.0

Sender



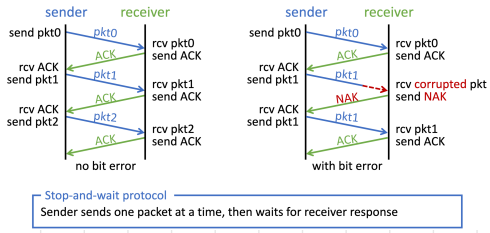
Receiver



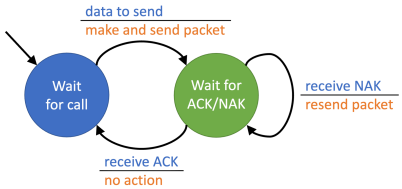
rdt 2.0: Channel may flip bits in packet

- use checksum
- use ACKs and NAKs to tell sender that packet has errors

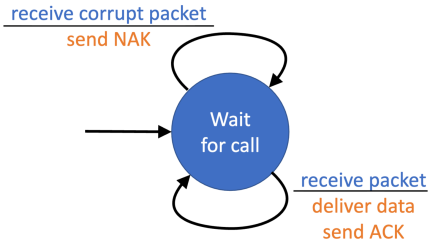
rdt 2.0 in action



rdt 2.0 sender



rdt 2.0 receiver

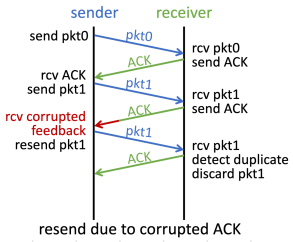


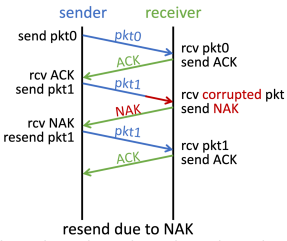
- rdt 2.0 also assumes stop-and-wait protocol
- key problem with rdt 2.0 is that if ACK/NAK is corrupted, the sender would not know

Solution: retransmit upon receiving corrupted ACK/NAK?
- NO!
- may cause retransmission of properly received data

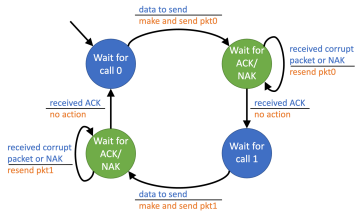
rdt 2.1 : Add a sequence number

- sender adds seq # to current packet
- Receiver discards duplicate packet

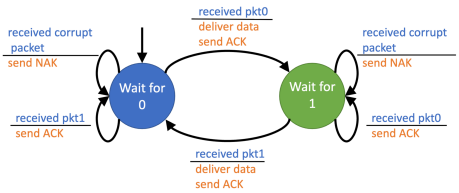




rdt 2.1 sender



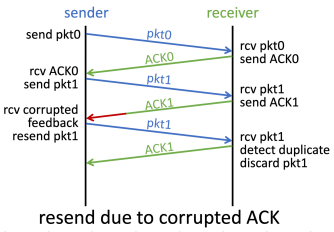
rdt 2.1 receiver



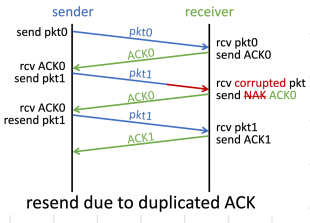
rdt 2.2 : a NAK-free protocol

- slight optimization of rdt 2.1
- return ACK + serial number of the last received sequence number

rdt 2.2 in action



rdt 2.2 in action



FSM Diagram would be similar to

※ up to now, we have not seen the case where data can be lost / arbitrarily long packet delay

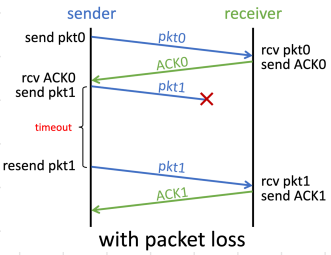
rdt 3.0 : Handling packet loss

Assumption : network will not reorder packets

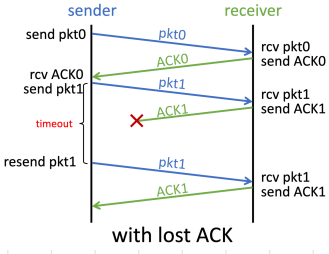
- wait a reasonable amount of time for ACK
- sender retransmit if no ACK is received till timeout

diff from rdt 2.0 is that you don't resend packet upon receiving corrupt packet

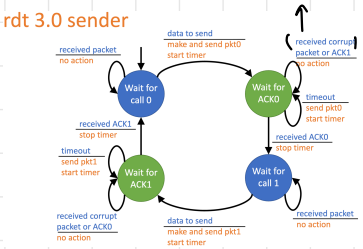
rdt 3.0 in action



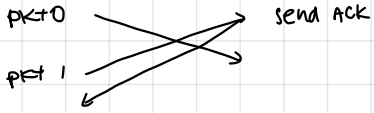
rdt 3.0 in action



rdt 3.0 sender

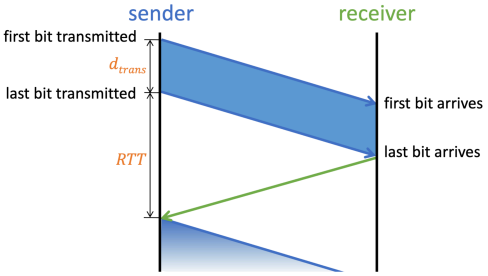


FSM Diagram for receiver is same as that for rdt 2.2



Problem : Link is underutilized

Problem with Stop-and-wait



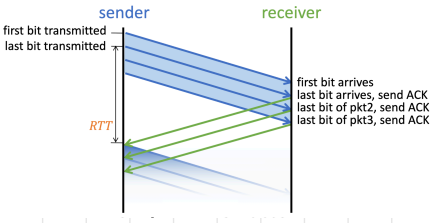
$$U_{sender} = \frac{Time\ sending}{Total\ time} = \frac{d_{trans}}{d_{trans} + RTT}$$

utilization is $\frac{\text{time spent sending}}{\text{Total time}}$

Solution : allow for pipelining

- need to increase the range of sequence number
- buffering at sender and/or receiver

Pipelining



Two forms of pipelined protocols

- same assumptions as rdt 3.0

Go-Back-N

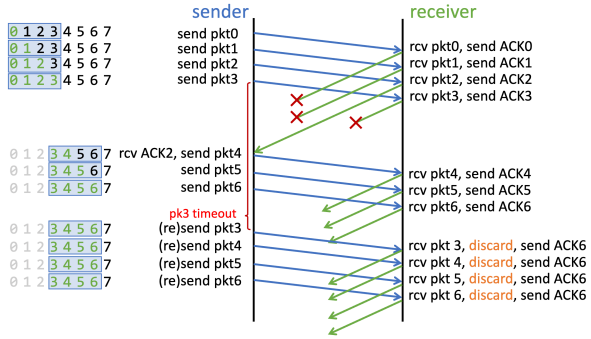
Sender:

- can have up to N unACKed packets in pipeline
- insert k-bits sequence # in packet header
- use a "sliding window" to keep track of unACKed packets
- keep a timer for the oldest unACKed packet
- timeout(n): retransmit packet n and all subsequent packets in the window

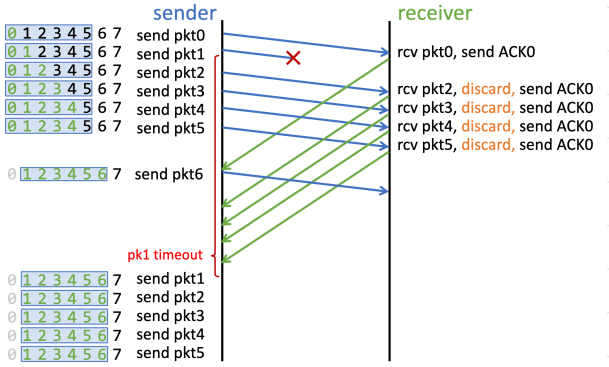
GBN Receiver:

- only ACK packets that arrive in order
  - ↳ need remember expectedSeqNum
- discard out-of-order packets and ACK the last in-order seq. #
  - → "cumulative ACK" "ACK m" means all packets up to m are received

Go-back-N in action



Go-back-N in action



properties: max window size = $2^n - 1$

- where n is the # of bits in the sequence # field
- e.g. seq: 0 1 2 3 → n = 4

  max window size = 3

  ↳ no wrap around, receiver knows if there is duplicate

Selective Repeat

Receiver:

- individually acknowledges all correctly received packets
  - ↳ buffer out-of-order packet, as needed, for eventual in-order delivery to application layer

Sender:

- sender maintains timer for each unACKed packet
  - ↳ when timer expires, retransmit only that unACKed packet.

Max window size is $\frac{2^n}{2} = 2^{n-1}$ ensures that there is no corruption

e.g. sender sends 0 1 2

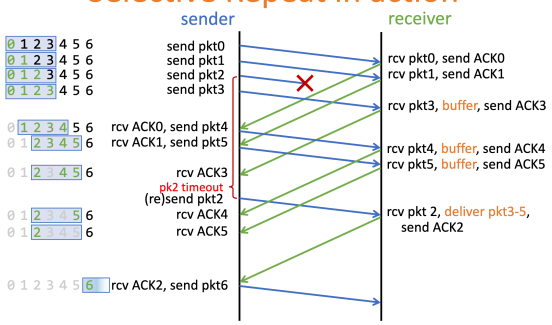
receiver sends ACK #0 #1 #2 ↗ lost

new window #1 #2 #0

sender sends #0 #1 #2 again

↳ receiver would think #1 and #2 are new data

Selective Repeat in action



Selective Repeat: Behaviors

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]
- mark pkt n as received
- if n is smallest unACKed pkt, advance window base to next unACKed seq. #

Receiver

pkt n in [rcvbase, rcvbase+N-1]
- 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