Reliable Data Transfer

Kai Shen

Reliable Data Transfer

- What is reliable data transfer?
  - guaranteed arrival
  - no error
  - in order delivery
- Why is it difficult?
  - unreliable underlying communication channel, which can be lossy, error-prone, and deliver packets out of order
- Where is it used in computer networks?
  - reliable transport service on top of unreliable network layer
  - reliable data link service on top of unreliable physical layer

Principles of Reliable Data Transfer

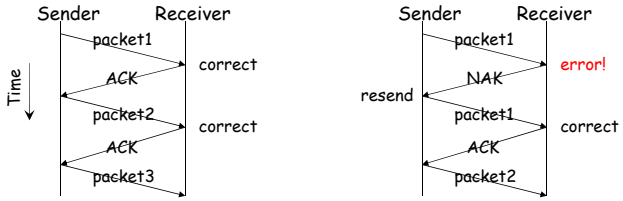
- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol
  - e.g., delay in the channel is bounded in physical layer, not so for network layer
- Other services may interact with RDT protocol
  - e.g., flow control, congestion control
- Here we study widely applicable RDT principles
  - we don't make assumptions about the unreliable channel
  - we don't consider interaction with other services
- Later we see what RDT is like in practice
  - in a transport layer protocol – TCP

Outline

- Overview of reliable data transfer
- A correct protocol: stop-and-wait
  - one packet at a time
- An efficient protocol: sliding window
  - multiple packets simultaneously

Deal with Errors

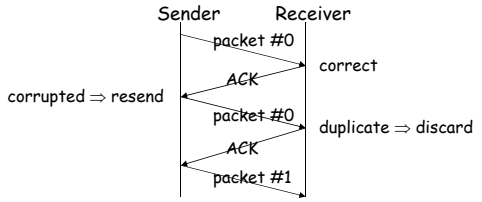
- First deal with errors, later deal with packet loss.
- ACK-based solution: receiver check errors
  - if correct, send back positive ACK
  - otherwise, send back negative NAK



What if ACK or NAK is corrupted?

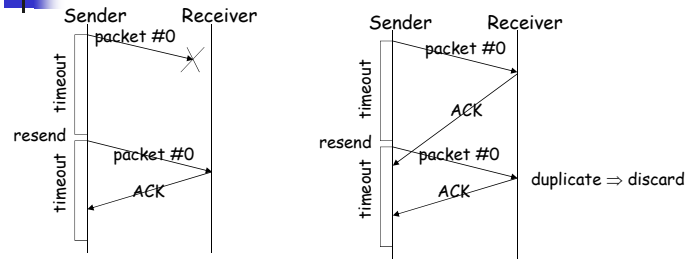
What if ACK or NAK is corrupted?

- Solution 1: creating special acknowledgments for ACKs/NAKs. What if they get corrupted too??
- Solution 2: treat corrupted acknowledgements as NAKs. Duplicated packets!!
- To solve duplicated packets: sequence number for each packet.



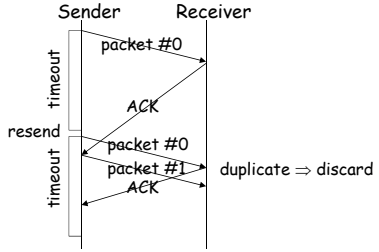
How many sequence numbers do we need here?

Deal with Packet Loss: Timeouts



- Early timeout ⇒ duplicated packet ⇒ sequence number. (not likely for data link protocol)

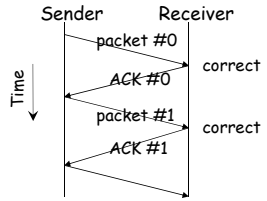
Deal with Duplicated ACKs



- Solution: each ACK carries sequence number.
- With timeout, NAK is not necessary any more.

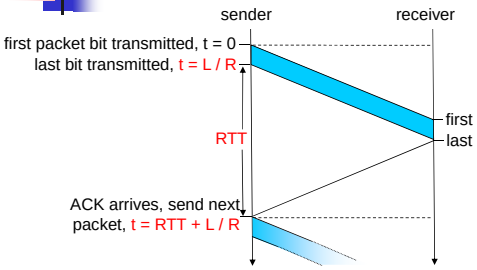
Stop-and-Wait

Now we have a *correct* protocol:



- Allow one outstanding (un-ACKed) packet – *stop-and-wait*
- By the way, we haven't talked about in-order delivery.

Efficiency of Stop-and-Wait



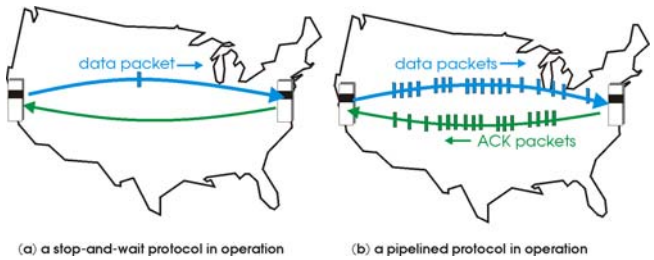
$$\text{efficiency} = \frac{L / R}{RTT + L / R}$$

Example:
- Packet size L = 1KB (8kbits),
- Transmission speed R = 1 Gbps,
- Roundtrip prop. delay RTT = 30ms.

⇨ 0.027% efficiency!

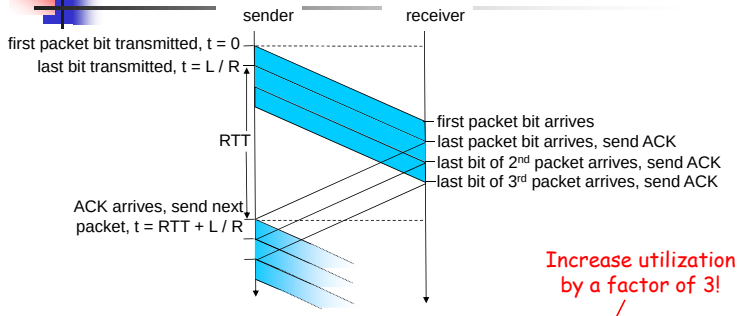
Pipelined Protocols

Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged packets



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Pipelining: Increased Efficiency

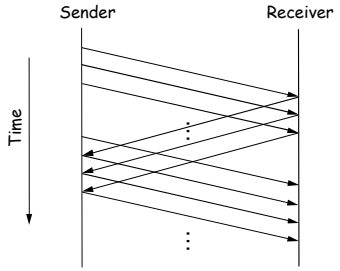


Increase utilization by a factor of 3!

$$\text{Channel util efficiency} = \frac{3*L / R}{RTT + L / R} = \frac{.000024}{.030008} = 0.08\%$$

A pipelined protocol: Sliding Window

- Allow multiple outstanding (un-ACKed) packets
- Upper bound on un-ACKed packets, called window



Two variations: go-back-N, and selective repeat.

Go-Back-N: normal operation

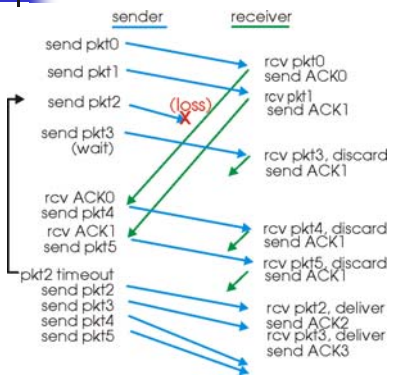
- Sender: "window" of up to N consecutive un-ACKed packets allowed; limit send buffer space

- Receiver: no buffering
- Cumulative ACK – ACK with seq #n stands for ACKs all packets up to, including seq #n
  - Receiver: acknowledge in-order packet arrival
  - Sender: if recv ACKs in send window, sliding send window

Go-Back-N: deal with problems

- Sender:
  - Timer for each in-flight packet (or first in-flight packet)
  - Packet with seq #n timeouts: retransmit #n and all higher seq # packets in window (buffering)
- Receiver: out-of-order packet:
  - Discard!
  - Optional: Re-ACK packet with highest in-order seq # (sort of a NACK)
    - alert sender something is wrong through duplicated ACKs
    - not critical for protocol correctness; but may improve performance

GBN in Action



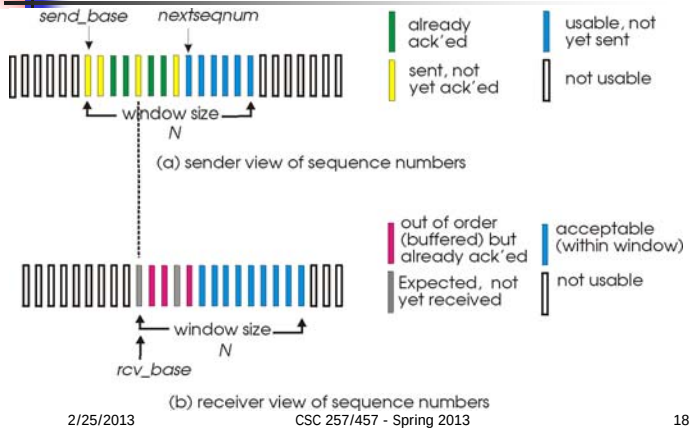
Drawback:
- Resend out-of-order packets

To fix it:
- Receiver buffering
- Selective acknowledgement

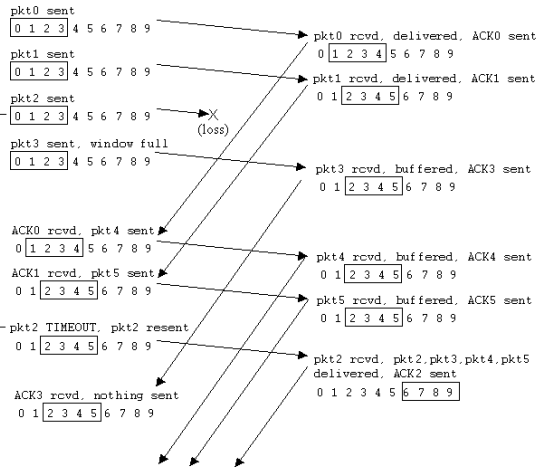
Selective Repeat

- Receiver
  - buffers out-of-order packets for eventual in-order delivery to upper layer
  - individually acknowledges all correctly received packets
- Sender
  - maintains timer for each un-ACKed packet
  - only resends packets whose timers expire before ACKs are received

Selective Repeat: Sender, Receiver Windows



Selective Repeat in Action



Selective Repeat: Sender Implementation

Sender

data from above:
- if there is available slot in window, send pkt

timeout(n):
- resend pkt n, restart timer

ACK(n):
- mark pkt n as received
- if n is smallest unACKed pkt, advance window base to next unACKed seq # (sliding!)

Selective Repeat: Sequence Numbers at Receiver

- rcvbase is the first expected packet.
- Is it possible to see an arriving packet with sequence number of rcvbase+N or greater?
  - No since rcvbase is still in send window which can't go beyond rcvbase+N-1.
- Is it possible to see an arriving packet with sequence number smaller than rcvbase?
  - Yes. For instance, due to lost acknowledgement.
- What is the smallest seq number receiver can possibly see?
  - All the way down to the first packet since a packet can hang inside the network for unbounded time.
- What is the smallest seq number that may still in the send window?
  - rcvbase-N.

Selective Repeat: Receiver Implementation

Receiver

pkt n in [rcvbase, rcvbase+N-1]
- send ACK(n)
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt (sliding!)
- out-of-order: buffer

pkt n in [rcvbase-N, rcvbase-1]
- ACK(n)

otherwise:
- ignore

Sequence Numbers

- Bounding of the currently relevant sequence number space?
  - We bound it at the first step of stop-and-wait when there is no packet loss and every packet is accounted for before the protocol moves on.
  - Cannot be bounded in practice when earlier sent packet (assumed lost by the protocol) may hang around in the network for a long time and then arrives suddenly.
- External (beyond the current communication session) reasons for out-of-bound sequence numbers:
  - Packets belonging to earlier sessions between the same communication hosts
  - Malicious attacker injects packets

Disclaimer

- Parts of the lecture slides contain original work of James Kurose, Larry Peterson, and Keith Ross. The slides are intended for the sole purpose of instruction of computer networks at the University of Rochester. All copyrighted materials belong to their original owner(s).