Reliable Data Transfer

Kai Shen

Dept. of Computer Science, University of Rochester

Reliable Data Transfer

- What is reliable data transfer?
  - guaranteed arrival
  - no error
  - in order delivery
- Why is it difficult?
  - end-system-only solution has no control of 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 IP
  - reliable data link service on top of unreliable physical layer

Principles of Reliable Data Transfer

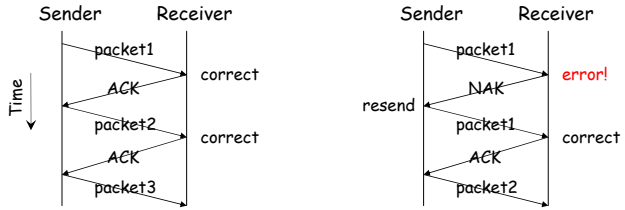
- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol
  - e.g., delay in the channel is bounded in physical layer, not so for network layer
- Other services may interact with RDT protocol
  - e.g., flow control, congestion control
- Here we study the RDT principles, so
  - 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

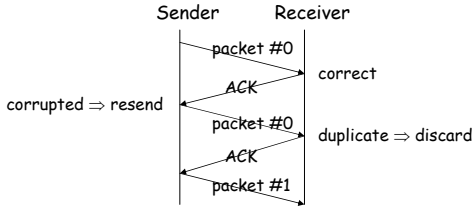
- First deal with errors, later deal with packet loss.
- ACK-based solution: receiver check errors
  - if correct, send back positive ACK
  - otherwise, send back negative NAK



What if ACK or NAK is corrupted?

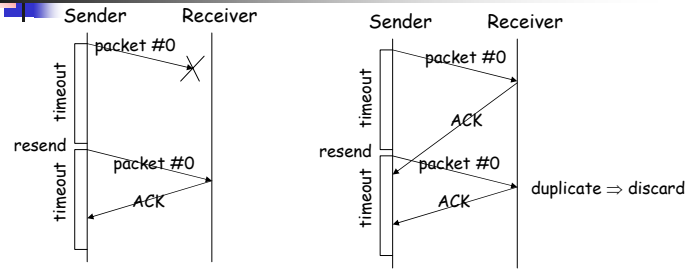
What if ACK or NAK is corrupted?

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



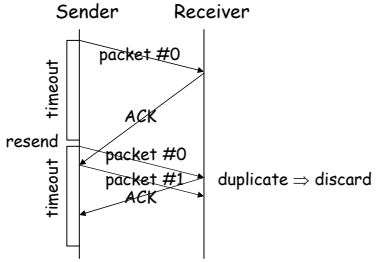
How many sequence numbers do we need here?

Deal with Packet Loss: Timeouts



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

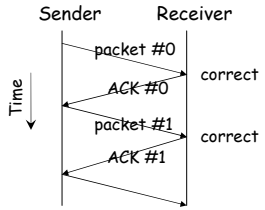
Deal with Duplicated ACKs



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

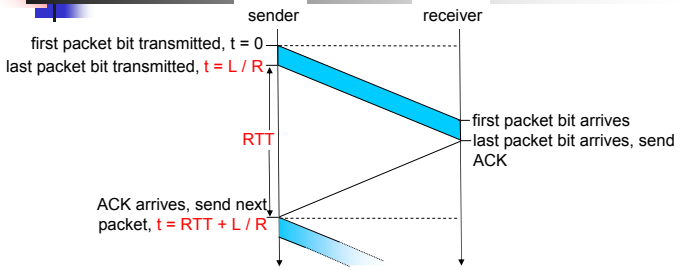
Stop-and-Wait

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



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

Efficiency of Stop-and-Wait (An Example)

- It works, but performance stinks!
- Example: 1 Gbps link, 30ms roundtrip prop. delay, 1KB (8kbits) packet.

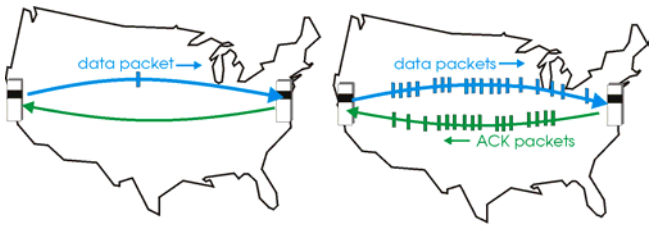
$$T_{transmit} = \frac{L \text{ (packet length)}}{R \text{ (transmission rate)}} = \frac{8 \text{ kb}}{1 \text{ gb/sec}} = 8 \text{ microsec}$$

$$\text{Throughput} = \frac{L}{T_{transmit} + RTT} = \frac{8kb}{0.03008sec} = 266Kbps$$

- 266Kps throughput over 1 Gbps link ⇒ 0.027% efficiency!
- network protocol limits use of physical resources!

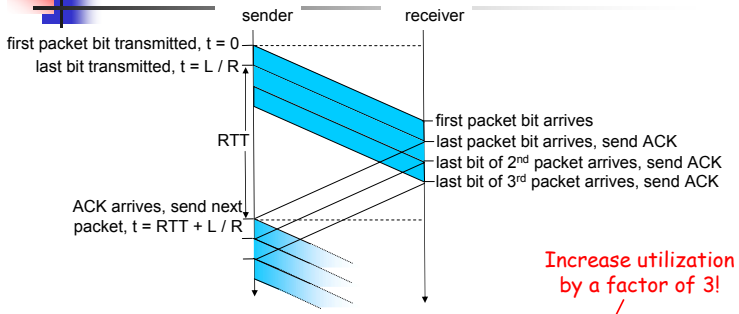
Pipelined Protocols

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

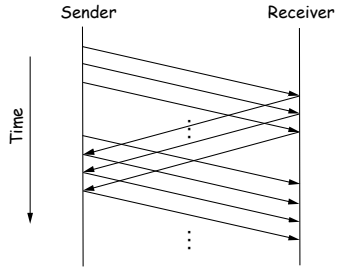


Increase utilization by a factor of 3!

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

A pipelined protocol: Sliding Window

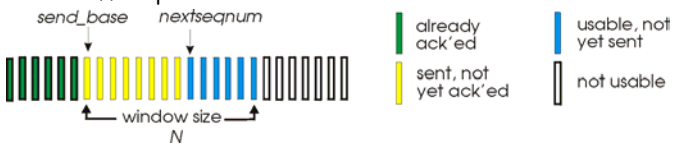
- Allow multiple outstanding (un-ACKed) packets
- Upper bound on un-ACKed packets, called window



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

Go-Back-N: Sender

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

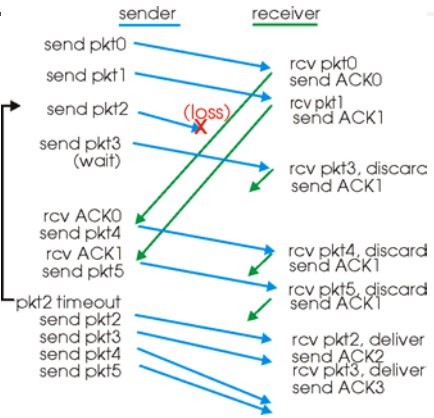


- cumulative ACK – ACK with seq #n stands for ACKs all packets up to, including seq #n
- recv ACKs in send window
  - sliding send window
- timer for each in-flight packet
- packet with seq #n timeouts:
  - retransmit #n and all higher seq # packets in window (buffering)

Go-Back-N: Receiver

- Send ACK for correctly-received packet with highest in-order seq #
  - Sliding receive window
- Out-of-order packet:
  - Discard (don't buffer) -> no receiver buffering!
  - 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

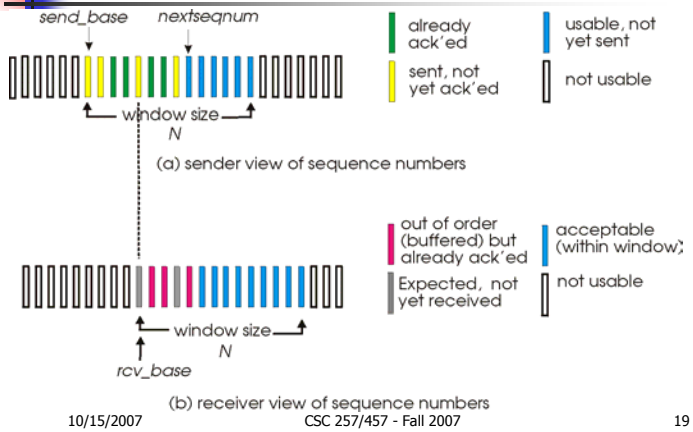
GBN in Action



Selective Repeat

- receiver individually acknowledges all correctly received packets.
  - buffer out-of-order packets for eventual in-order delivery to upper layer
- sender timer for each un-ACKed packet
  - sender only resends packets whose timers expire before ACKs are received

Selective Repeat: Sender, Receiver Windows



Selective Repeat

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!)

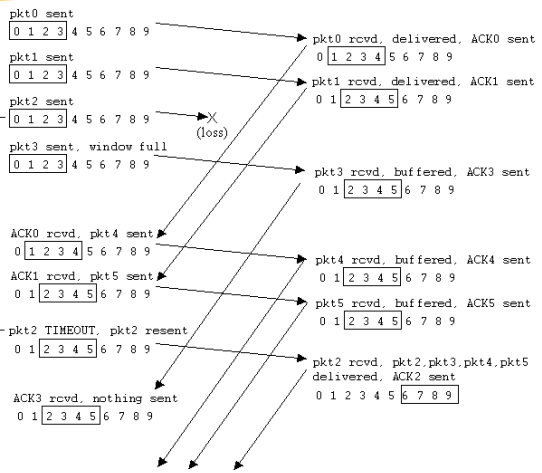
receiver

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 (sliding!)

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

otherwise:
- ignore

Selective Repeat in Action



Sequence Number Space

- **SeqNum** field is finite; sequence numbers wrap around
  - 0, 1, …, S-1, 0, 1, …, S-1, 0, 1, …
- Sequence number space must be at least as big as the window size ⇒ S ≥ N
- Is this enough?
  - out-of-order channel: an old packet carrying the same sequence number confused with a packet of current concern

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).