EECS 122, Lecture 19

Today's Topics:

More on Reliable Delivery

Round-Trip Timing

Flow Control

Intro to Congestion Control

Kevin Fall, kfall@cs.berkeley.edu

Reliable Delivery

- Stop and Wait
  - simple ARQ scheme, bad performance
  - degrades with increasing RTT
  - poor performance derives from not *filling the pipe*
- How to fill the pipe?
  - Recall the *bandwidth-delay product* is a measure of the bit storage capacity of a path
  - so, if we can keep a bw-delay product's worth of data in network, we fully utilize it

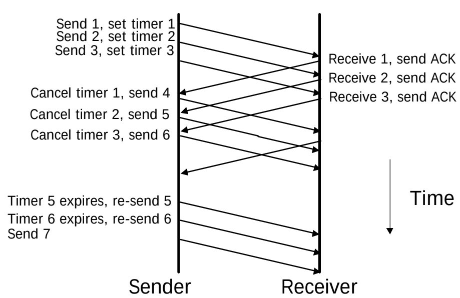
An Example

- Imagine a long-distance T1 line:
  - bandwidth: 1.544Mb/s, RTT about 45ms
  - bw-delay product about 70Kb (8700 bytes)
- Assuming Stop&Wait w/frame size 1KB:
  - performance is ~ (1frame)/(1 RTT) =
  - 1KB/(0.045s) = 182Kb/s
  - limits performance to about 1/8 of link

Improving over Stop & Wait

- Want a way to fill up the bandwidth-delay product of the path
- Extend S&W with the ability to introduce >1 packet into the network before receiving an ACK
- Go-back-n, also called Sliding Window
  - introduce a window of size n
  - can inject n packets into net before hearing an ACK

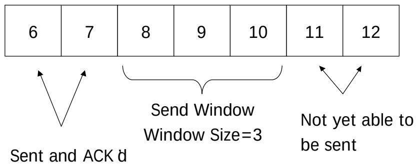
Picture of Go-back-n/Sliding Window



Send Window Maintenance

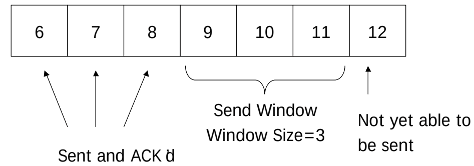
- So, how does the sender keep track of what to send?
- Sliding windows:
  - label each packet with a *sequence number*
  - a *window* is a collection of adjacent sequence numbers
  - the size of the collection is the *sender's window size*

Example (send window, w=3)



In-window packets are sent but not yet ACK'd, also called *unacknowledged* packets

Example (recv ACK for 8)

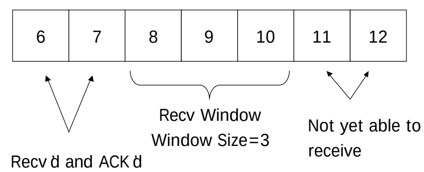


Window has "slid forward", allowing 11 to be sent, and 8 to be retired

Receive Window Maintenance

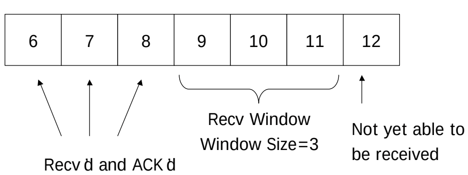
- Receiver keeps a similar window
- Why?
  - Receiver has a finite buffer
  - left window edge is first packet receiver wants to see
  - right edge is last packet it can hold
  - packets < left edge or > right edge dropped
  - other (good) packets are queued, allowing for fixing up out-of-order packets

Example (recv window, w=3)



In-window packets may be received, even out of order

Example (recv packet 8)



Window has "slid forward", allowing 11 to be received, and implying an ACK generation for 8

Some Observations

- With sliding windows, it is possible to fully utilize a link, provided the window size is large enough. Throughput is ~ (w/RTT); Stop & Wait is like w = 1.
- Sender has to buffer all unacknowledged packets, because they may require retransmission
- Receiver may be able to accept out-of-order packets, but only up to its buffer limits

Retransmissions

- So, the sender needs to set timers in order to know when to retransmit a packet the may have been lost
- How long to set the timer for?
  - Too short: may retransmit before data or ACK has arrived, creating duplicates
  - Too long: if a packet is lost, will take a long time to recover (inefficient)

Retransmission Timer

- The amount of time the sender should wait is about the round-trip time (RTT) between the sender and receiver
- For link-layer networks (LANs), this value is essentially known
- For multi-hop WANS, rarely known
- Must work in both environments, so protocol should adapt to the path behavior

Adaptive Retransmission Timer

- In order to set retransmission timer, must know approximate RTT for both WAN and LAN connections
- One way:
  - measure each send/ACK combination
  - take time-averaged estimate of RTT
  - set timer to some factor times this average
  - (used in early TCPs…we will see improvements once we cover TCP in detail)

The Question of ACKs

- What exactly should the receiver ACK?
- Some possibilities:
  - ACK every packet, giving its sequence number
  - use *cumulative ACK*, where an ACK for number *n* implies ACKS for all $k < n$
  - use *negative ACKs* (NACKs), indicating which packet did not arrive
  - use *selective ACKs* (SACKs), indicating those that did arrive, even if not in order

Issues with ACKs

- ACKs might be dropped in the network:
  - often results in similar behavior as though a packet was dropped
  - so, do ACKs need reliable transfer too?
  - If so, then chicken-and-egg problem…
  - note that with cumulative ACKs, not too bad if some ACKs are lost, provided there are many of them
  - focus on cumulative ACKS (used by TCP)

Cumulative and Delayed ACKs

- Cumulative ACK Example:
  - receiver receives packets 1,2,3,5,6,7,8
  - sends ACKs for 1,2,3 or maybe 1,2,3,3,3,3,3
  - upon receiving packet 4, ACKs 8
- Observations:
  - can t ACK out-of-order packets
  - can delay ACKs, say, for every other packet
  - delaying might be useful for *piggybacking* ACKs on data (on reverse-direction flow)

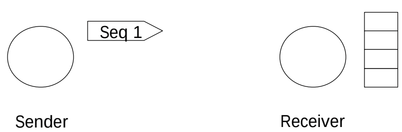
Send Window Size Issues

- A bigger send window size provides larger throughput… (well, maybe)
  - if w is too big for what the receiver can handle, extra data is discarded at the receiver and must be retransmitted (inefficient)
  - if w is too big for what the network can handle, extra is discarded within the network and must be retransmitted (also inefficient)
- Want flow and congestion control…

Avoiding receiver overrun

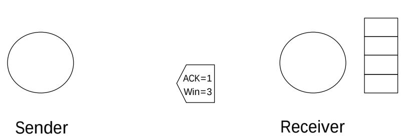
- Flow control
  - recall receiver's window is a measure of how much data receiver can buffer
  - would rather the sender not send more than the receiver can handle
  - need a way for the receiver to tell the sender how much buffer space is available
- Window "advertisement"
  - receiver tells sender how much space available

Example



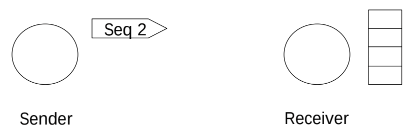
Example

Example

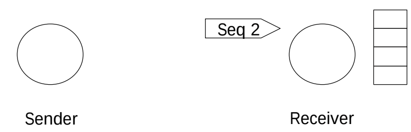


Example

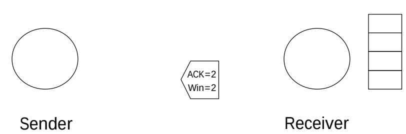
Example



Example

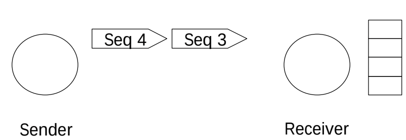


Example

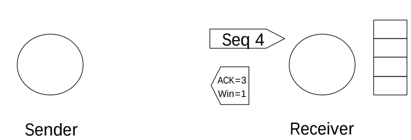


Example

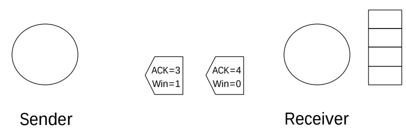
Example



Example

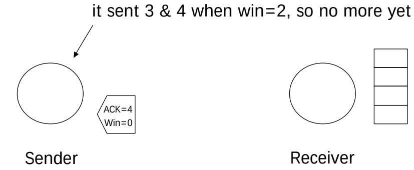


Example



Example

Sender receives ACK for 3, but remembers it sent 3 & 4 when win=2, so no more yet



Example

Sender now frozen due to flow control

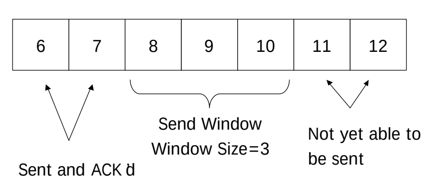
Flow Control

- Flow control happens when receiver is unable to keep up with sender's rate
  - consuming process may be busy
  - receiving computer may be slow
- Window information arrives with ACKs, so send window slides forward at the same time it might shrink or expand

Sliding Windows w/Flow Control

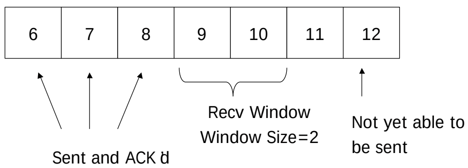
- Purposes of sliding window:
  - guarantees the reliable and efficient delivery of data
  - ensures data is delivered in order (or at least corrected at receiver)
  - provides for flow control
- Flow control is implemented by changing the sender's window based on the receiver's advertised window

Example (send window, w=3)



Sender now receives (ACK:8, win:2)

Example (recv ACK:8, win:2)

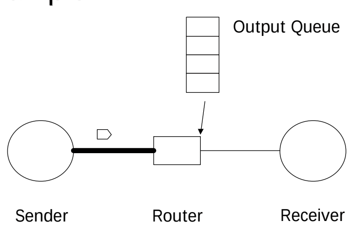


Window has "slid forward", but has also become smaller (by left window edge moving to the right). Window "shrinkage" (moving right edge to the left) is usually avoided.

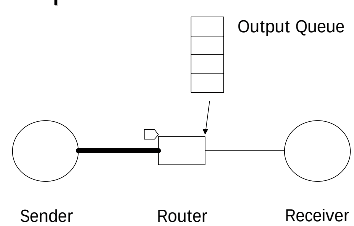
Flow and Congestion Control

- Limiting the sender's rate
  - based on receiver: flow control
  - based on network: congestion control
- Congestion Control
  - try to limit the sender's rate based on the ability of the network to deliver traffic

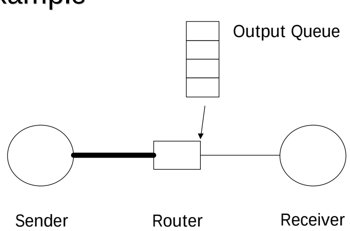
Example



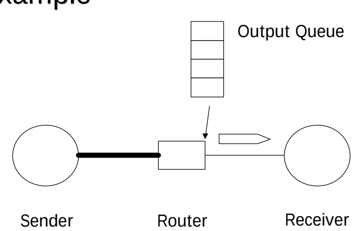
Example



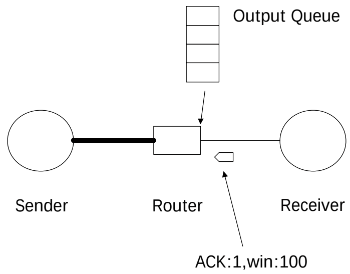
Example



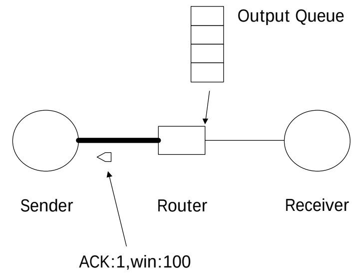
Example



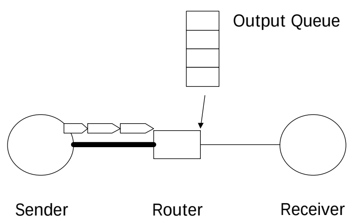
Example



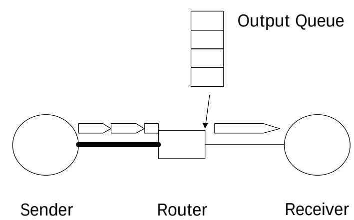
Example



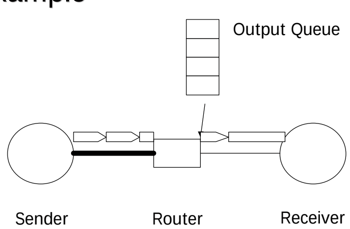
Example



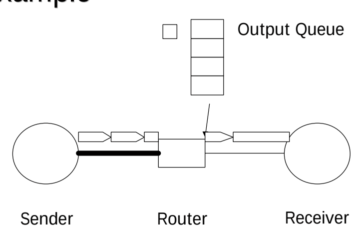
Example



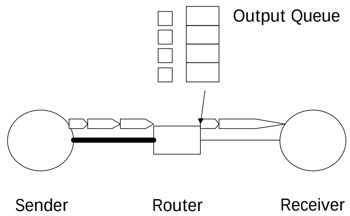
Example



Example



Example



Results of Congestion

- Large queue occupancy implies:
  - increased delay
  - possible increased jitter
  - increased loss probability
- Lost packet may trigger retransmissions:
  - sending more packets into the network while it is running over capacity exacerbates the congestion problem
  - in the limit, leads to *congestion collapse*

Congestion Collapse

- Condition in which network is busy, but no (not much) useful work is being accomplished
- Can occur with protocols that are not careful to avoid congesting the network
- Happened in real life in the ARPAnet about 1987 or so…

Dealing with Congestion

- Need these to deal with congestion:
  - a way to determine the network is becoming congested or is already congested
  - an algorithm to slow down during times of congestion
  - a way to speed up if the network becomes uncongested

Detecting Congestion

- Two approaches
  - explicit: network tells you
  - implicit: endpoint infers using traffic statistics
- On the Internet:
  - TCP uses packet loss to indicate congestion (an implicit approach)
  - ICMP Source Quench, and TCP ECN (experimental) provide explicit signaling
- Do lost packets always mean congestion?