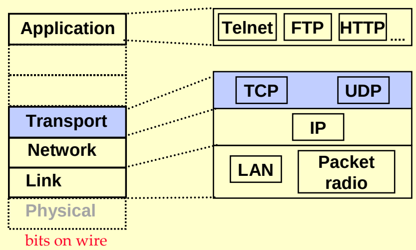
EEC173B/ECS152C

Review: TCP

Review: Internet Protocol Stack



- Do you remember the various mechanisms we have learned at network & link layers?

Transport Layer Services

- Underlying best-effort network
  - drops messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - delivers messages after an arbitrarily long delay
- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

Transport Layer: Connectionless Service

Goal: data transfer between end systems

- UDP - User Datagram Protocol
  - RFC 768
  - Internet's connectionless service
  - Unreliable (unordered) data transfer
  - No flow control
  - No congestion control

Transport Layer: Connection-Oriented Service

Goal: data transfer between end systems

- *Handshaking:* setup (prepare for) data transfer ahead of time
  - Hello, hello back human protocol
  - *set up "state"* in two communicating hosts
- TCP - Transmission Control Protocol
  - RFC 793, 1122, 1323, 2018, 2581
  - Internet's connection-oriented service

5

Design Issue

- At what rate do you send data?
  - What is max useful sending rate for different apps?
- Two components
  - Flow control
    - make sure that the receiver can receive
    - sliding-window based flow control:
      - receiver reports window size to sender
      - higher window → higher throughput
      - throughput = window/RTT
  - Congestion control
    - make sure that the network can deliver

6

Transmission Control Protocol (TCP)

- *Point-to-Point*
  - One sender, one receiver
- *Connection Management*
  - Connection oriented: handshaking (exchange of control messages) initialize sender, receiver state before data exchange
- *Reliable, in-order* byte-stream data transfer
  - loss: acknowledgements and retransmissions
- *Flow control:*
  - sender won't overwhelm receiver
  - Pipelined
- *Congestion control:*
  - senders "slow down sending rate" when network congested (so sender won't overwhelm network)

7

Examples

App's using TCP:

- HTTP (Web), FTP (file transfer), Telnet (remote login), SMTP (email)

App's using UDP:

- Streaming media, teleconferencing, DNS, Internet telephony

8

Some Flow Control Algorithms

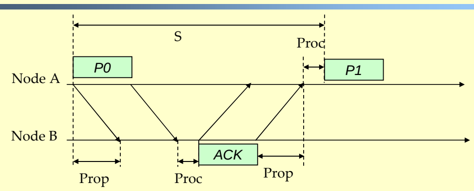
- Flow control for noisy channels
  - Packets may be lost
- Typically combined with error control
  - Reminder: data link layer also deals with encoding, framing, error detection like parity & polynomial code (CRC) …
- ARQ protocols
  - Stop and Wait
  - Go-Back-N
  - Selective Repeat

} Sliding Window protocols (Go-Back-N and Selective Repeat)

Flow Control: Quick overview

- What are common among them?
  - Basic concept: ask for retransmission to correct errors
  - Both data and ACK packets have *sequence numbers*
  - Receiver informs sender via *ACK or NACK* packets
  - Time-out period
- What make them different?
  - Sender window size
  - Receiver window size
  - How do they recover from errors?
  - Is a buffer required at the receiver

Stop And Wait

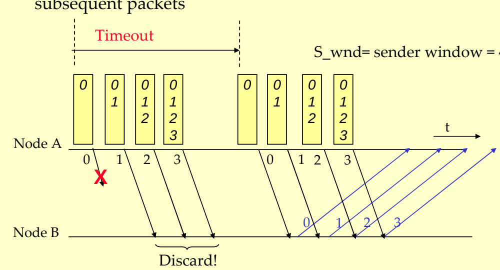


- Stop-and-wait is not very efficient
  - *Can we do something else while waiting for an ACK?*
- Sliding window protocol idea:
  - Allow Sender to keep transmitting a "window" of packets while waiting for ACKs

Remember how to compute efficiency?

Go-Back-N: Example

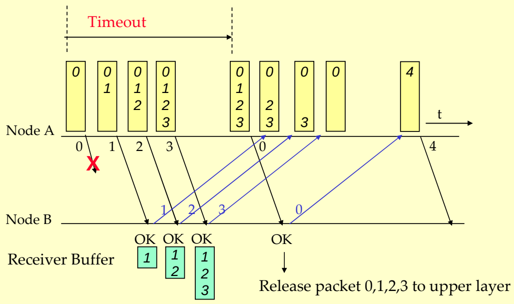
- Rule 1: can send up to s_wnd packets without acks
- Rule 2: when error occurs, retransmit packet plus all subsequent packets



- Receiver does not need to buffer packets

Selective Repeat: Example

- Packet n must not be sent until packet $n- W_S$ has been acknowledged (to avoid overloading receiver buffer)



Why do You Care About Congestion Control?

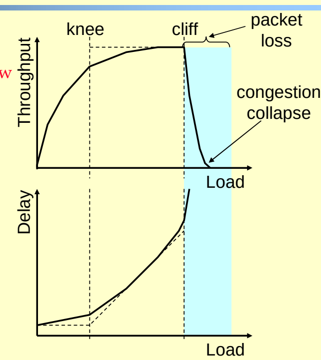
- Otherwise you get to congestion collapse
- How might this happen?
  - Assume network is congested (a router drops packets)
  - You learn the receiver didn't get the packet
    - either by ACK, NACK, or Timeout
  - What do you do? retransmit packet
  - Still receiver didn't get the packet
  - Retransmit again
  - …. and so on …
  - And now assume that everyone is doing the same!
- Network will become more and more congested
  - And this with duplicate packets rather than new packets!

Solutions?

- Slow down
  - If you know that your packets are not delivered because network congestion, slow down
- Questions:
  - How do you detect network congestion?
  - By how much do you slow down?

What's Really Happening?

- Knee – point after which
  - Throughput increases very slow
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to decrease very fast to zero (congestion collapse)
  - Delay approaches infinity
- Note (in an M/M/1 queue)
  - Delay ~ 1/(1 –load)



Goals

- Goal: Operate near the knee point and remain in equilibrium
  - Don't put a packet into network until another packet leaves. Maintain number of packets in network "constant"
- Detect when network approaches/reaches knee point and Stay there
  - How do you get there?
  - What if you overshoot (i.e., go over knee point) ?
- Possible solution:
  - Increase window size until you notice congestion
  - Decrease window size if network congested

17

Detecting Congestion-1

- Implicit network signal
  - Loss (e.g. TCP Tahoe, Reno, New Reno, SACK)
    - +relatively robust, -no avoidance
    - [FF96] compared Tahoe, Reno, and SACK TCP
  - Delay (e.g. TCP Vegas)
    - +avoidance, -difficult to make robust
  - Easily deployable
  - Robust enough? Wireless?

18

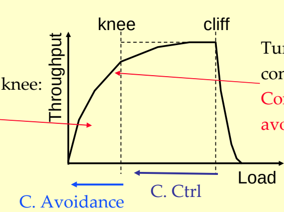
Detecting Congestion-2

- Explicit network signal
  - Send packet back to source (e.g. ICMP Source Quench)
    - Control traffic congestion collapse
  - Set bit in header
    - e.g., DEC DNA/OSI Layer 4 [CJ89], ECN [RFC2481]
    - Can be subverted by selfish receiver
  - Unless on every router, still need end-to-end signal
  - Could be be robust, if deployed

19

TCP: Basic idea

When a connection starts, want to quickly approach knee: Slow Start phase



Turn more conservative
Congestion avoidance

- Congestion control goal: stay left of cliff
- Congestion avoidance goal: stay left of knee

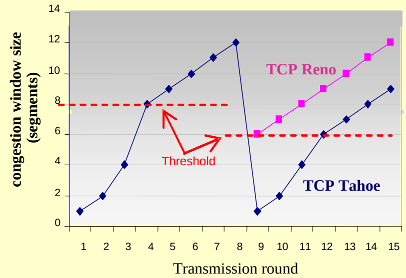
20

TCP: Slow Start & Congestion Avoidance

- Slow start
  - Goal: discover congestion quickly
  - How: Quickly increase *CongWin* until network congested → get a rough estimate of the optimal of *CongWin*
    - Set *CongWin* =1
    - Each time a segment is acknowledged, double *CongWin*
  - Slow Start is not actually slow
    - *CongWin* increases exponentially
- Congestion avoidance: slow down "Slow Start"
  - **If** *CongWin* > *Threshold* **then** each time a segment is acknowledged increment *CongWin* by *1.*

21

Illustration



22

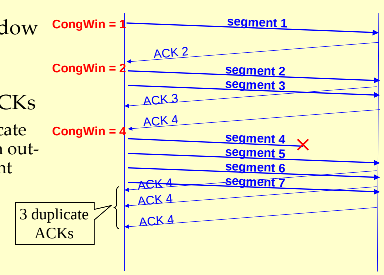
TCP Congestion Control

- Maintains three variables:
  - CongWin – congestion window
  - FlowWin – flow window; receiver advertised window
  - Threshold – threshold size (used to update cwnd)
- For sending use: win = **min**(FlowWin, CongWin)
- Timeout
  - When timer expires, TCP sender reduces rate
  - Set *Threshold* 1/2 of CongWin just before the loss event
  - Set *CongWin* = 1
    - Window then grows exponentially until it hits *Threshold*, and then grows linearly
- Recovery can be slow if we wait for timeout
  => Don't wait for window to drain
  => Look for duplicate ACKs

23

Refinement: Fast Retransmit

- Don't wait for window to drain
- Resend a segment after 3 duplicate ACKs
  - Remember a duplicate ACK means that an out-of sequence segment was received



Action: after 3 duplicate ACKs, go ahead and retransmit 4 without waiting for timeout

24

Refinement: Fast Recovery

- After a fast-retransmit,
  - Set *Threshold* -> ½ of *CongWin* just before the loss event
  - Set *CongWin* to the new *Threshold*
    - i.e., don't reset *CongWin* to 1
    - Start growing linearly, don't need slow start again

Philosophy:

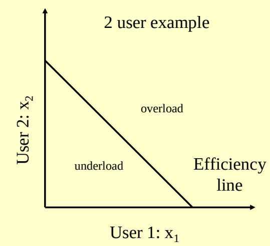
- 3 dup ACKs indicates network capable of delivering some segments
=> less aggressive congestion control

- Timeout before 3 dup ACKs is "more alarming"
=> Cut back aggressively

25

Efficient Allocation

- Too slow
  - Fail to take advantage of available bandwidth → underload
- Too fast
  - Overshoot knee → overload, high delay, loss
- Everyone's doing it
  - May all under/over shoot → large oscillations
- Optimal:
  - $\Sigma x_i = X_{goal}$
- Efficiency = 1 - distance from efficiency line



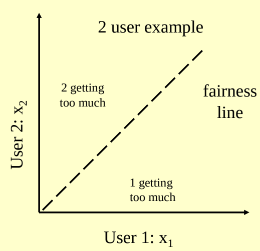
26

Fair Allocation

- Maxmin fairness
  - Flows which share the same bottleneck get the same amount of bandwidth

$$F(x) = \frac{\left(\sum x_i\right)^2}{n\left(\sum x_i^2\right)}$$

- Assumes no knowledge of priorities
- Fairness = 1 - distance from fairness line



27

Reflections on TCP

- Assumes that all sources cooperate
- Assumes that congestion occurs on time scales greater than 1 RTT
- Only useful for reliable, in order delivery, non-real time applications
- Vulnerable to non-congestion related loss (e.g. wireless)
- Can be unfair to long RTT flows
- TCP cannot distinguish between link loss and congestion loss (e.g., wireless environment)

28