TCP

Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute
shivkuma@ecse.rpi.edu
http://www.ecse.rpi.edu/Homepages/shivkuma

Based in part upon the slides of Prof. Raj Jain (OSU), K. Vastola (RPI)

- TCP Services
- Reliability mechanisms: soft state, sequence numbers, retransmissions, timeouts, 3-way handshake
- Congestion Control: dynamic windows, robust stability issues.

TCP: Key features

- *Connection-oriented*
- *Point-to-point*: 2 end-points (no broadcast or multicast)
- *Reliable transfer*: Data is delivered in-order
- *Full duplex* communication
- *Byte-stream* interface: sequence of octets
- *Reliable startup*: Data on old connection does not confuse new connections
- *Graceful shutdown*: Data sent before closing a connection is not lost. Reset or immediate shutdown also possible.

Reliability

- Reliability provided by:
  - Reliable connection startup: Data on old connection does not confuse new connections
  - Graceful connection shutdown: Data sent before closing a connection is not lost.
  - Data segmented for transmission and acknowledged by destination. Timeout + Retransmission provided if data unacknowledged. This is formally called "soft-state"
  - Checksum provided to catch errors.
  - Re-sequencing of out-of-order data; discarding of duplicate data.
  - Window flow control => sender cannot overrun receiver buffers

TCP Header Format

Source **Port**	Dest Port	**Seq** No	Ack No	Header length	Resvd	Control	Window
16	16	32	32	4	6	6	16

Check-sum	Urgent	Options	Pad	Data
16	16	x	y	

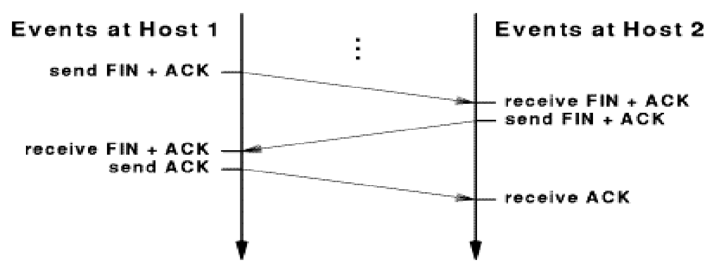
Size in ***bits***

TCP Header

- Source Port (16 bits): Identifies source user process
  20 = FTP, 23 = Telnet, 53 = DNS, 80 = HTTP, ...
- Destination Port (16 bits)
- Sequence Number (32 bits): Sequence number of the first byte in the segment. If SYN is present, this is the initial sequence number (ISN) and the first data byte is ISN+1.
- Ack number (32 bits): Next byte expected

Three-Way Handshake

- 3-way handshake: necessary and sufficient for unambiguous setup/teardown even under conditions of loss, duplication, and delay



TCP Interactive Data Flow

- Problems:
  - Overhead: 40 bytes header + 1 byte data
  - To batch or not to batch: response time important
- Batching acks:
  - Delay-ack timer: piggyback ack on echo
  - 200 ms timer (fig 19.3)
- Batching data:
  - Nagle’s algo: Don’t send packet until next ack is received.
  - Developed because of congestion in WANs

TCP Bulk Data Flow

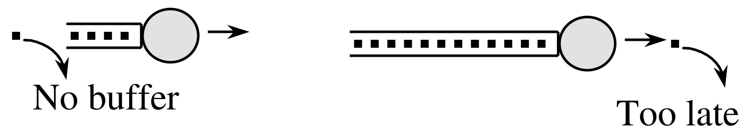
- Sliding window:
  - Send multiple packets while waiting for acks (fig 20.1) upto a limit (W)
  - Receiver need not ack every packet
  - Acks are cumulative.
  - Ack # = Largest consecutive sequence number received + 1
  - Two transfers of the data can have different dynamics (eg: fig 20.1 vs fig 20.2)
- Receiver window field:
  - Reduced if TCP receiver short on buffers

TCP Bulk Data Flow (Contd)

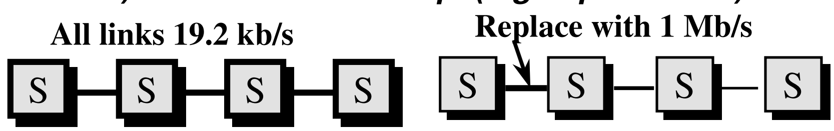
  - End-to-end flow control
  - Window update acks: receiver ready
  - Default buffer sizes: 4096 to 16384 bytes.
  - Ideal: window and receiver buffer = bandwidth-delay product
- TCP window terminology: figs 20.4, 20.5, 20.6
  - Right edge, Left edge, usable window
  - "closes" => left edge (snd_una) advances
  - "opens" => right edge advances (receiver buffer freed => receiver window increases)
  - "shrinks" => right edge moves to left (rare)

The Congestion Problem

- *Problem: demand outstrips available capacity …*
- Q: Will the “congestion” problem be solved when:
  - *a) Memory becomes cheap (infinite memory)?*

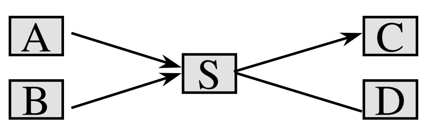


  - ***b) Links become cheap (high speed links)?***



File Transfer time = ***5 mins***   *File Transfer Time =* ***7 hours***

- ***c) Processors become cheap (fast routers switches)?***



Scenario: All links 1 Gb/s. A & B send to C.

Ans: None of the above solves congestion !

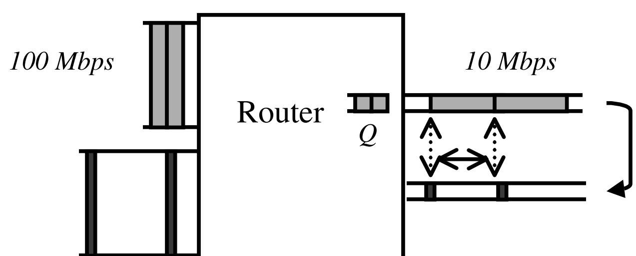
- Congestion: Demand > Capacity
  - It is a dynamic problem => Static solutions are not sufficient
  - TCP provides a dynamic solution

TCP Congestion Control

- *Window flow control*: avoid receiver overrun
- *Dynamic window congestion control:* avoid/control network overrun
  - <u>Observation:</u> Not a good idea to start with a large window and dump packets into network
  - Treat network like a black box and start from a window of 1 segment *("slow start")*
  - Increase window size exponentially *("exponential increase")* over successive RTTs => quickly grow to claim available capacity.
  - Technique: Every ack: increase *cwnd* (new window variable) by 1 segment.
  - Effective window = Min(*cwnd, Wrcvr*)

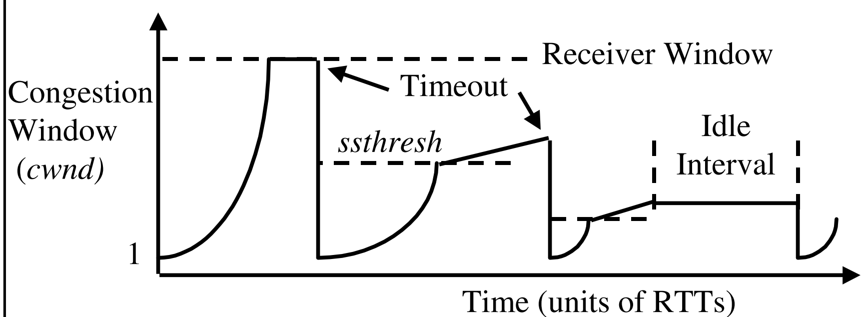
Dynamics

- Rate of acks = rate of packets at the bottleneck: *"Self-clocking"* property.



Congestion Detection

- *Packet loss as an indicator of congestion.*
  - Set slow start threshold *(ssthresh)* to min(*cwnd, Wrcvr)/2*
  - Retransmit pkt, set *cwnd* to 1 (reenter slow start)



Congestion avoidance

  - Increment *cwnd* by 1 per ack until *ssthresh*
  - Increment by 1/*cwnd* per ack afterwards *("Congestion avoidance" or "linear increase")*
- Idea: *ssthresh* estimates the bandwidth-delay product for the connection.
- Initialization: *ssthresh* = Receiver window or default 65535 bytes. Larger values thru options.
- If source is idle for a long time, cwnd is reset to one.

Timeout and RTT Estimation

- Timeout: for robust detection of packet loss
- Problem: How long should timeout be ?
  - Too long => underutilization; too short => wasteful retransmissions
  - Solution: *adaptive timeout: based on RTT*
- RTT estimation:
  - Early method: exponential averaging:
    - $R \leftarrow \alpha*R + (1 - \alpha)*M$ { M =measured RTT}
    - RTO = β*R {β = delay variance factor}
    - Suggested values: $\alpha = 0.9$, $\beta = 2$

Timer Backoff/Karn's Algorithm

- Timer backoff: If timeout, RTO = 2*RTO {exponential backoff}
- Retransmission ambiguity problem:
  - During retransmission, it is unclear whether an ack refers to a packet or its retransmission. Problem for RTT estimation
  - Karn/Partridge: don't update RTT estimators during retransmission.
    - Restart RTO only after an ack received for a segment that is not retransmitted

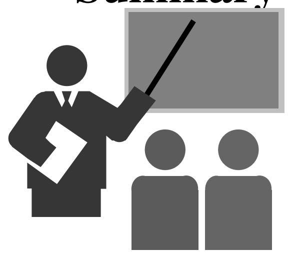
TCP Performance Optimization

- SACK: selective acknowledgments: specifies blocks of packets received at destination.
- *Random early drop (RED)* scheme spreads the dropping of packets more uniformly and reduces average queue length and packet loss rate.
- *Scheduling* mechanisms protect well-behaved flows from rogue flows.
- *Explicit Congestion Notification (ECN):* routers use a explicit bit-indication for congestion instead of loss indications.

Congestion control summary

- Sliding window limited by receiver window.
- Dynamic ***windows***: slow start (exponential rise), congestion avoidance (linear rise), multiplicative decrease.
- Adaptive ***timeout***: need mean RTT & deviation
- Timer back off and Karn's algo during retransmission
- Go-back-N or Selective ***retransmission***
- Cumulative and Selective ***acknowledgements***
- Timeout avoidance: FRR
- ***Drop*** policies, ***scheduling*** and ***ECN***

Summary



- TCP reliability model and mechanisms
- Congestion control mechanisms