Transmission Control Protocol

Internet Architecture

Internet Architecture

- End-to-end semantics:
  - functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level

  e.g., end-to-end caretaking

  A --- B --- C --- D --- E

  J.H. Saltzer, D.P. Reed and D.D. Clark. End-to-end design argument in system design, ACM TOCS, Vol 2, Number 4, November 1984, pp.277-288.

Internet Architecture (Cont'd)

- Intermediate node (e.g., routers): packet forwarding, FIFO (first in first out), discard packets when overflow, stateless, know almost nothing about end-to-end sessions

  → simple and robust
- End host: know almost nothing about network internals, manage all end-to-end session-related states

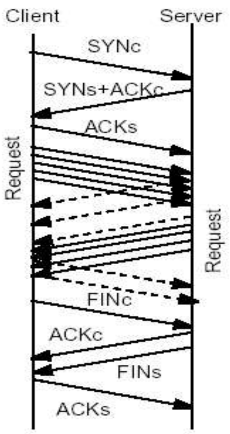
  → complex and intelligent

Transmission Control Protocol (TCP)

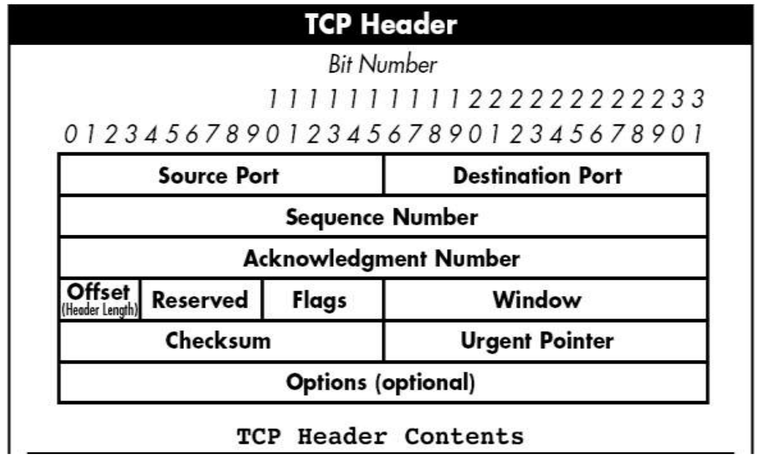
Transmission Control Protocol

- TCP
  - a *connection-oriented*, *end-to-end reliable*, and stream-like transport protocol over the connectionless, unreliable, and datagram-based IP service

Connection-oriented

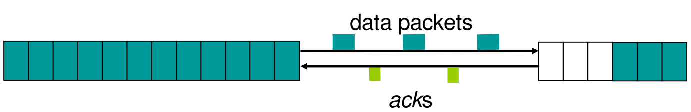


- connection establishment: associate endpoints
  - 3-way handshakes open bi-directional data channels
  - data only transferred after connection established (with bsd socket interface)
  - 2-way handshakes close data channels individually (i.e., graceful close. other termination forms exist)



Error Control

- error control: detect and recover packet errors
  - lost packet: accumulative acknowledgment from receiver, timeout at sender, retransmit if timeout occurs
  - duplicated packet: discard according to sequence number (note: the relation between window space and sequence space) at receiver, no further recovery needed

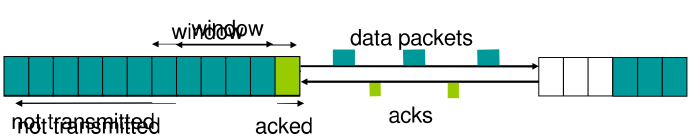


Error Control (Cont'd)

- corrupted packet: discard according to checksum (also include a pseudo IP header) at receiver, as a lost packet
- reordered packet: reorder according to sequence number at receiver, no further recovery needed

Flow Control

- flow control: coordinate endpoints
  - slide window based flow control (in sequence space)
  - receiver's acknowledgment advances window and adjust window size
  - sender can not exceed window-bottom plus window-size in sequence space

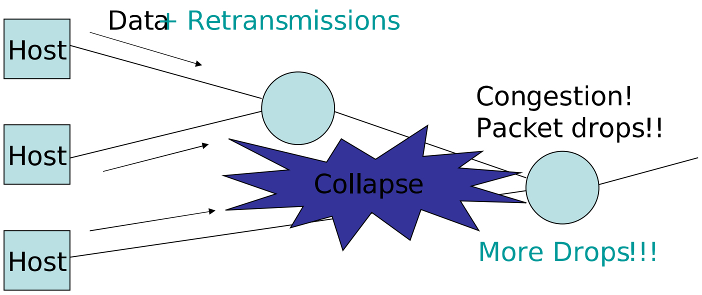


Congestion Control

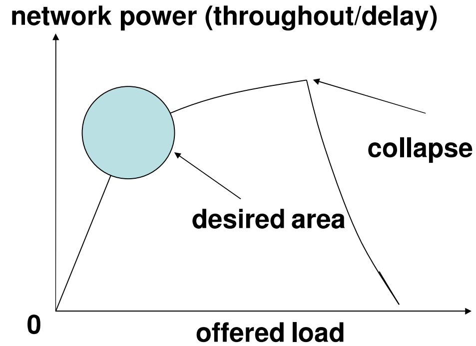
- congestion control: coordinate endpoints and network
  - was introduced in later 1980's
  - now standard in every TCP implementation
  - at the heart of TCP/IP research

Internet Congestion Collapse

- In the late 80s, the Internet suffered a series of congestion collapse

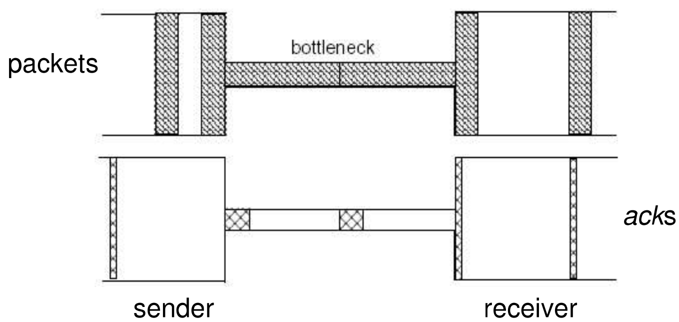


Congestion Control Objective



- load-power curve
  - low-load: significant gain
  - mid-load: negligible gain
  - high-load: negative gain (collapse)

ack self-clocking



In equilibrium: a new packet isn't put into the network until an old packet leaves.

TCP Variants – TCP Tahoe

- *cwnd* (congestion window)
- initial operation
  - *cwnd* = 1 mss (maximum segment size)
  - *ssthresh* (slow-start threshold)
  - max. outstanding data

    swnd = min{*cwnd*, *rwnd*, sender buffer size}

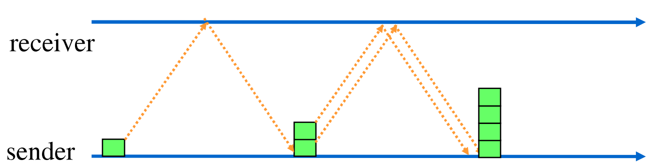
Van Jacobson and Mike Karels. Congestion avoidance and control. ACM Computer Communication Review, 18(4):314--329, August 1990. *Revised version of his SIGCOMM '88 paper.*

Slow-start

- slow-start: when *cwnd* < *ssthresh*
  - *cwnd* += 1 mss on a new *ack*
  - for ack-every-segment receiver

    $cwnd_{i+1} = cwnd_i * 2$
  - for delayed-ack receiver

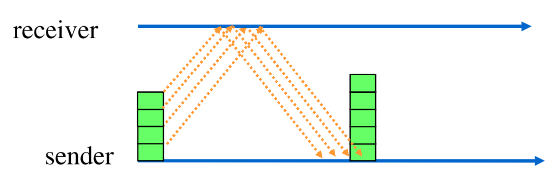
    $cwnd_{i+1} \approx cwnd_i * 1.5$



Congestion Avoidance

- congestion avoidance: when *cwnd* ≥ *ssthresh*
  - new *ack*, $cwnd \mathrel{+}= \frac{mss}{cwnd}$ *mss*
  - for ack-every-segment receiver,

    $cwnd_{i+1} = cwnd_i + 1mss$
  - for delayed-ack receiver $cwnd_{i+1} \approx cwnd_i + \frac{mss}{2}$



Timeout

- packet is assumed lost when timeout occurs
- packet loss is assumed due to network congestion
- retransmit lost packet

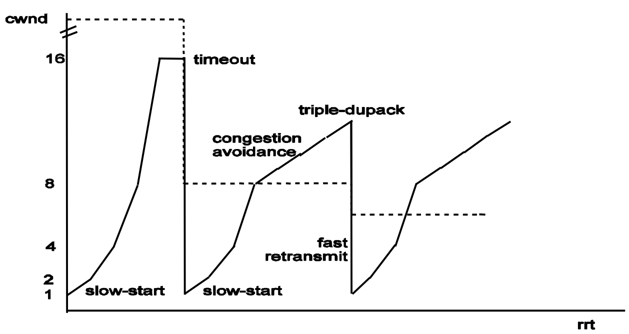
$$ssthresh = \max\{2 \text{ mss}, \frac{\min\{cwnd, amount-of-in-flight-data\}}{2}\}$$

$$cwnd = 1 \text{ mss}$$

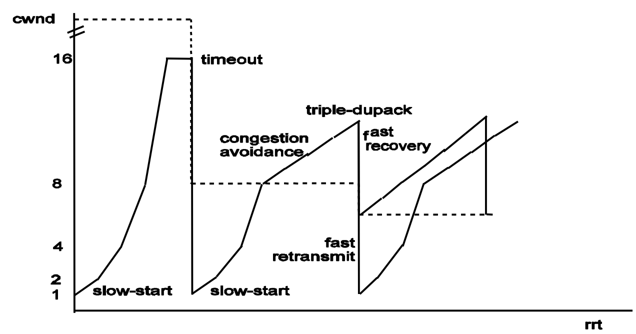
- followed by slow start

Fast Retransmit

- retransmission after timeout is slow
- receiver returns duplicated acknowledgment on non-in-order packets (due to reordered or lost packets)
- sender assumes the most unacknowledged packet is lost if number of *dupack*s exceeds a certain threshold (3 in most popular TCP implementations)
- retransmit the most unacknowledged packet, adjust *cwnd* and *ssthresh* as a timeout occurs
- followed by slow-start procedure



TCP Variants – TCP Reno



- Fast recovery: for triple-dupack, retransmit most unacknowledged packets, set ssthresh as fast retransmit, set cwnd = ssthresh

TCP New Reno

- partial acknowledgment: recover multiple packet losses
  - when entering fast recovery, record highest *seqno* ever sent as (*record*)
  - a *newack* not covers *record* is evaluated as a *pack*, retransmit the most unacknowledged packet
  - when *newack* covers *record*, exit fast recovery

TCP SACK

- selective acknowledgment: recover multiple packet losses
  - receiver returns *sack* option and the first field specifies a *hole* in queue when a packet arrives
  - duplicates the first two *sack* fields in previous *sack* option
  - if a packet is reported in a *hole* three times, it is assumed lost and is arranged for retransmission

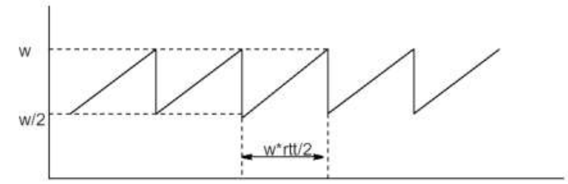
TCP Throughput

- For saturated sender, TCP throughput (bytes/sec) can be approximated as the sender window size (bytes) over round-trip time (seconds).
- Assuming rwnd > > cwnd, the sender window size equals cwnd, which is determined according to the loss events.

TCP Throughput

- triple-*dupack* model

$\frac{3w^2}{8} = \frac{1}{p}$, w: cwnd, p: loss rate



  - limitation: small packet loss rate, periodically loss
  - other models: consider *delack* and assume drop-window-tail loss

TCP Throughput

- model with timeout
  - BSD timeout backoff:
    1, 2, 4, 8, 16, 32, 64, 64, ...
  - correlated losses in *rtt*, independent between *rtt* packet
  - $\frac{1}{rtt\sqrt{\frac{2bp}{3}}+T_0\min(1,3\sqrt{\frac{3bp}{8}}p(1+32p^2))}$, b: *ack* pattern
  - TCP-Friendly Rate Control (TFRC) protocol