TCP congestion control

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley

Chapter 3 outline

3.1 Transport-layer services

3.2 Multiplexing and demultiplexing

3.3 Connectionless transport: UDP

3.4 Principles of reliable data transfer

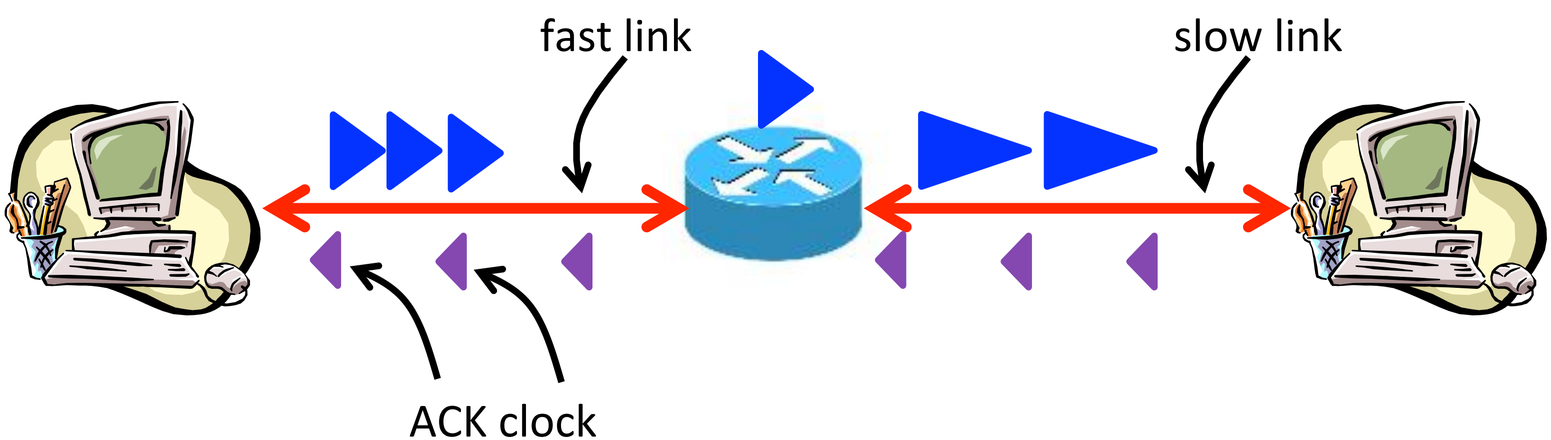
3.5 Connection-oriented transport: TCP
- Segment structure
- Reliable data transfer
- Flow control
- Connection management

3.6 Principles of congestion control

3.7 TCP congestion control

TCP congestion control

- TCP congestion control
  - Introduced by Van Jacobson in the late 80's
  - Done without changing headers or routers
  - Senders try and determine capacity of network
  - Implicit congestion signal: packet loss
  - ACK from previous packet determines when to send more data, "self-clocking"

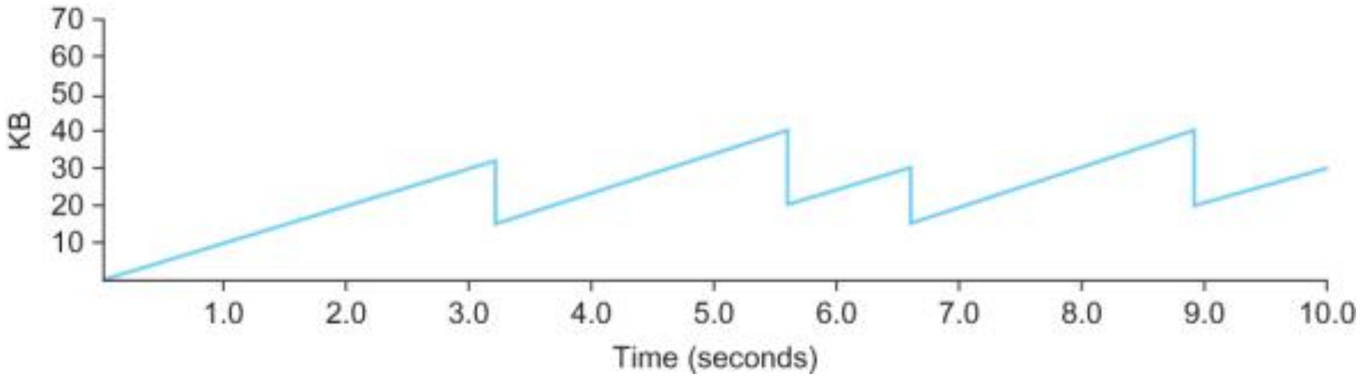


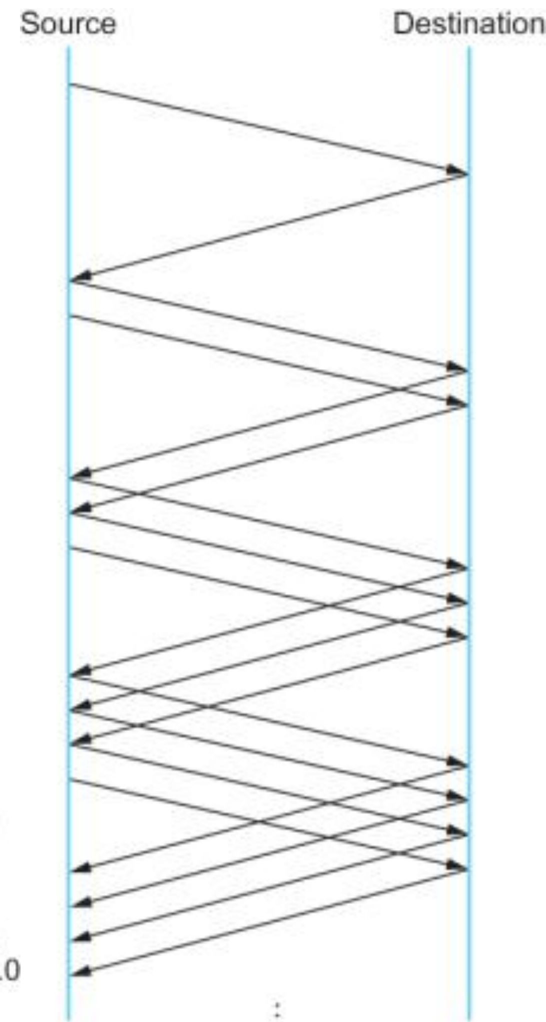
TCP congestion control

- Each TCP sender tracks:
  - rwnd = Advertised window, for flow control
  - cwnd = Congestion window, for congestion control
- Sender uses minimum of the two:
  - rwnd prevents overrunning receiver's buffer
  - cwnd prevents overloading network
- Situation is dynamic:
  - Network changes
    - e.g. new high bandwidth link, hosts start/stop sending
  - Sender always searching for best sending rate

Basic TCP congestion control

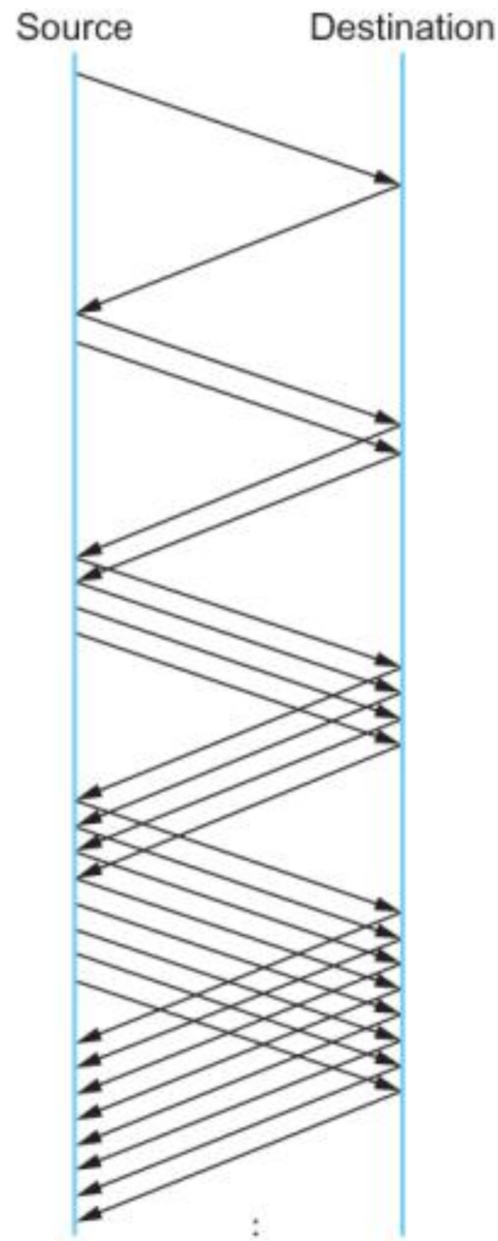
- Add one packet to window per RTT
  - Works well if we start near capacity
  - Otherwise could take a long time to discover real network capacity





Slow start

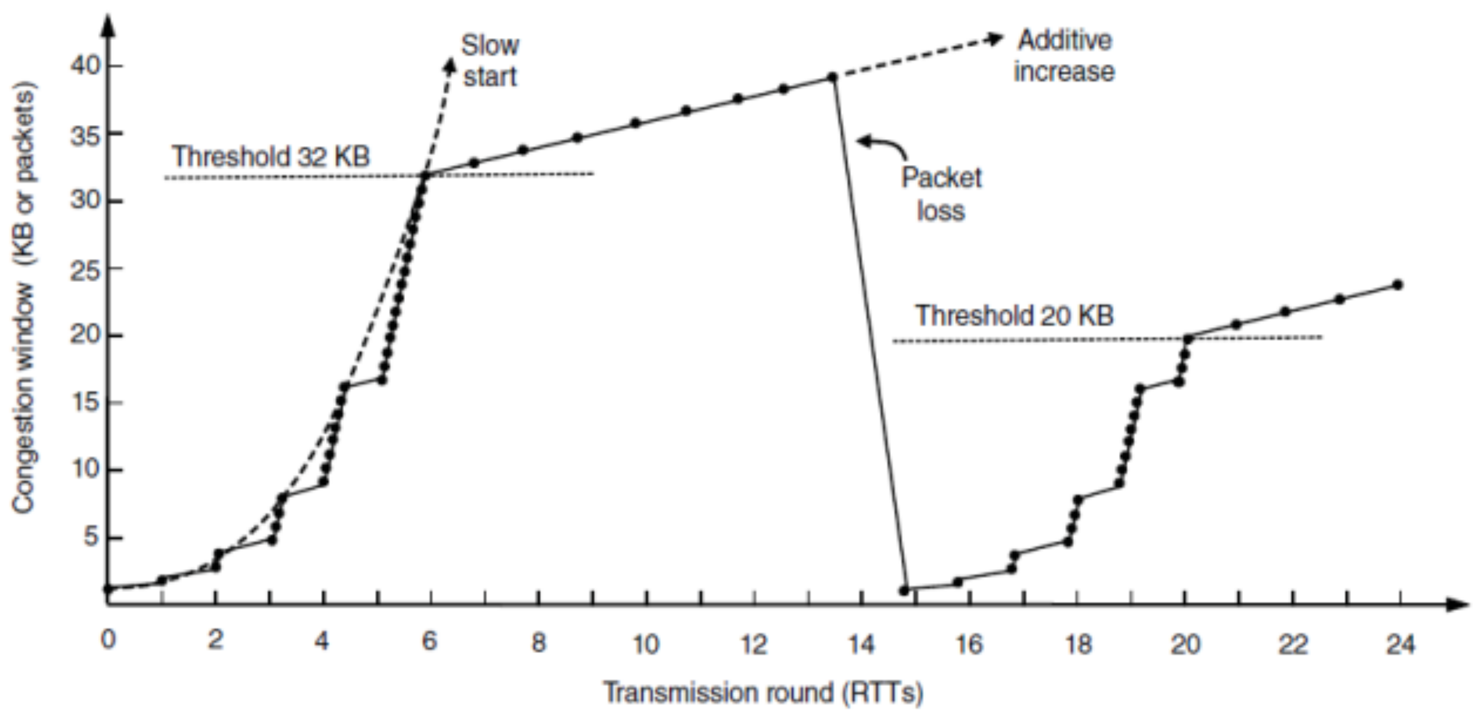
- Slow start
  - Increase congestion window rapidly from cold start of 1
  - Add 1 to window for every good ACK
    - Exponential increase in packets in flight
  - On packet loss, start over at 1
  - Slow in comparison to original TCP
    - Immediate sending up to advertised window (caused congestion collapse)



http://histrory.visualland.net/tcp_swnd.html

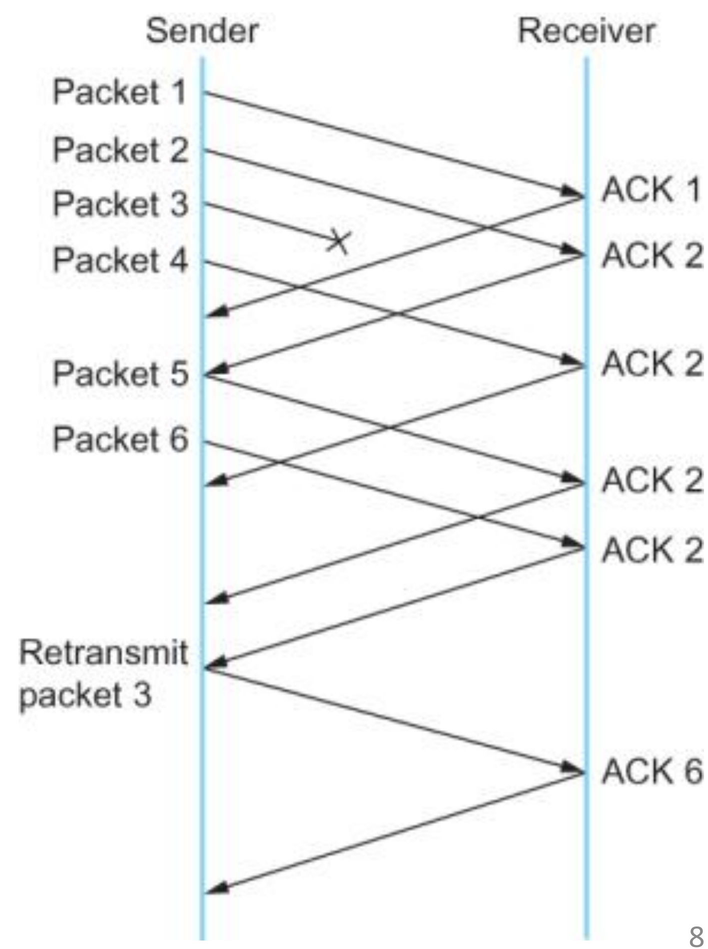
Slow start threshold, `ssthresh`

- Congestion threshold (slow start threshold)
  - Initially set to large value
  - On multiplicative decrease, `ssthresh = cwnd/2`
  - When ramping back up, switch to additive upon reaching `ssthresh`



Fast retransmission

- Problem: Timeouts take a long time
  - Connection sits idle waiting for a packet we are pretty sure is never going to be ACK'd
- Fast retransmission
  - Heuristic to retransmit packet we suspect was lost
  - Triggered when we observe 3 duplicate ACKs
  - 20% increase in throughput
- TCP "Tahoe"

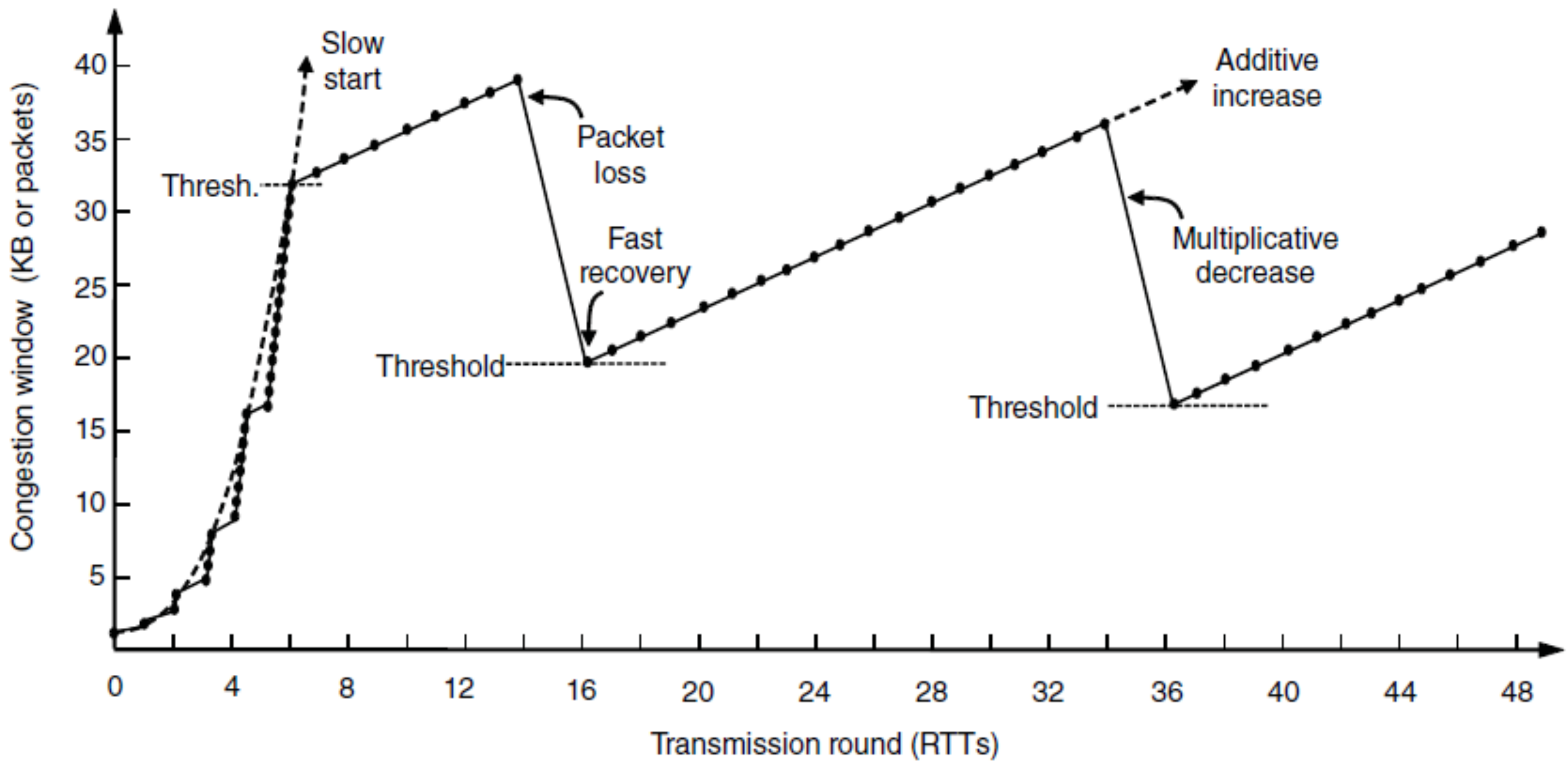


Fast recovery

- Problem: Restarting from 1 takes too long
  - We spend too long below "known" network limit
- Fast recovery
  - ACK clock still working even though packet was lost
  - Count up dup ACKs (including 3 that triggered fast retransmission)
  - Once packets in-flight has reached new threshold, start sending packet on each dup ACK
  - Once lost packet ACK'd, exit fast recovery and start linear increase

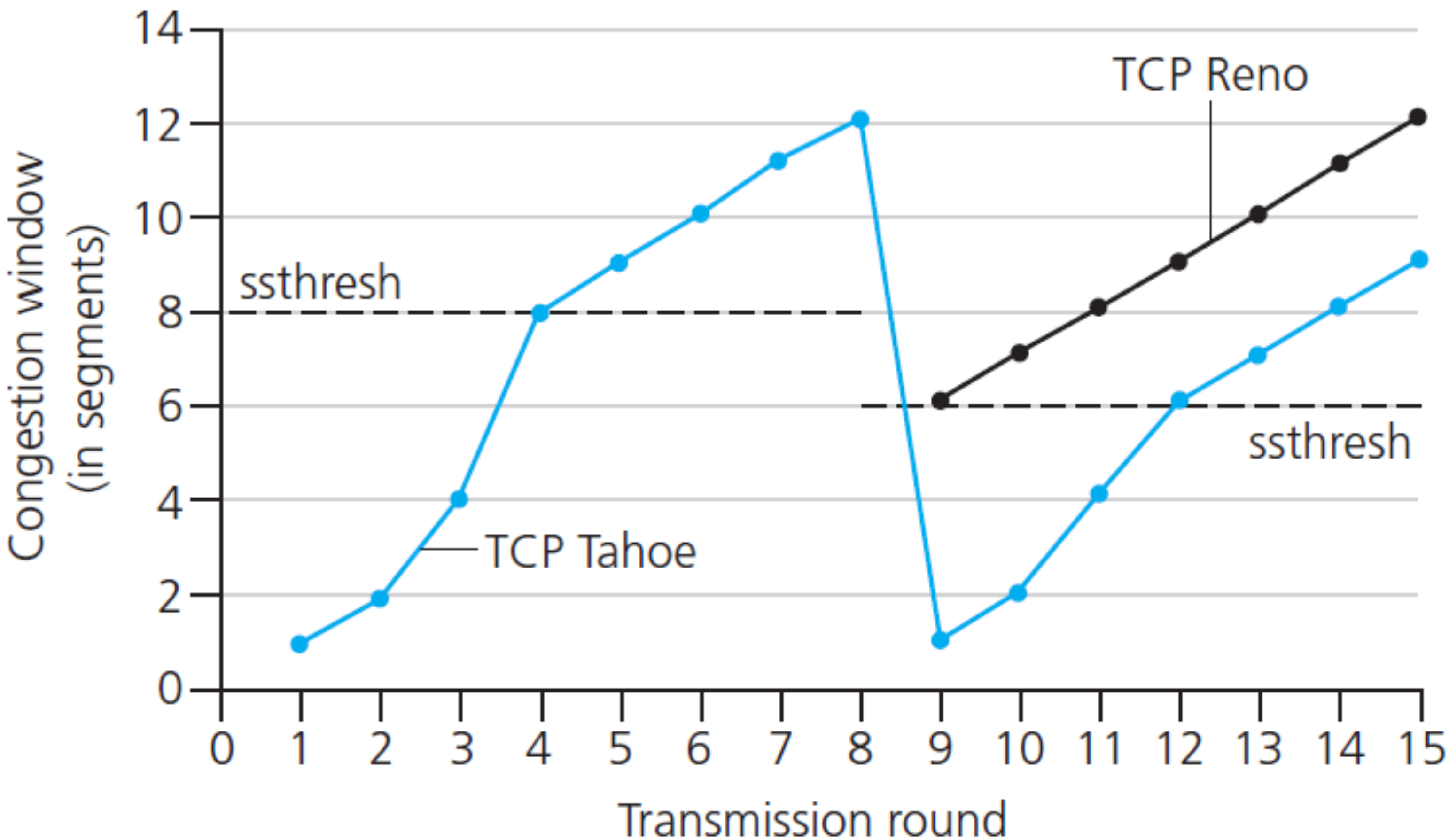
Fast recovery

- TCP "Reno"
  - Tahoe + fast recovery



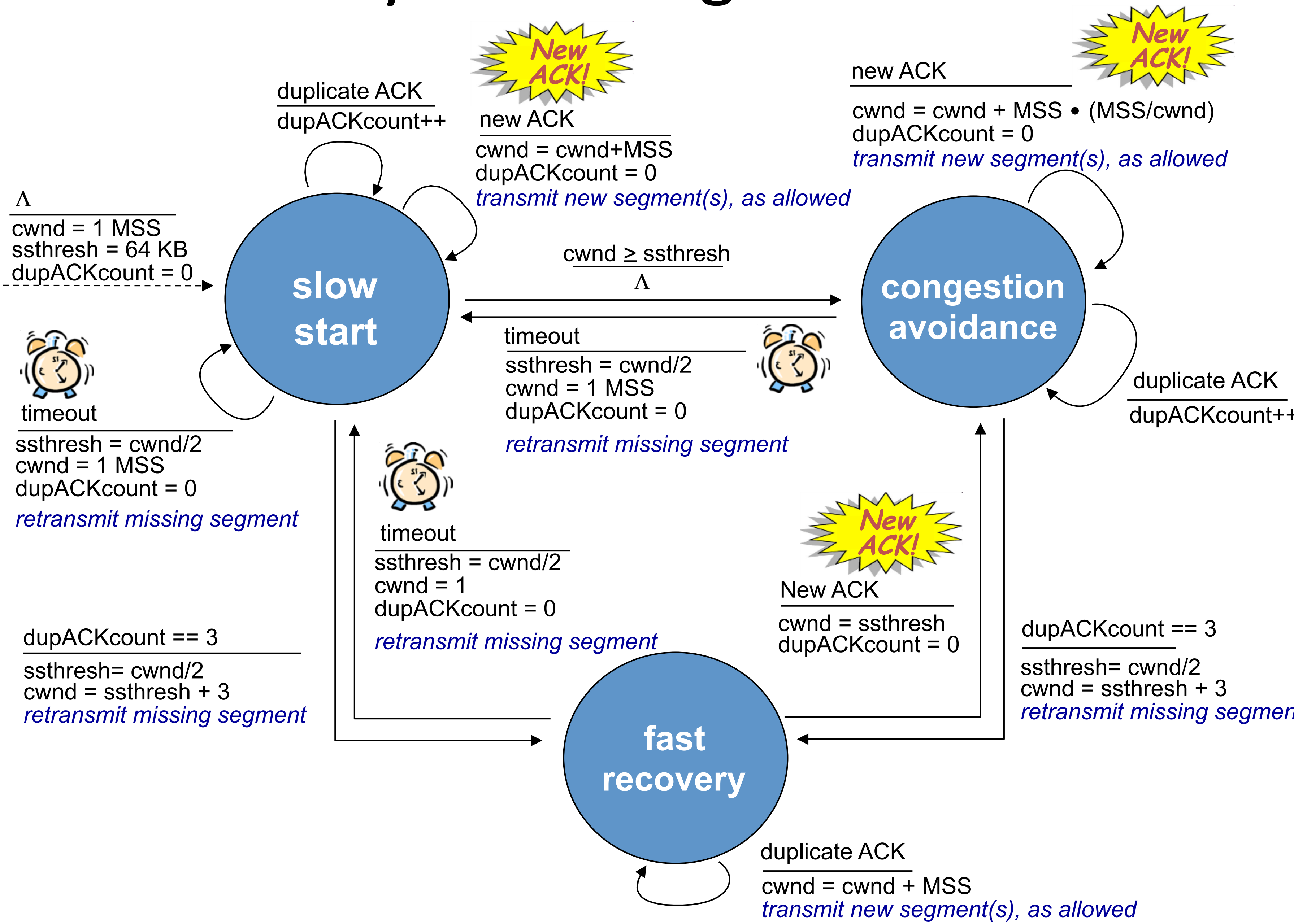
Fast recovery

- TCP "Reno"
  - Tahoe + fast recovery



http://www.brunocasari.net/projects_content/2?width=1000&height=500&iframe=true

Summary: TCP congestion control



Some of TCP's flavors

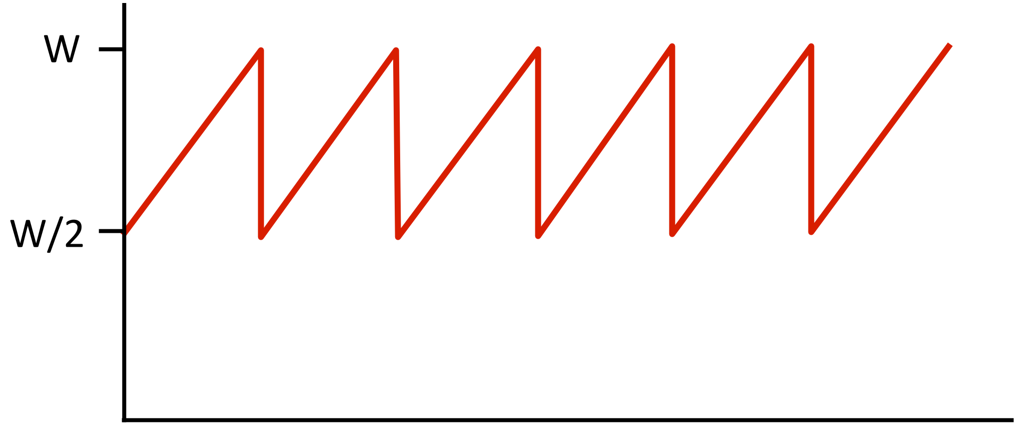
Name	Features
Tahoe	Slow start, congestion avoidance, fast retransmit.
Reno	Tahoe's features + fast recovery.
New Reno	Improves Reno to handle multiple packet loss within window. Changes to fast recovery, allows filling of multiple holes in sequence space.
Vegas	Monitor for signs of increasing congestion using RTT. Supports linear increase and *decrease* of congestion window.
BIC	Binary Increase Congestion control, optimized for high speed, long latency networks (long fat networks). Default in Linux 2.6.8-2.6.18.
CUBIC	Less aggressive that BIC, based on a cubic growth function. Default in Linux 2.6.19+
Compound	Microsoft, optimized for long fat networks while trying to remain fair. Default in XP and Vista, available in Windows 7.
...	

http://www.speedguide.net/articles/windows-7-vista-2008-tweaks-2574

TCP throughput

- Avg. TCP throughput as function of window size, RTT?
  - Ignore slow start, assume always data to send
- W: window size (measured in bytes)
  - Avg. window size (# in-flight bytes) is ¾ W
  - Avg. throughput is 3/4W per RTT

$$\text{Avg. TCP throughput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$



TCP over long, fat pipes

- Example:
  - 1500 byte segments, 100ms RTT
  - Want 10 Gbps throughput
  - Requires W = 83,333 in-flight segments
- Throughput in terms of segment loss probability, L

  [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\ \sqrt{L}}$$

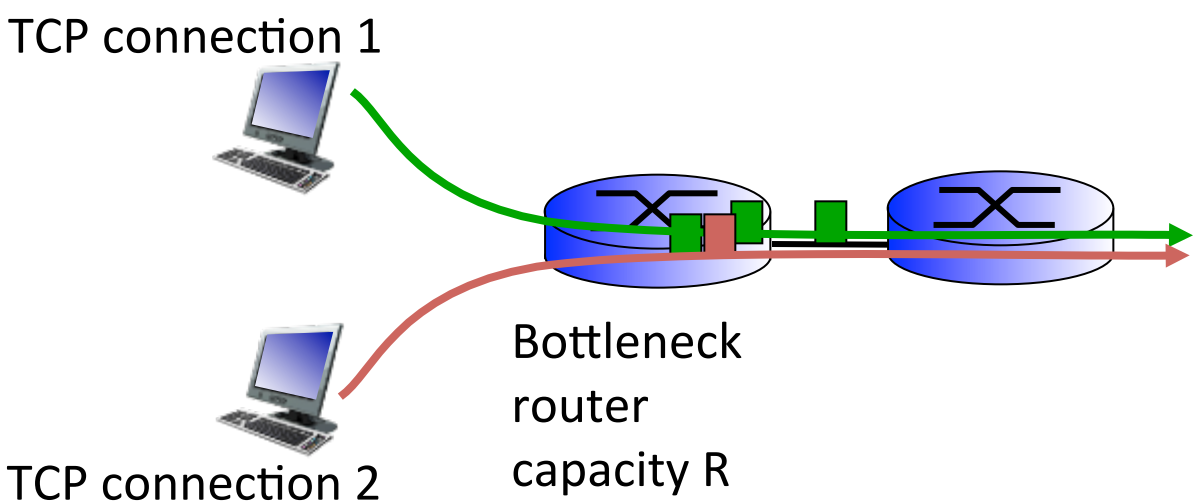
➜ To achieve 10 Gbps throughput, need a loss rate of $L = 2 \times 10^{-10}$ *– a very small loss rate!*

- New versions of TCP for high-speed environments

TCP fairness

Fairness goal:

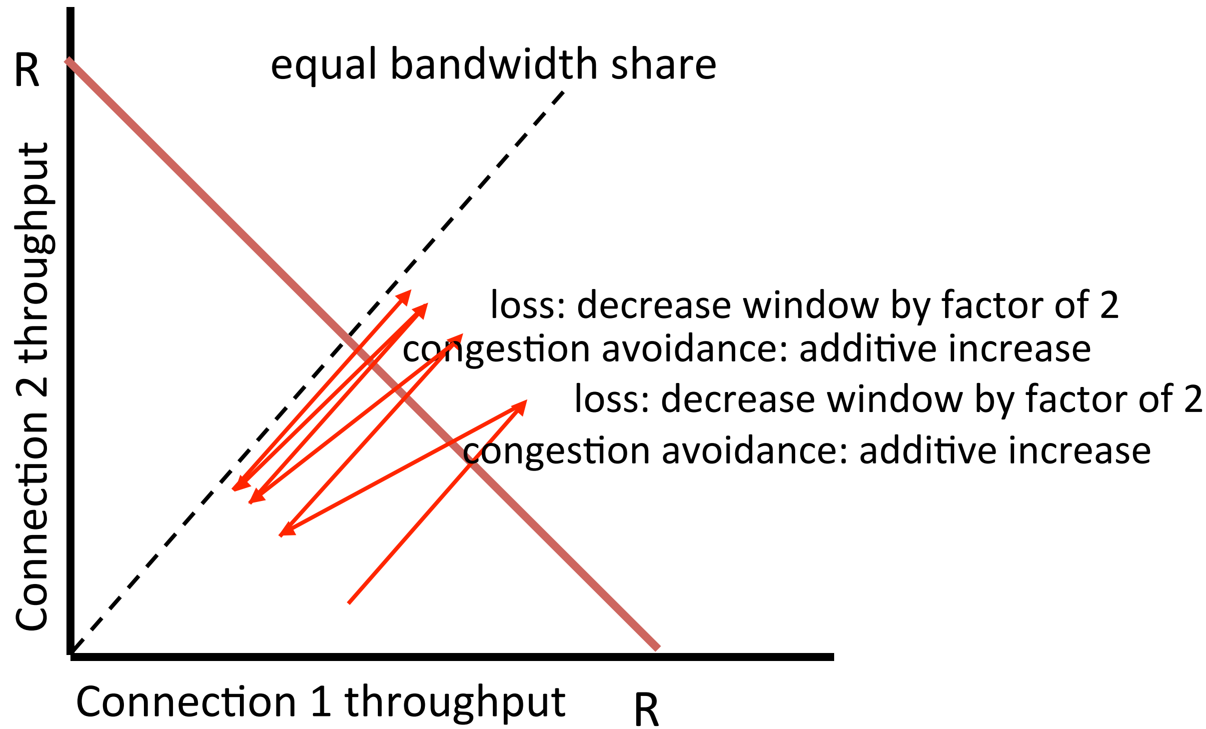
If K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- Multiplicative decrease decreases throughput proportionally

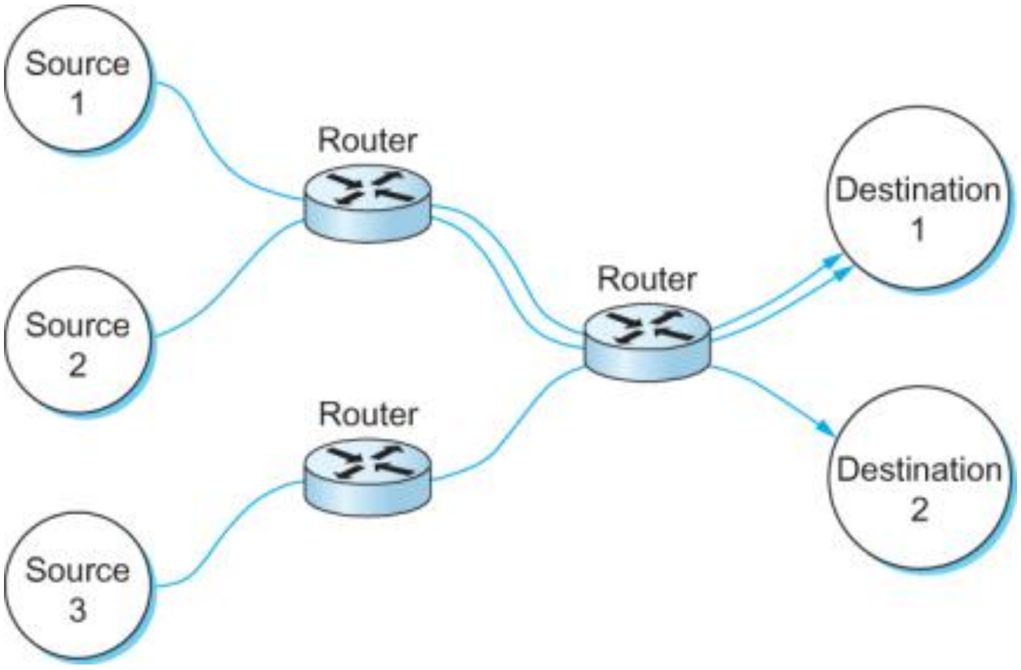


Cheating

- Not everybody plays fair:
  - Run multiple TCP connections in parallel
  - Change the TCP implementation
    - Starts your TCP connection off with > 1 MSS
  - Use a protocol without congestion control (e.g. UDP)
  - Good guys slow down to make way so others can have unfair share of bandwidth
- Possible solutions?
  - Routers detect cheating and drop excess traffic
  - Fair queuing

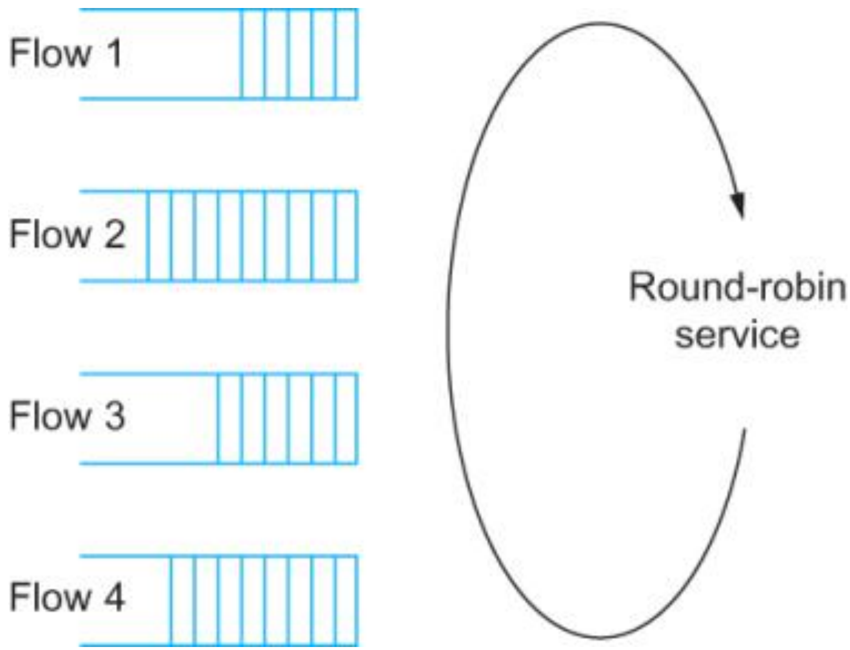
Network flows

- Connection flows
  - IP network is connectionless
  - Datagrams really not independent
  - Stream of datagrams between two hosts
  - Routers can infer current flows, "soft state"



Fair queuing

- Use flows to determine scheduling
  - Prevent hosts from hogging all the router resources
  - Important if hosts don't implement host-based congestion control (e.g. TCP congestion control)
  - Each flow gets its own queue, served round-robin



Wireless networks

- TCP congestion control uses packet loss as signal
  - Wireless/satellite links = high error rate
  - TCP may mistake bit errors as congestion
- Possible solutions:
  - Link layer acknowledgements and retransmission
  - Forward error correction
  - Split connection into wireless/wired segments
  - Use other signals than packet loss: increasing RTT

TCP splitting

- Optimize cloud-based services
  - e.g. Web search, e-mail, social networks
  - Give illusion of operating locally (i.e. low latency)
  - But: data center may be a long way and speed of light is a constant + new connection subject to TCP slow-start
- TCP splitting
  - Deploy front-end servers near to users
    - e.g. Google's "enter-deep" clusters at access ISPs
  - Client make TCP connection to front-end server, small RTT
  - Front-end maintains persistent connection to back-end with large congestion window

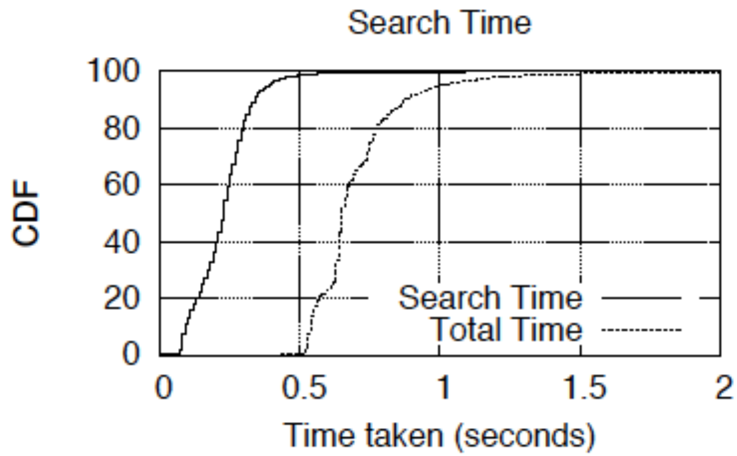


Fig. 2. CDF of response time of 200K search queries by popular search engine for search reply

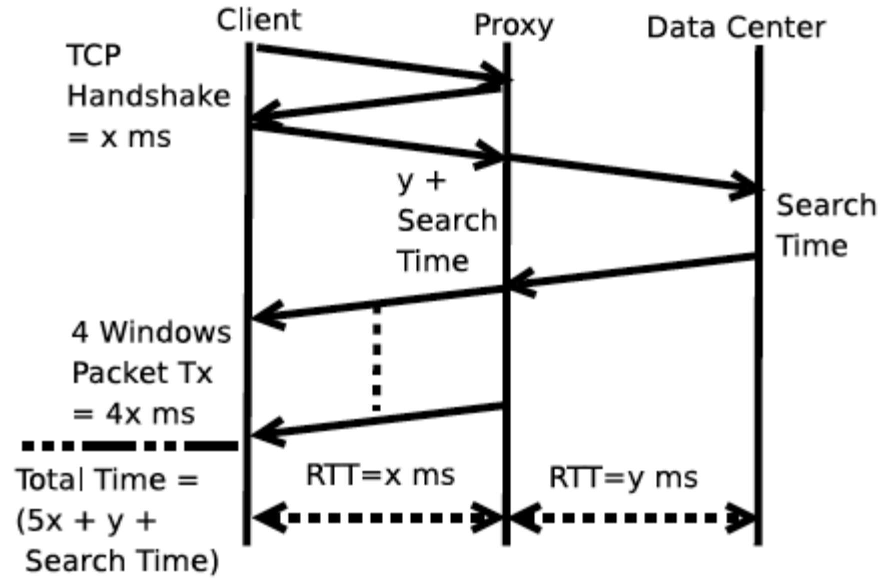
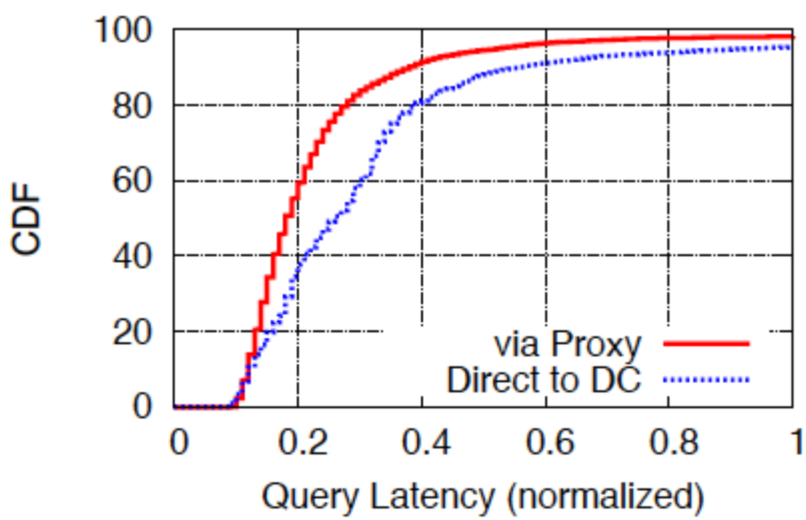


Fig. 3. TCP packet exchange diagram between an HTTP client and a search server with a proxy between them.



http://research.microsoft.com/en-us/um/people/chengh/papers/apollo10.pdf

Fig. 6. Gain of TCP Splitting

Chapter 3 summary

- Principles behind transport layer services:
  - Multiplexing, demultiplexing
  - Reliable data transfer
  - Flow control
  - Congestion control
- Instantiation in the Internet
  - UDP
  - TCP

<u>Next:</u>

- Leaving the network edge (application, transport layers)
- Into the network core!