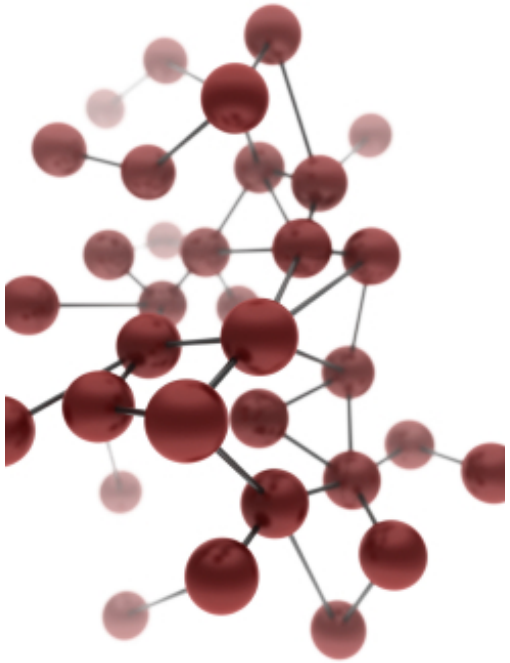


Lecture 8: TCP

Upcoming Schedule

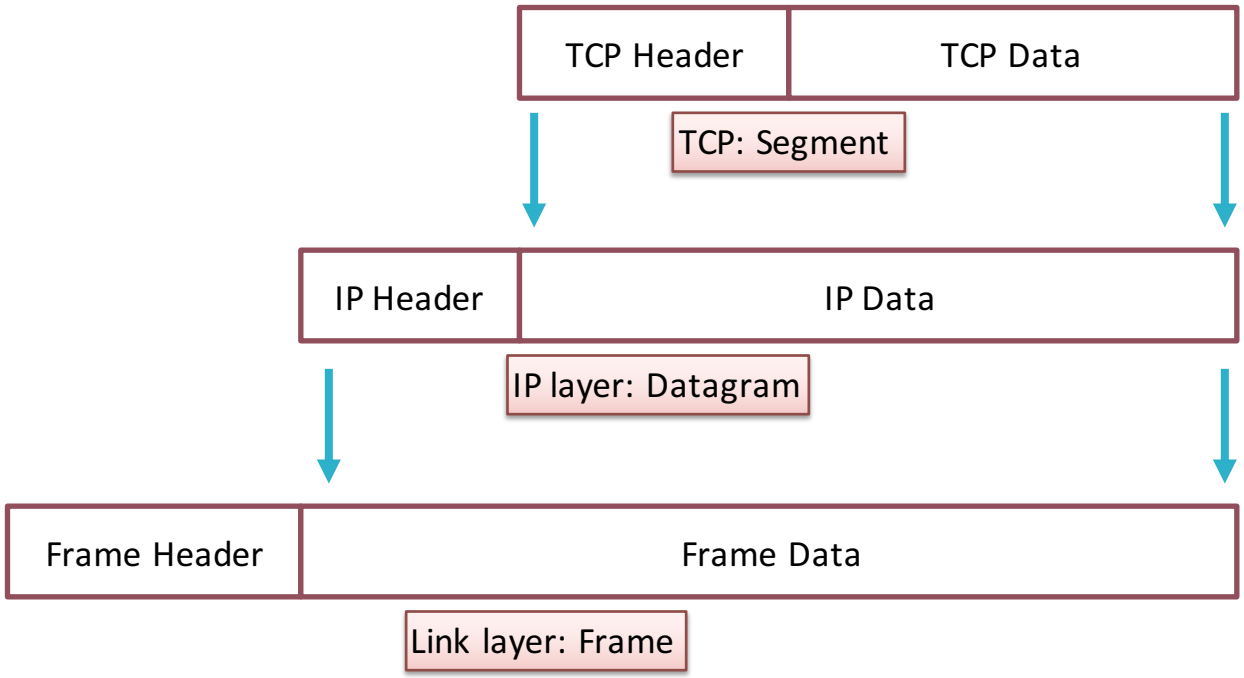
M	10/29	Congestion Control
W	10/31	DNS
M	11/5	TBD
W	11/7	Remote Procedure Calls
M	11/12	*In class Midterm*
W	11/14	Mobile Networking

- Mid-term I: Closed book, closed notes, allow 1 page of notes in A4 paper
- Project 2: multi-thread FTP, will before wed, due 11/19 (Monday)
- Project 3: something about a crawler

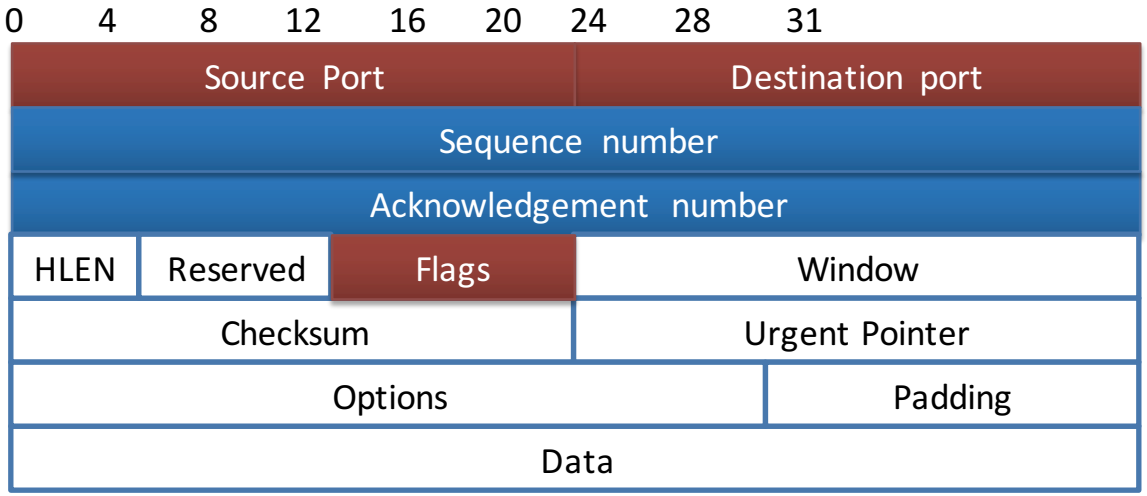
Transmission Control Protocol

- TCP, as UDP, provides the port abstraction
- Allows two nodes to establish a virtual circuit
  - Identified by source IP, destination IP, source port, destination port
  - Virtual circuit composed of two streams
    - full-duplex connection
- IP address/port # pair is sometimes called a socket (and the two streams are called a socket pair)

TCP Encapsulation

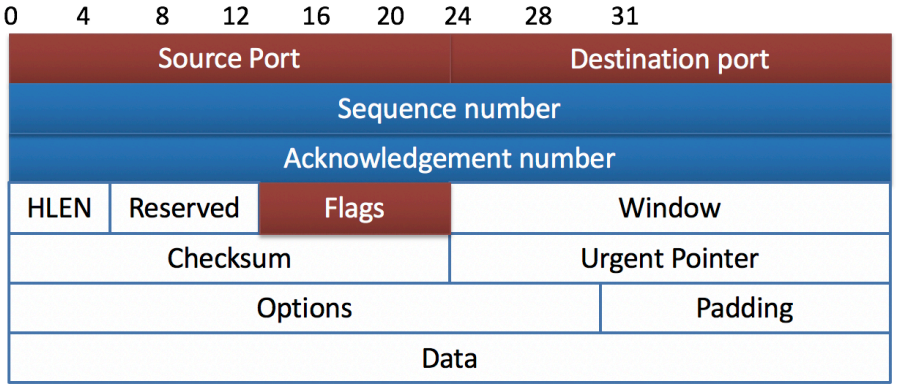


TCP Segment


0
4
8
12
16
20
24
28
31
Source Port
Destination port
Sequence number
Acknowledgement number
HLEN
Reserved
Flags
Window
Checksum
Urgent Pointer
Options
Padding
Data

TCP Header

- Sequence number, acknowledgement, and advertised window
  - used by sliding-window based flow control
- Flags:
  - SYN, FIN – establishing/terminating a TCP connection
  - ACK – set when Acknowledgement field is valid
  - URG – urgent data; Urgent Pointer says where non-urgent data starts
  - PUSH – don't wait to fill segment
  - RESET – abort connection



TCP Seq/Ack Numbers

- Sequence # specifies position of the segment data in communication stream
  - SYN=13423: segment payload contains data from byte 13423 to byte 13458
- Acknowledgment # specifies the position of **next** byte expected from communication partner
  - ACK = 16754: I have received correctly up to byte 16753 in the stream, I expect the next byte to be 16754
- These # used to manage retransmission of lost segments, duplication, flow control

What Should the Receiver ACK?

1. ACK every packet, giving its sequence number

2. Use *negative ACKs* (NACKs), indicating which packet did not arrive

Default configuration

3. Use *cumulative ACK*, where an ACK for number n implies ACKS for all k < n

4. Use *selective ACKs* (SACKs), indicating those that did arrive, even if not in order

Key Functions in TCP

- Flow control
  - Make sure sender sends at rate receiver can handle
- Reliable data transfer
  - Via packet retransmissions
- Congestion control
  - Make sure that the network delivers the packets to the receiver
  - Detects network congestion via lost packets
  - (Initially not included in TCP)
- Connection setup
  - 3 way handshake

TCP Flow Control

How Much Can you Send?

TCP Flow Control Window

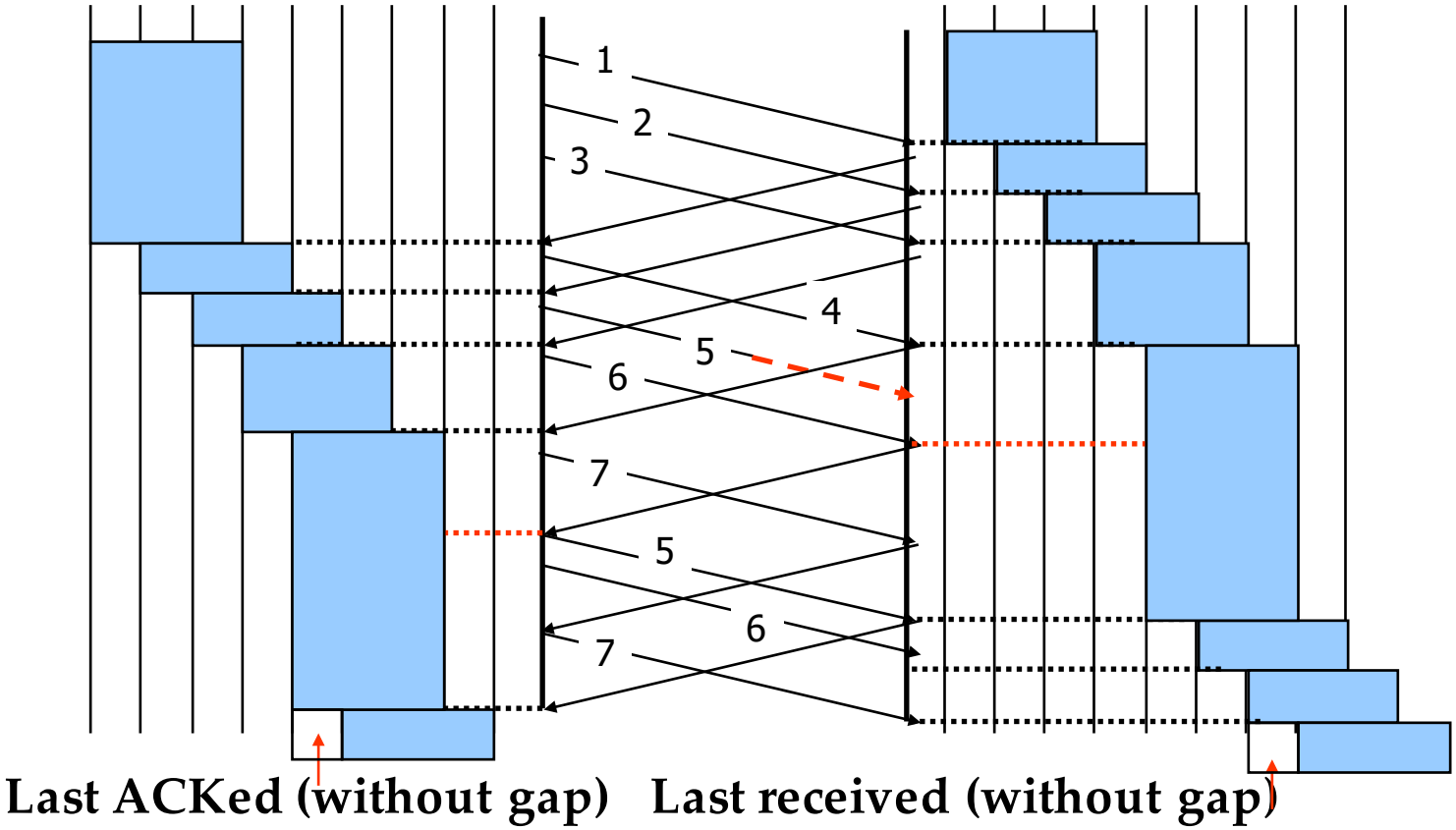
- Make sure receiving end can handle data
  - TCP window used to perform flow control
  - Negotiated end-to-end, **with no regard to network**
- Ends must ensure that no more than W packets are in flight
  - Receiver ACKs packets
  - When sender gets an ACK, it knows packet has arrived
- Segment accepted IFF sequence # is inside window:

```
ack # < sequence # < (ack # + window)
```

- Window size can change **dynamically** to adjust the amount of information sent by the sender

Sliding Window for Flow Control

Window size of 3



Observations

- Throughput is ~ w/RTT RTT: Round Trip Time
  - Longer RTT, longer pipe
- 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

TCP Reliability

TCP Reliability/Error Recovery

- Must retransmit packets that were dropped
- To do this efficiently
  - Keep transmitting whenever possible
  - Detect dropped packets and retransmit quickly
- Requires:
  - Timeouts (with good timers)
  - Other hints that packet were dropped

Hints

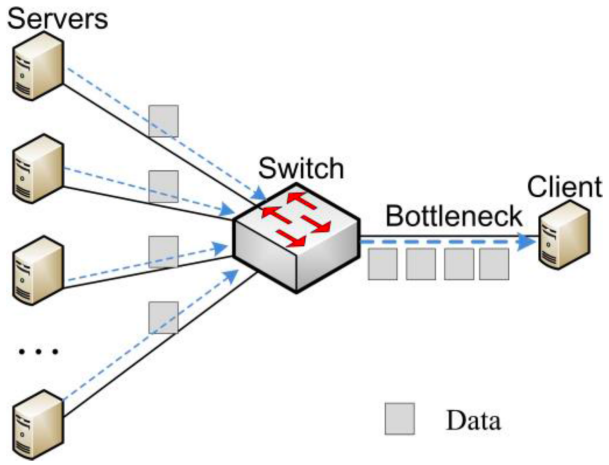
- When should I suspect a packet was dropped?
- When time out (NO ACKs received within a given time)
- When I receive several duplicate ACKs
  - Receiver sends an ACK whenever a packet arrives
  - ACK indicates seq. no. of last received consecutively received packet
  - Duplicate ACKs indicates missing packet

TCP Congestion Control

Again, How much can you send?

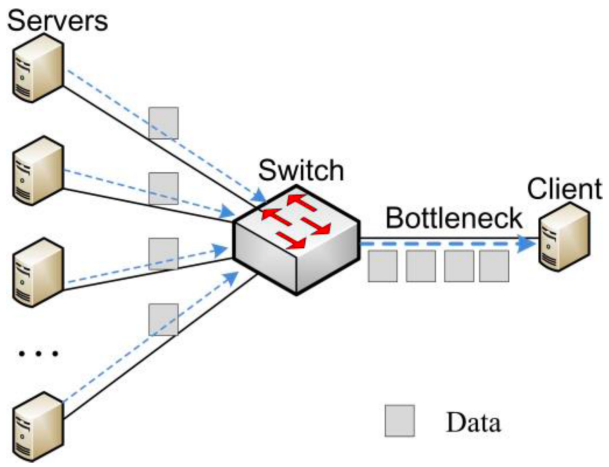
TCP Congestion Control

- Can the network handle the rate of data?
- Senders increase packet transmission rate until packets are dropped
- TCP treats packet loss as a sign of network congestion and slows down the packet transmission
- Control packet send rate via congestion window



If you don't do congestion control

- Assume the network is congested (a router drops packets)
- Receiver did not get the packet
- Sender retransmit
- Still receiver did not get the packet
- Sender retransmit
- ……
- → Congestion Collapse: Nothing is going through the router

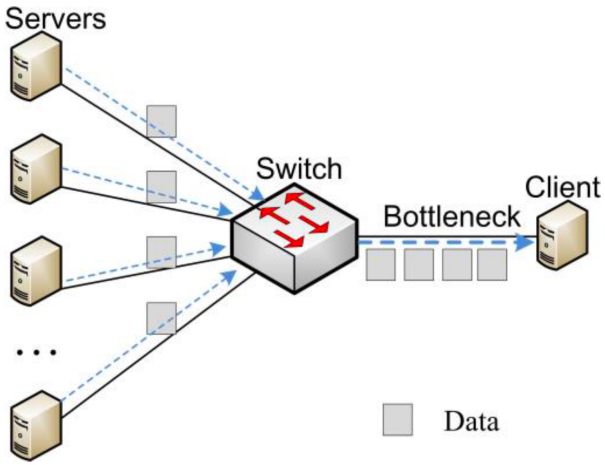


Basic Concept of Congestion Control

- Slow down
  - If the network is really congested, slow down the packet transmission
- But how do you know the network is really congested?
- And how much do you slow down?

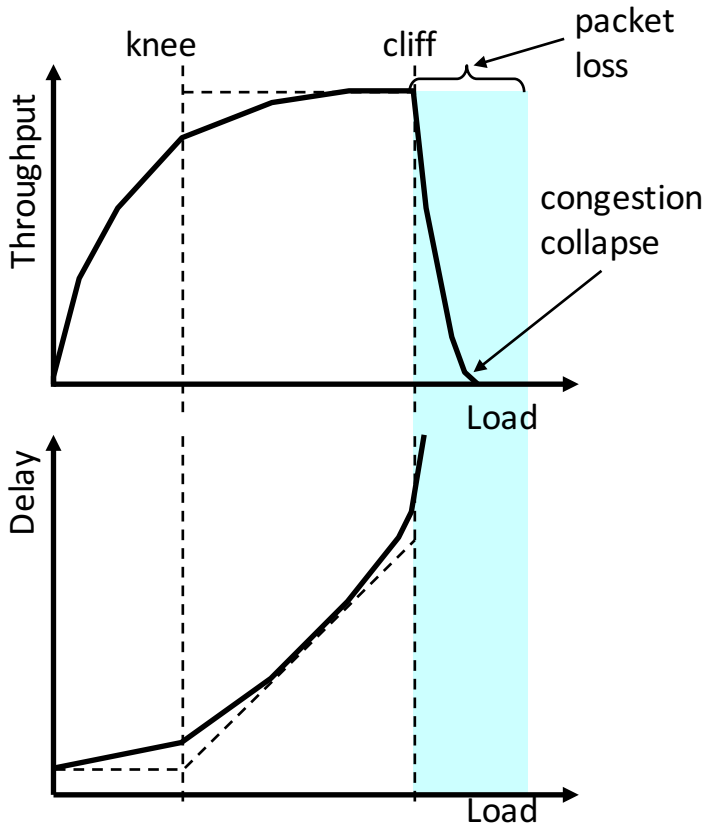
TCP Congestion Control

- Determine congestion window end-to-end
  - TCP is guessing about the state of the network
- How do we control rate of traffic: AIMD
  - Additive increase, multiplicative decrease (details later)
  - Proven to be stable and "relatively" efficient



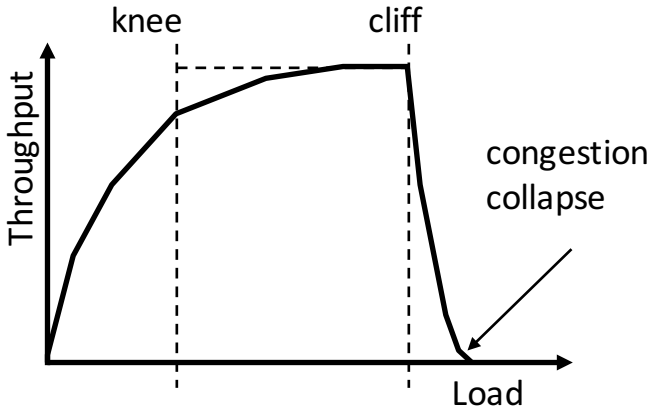
Dangers of Increasing Load

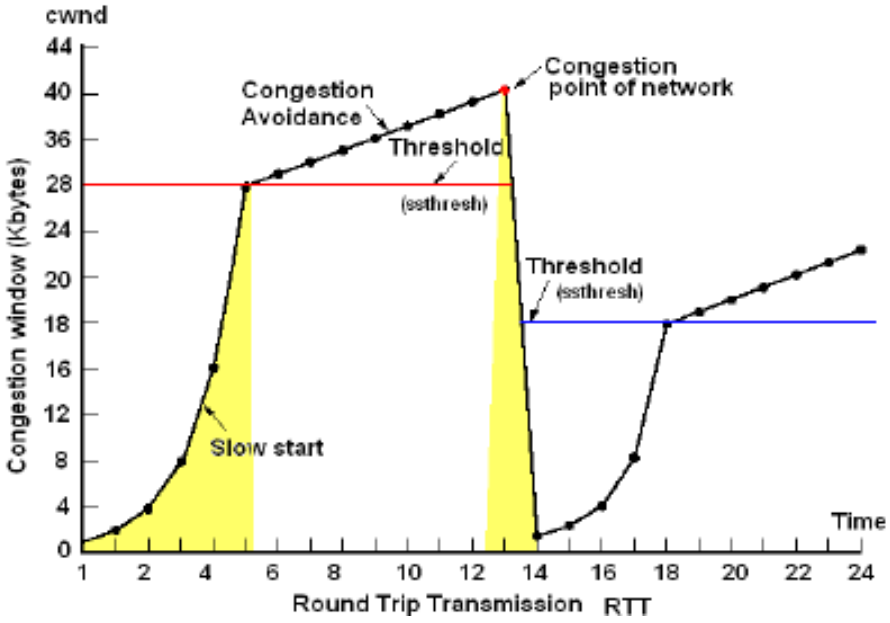
- Knee – point after which
  - Throughput increases very slow
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to decrease very fast to zero (congestion collapse)
  - Delay approaches infinity
- In an M/M/1 queue
  - Delay = 1/(1 – utilization)



Congestion Control vs. Congestion Avoidance

- Congestion control goal
  - Stay left of cliff
- Congestion avoidance goal
  - Stay left of knee
- Ultimate Goal: operate near the knee
  - Detect when reaching the knee, stay there!
  - But how do you really do this?





Two Key Issues

- How to detect congestion:
  - Duplicated ACKs indicating packet loss
  - RTT too large

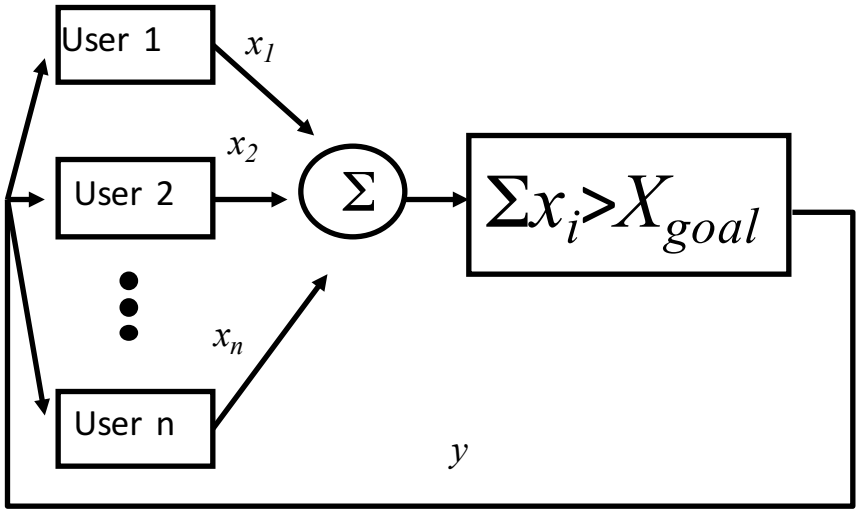
- How to partition bandwidth among users?
  - How do I control my packet transmission rate??

Modeling the TCP Congestion Control

- Can the network handle the rate of data?
- Determined end-to-end, but TCP is making guesses about the state of the network
- Modeling its behavior: Good science vs. great engineering

Control System Model [CJ89]

- Simple, yet powerful model
- **Explicit** binary signal of congestion



Possible Choices

$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & increase \\ a_D + b_D x_i(t) & decrease \end{cases}$$

- Multiplicative increase, additive decrease
  - $a_I=0, b_I>1, a_D<0, b_D=1$
- Additive increase, additive decrease
  - $a_I>0, b_I=1, a_D<0, b_D=1$
- Multiplicative increase, multiplicative decrease
  - $a_I=0, b_I>1, a_D=0, 0<b_D<1$
- Additive increase, multiplicative decrease
  - $a_I>0, b_I=1, a_D=0, 0<b_D<1$
- Which one?

Goals of the System

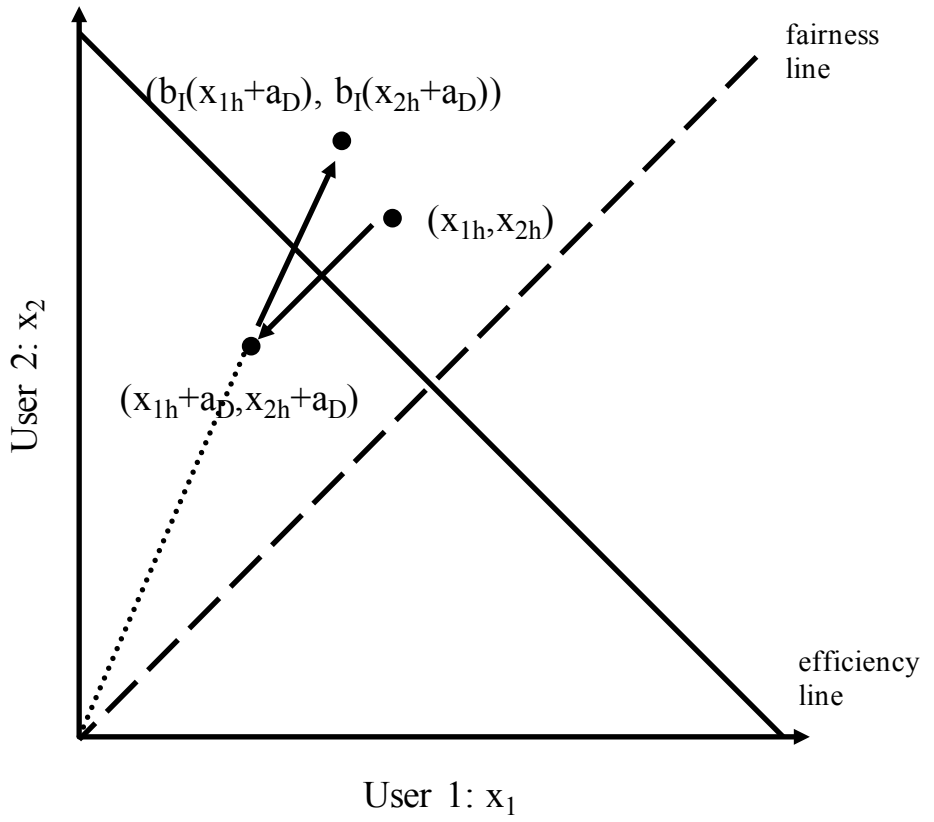
- Efficiency
  - Proximity of total throughput to the cliff
- Fairness
  - MaxMin fairness

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

- Distributed control scheme
  - Binary feedback from endpoint (congest/no-congest)
- Convergence
  - No pure convergence with binary feedback
  - Instead: equilibrium = oscillations around ideal state

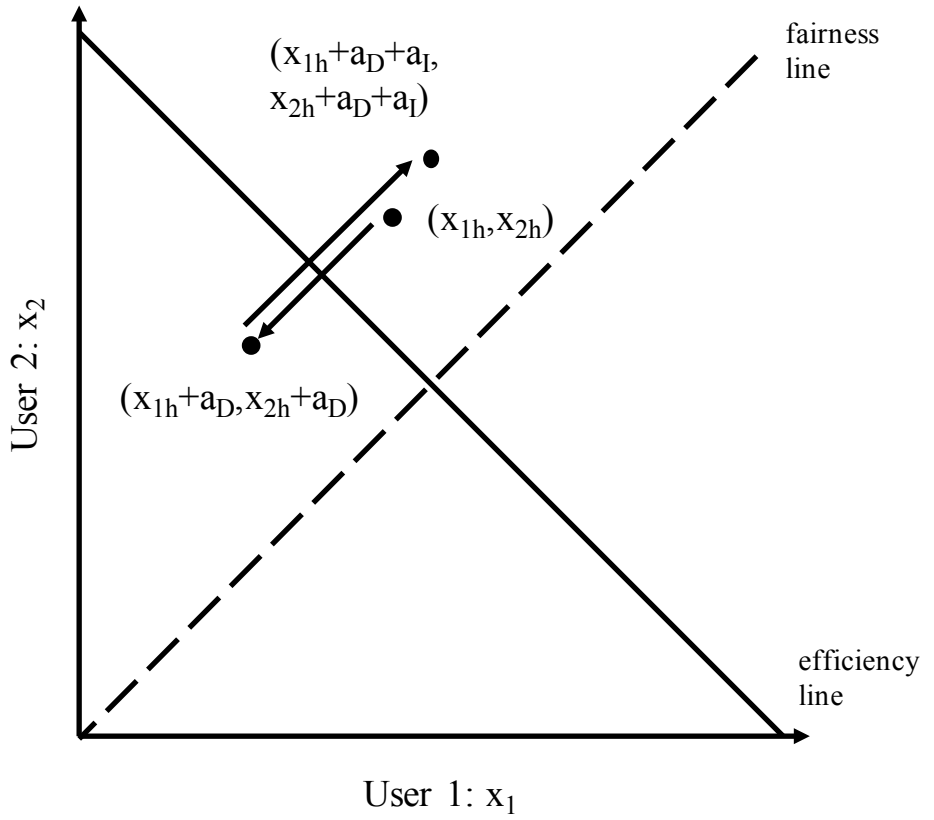
Multiplicative Increase, Additive Decrease

- Is not stable: veers away from fairness



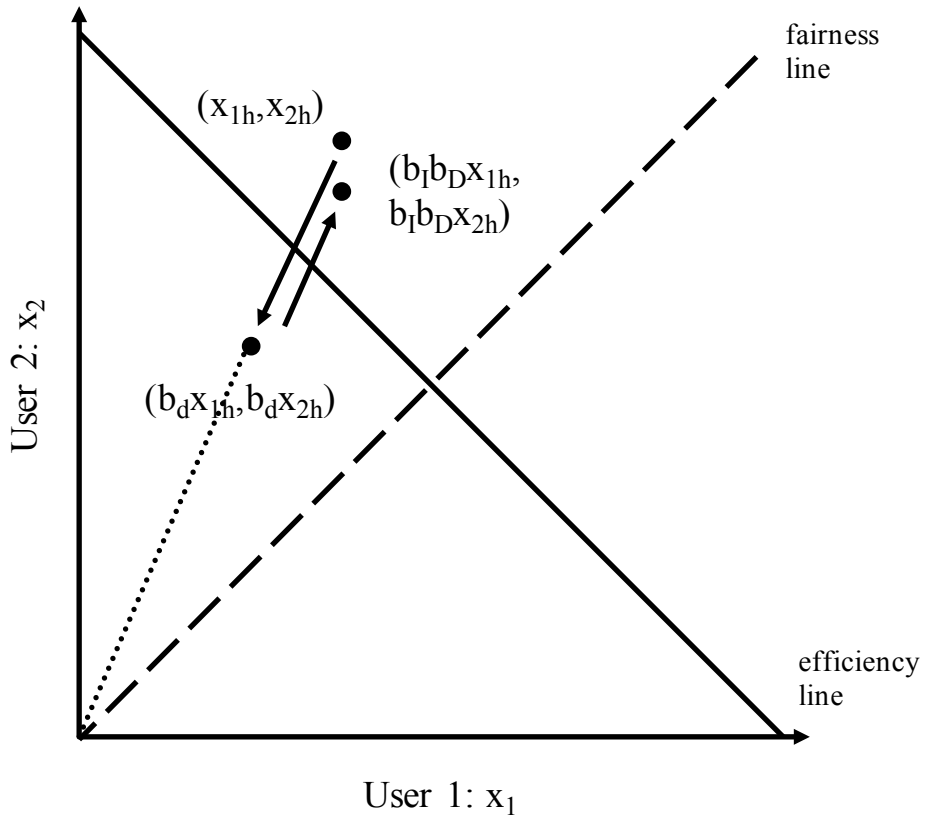
Additive Increase, Additive Decrease

- Reaches stable cycle, but does not converge to fairness



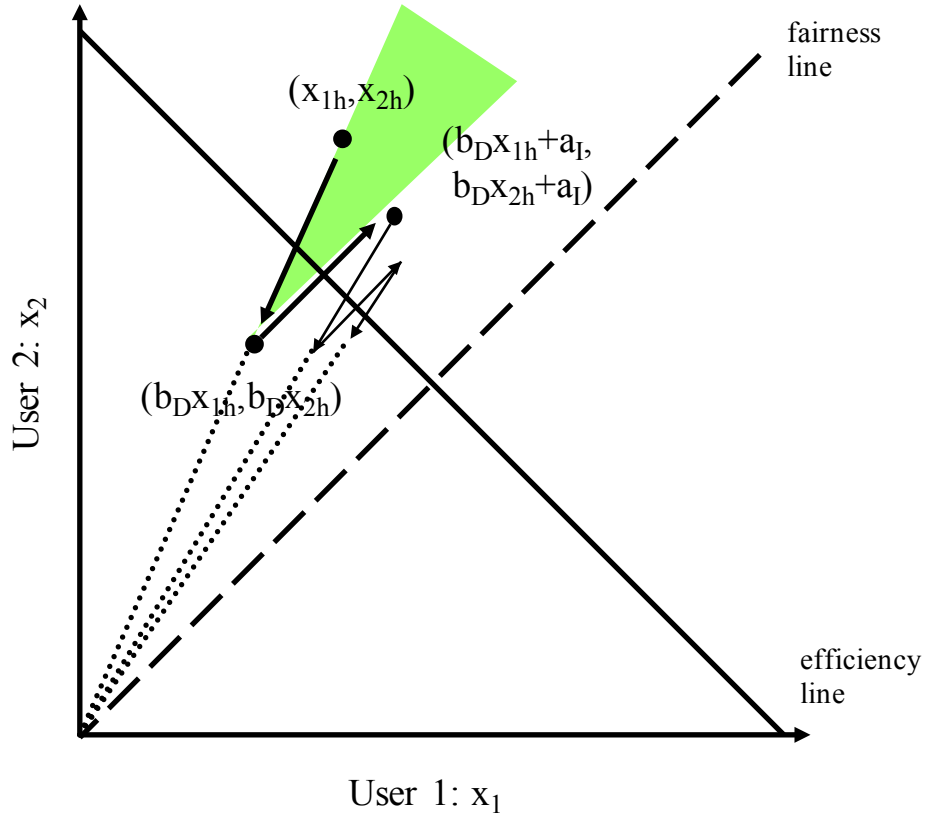
Multiplicative Increase, Multiplicative Decrease

- Converges to stable cycle, but is not fair



Additive Increase, Multiplicative Decrease

- Converges to stable and fair cycle



AIMD

- Converges to efficiency and fairness
- Easily deployable
- Fully distributed
- No need to know full state of the system (e.g. number of users, bandwidth of links)

- Forming the basis for TCP

Modeling is Important

- Critical to understanding complex systems
  - [CJ89] enabled the Internet to grow beyond 1989
    - 10^6 increase of bandwidth, 1000x increase in number of users
  - Criteria for good models
    - Realistic
    - Simple
      - Easy to work with
      - Easy for others to understand
    - Realistic, complex model → useless
    - Unrealistic, simple model → can teach something about best case, worst case, etc.

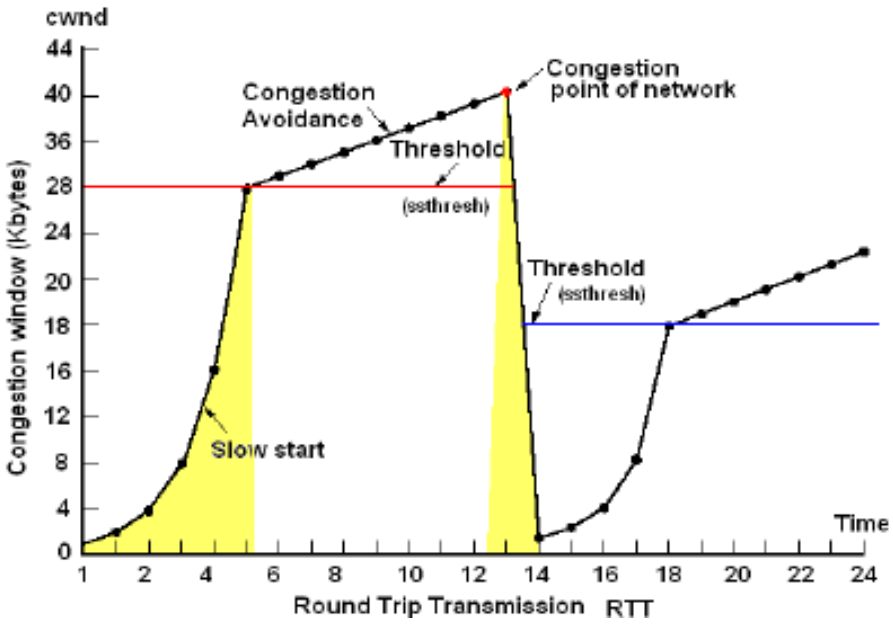
TCP Congestion Control (Practical Actions)

- Maintains three variables:
  - cwnd: congestion window
  - flow_win: flow window; receiver advertised window
  - Ssthresh: threshold size (used to update cwnd)

- For sending, use: win = min(flow_win, cwnd)

TCP: Slow Start

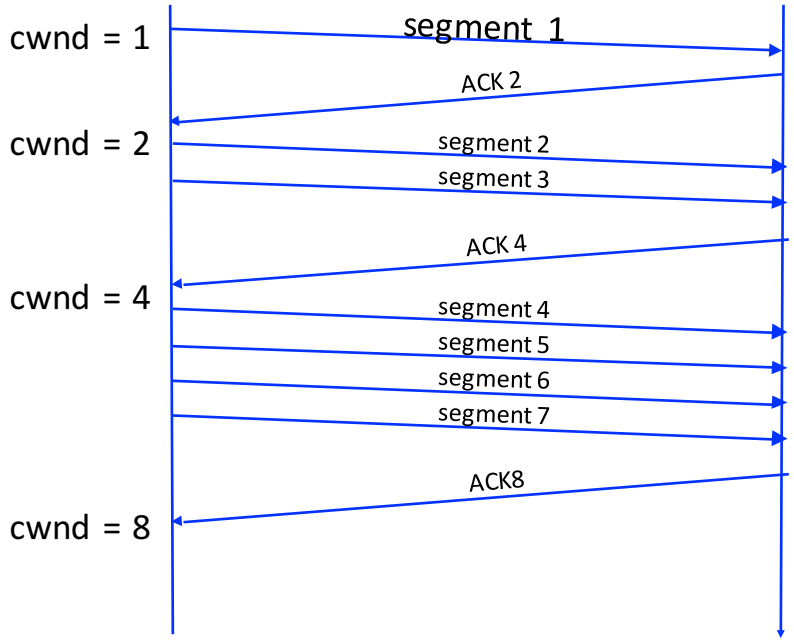
- Goal: ramp up speed quickly
- Upon starting (or restarting):
  - Set cwnd =1
  - Each time a segment is acknowledged increment cwnd by one (cwnd++).
- Slow Start is not actually slow
  - cwnd increases exponentially



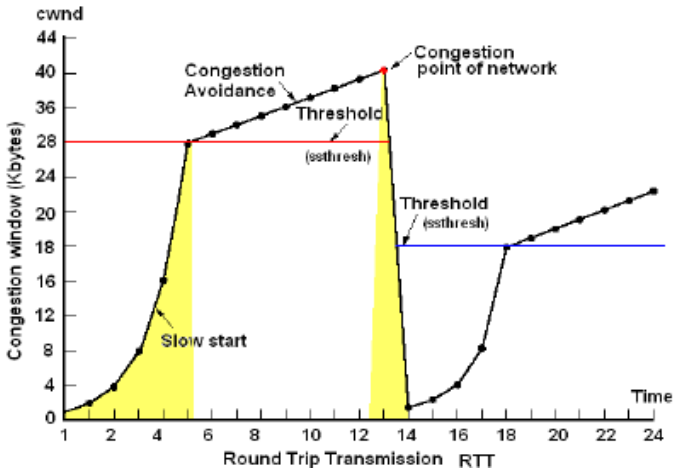
Slow Start Example

- Congestion window size grows very rapidly

- TCP slows down the increase of cwnd when cwnd >= ssthresh



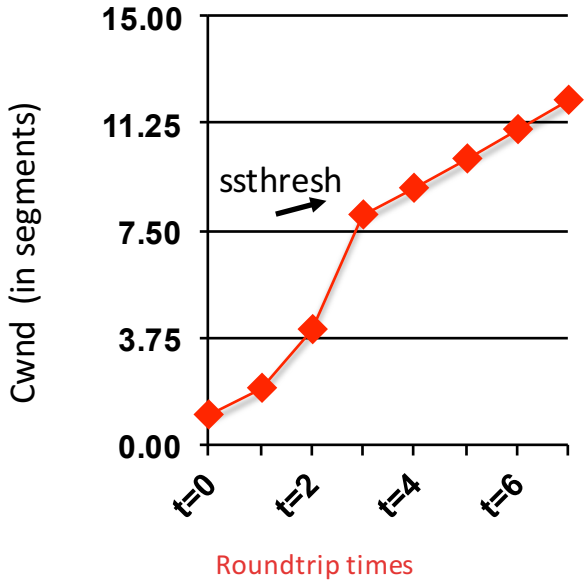
Congestion Avoidance

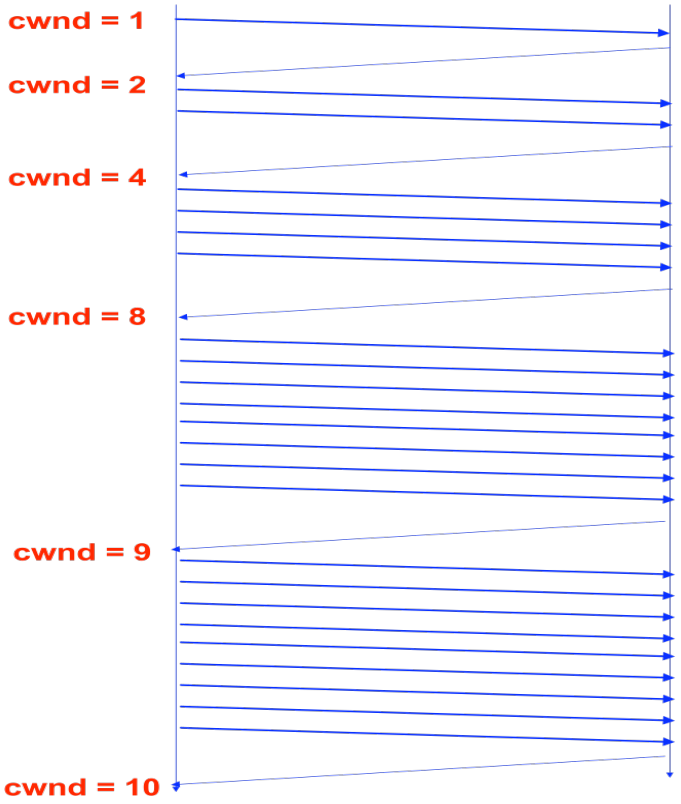


- Slow down "Slow Start"
- ssthresh is lower-bound guess about location of knee
- If cwnd > ssthresh then
  each time a segment is acknowledged
  increment cwnd by 1/cwnd (cwnd += 1/cwnd).
- So cwnd is increased by one only if all segments have been acknowledged.

Slow Start/Congestion Avoidance Example

- Assume that ssthresh = 8



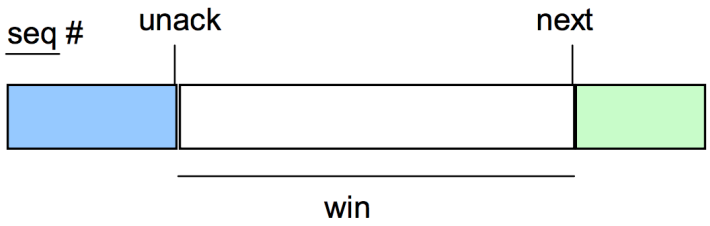


Putting Everything Together

```
Initially:
    cwnd = 1;
    ssthresh = infinite;
New ack received:
    if (cwnd < ssthresh)
        /* Slow Start*/
        cwnd = cwnd + 1;
    else
        /* Congestion Avoidance */
        cwnd = cwnd + 1/cwnd;
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2;
    cwnd = 1;
```

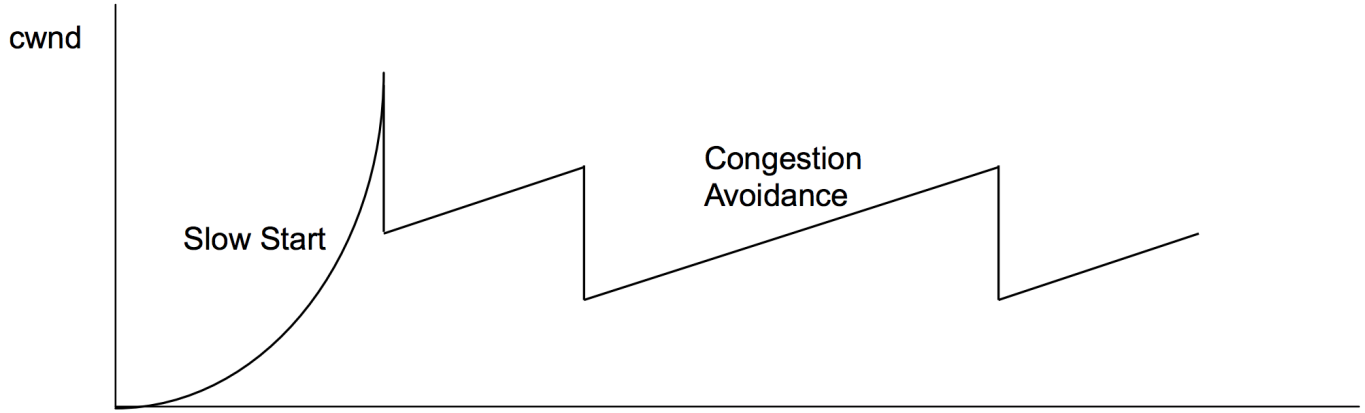
```
while (next < unack + win)
    transmit next packet;

where   win = min(cwnd,
                  flow_win);
```



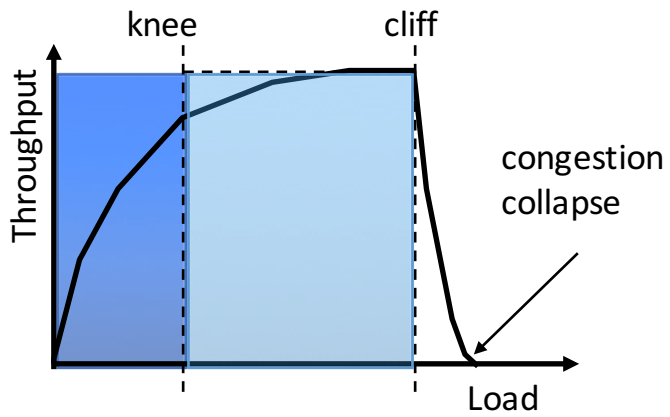
Fast Retransmit and Fast Recovery

- Don't wait for Window to drain
- Resend a (lost) TCP segment after 3 duplicate ACKs
  - Prevent expensive timeouts
- No need to slow start again



Cong. Control vs. Cong. Avoidance

- Congestion control goal
  - Stay left of cliff
- Congestion avoidance goal
  - Stay left of knee



How to set up TCP connection

TCP Flags

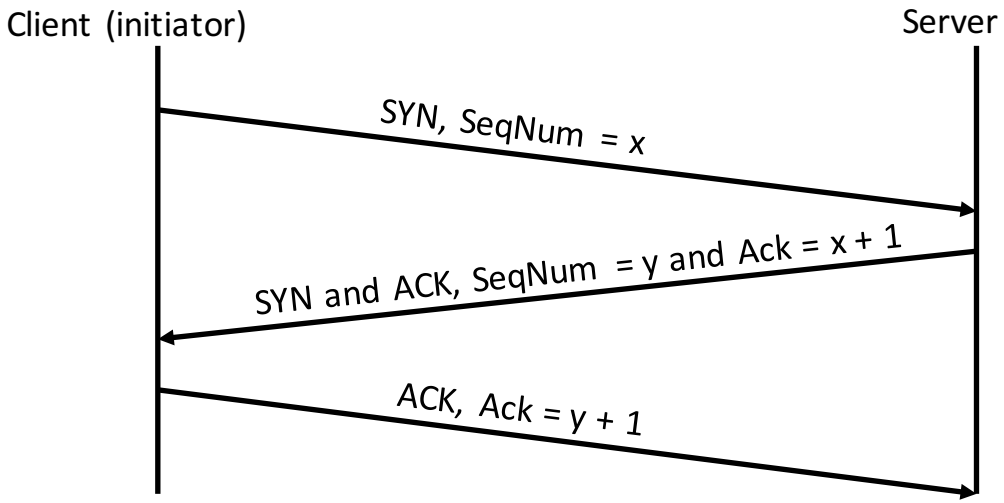
- Flags are used to manage the establishment and shutdown of a virtual circuit
  - SYN: request for the synchronization of syn/ack numbers (used in connection setup)
  - ACK: states the acknowledgment number is valid (all segment in a virtual circuit have this flag set, except for the first one)
  - FIN: request to shutdown one stream
  - RST: request to immediately reset the virtual circuit
  - URG: states that the Urgent Pointer is valid
  - PSH: request a "push" operation on the stream (that is, the stream data should be passed to the user application as soon as possible)

TCP Virtual Circuit: Setup

- A server, listening to a specific port, receives a connection request from a client: The segment containing the request is marked with the SYN flag and contains a random initial sequence number s_c
- Server answers with a segment marked with both the SYN and ACK flags and containing
  - an initial random sequence number s_s
  - $s_c + 1$ as the acknowledgment number
- The client sends a segment with the ACK flag set and with sequence number $s_c + 1$ and acknowledgment number $s_s + 1$

TCP Connection Establishment

- Three-way handshake
  - Goal: agree on a set of parameters: the start sequence number for each side



TCP: Three-way Handshake

Client

13987	23
seq: 6574	ack: 0
SYN:1	ACK:0 FIN:0

→

23	13987
seq: 7611	ack: 6575
SYN:1	ACK:1 FIN:0

←

13987	23
seq: 6575	ack: 7612
SYN:0	ACK:1 FIN:0

→

Server

TCP Virtual Circuit: Data Exchange

- A partner sends in each packet the acknowledgment of the previous segment and its own sequence number increased of the number of transmitted bytes
- A partner accepts a segment of the other partner only if the numbers are inside the transmission window
- An empty segment may be used to acknowledge the received data

TCP Virtual Circuit: Data Exchange

Client

13987	23
seq: 6575	ack: 7612
SYN:0	ACK:1	FIN:0
25 bytes	

23	13987
seq: 7612	ack: 6600
SYN:0	ACK:1	FIN:0
30 bytes	

13987	23
seq: 6600	ack: 7642
SYN:0	ACK:1	FIN:0

Server

TCP Virtual Circuit: Shutdown

- One of the partners, say A, can terminate its stream by sending a segment with the FIN flag set
- The other partner, say B, answers with an ACK segment
- From that point on, A will not send any data to B: it will just acknowledge data sent by B
- When B shutdowns its stream the virtual circuit is considered closed

TCP Virtual Circuit: Shutdown

Client

13987	23
seq: 6983	ack: 8777
SYN:0	ACK:1	FIN:1

23	13987
seq: 8777	ack: 6984
SYN:0	ACK:1	FIN:0
30 bytes	

23	13987
seq: 8807	ack: 6984
SYN:0	ACK:1	FIN:1

13987	23
seq: 6984	ack: 8808
SYN:0	ACK:1	FIN:0

Server

TCPDump

TCPDump/wireshark: Understanding the Network

- TCPDump: tool that analyzes traffic on a network segment, or packet sniffer
  - One of the most used/most useful tools
  - Based on libpcap, which provides a platform-independent library and API to perform traffic sniffing

- TCPDump and libpcap are both available at http://www.tcpdump.org
  - Allows one to specify an expression that defines which packets have to be printed

  - Detailed tutorials online: e.g

  - https://danielmiessler.com/study/tcpdump/
  - https://www.poftut.com/tcpdump-tutorial-with-examples/

TCPDUMP(8) System Manager's Manual TCPDUMP(8)

NAME

tcpdump - dump traffic on a network

SYNOPSIS

tcpdump [**-AbdDefhHIJKlLnNOpqRStuUvxX#**] [**-B** buffer_size]
[**-c** count]
[**-C** file_size] [**-G** rotate_seconds] [**-F** file]
[**-i** interface] [**-j** tstamp_type] [**-m** module] [**-M** secret]
[**--number**] [**-Q** in|out|inout]
[**-r** file] [**-V** file] [**-s** snaplen] [**-T** type] [**-w** file]
[**-W** filecount]
[**-E** spi@ipaddr algo:secret,...]
[**-y** datalinktype] [**-z** postrotate-command] [**-Z** user]
[**--time-stamp-precision=**tstamp_precision]
[**--immediate-mode**] [**--version**]
[expression]

DESCRIPTION

Tcpdump prints out a description of the contents of packets on a network interface that match the boolean expression; the description is preceded by a time stamp, printed, by default, as hours, minutes, seconds, and fractions of a second since midnight. It can also be run with the **-w** flag, which causes it to save the packet data to a file for later analysis, and/or with the **-r** flag, which causes it to read from a saved packet file rather than to read packets from a network interface (please note tcpdump is protected via an enforcing **apparmor**(7) profile in Ubuntu which limits the files tcpdump may access). It can also be run with the **-V** flag, which causes it to read a list of saved packet files. In all cases, only packets that match expression will be processed by tcpdump.