CS4220
Computer Networks

Lecture 7 The Transport Layer

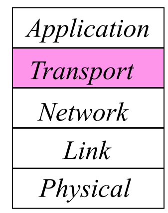
Dr. Xiaobo “Charles” Zhou
Department of Computer Science

Transport Layer
Chapter 6

- Transport Service
- Elements of Transport Protocols
- Congestion Control
- Internet Protocols – UDP
- Internet Protocols – TCP
- Performance Issues
- Delay-Tolerant Networking

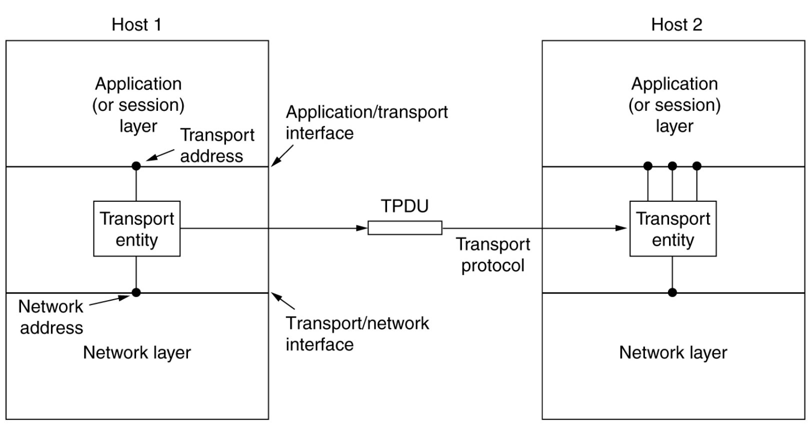
The Transport Layer

- Responsible for delivering data across networks with the desired reliability or quality



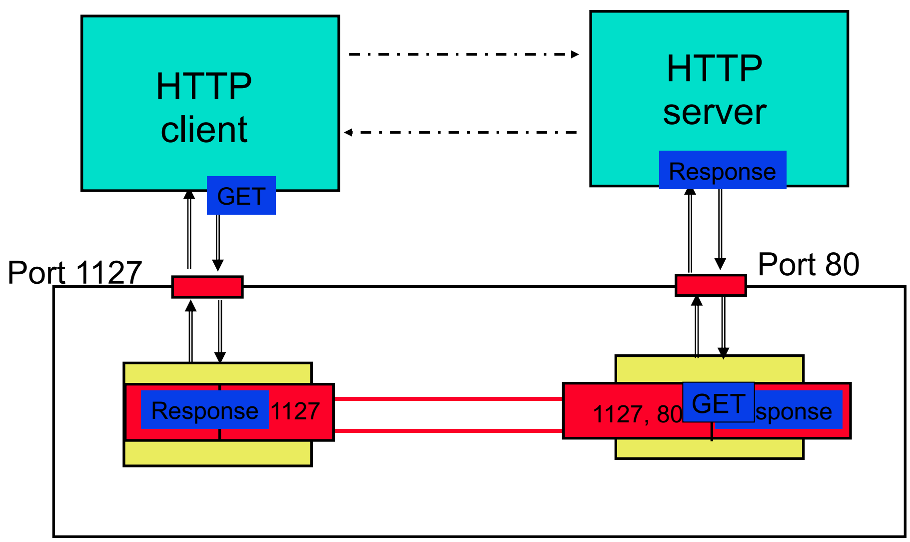
Services Provided to the Upper Layers

- Goal: to provide efficient, reliable, and cost-effective service to its users, processes in the application layer
  - Transport entity: hardware and/or software within the layer

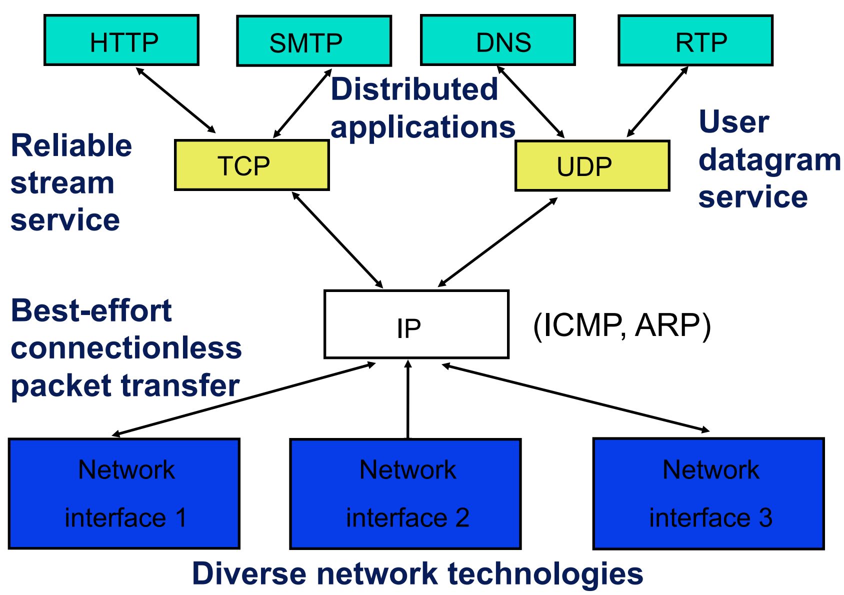


The network, transport, and application layers.

Example: HTTP Uses Service of TCP



TCP/IP Protocol Suite



UDP (User Datagram Protocol)

- UDP is a transport layer protocol
- Provides *best-effort datagram service* between two processes in two computers across the Internet
- Port numbers distinguish various processes in the same machine
- UDP is *connectionless*
- Datagram is sent immediately
- Quick, simple, but not reliable

TCP (Transmission Control Protocol)

- TCP is a transport layer protocol
- Provides *reliable byte stream service* between two processes in two computers across the Internet
- Sequence numbers keep track of the bytes that have been transmitted and received
- Error detection and retransmission used to recover from transmission errors and losses
- TCP is *connection-oriented*: the sender and receiver must first establish an association and set initial sequence numbers before data is transferred
- Connection ID is specified uniquely by

(send port #, send IP address, receive port #, receiver IP address)

Internet Names & Addresses

Internet Names

- Each host a a unique name
  - Independent of physical location
  - Facilitate memorization by humans
  - Domain Name
  - Organization under single administrative unit
- Host Name
  - Name given to host computer
- User Name
  - Name assigned to user
  - leongarcia@cs.utoronto.ca

Internet Addresses

- Each host has globally unique *logical* 32 bit IP address
- Separate address for each physical connection to a network
- Routing decision is done based on destination IP address
- IP address has two parts:
  - *netid* and *hostid*
  - *netid* unique
  - *netid* facilitates routing
- Dotted Decimal Notation:

  int1.int2.int3.int4

  128.100.10.13

How to resolve IP name to IP address Mapping?

Physical Addresses

- LANs (and other networks) assign physical addresses to the physical attachment to the network
- The network uses its own address to transfer packets or frames to the appropriate destination
- IP address needs to be resolved to physical address at each IP network interface
- Example: Ethernet uses 48-bit addresses
  - Each Ethernet network interface card (NIC) has globally unique Medium Access Control (MAC) or physical address
  - First 24 bits identify NIC manufacturer; second 24 bits are serial number
  - $\underbrace{\text{00:90:27}}_{\text{Intel}}$:96:68:07   12 hex numbers

Transport Layer vs. Network Layer

- Both layers provide connectionless services and connection-oriented services, why two distinct layers?
  - IPC vs. machine-to-machine
  - End host vs. routers
    - Reliability
    - Performance
    - QoS
    - programming convenience

The users have no real control over the network layer!

Transport Service Primitives

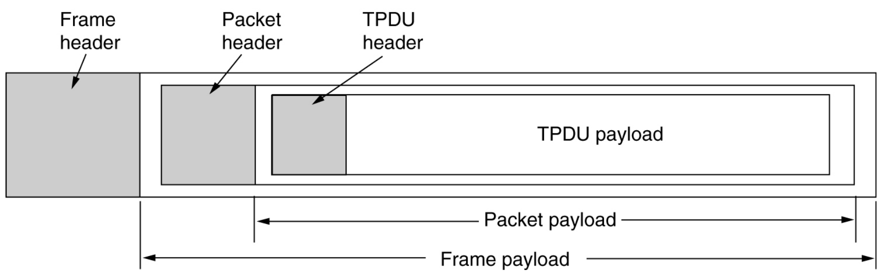
- Connectionless (datagram) service
- Connection-oriented (stream) service is three phase: establishment, data transfer, and release

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

The primitives for a simple transport service.

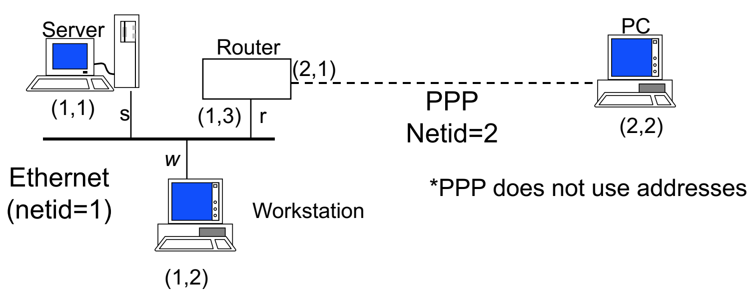
Transport Service Primitives (2)

- **TPDU (transport protocol data unit): messages sent from transport entity to transport entity.**



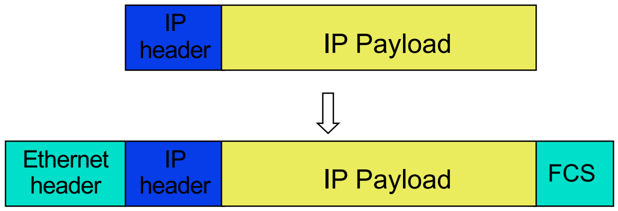
The nesting of TPDUs, packets, and frames.

Example Internet



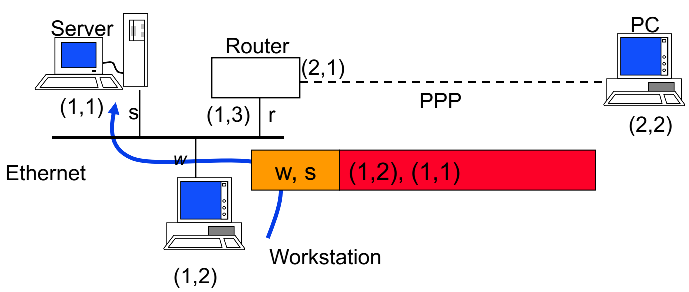
	netid	hostid	Physical address
server	1	1	s
workstation	1	2	w
router	1	3	r
router	2	1	-
PC	2	2	-

Encapsulation



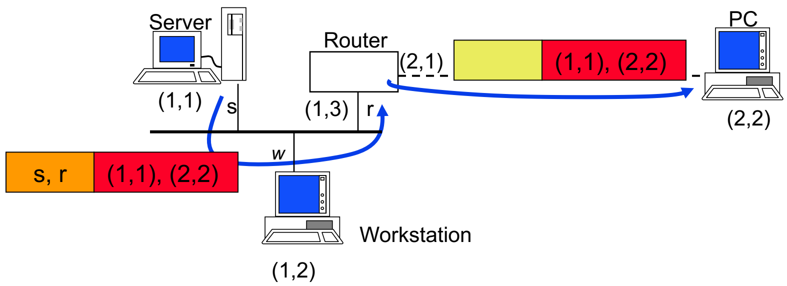
- Ethernet header contains:
  - source and destination physical addresses
  - network protocol type (e.g. IP)

IP Packet from Workstation to Server



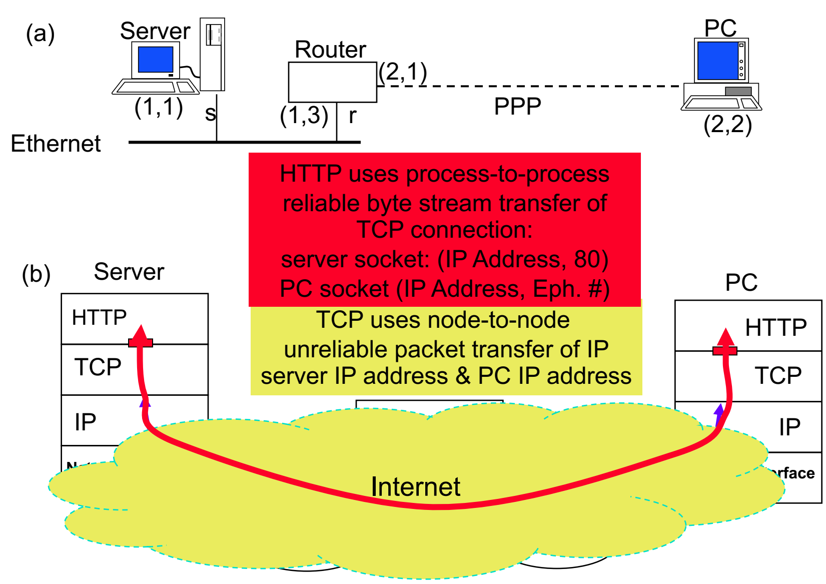
1. **IP packet has (1,2) IP address for source and (1,1) IP address for destination**
2. **IP table at workstation indicates (1,1) connected to same network, so IP packet is encapsulated in Ethernet frame with addresses w and s (by the use of ARP)**
3. **Ethernet frame is broadcast by workstation NIC and captured by server NIC**
4. **NIC examines protocol type field and delivers packet to its IP layer**

IP Packet from Server to PC



1. IP packet has (1,1) and (2,2) as IP source and destination addresses
2. IP table at server indicates packet should be sent to router, so IP packet is encapsulated in Ethernet frame with addresses s and r
3. Ethernet frame is broadcast by server NIC and captured by router NIC
4. NIC examines protocol type field and then delivers packet to its IP layer
5. IP layer examines IP packet destination address and determines IP packet should be routed to (2,2)
6. Router's table indicates (2,2) is directly connected via PPP link
7. IP packet is encapsulated in PPP frame and delivered to PC
8. PPP at PC examines protocol type field and delivers packet to PC IP layer

How the layers work together

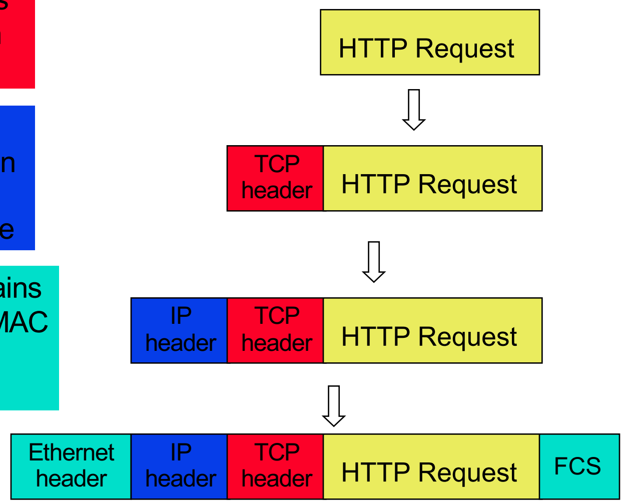


Encapsulation

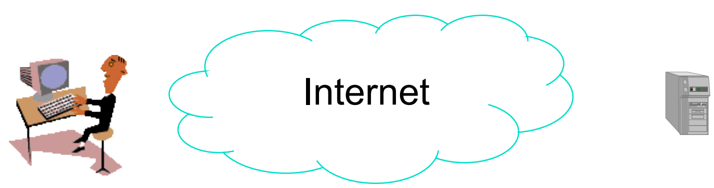
TCP Header contains source & destination port numbers

IP Header contains source and destination IP addresses; transport protocol type

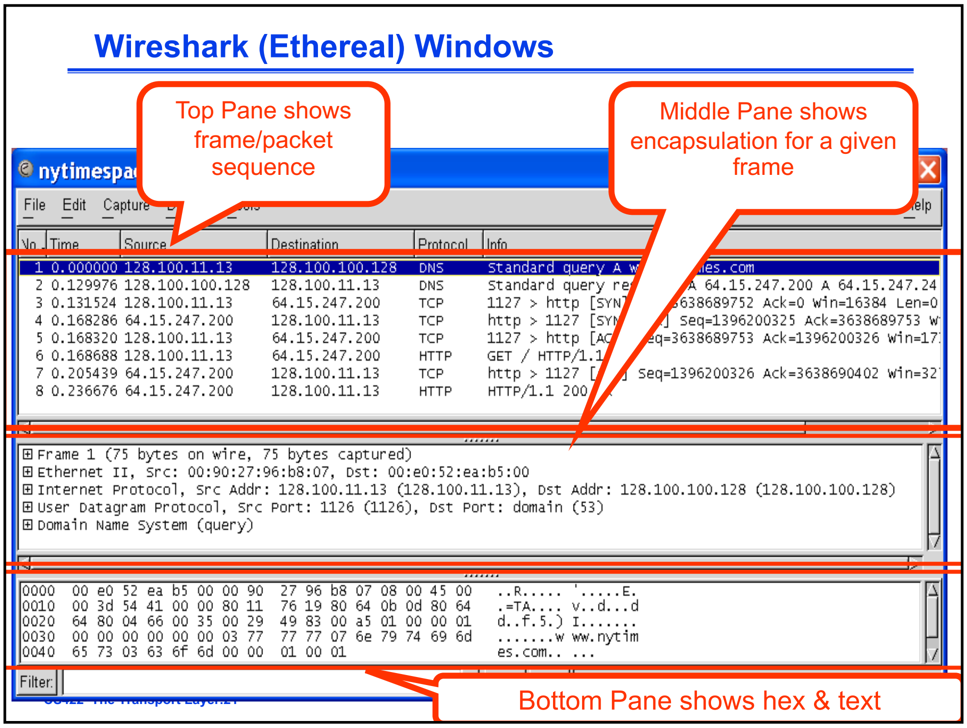
Ethernet Header contains source & destination MAC addresses; network protocol type



Network Analyzer Wireshark (formerly Ethereal)

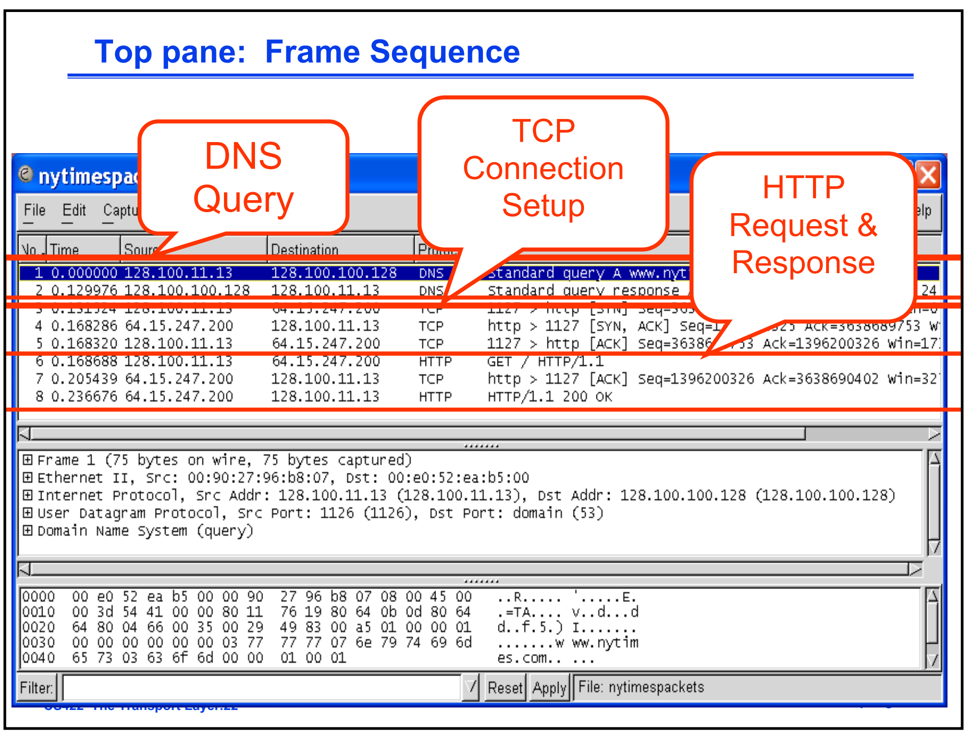


- User clicks on http://www.nytimes.com/
- *Ethereal* network analyzer captures all frames observed by its Ethernet NIC
- Sequence of frames and contents of frame can be examined in detail down to individual bytes



Wireshark (Ethereal) Windows
Top Pane shows frame/packet sequence
Middle Pane shows encapsulation for a given frame
Bottom Pane shows hex & text
nytimespa
File Edit Capture
No. Time Source Destination Protocol Info
1 0.000000 128.100.11.13 128.100.100.128 DNS Standard query A w
2 0.129976 128.100.100.128 128.100.11.13 DNS Standard query re
3 0.131524 128.100.11.13 64.15.247.200 TCP 1127 > http [SYN]
4 0.168286 64.15.247.200 128.100.11.13 TCP http > 1127 [SYN
5 0.168320 128.100.11.13 64.15.247.200 TCP 1127 > http [AC
6 0.168688 128.100.11.13 64.15.247.200 HTTP GET / HTTP/1.1
7 0.205439 64.15.247.200 128.100.11.13 TCP http > 1127 [
8 0.236676 64.15.247.200 128.100.11.13 HTTP HTTP/1.1 200
Frame 1 (75 bytes on wire, 75 bytes captured)
Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 128.100.100.128 (128.100.100.128)
User Datagram Protocol, Src Port: 1126 (1126), Dst Port: domain (53)
Domain Name System (query)
0000 00 e0 52 ea b5 00 00 90 27 96 b8 07 08 00 45 00 ..R..... '.....E.
0010 00 3d 54 41 00 00 80 11 76 19 80 64 0b 0d 80 64 .=TA.... v..d...d
0020 64 80 04 66 00 35 00 29 49 83 00 a5 01 00 00 01 d..f.5.) I.......
0030 00 00 00 00 00 00 03 77 77 77 07 6e 79 74 69 6d ....w ww.nytim
0040 65 73 03 63 6f 6d 00 00 01 00 01 es.com.. ...
Filter:

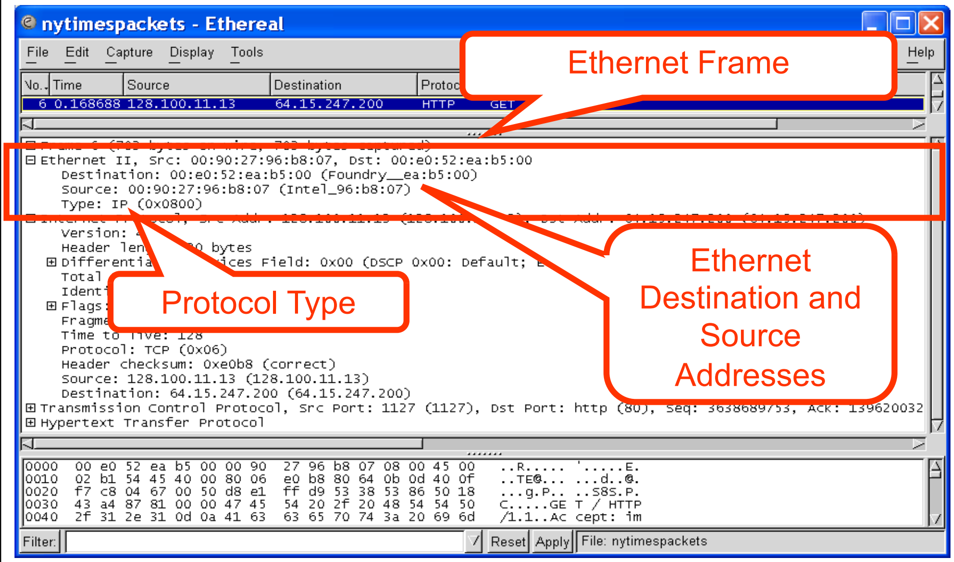
21



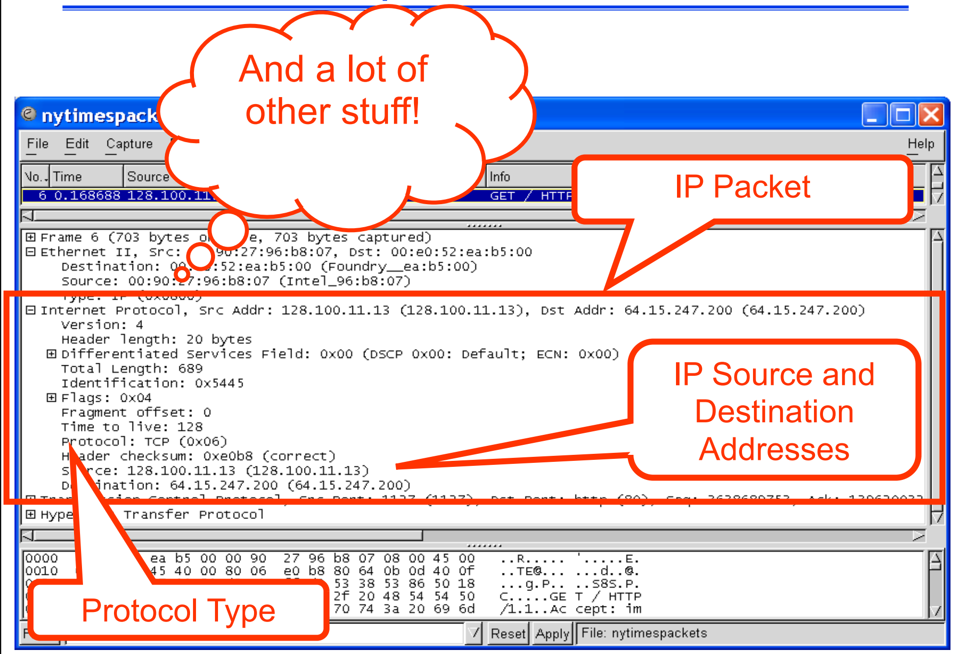
Top pane: Frame Sequence
DNS Query
TCP Connection Setup
HTTP Request & Response
nytimespa
File Edit Captu
No. Time Sourc Destination
1 0.000000 128.100.11.13 128.100.100.128 DNS Standard query A www.nyt
2 0.129976 128.100.100.128 128.100.11.13 DNS Standard query response
4 0.168286 64.15.247.200 128.100.11.13 TCP http > 1127 [SYN, ACK] Seq=1
5 0.168320 128.100.11.13 64.15.247.200 TCP 1127 > http [ACK] Seq=3638
6 0.168688 128.100.11.13 64.15.247.200 HTTP GET / HTTP/1.1
7 0.205439 64.15.247.200 128.100.11.13 TCP http > 1127 [ACK] Seq=1396200326 Ack=3638690402 Win=32
8 0.236676 64.15.247.200 128.100.11.13 HTTP HTTP/1.1 200 OK
Frame 1 (75 bytes on wire, 75 bytes captured)
Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 128.100.100.128 (128.100.100.128)
User Datagram Protocol, Src Port: 1126 (1126), Dst Port: domain (53)
Domain Name System (query)
0000 00 e0 52 ea b5 00 00 90 27 96 b8 07 08 00 45 00 ..R..... '.....E.
0010 00 3d 54 41 00 00 80 11 76 19 80 64 0b 0d 80 64 .=TA.... v..d...d
0020 64 80 04 66 00 35 00 29 49 83 00 a5 01 00 00 01 d..f.5.) I.......
0030 00 00 00 00 00 00 03 77 77 77 07 6e 79 74 69 6d ....w ww.nytim
0040 65 73 03 63 6f 6d 00 00 01 00 01 es.com.. ...
Filter: Reset Apply File: nytimespackets

22

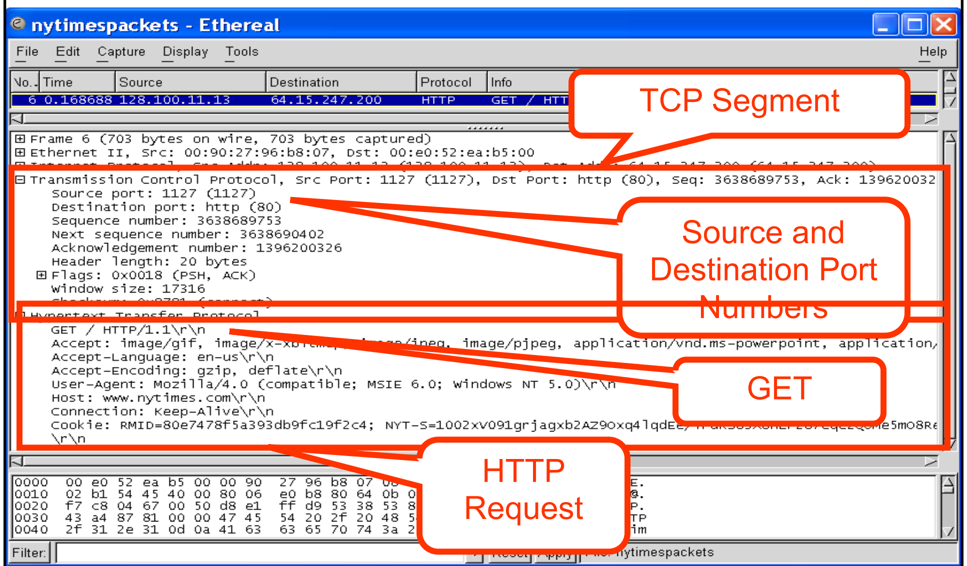
Middle Pane: Encapsulation



Middle Pane: Encapsulation



Middle Pane: Encapsulation



Summary

- Encapsulation is key to layering
- IP provides for transfer of packets across diverse networks
- TCP and UDP provide universal communications services across the Internet
- Distributed applications that use TCP and UDP can operate over the entire Internet
- Internet names, IP addresses, port numbers, sockets, connections, physical addresses

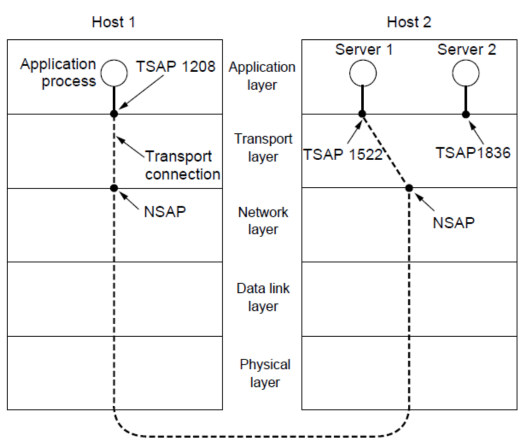
Elements of Transport Protocols

- Addressing »
- Connection establishment »
- Connection release »
- Error control and flow control »
- Multiplexing »
- Crash recovery »

Addressing

° TSAP (transport service access point): transport addresses of end points (Internet: ports, ATM: AAL-SAPs).

- Transport layer adds TSAPs
- Multiple clients and servers can run on a host with a single network (IP) address
- TSAPs are ports for TCP/UDP



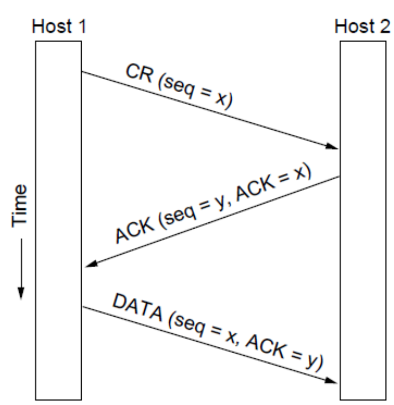
Connection Establishment (1)

- Key problem is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated
  - Don’t treat an old or duplicate packet as new
  - (Use ARQ and checksums for loss/corruption)
- Approach:
  - Don’t reuse sequence numbers within twice the MSL (Maximum Segment Lifetime)
  - Three-way handshake for establishing connection

Protocols must be designed to be correct in all cases!

Connection Establishment (2)

- Three-way handshake used for initial packet
  - Since no state from previous connection
  - Both hosts contribute fresh seq. numbers
  - CR = Connect Request

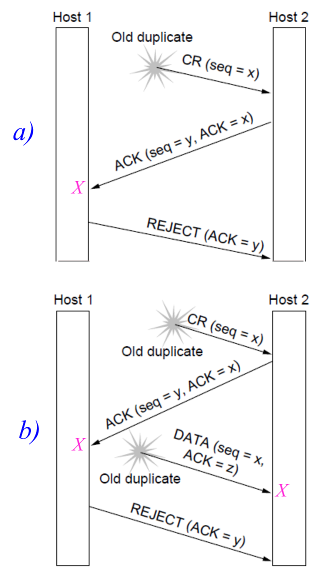


Connection Establishment (3)

Three-way handshake protects against odd cases:

a) Duplicate CR. Spurious ACK does not connect

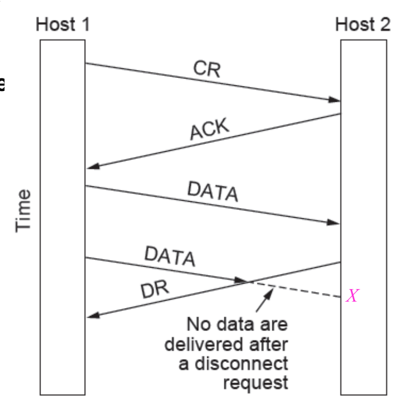
b) Duplicate CR and DATA. Same plus DATA will be rejected (wrong ACK).



Connection Release (1)

Key problem is to ensure reliability while releasing

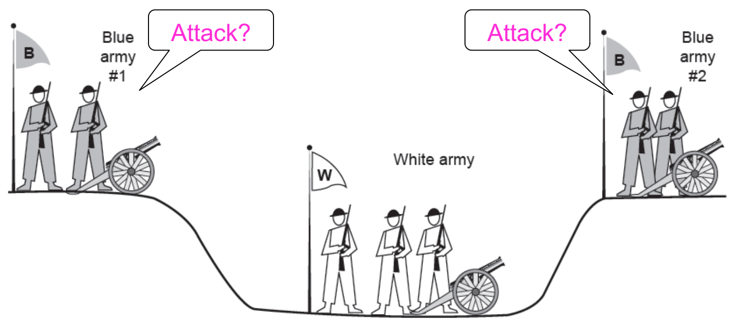
Asymmetric release (when one side breaks connection) is abrupt and may lose data



Connection Release (2)

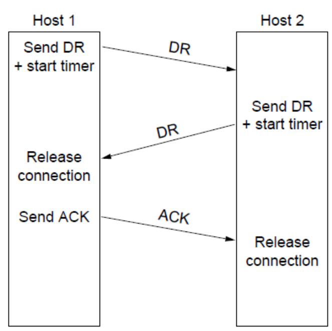
Symmetric release (both sides agree to release) can't be handled solely by the transport layer

- **Two-army problem shows pitfall of agreement**



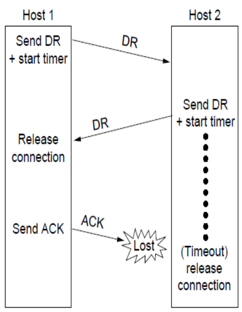
Connection Release (3)

- **Normal release sequence, initiated by transport user on Host 1**
  - **DR=Disconnect Request**
  - **Both DRs are ACKed by the other side**

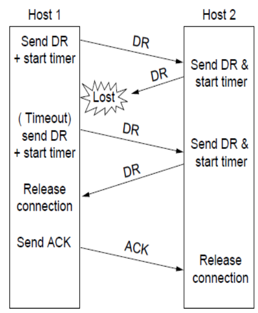


Connection Release (4)

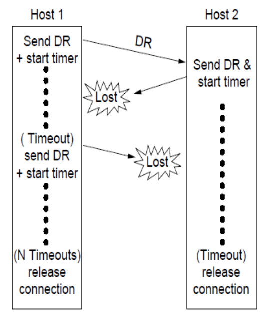
Error cases are handled with timer and retransmission



Final ACK lost, Host 2 times out



Lost DR causes retransmissions



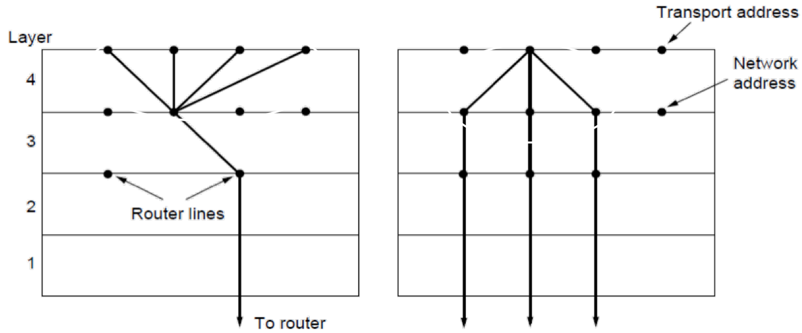
Extreme: Many lost DRs cause both hosts to timeout

Error Control and Flow Control

- Foundation for <u>error control</u> is a sliding window (from Link layer) with checksums and retransmissions
- <u>Flow control</u> manages buffering at sender/receiver
  - Issue is that data goes to/from the network and applications at different times
  - Window tells sender available buffering at receiver
  - Makes a variable-size sliding window

Multiplexing

° **Kinds of transport / network sharing that can occur:**
- **Multiplexing: connections share a network address**
- **Inverse multiplexing: addresses share a connection**



Multiplexing *Inverse Multiplexing*

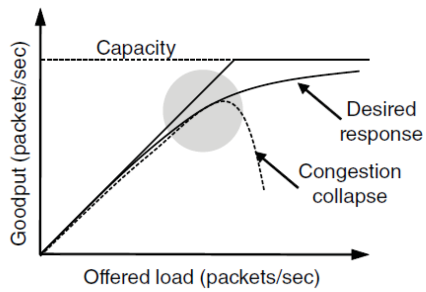
Congestion Control

○ **Two layers are responsible for congestion control:**
- ○ **Transport layer, controls the offered load [here]**
- ○ **Network layer, experiences congestion [previous]**

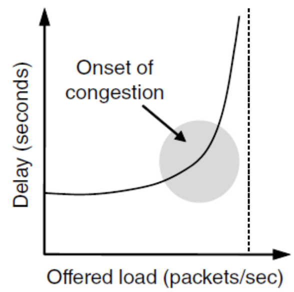
- ○ **Desirable bandwidth allocation »**
- ○ **Regulating the sending rate »**
- ○ **Wireless issues »**

Desirable Bandwidth Allocation (1)

Efficient use of bandwidth gives high goodput, low delay



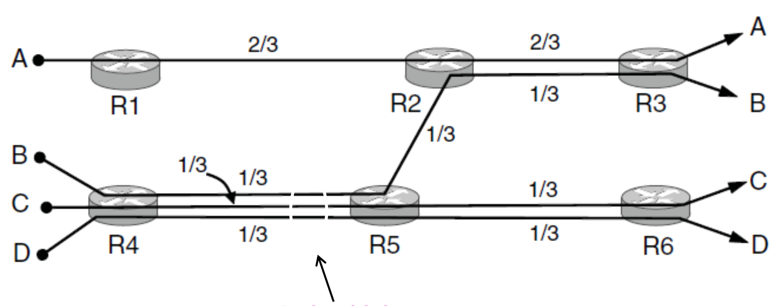
Goodput rises more slowly than load when congestion sets in



Delay begins to rise sharply when congestion sets in

Desirable Bandwidth Allocation (2)

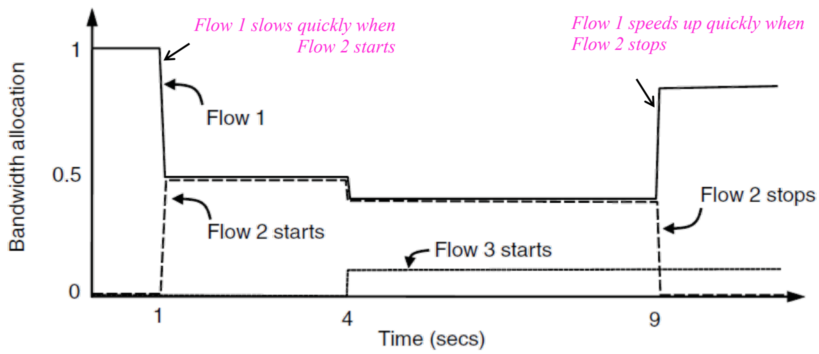
- **Fair use gives bandwidth to all flows (no starvation)**
  - **<u>Max-min fairness</u> gives equal shares of bottleneck**



Bottleneck link

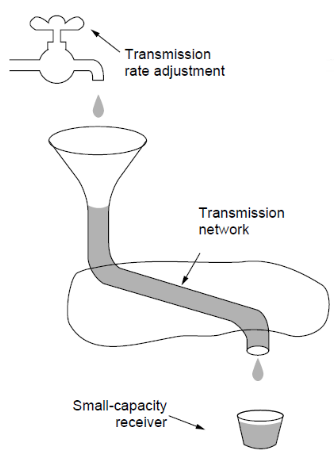
Desirable Bandwidth Allocation (3)

We want bandwidth levels to converge quickly when traffic patterns change



Regulating the Sending Rate (1)

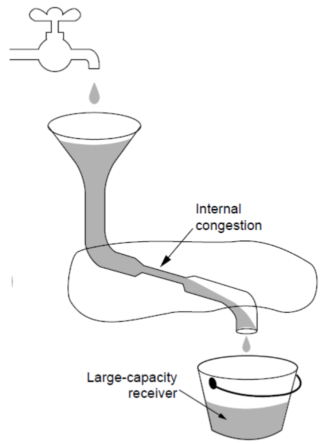
- **Sender may need to slow down for different reasons:**
  - **Flow control, when the receiver is not fast enough [right]**
  - **Congestion, when the network is not fast enough [over]**



A fast network feeding a low-capacity receiver → flow control is needed

Regulating the Sending Rate (2)

Our focus is dealing with this problem – congestion



A slow network feeding a high-capacity receiver → congestion control is needed

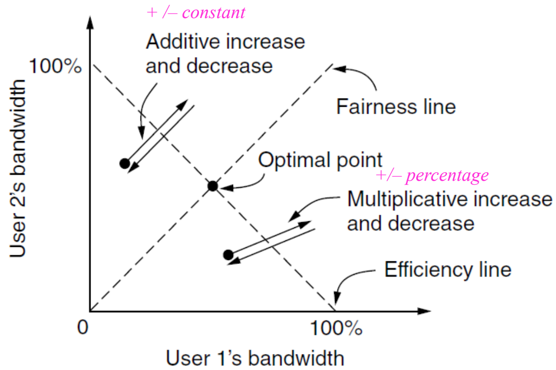
Regulating the Sending Rate (3)

Different congestion signals the network may use to tell the transport endpoint to slow down (or speed up)

Protocol	Signal	Explicit?	Precise?
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

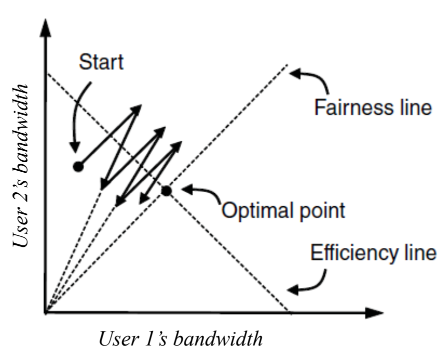
Regulating the Sending Rate (3)

If two flows increase/decrease their bandwidth in the same way when the network signals free/busy they will not converge to a fair allocation



Regulating the Sending Rate - AIMD

- The AIMD (Additive Increase Multiplicative Decrease) control law does converge to a fair and efficient point!
  - TCP uses AIMD for this reason



The Internet Transport Protocols: UDP

- Introduction to UDP
- Remote Procedure Call
- The Real-Time Transport Protocol

UDP

- Best effort datagram service
- Multiplexing enables sharing of IP datagram service
- Simple transmitter & receiver
  - Connectionless: no handshaking & no connection state
  - Low header overhead
  - No flow control, no error control, no congestion control
  - UDP datagrams can be lost or out-of-order
- Applications
  - multimedia (e.g. RTP)
  - network services (e.g. DNS, RIP, SNMP)

UDP Datagram

0 … 16	16 … 31
Source Port	Destination Port
UDP Length	UDP Checksum
Data	

0-255
- Well-known ports

256-1023
- Less well-known ports

1024-65536
- Ephemeral client ports

° Source and destination port numbers
  - Client ports are ephemeral
  - Server ports are well-known
  - Max number is 65,535

° UDP length
  - Total number of bytes in datagram (including header)
  - 8 bytes ≤ length ≤ 65,535

° UDP Checksum
  - Optionally detects errors in UDP datagram (and a pseudo-header with IP addresses)

UDP Checksum Calculation

0 … 8	*8 … 16*	*16 … 31*
Source IP Address		
Destination IP Address		
0 0 0 0 0 0 0 0	*Protocol = 17*	*UDP Length*

UDP pseudo-header

° UDP checksum detects for end-to-end errors

° Covers pseudo-header followed by UDP datagram

° IP addresses included to detect against mis-delivery

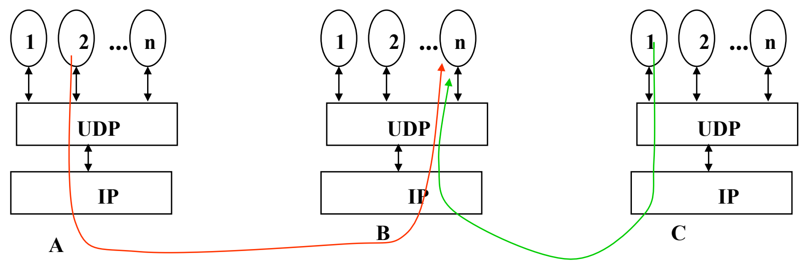
° IP & UDP checksums set to zero during calculation

° Pad with 1 byte of zeros if UDP length is odd

° Optional but hosts are required to enable it

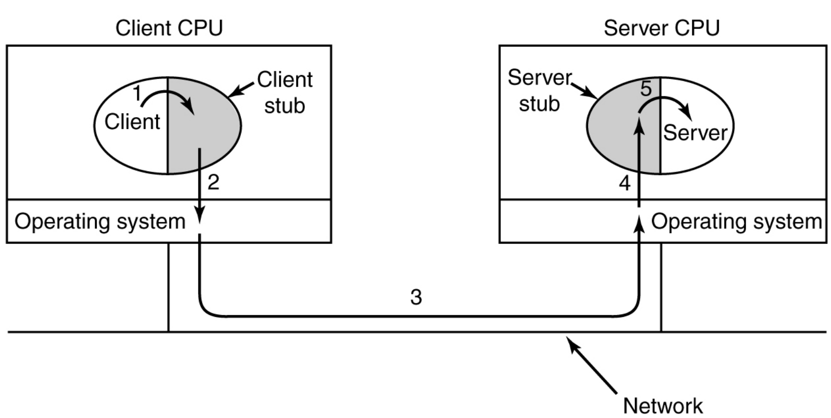
UDP Multiplexing

- All UDP datagrams arriving to IP address B and destination port number *n* are delivered to the same process
- What UDP does not do?
  - Flow control, error control & retransmission, congestion control



Remote Procedure Call (RPC)

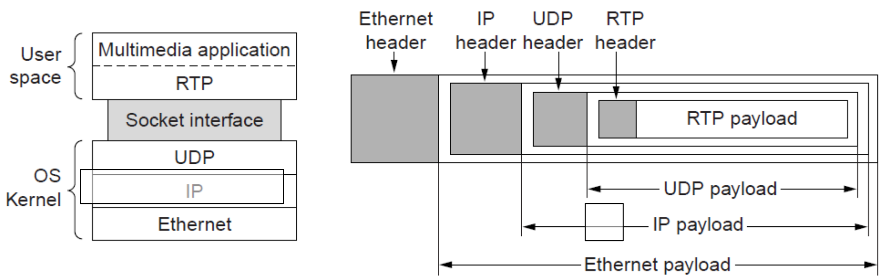
- RPC idea: make a remote procedure call look as much as possible like a local one
  - Another area of using UDP is RTP (real-time transport protocol)



Steps in making a remote procedure call. The stubs are shaded.

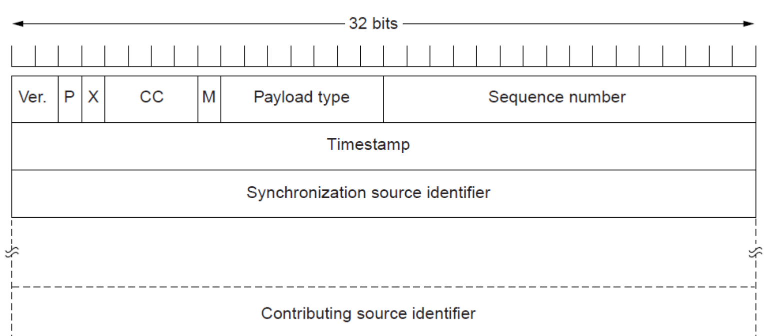
Real-Time Transport (1)

- RTP (Real-time Transport Protocol) provides support for sending real-time media over UDP
  - Often implemented as part of the application



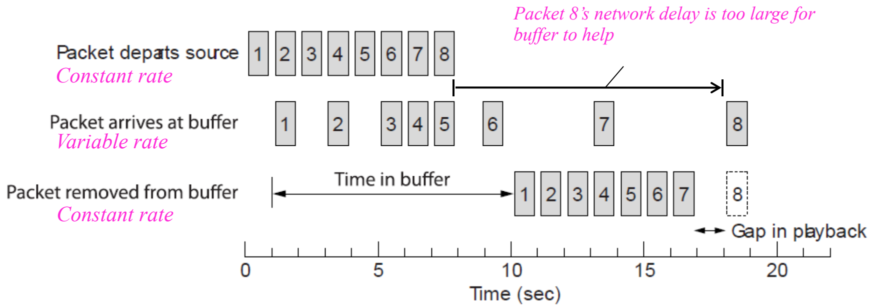
Real-Time Transport (2)

- RTP header contains fields to describe the type of media and synchronize it across multiple streams
  - RTCP sister protocol helps with management tasks



Real-Time Transport (3)

- Buffer at receiver is used to delay packets and absorb jitter so that streaming media is played out smoothly



Internet Protocols – TCP

- The TCP service model »
- The TCP segment header »
- TCP connection establishment »
- TCP connection state modeling »
- TCP sliding window »
- TCP timer management »
- TCP congestion control »

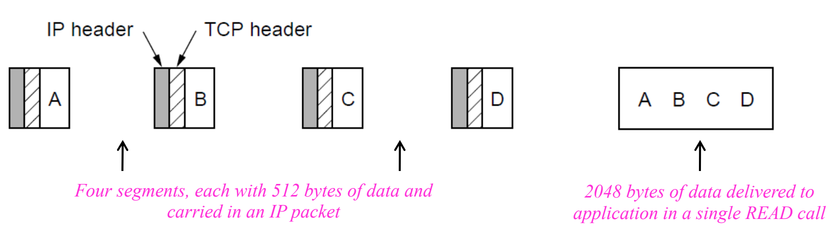
The TCP Service Model (1)

- TCP provides applications with a reliable byte stream between processes; it is the workhorse of the Internet
  - Popular servers run on well-known ports

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

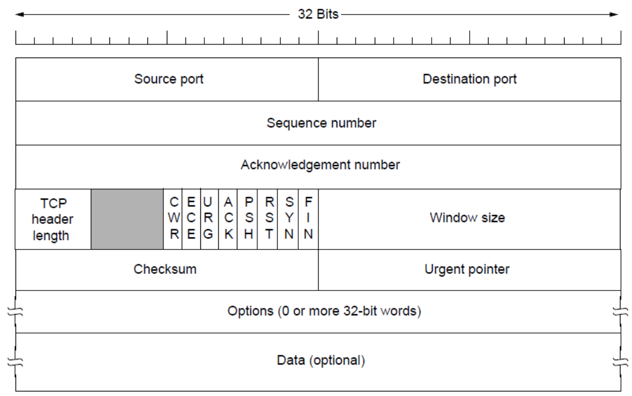
The TCP Service Model (2)

Applications using TCP see only the byte stream [right] and not the segments [left] sent as separate IP packets



The TCP Segment Header

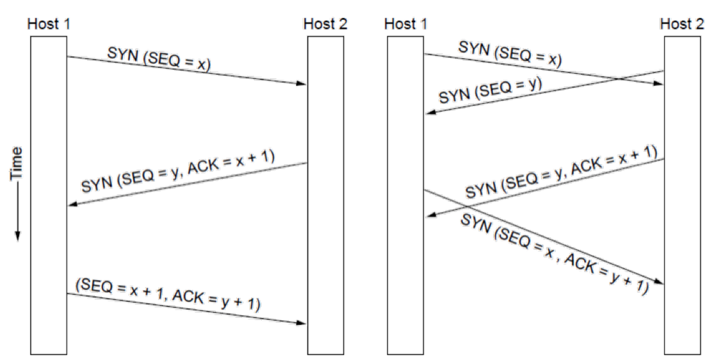
TCP header includes addressing (ports), sliding window (seq. / ack. number), flow control (window), error control (checksum) and more.



TCP Connection Establishment

TCP sets up connections with the three-way handshake

- **Release is symmetric, also as described before**



Normal case *Simultaneous connect*

TCP Connection State Modeling (1)

The TCP connection finite state machine has more states than our simple example from earlier.

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

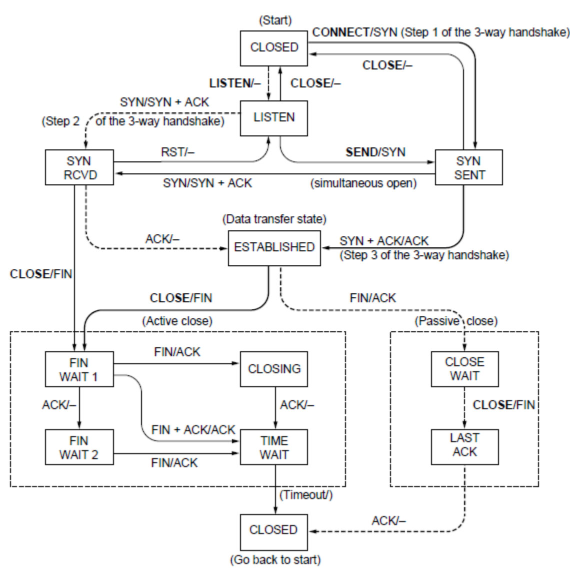
TCP Connection State Modeling (2)

Solid line is the normal path for a client.

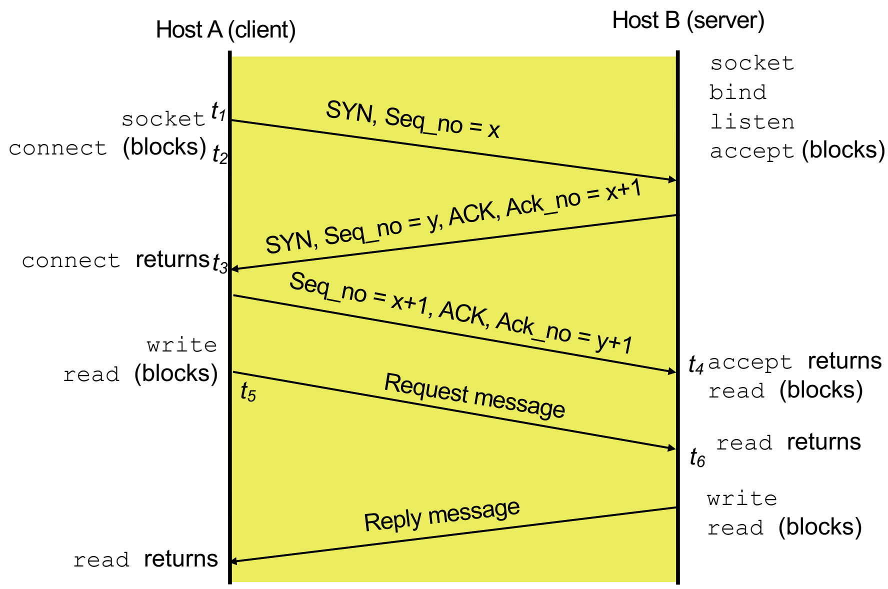
Dashed line is the normal path for a server.

Light lines are unusual events.

Transitions are labeled by the cause and action, separated by a slash.

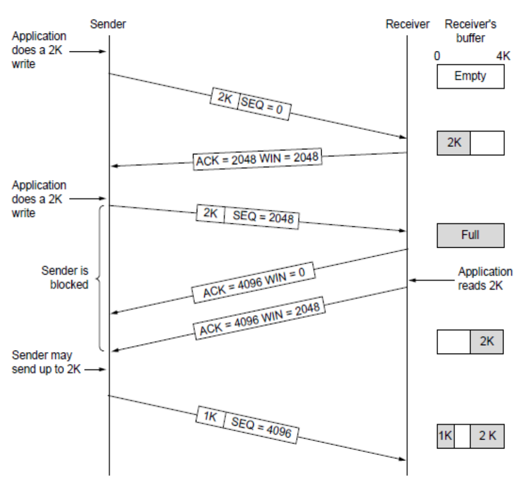


TCP Client-Server Application



TCP Sliding Window (1)

- TCP adds flow control to the sliding window as before
  - ACK + WIN is the sender's limit



Nagle Algorithm

° Situation: user types 1 character at a time
  - Transmitter sends TCP segment per character (41B)
  - Receiver sends ACK (40B)
  - Receiver echoes received character (41B)
  - Transmitter ACKs echo (40 B)
  - 162 bytes transmitted to transfer 1 character!

° Solution:
  - TCP sends data & waits for ACK
  - New characters buffered instead
  - Send new characters when ACK arrives
  - Algorithm adjusts to RTT
    - Short RTT send frequently at low efficiency
    - Long RTT send less frequently at greater efficiency

Silly Window Syndrome

° Situation:
  - Transmitter sends large amount of data
  - Receiver buffer depleted slowly, so buffer fills
  - Every time a few bytes read from buffer, a new advertisement to transmitter is generated
  - Sender immediately sends data & fills buffer
  - Many small, inefficient segments are transmitted

° Solution:
  - Receiver does not advertize window until window is at least ½ of receiver buffer or maximum segment size
  - Transmitter refrains from sending small segments

Sequence Number Wraparound

° $2^{32} = 4.29 \times 10^9$ bytes = 34.3×10^9 bits
  - At 1 Gbps, sequence number wraparound in 34.3 seconds (Max. Segment Lifetime = 120 seconds).

° Timestamp option: Insert 32 bit timestamp in header of each segment
  - Timestamp + sequence no → 64-bit seq. no
  - Timestamp clock must:
    - tick forward at least once every 2^{31} bits
    - Not complete cycle in less than one MSL
    - Example: clock tick every 1 ms @ 8 Tbps wraps around in 25 days

Where this timestamp can be filled in?

Delay-BW Product & Advertised Window Size

° Suppose RTT=100 ms, R=2.4 Gbps
  - # bits in pipe = 3 Mbytes

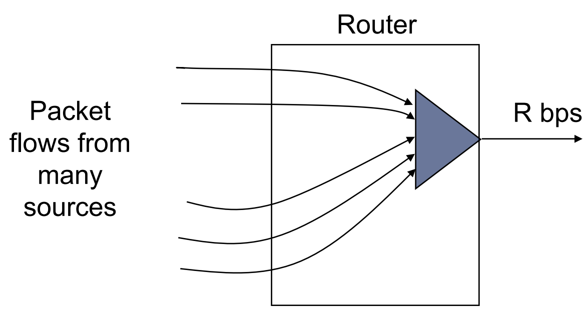
° If single TCP process occupies pipe, then required advertised window size is
  - RTT x Bit rate = 3 Mbytes
  - Normal maximum window size is 65535 bytes

° Solution: Window Scale Option
  - Window size up to 65535 x 2^{14} = 1 Gbyte allowed
  - Requested in SYN segment

Where the information can be filled in?

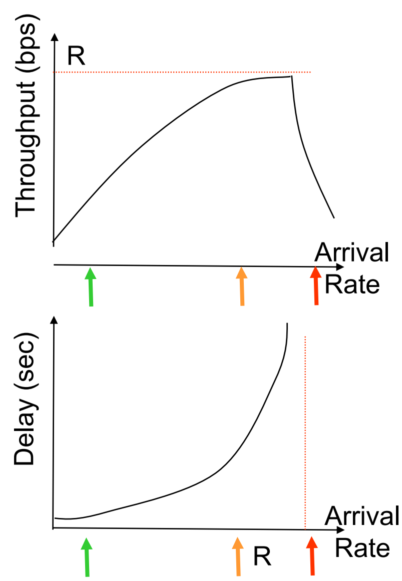
TCP Congestion Control

- *Advertised window* size is used to ensure that receiver's buffer will not overflow
- However, buffers at intermediate routers between source and destination may overflow



- Congestion occurs when total arrival rate from all packet flows exceeds R over a sustained period of time
- Buffers at multiplexer will fill and packets will be lost

Phases of Congestion Behavior



1. Light traffic
   - Arrival Rate << R
   - Low delay
   - Can accommodate more
2. Knee (congestion onset)
   - Arrival rate approaches R
   - Delay increases rapidly
   - Throughput begins to saturate
3. Congestion collapse
   - Arrival rate > R
   - Large delays, packet loss
   - Useful application throughput drops significantly

Congestion Window

- Desired operating point: just before knee
  - Sources must control their sending rates so that aggregate arrival rate is just before knee
- TCP sender maintains a *congestion window* (cwnd) to control congestion at intermediate routers
- Effective window is the minimum of congestion window and advertised window (for flow control)
- Problem: source does not know what its “fair” share of available bandwidth should be
- Solution: adapt dynamically to available BW
  - Sources probe the network by increasing cwnd
  - When congestion detected, sources reduce rate
  - Ideally, sources sending rate stabilizes near ideal point

Congestion Window (Cont.)

- How does the TCP congestion algorithm change congestion window dynamically according to the most up-to-date state of the network?
- At light traffic: each segment is ACKed quickly
  - Increase cwnd aggressively
- At knee: segment ACKs arrive, but more slowly
  - Slow down increase in cwnd
- At congestion: segments encounter large delays (so retransmission timeouts occur); segments are dropped in router buffers (resulting in duplicate ACKs)
  - Reduce transmission rate, then probe again

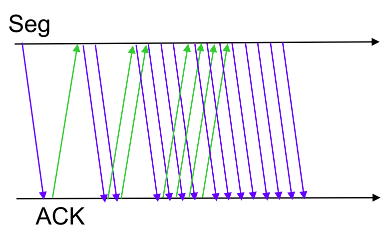
TCP Congestion Control - AIMD

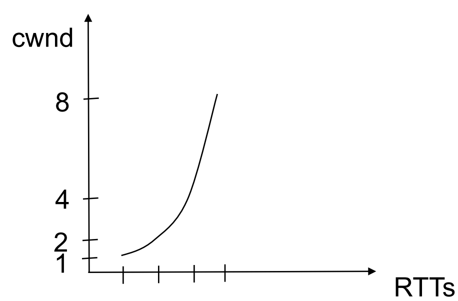
- TCP uses AIMD with loss signal to control congestion
  - AIMD: Additive Increase Multiplicative Decrease
  - Implemented as a congestion window (cwnd) for the number of segments that may be in the network
  - Uses several mechanisms that work together

Name	Mechanism	Purpose
ACK clock	Congestion window (cwnd)	Smooth out packet bursts
Slow-start	Double cwnd each RTT	Rapidly increase send rate to reach roughly the right level
Additive Increase	Increase cwnd by 1 packet each RTT	Slowly increase send rate to probe at about the right level
Fast retransmit / recovery	Resend lost packet after 3 duplicate ACKs; send new packet for each new ACK	Recover from a lost packet without stopping ACK clock

TCP Congestion Control: Slow Start

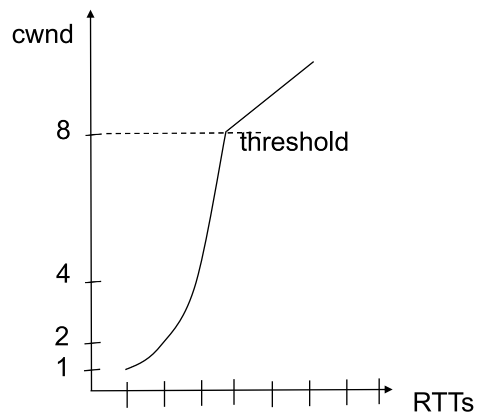
- Slow start: increase congestion window size by one segment upon receiving an ACK from receiver
  - initialized at ≤ 2 segments
  - used at (re)start of data transfer
  - congestion window increases exponentially



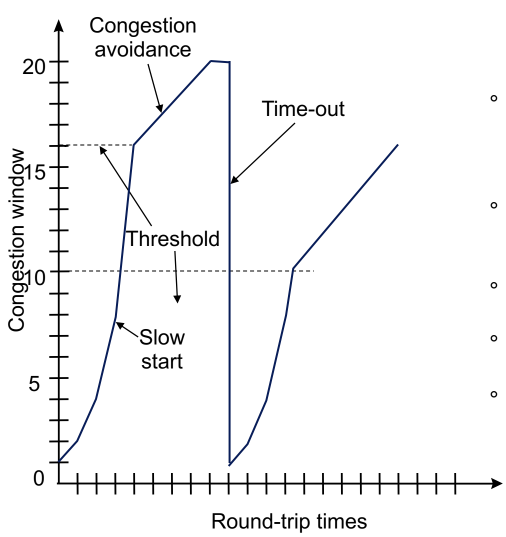


TCP Congestion Control: Congestion Avoidance

- Algorithm progressively sets a *congestion threshold*
  - When cwnd > threshold, slow down rate at which cwnd is increased
- Increase congestion window size by one segment per round-trip-time (RTT)
  - Each time an ACK arrives, cwnd is increased by 1/cwnd segment
  - In one RTT, cwnd segments are sent, so total increase in cwnd is cwnd x 1/cwnd = 1
  - cwnd grows linearly with time



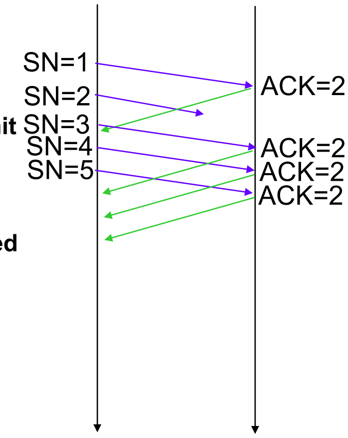
TCP Congestion Control: Congestion



- Congestion is detected upon timeout or receipt of duplicate ACKs
- Assume current cwnd corresponds to available bandwidth
- Adjust congestion threshold = ½ x current cwnd
- Reset cwnd to 1
- Go back to slow-start
- Over several cycles expect to converge to congestion threshold equal to about ½ the available bandwidth

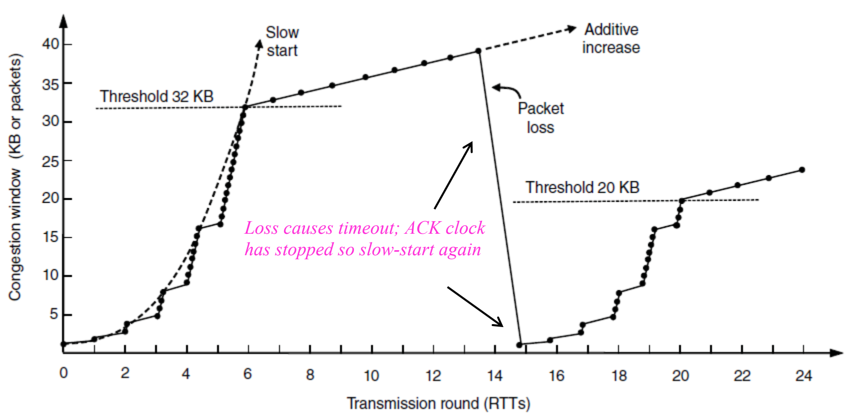
Fast Retransmit & Fast Recovery

- Congestion causes many segments to be dropped
- If only a single segment is dropped, then subsequent segments trigger duplicate ACKs before timeout
- Can avoid large decrease in cwnd as follows:
  - When three duplicate ACKs arrive, retransmit lost segment immediately
  - Reset congestion threshold to ½ cwnd
  - Reset cwnd to congestion threshold + 3 to account for the three segments that triggered duplicate ACKs
  - Remain in congestion avoidance phase
  - However if timeout expires, reset cwnd to 1
  - In absence of timeouts, cwnd will oscillate around optimal value



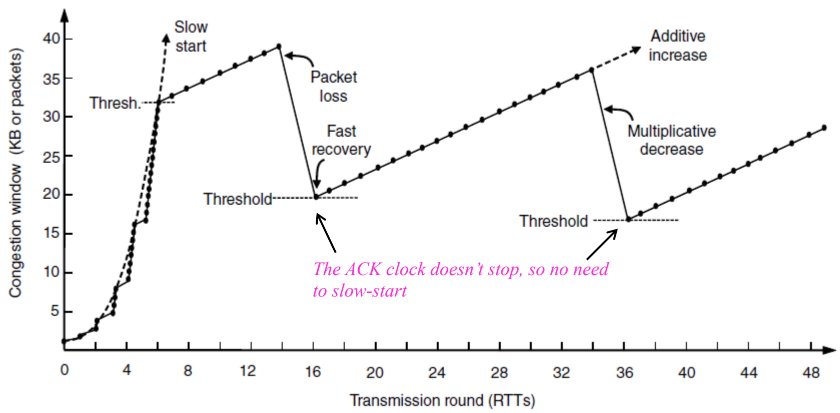
TCP Congestion Control – TCP Tahoe

- Slow start followed by additive increase (TCP Tahoe)
  - Threshold is half of previous loss cwnd



TCP Congestion Control – TCP Reno

° With fast recovery, we get the classic sawtooth (TCP Reno)
  - Retransmit lost packet after 3 duplicate ACKs
  - New packet for each dup. ACK until loss is repaired



Performance Issues

○ Many strategies for getting good performance have been learned over time
  - Performance problems »
  - Measuring network performance »
  - Host design for fast networks »
  - Fast segment processing »
  - Header compression »
  - Protocols for “long fat” networks »

Performance Problems

- Unexpected loads often interact with protocols to cause performance problems
  - Need to find the situations and improve the protocols
- Examples:
  - Broadcast storm: one broadcast triggers another
  - Synchronization: a building of computers all contact the DHCP server together after a power failure
  - Tiny packets: some situations can cause TCP to send many small packets instead of few large ones

Measuring Network Performance

- Measurement is the key to understanding performance – but has its own pitfalls.
- Example pitfalls:
  - Caching: fetching Web pages will give surprisingly fast results if they are unexpectedly cached
  - Timing: clocks may over/underestimate fast events
  - Interference: there may be competing workloads

Host Design for Fast Networks

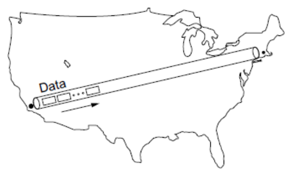
- Poor host software can greatly slow down networks.
- Rules of thumb for fast host software:
  - Host speed more important than network speed
  - Reduce packet count to reduce overhead
  - Minimize data touching
  - Minimize context switches
  - Avoiding congestion is better than recovering from it
  - Avoid timeouts

Header Compression

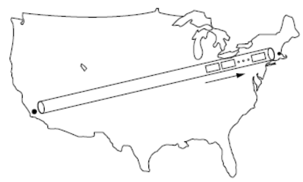
- Overhead can be very large for small packets
  - 40 bytes of header for RTP/UDP/IP VoIP packet
  - Problematic for slow links, especially wireless
- Header compression mitigates this problem
  - Runs between Link and Network layer
  - Omits fields that don't change or change predictably
    - 40 byte TCP/IP header → 3 bytes of information
  - Gives simple high-layer headers and efficient links

Protocols for “Long Fat” Networks (1)

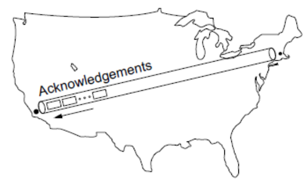
- Networks with high bandwidth (“Fat”) and high delay (“Long”) can store much information inside the network
  - Requires protocols with ample buffering and few RTTs, rather than reducing the bits on the wire



Starting to send 1 Mbit San Diego → Boston



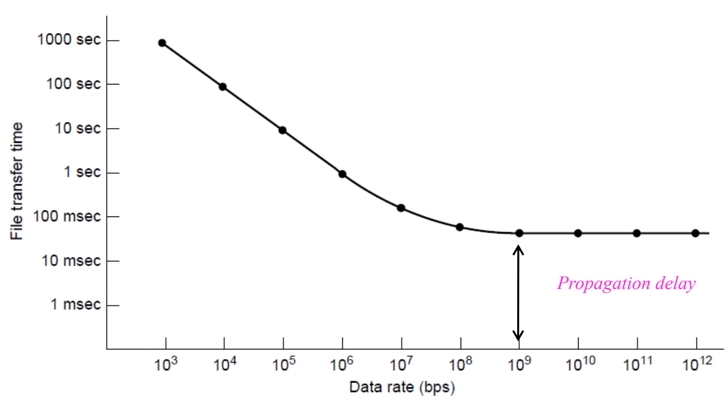
20ms after start



40ms after start

Protocols for “Long Fat” Networks (2)

- You can buy more bandwidth but not lower delay
  - Need to shift ends (e.g., into cloud) to lower further



Minimum time to send and ACK a 1-Mbit file over a 4000-km line

Homework 5

- Reading
- Project 2
- Final exam 12:40PM – 2:40PM, Wednesday, May 13, ENG 107

  NO MAKE-UP!