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**

HTTP client

HTTP server

Port 1127

Port 80

GET

Response

1127

80

GET

Response

HTTP server

HTTP client

TCP/IP Protocol Suite

HTTP

SMTP

DNS

RTP

TCP

UDP

IP

(IP: ICMP, ARP)

Network interface 1

Network interface 2

Network interface 3

Diverse network technologies

TCP

UDP

IP

Best-effort connectionless packet transfer

Distributed applications

Reliable stream service

User datagram service
**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

\[(send \text{ port } \#, \ send \text{ IP address}, \ receive \text{ port } \#, \ receive \text{ IP address})\]
### Internet Names & Addresses

#### Internet Names
- Each host 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
  - 00:90:27:96:68:07 12 hex numbers
    - Intel
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

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

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 TPUDUs, packets, and frames.

Example Internet

PPP does not use addresses

<table>
<thead>
<tr>
<th></th>
<th>netid</th>
<th>hostid</th>
<th>Physical address</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>1</td>
<td>1</td>
<td>s</td>
</tr>
<tr>
<td>workstation</td>
<td>1</td>
<td>2</td>
<td>w</td>
</tr>
<tr>
<td>router</td>
<td>1</td>
<td>3</td>
<td>r</td>
</tr>
<tr>
<td>router</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>PC</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>
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
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

HTTP uses process-to-process reliable byte stream transfer of TCP connection:
server socket: (IP Address, 80)
PC socket (IP Address, Eph. #)

TCP uses node-to-node unreliable packet transfer of IP server IP address & PC IP address
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

HTTP Request

° 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

Network Analyzer Wireshark (formerly Ethereal)
Wireshark (Ethereal) Windows

Top Pane shows frame/packet sequence
Middle Pane shows encapsulation for a given frame
Bottom Pane shows hex & text
Middle Pane: Encapsulation

And a lot of other stuff!
**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
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 error control is a sliding window (from Link layer) with checksums and retransmissions

- Flow control 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

**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)
  - Max-min fairness gives equal shares of bottleneck
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
  - Congestion, when the network is not fast enough

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

![Diagram showing a slow network feeding a high-capacity receiver with congestion control needed.]

Regulating the Sending Rate (3)

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

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Signal</th>
<th>Explicit?</th>
<th>Precise?</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCP</td>
<td>Rate to use</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TCP with ECN</td>
<td>Congestion warning</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>FAST TCP</td>
<td>End-to-end delay</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CUBIC TCP</td>
<td>Packet loss</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TCP</td>
<td>Packet loss</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>0</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td>UDP Length</td>
</tr>
</tbody>
</table>

- **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)

0-255
- Well-known ports

256-1023
- Less well-known ports

1024-65536
- Ephemeral client ports

---

**UDP Checksum Calculation**

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source IP Address</td>
<td>Destination IP Address</td>
<td>0 0 0 0 0 0 0 0</td>
<td>Protocol = 17</td>
</tr>
</tbody>
</table>

- **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)
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.

![Diagram]

Packet 8's network delay is too large for buffer to help.

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

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

The TCP Service Model (2)

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

Four segments, each with 512 bytes of data and carried in an IP packet

2048 bytes of data delivered to application in a single READ call
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
The TCP connection finite state machine has more states than our simple example from earlier.

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

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

Host A (client)  
socket (blocks) $t_1$
connect (blocks) $t_2$
write (blocks) $t_3$
read (blocks) $t_4$
read returns $t_5$

SYN, Seq_no = x
SYN, Seq_no = y, ACK, Ack_no = x+1
Seq_no = x+1, ACK, Ack_no = y+1
Request message
Reply message

Host B (server)  
socket bind listen (blocks)
accept (blocks) $t_6$
accept returns read (blocks)
read returns write read (blocks)

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 (40B)
- 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 advertise 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 = $34.3 \times 10^9$ 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 $\rightarrow$ 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 $65535 \times 2^{14} = 1$ 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

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

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

![Graph showing cwnd increase over RTTs]
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 ⅓ 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 $\frac{1}{2}$ 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

The ACK clock doesn’t stop, so no need to slow-start

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

![Diagram of data transmission from San Diego to Boston](image)

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

![Graph showing file transfer time vs. data rate](image)

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

Propagation delay
Homework 5

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

NO MAKE-UP!