CSCD 330
Network Programming
Winter 2020

Lecture 11a
Transport Layer

Reading: Chapter 3

Who is this?

Some Material in these slides from J.F Kurose and K.W. Ross
All material copyright 1996-2007
Bill The Cat

**Bloom County** was an American comic strip
By Berkeley Breathed which ran from December 8, 1980, until August 6, 1989
It looked at events in politics and culture through viewpoint of small town in Middle America and where animals can talk

**Bill the Cat** is a large orange tabby cat, a parody of Garfield, says little beyond, "Ack" and "Pbthhh"
His persistent near-catatonic state was result of drug use or brain damage resulting from once being legally dead and then revived after too long of a period

http://en.wikipedia.org/wiki/Bloom_County#
Chapter 3 Sections

• Transport-layer services
• Multiplexing and demultiplexing
• Connectionless transport: UDP
• Principles of reliable data transfer
  • Pipelined vs. Stop and Wait Protocols
• Connection-oriented transport: TCP
  • segment structure
  • reliable data transfer
  • Flow control
  • Connection management

Stop Here
Introduction

• Said that TCP is reliable
  • TCP protocol does many things to manage reliability
    • Acknowledgments,
    • Retransmit lost segments,
    • Timer for retransmit management

• Now, look at other features used by TCP
  • Flow Control
  • Congestion Control
• First, examine Connection Management
  • The States of a TCP connection
Connection Management
TCP Connection Management

• TCP sender, receiver establish ‘connection’ before exchanging data segments

• Initialize TCP variables
  • Sequence numbers
  • Buffers,
  • Flow control info (e.g. RcvWindow)

• Client - Connection initiator
  Socket clientSocket =
  new Socket("hostname",6789);

• Server - Contacted by client
  Socket connectionSocket = welcomeSocket.accept();
TCP Connection Management

Opens Connection

Three way handshake

Step 1  Client host sends TCP SYN segment to server, 
       Sets SYN flag
       • Specifies initial sequence number
       • No data

Step 2  Server host receives SYN, replies with SYN/ACK segment
       • Server allocates buffers
       • Specifies server initial sequence number

Step 3  Client receives SYNACK, replies with ACK segment
       • Which may contain data
       • Allocates buffers
TCP 3-Way Handshake
Packet 1: source: 130.57.20.10  dest.:130.57.20.1  
TCP: ----- TCP header -----  
  TCP: Source port = 1026  
  TCP: Destination port = 524  
  TCP: Initial sequence number = 12952  
  TCP: Next expected Seq number = 12953  
  TCP: Acknowledgment number = 12953  
  TCP: Window = 8192  
  TCP: Checksum = 1303 (correct)  
  TCP: Maximum segment size = 1460 (TCP Option)  

Packet 2: source: 130.57.20.1  dest: 130.57.20.10  
TCP: ----- TCP header -----  
  TCP: Source port = 524  
  TCP: Destination port = 1026  
  TCP: Initial sequence number = 2744080  
  TCP: Next expected Seq number = 2744081  
  TCP: Acknowledgment number = 12953  
  TCP: Window = 32768  
  TCP: Checksum = D3B7 (correct)  
  TCP: Maximum segment size = 1460 (TCP Option)  

Packet 3: source: 130.57.20.10  dest: 130.57.20.1  
TCP: ----- TCP header -----  
  TCP: Source port = 1026  
  TCP: Destination port = 524  
  TCP: Sequence number = 12953  
  TCP: Next expected Seq number = 12953  
  TCP: Acknowledgment number = 2744081  
  TCP: Window = 8760  
  TCP: Checksum = 493D (correct)  
  TCP: No TCP options  

• Only part of the TCP headers are displayed.
TCP Connection Management

• Closing a Connection

  • While it takes three segments to establish a connection ...
    • Takes four to terminate a connection
  • Why do you think it takes four segments?
TCP Connection Management

• Why 4 Segments to Close TCP?
  • TCP connection full-duplex
    • Each direction must be shut down independently!!
    • Rule that either end can send FIN when done sending data
    • When TCP receives FIN, notifies application that other end has terminated that direction of data flow
    • FIN is typically result of application issuing close, `Socket.close();`
TCP Connection Management (cont.)

Closing a connection

Client closes socket:
clientSocket.close();

Step 1: Client end system sends TCP FIN control segment to server.

Step 2: Server receives FIN, replies with ACK
Closes connection, sends FIN
Step 3: Client receives FIN, replies with ACK.

- Enters “timed wait” - will respond with ACK to received FINs

Step 4: Server, receives ACK. Connection closed.

Timed wait is so that make sure ACK truly makes it, or Fin will get resent, connection will not be over!!
TCP Connection Management (cont)

TCP Client lifecycle:
- CLOSED
  - wait 30 seconds
  - receive FIN send ACK
  - receive ACK send nothing

- TIME_WAIT
  - receive ACK send nothing

- FIN_WAIT_2
  - receive FIN send ACK

- FIN_WAIT_1
  - send FIN

TCP Server lifecycle:
- CLOSED
  - server application creates a listen socket
  - receive SYN send SYN & ACK
  - receive ACK send nothing

- LISTEN
  - receive SYN & ACK

- SYN_RCVD
  - receive FIN send ACK

- ESTABLISHED
  - client application initiates a TCP connection
  - send SYN
  - receive SYN & ACK send ACK

- ESTABLISHED
  - client application initiate
  - send FIN

- LAST_ACK
  - receive ACK send nothing

- CLOSE_WAIT
  - receive FIN send ACK
Wireshark Tutorial on TCP Sequence Numbers and ACK's

• Nice short tutorial plus example file on how TCP manages sequence numbers and acks

http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/
Two Types of TCP Performance

• Two issues to consider
  • Flow control
    • Coordination between sender/receiver

• Congestion control
  • Behavior altered based on network conditions

• First, look at Flow Control
Buffers at Hosts

• Sending Buffer
  • Maintains data sent but not ACKed
  • Data written by application but not sent

• Receive Buffer
  • Data that arrives out of order
  • Data that is in correct order but not yet read by application
TCP Flow Control

• TCP connection created between Hosts A, B
  • Buffers play important role in conversation between A and B!!!
    • May be slow to read information
    • Not required to be in sync with sender
  • Sender can overflow buffer – send data too fast
  • How to coordinate sender/receiver flow?
• Use something called, Receiver Window
TCP Flow Control

• TCP is a sliding window protocol

  • For window size $n$, can send up to $n$ bytes without receiving an Acknowledgment

  • When data is acknowledged then window slides forward

  • Each packet advertises a window size
  • Indicates number of bytes for which receiver has space

Short example on Sliding Window
http://wikistack.com/what-is-tcp-sliding-window-protocol/
Example - Window Size is 7

Sender window

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6
```

Receiver Window

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6
```

Frame 0, Frame 1 transmitted

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6
```

ACK2 Received

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6
```

Frame 2,3,4 transmitted

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6
```

ACK5

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6
```
TCP Flow Control

- **Receiver Advertises** window size to **sender** based on buffer size allocated for the connection
  - *Advertised Window* field in TCP header
- Sender cannot have more than
  - *Advertised Window* bytes of unacknowledged data
- Buffers are of finite size
  - **RcvBuffer**
  - **SendBuffer**
Setting the Advertised Window

- On TCP Receiver side,
  - \( \text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer} \)
- Thus, it advertises space left in buffer i.e.,

\[
\text{RcvWindow} = \text{RcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead})
\]

- \( \text{RcvWindow} \) becomes the Advertised Window

- As more data arrives i.e., more received bytes than read bytes, \( \text{LastByteRcvd} \) increases and hence, Advertised Window reduces
Sender Side Response

• At sender side, TCP sender should ensure that 
  \[ \text{LastByteSent} - \text{LastByteAcked} \leq \text{Advertised Window} \]

• In order to prevent overflow, if Advertised Window goes to zero ....

  • If application tries to write more, TCP blocks 
    TCP Sender
TCP Receiver Window

- Look at a couple of examples how Receiver Window changes

Client keeping up with data sent by Server
TCP Receiver Window

Client not keeping up with data sent by Server – Reduced Window
TCP Congestion Control
Principles of Congestion Control

Congestion: What causes congestion?

• Informally: “too many sources sending too much data too fast for network to handle”
• Different from flow control!
• Problem is result of many senders/receivers

• Symptoms of Congestion
  • Lost packets (buffer overflow at routers)
  • Long delays (queuing at routers)
Performance Metrics

• Two important performance measures

• Throughput
  — Data rate in bps - protocol overhead
    (trans. delay, propagation, queueing and processing delays)

• Delay
  — Transmission delay
    • Time for transmitter to send all bits of packet
  — Propagation delay
    • Time for one bit to transit from source to destination
  — Processing delay
    • Time required to process packet at source prior to sending, at any intermediate router or switch

• We discussed these. What we did not discuss is
  — Queuing delay: Time spent while waiting in queues
Queuing Delays

• Queuing delays are significant in performance of communications networks
  — Grow dramatically as system approaches capacity

• In shared facility
  – Network,
  – Transmission line,
  – Road network,
  – Checkout lines

• Performance worsens exponentially as demand approaches capacity
What Is Congestion?

- Data network is basically a network of queues
- Congestion occurs when number of packets being transmitted through network approaches packet handling capacity of network

Input and Output Queues at a Network Node
Congestion is unavoidable
Arguably it’s good!

• Packet switching makes efficient use of links
• So, router buffers **should** be occupied
• If buffers are always empty, delay is low ... but
  • Usage of network is low too
• If buffers are always occupied, delay is high
  • Using network more efficiently

• Goal is to manage congestion through protocols!!!
Response to Congestion

• Most versions of transmission protocols use timeout and/or retransmission
  – Even UDP retransmits if not successful
• Respond to increased delay by retransmitting datagrams, increasing congestion
• This leads to congestion collapse (next slide)
• So, TCP must reduce transmission rates when congestion occurs
Congestion Collapse

- Congestion collapse is a case in which congestion prevents or limits useful communication.
- Congestion collapse generally occurs at choke points in the network.
  - Incoming traffic exceeds outgoing bandwidth.
- Congestive collapse was identified as a possible problem by 1984.
- When packet loss occurred, endpoints sent extra packets that repeated information lost, doubling the incoming rate.
What Options are Available for Network Congestion Control?

1. Host versus network
   - Endpoints deal with congestion vs. routers

2. Reservation-based, versus Feedback-based
   - Reservation based is to reserve bandwidth as in Circuit Switched Networks
   - Feedback allows adaptation as conditions change

3. Window-based versus Rate-based
   - Window sets an amount for segments in flight
   - Rate based negotiates a rate at time of connection
TCP Congestion Control

- TCP implements
  - Host-based,
  - Feedback-based,
  - Window-based

- TCP sources attempt to determine how much capacity is available
- TCP sends packets, then reacts to observable events (loss)
TCP Congestion Control

• TCP doesn’t rely on IP layer to provide congestion control
• Limits rate at which senders can send traffic using perceived congestion on path

• Questions:
  • How to detect congestion?
  • How to limit congestion?
  • What algorithm to use to limit traffic?
TCP Congestion Control

- **Old Days ...**
  - Older TCP stacks would start a connection with sender injecting multiple segments
    - Up to window size advertised by receiver
    - No real congestion control, only flow
  - While this is OK when two hosts on same LAN,
    - Could be many routers and slow links between sender and receiver ... so
  - Intermediate router(s) packet queues run out of space
TCP Congestion Control

- **Main Goal of TCP Congestion Control!**
  - **Avoidance!!!**
  - Try to detect congestion and reduce load
  - Because once congestion happens
    - Happens at exponential rate
    - Then, takes a long time for queues to drain
  - Want to keep network pipes full but without danger of saturation
TCP Congestion Control

• Overall Strategy - simple
  • Good times, no congestion
    -> Increase sending rate

• Bad times, congestion
  -> Decrease sending rate

• Details ... a little more complex
TCP Congestion Control

How does the sender detect congestion?

- Concrete evidence...
  - Loss event
    - Timeout, or
    - 3 duplicate ACK's

- What could you conclude about congestion if you get back Acks?
  - Acks getting back, conclude that congestion is not extreme ...
    ... network not completely deadlocked
TCP Congestion Control

How does sender limit congestion?

- TCP sender reduces rate after loss event
  - TCP has one more variable – CongWin
  - Amount of unACKed data sender can send
  - Separate from RcvWin – flow control variable

- What does it do?
  - Imposes constraint on rate sender sends traffic into network
  - Think of the Congestion Window like this ...
TCP Congestion Control

How does sender limit congestion?

• The Congestion window is flow control imposed by the sender
• While Receiver window is flow control imposed by the receiver

• Congestion Window based on sender's
  • Assessment of perceived network congestion
• Receiver Window is related
  • Amount of available buffer space at the receiver for this connection.
TCP Congestion Control

- Essential strategy
- TCP host sends packets into network without reservation and then host reacts to observable events.
  - Originally TCP assumed FIFO queuing
- Basic idea Each source determines how much capacity is available to a flow
- ACKs are used to ‘pace’ transmission of packets such that TCP is “self-clocking”
TCP Congestion Control

• What algorithm(s) to use to limit traffic? 3 of them
  • **AIMD** – Additive Increase Multiplicative Decrease
  • **Slow start** (SS)
  • **Congestion Avoidance** (CA) - After timeout events

• **Basically,**
  • Algorithms allow TCP to probe bandwidth
  • Increases rate in response to ACK's until loss occurs
  • Decreases rate strategically
TCP Congestion Control: details

- Adds one more constraint based on congestion
- So, sender limited by **Either** Congestion or Flow
- **Sender** limits transmission:
  \[ \text{LastByteSent - LastByteAcked} < \min (\text{CongWin}, \text{RcvWin}) \]

- Roughly, if we ignore RcvWin ...

\[
\text{SendRate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}
\]

- **CongWin** is dynamic, function of perceived network congestion
  - Set to 1 MSS at beginning of connection
TCP Congestion Control

• Say, detect a loss (Lost) event
• Basic Idea
  • TCP Sender decreases send rate by decreasing CongWin size
  • Question is, by how much?

• Multiplicative Inverse
  • Half the current value of CongWin size

**Example**

<table>
<thead>
<tr>
<th>CongWin</th>
<th>Can’t go lower than 1 MSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 Kb</td>
<td>10 Kb</td>
</tr>
<tr>
<td>5 Kb</td>
<td>1 loss</td>
</tr>
<tr>
<td></td>
<td>2 loss</td>
</tr>
</tbody>
</table>
TCP Congestion Control

• But, if no Detected Loss
  • TCP increases its CongWin by 1 MSS, up to maximum, adjustable by system administrator
  • Additive Increase
    • Cautious increase since its probing for additional bandwidth
    • Assume there is unused bandwidth, could possibly take advantage of
    • Does this every RTT !!!!!

• Thus, TCP
  • Decreases its rate multiplicatively if congestion
  • Increases its rate additively when no congestion
TCP Congestion Control: Additive Increase, Multiplicative Decrease

- **Approach**  Increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - **Additive Increase:** Increase $\text{CongWin}$ by 1 MSS every RTT until loss detected
  - **Multiplicative Decrease:** Cut $\text{CongWin}$ in half after loss

Saw tooth behavior: **probing** for bandwidth
TCP Slow Start

• However, that wasn’t fast enough ....

• **When Connection Begins**, $\text{CongWin} = 1$ MSS
  • Example: MSS = 500 bytes & RTT = 200 msec
  • Initial rate = MSS/RTT or roughly 20 kbps

• Available bandwidth may be **more than** MSS/RTT
  • Want to quickly ramp up to respectable rate
  • Or, waste bandwidth

• **So ... when connection begins**
  • Increase rate exponentially fast until first loss event
TCP Slow Start

- When connection begins, increase rate exponentially until first loss event:
  - Double **CongWin** every RTT
  - Done by incrementing **CongWin** for every ACK received
- **Summary:** Initial rate is slow but ramps up exponentially fast
TCP Slow Start

- Philosophy Behind Slow Start ....
- It operates by
- Observing that rate at which new packets should be injected into network
- Is rate at which acknowledgments are returned by other end
Refinement
“Reno”

Congestion Avoidance

Implementation
• Yet another variable, Threshold
• Window size where slow start ends and linear increase begins, Congestion Avoidance
• Reno: At loss event, Threshold is set to 1/2 of CongWin just before loss event
• Tahoe: At loss event, Threshold is ½ of CongWin, Window is 1 MSS and Slow Start
Major TCP Variants

- **TCP Tahoe (older)**
  - Assumed congestion signals = Lost segments
  - Losses due to packet corruption less frequent than buffer overflows on routers
  - Resorts back to Slow Start, until threshold where it does Congestion Avoidance, increase by 1 MSS

- **TCP Reno**
  - Changed way it reacts to a loss
  - If sender is still getting Acks, congestion is not so heavy, so sender can still send, since flow still exists
  - But, sender should send with less vigor
  - So, does not fall back to slow start
More Refinement: Inferring loss

• After 3 dup ACKs:
  • CongWin is cut in half
  • Window then grows linearly

• But after timeout event:
  • CongWin instead set to 1 MSS;
  • Window then grows exponentially
  • To a threshold, then grows linearly

Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- Timeout indicates a “more alarming” congestion scenario
Summary: TCP Congestion Control

- When \texttt{CongWin} is below \texttt{Threshold}, sender in \texttt{slow-start} phase, window grows exponentially
- When \texttt{CongWin} is above \texttt{Threshold}, sender is in congestion-avoidance phase, window grows linearly
- When a triple duplicate ACK occurs, \texttt{Threshold} set to \texttt{CongWin}/2 and \texttt{CongWin} set to \texttt{Threshold}
- When \texttt{timeout} occurs, \texttt{Threshold} set to \texttt{CongWin}/2 and \texttt{CongWin} is set to 1 MSS
Many TCP ‘flavors’

- **TCP New Reno** – Variation of Reno,
- **TCP Vegas**
  - Adjusts window size based on difference between expected and actual RTT
- **TCP Binary Increase Congestion (BIC)**
  - Uses optimized congestion control algorithm for high speed networks with high latency
- **TCP Cubic**
  - TCP Cubic allows very fast window expansion however, it also makes attempts to slow the growth of cwnd sharply as cwnd approaches the current network ceiling, it is the default in Linux

Nice overview of newer TCP Congestion Algorithms
http://intronetworks.cs.luc.edu/current/html/newtcps.html#tcp-cubic
## TCP Sender Congestion Control

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>TCP Sender Action</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow Start (SS)</td>
<td>ACK receipt for previously unacked</td>
<td>$\text{CongWin} = \text{CongWin} + \text{MSS}$, If $\text{CongWin} &gt; \text{Threshold}$ set state to “Congestion Avoidance”</td>
<td>Resulting in a doubling of CongWin every RTT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Congestion Avoidance (CA)</td>
<td>ACK receipt for previously unacked</td>
<td>$\text{CongWin} = \text{CongWin} + \text{MSS} \times \left(\frac{\text{MSS}}{\text{CongWin}}\right)$</td>
<td>Additive increase, resulting in increase of CongWin by 1 MSS every RTT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS or CA</td>
<td>Loss event detected by triple duplicate ACK</td>
<td>$\text{Threshold} = \text{CongWin}/2$, $\text{CongWin} = \text{Threshold}$, Set state to “Congestion Avoidance”</td>
<td>Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS or CA</td>
<td>Timeout</td>
<td>$\text{Threshold} = \text{CongWin}/2$, $\text{CongWin} = 1$ MSS, Set state to “Slow Start”</td>
<td>Enter slow start</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS or CA</td>
<td>Duplicate ACK</td>
<td>Increment duplicate ACK count for segment being acked</td>
<td>CongWin and Threshold not changed</td>
</tr>
</tbody>
</table>
Summary

- TCP interacts with routers and reacts to implicit congestion notification,
  - Packet drop
- By reducing TCP sender’s congestion window
- TCP increases congestion window using slow start or congestion avoidance
- Does this for every user in the network

Currently, there are many newer TCP algorithms for congestion control
Each has slightly different attributes for optimal congestion control
Assignment due Monday – WebServer
Midterm next Wednesday – Take home