Lecture 15 – Transport Protocols
Eric Anderson

Fall 2013
www.cs.cmu.edu/~prs/15-441-F13
Outline

- Transport introduction
- Review: Error recovery & flow control
- TCP flow control
- Introducing congestion control
Transport Protocols

- Lowest level end-to-end protocol.
  - Header generated by sender is interpreted only by the destination
  - Routers view transport header as part of the payload
Functionality Split

- Network provides best-effort delivery
- End-systems implement many functions
  - Reliability
  - In-order delivery
  - Demultiplexing
  - Message boundaries
  - Connection abstraction
  - Congestion control
  - ...

Transport Protocols

- UDP provides just integrity and demux
- TCP adds...
  - Connection-oriented
  - Reliable
  - Ordered
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled
UDP: User Datagram Protocol [RFC 768]

- “No frills,” “bare bones” Internet transport protocol
- “Best effort” service, UDP segments may be:
  - Lost
  - Delivered out of order to app
- Connectionless:
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

Why is there a UDP?
- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver
- Small header
- No congestion control: UDP can blast away as fast as desired
UDP, cont.

- Often used for streaming multimedia apps
  - Loss tolerant
  - Rate sensitive

- Other UDP uses (why?):
  - DNS, SNMP

- Reliable transfer over UDP
  - Must be at application layer
  - Application-specific error recovery

UDP segment format:

<table>
<thead>
<tr>
<th>Source port #</th>
<th>Dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

Length, in bytes of UDP segment, including header

Application data (message)
High-Level TCP Characteristics

- Protocol implemented entirely at the ends
  - Fate sharing
- Protocol has evolved over time and will continue to do so
  - Nearly impossible to change the header
  - Use options to add information to the header
    - These do change sometimes
  - Change processing at endpoints
  - Backward compatibility is what makes it TCP
## TCP Header

<table>
<thead>
<tr>
<th>Flags:</th>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence number</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Acknowledgement</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Advertised window</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Urgent pointer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Options (variable)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
</table>

Evolution of TCP

1975
Three-way handshake
Raymond Tomlinson
In SIGCOMM 75

1974
TCP described by
Vint Cerf and Bob Kahn
In IEEE Trans Comm

1982
TCP & IP
RFC 793 & 791

1983
BSD Unix 4.2
supports TCP/IP

1984
Nagel’s algorithm
to reduce overhead
of small packets;
predicts congestion collapse

1985
Congestion collapse
observed

1986
Van Jacobson’s algorithms
congestion avoidance
and congestion control
(most implemented in
4.3BSD Tahoe)

1987
Karn’s algorithm
to better estimate
round-trip time

1988

1990
4.3BSD Reno
fast retransmit
delayed ACK’s
TCP Through the 1990s

- 1993: TCP Vegas (Brakmo et al) delay-based congestion avoidance
- 1994: ECN (Floyd) Explicit Congestion Notification
- 1994: T/TCP (Braden) Transaction TCP
- 1996: SACK TCP (Floyd et al) Selective Acknowledgement
- 1996: Hoe NewReno startup and loss recovery
- 1996: FACK TCP (Mathis et al) extension to SACK
TCP Through the 2000s

- 2004 NewReno
  (Floyd et. al.)
  Partial ACK in Fast Recovery

- 2007 CUBIC
  Rhee, Xu, Ha
  Convex-Concave Response Fn.

- 2010 Data Center TCP
  (too many authors)
  ECN, proportional window scaling

- 2011 Multi-Path TCP
  Barré, Bonaventure
  TCP over multiple subflows
Outline

• Transport introduction
• Error recovery & flow control
• TCP flow control
• Introducing congestion control
Review: Stop and Wait

- ARQ
  - Receiver sends acknowledgement (ACK) when it receives packet
  - Sender waits for ACK and timeouts if it does not arrive within some time period
- Simplest ARQ protocol
- Send a packet, stop and wait until ACK arrives
How to Keep the Pipe Full?

- Send multiple packets without waiting for first to be acked
  - Number of pkts in flight = window
- Reliable, unordered delivery
  - Several parallel stop & waits
  - Send new packet after each ack
  - Sender keeps list of unack’ed packets; resends after timeout
  - Receiver same as stop & wait
- How large a window is needed?
  - Suppose 10Mbps link, 4ms delay, 500byte pkts
    - $1? \ 10? \ 20?$
    - Round trip delay * bandwidth = capacity of pipe
Sliding Window

- Reliable, ordered delivery
- Receiver has to hold onto a packet until all prior packets have arrived
  - Why might this be difficult for just parallel stop & wait?
  - Sender must prevent buffer overflow at receiver
- Circular buffer at sender and receiver
  - Packets in transit $\leq$ buffer size
  - Advance when sender and receiver agree packets at beginning have been received
Sender/Receiver State

Sender

- Max ACK received
- Next seqnum
- Sender window
- Sent & Acked
- Sent Not Acked
- OK to Send
- Not Usable

Receiver

- Next expected
- Max acceptable
- Receiver window
- Received & Acked
- Acceptable Packet
- Not Usable
Important Lessons

• Transport service
  • UDP → mostly just IP service
  • TCP → congestion controlled, reliable, byte stream

• Types of ARQ protocols
  • Stop-and-wait → slow, simple
  • Go-back-n → can keep link utilized (except w/ losses)
  • Selective repeat → efficient loss recovery

• Sliding window flow control
  • Addresses buffering issues and keeps link utilized
Next Lecture

- Congestion control
- TCP Reliability
Outline

- Transport introduction
- Error recovery & flow control
- TCP flow control
- Introducing congestion control
Bandwidth-Delay Product

\[
\text{Max Throughput} = \frac{\text{Window Size}}{\text{Roundtrip Time}}
\]
Automatic Repeat Request (ARQ)

- Sender retransmits packet after timeout
- Recognize retransmissions using sequence numbers
  - both packets and acks
  - How big should it be?
  - For stop&wait, sliding window?
- Ack can acknowledge one packet or all earlier packets
  - “Selective” vs cumulative
Sequence Numbers

- How large do sequence numbers need to be?
  - Must be able to detect wrap-around
  - Depends on sender/receiver window size

- Example
  - Assume seq number space = 7, window = 7
  - If pkts 0..6 are sent successfully and all acks lost
    - Receiver expects 7,0..5, sender retransmits old 0..6!!!

- Condition: max window size < size sequence number space
Sequence Numbers

- 32 Bits, Unsigned $\rightarrow$ for bytes not packets!
  - Circular Comparison

- Why So Big?
  - For sliding window, must have $|\text{Sequence Space}| > |\text{Window}|$
    - No problem
  - Also, want to guard against stray packets
    - With IP, packets have maximum lifetime of 120s
    - Sequence number would wrap around in this time at 286MB/s
TCP Flow Control

- TCP is a sliding window protocol
  - For window size $n$, can send up to $n$ bytes without receiving an acknowledgement
  - When the data is acknowledged then the window slides forward
- Each packet advertises a window size
  - Indicates number of bytes the receiver has space for
- Original TCP always sent entire window
  - Congestion control now limits this
Window Flow Control: Send Side

- Sent and acked
- Sent but not acked
- Not yet sent

Next to be sent
Window Flow Control: Send Side

Packet Sent

Source Port   Dest. Port
Sequence Number
Acknowledgment
HL/Flags      Window
D. Checksum   Urgent Pointer
Options...

Packet Received

Source Port   Dest. Port
Sequence Number
Acknowledgment
HL/Flags      Window
D. Checksum   Urgent Pointer
Options...

App write

acknowledged  sent  to be sent  outside window
Window Flow Control: Receive Side

What should receiver do?

Receive buffer

Acked but not delivered to user

Not yet acked

window

New
Aside: TCP Persist

• What happens if window is 0?
  • Receiver updates window when application reads data
  • What if this update is lost?

• TCP Persist state
  • Sender periodically sends 1 byte packets
  • Receiver responds with ACK even if it can’t store the packet
Performance Considerations

- The window size can be controlled by receiving application
  - Can change the socket buffer size from a default (e.g. 8Kbytes) to a maximum value (e.g. 64 Kbytes)
- The window size field in the TCP header limits the window that the receiver can advertise
  - 16 bits $\rightarrow$ 64 KBytes
  - 10 msec RTT $\rightarrow$ 51 Mbit/second
  - 100 msec RTT $\rightarrow$ 5 Mbit/second
- TCP options to get around 64KB limit $\rightarrow$ increases above limit
Outline

- Transport introduction
- Error recovery & flow control
- TCP flow control
- Introducing congestion control
Internet Pipes?

- How should you control the faucet?
Internet Pipes?

- How should you control the faucet?
- Too fast – sink overflows!
Internet Pipes?

- How should you control the faucet?
  - Too fast – sink overflows!
  - Too slow – what happens?
Internet Pipes?

• How should you control the faucet?
  • Too fast – sink overflows
  • Too slow – what happens?

• Goals
  • Fill the bucket as quickly as possible
  • Avoid overflowing the sink

• Solution – watch the sink
Plumbers Gone Wild!

- How do we prevent water loss?
- Know the size of the pipes?
Plumbers Gone Wild 2!

- Now what?
- Feedback from the bucket or the funnels?
Congestion

- Different sources compete for resources inside network
- Why is it a problem?
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queuing in router buffers)
  - Can result in throughput less than bottleneck link (1.5Mbps for the above topology) \(\Rightarrow\) a.k.a. congestion collapse
Causes & Costs of Congestion

- When packet dropped, any “upstream transmission capacity used for that packet was wasted!
Congestion Collapse

- Definition: *Increase in network load results in decrease of useful work done*

- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - How can this happen with packet conservation
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic
Congestion Control and Avoidance

- A mechanism that:
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids collapse

- Congestion collapse is not just a theory
  - Has been frequently observed in many networks
Approaches Towards Congestion Control

- Two broad approaches towards congestion control:
  - End-end congestion control:
    - No explicit feedback from network
    - Congestion inferred from end-system observed loss, delay
    - Approach taken by TCP
  - Network-assisted congestion control:
    - Routers provide feedback to end systems
      - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
      - Explicit rate sender should send at
    - Problem: makes routers complicated
TCP Congestion Control

- Very simple mechanisms in network
  - FIFO scheduling with shared buffer pool
  - Feedback through packet drops
- TCP interprets packet drops as signs of congestion and slows down
  - This is an assumption: packet drops are not a sign of congestion in all networks
    - E.g. wireless networks
- Periodically probes the network to check whether more bandwidth has become available.
Objectives

- Simple router behavior
- Distributedness
- Efficiency: $X = \sum x_i(t)$
- Fairness: $\frac{(\sum x_i)^2}{n(\sum x_i^2)}$
  - What are the important properties of this function?
- Convergence: control system must be stable
Basic Control Model

- Reduce speed when congestion is perceived
  - How is congestion signaled?
    - Either mark or drop packets
  - How much to reduce?
- Increase speed otherwise
  - Probe for available bandwidth – how?
Linear Control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
    - $\text{Window}(t + 1) = a + b \text{Window}(t)$
    - Different $a_i/b_i$ for increase and $a_d/b_d$ for decrease
- Supports various reaction to signals
  - Increase/decrease additively
  - Increased/decrease multiplicatively
  - Which of the four combinations is optimal?
Phase Plots

- Simple way to visualize behavior of competing connections over time
Phase Plots

- What are desirable properties?
- What if flows are not equal?
Additive Increase/Decrease

- Both $X_1$ and $X_2$ increase/ decrease by the same amount over time
  - Additive increase improves fairness and additive decrease reduces fairness
Multiplicative Increase/Decrease

- Both $X_1$ and $X_2$ increase by the same factor over time
  - Extension from origin – constant fairness
Convergence to Efficiency

![Graph showing the efficiency line and the fairness line with User 1's Allocation $x_1$ and User 2's Allocation $x_2$.]
Distributed Convergence to Efficiency

- User 1’s Allocation $x_1$
- User 2’s Allocation $x_2$

- Fairness Line
  - $a=0$
  - $b=1$
  - $a>0 \& b>1$
  - $a<0 \& b>1$

- Efficiency Line
  - $a>0 \& b<1$
  - $a<0 \& b<1$
Convergence to Fairness
Convergence to Efficiency & Fairness

- Intersection of valid regions
- For decrease: $a=0 \& b < 1$
What is the Right Choice?

- Constraints limit us to AIMD
  - Can have multiplicative term in increase (MAIMD)
  - AIMD moves towards optimal point

![Diagram showing user allocations and lines representing fairness and efficiency.](Image)
Important Lessons

- Transport service
  - UDP $\rightarrow$ mostly just IP service
  - TCP $\rightarrow$ congestion controlled, reliable, byte stream

- Types of ARQ protocols
  - Stop-and-wait $\rightarrow$ slow, simple
  - Go-back-n $\rightarrow$ can keep link utilized (except w/ losses)
  - Selective repeat $\rightarrow$ efficient loss recovery

- Sliding window flow control

- TCP flow control
  - Sliding window $\rightarrow$ mapping to packet headers
  - 32bit sequence numbers (bytes)
Important Lessons

• Why is congestion control needed?

• How to evaluate congestion control algorithms?
  • Why is AIMD the right choice for congestion control?

• TCP flow control
  • Sliding window → mapping to packet headers
  • 32bit sequence numbers (bytes)
Good Ideas So Far...

- Flow control
  - Stop & wait
  - Parallel stop & wait
  - Sliding window
- Loss recovery
  - Timeouts
  - Acknowledgement-driven recovery
    - Selective repeat versus go-back-N
    - Cumulative acknowledgement
- Congestion control
  - AIMD → fairness and efficiency
- Next Lecture: How does TCP actually implement these?