TDC 563
Protocols and Techniques for Data Networks

Transport Layer
Why a transport layer?

- IP gives us end-to-end connectivity doesn't it?
- Why, or why not, more than one transport layer?
- What does a transport layer typically do?
  - Process identification
  - Reliability
  - Flow control
- Are there times when those are unnecessary?
- What are the security / performance issues?
If no L4, HTTP data goes where?

<table>
<thead>
<tr>
<th>HTTP data</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPs: 192.0.2.1</td>
</tr>
<tr>
<td>IPd: 192.0.2.2</td>
</tr>
<tr>
<td>Proto: n/a</td>
</tr>
</tbody>
</table>

Thanks!!
But...
How to get this to my to HTTPd?
Try to parse and interpret!?!? Help me out dude!!
Application multiplexing

- OS independent identifier for a network process
- Each process assigned a locally unique 16-bit id
- Server (listener) apps
  - tend to use standard, “well-known” port numbers
  - see /etc/services (UNIX / MacOS X)
- Client (opener) apps
  - tend to use ephemeral (dynamic) port numbers
  - usually >1023, range depends on OS and app
- See http://www.iana.org/assignments/port-numbers
# User Datagram Protocol (UDP)

| bit  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|      | Source Port | Destination Port |
|      | Length      | Checksum        |
|      | Data        |
|      | ...         |
UDP is very simple

• Basically just an application multiplexer
• The length field is practically redundant
  • IP total length – 8 = UDP payload
• Source port may be zero if no reply expected
• Even the checksum is optional!
  • though it is inexpensive, recommended to use it
• No flow control, reliability, nor connection setup
• Why is this good, how could this be bad?
What uses UDP?

- SNMP, TFTP, DHCP, syslog, NetFlow
- Streaming real-time media (VoIP, radio, video)
  - some use TCP (thank you security dilettantes)
- DNS, NTP
  - if not for these, UDP death by filtering likely
Common IP transport protocols

- UDP – very common, but usually low rate of pkts
- TCP – most common, typically most of your pkts
- Some usage, but not widely deployed:
  - UDP-Lite
  - SCTP
  - DCCP
- Note, not all IP protocols considered a “transport”
  - e.g. we don't think of ICMP/IGMP as a transport
IP review

- IP provides just enough *connected-ness*
  - global addressing
  - hop-by-hop routing
- IP over everything
  - Ethernet, ATM, X.25, fiber, etc.
- Minimizes in-network functionality
- Unreliable datagram forwarding
TCP key features

• Sequencing
• Byte-stream delivery
• Connection-oriented
• Reliability
• Flow-control
• Congestion avoidance
TCP feature summary

Provides a completely reliable (no data duplication or loss), connection-oriented, full-duplex byte stream transport service that allows two application programs to form a connection, send data in either direction simultaneously and then terminate the connection.
Apparent contradiction

- IP offers best effort (unreliable) delivery
- TCP uses IP
- TCP provides completely reliable transfer
- How is this possible?
Achieving reliability

- Reliable connection start-up
- Reliable data transfer
  - sender starts a timer
  - receiver sends ACK when data arrives
  - sender retransmits if timer expires before ACK is returned
- Reliable connection shutdown
TCP connection start-up

- A “three-way handshake” is used
- Servers use a passive open
  - application sits waiting on an open port
- Clients use an active open
  - application requests a connection to server
- Initial sequence number (ISN) exchange is the primary goal
- Other parameters/options can also be exchanged
  - e.g. window scale, maximum segment size, etc.
3-way handshake illustrated

Time

Host A

Send SYN seq=x

Receive SYN + ACK

Send ACK y+1

In the Network

Receive SYN

Send SYN seq=y, ACK x+1

Receive ACK

Host B
Byte stream sequencing

• Each segment carries a sequence number
• Sequencing helps ensure in order delivery
• TCP sequence numbers are fixed at 32 bits
  • byte stream is not limited to $2^{32}$ bytes
  • sequence number space can wrap
• Each side has an initial sequence number (ISN)
  • exchanged during connection establishment
• Receiver ACKs cumulative octets (bytes)
Reliability illustrated
*diagram courtesy of http://www.netbook.cs.purdue.edu*
When do you retransmit?

• The time for an ACK to return depends on:
  • distance between endpoints (propagation delay)
  • network traffic conditions (congestion)
  • end system conditions (CPU, buffers)
• Packets can be lost, damaged or fragmented
• Network traffic conditions can change rapidly
Solving retransmission problem

- Keep running average of round trip time (RTT)
- Current average determines retransmission timer
- This is known as adaptive retransmission
- This is key to TCP's success
- How does each RTT sample affect the average?
  - what weight to you give each sample?
  - higher weight means timer changes quickly
  - lower weight means timer changes slowly
Adaptive retransmission illustrated
*diagram courtesy of http://www.netbook.cs.purdue.edu*
Connection shutdown illustrated
Flow control

• Match the sending rate with allowable receiver rate
• TCP uses a sliding window
  • receiver advertises available buffer space
  • also known as the window
  • sender can transmit a full window without receiving an ACK for that transmitted data
• Ideally the window size allows pipe to remain full
Window size advertisement

- Each ACK carries receiver's current window size
  - called the window advertisement (aka rwnd)
  - if zero, window is closed, no data can be sent
- Interpretation of window advertisement:
  - receiver: I can accept $X$ octets or less unless I tell you otherwise
Window size illustrated
*diagram courtesy of http://www.netbook.cs.purdue.edu
Window size: another picture

(a)

(b)

(c)
TCP segment illustrated

<table>
<thead>
<tr>
<th></th>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acknowledgment Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HLEN</th>
<th>rsrvd</th>
<th>CE</th>
<th>WR</th>
<th>C</th>
<th>EG</th>
<th>AP</th>
<th>CS</th>
<th>PR</th>
<th>RS</th>
<th>SY</th>
<th>IN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Window Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Urgent Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>options (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>padding (if required)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Congestion principles

- Flow control
  - matching the sending and receiving rates
- Congestion control
  - active response to network overload conditions
  - end hosts cannot control congestion per se
  - network devices (routers) do this
- Congestion avoidance
  - cautionary response to presumed conditions
  - TCP does this
TCP congestion control

- Recall sliding window (advertised window)
  - receiver based control of sending rate
- Congestion window is sender based control
- Sender transmits min(cwnd, rwnd)
  - this value is the transmission window
- TCP sender infers network conditions and adjusts
- See IETF RFC 5681
TCP retransmission

- TCP starts timer after sending a segment
- If ACK returns, reset timer
- If time-out occurs, retransmit and increase timer
  - this is a back-off process
- Can't retransmit forever, need some upper bound
- Eventually TCP would give up
  - maximum time-out must be at least 60 seconds
Estimating round trip time (RTT)

- TCP measures RTT for which to calculate timers
- If ACKs return quickly, timers should be short
  - if loss occurs, recovery happens quickly
- If ACKs return slowly, timers should be long
  - if delays occur, retransmits not sent needlessly
- Keep a smoothed running average of RTT
  - smoothed RTT used to adjust retransmit timer
  - Karn's algorithm says ignore ACKs of retransmits
TCP slow start

• Recall that min(cwnd, rwnd) = transmission window
• Rather than sending a full window at start-up...
• Initialize cwnd to 1 maximum segment size (MSS)
• Increase cwnd by 1 MSS for every ACK returned
• Obviously don't go past advertised window!
• This can actually be quite fast, exponential!
TCP slow start illustrated
TCP congestion avoidance

- If a retransmission timer expires, slow down
- Set slow start threshold = transmission window $\times \frac{1}{2}$
  - this is sshthresh
- Set cwnd back to 1 MSS
- Transmit $\min(cwnd, rwnd)$ as usual
- Do slow start until transmission window = sshtresh
- Thereafter, increase cwnd by $\frac{1}{cwnd}$ per ACK
  - linear increase instead of exponential
Congestion avoidance illustrated

- Exponential increase
- Linear increase
- Timeout

Transmission window vs. time
Duplicate ACKs

• Recall ACKs acknowledge cumulative octets
• TCP receiver sends an immediate ACK if it receives an out-of-order segment
• This is a duplicate ACK
• This dupe ACK informs the sender and tells it what sequence number the receiver expected
• It's unclear whether dupe ACKs indicate loss or simply packet re-ordering on the network
• But, multiple duplicate ACKs probably indicate loss
TCP fast retransmit

• If sender gets >=3 dupe ACKs, assume loss
• Immediately retransmit, don't wait for timer to expire
• Goto fast recovery
TCP fast recovery

- Duplicate ACKs indicate data is still flowing
- If there was a loss event, it was probably temporary
- Go directly to congestion avoidance
  - not all the way into slow start!
  - don't want to start off with just a 1 MSS window
- This is the fast recovery algorithm
  - minus a few minor details
Other TCP stuff

- Selective ACK (SACK) option (re-tx optimization)
- Window scale option (for high capacity paths)
- Timestamp option (RTT measurement, PAWS)
- Persist timer (window probes)
- Keepalive timer (disabled by default, app can set)
- Nagle algorithm (sender side)
- Silly window syndrome (receiver side)