CSCI 339
Distributed Systems

Lecture 4
Layer 4: TCP and UDP
Feb 13, 2023
Administrative Details

• Written homework 2 is due Thursday on Glow
• Web server stress testing tool: Apache benchmark (ab)
  • Installed on all lab machines
  • Don’t worry about this until you have mostly complete implementation
• Web server is due next Wed, Feb 22nd
  • Milestone due tonight!
  • Don't forget about correctly formatted HTTP headers in response
• **No class next Monday Feb 20?**
  • TBD. Will confirm on Thursday
• Questions??
Last Time

• We learned about abstractions in network protocols
  • Network stack “defines” abstractions (as layers)
  • Two different types of addresses:
    • Layer 3 (IP) and Layer 2 (Ethernet/WiFi)
  • Both are important!
• Talked about the E2E argument
• Any questions?
Today's Outline

• Discuss routing
• Discuss transport layer protocols (layer 4)
  • UDP - User Datagram Protocol
  • TCP - Transmission Control Protocol
• Focus on flow control in TCP
How Does a Packet Get Through the Internet?

Routers send packet to next closest point

137.165.10.100

142.250.72.100

H: Hosts
R: Routers

Each router uses IP prefix matching to get our packet one step closer to Google. **Dest Ethernet addr on packet gets updated with each hop!** Dest IP addr never changes...
How Does a Packet Get Through the Internet?

Routers send packet to next closest point.

Each router uses IP prefix matching to get our packet one step closer to Google. **Dest Ethernet addr on packet gets updated with each hop!** **Dest IP addr never changes...**
How Do the Routers Know Where to Send Data?

- Forwarding tables at each router
- Original Internet: manual update
- Today: automatic update based on "cost"
  - Link State Protocol
    - Broadcast local connectivity information throughout the network periodically
    - Maintain table of cost to get to other destinations
    - Choose path with smallest cost (remember Dijkstra's?)
Link State Example

- Flood following information through the network
  - A: \((A \rightarrow C, 1) \ (A \rightarrow B, 1)\)
  - B: \((B \rightarrow A, 1) \ (B \rightarrow C, 4)\)
  - C: \((C \rightarrow A, 1) \ (C \rightarrow B, 4) \ (C \rightarrow D, 2)\)
  - D: \((D \rightarrow C, 2) \ (D \rightarrow E, 3)\)
  - E: \((E \rightarrow D, 3)\)

- Each host runs Dijkstra's algorithm to find shortest path to all other hosts
Have Address, Know How to Route, Now Send Data

• So now we know how to find Google and send data
• Real networks must deal with congestion, bit errors, etc. that occur during data transmission
  • Data can be corrupted
  • Data can get lost
  • Data might not fit in a single packet
  • Data can be delivered in the wrong order
  • ...
• None of this is layer 3/IP’s problem!
  • Need a higher layer abstraction
End-to-End Protocols

- Layers 2 & 3 (Ethernet/WiFi & IP) focus on delivering packets/frames of data to arbitrary hosts connected to the Internet
  - We have routing protocols for getting packets to destination (Link State Protocol)
  - IP is best effort delivery (provides **no reliability**)
End-to-End Protocols

• Layer 4 (TCP/UDP) focuses on arbitrary **processes** communicating together
  • Provide illusion that all processes are located on one large computer – hide the network!
  • `send()` and `recv()` systems calls are intentionally similar to `read()` and `write()`
  • Goal: Deliver data (reliably?) to any process running on any host
What If the Data Gets Corrupted?

Solution: Add a checksum
What If the Data Gets Lost?

Solution: *Timeout and retransmit (if no acknowledgement of receipt)*

GET index.html

Internet

wait a bit, retry

GET index.html

Internet

GET index.html

GET index.html
What If the Data Does Not Fit in a Single Packet?

On Ethernet, max IP packet is 1.5Kbytes
Typical web page is 10Kbytes (or more)

Solution: Fragment data across many packets (and reassemble at receiver)
What If the Data Arrives Out of Order?

Solution: Add sequence numbers

GET x.thindeml

GET index.html
What If Network is Overloaded?

- Data can arrive at router faster than it can be forwarded
- Short bursts: buffer at router (using a queue)
- What if buffer overflows?
  - Packets dropped and retransmitted
  - Sender decreases sending rate until congestion subsides
- Called "Congestion control"
  - Broadcast network (Ethernet): bus arbitration (exp backoff)
  - Wide-area networks: typically handled by TCP
  - Necessary for global fairness, but in the interests of individuals to ignore
Layer 4, Option 1: UDP

- User Datagram Protocol (UDP) - invented in 1980
  - Simple transport layer protocol
    - No guarantees about reliability, in-order delivery
    - Send and pray
  - "Thin veneer" on top of IP
    - Adds src/dest port numbers
    - 16 bit port number allows for identification of 65536 unique communication endpoints (i.e., services) per host
    - IP addr + port number uniquely identifies all Internet endpoints
  - UDP Datagram:

<table>
<thead>
<tr>
<th>Link-layer</th>
<th>IP</th>
<th>SrcPort</th>
<th>DestPort</th>
<th>Checksum</th>
<th>Len</th>
<th>Data…</th>
</tr>
</thead>
</table>

UDP Header
Layer 4, Option 2: TCP

- Transmission Control Protocol (TCP) - 1974/1982
  - Reliable in-order delivery of byte streams
  - Full duplex (endpoints simultaneously send/receive)
    - Two-way traffic
    - e.g., single socket for web browser talking to web server
  - Provides flow control
    - To ensure that sender does not overrun receiver
    - e.g., fast server talking to slow client
  - Provides congestion control
    - Keep the sender from overrunning the network
    - e.g., fast sender on low bandwidth Internet connection
    - Many simultaneous connections across routers (cross traffic)
TCP Flow & Congestion Control

- Sender must determine maximum amount of data in transit that will not overrun either receiver or network.
- Solutions?
TCP Flow Control

- Sender must determine maximum amount of data in transit that will not overrun receiver

- Solutions for flow control:
  - Maintain "sliding window" to track data in transit
  - Size of window determined by minimum of "flow window" and "congestion window"
  - Receiver ACKs "slide" left side of window forward (right)
    - Opens up another "slot" at right side of window for transmission

Data in transit
TCP "Sliding Window" Protocol Issues

• Need for connection establishment
  • No dedicated cable between end hosts
• Varying round trip times (RTTS) over life of connection
  • Different paths, different levels of congestion, etc
• Must be ready for very old packets to arrive
• Delay-bandwidth product highly variable
  • Amount of available buffer space at receivers also variable
• Sender has no idea what links will be traversed to receiver in advance
  • Must dynamically estimate changing end-to-end characteristics
  • How?
TCP Header Format

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>10</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>DestPort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SequenceNum</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>000000</td>
<td>Flags</td>
<td>AdvertisedWindow</td>
<td></td>
</tr>
<tr>
<td>CheckSum</td>
<td></td>
<td>UrgPtr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (variable – max of 320 bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Without options, TCP header 20 bytes
- IP header is also 20 bytes
  - Thus, typical Internet packet min of 40 bytes (+link layer header)
TCP Connection Establishment

- Exchange necessary information to begin communication
- Three-way handshake
  - E.g., server listening on socket

```
Client
SYN, sequence # = x
ACK, Acknowledgement = x + 1
SYN+ACK, sequence # = y
Server
ACK, Acknowledgement = y + 1
```
TCP Connection Teardown

• Each side of a TCP connection can independently close the connection
  • Thus, possible to have a half duplex connection
  • Possible problems?
  • Solutions?

• Closing process sends a FIN message
  • Waits for ACK of FIN to come back (so another 3-way handshake)
  • FIN: This side of the connection want to close
  • FIN+ACK: Ok, go ahead
  • ACK of FIN+ACK: Close
Reliability, First Cut: Stop and Wait

- Reliability, two principal mechanisms:
  - ACKs and timeouts
- Send a packet, stop and wait until acknowledgement arrives before sending next packet
- Problems?
Recovering From Error

- ACK lost
- Packet lost
- Early timeout/Delayed ACK

- Timeout
- Packet
- ACK
- ACK
- Timeout
- Packet
- ACK
- ACK
- Timeout
- Packet
- ACK
- ACK
- Timeout
- Packet
- ACK
- ACK
- Timeout
Problems with Stop and Wait

• How to recognize a duplicate transmission?
  • Solution: Put sequence number in packet

• Performance
  • Unless bandwidth-delay product is very small, sender cannot "fill the pipe"
  • Solution: Sliding window protocol with dynamically changing window size
Keeping the Pipe Full

- Bandwidth-Delay product measures network capacity
  - How much data can you put into the network before the first byte reaches the receiver
- Stop and Wait: 1 data packet per RTT (round trip time)
  - Compute throughput of 1.5-Mbps link with 45-ms RTT and 1KByte data packet
  - With Stop-and-wait: 182-Kbps throughput
    - 1 Kbyte = 1024x8 bits, Throughput = 8192 bits / 45 ms = 182 Kbps
- Ideally, send enough packets to fill the pipe before requiring first ACK packet
How Do We Keep the Pipe Full?

- Send multiple packets without waiting for first to be ACK'd
- Reliable, unordered delivery:
  - Send new packet after each ACK
  - Sender keeps list of unACK'd packets; resends after timeout
- Ideally, first ACK arrives immediately after pipe is filled
  - Opens up another "slot"
- Example: 10 Mbps link, 100 ms RTT:
  - How much data is needed to keep pipe full?
    - $10 \times 10^6 \text{bps} \times 100 \times 10^{-3} \text{s} = 1,000,000 \text{ bits} = 125 \text{ KB}$
Reliable, *In-Order Delivery & Flow Control*

- To support in-order delivery, add sequence number
  - Receivers buffer later packets until prior packets arrive
  - When a packet arrives out of order, receiver ACKs largest sequence # received *in order*
    - What happens when receiver receives 1, 2, 3, 5, 6, 7?
    - Receiver ACKs 1, 2, 3, 3, 3, 3

- Sender must still prevent buffer overflow at receiver
  - We can't forget about flow control

- Solution?
  - Sliding window with changing window size
  - Circular buffer at sender and receiver
    - # packets in transit <= buffer size
    - Advance window when sender and receiver agree packets at beginning have been successfully received
TCP Flow Control

- TCP is a sliding window protocol based on byte streams, not packets
  - 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 inside TCP header field
  - This number indicates number of bytes the receiver is willing to buffer
TCP Flow Control: Visualizing the Sliding Window

offered window = 6 bytes
(advertised by receiver)

usable window

Left side of window advances when data is acknowledged. Right side controlled by size of window advertisement.
Initial State, Receiver has 6 slots to buffer data
Bytes 4, 5, 6 sent, but not yet received

Sender

1 2 3
sent and acknowledged

4 5 6
sent, not ACKed

7 8 9
can send ASAP

10 11 12
can't send until window moves

Receiver

1 2 3
ACK'd and read

4 5 6
Available bufs

7 8 9

10 11 12
can't recv until window moves
Visualizing the Window: Example

Receiver to Sender ➞ ACK 5, Window 4

Sender

1 2 3 4 5

sent and acknowledged

6 7 8 9

advertised window

can send ASAP

can't send until window moves

Sender

1 2 3

ACK'd and read

4 5

offered window

Available bufs

10 11 12

Receiver

ACK'd, not read

Sender

1 2 3

ACK'd and read

4 5

Receiver

6 7 8 9

can't recv until window moves

Receiver

10 11 12
Visualizing the Window: Example

Sender to Receiver ➔ Send 7, 8, 9

- **Sender**: 1 2 3 4 5 6 7 8 9 10 11 12
  - sent and acknowledged
  - sent, not ACKed
  - can't send until window moves

- **Receiver**: 1 2 3 4 5 6 7 8 9 10 11 12
  - ACK'd and read
  - ACK'd, not read
  - can't recv until window moves

- **advertised window**: 6 7 8 9
- **offered window**: 4 5 6 7 8 9
Visualizing the Window: Example

Receiver to Sender ➔ ACK 9, Window 0

advertised window = 0

Sender

1 2 3 4 5 6 7 8 9 10 11 12

sent and acknowledged

can't send until window moves

Receiver

1 2 3 4 5 6 7 8 9 10 11 12

ACK'd and read

ACK'd, not read

can't recv until window moves

offered window = 0
Visualizing the Window: Example

Receiver App reads packets 4, 5, 6
But sender has no way of knowing that more room is available!

advertised window=0

Sender
1 2 3 4 5 6 7 8 9 10 11 12
sent and acknowledged

Receiver
1 2 3 4 5 6
ACK'd and read

7 8 9
ACK'd, not read
10 11 12
Available bufs

offered window=3

Options for Sender Discovery of Increased Advertised Window

• Receiver sends duplicate ACK with a larger advertised window
  • Complicates receiver design
  • TCP design philosophy: keep receiver simple

• Sender periodically transmits a 1-byte packet
  • If no space available at receiver ➞ packet dropped, no ACK
  • If additional space became available ➞ ACK contains new advertised window

• NOTE: Advertised window expressed in bytes, not packets!