ECE 435 – Network Engineering Lecture 15

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

26 October 2016

Announcements

- HW#5 due
- HW#6 posted
- Broadcasts on the MBONE

The Transport Layer

- Responsible for the end-points of a channel
- Provide process-to-process connectivity, and per-segment error control and per-flow reliability, as well as rate control
- Can be more reliable than underlying network
- TCP (Transmission Protocol Layer)
  - connection oriented
  - stateful
  - per-flow reliability and rate control

- UDP (User Datagram Protocol)
  - stateless
  - connectionless
- the “socket” is the API from old homework

The Transport Layer

- application = process, data-transfer-unit is a segment, traffic is a flow
- addressing – each process needs a unique ID. For internet, this is the "port" number (16-bit)
- Rate control
  - Flow control – between source and destination
  - Congestion control – between source and network
    None in link layer because only one hop?

Can be done by sender or network

- Real time requirements – things like video and audio need extra info such as timestamp, loss rate, etc. So hard to do with raw TCP/UDP

Unreliable, Connectionless – UDP

- User Datagram Protocol
- No reliability, no rate control, stateless
- Error control optional
- Provides process-to-process communication and per-segment error control
- Packet header:

- 16-bits: source port
- 16-bits: destination port
- 16-bits UDP length
- 16-bits checksum (optional)
- data

- Can send UDP packets to a destination without having to set up a connection first

Port Numbers

- 16-bit, so 64k of them
- Can map to any you want, but there are certain well-known ones. Look in `/etc/services`. For example. WWW is 80
- On most operating systems, ports below 1024 require root (why?)
- Source/destination addr + source/destination port + protocol ID (TCP or UDP) is a socket pair (or 5-tuple)

is 104 bits that uniquely identify a flow for IPv4. IPv6 has a specific field for this

UDP checksum

- If set to zero, ignored
- Receiver drops invalid checksums (does not request resend)
- 1s complement of sum all 16-bit words in header and payload
  padded with 0s to be multiple of 16-bits
- Also added to the checksum is a 96-bit pseudo header that has source IP, dest IP, protocol, length. Enables receiver to catch problems with there to (delivered to

wrong machine)

- What happens if checksum is 0? entered as 0xffff
- Checksum considered mandatory on IPv6 because header not checksummed
- Why would you ever leave checksum out? Takes time to compute, might care about latency over errors [video?]
- Example:

```
0x0000:  8875 563d 2a80 0030 18ab 1c39 86dd 6002  .uV=*..0...9..`.
0x0010:  2618 0031 1140 2610 0048 0100 08da 0230  &..1.@&..H.....0
0x0020:  18ff feab 1c39 2001 4860 4860 0000 0000  .....9..H`H`....
0x0030:  0000 0000 8844 e239 0035 0031 9c0e 8657  .....D.9.5.1...W
0x0040:  0120 0001 0000 0000 0001 0377 7777 0465  ...........www.e
0x0050:  7370 6e03 636f 6d00 0001 0001 0000 2910  spn.com.......).
0x0060:  0000 0000 0000 00
```

- 16-bit sum of “virtual header” (two IPv6 addresses, protocol (0x0011) and length of udp packet/header (0x0031)) is 0x29f8c
- 16-bit sum of UDP header leaving off checksum is 0xe29f
- 16-bit sum of UDP data is 0x2e1c0
- Add them get 0x6 63eb
- It’s a 16-bit sum, so add 0x6 + 0x63eb = 0x63f1 ones complement is 0x9c0e, which matches the UDP checksum field

UDP real-time

- Real-Time Protocol (RFC1889)
- On top of UDP, multiplexes
- data streams
- timestamps

TCP

- Transmission Control Protocol
- RFC 793 / 1122 / 1323
- Reliable, in-order delivery.
- Adapts to network congestion
- Takes data stream, breaks into pieces smaller than 64k (usually 1460 to fit in Ethernet) and sends as IP

- No guarantees all packets will get there, so need to retransmit if needed.
- Multiple connections can share same port (i.e. webserver on port 80 can handle multiple simultaneous requests)
- Point-to-point (can't multicast)
- Full duplex
- Byte stream, if program does 4 1024byte writes there's no guarantee how that will be split up and the other end doesn't see.

- PUSH flag can be sent that says not to buffer (For example, if interactive command line)

- URGENT flag can be sent that says to transmit everything and send a signal on the other side that things are urgent.

TCP Header Format

- 16-bit source port
- 16-bit dest port
- 32-bit sequence number
- 32-bit ack number next byte expected, not last one received
- 4-bit header length number of 32-bit chunks (includes header)
- 6-bit reserved (not used)
- 6 bits of flags

- U (URGent) – also the urgent pointer puts to urgent byte
- ACK – 1 if ack field valid, otherwise ack field ignored
- PSH – receiver should process the data immediately and not buffer it waiting for more to come in
- RST (reset) – reset a connection because something has gone wrong
- SYN – used to establish connection CONNECTION REQUEST (SYN=1,ACK=0) and CONNECTION ACCEPTED (SYN=1,ACK=1)
- FIN – used to release a connection

- 16-bit window size – Only in ACK, says how many bytes to send back. This can be 0, which means I received everything but I am busy and can't take any more right now (can send another ACK with same number and nonzero window to restart)
- 16-bit checksum – similar to UDP also with pseudo header
- 16-bit urgent pointer
- options (32-bit words)
  - End of option – end of all options
  - No operation – for padding

- MSS maximum segment size (only in initial SYN packet)
- Fast connections sequence can wrap quickly.
- RFC1323 –PAWS, window scaling factor, specify larger transfer size as on long-latency high-bandwidth connections can end up idle a lot waiting for ACK
- RFC1106 allows selective resend – if lost packet in long stream, instead of sending all, just resend missing

• data

TCP Connection Management

- Like UDP, 5-tuple
- How to handle delayed or retransmitted packets?
- Maximum 120s delay
- Three-way handshake (Tomlinson 1975)
  - Choose random initial sequence number (ISN)
  - Send SYN(SEQ=X) with port and sequence number

- Server sends back ACK(X+1) plus SYN(Y) with sequence of own
- Client then ACK(Y+1) the server SYN,

- SYN number picked, not to be 0. Originally clock based (random these days?). If machine reboots should wait for maximum lifetime to make sure all close

- Closing connection
  - client sends FIN
  - server sends ACK of FIN
  - server sends FIN

- client sends ACK of FIN
- If only one side sends FIN, other can still keep sending data indefinitely
- Two army problem? If FIN not ACKed within two packet lifetimes, will close anyway. The other side eventually notices and closes too.