ECE 435 – Network Engineering
Lecture 16

Vince Weaver

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

vincent.weaver@maine.edu

31 October 2016

Announcements

- HW#6 was posted, due Wed
- I'll be at Linux Plumbers on Wednesday, so watch your e-mail for the plan.

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 simultaeneous 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) ECN bits
- 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 Opening Connection

- 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 sequece 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

TCP Closing Connection

- Closing connection
- Almost like two independent one-way connections, released independently
  - one side sends packet with FIN
  - other side sends ACK of FIN, that direction is shut down
  - other direction can keep sendin data though
  - at some point other side sends FIN
  - this is ACKed

– Two army problem? If FIN not ACKed within two packet lifetimes, will close anyway. The other side eventually notices and closes too.

TCP State Machine

- 11 possible states
  - starts in CLOSED
  - LISTEN – waiting for a connection
  - SYN-SENT – started open, waiting for a returning SYN
  - SYN-RECEIVED – waiting for ACK
  - ESTABLISHED – open, two-way communication can happen
  - FIN-WAIT-1 – application has said it's finished

- FIN-WAIT-2 – the other side agreed to release
- CLOSE-WAIT – waiting for a termination request
- CLOSING – waiting for an ACK of closing request both sides closed at once
- LAST-ACK – waiting for ACK from last closing
- TIME-WAIT – waiting to transistion to CLOSED long enough to ensure other side gets last ACK

- large state diagram

Typical Connection seen from Client

- CLOSED, CONNECT/sends SYN (step 1 of handshake)
- SYN-SENT, gets SYN+ACK/sends ACK (step 3 of handshake)
- ESTABLISHED, whent time to CLOSE send FIN
- FIN-WAIT-1, received ACK
- FIN-WAIT-2, one side not closed, wait for FIN/send ACK
- TIME-WAIT (wait for timeout to ensure all packets done in case ACK got lost)

- CLOSED

Typical Connection seen from Server

- CLOSED, wants to LISTEN
- LISTEN, gets SYN, sends SYN+ACK (step 2 of handshake)
- SYN-RECVD, gets ACK
- ESTABLISHED, FIN comes in from client, sends ACK
- CLOSE-WAIT, closes itself, sends FIN
- LAST-ACK, gets ACK
- CLOSED

TCP Reliability

- Per-segment error control
  - checksum, Same as UDP.
  - also covers some fields in IP header to make sure at right place
  - TCP checksum is mandatory
  - Checksum is fairly weak compared to crc32 in ethernet
- Per-flow reliability
  - What to do in face of lost packets? Need to notice

and retransmit and handle out-of-order

- Sequence number generated for first blob (octet?), 32-bit number in header
- Sender tracks sequence of what has been sent, waiting for ACK
- On getting segment, receiver replies with ACK with number indicating the expected next sequence number, and how much has been received. "All data preceeding X has been received, next expected sequence number is Y. Send more"
- Selective ACK – has received segment indicated by

ACK

– Cumulative ACK – all previous data previous to the ACK has been received

Window Management / Flow Control

- Sliding window
- example
  - Receiver has 4k buffer
  - Sender does 2k write (2k/SEQ=0)
  - Receiver sends back ACK=2k, WIN=2048 (can take up to 2k)
  - Application sends 2k (2k, SEQ=2k)
  - If it is full, receiver might send ACK=4k, WIN=0
  - Later once buffer clears up a bit (application reads 2k

maybe) sends ACK=4096, WIN=2k
  - Sender then sends some more
- When waiting on a WIN=0 can send two things, URG to kill the connecction, or a 1-byte packet to have retransmit window and next byte expected (in case the ack restarting was lost, otherwise deadlock)
- Senders can buffer data, for example if know window is 4k can wait until they have 4k. Can help performance.
- For example, typing at keyboard on telnet/ssh might send when an editor. Press key, send a packet. Get ACK. Then when read, another ACK updating window

size. Then finally draw char on screen, send packet with that. 4 packets for one keypress

- One way to help is avoid window updates for up to 500ms in hope they can tag along with a real outgoing packet
- Nagel's algorithm – when data coming in one byte at time, send first then buffer rest until the first byte acknowledged. Also take into account window size. Widely used, can be bad for things like X window forwarding as mouse movements bunched together. TCP_NODELAY option disables.

- Silly window syndrome – application reading out the bytes one at a time. Send window updates each time, other side resend one byte, send message window full, etc. Solution (Clark) to wait until buffer is the original max segment size, or half empty

TCP Congestion Control

- Fast network feeding small receiver (flow control?)
- Slow network feeding big receiver (problem on sending side, lose packets, congestion control)
- Two windows, receiver window looked at previously, and congestion window (kept locally)
- Amount can send is the minimum of the two windows
- Setup
  - Congestion window set to max segment size
  - Send one max segment, arm timer

- ◦ If ACK received before timer goes off, good. Double the size.
- ◦ Repeat, exponetial growth. Called ”slow start”
- ◦ “internet (?)” has a limit where it stops exponential and moves to linear growth
- ◦ eventually hit receiver window size and stop

- Changed over the years.
- Initial implementation no congestion control, not needed (not that much traffic)
- After 8 years (1980s) introduced by Van Jaconson – internet facing “congestion collapse” – would send as fast

as possible, packets would be dropped, hosts retransmit, even more congestion

- TCP Tahoe (v2) (BSD 4.2 1988) added congestion avoidance, fast retransmit (Van Jacobsen)
  Slow start – probing bandwidth with few roungs.
  cwnd set to 1 and exponentially increases with each ACK until hits ssthreah
  congestion avoidance – slow probing but rapid respond to congestion
  AIMD additive increase multiple decrease.
  Fast retranmit– transmitting lost packets immediately,

no wait for timer. If get three duplicate ACKS in a row, assume packet loss, resend. Drop sshthreah to half and start slow-start again
retransmission timeout – halve sshthreash and restart slow-start

- TCP Reno (v3) added fast recovery
  set sshthread to cwnd+3 because of triple ack
- TCP New Reno

When to retransmit

- If packet lost, then will receive duplicate ACK on next transmission
- If get three identical ACKs, probably means packet lost, resend
- Why not two? Because if packets arrive out of order can also cause duplicate ACK

TCP Timer Management

- What should the timer value be? Too short, send extra packets, too long and takes long time to notice lost packets.
- On the fly measures round trip time. When send segment, start timer, updates.
- Connection Timer – send SYN. If no response in time, reset
- Retransmission Timer – retransmit data if no ACK

- Delayed ACK timer – if send a packet, tag an ACK along if timer expires and no outgoing data, have to send stanadlone ACK
- Persist Timer – solve deadlock where window was 0, so waiting, and missed the update that said window was open again
- Keepalive Timer – if connection idle for a long time, sends probe to make sure still up
- FIN_WAIT_2 Timer – avoid waiting in this state forever if other side cashes

- TIME_WAIT_TIMER – used in TIME_WAIT to give other side time to finish before CLOSE

When Things Go Wrong

- Data loss – after retransmit timeout, will notce and retransmit
  If packets just taking a long time, could end up always retransmitting. Data gets in but huge waste of bandwidth. see later.
- ACK loss
- out-of-sequence
- ECN explicit congestion notification – uses extra bits in

reserved flags, routers initially had trouble with this.

Security Issues

- SYN Flood attack – Denial of service – spoof IP address, send lots of spurious SYNs followed by ACK, tie up lots of resources
  Spoof, because responds to wrong address which just ignores. Causes lots of half-open connections
  One solution is SYN cookies – (pick special sequence that allow throwing out connection info but able to reconstruct if an ACK comes back).

- Connection hijacking – guess a proper sequence number and forge a packet that looks like it should be next. If you can also take down the real IP (DOS?) can take over the connection Helps to have good random sequence numbers

- TCP veto – inject packet with sequence and payload of next expected. That way when the real actual next one comes in, it will be silently dropped as a duplicate

- NMAP port scanning – send packets and find if connections are open, determine host system. Christmas Tree packets

Find out host? options supported, sequence numbers. Also can find out uptime of system as timestamps from extension usually aren't reset each time.

- Martian packets – packets with a source of a reserved network found on routed internet

Making things faster

- Offload engines

Proposed Replacements

- T/TCP
- SCTP – stream control transmission protocol. More complex
- Whatever google is up to