ECE 435 – Network Engineering Lecture 9

Vince Weaver

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

vincent.weaver@maine.edu

25 February 2021

Announcements

- HW#4 due Friday
- HW#5 will be posted

HW#3 Review

- md5sum/encryption, seems to have gone well
- How to validate PGP key is indeed for who it says?
  Certificate Authority (costs money)
  Distributed Web of Trust (key signing party).
  Compare in person/phone, key fingerprint if not want to send whole thing.
- e-mail
  - First warning sign – says its from a bank, but the return address is from a Florida dental school

Also not a bank of mine

- encrypted and verified from UFL, but sent from videotron.ca cablemodem
- Virus scanned and SPAM scanned, just sort of barely passed
- pop from deater.net via fetchmail (this isn't suspicious, it's the sender not receiver you have to look at)
- LMTP – local mail transport. LHLO. No mail queue, says right away whether deliver mail is possible.
- pdf attached probably had some sort of exploit or phishing document. Didn't open.

  - ○ Note, the attachment being listed as “Application” does not mean it’s an executable
- For the headers, was looking for MIME as what’s going on. Also base64

TCP

- Transmission Control Protocol
- RFC 793 (from 1981) / 1122 / 1323 2018 / 2581 / 2873 / 2988 / 3105, summary in 4614
- 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 the other end sees 4 chunks of 1024, only 4k stream of bytes is guaranteed.

- 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

Fixed 20-byte header. From RFC793:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |       |C|E|U|A|P|R|S|F|                               |
| Offset|Reservd|W|C|R|C|S|S|Y|I|            Window             |
|       |       |R|E|G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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)
- 4-bit reserved (not used) ECN bits
- 8 bits of flags

- CWR / ECE – for ECN congestion
- U (URGent) – urgent pointer points to urgent byte
- ACK (acknowledge) – 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 (synchronize) – 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) – we'll discuss these later
- data

TCP Header – Options

- One-byte
  - End of option – end of all options. Only one allowed (not always needed?)
  - No operation – for padding
- Multi-byte
  - MSS maximum segment size (only in initial SYN packet)

    Byte1: 2, Byte 2: len=4, max size=2 bytes
  - RFC1323 –PAWS, Protection against Wrapped

Sequence-number

High bandwidth, seq num can wrap. Use timestamps to recognize when this happens. window scaling factor, specify larger transfer size as on long-latency high-bandwidth connections can end up idle a lot waiting for ACK

Scales the window size by value

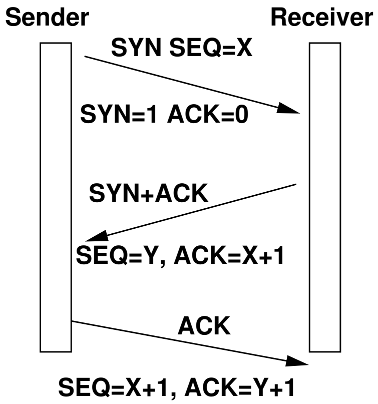
Byte 1: 3, Byte 2: Len=3, Scale factor=1 byte

- Timestamp: used to calculate round-trip time. Leaks info? Send along timestamp when you send, other side keeps that and returns it (as echo) with the relevant

ACK.
10 bytes long Byte 1: 8, Byte 2: Len=10, Timestamp=4 bytes, Echo=4 bytes

- RFC1106 allows selective resend – if lost packet in long stream, instead of sending all, just resend missing
- Fast connections sequence can wrap quickly (orig internet 56k, modern 1Gb connectios wrap in seconds rather than weeks)

TCP Opening Connection



- Three-way handshake (Tomlinson 1975)
  - Server does LISTEN/ACCEPT to wait for connection.
  - Client issues CONNECT: destination/port/size, etc.
  - CONNECT chooses random initial sequence number (ISN) X

Sends SYN(SEQ=X) (SYN=1 ACK=0) with port and sequence number

- Server receives packet. Checks if listening on that port; if not send back a packet with RST to reject.
- Otherwise it can accept

  sends back ACK(X+1) plus SYN(SEQ=Y) with sequence of own
- Client then responds with the server SYN ACK(Y+1) SEQ=x+1
- Connection is established

- 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

- Why do this? What happens with simultaneous connection?

TCP Closing Connection

- Closing connection
- Although full duplex, 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 sending data though
  - at some point other side sends FIN
  - this is ACKed

- Two army problem?
  Two generals on opposite side trying to co-ordinate attack. Any message can be intercepted by enemy. So say “attack at 9pm” but that could be lost. Could require other side to send reply, but that could be lost. You need infinite messages to guarantee it got through.
  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 transition to CLOSED long enough to ensure other side gets last ACK
- large state diagram

Typical Connection seen from Client

- CLOSED

  user does connect(), SYN sent (step 1 of handshake)
- SYN-SENT

  waits for SYN+ACK, sends ACK (step 3 of handshake)
- ESTABLISHED

  sends/receives packets

  eventually user will close() and send FIN
- FIN-WAIT-1

  FIN sent, waiting for ACK

- FIN-WAIT-2

  one direction closed

  received ACK of FIN, wait for FIN from other side, respond with ACK
- TIME-WAIT

  wait until timeout to ensure all packets done in case ACK got lost
- CLOSED

Typical Connection seen from Server

- CLOSED
  waits for listen()
- LISTEN
  gets SYN, sends SYN+ACK (step 2 of handshake)
- SYN-RECVD
  waits for ACK
- ESTABLISHED
  sends/receives
  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 preceding X has been received, next expected sequence number is Y. Send more”
- Selective ACK – has received segment indicated by

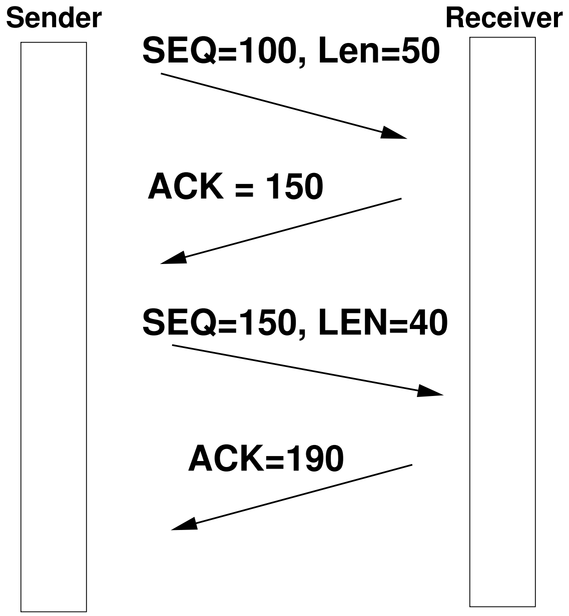
ACK

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

Error Correction

- Ways to Catch Errors
  - Checksum
  - Acknowledgement
  - Time-out

Comparison: Good Transaction

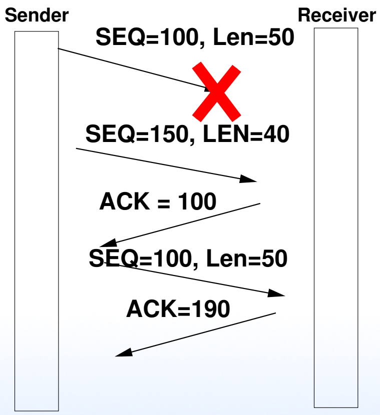


Error: Corrupted or Lost Packet

- SEQ 1401, Len200 bytes, SEQ 1601+200, SEQ 1801+200

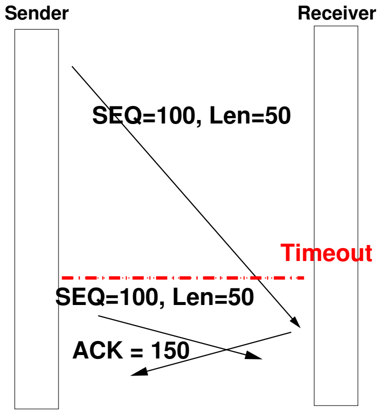
  Last one corrupted receiver only acks through ACK=1601

  Eventually timeout, and sender will retransmit



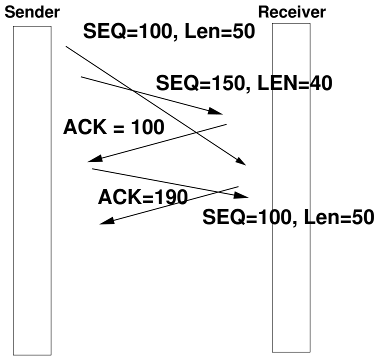
Error: Delay or Duplicate Packet

- Duplicate packet (how can happen? a timeout happens and is resent just before ACK gets in)
  TCP discards packets with duplicate SEQ



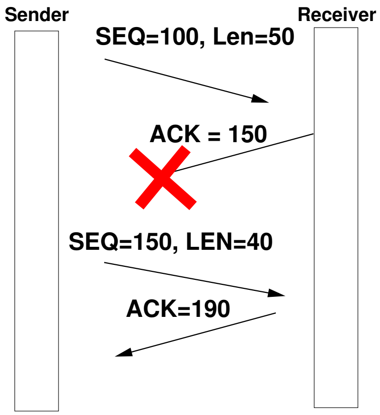
Error: Out-of-order Packet

- Out-of-order packet
  Do not ACK packet until preceding ones make it.
  For performance can queue up out of order ones so they don't have to be resent



Error: Lost ACK

- ACKs cumulative, so if the next packet causes an ACK then it doesn't matter. Otherwise a timeout?



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. (RTT) When send segment, start timer, updates. Various algorithms. Often 2 or 4x

- 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.
  Sends special probe packet. Keep trying every 60s?
- 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 crashes

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