ECE 435 – Network Engineering
Lecture 10

Vince Weaver

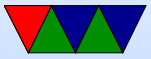
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

17 February 2022

Announcements

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

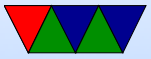


HW#5 Notes

- Decoding a hexdump

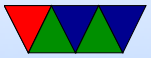
```
hexdump -C ece435_lec08.pdf
00000000  25 50 44 46 2d 31 2e 35  0a 25 d0 d4 c5 d8 0a 39  |%PDF-1.5.%.....9|
00000010  20 30 20 6f 62 6a 0a 3c  3c 0a 2f 4c 65 6e 67 74  | 0 obj.<<./Lengt|
00000020  68 20 33 37 33 20 20 20  20 20 20 20 0a 2f 46 69  |h 373       ./Fi|
00000030  6c 74 65 72 20 2f 46 6c  61 74 65 44 65 63 6f 64  |lter /FlateDecod|
00000040  65 0a 3e 3e 0a 73 74 72  65 61 6d 0a 78 da 9d 52  |e.>>.stream.x..R|
```

- First column is offset into the file or packet (usually in hex).
- The next set of columns are the raw bytes, in hex.
- The last column is the ASCII char equivalent of the raw data. a '.' often indicates non-printable ASCII.

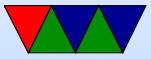


HW#3 Review

- md5sum/encryption, seems to have gone well
- How to validate PGP key is indeed for who it says?
  - https isn't enough, what if the person who admins the webserver is evil?
  - Certificate Authority (costs money)
  - Distributed Web of Trust (key signing party).
  - Compare in person/phone, key fingerprint if not want to send whole thing.
- Encrypted message went fine



- Why not use SHA-1 for git anymore? It's been "broken" which means possible to generate a collision
- Can you use virtual hosting with https? Problem is host header isn't received until after the SSL conection set up.

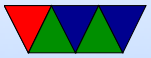


Transmission Control Protocol (TCP)

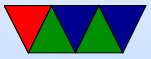
- RFC 793 (from 1981) / 1122 / 1323

  2018 / 2581 / 2873 / 2988 / 3105, summary in 4614

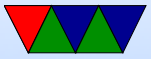
- Generally attributed to Vint Cerf and Bob Kahn
- Reliable, in-order delivery.
- Adapts to network congestion
- Takes data stream, breaks into pieces smaller than 64k (usually 1460 to fit in Ethernet) and sends as IP



- 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.

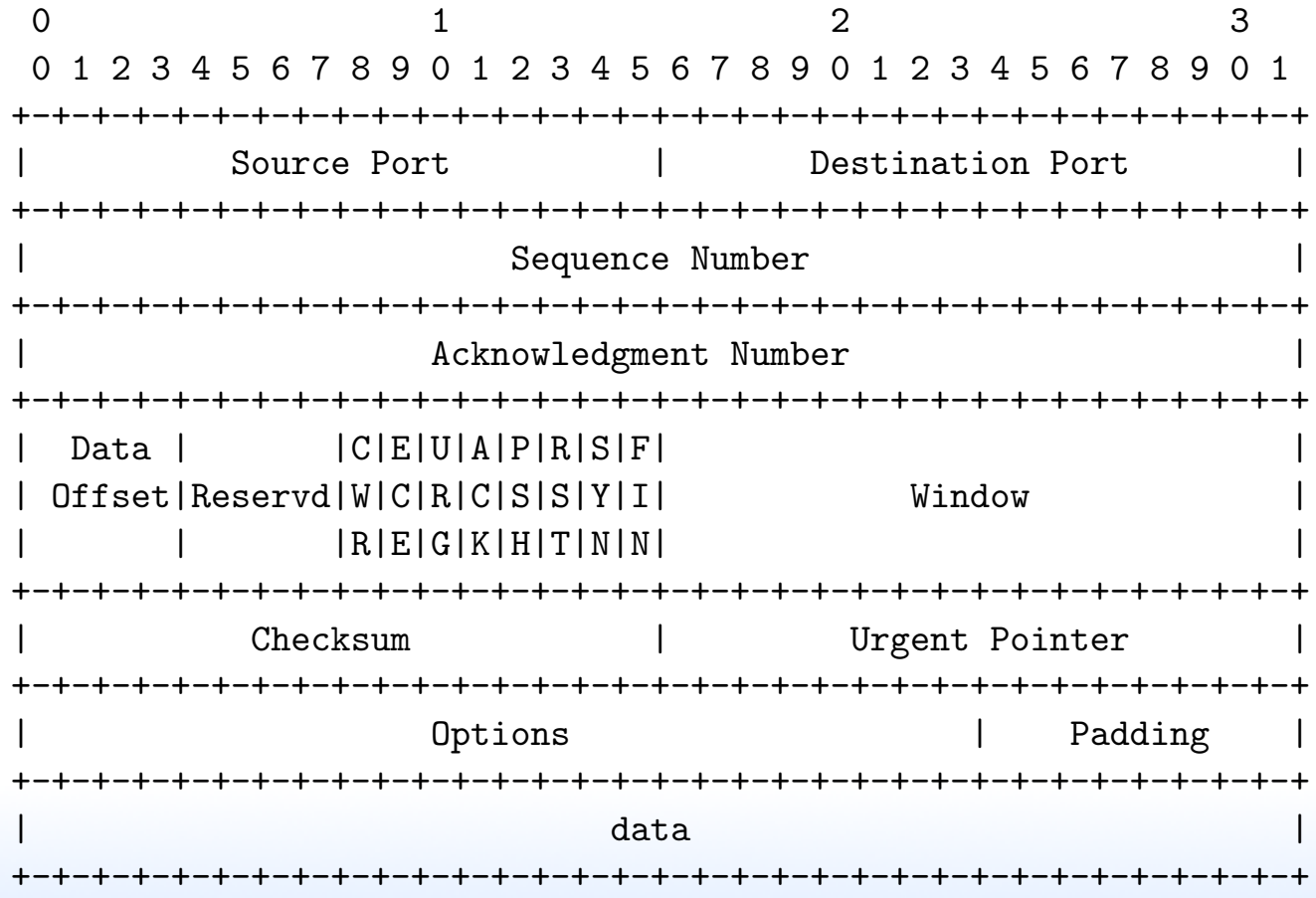


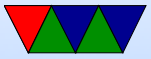
- PUSH flag can be sent that says not to buffer (For example, if interactive command line)
- URGENT flag can be sent that says to transmit everything and send a signal on the other side that things are urgent.



TCP Header

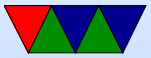
Fixed 20-byte header. From RFC793:



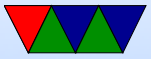


TCP Header Format

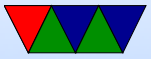
- 16-bit source port
- 16-bit dest port
- 32-bit sequence number
- 32-bit ack number
  next byte expected, not last one received
- 4-bit data offset (mul by 4) (min 5 (20), max 15 (60))
  header length/points to start of data
- 3-bit reserved zero (not used)
- 9 bits of flags



- NS / 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)

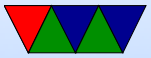


  - FIN – used to release a connection
- 16-bit window size – Only in ACK, says how many bytes to send back. This can be 0, which means I received everything but I am busy and can't take any more right now (can send another ACK with same number and nonzero window to restart)
- 16-bit checksum – similar to UDP also with pseudo header
- 16-bit urgent pointer
- options (32-bit words) – we'll discuss these later
- data



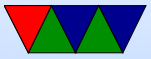
TCP Header – Options

- **type=0** End of option
  End of all options. Only one allowed (not always needed?)
- **type=1** No operation (for padding to 32-bit boundary)
- **type=2, Len=4, Value=16-bits** Maximum Segment Size
  only in initial SYN packet
- **type=3, Len=3, Value=8-bits** Window size
  Scaling factor to shift window size by (0..14), raising



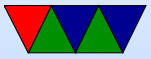
limit to 1GB. Only set during handshake

- **type=4, len=2** Selective ACK permitted
- **type=5, len=?** Selective ACK
  list of 1-4 blocks being selectively acknowledged, as 32-bit begin/end pointers
  allows only resending missing packets instead of having to restart at last ACK (RFC1106?)
- **type=8, len=10** Timestamp and echo of last timestamp
  Not necessarily current time. (RFC1323) PAWS, Protection against Wrapped Sequence-number
  High bandwidth, seq num can wrap. Use timestamps to

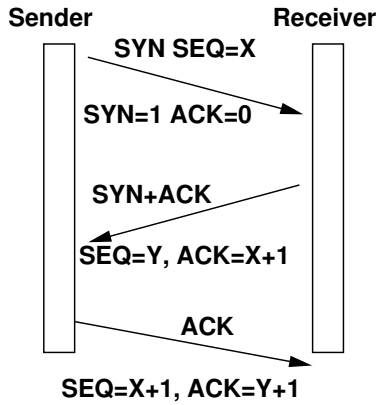


recognize when this happens.

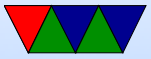
Fast connections sequence can wrap quickly (orig internet 56k, modern 1Gb connectios wrap in seconds rather than weeks)



TCP Opening Connection



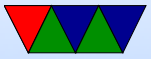
- Three-way handshake (Tomlinson 1975)
  - Server does LISTEN/ACCEPT to wait for connection.
  - Client issues CONNECT: destination/port/size, etc.
  - CONNECT chooses random initial sequence number (ISN) X



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

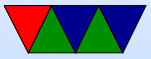
- Server receives packet. Checks if listening on that port; if not send back a packet with RST to reject.
- Otherwise it can accept
  sends back ACK(X+1) plus SYN(SEQ=Y) with sequence of own
- Client then responds with the server SYN ACK(Y+1) SEQ=x+1
- Connection is established

- SYN number picked, not to be 0. Originally clock based



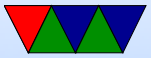
(random these days?). If machine reboots should wait for maximum lifetime to make sure all close

- Why do this? What happens with simultaneous connection?



TCP Closing Connection

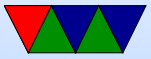
- Closing connection
- Although full duplex, almost like two independent one-way connections, released independently
  - one side sends packet with FIN
  - other side sends ACK of FIN, that direction is shut down
  - other direction can keep sending data though
  - at some point other side sends FIN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



– Two army problem?

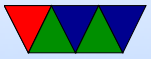
  Two generals on opposite side trying to co-ordinate attack. Any message can be intercepted by enemy. So say "attack at 9pm" but that could be lost. Could require other side to send reply, but that could be lost. You need infinite messages to guarantee it got through.

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



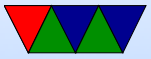
TCP State Machine

- 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



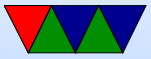
- FIN-WAIT-2 – the other side agreed to release
- CLOSE-WAIT – waiting for a termination request
- CLOSING – waiting for an ACK of closing request both sides closed at once
- LAST-ACK – waiting for ACK from last closing
- TIME-WAIT – waiting to transition to CLOSED long enough to ensure other side gets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

- large state diagram

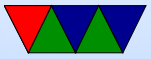


Typical Connection seen by Client

- 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

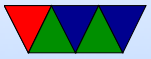


- FIN-WAIT-2
  one direction closed
  received ACK of FIN, wait for FIN from other side, respond with ACK
- TIME-WAIT
  wait until timeout to ensure all packets done in case ACK got lost
- CLOSED

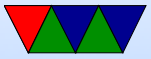


Typical Connection seen by Server

- 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

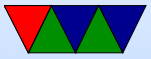


- CLOSE-WAIT
  , closes itself, sends FIN
- LAST-ACK
  gets ACK
- CLOSED



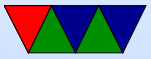
TCP Reliability

- 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



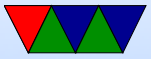
and retransmit and handle out-of-order

- 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



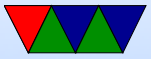
ACK

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

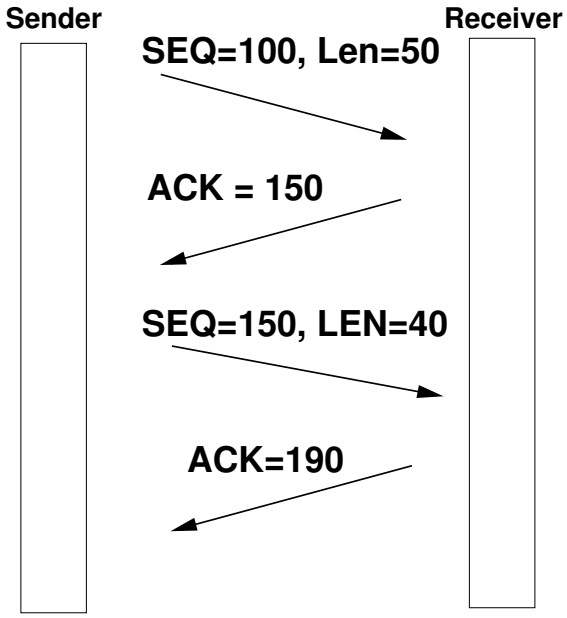


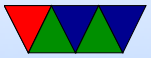
Error Correction

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



Comparison: Good Transaction

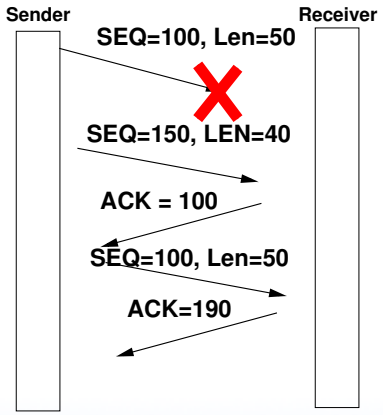


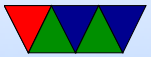


Error: Corrupted or Lost Packet

- SEQ=100 Len=50 bytes, SEQ=150 LEN=50, SEQ=200 LEN=50

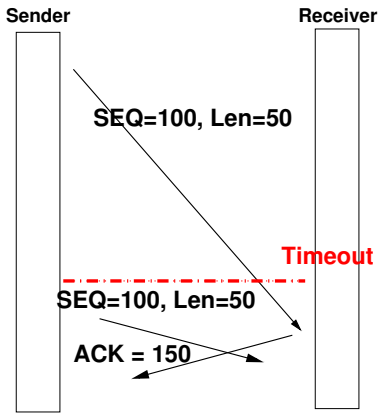
  First one never made it, receiver only acks through ACK=100 After three duplicate ACKs, sender retransmit

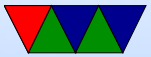




Error: Delay or Duplicate Packet

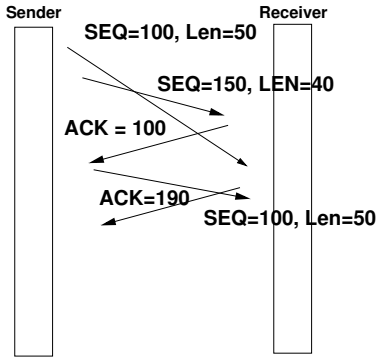
- Duplicate packet (how can happen? a timeout happens and is resent just before ACK gets in)
  TCP discards packets with duplicate SEQ

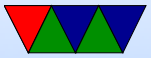




Error: Out-of-order Packet

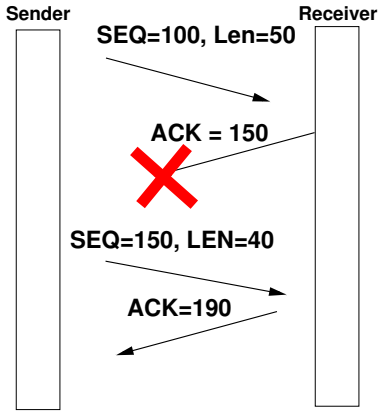
- Out-of-order packet
  Do not ACK packet until preceding ones make it.
  For performance can queue up out of order ones so they don't have to be resent

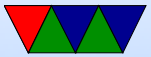




Error: Lost ACK

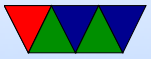
- ACKs cumulative, so if the next packet causes an ACK then it doesn't matter. Otherwise a timeout?



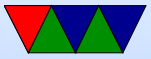


TCP Timer Management

- 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