CS519: Computer Networks

Lecture 5, Part 2: Mar 8, 2004

Transport: TCP mechanics

(RFCs: 793, 1122, 1323, 2018, 2581)

TCP as seen from above the socket

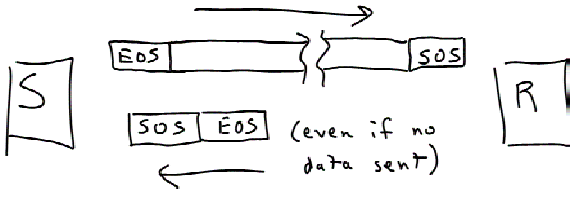
CS519

- The TCP socket interface consists of:
  - Commands to start the connection
    - `connect(), listen(), accept()`
  - Commands to send to and receive from the connection
    - `read(), write()`
  - Commands to end the connection
    - `close(),`
    - (but also `read(), write()`!)

TCP as a "marked" stream

CS519

- Think of TCP as having "start-of-stream" and an "end-of-stream" tags
  - EOS means "no more data will be sent"
  - And, "you got all the data that was sent"



TCP as a "marked" stream

CS519

- `connect(), listen(),` generate the "SOS" (TCP SYN)
- `accept()` signals reception of the "SOS"
- `close()` generates the "EOS" (TCP FIN)
- `read()==0` signals reception of the "EOS"
- A connection can end without an "EOS"
  - `read()==-1` or `write()==-1`
  - In this case, some sent bytes may not have been received

TCP “keepalive”

CS519

- A TCP connection can stay “up” forever even without sending any packets
  - Indeed, if one end crashes silently, the other end won’t notice until it sends a packet
  - Sometime called a half-open connection
- TCP implementations have a “keepalive” option
  - Settable through `sockopts()`

TCP “keepalive”

CS519

- Periodically sends a TCP packet with no data
- The other end responds with an ACK if it is alive
- If not, the other end is declared down, and a pending `read()` is returned with -1
- This is not part of the TCP spec per se
- This can just as well be done at the application layer

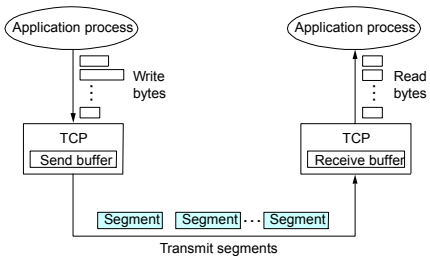
Some TCP issues we’ll look at

CS519

- TCP uses a sliding window as we saw with link protocols
- However, TCP must content with different issues
  - Round trip may vary
    - (so don’t know how best to fill the pipe)
  - The network may be congested
    - (so may need to go even slower than receive window allows)
  - Packets may not arrive in order
  - A TCP connection has to synchronize the beginning and end (SOS and EOS)

TCP bytes and “segments”

CS519



TCP Header (segment)

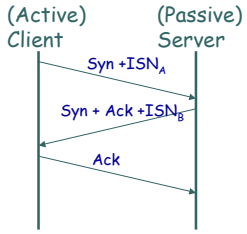
CS519

Flags: SYN
FIN
RESET
PUSH
URG
ACK

Source port	Destination port
Sequence number	
Acknowledgement	
Hdr len | 0 | Flags	Advertised window
Checksum	Urgent pointer
Options (variable)	
Data	

Connection Establishment (with Initial Sequence Numbers)

CS519



Connection Setup
3-way handshake

Connection terminate

CS519

- Connection establish is fairly straightforward
- Connection terminate is more complex
  - Because both sides must fully close
  - One side can close while the other still sends the last of its data
  - Or both can close at once

TCP Connection terminate

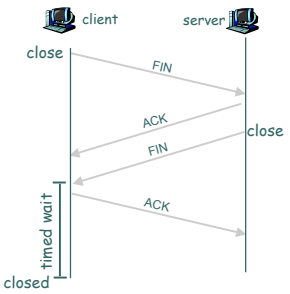
CS519

Closing a connection:

client closes socket:
`clientSocket.close();`

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



TCP Connection Terminate

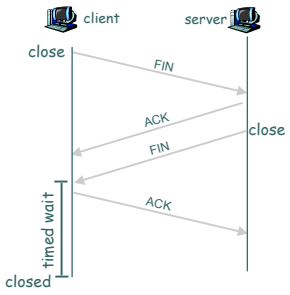
CS519

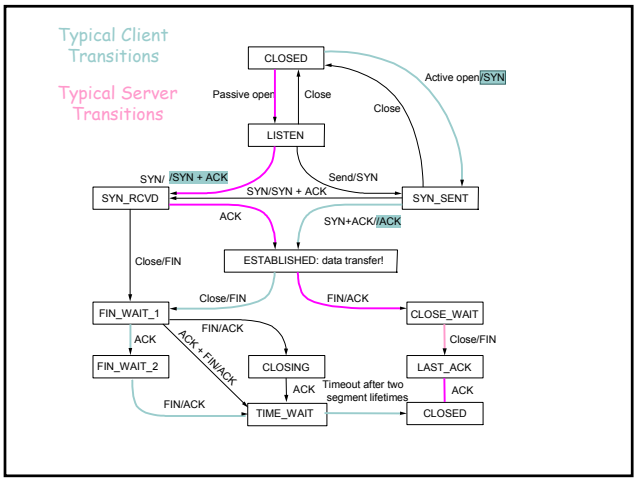
Step 3: client receives FIN, replies with ACK.

- Enters "timed wait" - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

Note: with small modification, can handle simultaneous FINs.





TIME_WAIT state

CS519

- On client, Wait 2 times Maximum Segment Lifetime (2 MSL)
  - Provides protection against delayed segments from an earlier incarnation of a connection being interpreted as for a new connection
- Maximum time segment can exist in the network before being discarded
  - Time-To-Live field in IP is expressed in terms of hops not time
  - TCP estimates it as 2 minutes

TIME_WAIT state

CS519

- During this time, combination of client IP and port, server IP and port cannot be reused
  - Some implementations say local port cannot be reused at all while it is involved in time wait state even to establish a connection to different dest IP/port combo

TCP advertised window (the receive window)

CS519

- In TCP, the receiver sends its receive window to the sender
  - Note that both ends are both sender and receiver
  - The receiver sends its receive window with every ACK
- The sender sets its send window to that of the receive window
- Therefore, we only really speak of one window, the send window

TCP advertised window

CS519

- Why does the TCP receiver need to convey its receive window, whereas in the link-layer sliding window, we didn't need that?

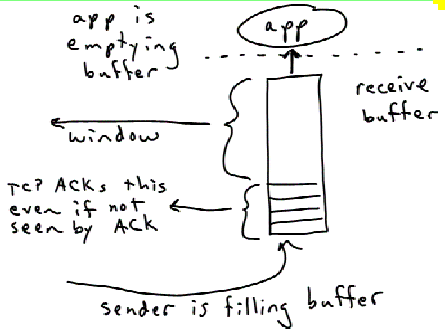
TCP advertised window

CS519

- Why does the TCP receiver need to convey its receive window, whereas in the link-layer sliding window, we didn't need that?
- ANS: because the TCP layer can ACK data even though the application hasn't read it

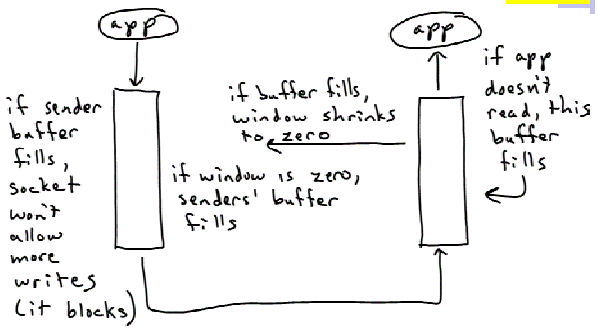
TCP advertised window

CS519



TCP flow control

CS519



TCP retransmission mechanism originally Go-back-N

CS519

- Say sender sends bytes 1000 – 1499, in 5 100-byte packets
- Receiver ACKs up to 1100
- Sender knows that receiver missed packet 1100-1199, but doesn't know about other three packets
- Sender "goes back" to 1100, and starts retransmitting everything
- It may therefore resend received packets
  - Lots of them, if the pipe is long and fat

Later TCP added Selective Acknowledgement (SACK)

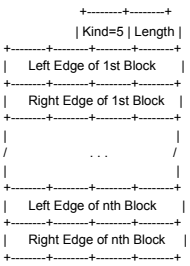
CS519

- Use TCP option space during ESTABLISHED state to send "hints" about data received ahead of acknowledged data
- TCP option that says SACK enabled on SYN => "I am a SACK enabled sender, receiver feel free to send selective ack info"
- Normal ACK field still authoritative!
- SACK usage is growing, but still not universal

SACK Details

CS519

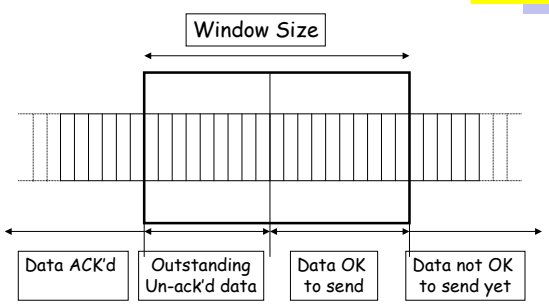
- Format:



- TCP option 5
- In 40 bytes of option can specify a max of 4 blocks
- If used with other options space reduced
- Ex. With Timestamp option (10 bytes), max 3 blocks

TCP sliding window

CS519



Big Fat Pipes and TCP

CS519

- TCP has a 32-bit sequence number space, and a 16-bit window size (65Kbytes)
- At 1.2 Gbps:
  - the seq number can wrap in 28 seconds
  - The delay x BW at 100ms is 14.8MB
    - 200 window's worth!!!

TCP extensions for big fat pipes (RFC 1323)

CS519

- Timestamp extension
  - Allows the sender to put a 32-bit timestamp in the header
  - Mainly for RTT estimation
    - Receiver echoes it back, sender gets an accurate RTT
  - But receiver can also use it to detect wraparound
- Window scaling extension
  - Negotiate to interpret window as power-of-2 factor (i.e., left-shift window X bits)