EECS 122, Lecture 23

Today's Topics:

TCP Connection Management

Implementation Issues

Kevin Fall, kfall@cs.berkeley.edu

TCP Connections

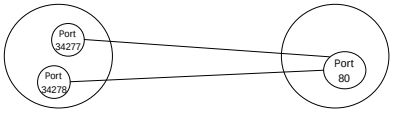
- TCP is a connection-oriented transport protocol which runs on top of a connectionless datagram network layer
- TCP connections are bi-directional
- TCP connections do not preserve *message boundaries*
- A TCP connection consists of two connected TCP endpoints

TCP Endpoints

- TCP uses port numbers similar to UDP, but the port space is disjoint from UDP
- Each TCP endpoint is identified by:
  - (IP address, port number)
- So, an entire connection is identified by the 4-tuple:
  - (IP1, port; IP2, port)

TCP Connections

- TCP connection 4-tuple provides the ability for a server to provide service to multiple clients:



A	B	34277	80
A	B	34278	80

Connections

The TCP Header

Source Port	Dest Port
Sequence Number	
ACK Number	
Data Offset / Reserved (zero) / Flags	Window
Checksum	Urgent Pointer
Options (if present), data, pad	

Flags Detail

TCP Header Fields

- source, dest port: source and destination port numbers at sender/receiver of this segment
- sequence number: offset of first data byte in transfer from sender of this segment to receiver
- ACK number: next sequence number the sender of this segment expects to see from peer on reverse-direction data flow

TCP Header Fields

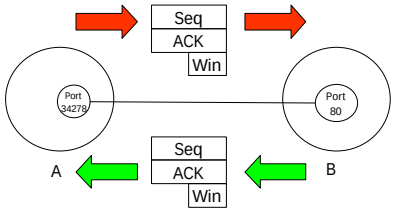
- data offset: number of 32-bit words comprising the TCP header (allows for variable-length header to contain TCP options; similar to the way IP works)
- window: space available at the sender of this segment (to receive data from peer on reverse-direction data flow)
- checksum: pseudoheader checksum (required in TCP)
- urgent pointer: offset of last byte of urgent data

TCP Header Fields

- control flags:
  - URG: segment contains urgent data
  - ACK: ACK field is valid (normally on)
  - PSH: TCP push function
  - RST: reset connection immediately
  - SYN: synchronize sequence numbers (open connection)
  - FIN: close connection

Steady State Operation

- Typical operation looks like this:



TCP Sequence Numbers

- Sequence numbers are used to provide reliability (ordering, protection from duplicates, etc)
- Each endpoint chooses its own initial sequence number (which is not fixed)
- Need a way for each endpoint to learn the other endpoint's sequence number space…
- Do this during connection establishment

TCP Connection Setup

- TCP Connections are bi-directional, so need each end to understand its peer's starting sequence number (not zero). Uses a three-way handshake:
  - A ----------> Seq k, SYN ----------> B
  - A <--- Seq j, ACK k+1, SYN+ACK <-- B
  - A ----> Seq k+1, ACK j+1, ACK ----> B

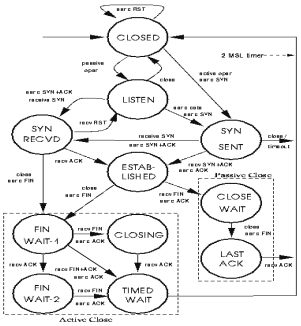
TCP Connection Tear-Down

- Need to tear down both directions of the TCP connection. Uses modified 3-way handshake:
  - A ---- Seq k, ACK j+1, FIN+ACK ------> B
  - A <--- Seq j, ACK k+1, ACK <-- B
  - … sometime later …
  - A <--- Seq j, ACK k+1, FIN+ACK <-- B
  - A ----> Seq k, ACK j+1, ACK ----> B

TCP State Transitions

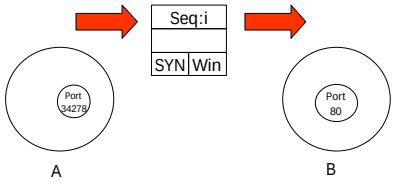
- Each end of each TCP connection operates as a finite state machine
- States are used primarily to indicate pending connection open or pending connection close

TCP State Diagram



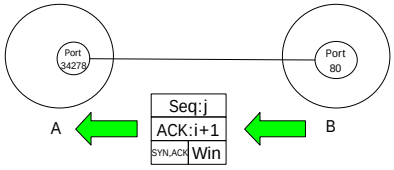
Example Connection

- Connection setup...



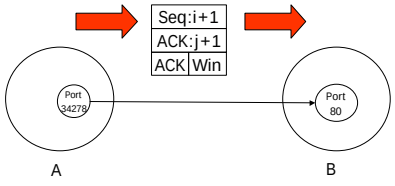
Example Connection

- Connection setup...



Example Connection

- Connection setup...

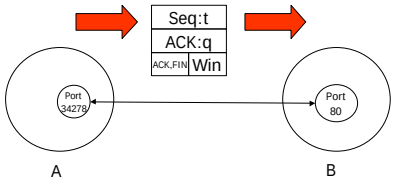


Example Connection

- Both ESTABLISHED, exchange data...

Example Connection

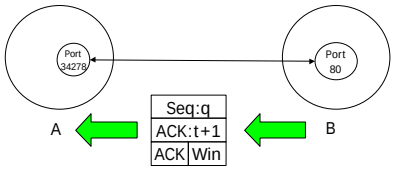
- Connection tear-down...



FIN_WAIT_1 ESTABLISHED

Example Connection

- Connection setup...



FIN_WAIT_1 CLOSE_WAIT

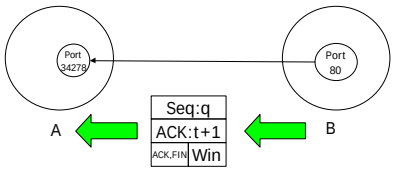
Example Connection

- App on B must now close...

FIN_WAIT_2 CLOSE_WAIT

Example Connection

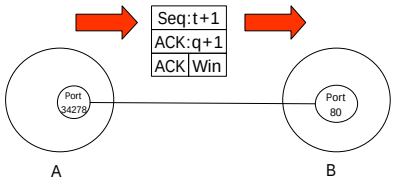
- Connection tear-down...



FIN_WAIT_2 LAST_ACK

Example Connection

- Connection tear-down...



TIMED_WAIT LAST_ACK

Example Connection

- Client now waits 2*MSL before returning to CLOSED state (B may RTX a FIN)

TIMED_WAIT CLOSED

2MSL Wait

- MSL (Max Segment Lifetime)
  - specified as 2 minutes (RFC793, 1122)
  - When cleaning up a connection, it is conceivable that packets belonging to the connection are still in the network
  - By holding the connection state, can absorb these without them returning to new instantiations of the same connection

A Related Issue

- So, we mentioned that TCPs do not use zero for an initial sequence number
- ISNs are supposed to be chosen so that packets belonging to an earlier instantiation of the same connection aren t accepted as valid
- One way is to use a system-wide ISN counter (increment every 4us say)
- Security concerns suggest random ISNs

Urgent Data

- In the TCP header, we see support for urgent data
- Urgent data may be indicated by an application, and appears immediately after a TCP header in the data stream
- Marked in a segment by the URG bit being on, and the pointer indicating the offset of the last byte of urgent data
- Urgent data overcomes flow control

Maximum Segment Size (MSS)

- TCP chooses the size of segments it will send
- During connection establishment, a TCP can specify to its peer its MSS. Peer will not send segments of size > MSS.
- Congestion control computations usually performed in terms of MSS
- Is peer s MSS the right size to use?

Choosing a Segment Size

- Never want to send segments longer than peer s MSS, but what about smaller:
  - if larger than path MTU, may fragment
  - if smaller than path MTU, inefficient
- Recall, we may use path MTU discovery:
  - set DF bit in IP header of each packet
  - look for ICMP unreachable messages; newer ones contain next-hop MTU
  - if lacking that, just choose common sizes…

The Push Function

- TCP spec has a notion of "push" function which would presumably cause data to be delivered without delay
- Generally, no way for app to express its desire to perform "push", thus no guarantee of effectiveness
- That said, most TCPs turn on the PSH bit when emptying send buffer

The TCP Checksum

- TCP checksum is completely analogous to the UDP checksum (same algorithm, but required for TCP)
- Uses slightly different pseudoheader:

Source IP		
Destination IP		
zero	proto	TCP length

- length field is derived entirely from IP

TCP Performance Issues

- Recall sliding window protocols obtain throughput ~ (w/RTT)
- If w is artificially limited, can reduce throughput:
  - send buffer limitation at sender
  - receive buffer limitation at receiver
  - inability of protocol to represent large windows

TCP Window Scaling Option

- Note that ordinary TCP header allocates only 16 bits for window advertisement
- Limits maximum window to 64KB, limiting max throughput
- TCPs supporting RFC1323 provide window scaling, shifts window field left by up to 14 (about 1GByte)

TCP Timestamp Option

- Conventional TCP keeps only 1 timer, maintained and endpoint
- With RFC1323 timestamps, each segment contains a timestamp
- Sender computes RTT estimate based on returned timestamps in ACKs
- Provides another useful property…

Wrapped Sequence Numbers

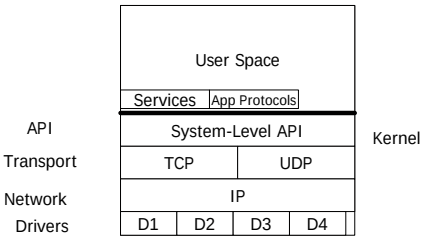
- TCP Sequence and ACK fields contain 32-bit values
- If we used a 1GB window, we could conceivably wrap the sequence number space to a point where old (misdirected) packets could reappear and contain a valid sequence number
- Timestamp field extends sequence number to resolve this condition

TCP Summary

- Reliable, connection oriented transport
- Stream service on IP datagram network
- Flow control, congestion control
- Silly window avoidance (Nagle)
- Adaptive retransmission with exponential backoff (Karn)
- Several options improve performance (described in RFC1323)

A Word on Implementation

- Typical implementations are of the following form:



The Application View

- Applications are written to a particular API (e.g. sockets) providing access to multiple protocols
- Basic functions:
  - open and close connections
  - send and receive datagrams
  - read and write data
  - set options

API Implementation

- Networking API is a combination of libraries and system calls:
  - libraries: application-layer, linked with app
  - system calls: function calls implemented within the operating system (executed via traps)
- Important internal objects:
  - buffering data structure (e.g. Unix mbufs)
  - lookup maps (e.g. Unix PCB's)
  - event and timer management

Buffers

- Used pervasively to hold packets, and sometimes for other things too
- Used to form queues:
  - Input: interface receive queue, IP input queue, IP fragment reassembly queues, TCP reassembly queues, app queue (socket buffer)
  - Output: app queue (socket buffer), TCP retransmission queue, interface output queue

Operations on Buffers

- Important operations on buffers:
  - enque, deque
  - prepend/remove headers
  - partial or full copy
- Other issues:
  - placement of packet in buffer may have profound impact on performance... for example, typically better to arrange for word alignment of IP header on receive and extra space at beginning on allocation

Lookup Maps

- Often need a way to map a received network message to its intended receiver:
  - Ethernet type field to network protocol
  - IP protocol field to transport protocol
  - port number pair to intended receive process
    - [note: this one may be very large]
- Original techniques based on linear search didn't scale. Now mostly hashing.

Timers and Events

- Uses for timers: TCP timers (RTX, persist, keep-alive, 2*MSL, delayed ACK), frag reassembly, ARP cache, IGMP reports
- Timers generally run off the computer's clock interrupt, often divided down to intervals such as 10ms up to 500ms
- Two common techniques: use system-defined periodic dispatcher [common in TCP/IP], use specific deadline