Monday, May 29, 2017

Introduction to Computer Networking

Transport

TCP Service Model

- Transmission Control Protocol
- used by 95% of Internet applications
- reliable, end-to-end, bidirectional byte-stream service
- Transport layer
- two way communication channel between TCP peers = connection
- At both ends TCP keeps state machine
- 3 way handshake
  - Establish connection
  - SYN = synchronize
    - A wants to establish connection with TCP layer at B
    - A sends base number it will use to identify bytes in the byte stream
      - “0” = numbers will start at zero
      - “1000” = numbers will start at thousand
  - SYN + ACK = synchronize, acknowledgment
    - B sends ACK because it is acknowledging A’s request, agreeing to establish connection
  - ACK
    - A wants to accept the connection
- Stream of bytes
- data in order
- Stream of bytes is delivered by TCP segments
  - A puts bytes from the stream into a TCP segment

    - A hands it to the IP layer
    - IP layer delivers it to B
- in practice TCP layer may need to be transmitted multiple times
    - Segment is dropped, A doesn't receive acknowledgement
- it isn't very efficient to send lots of data at once
- A and B finish sending data
    - close connection = “teardown”
    - tell each other they are closing connection, clean up the state associated with the state machine
- Teardown
    - FIN = finish
    - ACK = acknowledgement
        - This closes down the data stream from A to B
        - if B has new data to send to A, B can keep sending data
            - The ACK can also carry new data
        - FIN
        - ACK
- TCP service model
- Sequence numbers
    - every segments header carries the sequence number - in the stream of bytes - of the first byte in segment
        - Both sides agree that the sequence numbers start at 1000
        - first segment will have sequence number of 1000
        - segment carries 500 bytes then next segment will carry the sequence number 1500
        - segment gets lost -> sequence number will be incorrect

    - if data arrives out of order -> resequenced them to the correct order using sequence number

- Flow control
  - if A is much faster than B
  - Receiver keeps telling the sender if it can keep sending
    - How much room has it for new data
- Congestion
  - TCP tries to divide up the network capacity equally among all the TCP connections using the network
- TCP segment header is much more complicated than IP and Ethernet headers
  - Because TCP connection is reliable
- Acknowledgment sequence number
  - Which byte we are expecting next
  - 501 - we received byte 500
- sequence numbers for every direction
- TCP options fields
  - Extra fields that were added after the TCP standard was created
  - Header Length field tells us how many option fields are present
- Flags
  - ACK
  - SYN
  - FIN
  - PSH
    - tells TCP layer at other end to deliver data immediately upon arrival
    - don't wait for more data
    - short segments carrying time critical data (key stroke)

  - URG
- TCP connection is uniquely identified by 5 pieces of information in TCP and IP headers
  - IP source and destination address
  - IP Protocol ID for TCP - tells us the connection is TCP
  - TCP source and destination ports
- A must have unique source port ID
  - it can’t use the same port it is already using with another connection to the same service on B
  - A minimize chances
    - Increments the source port number for every new connection
- ISN = initial sequence number
  - Random

UDP Service Model

- User Datagram Protocol
- no guarantee delivery service
  - Application doesn't need it
  - Application handles retransmission in its own way
- UDP takes application data -> create UDP datagram -> hands it to the network layer
- Four header fields
- Length
  - Data + header
- UDP checksum
  - Optional when using IPv4
  - sender doesn't include checksum - field is filled with 0s
  - UDP header + data

        - Includes part of the IP header
            - allows the UDP layer to detect datagrams that were delivered to the wrong destination
    - port numbers
    - UDP demultiplexing Protocol
        - Demultiplexing mechanism to divide to the stream of UDP datagrams and send them to the correct process
    - connectionless datagram service
        - no connection is establish because none is needed (no handshake)
        - all of the information is self contained in the datagram
        - any order of packet - application needs to resequenced the data itself
    - unreliable delivery
        - no acknowledgement that data reached the other end
        - cant detect missing datagrams
        - application can ask for the data again by itself
            - build retransmission mechanism on top of UDP
            - early version of NFS (network file system) did this
                - they didn't want to use the sliding window used by TCP
                - they created their own inside application
    - application that doesn't need reliable delivery
        - simple request-response application
        - DNS
            - domain name system
            - turns a hostname into an IP address
            - uses UDP
                - request is fully contained in one UDP datagram

- DHCP
    - uses UDP for same reasons
- real-time streaming audio
    - less common to use UDP instead of TCP today

ICMP Service Model

- Internet Control Message Protocol
- used to report errors and diagnose problems with the network layer
- 3 mechanisms that we use to make the network layer work in the internet
    - IP
        - creation of IP datagrams
        - Hop by hop delivery from end to end
    - Routing tables
    - ICMP
        - Communicates network layer information between end hosts and routers
        - reports error conditions
        - Help diagnose problems
- runs above network layer, transport layer protocol
- end host or router want to report an error using ICMP
    - it puts information into an ICMP payload
    - hands it to IP -> send
- Example
    - Web client running as application
    - Client wants to access an HTTP server
    - packet comes to router
    - Address that is put inhere is to a network that this router has no information about in its forwarding table

  - router will send back a message “destination network unreachable”
- ICMP doesn't attempt to resend its message,
- Doesn't maintain any state
- router wants to send back ICMP message
  - it takes the header
    - Takes source and destination address
    - Place it into an ICMP message
  - Type of error and its code
- ICMP message types
  - Destination Network Unreachable
    - Router doesn't know network where it should send packet
  - Destination Host Unreachable
    - IP datagram gets to the last router but then the last router doesn't know where the host is
  - Destination Port Unreachable
    - it doesn't recognize port or protocol ID at the other end
- Ping command
  - Uses ICMP
  - Test the liveness and connectivity of another host
  - ping 156.1.1.1 or ping www.seznam.cz
  - Calls ICMP directly, sends ICMP echo request
  - B echo reply
- Traceroute
  - application that tells us the path that packets take through the network
  - Traceroute seznam.cz
  - path and round trip delay to each of the routers

- ICMP
- A sends UDP message to B
  - Time to live in IP header is set to one
  - router must discard the message when TTL is 0 and send back ICMP message
- Router sends ICMP message to A “TTL expired”
- and so on until datagram gets to B
- B sends back “port unreachable”
  - A chose weird port that B will not know

End-to-End Principle

- Refers to 2 principles
  - Correctness
  - Strong end-to-end principle
- why doesn't network layer do more
  - Compress data, add security, reformat
- end to end principle
  - Network can do many things to help
  - end points must be responsible
    - Nobody else has the information necessary to do this correctly
  - you want your application secure = end-to-end security implemented in the application
- End points also have to detect errors, not just network
  - link error detection was designed for errors in transmission, not errors in storage
- Wire link layers are very reliable but wireless are not
- Stronger than the first one
  - First one said that you have to implement something end-to-end but you can also implement it in the middle for performance improvements

- strong principle says not to implement it in the middle, only in the end
  - Flexibility and simplicity

Error Detection

- Network error detection algorithms
  - Checksum
  - cyclic redundancy code (CRC)
  - Message authentication code (MAC)
- We have a payload of data
  - we calculate some error detection bits over that data
  - either append it or prepend it to the payload
    - Ethernet - appends cyclic redundancy code
    - Transport Layer Security - appends a message authentication code
    - IP - prepends a checksum placed in IP header
  - TLS and Ethernet - footers, protocol information which follow the payload
- 3 commonly used error detection algorithms
  - Checksum
    - IP, TCP
    - easy to compute
    - weak error detection guarantees
  - Cyclic Redundancy code
    - Ethernet
    - Harder to compute
    - computes the remain of a polynomial
  - Message Authentication Code
    - TLS
    - Cryptographic transformation of data

- Combines the packet with some secret information to generate a value
- Someone can only generate or check the MAC if they have the secret
- so if you receive a packet and its MAC is correct then you are sure that computed MAC has the secret
- https
- not great for catching errors, mainly for security

Finite State Machines

- finite state machine is composed of a finite number of states
- State
  - Particular configuration of the system
- Edges between states define how we transition between them
  - What events cause transition to occur
  - what actions the system will take when that transition occurs
    - not all transition have actions associated with them
    - system is in state and an event arrives for which there is no transition -> FSM is undefined
    - Multiple transition from single state
- Example - HTTP request
  - Page = idle
  - we want to load a new page requesting state
    - event is load a new page
    - Action is open a connection to the web server
  - Request pending state
    - Having more resources to request, requesting recourse with HTTP GET
- if you want to be explicit, what happens on each state for every event
  - What happens if the connection closes when we are in request pending state

- TCP connection
  - 12 state
  - 4 parts
    - Top 4 states
      - Describes how you open a TCP connection
    - Center state “estabilish”
      - TCP is sending and receiving data
    - Bottom 6 states
      - How connection close
    - Bottom state “close”
      - Connection is closed and node can forget about it

Reliable Communications - Stop and Wait

- Flow control
- Basic problem
  - sender can send data faster than receiver can process it
    - Slower processor, networking card
- Approach
  - Don’t send more packets than B can process
    - Receiver gives feedback
  - stop and wait
  - sliding window
- Principles
  - At most one packet in flight at any time
    - Waits for acknowledgement
    - If it doesn't receive packet -> assume that packet is lost -> resend
  - sender sends one packet

- 4 cases
  - nothing is lost, data is lost, acknowledgement is lost, acknowledgement is delayed
  - acknowledgement is delayed
    - the second ACK is sent by receiver -> sender doesn't know whether this ACK is for retransmission or new data packet -> we can have error
    - finite state machine has to keep track of it
- how do you detect duplicates?
  - Is ACK for retransmission or duplicated copies of packets vs new data
  - solution
    - one big counter on all data and ACK
      - Receiver can tell if new data or duplicate
    - simplifying assumptions
      - Network doesn't duplicate packets
      - Packets are not delayed multiple timeouts
        - these problems can be solved by increasing sequence number

Reliable Communication - Sliding Window

- Used in most high performance protocols
- Stop and wait
  - 10 Mbps link -> Boston node can receive 10 Mbps
  - Round trip time (RTT) is 50 milliseconds
  - We send Ethernet frames (Ethernet frame = max 12 kb)
  - 1000/50 milliseconds = we can send 20 packet a second
  - 20 packets * 12 kb = 240 kbps
  - Stop and wait protocol is using 2% of what could be used
  - Inefficient

- Sliding window
  - Generalization of stop and wait
  - Allow multiple un-ACKed segments
    - n=5
      - San Fransisco can have 5 packets in flight
      - Boston can have 5 ACK on flight
  - Bound on number of un-ACKed segments called window
  - can keep pipe full
  - 10 Mbps / 20 trip times = 500 kb per trip time
  - 500 kb / 12 kb = 41 packets
  - it allows 10 Mbps connection
- sliding window sender
  - Every segment has sequence number
  - Sender maintains 3 variables
    - send window size (SWS)
    - Last acknowledgement received (LAR)
    - Last segment sent (LSS)
  - Maintain invariant (LSS- LAR <= SWS)
  - Receive ACK -> advance LAR on new ACK -> buffer up to SWS segment
  - Stalling window
    - n=3, send packet 1, 2, 3
    - packet 1 is lost, 2 and 3 are ACK
    - Sender can’t send packet 4,… until it receives ACK for packet 1
    - Although most of the data in the window has been delivered it can’t move past the first unACKed piece of data
- sliding window receiver

- Receiver maintains 3 variables
  - Receive window size (RWS)
  - Last acceptable segment (LAS)
  - Last segment received (LSR)
- Maintains invariant (LAS - LSR <= RWS)
- if received packet is < LAS, send ACK
  - Cummulative ACK
    - what is the end of contiguous data you received
    - if received 1,2,3,5 -> ACK 3
  - note TCP ACKs are next expected data (ACK 4)
    - n+1
    - first missing byte

- RWS, SWS and sequence space
  - RWS >= 1, SWS >= 1, RWS <= SWS
  - if RWS =1m “go back N” protocol, need SWS+1 sequence numbers
    - Example
      - RWS =1, SWS=3
      - A sends 0,1,2 -> B ACKs 0,1,2
        - when B ACK 0 -> A sends 3
        - When B ACK 1 -> A sends 4
      - A sends 3,4,5
      - 3 is dropped
      - B receives 4,5 -> sends ACK for them
      - A times out and resends 3
      - RWS was size one so receive can’t buffer 4 and 5
      - sender has to retransmit 3,4,5

- if RWS=SWS, need 2SWS sequence numbers
- Generally need RWS+SWS sequence numbers
  - RWS packets in unknown state (ACK doesn't have to arrive)
  - SWS packets in flight must not overflow sequence number space
- TCP flow control
  - TCP is sliding window protocol and uses flow control
  - Receiver advertises RWS using window field
  - sender can only send data up to LAR + window
  - Receiver specifies the flow control window using the window field
    - = buffer size, how much I can accept
  - TCP receiver will only handle data equal to the acknowledged sequence number plus the window
    - so sender isn't allowed to send data past ACK+window
- Example
  - RWS=2, SWS=3
  - A sends 0,1,2
    - 0 is ACK -> send 3, 1 is ACK -> send 4, 2 is ACK send 5
    - 3 is ACK -> send 6
    - 4 lost -> B waits for 4 -> send another ACK for 3 (cumulative ACK)
    - A resends 4 -> 4 arrives to B
    - RWS=2 so 5 is buffered, B can also ACK 5

Reliable Communications - Retransmission Strategies

- per-packet timer
- Strategies
  - Go back N
    - single is lost -> retransmit outstanding window of packets

    - assumes all packets are lost
  - selective repeat
    - Retransmit only lost packet
    - assumes only one packet is lost
- if there os bunch of loses, selective repeat protocol is slower
- Better to use go back n
  - Because receiver has RWS=1 sender will have to retransmit every single packet in the window
- you shouldn't retransmit earlier than you should
  - Inflating number of packets that are beyond your window size

Reliable Communications - TCP header

- 20 bytes (5x32 octets)
- Options after TCP header
- Acknowledgement number=5000 - received up to 5000 byte
  - Acknowledges last byte received + 1
- Sequence number=4000 and 500 bytes data - byte 4000 to 4499
  - Receiver sends ACK number=4500
- ACK packet
  - TCP segment that has no data
    - Trafic is undirectional
  - ACK added to TCP segment with dat
    - Traffic is bidirectional
- Checksum
  - Computed over TCP pseudo header
    - TCP header + some of the IP header + data
- Control bits - U, A, P, R, S, F

- U
  - Urgent bit
- P
  - Push bit
  - push data to the receiving application
- A = ACK
- R = reset
  - Reset connection
- S = SYN
  - “this is my starting sequence number, synchronize to this number”
- F = FIN

- it is better not to have starting sequence number 0
  - Security problems - people can guess it
- Offset
  - what offset within the segment does data begin

Reliable Communications Connection Setup and Teardown

- 3 way handshake
  - SYN, Sn
    - random sequence number
  - SYN, Sp, ACK, S(a+1)
  - S(a+1), ACK, S(p+1)
    - Sequence number - where bytes would start but there are no byte
- Simultaneous open
  - another way how to open connection
  - if both side know each other port numbers
  - Both sides send SYNs at the same time

- how to open connection
  - SYN, Sa
  - SYN, Sp
  - SYN, Sa, ACK, S(p+1)
  - SYN, Sp, ACK S(a+1)
- Wireshark
  - Sequence number = 0
    - It actually isn't 0
    - Wireshark display relative sequence number - relative to beginning of the stream
- connection teardown
  - FIN but says no more data to send
    - Caused by close() or shutdown() on other end
  - Both sides must send FIN to terminate a connection
  - Typical teardown exchange
    - A -> B: FIN, seq Sa, ack Sb
    - B -> A: ack S(a+1)
    - B -> A: FIN, seq Sb, ack S(a+1)
    - A -> B: ack S(b+1)
  - can also have simultaneous close
- when can we close connection?
  - Problems with closed socket
    - Final ack is lost
    - same port is immediately reuse for a new connection
  - Solution
    - Active closer goes into time wait before it can reuse its state
    - Keep socket around for 2 maximum segment lifetime

- this solution can cause problems to servers
    - OS has too many sockets in time wait
    - there are trick - send a reset, delete the socket, set socket option to time 0
    - OS won't let you restart server because port still in use
- for both - active and passive