3600 Exam 2 Study Guide

horizontal line

# UDP and Internet Checksum

* Goals of a transport layer protocol is to get one application to another application.
* Network layer protocols move data from host to host
* Link layer protocols move data from machine to machine.
* UDP
  + Adds source port
  + Adds destination port
  + Adds payload length
  + Adds packet checksum
  + Ports are the address for applications
    - MAC addresses identify individual network interface cards
    - IP addresses identity individual machines
    - Ports direct info to individual applications
* UDP is UNRELIABLE
  + No packet acknowledgements, retransmission on drops, timeouts
* UDP is CONNECTIONLESS
  + If a host has an open uDP port, another other host can deliver packets to that <host, port> without prior negotiation
* Udp is good for single request-reply messages
  + No ongoing communication, just request followed by reply
* Udp accomplishes request-reply messages with two packets
  + Tcp require five packets for request-reply because of establishing connection prior to communication
* DNS and DHCP queries both use UDP (they rely on request-reply)
* UDP is good for realtime transport
  + Think of like skype or a video game. Better to lose a single packet than many.
  + Loss of a single packet is easier to recover from.
* When TCP drops a packet, all later data os queued until the dropped packet is successfully delivered
* When UDP drops a packet, it just moves on to next packet
* Why NOT use UDP
  + You have to implement yourself
  + Security issues because it is easy to flood
  + Congestion issues. USP doesnt self limit sending rate if a network is congested but TCP does automatically
* UDP Header
  + 8 byte header
    - Source port
    - Destination port
    - Length of header+payload in bytes
    - Checksum
* TO CHECK INTERNET CHECKSUM
  + Divide the packet into 16 bit words
  + Sum all of the words, carry any overflow bits
  + Perform 1s complement on result
  + When you get the checksum, add it with the packet and if it was received correctly it will result in all 1s
  + If the packed data has an odd number of bytes, pad the data with 0x0000 DURING the checksum computation process. Do NOT include the 0x0000 when sending data.

[UDP Checksum Practice Problems](https://docs.google.com/document/d/1T2R0iR0SWDXDZaKIwtcebHjknsb7OyEHw3IuYWhNPow/edit?usp=sharing)

# TCP Transmission Control Protocol

* TCP is:
  + A transport layer protocol
  + Reliable
  + Connection-oriented
  + Stream oriented
  + An end-to-end protocol
* TCP is responsible for delivering data to specific applications on specific machines
  + Packets are multiplexed/demultiplexed using port numbers
* Multiplexing: TCP takes data from a bunch of different applications and seds them all out through the same network link
* Demultiplexing: TCP receives a bunch of data from a network link and delivers it to each of the appropriate applications
* In order, TCP does these:
  + Checksums
  + Ack messages
  + Sequence numbers
  + Timers
  + Pipelining
  + Windowing
* TCP is CONNECTION ORIENTED
  + Establishes an ongoing connection between hosts
    - Established via a three way handshaking procedure
    - A logical connection, resides in software, no dedicated resources
    - Only the hosts know about this connection, intermediate network elements dont implement the transport layer
* TCP is STREAM ORIENTED
  + Tcp sends data through discrete packets, sending and receiving applications are not responsible for parsing these discrete packets
  + Tcp functions as a byte stream
    - Applications give to tcp which splits into packets and sends it out
    - Tcp receives packets, combines the data into a stream of bytes and delivers this to the appropriate application
* TCP is an END TO END protocol
  + Tcp places full responsibility for all transport issues on the endpoints
    - The underlying network is not responsible or managing either data loss, data corruption, or network congestion
  + Tcp is oblivious to what goes on in lower layers
  + Tcp behaves like there is a direct connection between two applications. It has to manage everything required to maintain this behavior
* TCP header
  + Source port is the port data is sent from
  + Destination port s the actual port number the data is sent to
  + Sequence number is the position the first data byte should be inserted into the receivers data stream
  + Acknowledgement number is the position of the next expected byte in the senders data stream
  + Data offset: the length of the header in 32 bit words
    - If no options are present then max number of optional words is 10
    - Data offset is 4 bits, max number of optional words is 10
  + Window size is the current size of sliding window employed by tcp
  + Checksum serves the same function as the udp checksum
  + Urgent pointer is seldom used field related to high priority data
* TCP HEADER FLAGS
  + SYN: synchronize. Set to true for all packets involved in the handshake
  + ACK: acknowledgement: indicates the ack field is valid. Set to true for all but the first packet in the handshake
  + FIN: finish. Set to true for all packets involved in closing the connection
  + PSH: push. Indicates packet should be delivered at destination.
  + RST: reset. Indicates various errors.
  + URG: urgent seldom used, for high priority data
  + CWR and ECE: part of explicit congestion notification mechanism.
* SEQ numbers dont refer to a specific packet, to the position of the data in the tcp byte stream.
* ACK number doesnt refer to a specific packet but to the position of the data in tcp’s byte stream
  + No special ack packets, now all packets contain ack message
  + A special ack can be sent if no data ready to be sent
* Each tcp host manages their own byte streams
* Three stage handshake:
  + Most accomplish 4 things:
    - Communicate client\_isn to server
    - Send an ack to the client that the server knows client\_isn
    - Communicate server\_isn to the client
    - Send an ack to the server that the client knows server\_isn

# More TCP

* TCP samples the RTT by computing the time between when a segment is sent and ack’d
  + Doesnt calculate rtt for all segments, only running one at a time
  + Dont calculate sampleRTT for retransmitted segments
* We want to compute an average of these sample readings as individual readings may not accurately represent RTT
* Estimated RTT = (1 - a) \* EstimatedRTT n-1 + a \* sampleRTT
  + A recommended to be ⅛
* DevRTT = (1 - B) \* DevRTT n - 1 + B \* |SampleRTT - EstimatedRTT|
  + Common value for b is 0.25
  + This is weighted oving average of the difference between the sampleRTT and estimated RTT
* Timeout = EstimatedRTT + 4 \* DevRTT
* TCP ACK (No Data) Generation

| **Event at receiver** | **TCP receiver action** |
| --- | --- |
| Arrival of an in-order segment with the expected seq #. All data up to expected seq # has already been ACK’d | Delayed ACK. Wait up to 500 ms for next segment. If no next segment, send ACK |
| Arrival of in-order segment with expected seq #. One other segment has an ACK pending | Immediately send a single cumulative ACK, ACKing both in-order segments |
| Arrival of an out-of-order segment with higher-than-expected seq #. Gap detected! | Immediately send duplicate ACK, indicating seq # of next expected byte |
| Arrival of segment that partially or completely fills a gap | Immediately send ACK, provided the segments starts at the lower end of the gap |

* Timeouts indicate poor network performance
  + Either high latency or packet loss, prevented an ack from being received.
  + Tcp doubles the timeout every time the timer expires without an ack
  + This can lead to rapid growth of timeout, resulting in a long delay before resending packets
* Fast Retransmit
  + Timeout indicate poor network performance
  + We can detect lost segments via duplicate acks
  + Tcp fast retransmit
    - If the sender receives 3 duplicate acks for the same segment, it will immediately resent the segment with smallest seq #
* Congestion control is about preventing this network from getting overloaded and dropping packets
* Flow control is about preventing the receivers buffer from getting overloaded and dropping packets
  + Data isnt delivered to the receiving program immediately, buffers incoming data until it is requested
  + Possible to overflow this buffer by sending data too fast
  + Flow control is a speed matching service, matches speed of requester is reading and sender is sending
  + We want to match the rate at which the application process requests data and the sender sends data
* Bytesbuffered = lastbyterecvd - lastbyteread
* lastbytercvd/read are the # of bytes in the byte stream
* Spaceremaining = buffersize-bytesbuffered
* Sender must ensure that lastbytesent - lastbyteacked <= spaceremaining
* If host b has 0 space remaining, host a will never learn when host b has space available because host b will not send any new packets
* Host a wont send any new packets since host bs buffer is full
  + To avoid this problem, host a will keep sending 1 byte segments when spaceremaining = 0 bytes

# TCP and Congestion Control

* Approaches to Congestion Control
  + End-to-End Congestion Control
    - Network layer provides no support to the transport layer, all congestion must be managed at transport layer
    - Tcp takes this approach
  + Network assisted congestion control
    - Network layer provides some information to the transport layer about congestion

# IPv4 and IPv6 Protocol

* IPv4 Protocol Header
  + First 32 bit word
    - IHL: the length of the header in 32 bit words
    - DS field: used to specify preferential handling for designated packets VoIP packets, or other real-time protocols.
    - ECN: explicit congestion notification allows routers to mark packets when they are experiencing congestion
    - Total length: length of the IPv4 packet
  + Second 32-bit word
    - Identification
    - Flags
    - Fragment offset
  + Third 32 bit word
    - TTL: TTL decreases by 1 for each hop taken. If TTL reaches 0, the packet is discarded. This helps prevent routing loops
    - Protocol: Identifies the protocol used in the payload
    - Header Checksum: checksum used to detect corrupted headers.
  + Remaining 32-bit words
    - Source address: IPv4 address of the source
    - Destination address: IPv4 address of the destination
    - IPv4 Options: many optional features supported, rarely ever used.
* IPv4 addresses are made of 4 bytes
* IP addresses are divided into subnets
  + IP address made of four bytes, designating the lowest address in the subnet block
  + Number of bits in the IP address allocated to the subnet block, starting with the highest bit
* Organizations are allocated by subnet blocks by ICANN and their subsidiaries
  + ISP’s get a block of subnets that they then allocate to customers
* Lookup Tables
  + When a router receives a packet, it determines which outgoing link to forward it on based on the subnet the destination IP address belongs to
    - Different subnets are assigned to specific outgoing links based on the shortest path to where that subnet is located
  + Longest prefix rule: if an address matches more than one entry in the table, forward to the entry with the longest match
* Special IPv4 addresses
  + Each network interface on a machine is assigned an IP address
  + Three blocks of addresses are reserved for private use
  + If you run command ipconfig, you’ll get a report about network interfaces
  + Each machine also has a loopback address
    - Helpful for testing networks because it allows two or more clients running on the same machine
  + Includes a broadcast address
    - Forwards all data to all computers connected to the local network
* IPv4 Fragmentation
  + Maximum transmission unit (MTU) is 1500 B
  + Sometimes a packet will need to cross a link with a smaller effective MTU than 1500 b
    - We handle this:
    - Fragment the packet, send it out in smaller chunks
    - Reassembled at final destination
    - Less efficient, duplicates headers
  + Three header fields make this work
  + 16-bit fragments for each datagram send fragments all share the ID of the original datagram
  + Flags contain
    - Dont fragment mark, will drop packet
    - More fragments flag, set if part of a fragment, unless its the last fragment
  + Fragment offset is the byte offset of the included data in 8 byte increments
* IPv6
  + Created to extend the address space
  + Uses 32 bit addresses
    - Max unique address 2^32
    - We have run out of unique addresses
  + IPv6 uses 128 bit addresses
    - Max unique address 2^128
    - Conventionally written as 32 nibbles divided into 8 hexadecimal tetrads
  + By convention, most ipv6 addresses are divided into two parts
    - Higher 64 bits subnet prefix
      * Used for routing, identifying special classes of messages
    - Lower 64 bits, network interface card identifier
      * Identify unique NICs within a network
  + No more variable length subnets
  + Instead of broadcasting, IPv6 implements multicasting groups
    - Broadcasting, send to everyone
    - Multicasting, send to everyone registered within group
  + Cuts down on unnecessary broadcast traffic

# More Network Latency Stuff

* Network Address Translation (NAT)
  + Many computers with distant addresses can be hidden behind a single IP address assigned to a NAT router, which connects to the internet
  + NAT dramatically increases the number of devices the internet can support
  + The NAT router can improve security with the inclusion of a firewall
  + IPv4 supports around 4 billion unique addresses
    - This is nowhere near enough addresses
    - There are over 20 billion devices connected to the internet today
  + NAT provides a solution to many devices being connected
    - NAT enabled routers connect to the outside world
    - The NAT receives a single IP address which is exposed to the outside
    - Internally, the NAT hosts a network on one of the reserved subnets
* Sending data through a NAT is easy
  + When the nAT receives a packet from a host inside the NAT:
    - It replaces the source IP with the NATs IP
    - It replaces the source port with a random port
  + The source port is changed to the server can respond to the NAT
    - External machines can’t send messages directly to hosts behind a NAT
  + To send data through a NAT back to the original host, we need to do this with a forwarding table
    - This forwarding table specifically supports packets begging set in response to a message from a machine inside of the NAT
* We need five pieces of information to create the forwarding table
  + The remote host
  + The remote port
  + The outside port
  + The inside host
  + The inside port
* There can be three different scenarios when using IPv4 and IPv6
  + A host is running both of them at the same time
    - This is referred to as dual stack
    - If a client wants to connect to an IPv4 only host, it uses IPv4
    - If a client wants to connect to an IPv6 only host, it uses IP6
    - If a client want to connect to another dual stack host, it performs a dns lookup
    - IPv6 addresses are stored in DNS AAAA records
    - A client can request both A and AAAA records, but they have to make a decision on which to use.
      * Sometimes DNS returns only one, therefore making the decision for the client
    - Addresses are looked up in the table, using longest match rule
    - Each address is assigned a precedence based on the table
    - We usually want to use IPv6 when available
  + A host is running IPv4 and needs to talk to a host using IPv6
    - If native IPv6 is not supported, clients can connect to IPv6 hosts using 6in4 packet tunneling
      * A tunnel broker sits between client and the rest of the internet
      * Traffic from IPv6 hosts is funneled to the client through the broker
      * Client funnels traffic to IPv6 host through broker
    - Done by creating a virtual network interface on the client
      * Virtual interface assigned an IPv6 address and its default router is assigned to the tunnel broker
    - Tunnel broker must be configured to accept traffic
  + A host is running IPv6 and needs to talk to a host using IPv4
    - We will use an IPv6 to IPv4 translator, NAT64 and DNS64
    - Client will make a DNS request like normal to a DNS64 resolver
    - If the destination only has an IPv4 address, it creates a synthetic IPv6 address and r returns it as an AAAA record
      * Synthetic address consists of a prefix and the IPv4 address of the host
      * Prefix connects the client to a NAT64 translator which will handle the next part of the process
    - Any address beginning with the nAT64 prefix will be delivered to it
    - the NAT64 has several IP4 addresses assigned to it. It will translate requests from clients so that they come from one of these addresses
      * One IPv4 address may be assigned to multiple IPv6 addresses, due to the limited supply of IPv4 addresses
* DHCP: Dynamic Host Configuration Protocol
  + Machines get assigned IP addresses via static IPs, configured by an administrator, permanent association
  + And via dynamic assignment
  + Static assignment works for some machines, terrible for others
  + DHCP server automatically assigned IP addresses to computers based on the addresses that are available
  + DHCP is a client-server process that supports the network layer
    - Actually runs the application layer
    - A DHCP server must exist on the network for it to function
  + There is a four stage process
    - Server discovery
      * New host sends DHCP discovery message to UDP port 67
      * Broadcasts a message that is sent to all nodes within the current subnet
    - Server offer
      * When a server receives a discovery message, it can make an offer of an address to the client
      * Server responds with a broadcast message which includes the transaction ID ( must be a broadcast message because client doesnt have IP address yet)
    - Request
      * Client chooses from the offers it receives and responds with a DHCP request message that echos the parameters offered by the DHCP server
    - ACK
      * Finalizes the whole process
* ARP and ICMP
  + Both are network protocols
  + ARP: Address resolution protocol
    - Enables devices to discover the MAC address associated with IP addresses
  + ICMP internet control message protocol
    - Communicates IP-level status and error messages to other hosts
* ARP
  + We use ARP protocol to convert IP addresses to MACs
  + Each IP node has an ARP table
    - It contains IP address, MAC address, and TTL (usually between 30 sec to 10 min)
  + ARP cn be used to detect duplicate IP addresses
  + On connection to a LAN hosts broadcasts a ARP packet addressed to its own IP address
* ICMP
  + Used by network devices to share information about status and errors as well as by network administrators.
  + 4 fields
    - Type of message
    - Sub type of message (code)
    - Checksum to detect if packet is valid
    - Rest of heder stores any data important to message
  + ICMP data is encapsulated by an IPv4 or IPv6 header
  + The rest of the header data can be used to determine what packet this ICMP message is in response to
  + Echo
    - Used by ping utility
    - Identifier and sequence number are used by the host to determine which reply is in response to which request
    - Payload is variable, could contain the time the packet was transmitted allowing for stateless computation of RTT
      * Echo replies must contain same payload as the echo request
  + Time exceeded
    - If a router discards a packet due to the TTL value reaching zero it will send a time exceeded message to the host
    - Used to traceroute to identify the router on the path to the specific host and the time to reach them
  + Destination unreadable
    - Generated by a router to inform the original sender that its destination is unreachable, many possible reasons
  + Redirect
    - The router can redirect the packet to a more optimal route, sends this to the host via an ICMP redirection
  + ICMP messages don't transmit useful data from host to host, but are essential for network utilities and error detection
  + ICMP messages are also used by some routing update algorithms which are used to construct the forwarding tables of routers