Study Guide Midterm

Wednesday, February 22, 2017 1:34 AM

CHAPTER 1:

1.1 What Is the Internet?

This section focuses on the public internet, describing it, through the basic hardware and software that makes up the internet, as well as the networking infrastructure that provides services to distributed applications.

1.1.1 Nuts and Bolts Description

- Internet is a network that connects millions of devices in the world.
- Devices that are hooked up into the internet are called hosts, or end/edge systems.
- End systems are connected by a network of communication links and packet switches, with the links being ethernet or wifi waves. Different comm links communicate at different rates, called transmissions rates measured in bandwidth.
- A packet is a collection of data, part of a larger piece of data that has been segmented into single packets. These are sent between end systems, and reassembled on the end systems.
- A packet switch takes packets arriving on one of its incoming comm links, and forwards it to one of its outgoing comm links. Two common packet switches are routers and link layer switches.
- A router is usually used in the network core, while link layer switches are used in access networks. The sequence that packets take through an end system to another end system through a series of comm links and packet switches is called a route. These routes are kinda like mail, with packet switches being intersections, and packets being trucks between the end system buildings.
- End systems access the internet through ISPs, which are commercially owned packet switch and comm link collections that travel out to residential and commercial networks. Lower level ISPs connect into national ISPs, so something like WOW will pay to feed off of a network like Verizon or ATT.
- All of these networks run protocols that control sending/receiving data. Two important ones are Transmission Control Protocol (TCP) and Internet Protocol (IP). IP specifies the format of the packets that are sent through routers and end systems.
- Internet standards are used to agree on what each protocol does. The Internet Engineering Task Force develops these standards, and the actual standards are found in RFC docs, or Requests for Comments. These docs are very technical.

1.1.2 A Services Description

- The internet can also be described as an infrastructure that provides services to applications. Applications that use the internet are called distributed applications, due to multiple end systems exchanging info with each other.
- Application Programming Interfaces (API) provided by end systems on the internet specify how a program running on one end system asks the internet infrastructure to deliver data to a specific destination on another system. The internet has an API itself, which tells us how to send data.

1.1.3 What's A Protocol?

- A protocol at a human level dictates how a certain action is carried out. Asking a question in class involves the prof offering to answer, you raising your hand, them acknowledging, and you asking the question and them answering.
- All activity on the internet that involves two entitites that communicate is governed by protocols. A lot of the book is just basically learning protocols. A protocol, oficially, defines the format and order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission/receiving of a message. Mastering networking means understanding the what, why, and how of protocols.

1.2.1 Access Networks

- The applications and end systems are the edge of the network. The access network is the network that physically connects an end system to the first router on a path from the end sys to any other end sys. These are home networks, business networks, and even mobile networks. Home networks can use DSL, which is provided directly from telcom companies. DSL sends network data through a modem into the ISP office to other routers across the world. Cable networks use coax cables to connect to nodes that forward to an ISP as well. Since coaxial cables are shared in a neighborhood, internet access is slowed due to multiple users using the same line. DSL has a direct connection, but offers slower speeds. Fiber optic networks are the newest, with fibre cable running from the ISP directly to the homes with no coax cables in between. It is more expensive but way faster.
- Enterprise and universities use a large LAN to connect out, they have an ethernet switch to dorms and colleges, that goes out to the university ISP, like OARNET. However, wireless routers/switches also exist, using WiFi. These act almost like ethernet switches in an institutional setting.
- In mobile devices, cell towers act as packet switches for mobile data. 3G towers allow access at speeds of 1 Mbps, but 4G allows for 10!

1.2.2 Physical Media

- A single bit has to travel as an electromagnetic wave or optical pulse over an incredible amount of physical mediums. These mediums can be copper wire, coax cable, fiber optic cable, and radio waves. Guided media goes along a solid medium, like a coax cable, while unguided media goes over Wifi or sattelitte waves.
- Twisted pair copper wire today transmits at about 10 Gbps, and is primarily used for in building LANS.
- Coaxial Cables can produce incredibly high data transmission rates, and are a shared medium, meaning multiple end systems can be connected to one coax cable, and info can be shared between them.
- Optical fiber sends data using pulses of light instead of waves of magnetism. These are used for long distance data transmission, and are even buried under seas in pipes. These are expensive though, so we don't use them a lot in buildings for LANS. They can go up to 39.8 Gbps.
- Terrestrial Radio Channels carry signals that represent bits in the eM spectrum. These are used in WiFi, and mobile cell towers. However, they are vulnerable to interference.
- Satellite Radio Channels link two ground stations that transmit microwaves to satellites in space. There are two satellites used, geostationary satellites and Low Earth Orbiting Satellites. Geostationaries sit in the same spot 36,000 KM above the Earth, with 280 ms of delay in sending signals, but can operate at speeds of hundreds of Mbps. LEO satellites are closer to Earth, but rotate around it and communicate with other LEOs.

1.3.1 Packet Switching

- The **network core**, remember, is the mesh of packet switches and links that connect end systems within the internet. In networks, end sys exchange messages with each other. To send data that represents these messages, the data has to break down into small chunks of data called packets. Each packet will travel through packet switches, which are routers and link layer switches. This travel time takes the size of the packet divided by the transmission rate of the route in seconds.
- **Packet switches use store-and-foreward transmission at the inputs to the links. This means that switches need the entire packet before they send any of the packet out. Storing all the bits as they come is called buffering. The end to end delay of a single network with a path of N links each of rate R is N*(L/R), with L being the size of the packet. Remember that there should be N - 1 routers in the path. L/R is the transmission rate of pushing a packet into a single link.**
- The output buffer is where incoming packets must wait until it is their turn to be sent over a comm link. If a comm link has many packets on it, the packet in the output buffer must wait a long time. This is called a queueing delay, and if the buffer fills completely, packet loss occurs and a

packet is dropped, either in the buffer or the packet trying to get into the buffer.
- Every end sys on the internet has an IP address. This is used to determine in routers where to forward a packet once it comes out of the output buffer. A source end sys includes the destination IP address for a packet in the header of a packet. Every router has a forwarding table that maps destination addresses to the comm links the router has going out of it. The packet then is sent to this link. To make these tables, a special routing protocol is used to automatically set the tables, like using the shortest path of routers and comm links.

1.3.2 Circuit Switching

- Packet switching is not the only way to move data through a network. Circuit switching also exists. In a circuit switched network, all buffers, comm links, and rates are reserved for the entire duration of communication between two end systems. This differs from packet switching, which makes packets wait and share resources, making queueing for access to a comm link necessary.
- Circuit switching gaurentees a consistent rate of data transfer, which is its upswing, but its downswing is the need for reservation of many resources. These connections are called end to end connections.
- Circuits in links are implemented two different ways: Frequency Division Multiplexing or Time Division Multiplexing. In FDM, an end to end connection is placed on a specific frequency band, meaning that two end to end connections can exist at once, operating on alternate frequencies. The width of the band is called the bandwidth. A TDM end to end connection gives each connection a time slot, and carries bits in the connection during that time slot. So a slot of 8 bits run at 8,000 slots per second can transmit at 64kbps.
- Circuit switching leads to circuits that are idle during silent periods, meaning that when no data is transferred, the link is wasted on the end to end connection.
- Packet switching is more efficient, but less reliable. Due to packet loss, data can be lost, and this will not work for phone or video calls.

1.3.3 A Network of Networks

- The internet is created by a network of networks, thousands of linked ISPs.
- Regional ISPS connect cities and towns, which connect themselves to tier-1 ISPs, of which there are about a dozen scattered throughout the entire world. These tier 1 ISPs allow regional ISPS to connect to each other for a paid amount, and create the global link of the internet.
- A point of prescence is where customer ISPs can connect to provider ISPs. These PoPs are just groups of routers owned by the provider. Access ISPs (those that provide actual internet to people) can multihome, meaning they can connect to multiple regional and tier 1 ISPs. Two ISPs can also peer, meaning they set up direct connections between each other and don't pay for it. An internet exchange point (IXP) is where multiple ISPS peer together.
- Content provider networks are the last piece of the puzzle, which are servers peering all over the globe with IXPs, bypassing tier 1 ISPS and creating their own network.

1.4.1 Overview of Delay in Packet Switched Networks

- A packet can suffer from many types of delays in its path through the network core. The overally delay on a packet is called the total nodal delay.
- A processing delay is how long it takes to process the packets headers to ensure there are no bit level errors in the packet, as well as determining where to direct the packet based on table lookup. These usually only take microseconds
- A queueing delay is the delay a packet experiences as it waits to be sent into a comm link. This length depends on the amount of packets in the queue. These can take micro to milliseconds.
- A transmission delay is the amount of time it takes to push all of a packets bits into a comm link from a router. This takes L bits / R bits/sec comm link capacity.
- Propogation delay is the time required to get a packet from rotuer A to router B through a comm link. Bits can travel thru a comm link at c or almost c. The delay takes d/s, where d is the distance of the comm link and s is the speed at which bits travel.

1.4.2 Queuing Delay and Packet Loss

- Queuing delay is the most complicated/interesting component of nodal delay. It is described as traffic intensity, or La/R, where L is the size of a packet in bits, R is the transmission rate in bits./sec, and a is the average rate that packets arrive in the queue. If you have a small traffic intensity, queuing delay is small. If its greater than 1, than packet loss occurs, as the rate is large and it is very probable that a queue exists that is almost greater than the router can handle.
- Packet loss occurs when the traffic intensity > 1, or the queue is full. At this point a router will drop the packet and it will be deleted and lost.
- Processing delay is the time it takes for a router to determine where a packet should go based on its header data. It also checks for errors in the data as well. The time this takes can cause queuing delay, as other packets will have to wait at a router while it figures this stuff out.
- Transmission delay is L/R, or how long it takes the packets to travel into a comm link. Bandwidth makes this slow.
- Propagation delay is the time it takes for a packet to travel through a comm link. Distance makes this slow.
- End to end delay, or total nodal delay can be described as N(processing delay + transmission delay + propagation delay), where N is the number of routers assuming all routers are the same and the network is uncongested.

1.4.4 Throughput in Computer Networks

- Instantaneous throughput is the rate in bits/sec that a host is receiving a file. The average throughput is F bits / T seconds.
- To avoid packet loss, we have to send packets through at the transmission link of a bottleneck link, or the transmission rate of the slowest comm link in the network core between hosts A and B.

1.5.1 The Network Layers

- We split the entirety of the internet into layers to help us understand how it works. We describe each layer by the services it provides, called the service model of a layer. The protocols described in each layer make up the protocol stack, consisting of, from top to bottom, the Physical, Link, Network, Transport, and Application layers. The class will follow these top down.
- The application layer is where network apps live, as well as their protocols like HTTP, SMTP, FTP. DNS is also on the application layer. A message is defined as an exchange of packets from one host to another.
- The transport layer is the layer that handles the transport of application layer messages beween hosts. TCP and UDP reside here. A transport layer packet is called a segment. The transport layer provides the multiple hosts to communicate between to the network layer.
- The network layer is used for moving network layer packets called datagrams from one host to another. After being given the end systems to transport data to, the transport layer uses the IP protocol to send the datagrams specified.
- The link layer is responsible to transfer packets from one node or router to the next node in the path to the destination host. So a datagram is given to the link layer, which passes it to the next router, where it is then passed back to the network layer to determine the next node to go to. Protocols include ethernet, WiFi, and DOCSIS. Different protocols exist on different routers. Link layer packets are called frames.
- The physical layer is just responsible for transferring bits over a comm link.
- The OSI layers have session and presentation between transport and application. The presentation layer handles services that allow for interpretation of communicated data, like encrypted data or compressed data. The session layer provides for synchronization of data exchange.

CHAPTER 2:

- A socket is a network interface that sends messages into and gets messages from. It is like a door to a house.
- An IP address is a number assigned to each device that connects to the internet. It is 32 bits wide. A IPv6 address has 128 bits. It uniquely ids a host.
- A port number is the opening in the computer that an application process runs on that listens for internet data.

2.2 The World Wide Web and HTTP

- The HyperText Transfer Protocol is the web's application layer protocol, and is implemented in 2 programs, client and server. Clients and servers can talk to each other by exchanging HTTP messages.
- Browsers implment the client side of HTTP. Web servers implement the server side of HTTP obviously.
- HTTP determines how we request and send from a server. We usually request many types of data from a server called objects.
- HTTP runs on top of TCP. It only sees sockets, as it does not want to touch TCP. Thus, we get and send HTTP requests and responses through a socket, and they are handled by lower layers.

2.2.2 Non-Persistent and Persistent Connections

- We need to decide if the HTTP requests we make with a server are done over the same long TCP connection, or multiple small ones? The former is a persistent connection, and the latter is a non-persistent.
- In a non per connection, if we request something like a webpage with 10 images on it, we will have to open 11 TCP connections. These connections can run in parallel in most modern browsers, at about 5-10 TCP connections at the same time requesting and responding once each.
- The RTT (Round Trip Time) is the amount of time it takes for packet to travel from client to server to client. Thus, a HTTP request takes one RTT to establish a TCP connection, then another and then some to actually get the data it wants.
- By default, HTTP uses persistent connections, with pipelining, meaning we send multiple requests back to back without waiting for HTTP responses.

2.2.3 HTTP Message Format

- There are two types of HTTP messages, requests and responses.
- A HTTP request is written in normal ASCII text, the message consists of five lines, with the lines being:
    - 1: Request line, can have three fields:
        - The method field which can have GET, POST, HEAD, PUT and DELETE.
        - The URL field, the URL to request the data from.
        - The HTTP version field, obvious.
    - Header lines. Which can specify the host we are looking at (www.host.com), a line that indicates persistent or non connections, and a user agent that specifies the browser type or client type. More headers exist, like accept-language, which tells the language to use.
    - It can also have an additional message body, that contains the data to be sent when POST is called. We can also use data requested in the URL as parameters, and send a GET request. HEAD just gets a response with a requested object left out so we can debug the headers. The PUT method is used to upload an object to a specific path on a specific server. DELETE deletes an object on a server.
- A response message has the status line, the six header lines, then the message body.
    - The status line responds with 3 fields, the protocol version, the status code, and a status message. The protocol version is the same one used in the request, the status code indicates what the server was able to do with a request, and the message reflects the code.
    - The header lines have the date the response was sent, the server describes the web server

that responded, the last-modified indicates when the object was changed last, the content length indicates the byte size of the response message, and the content type indicates the type of data in the response message.
    - Codes that exist are 200 OK, 301 Moved Permanently with the new location in the Location header added on to this as a URL, 400 Bad Request we didn't understand your message, 404 not found, and 505 HTTP version not supported.

2.3 FTP

- FTP is used to transfer files from one host to and from another host. Authorization details must be provided to use this service. First, TCP gets the connection to the host, then authorization details are sent in another request.
- FTP uses 2 parallel connections to transmit files, control on port 21 and data on port 20. Control sends commands and authentication to the server, and data sends files. FTP is said to send control out of band, and HTTP is in band as it sends control requests in the same TCP connection as its data.
- FTP must keep state info about a user like their directory and account. HTTP is stateless, it doesn't track a user.
- Commands:
    - USER name - sends user id to server
    - PASS password - sends password to server
    - LIST - displays all files in current dir
    - RETR file - copies a file to the local host
    - STOR file - stores a file into the remote host

2.4 Electronic Mail in the Internet

- A user agent is a mail program like Outlook. A mail server is a server containing mailboxes belonging to specific accounts, and an agent gets and sends mail from and through this server. SMTP is the app layer protocol for sending mail over TCP.

2.4.1 SMTP

- SMTP transfers messages from senders mail servers to recipients mail servers. It only allows 7 bit ASCII mail messages, meaning all data must be converted to ASCII and back by user agents.
- Mail is sent by an agent sending mail to another email address, the message going into a queue in the mail server, and when it sends a TCP is opened between the two mail servers, and the message is put on the other mail server where it can be read by another user agent.
- The actual SMTP sending has the sending server open a connection with the getting server, specify the email address of the sender, then the getter, then indicates it will start sending data, which is then specified how to be sent by the getter server, the message is sent, and the connection is closed after the server validates the message got through. The server will also validate all addresses used.
- Mail servers do not run on user machines, because they cannot always be on. So they have to run on third party always on servers. So we have protocols to access these servers from user machines.
- POP3 allows the user to login to a server, and read messages on that server over a TCP connection as well as issue commands. You can then download and copy messages onto it.
- IMAP is more complex, with the same functionality, but maintaining user state on the machine using inboxes and the like.

2.5 DNS - Internet Directory Service (Domain Name Service)

- Internet hosts can be identified by hostname, like osu.edu. Hosts are also identified by IP addresses, made out of 16 bits (IPv4 - 4 bytes, IPv6 - 128 bits). An IP is hierarchial, the more detailed data is on the right side.
- DNS translates between IP addresses and host names. DNS servers host different parts of the

entire DNS database, and an app layer protocol allows access to these servers for host name IP lookup.

- A hostname can have aliases for easier memorization of the host name. The original is called the canonical hostname.
- A DNS server can bounce a user to one of many identical web servers that host a website, like google.com. Google.com is the alias for hundreds of web servers all processing data from users at once. DNS will choose a server that is not populated and send you to it, and you have no idea this even happens.
- DNS servers have to be distributed, otherwise they would die really quickly from hackers or traffic overload. So we have a hierarchy of domain name service servers:
    - The top level root DNS server will return a list of servers for the top level domains like com, org, edu and gov.
    - The TLD domains are then contacted for something like osu.edu or google.com. These are sent to the sites authoritative DNS server
    - The authoritative DNS server delivers the actual IP address, as each of these are one of the multiple web servers running a website.
- Every ISP has a local DNS server, or a default name server. This will handle all outgoing DNS requests for a host computer end system.
- DNS caching allows DNS servers to store the IP addresses of frequently requested host names. A local server can store lots of these for an ISP. It makes browsing faster.

2.6 P2P Apps

- Peers in P2P are actual end users on desktop machines. To download files, peers upload files to each other and download from each other in very small packages.
- The distribution time is the time it takes to get a file out to all people downloading from you. For a server, this takes the maximum of either (number of downloads * f bits in file)/the output speed of the server, OR the f bits / the slowest downloader from the server.
- For P2P, it is this:

ed by D_{P2P}.

- $$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}$$

- Where us is the upload speed of of the server that intially hosts the file, d min is the slowest speed of the downloaders, and ui is the individual upload speed of every peer, with N representing the number of peers and F being bit file size.
- More users means that P2P will download faster than client server.
- In BitTorrent, peers participate in a torrent, which is a single distribution of a file. Peers download equal size chunks of a file from each other, and send chunks to each other as well. A tracker is a node in the torrent that keeps track of everyone in it. This tracker will give users peers to connect to over TCP and share data with.
- To get chunks of data, users will ask their neighbors for lists of all the chunks they have. They will then try to download the chunk that shows up the least in the lists the neighbors return.
- To choose who to upload to, the user will choose the neighbors that are uploading to them at the highest rate. Every couple of seconds, the user will choose a new trading partner and start trading, and if the rate is higher, they will kill one of their other connections and keep the new one.
- A distributed hash table is a database that is stored over millions of peers, with each peer only holding onto a subset of the total data. A peer will query the database for a key, and then the database will find the peer with that key and return the key pair value to the querying peer.
- In a DHT, each peer gets an ID, an integer in the range 2^n-1. Each key will be an integer in the same range, and the integers will be made by hash functions taking in things like strings. We then distribute these keys to the peers with the sucessor IDs closest to the hashed key value.
- We organize peers into a circle, with each peer only knowing the identities of its predecessor and

successor. If we need to query for a value with key ("obbobob"), this would translate over a hash function to say 22. Our ID is 10. We would then query our neighbor, 12, to look ahead for the key value pair. When the message reaches peer ID 23, 23 will send 10 the data. In this manner, the number of peers / 2 messages are sent on average, with an N worst case. We can also have each peer keep info about peers farther ahead in the circle, to reduce message time.

CHAPTER 3:

3.1 Introduction and Transport Layer Services

- **Logical communication is the idea that from an applications perspective, the hosts running the processes are connected directly.**
- The transport layer provides the logical communication.
- Transport layer protocols are implemented in the end system, not routers.
- The TL will turn app layer messages into packets called segments, by breaking them into chunks and adding a header specific to the TL. After this, the segment is passed into the network layer.

3.1.1

- The network layer provides logical communications between hosts, and the transport layer provides logical communication between processes. In a postal service analogy, someone is responsible for getting mail ready to be sent to another house. The postal service is responsible for getting mail between two houses. So the transport layer has the job of getting things to the USPS, and the network layer has to deliver the mail.
- From this analogy, we can see that TL protocols live on end systems.

3.1.2

- UDP is an unreliable connectionless transport layer protocol. TCP is a reliable connection oriented protocol.
- For this chapter, we just have to know that each computer has an IP address, and that IP tries its hardest to deliver segments between two hosts, but it makes no promises.
- UDP/TCP extend IP, which delivers segments between two end systems to a service that delivers segments between two processes running on end systems. This is a process called transport layer multiplexing/demultiplexing.
- TCP/UDP also provide integrity checking by including error detection fields in the segments headers.
- UDP is like IP, it makes no promises on segment correctness. TCP is reliable and ensures data is transported between processes. It also provides congestion control, preventing swamping the network core with excessive traffic.

3.2 De/Multiplexing

- Remember that a process can have one or more sockets, dorrs which data passes through from the network to the process and vice versa. When the transport layer delivers segments, it does not delvier them directly to the process, but to the socket assigned to the process.
- **The job of delivering the data in a segment to the correct socket is called demultiplexing. This is done by examining a set of fields in the header of a segment.**
- **The job of creating segments grabbed from different sockets and giving each data chunk header info so it will be a segment is called multiplexing.**
- To actually do multiplexing, ports need to have IDs, and headers need to have fields in segments. These fields are the sourceport number field and the desitnation port number field. Each port # is 16 bits, with 0 to 1023 IDS being reserved as they are well known port numbers.
- When devving a new app, we gotta assign the app a port number.
- UDP has a pretty simple way to de/multiplex. A UDP app is assigned a port on both systems. In the segment headers for a UDP segment, both port numbers for the sender and the reciever are put

into the header. The segment is then sent by the network core to the reciever IP, and demultiplexes it by examining its destination port and sending it to the socket that handles it.

- A UDP segment is fully identified by a two-tuple of the desintation IP and Port number. We include the source port in the packet so we can send packets back.
- A TCP socket is identified by a 4 tuple, with source IP, desination IP, and source/destination port numbers. All four values will be used to demultiplex a segment.
- To demultiplex a segment, TCP will listen for connection establishing socket called a "welcoming socket" that listens on a specific port. Clients will send a TCP segment with a special bit to indicate it is a connection segment. The server process will accept the information, and create a new socket, identified by the 4 tuple for TCP identification.

3.3 Connectionless Transport: UDP

- UDP does about as little as a transport protocol can do.
- UDP has no handshaking between sending and getting transport segments, so it is called connectionless. It just assumes that the two hosts are there to communicate.
- UDP allows for finer app level control over what data is sent and when. As TCP has congestion control, which can throttle the sender in the transport layer if the network is congested, UDP can just be made to shoot data into an unopen network core. Thus, if some data can be lost like movies or music quality, we could use UDP to send it faster.
- UDP has no connection establishment, it can just blast its data between hosts.
- UDP has 8 bytes of header overhead in the segment. TCP has 20.

3.3.1 UDP Segment Structure / 3.3.2 UDP Checksum

- The segment structure of UDP is split into four parts:
    - Each field is two bytes.
    - The source port number has the port the segment was sent from.
    - The destination port number has the port we want to multiplex to.
    - The length has the length of the number of bytes in the UDP segment it is in, measuring both header and the application data.
    - The checksum is used by the getter to check if errors have occurred when sending the UDP segment.
- To calculate the UDP checksum, add all the 3 other segments together and take the 1's complement of it. This will be the checksum.
- To check for errors in the getter, add all the 2 byte words together, and if it adds up to 1111111111111111, you have a correct UDP segment.

3.4 Principles of Reliable Data Transfer

- A reliable data transfer protocol has the responsibility to abstract to upper level app services the function of creating a reliable channel that data can travel through.
- The level below the protocol makes this hard, as this level may be unreliable.
- In the diagrams for this chapter, rdtsend() sends data to another host. Rdtrcv() recieves data. Deliverdata() sends data to the upper layer from the transport layer.
- We only consider unidirectional data transfer, with one side sending and the other getting. In reality, both hosts will use udtsend() to send packets/segments to each other unreliably through the network layer.

3.4.1 Building a Reliable Data Transfer Protocol

- In the first protocol, we have the sender waiting from calls from the application layer that then make packets and send to a getter. The getter just waits to get packets from the sender, then sends them up to the application layer. This does not protect against corrupted bits in the packet.
- **ARQs are used to note whether a packet was gotten or not. An ARQ is a protocol called Automatic Repeat Request. It requires that we can detect errors in a packet, have the getter send feedback to the sender in the form of ACK (we got it) or NAK (it failed). These only have to**

be a bit long. It must also have a sender that can resend a packet.

- With this, rdt 2.0 senders will wait for the application layer for data, send the data and then wait for an ACK or NAK. It will then read the bit it gets, if NAK it will resend the previous packet, and if ACK it will wait for more data from the application layer. Getters will get packets, check them for errors, and send the correct message based on if there are errors or not. This is called a stop and wait protocol.
- RDT 2.0 looks good, but it falls apart if the ACK/NAK packet gets corrupted. To solve this, we add a sequence number into the headers of the packet. If the getter has already processed the number in the header, then it knows the sender is retransmitting.
- We have RDT 3.0 as well, which handles packet errors and packet loss. To handle packet loss, protocols usually have a sender wait for an ACK to be received, and if it isnt in time, it will resend the whole packet again. The wait has to be longer for the round trip time between the two hosts. Since we can use sequence numbers to recognize at a getter that a packet was aldready sent, we can handle duplicate packet transmission. So, after a countdown timer is set, the sender will just send another packet if it goes over that time.
- In RDT 3.0, the sender initially waits for the call 0 from above, which indicates sending has to be done. It will then send the packet with 0 in the header, and wait for an ACK with 0 in the header from the getter. This is where a TCP connection is established in theory. It then has a timer that waits for an ACK with header 0, if it times out it retransmits. Once it lets the app layer know it has made a connection and gotten an ACK from the getter, it will get data from the app layer and send it to the getter. It will then do the same thing it did waiting for the first ACK.
- Pipelined protocols for sending data are much better than stop and wait protocols.
- **Channel utilization is defined as (L/R) / (RTT + (L/R)) %. L is packet size in bits, R is transmission time in bits per second. RTT is the round trip time**

3.4.3 Go Back N (GBN)

- The sender is allowed to transmit multiple packets without waiting for an ACK, but is constrained to have no more than some max number N of unACK'd packets in the pipeline.
- In GBN we define base to be the sequence number of the oldest unACKd packet, and nextseqnum to be the smallest unused sequence number (the next packet to be sent). There are 4 intervals in the range of sequence numbers that can be IDd:
    - [0,base-1] are the packets that have already been transmitted and ACKd.
    - [base, nextseqnum-1] are the packets sent but not ACKd
    - [nextseqnum, base + N –1] are the packets that can be sent but have not been yet due to data not coming in from the upper layer.
    - [base + N, end index] are the packets that we cannot send even if we had them.
- N is referred to the window size, as it covers both the already sent and able to be sent packets:

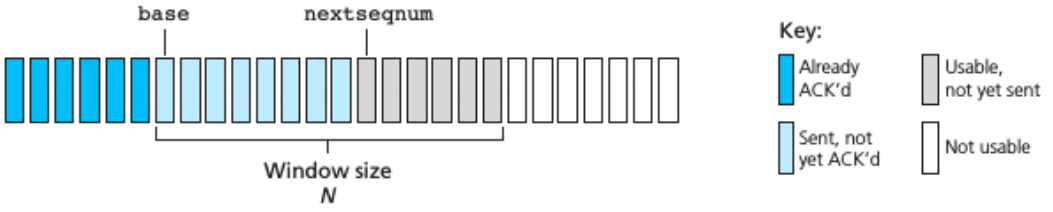


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

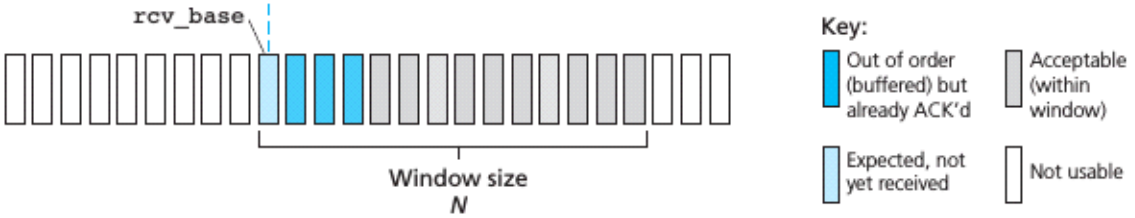
- All sequence numbers can be a size of 2^k –1, and when it hits 2^k-1, the next s number will have to be zero.
- As more and more packets are ACK'd, we can slide the window to the right to allow for more not yet sent packets to start being sent.
- A sender must respond to three events:
    - When rdtsend() is called from above, the sender checks if the window is full (N packets that

are unACKd). If it is, it tells the upper layer it cannot send, if it is not, it creates and sends a packet, and then updates the variables nextseqnum.
    - When an ACK is gotten, all packets with sequence number N (indicated in the ACK) and below will be removed from the window in a cumulative ACK. The base will then move to the most recent unACKd packet, and N will shift right.
    - A timeout event occurs. A timer is started on the oldest and unACKd packet, and if a timeout occurs on this, then all packets that have been already sent are sent again, and a timer is restarted for the sending.

- A getter is simple. It waits for packets, and upon packet n being gotten, it will send an ACK for packet n with n as the sequence number. Note that the packets have to be sent in order for this to occur. If we lose packet n, though, packets will appear out of order and we will toss any new packets until we get the one we want, n.

3.4.4 Selective Repeat

- GBN has a flaw, and that is a large window size and one error causes a lot of bandwidth to be used when sending packets.
- Selective repeat avoids uneeded transmissions of repeated packets, and only retransmits the packets it thinks were in error. This means the getter has to individually ACK correctly gotten packets.
- The getter now has a window as well:
-



-

b. Receiver view of sequence numbers

- The recv base is the first packet with the first seq num expected by the getter. The window holds all acceptable packet seq nums.
- For the sender, the window will now have ACK'd packets in the window, with base being the oldest UNACK'd packet and all other variables the same as GBN. The sender has three actions based on events:
    - Data is gotten from above – When this happens, the SR sender checks the next available sequence number for the packet, and sends it if its in its N window.
    - Timeout – Since EVERY packet now has a timer, on its timeout the packet will be resent.
    - If an ACK is gotten, the SR sender marks the packet as gotten. If the packet's seq num is equal to the base, the window moves forward.
- The getter has actions as well:
    - PAcket with seq num is gotten that is appropriate in the window – An ACK is sent to the sender with the seq number, and bufferred if not previously received. If the seq num is equal to the base, it, and the other consecutive packets in the window are sent to the application. The window then moves forward to the next ungotten acceptable packet.
    - Packet with seq num before window – An ACK is sent and the packet is discarded.
    - Anything else – packet discarded.

- SR senders and getters will not always be in sync, so we have to accept and resend packets that we don't really need to keep things running smooth on the getter side.

3.5 The TCP Connection

- TCP is called connection oriented because two processes must handshake with each other to establish the parameters of data transfer.

- TCP is a full duplex service. This means that app layer data can flow from process A to B as it flows from B to A simultaneously. It is also point to point, between exactly 1 sender and getter.
- The connection between two hosts is a three way handshake, where a client sends a no content packet to a server, the server responds with another one, and then the client can send a packet with data.
- TCP will take data from the socket between the app and transport layer, the data will be put in a TCP connection send buffer. From time to time, TCP will grab data from the buffer and send it through the network. The max amount of data that can be put in a segment is the max segment size (MSS), which is usually the length of the largest link layer frame that can be sent by the host (MTU). It will ensure that a header and an amount of data can be sent in one MTU, and set the MSS to this size.
- TCP segments are made with TCP headers and the sent data.

3.5.2 TCP Segment Structure

- The TCP segment consists of header and data fields. The data field is just a chunk of the app layer data. The TCP header is usually 20 bytes. The header conists of:
    - SOurce and destination port numbers, like UDP
    - The sequence number and ACK number, each 32 bits large.
    - The 16 bit receive window, used for flow control. Used to indicate the number of bytes a getter is willing to accept.
    - The 4 bit header length field specified the length of the TCP header in 32 bit words.
    - The opitonal options field, of variable bit length, is used when a sender and getter negotate the MSS.
    - The flag field has six bits:
        - The ACK bit is used to indicate the value in the ACK number field is correct.
        - The RST, SYN, FIN are used for connection setup and teardown.
        - The PSH bit indicated the getter should pass the data to the app layer immediately.
        - The URG bit indicates that the segment has data the app layer has marked as urgent. The location of the last byte of this urgent data is indicated by the 16 bit urgent data pointer field.

- The sequence number for a segment is the byte stream number for the first byte in a segment. For example, a file has 500,000 bytes. If the MSS is 1,000 bytes, 500 packets will be sent. The first 1000 bytes will have seq num 0 (actually a random number, so we don't use a byte from a previous connection still in the network) , and the next 1000 bytes gets seq num 1000.
- The ACK number for a segment is the seq num of the next byte Host A is expecting from Host B. If Host A gets 535 bytes from Host B, it sends 536 as its ACK num. As TCP only ACKS bytes up to the first missing byte in the stream, TCP provides cumulative acknowledgements, meaning that it wont ACK byte 1000 before ACKing byte 999.

3.5.3 Round Trip Time Estimation and Timeout

- TCP uses a timeout mechanism like our rdt protocol established earlier.
- For TCP, we use a sample RTT estimation. SampleRTT is the amount of time it takes between a segment is passed to the IP protocol and the ACK to be received by the sender.
- EstimatedRTT, the amount of time the protocol thinks it will take for a RTT is calculated as:
- $$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$
- Where alpha is a weight, typically 0.125. This provides an average RTT time that we have estimated and standardized. We put more weight on new samples as well.
- DevRTT is a measure of the variability of the RTT:
- $$\text{DevRTT} = (1 - \beta) \cdot \text{DevRTT} + \beta \cdot |\text{SampleRTT} - \text{EstimatedRTT}|$$

- The reccomended value of beta is 0.25.
- Using the above information, the timeout interval to be used is 4*DevRTT + EstimatedRTT. As estimatedRTT is our average RTT, and DevRTT is one standard deviation, we allow for FOUR times the deviation from the mean.
- An initial timeout interval of 1 second is recommended. This is changed every time we receive an ACK on the senders side.

3.5.4 Reliable Data Transfer

- TCP creates a reliable data transfer service on top of IP's best effort service. This means the byte stream a process gets out of its getter buffer is exactly the same as the one that was sent by the other end system.
- Our simple TCP sender will loop forever and respond to three events:
    - When data is grabbed from the above app, we create a TCP segment with seq num equal to the first data byte in the segments byte stream number. We then start the timer if it is not running, send the segment, and add the length of the data to the value of the next seq number.
    - When the timer timesout, we retransmit the non ACKd segment with the smallest sequence number and start the timer again.
    - When we get an ACK, we check to see if it is the base of our window, if it is, we make the new base the ACK field of the segment gotten from the getter (and ACKS all segments before the ACK field), and then start the timer if any other segments have not been ACKd.
- Each time TCP retransmits, it doubles the next timeout interval. But when the timer is started in the other two events, it gets the timeout from Estimated and DevRTT. This allows for limited congestion control. If everything kept hammering the network core, we would congest more routers, but larger timeouts allow for longer waiting between sending segments.
- TCP senders can detect packet loss before timeout occurs, by noting duplicate ACKs, which is an ACK that reACKS a segment for which the sender has already gotten an earlier ACK. The receiver can send four types of actions:

Event	TCP Receiver Action
Arrival of in-order segment with expected sequence number. All data up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 msec for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK transmission.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. Gap detected.	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Table 3.2 ♦ TCP ACK Generation Recommendation [RFC 5681]

- Since TCP does not use NAKS, so it has to send an ACK for the last in order byte of data it got when it gets an out of order byte. If a sender gets 3 dupe ACKS, it will retransmit the segment before the its timeout runs out in a fast retransmit.
- TCP is basically a hybrid of GBN and SR protocols.

3.5.5 Flow Control

- TCP provides a flow control service so that a sender will not overflow a getters get buffer, and it sends as much data as the getter is reading.
- A sender maintains a receive window, which is used to give the sender an idea of how much free buffer the getter has.
- The amount of space in the buffer is denoted by

- `rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]`
- Where rcvbuffer is the size of the buffer, and lastbytercvd is the last byte taken in from the connection and last byte read is the last byte read by the app process. In each segment sent, the getter will tell the sender the size of rwnd in the receive window field. The sender will not let the amount of unACKd bytes it has, and make sure it is less than the latest value it has of rwnd.
- If rwnd is ever 0, the sender must continue to send segments with one byte of data to the getter.

3.5.6 TCP Connection Management

- To establish a TCP connection, a client will:
    - Send a special TCP segment to the server side TCP. The segment has a SYN bit set to 1 and has a random initial sequence number.
    - The receiver then sends a segment with a 1 SYN bit, the acknowledgement field is set to the client sequence number + 1, and puts its own random sequence number in the header. This is called a SYNACK segment.
    - The sender allocates buffers and variables upon getting the SYNACK, and sends a segment with a SYN 0 and can have data in the payload.
- When a TCP connection is closed, the client TCP will send a special TCP segment to the server, with FIN bit 1. The server will respond with the same
- TCP connections have many states:

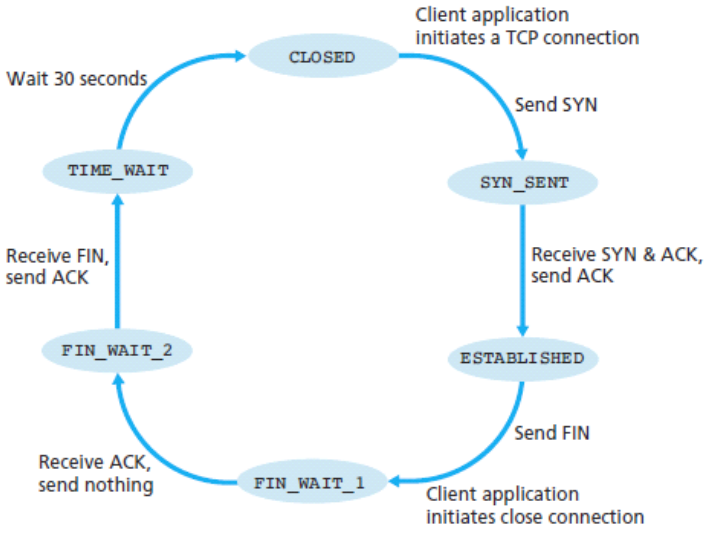


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

-

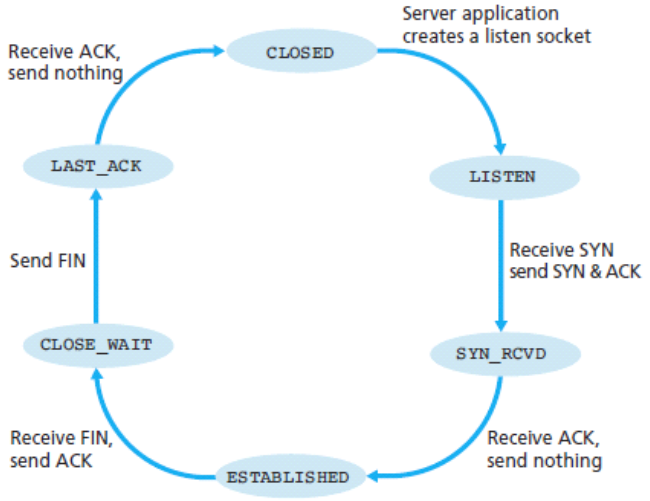


Figure 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP

3.7 TCP Congestion Control

- TCP will have each sender limit the rate at which it sends traffic into its connection as a function of percieved network congestion. If there is little congestion, it will increase send rate, if there is large, it will reduce the rate.
- TCP has something called a congestion window, cwnd, imposes a constraint on the rate at which a TCP sender can send traffic. Therefore, we make sure that the total amount of unACKd bytes is less than the smaller value of the receive window or the congestion window.
- An indication of congestion in the network is that a timeout occurs or we get three dupe ACKs. When this happens, we could limit send rate.
- If everything goes well, we could also always increase send rate.
- To define the rate, TCP uses the following principles:
    - Upon lost segment, the TCP send rate is decreased.
    - Upon ACKd segments, the rate will be increased.
    - Bandwidth probing is done by the combination of the previous two principles. If we increase, we will increase until we start losing packets, in which we will back off.

- The TCP congestion control algorithm was made in 1988, and has three components:
    - Slow Start
    - Congestion Avoidance
    - Fast Recovery

Slow Start

- CWND is initialized usually to 1 MSS (max segment size) with a sending rate of MSS/RTT. In slow start, the rate is increased by 1 MSS every time a segment is ACKd. Since we would start sending multiple segments at once, the sending grows slow but exponentially, as the CWND is basically allowing double the amount of segments to be sent for every ACK.
- We back off when a loss event is encountered that is indicated by a timeout, resetting the cwnd to 1 and retransmit the segment. Ssthresh is another variable set during this, and it is made cwnd/2. If we hit ssthresh with cwnd again, we transition into congestion avoidance mode. We also back off if we get 3 ACKS, where we transition into fast recovery mode and retransmit the missing segment.

Congestion Avoidance

- Cwnd will be half the size of what is was when congestion was encountered. So, upon every new ACK, the cwnd is set to cwnd + MSS(MSS/cwnd). When all segments are received, the cwnd should be increased by 1 MSS.
- This will end when we reach a timeout, and ssthresh is set to cwnd/2 again, with cwdn then being set to 1, and slow start is entered. It will also end when we get 3 dupe ACKS, where we set the ssthresh to cwnd/2 again, set cwnd to ssthresh + 3*MSS, and transition into fast recovery and retransmit the missing segment..

Fast Recovery

- In this method, the value of cwnd is increased by 1 MSS for every duplicate ACK gotten for the missing segment that caused TCP to enter this state.
- If a timeout occurs, we transition to the slow start method and cut our ssthresh in 2, and set cwnd to 1. If we get a new ACK, we transition back into congestion avoidance mode and set the cwnd to ssthresh.
- TCP Tahoe unconditionally cut the cwnd to 1 MSS and entered slow start after either loss event. Reno incorporates fast recovery.

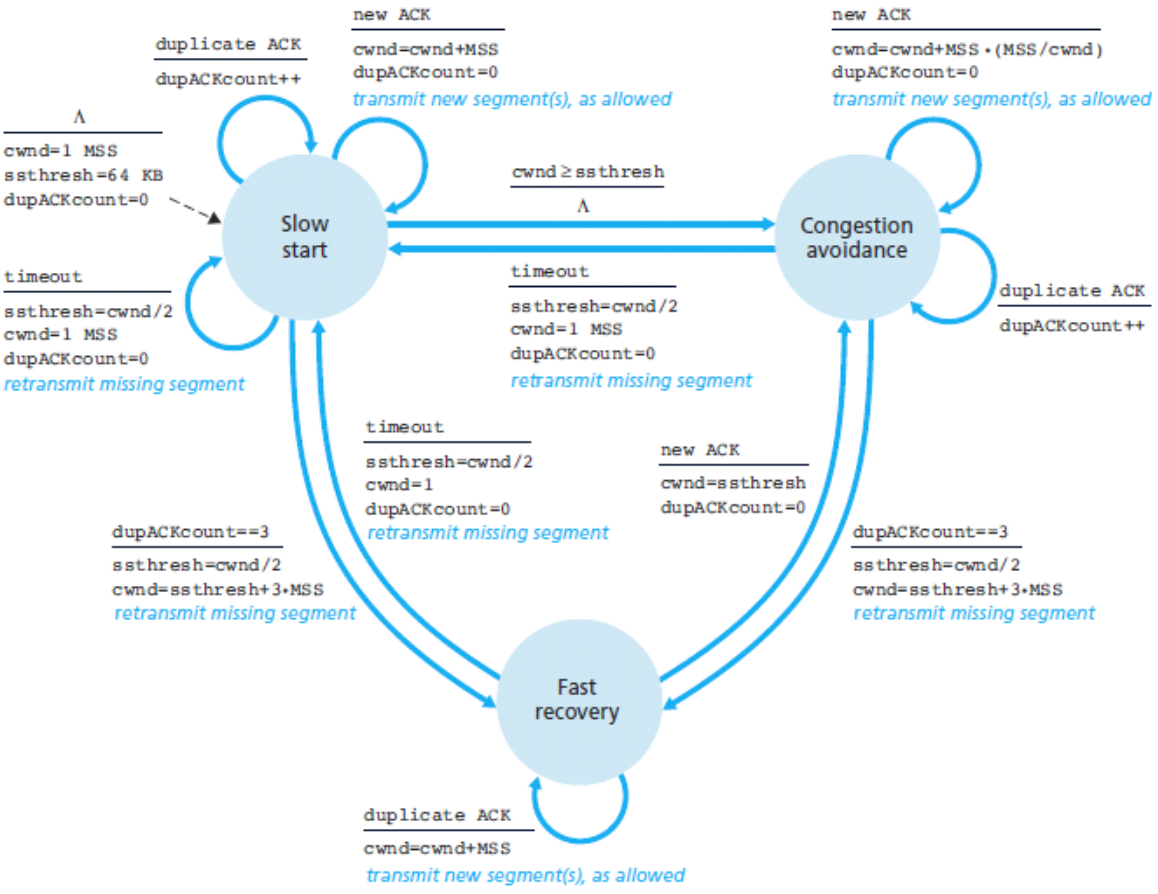


Figure 3.52 ♦ FSM description of TCP congestion control

-

$$\text{average throughput of a connection} = \frac{0.75 \cdot W}{RTT}$$

-

Where W is the value of the size of the window for sending segments, as during slow start and congestion control, we send more packets by allowing a larger window size in TCP on the sender.