CS 3516 Computer Networks Notes

Christopher Myers

June 13, 2020

Contents

1 Network Basics

Networks are formed from end points, various nodes, and the links between them. End points are called *hosts*, a type of node that form the "leaves" on a hierarchical view of a network, nodes are network devices such as switches or routers, and the links between them are physical modes of communication, be it by cable or by wireless.

1.1 Transmission Factors

Information is sent across networks in packets, segments of information of size L bits, using a network transmission rate of R bits/second, otherwise known as bandwidth. If you know the values R and L, you can estimate the packet transmission delay using the formula $\frac{R}{L}$. For example, transmitting a packet 7.5 Mbits in size across a link with speed of 1.5 Mbps would take $\frac{7.5}{1.5} = 5$ seconds.

This isn't a perfect value, however, because there are various factors on each node that may affect the time it takes for a packet to be retransmitted. For instance, packets must fully arrive at a node before the node can transmit them onto the next link (store and forward). There is also more than one node on most networks, so the link speed between each node along the route must be considered. Furthermore, this only works for communication in one direction; bidirectional communication requires that the time be *doubled*.

Other factors in network communication are:

- **Queuing** – packets will queue at a node in case they cannot be immediately forwarded.
- **Loss** – routers have finite memory, so they can't store an infinite amount of packets. If a router cannot store another packet on its queue, the packet will be dropped. If the packet is corrupted, it will also be dropped.

1.2 Routing

If a router has multiple links, it must decide which link to send packets out along by checking against its routing table. This is called *packet switching*.

Circuit Switching Circuit switching is a method that guarantees a link (reserved resources), but isn't used on computer networks (?), instead finding use on things such as phone lines. If multiple users need to use a circuit (line), either the bandwidth can be divided (FDM), or users can be allocated small timeslots that are rapidly rotated, providing the illusion of continuity.

Circuit switching drastically reduces congestion and lessens the need for protocols to ensure reliable data transfer. However, it isn't well suited for burst data, it isn't as easy to set up, and it doesn't allow for the same level of resource sharing.

Suppose there are an arbitrary number N users that must share a 1 Mbps link, but they each use their connection only 25% of the time. If circuit switching is used, only four users can use the line, as circuit switching preallocates resources. If packet switching is used instead, more users can share the line – the probability of 5 users using the line at the same time is less than 0.001, meaning that queuing delay or packet loss is unlikely.

1.3 Structure of the Internet

Access networks are networks that provide internet access; in order to ensure that a packet can be delivered between any two access networks, they must all have a path to one another, no matter how direct or indirect. In reality, much of the higher level structure of the internet has been driven by economics and politics more than technical decisions. Regardless, there are several possibilities:

- **Mesh network** – directly connect every access network to every other access network. The number of links increases with the square of then number of networks, so this doesn't scale well.
- **Centralized ISP** – connect every access network to a central ISP. This results in a very centralized system with effectively a single point of failure

- **Distributed ISP** – use multiple ISPs with each ISP having only a subset of access networks directly connected. Form peering links between ISPs to allow networks to connect to networks that belong to other ISPs.

The internet as it is today currently fits best into the "distributed ISP" model.

1.3.1 Access Network

Access networks are the lowest tier of network available (?). In residential terms, they're the same as someone's home network (???), with consumer devices like computers, phones, and smart [dumb] devices. In an enterprise context, an access network would be a much larger network used for day-to-day operations.

1.3.2 Regional Network

Regional networks are sometimes established by ISPs as an additional layer of the hierarchy; they form a small subset of the ISPs subscriber base, and usually join access networks onto one regional network based on location.

1.3.3 Internet Service Provider (ISP)

Internet service providers connect multiple access networks together, receiving payment for their service. In order to ensure that users connected to different ISPs can still contact each other, peering links are established amongst themselves.

1.3.4 Internet Exchange Point (IXP)

IXPs are used to streamline peering links and reduce the complexity of related systems. Instead of having each ISP connect to each other ISP, multiple ISPs can connect to one IXP, reducing the number of peering links required.

1.3.5 Content Provider

Content providers are entities such as Google that host their own content which others may access. When content is distributed on such a large scale, the physical location of that data is often distributed too, by means of a kind of caching (?). Load is distributed across various servers, and users get faster access to the content they need. Often this involves large data centers, the most extensive of which have their own dedicated links between one another. For instance, Google can build dedicated links between its data centers, links that are not available for direct use by the rest of the internet.

1.3.6 Point of Presence (POP)

POPs are essentially edge networks of a tier 1 ISP. For example, Sprint is a Tier 1 ISP, but if they want to deliver service of their own to an area, they must set up a POP.

Often, ISPs will multihome, or connect to multiple other ISPs that are higher in the hierarchy, for various reasons like speed, redundancy, or business economics.

2 Socket Programming

Sockets are a kind of "door" between two application processes that are often on different computers. Sockets are ultimately handled by the operating system, and usually run one of two different protocols, UDP or TCP. UDP, or **U**ser **D**atagram **P**rotocol, is a fast but unreliable protocol that sends discrete messages at the application layer, while TCP, or **T**ransmission **C**ontrol **P**rotocol, is a slower but more reliable protocol that allows for a "stream" to be opened between two computers, in such a way that the user can interact with the socket using some of the same functions that files can be interacted with. Both of these protocols are

handled by the operating system, unlike application layer protocols like HTTP or FTP. Applications may use as many sockets of as many different types as they need.

2.1 TCP Socket Calls

The following calls will set up a TCP connection:

- socket() — open a socket
- bind() — bind to an address, usually server-side.
- listen() — listen for an incoming connection, server-side.
- accept() — accept the first available connection.
- connect() — connect to a server, usually client-side.
- read() — generic I/O function, will allow reading from a socket or file descriptor.
- write() — generic I/O function, will allow writing to a socket or file descriptor.
- close() — used by either side, terminates the connection.

Further details of these functions' arguments are not provided here; please see their manual pages ("man connect", for instance) for usage information.

3 Network Performance

Various delays have a large effect on network speed and overall performance:

- **Processing delay** — When a packet is received, the receiving node must process the packet to determine where it should go, to verify its integrity, decrement its TTL, and so on.
- **Queuing Delay** — Discussed above; the amount of time it takes until a packet exits the transmission queue. As the ratio $\frac{La}{R}$ (where L is the packet size, R is the speed of the link, and a is the number of packets received per second) approaches 1, the size of the packet queue (and thus the queuing delay) will approach infinity. Since this can't actually happen in reality, real systems just drop packets. This ratio is known as the *traffic intensity*.
- **Transmission delay** — how long it takes for the node to transmit the packet to the link.
- **Propagation delay** – how long it takes for the packet to physically travel along the link. Usually this has to do with geographical distance and the speed of light, but more exotic forms of propagation delay are known to exist.

Note that in most cases, the transmission delay is much larger than the propagation delay. In such cases, there is little chance that there are two packets on the same link (for more on this, calculate the propagation constant β by dividing the propagation delay by the transmission delay. In some extreme cases (undersea links, interplanetary communications, IP over carrier pigeon, etc.) there can be multiple packets on the link at one time.

Nodal delay is the delay caused by crossing one node (minus things like processing delay and queuing delay), and can be calculated as $d_{nodal} = d_{trans} + d_{prop}$. The total end-to-end delay is just the sum of nodal delays along the route that a packet takes. For instance, in a system with 3 links that you send 10 packets across, you might calculate the end-to-end delay as $D_{end-to-end} = 3 * d_{nodal} + 9 * d_{nodal}$.

3.1 Throughput

Throughput is the rate (in units of bits/time) at which bits are transmitted between sender and receiver. There are two ways of measuring it: *instantaneous* throughput is the rate at a single point in time, while *average* throughput is the rate averaged over a longer period of time.

Generally, the throughput is limited by the slowest link in the route, i.e. a bottleneck. If you have multiple transmission paths that do not overlap, each with its own transmission rate, the throughput is then equal to the fastest path. This gets still more complicated when you try to route multiple low-throughput connections (limited by bottlenecks at the end of the route) through one high-capacity link. For instance, if you have ten clients, each using 1 Mbps, connecting each to a server with a 2 Mbps link, but all connections go through a single 5 Mbps link in the middle, each client will get a throughput of 500 Kbps.

4 Protocols

Modern networking runs on something of a protocol stack, where each protocol more or less relies on the others, but maintains its independence. These protocols are divided into different layers, designed to do different things. There are arguments for and against layering: on one hand, layered systems are much more abstracted than unlayered systems, so users at higher levels don't need to worry about what the lower levels are doing. It's also easy to change the stack without affecting other layers of the stack, for example by changing the link layer out from ethernet to wireless. However, some argue that layering results in duplicated functionality at different layers, such as error checking. Also, functionality at one layer often needs information from other layers.

The internet protocol stack, in order from highest to lowest layers, looks something like this:

- Application – how applications themselves communicate, e.g. HTTP, FTP, or SSH.
- Transport – processing of data transfer, usually either TCP or UDP. This layer is often involved in making sure data gets from one point to another.
- Network – how data is routed around and delivered to its destination
- Link – the protocol used across individual links, like ethernet or 802.11 wireless.
- Physical – how data physically moves around – "bits on a wire", for instance. This can mean different types of cables, different types of radio signals, or in the case of RFC 1149, birds carring scrolls with packets written on them.

At the application layer, applications have different requirements for characteristics of the connection. For instance, file transfers and HTTP require 100% reliable data transfer, while streaming services can usually tolerate some loss. There are similar requirements on delay — HTTP doesn't care much about delay (unless you set something like an Expires header), but a 2500 ms delay on a video call is unacceptable. For transport layer purposes, applications that require reliability but disregard delay usually use TCP, while applications that can tolerate loss but require high speed usually use UDP.

4.1 Application Layer

4.1.1 HTTP

HTTP, or **H**yper**t**ext **T**ransfer **P**rotocol, is an application layer protocol used by web clients and servers to exchange information across the internet. HTTP is a stateless protocol (even though it is usually run over TCP), so an HTTP server need not keep track of its status with individual clients. This greatly simplifies both the server and client.

When clients use HTTP, they initiate a TCP connection across port 80, which the server accepts. HTTP messages are then exchanged between the client (usually a web browser) and the server (software like Apache or NGINX). Once the exchange is concluded, the connection is closed.

The actual HTTP messages exchanged depend on which end is communicating. All HTTP messages begin with header composed of a number of lines following a standardized format; these tell the server/client

information about who's doing the communication, what sort of data is being requested, what cookies to set/retrieve, etc. Clients will send a request encoded by a method (e.g. GET or POST), while servers respond with a status code (e.g. 200 OK, 404 Not Found) and the data requested, if applicable.

HTTP allows the server to set "cookies" on the client's computer, which are usually values stored in a file somewhere by the browser. The client can then provide that cookie value to the server, which can use that information to maintain a degree of state (this is needed since HTTP on its own is a stateless protocol). Using cookies, servers can do anything from allow users to stay logged in across different interactive pages, to setting tracking cookies that let servers track users across the internet.

4.1.2 DNS

DNS, or **D**omain **N**ame **S**ervice is used to resolve human-readable names — like "www.google.com" or "sathanas.dyn.wpi.edu" — into an IP address that a client can connect to. DNS is an application-layer protocol run over UDP, and is implemented in a hierarchy of many servers called *name servers*. These name servers can either provide the client with the information it wants or forward it on to a server than can — hence the hierarchical part. DNS is regarded as a core internet protocol.

At the top of the DNS hierarchy are the top-level domains, like .com, .net, or .org. Queries are sent to the root DNS servers (if applicable), which then forward the client onto one of these top level domain servers. Those servers in turn forward the client onto whichever server might be able to further resolve that request, and so on. For example, if you try to resolve www.google.com, you will first be told by the root-level servers to check with the com servers. The com servers will forward you onto the google servers, which will in turn finally tell you the IP address of some machine named "www" (?), which you will finally be able to connect to.

In reality, requests don't often happen in this exact way. There is a *lot* of caching involved at every level. ISPs, for instance, usually provide their own local DNS server (sometimes called a "default name server"), which has a local cache of recent requests and the results of those requests. This local server also acts as a proxy when needed, forwarding requests into the hierarchy as described above.

There are different types of records, stored in a DNS record following the format of name, value, type, ttl:

- A — Name is hostname, value is IP address
- NS — Name service. Name is domain, value is the hostname of the authoritative name server for this domain.
- CNAME — Name is an alias for some "canonical" (real) name, value is the actual canonical name. For example, www.ibm.com is actually servereast.backup2.ibm.com.
- MX — Value is the name of a mail server associated with the name field.

Only an A record gives you an actual IP address. There is also an AAAA record, for IPv6 addresses.

DNS packets usually contain a 16 bit number ID (the response uses the same ID), data fields about how many data fields there are, any any number of questions or responses.

4.1.3 SMTP

SMTP is **S**imple **M**ail **T**ransfer **P**rotocol, and is used to deliver emails. There are three major components to mail systems: user agents, mail servers, and SMTP itself. The user agent, also known as a "mail reader", is responsible for the composition, editing, and reading of mail messages that a user sends or receives. For example, Thunderbird is a user agent made by Mozilla. User agents get incoming mail and send outgoing mail to the mail server. Mail servers contain a mailbox for incoming messages — the inbox — and maintain a message queue for messages waiting to be sent. SMTP is used between mail servers to send messages between them.

The process works something like this: a user composes a message using a user agent, which sends the message onto a mail server. The message is then placed into a message queue until transmission. The mail server will use SMTP to transmit the message to an appropriate mail server, at which point the message

pops up in the appropriate inbox. The receiving user then uses their user agent to check the inbox, and is able to download and view the message.

SMTP itself runs over TCP on port 25, and operates directly between mail servers (there are no intermediary servers). In comparison to HTTP:

- SMTP uses persistent connections (compared to HTTP's option to use persistent or non-persistent connections)
- SMTP requires to the message, both header and body, to be in 7-bit ASCII, whereas HTTP allows for binary code in the message.
- SMTP ends messages using a CRLF, a period, and another CRLF
- HTTP runs on a "pull" model (it is up to the client to request data), but SMTP runs on a "push" model (mail servers push mail to one another, instead of mail servers querying one another)

4.1.4 P2P Protocols

P2P is a class/"genre" of protocols, there are many different implementations in existence. Only the base concepts of P2P protocols will be discussed here.

P2P ("peer-to-peer") protocols are protocols that operate without the same notion of a server — every client is also a server, and has to send data as well as receive. For instance, if you download a file over a P2P network, your client receives data from any number of other clients on the network, but it also is responsible for sharing some of the data it already has. In this way, users can share files across the system in a decentralized way that scales extremely well. The standard client-server model has the limitation that download speed slows as the number of users increases, but for P2P the opposite is true.

The main challenge is to actually implement the distributed network — how do clients figure out which other clients to contact? Who keeps track of where files are? There are two ways of doing this, one a centralized method using a server (called a tracker, in most cases), the other using some distributed magic called a distributed hash table (DHT). Using either method, clients can figure out which clients they should contact if they need data for some file they're downloading.

Napster A notable early implementation of this technology is the now-ancient program "Napster". Napster allowed users to share music files by downloading the program and registering with Napster's servers. Those same servers would keep track of what files the user could provide, so if another user wanted those files, the Napster servers could tell the client to contact the client with those files. As it turns out, the music industry wasn't too fond of this scheme, so in 2001 Napster ended up shutting down its entire network in order to comply with an injunction.

Gnutella After the collapse of Napster, numerous other services (like Gnutella [not associated with GNU] or Limewire) emerged, and in 2001 the protocol was enhanced to allow for "ultrapeers", moving the P2P model away from a centralized model. This process involved query flooding — contact a few nodes to become neighbors in order to join, search by asking your neighbors (who might in turn ask their neighbors), and download by getting a file directly from another node. Note that the links on such networks are not actually physical links, they're just logical (i.e. software only) links between peers. Physically, two neighboring notes could be located anywhere in the world.

Unlike Napster, Gnutella had the advantage of being fully decentralized, but at a number of costs. The search scope might end up being quite large, resulting in high overhead and long search times. Worse, nodes come and go often, making for a significant logistics and overhead challenge.

BitTorrent BitTorrent is currently a very popular P2P networking system that solves many of the issues of its predecessors. BitTorrent works by dividing a large file into many small size pieces (256 KB) that are then individually replicated across different peers. A peer can share pieces of a file that it has even if it doesn't yet have the entire file. Here are the steps it takes to download a torrent:

1. Download a .torrent file from a web server somewhere This .torrent file contains information about the file(s) to be downloaded a tracker to contact about them. Note that many places have since switched to using "magnet links", which do not require downloading this file.

2. The peer (called a leech until they start seeding) contacts the tracker and sends a request for information about other peers, which the tracker provides. The peer then performs a handshake process with the peers it decides to connect to (called seeders), and retrieves pieces from them over time. Note that the rarest pieces are always downloaded first.

3. During the download, the peer will usually share pieces it has downloaded with other peers on the network, allowing the file(s) to continue to propagate.

4. The peer will occasionally (usually on some set interval) contact the tracker again, both to provide information about itself and to update its own cache of information about other peers.

This mechanism solves the scalability problem by providing different torrent files that correspond to different actual files. When peers provide chunks, they usually do it on a basis of whoever is sending them chunks at a highest rate; peers below some predefined limit are choked off, although there is an "optimistic unchoking" process to allow for new peers to join the swarm and contribute. This motivates peers to contribute to the network.

Unfortunately, this process still involves some centralized server, the tracker. This is why the distributed hash table (DHT) was created. Consider a traditional database: a series of keys that correspond to some value. As it turns out, it's much more convenient/fast to search a table using the *hash* of the key instead, so a hash table would store the original key, the hash of the key, and finally the value associated with the key. The distributed hash table makes this even more complex. These key-value pairs are distributed over potentially millions of peers, in such a way that any peer can query the database if it has an appropriate key.

In practice, since we need to know which peers to contact in order to get the values needed, some queries are forwarded on as needed, and key-value pairs must be assigned to peers in some logical way. This is done by assigning key-value pairs to the peer that has the closest ID, which by convention is the immediate successor of the key (keys are stored in some circular structure). For example, if you have a key space of 0 to 63 and peers with IDs of 1, 12, 13, 25, 32, 40, 48, and 60, key #53 would be assigned to peer #60.

To resolve this query, the peer will first contact its immediate successor. If that successor cannot resolve the query, it will contact its successor, and so on until the query is resolved (at which point the final peer directly contacts the original sender). For example, if peer 12 wants to know the value associated with key 53, it will ask peer 13. Peer 13 will ask peer 25, and the process is repeated until it finally reaches peer 60. In order to optimize this process, peers also maintain a cache of IP addresses for its predecessor, successor, and various shortcuts that it might find.

The problem with distributed hash tables is "peer churn", or that peers can come and go at any moment. Each peer periodically pings its predecessor and successor to check whether they are still there; if the immediate successor leaves, it chooses the next most immediate successor as a replacement.

4.2 Transport Layer

Unlike the application layer, the transport layer (which operates one layer below the application layer) is concerned with getting data from point A to point B across any arbitrary network. This need arises from the fact that IP routing is not guaranteed or even completely optimal. Transport across networks is actually done on a best-effort basis, meaning that packets can get delayed, dropped, or received out of order. If the client is expecting a stream of data (like what happens when you open a SOCK_STREAM-style socket), this kind of data loss or corruption is unacceptable.

Hence, we need transport layer protocols. There are different types of transport layer protocols for the different needs of applications. TCP, for example, is completely reliable and guarantees that data will arrive complete and in-order, but it's also much slower and has plenty of overhead. UDP, on the other hand, is very fast and efficient, but it's unreliable as it doesn't guarantee that data arrives in order, or that it will even arrive at all (although it does guarantee that data won't be corrupted if/when it does arrive).

Transport layer protocols are also involved in multiplexing & demultiplexing, often using port numbers to keep track of what data belongs to which application. In UDP, for example, connectionless multiplexing is used, so the operating system will direct packets to the correct socket based on the port number. For example, if two applications each have a socket open, one for port 1701 and one for port 1702, and the operating system receives a packet with port number 1701, the first application (running on port 1701) will receive the data and the second application won't.

4.2.1 UDP

UDP, or **U**ser **D**atagram **P**rotocol is a simple, connectionless protocol that only provides a best-effort service. It has an 8 byte header, containing the source port, the destination port, the packet length, and the checksum. After the header is just the user data; the packet length is the sum of the header size and the data size (that is, if you were to send 8 bytes of data, the length field would read as 16). Note that if the checksum field doesn't match the computed checksum, the packet will simply be discarded. Also know that it is possible for the packet to be corrupted but still have the same checksum, although it is unlikely.

UDP is obviously unreliable, but has advantages. UDP is completely connectionless, so there's no delay involved in establishing connections. It's very simple, involving no connection state at the either the sender's end or the receiver's end, and there is no congestion control. Technically, there's nothing stopping the user from sending as many packets as they can push out onto the network.

4.2.2 TCP

TCP is a streaming protocol designed with reliability in mind, as it guarantees that data will arrive at its destination completely intact and in order. Unlike UDP, however, this means that TCP can't offer nearly the same speed or fire-and-forget aspect that UDP does. TCP is a stateful protocol with a great deal of overhead involved (both for opening/closing connections and for transferring the data itself), so its best use is for cases where data must be reliably transmitted.

TCP packets contain a number of fields:

- Source port number
- Destination port number
- Sequence number
- Acknowledgement number
- Flags
- Receive window (number of bytes that receiver is willing to accept, used for flow control).
- Checksum
- Urgent data pointer
- Options (variable length field)
- Application data (variable length field)

Sequence numbers are assigned to packets so that they can be used in the corresponding ACK (acknowledgement) packet, which should have the ACK flag turned on.

TCP uses a 3-way handshake to initiate connections. When the client connects to a server, it chooses an initial sequence number and sends it in a packet with the SYN (synchronize) flag set. When the server receives the packet, it too will generate a sequence number and return it in a packet with both the SYN and ACK flags set (so this packet acknowledges the previous packet by setting its acknowledgement number to 1 plus the client's sequence number). Finally, the client will respond with an ACK packet with the acknowledgement number set to the server's sequence number plus 1.

From here on out, TCP operates by sending data in packets which are then acknowledged, using similar mechanisms described above (just without the SYN flag set). In the event of packet loss, TCP uses an

estimate of the RTT to determine when to resend packets; this estimate is an exponential weighted moving average, so it is usually resistant to rapid fluctuations. A safety margin is also used, based on the deviation of the RTT over time. The final timeout interval is defined as the estimated RTT plus four times the safety margin. In order to implement cumulative ACKs, TCP calculates sequence numbers for future packets by adding the current sequence number to the length of the data being transmitted.

4.3 Network Layer

The network layer is responsible for the forwarding and routing of packets, all the way from source to destination. This is done using a forwarding table that the router checks against. The routing algorithm itself is responsible for the end-to-end path along the network, while the forwarding table dictates how individual routers along the network treat different packets.

The forwarding table has two parts: the destination IP address range and the link interface. Each interface has any number of destination IP address range entries that dictate how packets with addresses between different ranges are forwarded through different physical links. For instance, if a routing table has an entry for link 1 for range 192.168.0.1/24, and a packet destined for 192.168.0.31 arrives, the packet will be sent out through link 1. There can also be default route entries, so if packets don't fall into one of the listed ranges, they will be sent out some default interface. If ranges overlap, the range with the longest prefix is selected as the appropriate entry.

On the network layer, packets can sometimes be divided (fragmented) into different pieces if they are too big. One datagram becomes several datagrams, but they are only reassembled into one IP packet once they reach the final destination. The IP header bits are used to identify and order related fragments into the original packet. Note that IP headers are 20 bits long and are not included in fragmented packets (that is, each packet gets its own IP header, but the original header of the larger packet is not considered data to be transmitted in the fragments).

4.3.1 IPv4 Addressing

All internet addresses are ultimately done in binary, and allow for up to 2^k addresses, where k is the length of the address in bits. For example, a network with 8 bit addresses supports up to 256 distinct addresses, while one with 16 bits supports up to 65,536 addresses. The original internet system — back in the 80's — used classful addressing under 32 bit addresses. The first half of the address was class A, allowing for up to 2^{31} addresses. If you halve the lower half of that, the upper part of that is a class B network. This process of halving the lower half and taking the upper part is repeated again for classes C and D. Each class allows for up to $2^{31}, 2^{30}, 2^{29}, 2^{28}$ addresses, respectively. Something like that anyways, because the network ID — the first set of bits in the address — varies with the class. Specifically, it's $8, 16, 24$ for classes A, B, and C, respectively. Since the first bit for each class is always fixed, class A networks start with 0-127. However, since class A networks beginning with 0 and 127 are not used, the actual range is 1-126.

For class B networks, the first two bits are set to 10, so class B networks start with 128-191. Class C starts with 110, class D with 1110, and class E with 1111, so their ranges are 192-233, 224-239, and 240-255 for the first number in the XXX.XXX.XXX.XXX address, respectively.

Unfortunately, this is all older material, and is no longer relevant (*note to self: why did we just spend a day and a half of class talking about it then?*). The modern internet uses a different system known as **C**lassless **I**nter-**D**omain **R**outing, or CIDR for short. CIDR uses network masks instead, where the first n bits are set as the subnet part and the remaining bits (the host part) are free to vary within a network. This is denoted using the $/n$ notation. For example, a network 192.168.1.0/24 would always start with 192.168.1, but the remaining number is free to vary from 0-255 since the last 8 bits are unmasked.

Subnets are groups of IP addresses assigned to interfaces that can physically reach each other without an intervening router. If two IP addresses are part of the same subnet, they share the same subnet part of their address. To identify subnets in a network scenario, detach each interface from its host or router. This will create islands of isolated networks, which are the actual subnets.

4.3.2 DHCP

DHCP, or **D**ynamic **H**ost **C**onfiguration **P**rotocol, is used on networks to dynamically assign IP addresses to hosts without need to manual intervention. When a host joins a new network, it will optionally send a DHCP discover message, which the server will respond to with a DHCP offer response. Even if that doesn't happen, the client will then send out a DHCP request response, which the server will reply to with a DHCP ACK.

DHCP discover messages are always sent to 255.255.255.255:67 (broadcast on port 67) with a source address of 0.0.0.0:68, along with some other data like a *yiaddr* (no idea what that is) and a transaction ID. The corresponding DHCP offer will have the IP address of the DHCP server on port 67, and the destination will be set to broadcast on port 68. The *yiaddr* field will contain the offered address, the transaction ID will be the same as the discover packet's ID, and an additional field for lifetime will be included. Once the lifetime expires, the offer is no longer valid. After the DHCP offer, the client will send a DHCP request that is largely identical to the last message, but with an incremented transaction ID. The DHCP server will then confirm the request.

Note that DHCP can return more than just the allocated address on the subnet. For example, it can send the address of the first-hop router for the client (default gateway?), the name and IP address of the DNS server, and the network mask. Technically, DHCP is a transport layer protocol that runs on top of UDP, but its purpose it network-layer oriented, so it's discussed here instead of in the application layer section.

4.3.3 RIP

RIP is a routing protocol, **R**outing **I**nformational **P**rotocol. It uses a distance vector algorithm with a distance metric as the number of hops between nodes. Each link has a cost of 1, with DVs exchanged with neighbors every 30 seconds, via advertisement. Each advertisement contains a list of up to 25 destination subnets (in the IP addressing sense). If no advertisement is heard after 180 seconds, a neighbor/link is declared dead.

4.3.4 OSPF

OSPF is a routing protocol, **O**pen **S**hortest **P**ath **F**irst. It uses a link state algorithm that is a little more robust than RIP. OSPF messages are authenticated, to prevent malicious intrusion, and it allows for multiple same-cost paths (as opposed to only one allowed path in RIP). OSPF can also be set up to be hierarchical.

4.3.5 BGP

BGP, or **B**order **G**ateway **P**rotocol, is yet another routing protocol. eBGP is used to propagate routing information between gateway routers, and iBGP is used to propagate that information amongst routers within an autonomous system. In general, when routers learn of a new prefix, they add it to their routing tables.

The advertised prefix includes BGP attributes; prefix + attributes = a "route". There are two important attributes: the AS-PATH attribute contains autonomous systems (AS) through which prefix the advertisement has passed; these AS numbers are maintained by ICANN (Internet Corporation for Assigned Names and Numbers). The NEXT-HOP field indicates specific internal-AS router to next-hop AS. There may be multiple links from the current AS to the next-hop AS.

4.3.6 IPv6 Addressing

IPv6 is a replacement for IPv4, motivated by the depletion of available IPv4 addresses. There can be at most 2^{32} (or about 4,000,000,000) IPv4 addresses, but this isn't enough for the modern internet.

IPv6 headers are somewhat longer than IPv4 (especially since they have 128 bit addresses), but they are also simpler. They contain a version field, a priority field, the flow label (or type of service in IPv4), the payload length, a field for the next header (to identify the upper layer protocol for data), the hop limit (also known as TTL in IPv4), the source address, the destination address, and the actual payload field. Notably, the packet checksum field has been entirely removed. This might seem strange at first, but it turns

out there's already a checksum at the link layer, so the complexity and redundancy of IPv6 is reduced as a result.

Turns out, transitioning from IPv4 to IPv4 isn't as easy as it sounds. Not all routers can be upgraded simultaneously, so the solution found is tunneling. IPv6 datagrams can be carried as the payload in IPv4 datagrams amongst IPv4 routers, allowing devices to use IPv6 even though the intermediary links use IPv4 only.

4.4 Link Layer

The link layer concerns the actual network links between different devices on a network (hosts). While the job of the network layer is to route data across the internet, the job of the link layer is simply to handle individual hops. For example, Ethernet is a link layer protocol, as is 802.11 ("WiFi"). Each link layer protocol provides different services; a given link may or may not provide reliable data transport, for instance.

4.4.1 Error Checking

The link layer may add error detection and correction bits (EDC), in order to protect data. Error detection is not 100% reliable, as it may miss some errors, but this happens rarely. A good rule of thumb is that larger EDC fields yield better results (i.e. lower chances of missing errors).

In particular, parity checking can be used to check for errors. This is done by maintaining a single bit that tells whether the number of '1' bits in a byte is even or odd; a 0 indicates odd parity, a 1 indicates even parity. This is effective at checking simple errors like single bits being flipped, but it won't help with out-of-order bits or two bits being flipped just right.

To fix this, two-dimensional parity checking can be used. Bytes are lined up in a column, and the parity bit is calculated for each row of bits *and* each column of bits. If the parity check fails, you can often pinpoint the exact bit that was flipped, so some single-bit errors are actually correctable. Note, however, that this still is not perfect.

An alternative to parity checking is the cyclic redundancy check, or CRC, which is a more powerful error detection coding. CRC views the data bits, D, as a binary number, and chooses an $r+1$ bit pattern (generator) G, with the goal of choosing r CRC bits R such that $< D, R >$ is exactly divisible by G (modulo 2). The receiver knows G, divides $< D, R >$ by G, and checks the remainder. If the remainder is non-zero, an error has been detected. There's an actual mathemagical formula for this: $D \cdot 2^r = nG \bigoplus R$. In other words, if you divide $D \cdot 2^r$ by G, the remainder R should satisfy $R = remainder(\frac{D \cdot 2^r}{G})$.

4.4.2 ARP

<++> Ethernet is run over wires and switches to provide network access to connected devices. Switches along the local area network can make actual physical topology easier, but note that they are layer 2 devices, and thus do not have IP addresses.

On the link layer, each device has a MAC address, or **M**edia **A**ccess **C**ontrol address (also known as a "physical address"). These are 48-bit numbers that are usually written as 6-part hexadecimal numbers. On a local area network, devices can use MAC addresses to communicate with one another... more or less. To ensure MAC address uniqueness, manufacturers buy portions of MAC address space (2^{24}); actual allocation is administered by the IEEE.

The question then becomes: if you know an interface's IP address, can you find its MAC address? The answer is ARP, or **A**ddress **R**esolution **P**rotocol. Each node maintains an ARP table, which maps IP addresses to MAC addresses, along with a TTL value that tells the node when to forget the mapping (usually after 20 minutes). If a node wants to get another node's IP address, it will broadcast an ARP request to the entire local area network; the appropriate node will receive (hopefully...) the message and will respond directly to the sender with its MAC address. Like DHCP, this is another "plug-and-play" protocol, in that it requires no further intervention from an administrator.

4.4.3 Ethernet

Ethernet headers are very simple. The first section is 8 bytes, the first 7 of which are just "10101010" and the last of which is "10101011", used to synchronize the receiver and sender clock rates. The next two fields are destination and source addresses, both 6 bytes (they're MAC addresses). The "type" field contains information on what protocol might be contained inside; usually this is IP, but there are alternatives like Novell IPX or AppleTalk. The final field is a 4-byte checksum field that uses CRC for error detection.

Switch Switches are layer 2 devices that store and forward ethernet frames without actually making any modifications. They examine the ethernet frame and selectively forward the packet out of some interfaces. Switches are full duplex and allow for multiple systems to transmit across them without collisions (packets are buffered as needed). They are also entirely transparent, so network nodes don't really interact directly with the switch (more or less).

Each switch maintains a switch table that looks a lot like a routing table; each entry contains the MAC address of a host, the interface needed to reach the host, and a TTL value. Every time a switch receives a frame from one system, it records ("learns") the location of the sender. When a frame is received at the switch, it first records the incoming link and MAC address of the sending host. It will check the switch table for the destination MAC address; if found, it will forward the frame out through the specified interface. Otherwise, the switch will broadcast the frame on all interfaces.

Note that switches can be connected directly to each other, and the self-learning property will still apply. This allows complex, hierarchical networks to be formed using switches.