February 27

una.3daystartup.org

What we should know:

We need to understand data encoding techniques.

We need to know the layering techniques.

We need to understand common

Networking has changed the way the world works.

What are the requirements for building a network system?

1. The network should be capable of growing to a global size.

2. The network needs to handle a bulk number of applications.

3. Should support telephony.

4. Should support streaming video and audio.

What is a network?

1. Terminals connected to a computer.

2. Telephone network (POTS – Plain Old Telephone System)

3. Cable television

4. Water/sewer systems

5. Distributing electricity

All of these networks are designed to do one specific task.

Computer networks are not designed to do anything specifically, but to do many things generally. For this reason, computer networks are not optimal. Networks are made up of general purpose hardware.

It's important to know what type of applications are going to be used, and the users of that network.

Terminology

1. Connectivity

a. Sometimes limited connectivity is good enough, as you might not want any possible corruption in the network.

2. Scalability – The ability for a network to grow or scale in size. A usually desirable feature is that the network can scale.

3. Nodes (vertex) – an individual component within some graph

4. Links (edge) – a representation of some physical path between two nodes

a. Ethernet

b. Wifi

c. Radio

d. Coaxial

5. Topology

a. Direct connections

1) Point-to-point – A direct link between two nodes.

2) Multi-point (multi-access) – A single link between more than 2 nodes.

b. Indirect connectivity

1) Used to allow scalability that direct connections cannot handle.

2) Nodes are connected to a device (**switch**) that utilizes software to forward messages from one link to another link.

3) Switched network – utilizes a device (**switch**) that forwards messages between links.

a) \*Two types:

I. Packet switched (connectionless, packet-switched/oriented)

I) Postal model

1. We write a message and give the message with the appropriate address to the network, which handles passing the message to other networks (a **hop**) and so on and so forth.

2. Messages are sent hop by hop.

3. The network moves form place to place by forwarding towards the destination “hop by hop.”

4. Data carries the destination address

5. Overhead is associated with reading and processing the destination address at each intermediate node.

6. The setup overhead is very low or non-existent

7. This is best for short-lived interactions between a set of frequently changing nodes.

II. Circuit switched (connection-oriented)

I) Telco model

1. A complete source-to-destination path is established before any data is sent (or before the data even CAN be sent). The path is determined by the destination address.

2. Data does not need the address, but just follows the established path.

3. There is overhead associated with setup and tear-down

4. Forwarding overhead is very low or non-existent

5. Best for long lived connections between a set of static nodes.

b) Edge devices – devices on a network that communicate through some “cloud” to another device on the network

I. Cloud – a symbol that represents a network that may contain subnetworks that we don't care about.

c. Point-to-point

1) Simplex – a connection that goes only one way

2) Half-duplex – communication messages can flow in both directions, but can only go in one direction at a time.

3) Full-duplex – communicate messages can flow in both directions at the same time.

d. Ring – Circular connections

e. Bus – a single line that one or more devices may connect to

f. Star – a switch device in the middle to which other devices are connected

g. Switched (different from previously mentioned switches)

Internetworks of subnetworks

1. Subnetwork – a small network that is part of another network

2. Internetworks – a connection of two or more networks

a. Also known as **internets**

b. The Internet is the foremost example of internets

3.. Connected with

a. **Gateways** – some node that connects two or more networks

c. Routers

d. Bridges

Network requirements

1. Connectivity

a. Each node must be able to specify what other node it wishes to connect to.

b. This means that each node must have an address, which must be unique.

c. If the nodes are not on the same network, then messages will have to be forwarded between other networks.

d. Forwarding is the process of determining the direction of the messages.

1) Routing – a forwarding table that determines the destination address with some link.

e. The ?ANS

1) PAN – personal area network (such as bluetooth)

a) Allows a limited connection in terms of length

2) LAN – local area network

a) A few hundred nodes over a few hundred meters

b) These run from kilobits (Kb) up to 10 gigabits (Gb)

3) MAN – metropolitan area network

a) A city sized network

b) Ranges from kilobits up to 10s to 100s of megabits (Mb) per second

4) WAN – wide area networks

a) State or country-wide networks

b) Can be slower by using satellites in remote areas

f. Bits vs Bytes and Quantities

g. Network – two or more nodes connected by a link, two or more networks connected by one or more links, two or more internets connected by one or more links, etc.

a. Due to the generality of the Internet, innovation by use of the Internet has exponentially increased over the last two decades.

2. Cost Effective Resource Sharing

a. Take the things/resources that are expensive, and share them between many users.

b. Multiplexing – share a link between multiple nodes simultaneously

1) Queuing methods

a) Round robin

b) FIFO

2) We have two multiplexers, a multiplexer to take the messages to be sent, and a multiplexer to receive the messages and pass them to the correct nodes (demultiplex).

3) Multiplexing types

a) STDM – synchronous time division multiplexing

I. Allocates a slice of time to each node that needs to pass messages

II. The problem with this is that some node may have the time slice but does not have a message to send, and we have to predetermine how many nodes we will have in advance (not scalable).

b) FDM – frequency division multiplexing

I. The link between the two multiplexers is able to support multiple frequencies for each node, allowing multiple nodes to talk at the same time.

II. This has the same problem of wasted resources as STDM if the node has no message to send, and we have to predetermine how many nodes we will have in advance (not scalable).

c) Statistical multiplexing

I. The line is still shared by time slice, but the time slots are dynamic, meaning that a node may use the first available empty time slot without waiting for a specified time slice.

II. The problem is that there is a possibility for starvation of a node if another node is hitting the multiplexer with many messages.

i) We may fix this by putting a limit on the number/size of messages that can be sent. This results in a large message being broken up into several smaller (fixed-size) messages (aka **packets**).

ii) In this case, a node can only send messages if no other node has a message to send.

4) Congestion

a) The problem of **congestion** then arises if the senders send more messages than the available space of the “pipe”/link between multiplexers.

3. Support for common services

a. The network should support services to make building applications easier.

b. Two means of accomplishing this:

1) Every program must require all the network functionality

2) Provide the services necessary to perform certain functionality

c. A logical channel is a pipe that provides a service between two nodes.

1) Possible services

a) Provide security

b) Deliver messages in order

c) Deliver error-free messages

2) Examples:

a) Request/Reply – guarantees all delivered, no duplicates, integrity (message has not been altered.

b) Message Stream - -guarantees messages are all delivered, in order, confidential, and multicast.

3) We have to be careful about the types of services that we choose

a) We don't want to include services that are not needed, as it results in some overhead cost.

4) Where does the intelligence go (bellheads vs netheads)?

a) In the network (adopted by the telephone engineers)

b) In the ends (adopted by the intenet engineers)

4. Reliability

a. 3 classes of failures

1) Bit errors – one or more bits get flipped

a) Usually occurs in bursts

b) Caused by lightning, power failures, etc.

b) Can often be detected and corrected

2) Packet errors – the packet is lost

a) Package never is transferred

b) A node or network is down

3) Link level

Communication

1. Unicast – point-to-point communication between two nodes on a network

2. Broadcast – a connection of one node to every node on a network

3. Multi-cast – a connection from one node to a subset of nodes on a network

Terminology

1. Bits – b

2. Bytes – B

3. K, M, G, …

a. Kilometer (K) = 103, M=106, G=109

b. Kilobytes/Kilobits (K) = 210, M=220, G=230

4. Example

a. Data at rest (does not hold true for marketing weasels)

1) Data storage – 100MB = 100 \* 220 B​

b. Data in movement

1) Rate or distance – 100Mbps = 100 \* 106 bps

c. Comparison

1) (220 = 1,048,576B) != (106 = 1,000,000)

5. Bandwidth

a. Data rate in bps (this is what we usually mean)

b. Frequency range in hertz (cycles per second – cps)

6. Networking architectures (blueprints, templates)

a. Layers

1) Layers give us abstraction to help with more efficient code

2) This allows us to easily modify the modules that need to be changed without breaking other modules.

3) Higher layers are not dependent upon the details of lower layer implementation, while lower layers are not dependent upon the different ways that higher layers access/move data.

4) Structure (high to low layers, each is a **protocol**)

a) Application

b) Process to process (maybe even channels?)

I. Request/Reply channel

II. Message send channel

c) Host to host

d) Hardware

5) Definitions

a) Protocol – Rules and formats of communication that define the meaning and validity of messages

b) Service interface – the service that each layer provides to the one above or below it.

c) Peer interface – the interface that allows connection between two peers. The actual connection is a direct connection on a hardware layer between two peers, but it seems to the process to process/channel layer that it is directly communicating with another process to process/channel layer as a virtual connection.

d) Encapsulation – the process of adding or removing headers/trailers to the message as it is passed between layers. Each time a layer adds its own header/trailer, it considers the rest of the message that was passed to it as the “payload” (including the headers from other layers).

e) Headers – contain the control information for the layer's peer.

f) Demultiplexer key (demux key) – a key passed within the header that specifies the destination of the message, that is, which application gets the messages.

b. Monolithic

1) This takes less time for up front development, but the maintainability is much worse since code is not written as modules that can easily be modified without breaking the other code.

c. OSI Model (7 layer model, protocol graph, wedding cake model) (high to low) (better conceptually, but not actually used)

1) The layers

a) Application – applications; communication services exposed to the program

b) Presentation – formatting of data; handles data formats

c) Session – manages sessions; provides a name space to manage message streams.

d) Transport – host to host; process-to-process communication, where bits are organized into messages; which process gets the message?

e) Network – moving messages from host to host, end to end; handles connections between nodes; bits are organized into packets

f) Data [Link] – organizes bits into frames, and provides reliable delivery

g) Physical – hardware and actual transfer of bits on the “wire”

2) Acronym (Low to High) – “Please Do Not Take Sausage Pizza Away”

d. TCP/IP Network Architecture

1) IETF – Internet Engineering Task Force

a) Handles the documentation and protocols that define the Internet

2) 4 Layer Model (this model was the one that actually implements networks)

a) Two graphical representations (high to low)

I. Hourglass (narrow-waisted)

I) HTTP, FTP RTSP

II) TCP, UDP

III) IP

IV) NET-1, NET-2, NET-3

II. Box

I) Application (heading)

II)TCP, UDP (each takes up 1/3 of the level below it

III) IP (takes up half width)

IV) Network (takes up full width)

III. The Box model shows that it is possible to go straight from the application to the Network, but the hourglass is better for abstraction of implementation.

b) Layers (in their relation to the OSI model)

I. Network – physical, data link (only some of data link)

II. IP – network

III. TCP/UDP – transport, Session and Presentation (these two are sometimes in TCP/UDP, and sometimes in Application)

IV. Application – application

c) The layers are not always strictly implemented in this way. This allows the possibility for new technologies to replace the commonly used layers if necessary.

BSD (Berkely System Distribution)

1. History

2. API

a. In most OS (apart from Windows, who wrapped the API), we can directly to the API.

b. Some programming languages have wrapped this API.

c. The API is an implementation of some concept, not a protocol.

c. Types

1) Socket

a) Under the BSD License, which allows free modification and use of some software, as long as the original copyright is kept.

b) A socket is an endpoint of communication

c) Socket (2) – means that on a UNIX or Unix-like system, there is a manual page in section (n). We can access the manual with “man socket.”

Client-Server

1. We must be able to create a server and a client.

2. To create a server, we must be able to create a socket.

a. Functional prototype: int socket(int domain, int type, int protocol);

1) This function returns a “handle” for the socket as an integer

2) If it fails, then it will return -1

b. Domain/Protocol Family

1) PF\_INET – we will mostly use this

2) PF\_UNIX

c. Type

1) SOCK\_STREAM – a stream oriented channel; TCP

2) SOCK\_DGRAM – connectionless; UDP

d. Protocol

1) UNSPEC – unspecified, since the protocol family and the type has enough information to characterize the type of socket. This must be passed.

3. Steps

|  |  |
| --- | --- |
| **Server** | **Client** |
| 1) Socket  2) Bind  3) Listen  4) Accept  5) Send/receive  6) Close | 1) Socket  2) Connect  3) Send/receive  4) Close |

a. Bind – which address we want to connect to

1) Function prototype: int bind(int socket, struct sockAddr \*address, int addr\_len);

a) In C, the type is explicitly “stuct someTypeName”

b) The demux key is stored inside “struct sockAddr \*address”, which is also referred to as a port.

I. Ports may be found under /etc/services

II. IANA – a companion organization to the IETF that handles these standards.

c) Errors will return -1

b. Listen

1) Function prototype: int listen(int socket, int backlog);

a) The backlog is the maximum number of connections that can be pending/queued at one time.

c. Accept

1) Function prototype: int accept(int socket, struct sockAddr \*address, int \*addrLen);

a) We initially pass empty values to \*address and \*addrLen.

b) This allows the server to accept requests on port 80, and then return a new socket on which the request may be handled.

c) Blocks until a connection is received.

d. Connect

1) Function prototype: int connect(int socket, struct sockAddr \*address, int addrLen);

e. Send – send a message

1) Function prototype: int send(int socket, char \*message, int msgLen, int flags);

a) Returns the number of bytes that were actually sent

f. Receive

1) Function prototype: int recv(int socket, char \*buffer, int bufLen, int flags);

a) We provide an empty buffer to put the received message into

g. Close

1) Function prototype: int close(int socket);

h. DNS resolver

1) Function prototype: struct hostent \*gethostbyname(const char \*name);

3. Ports

a. Defined by IANA

b. Defined within /etc/services on Unix-like machines

1) Any port < 1025 is considered “privileged,” meaning that only the root user can initialize these ports.

2) We will run a datetime server on port 1113 instead of port 13.

4. Daemon

a. Definition – a program without an interface

b. We want to be able to kill the process nicely

1) When the process is killed abruptly, allocated memory and/or open files are not cleaned up.

2) We are going to write a signal handler to handle SIGKILL (in this case, SIGTERM (terminate))

5. Datetime server

a. gethostbyname vs getaddrinfo

1) gethostbyname – the original version that only works on IPV4

2) getaddrinfo – the newer version that works on IPV6

b. The UNA server is: cs-srv-01

Note: Use the alternatives mem\* for bcopy, bzero on Windows. Also, they have moved the net connection library to a new file, winsock2.h, which does not include bcopy or bzero.

Linux Toolbox

1. telnet
2. netstat – see IPs on a network.
3. nc (netcat) – good for simulating both client and server
4. apropos – search for manual pages of commands that do certain things
5. ping – check if server/host is alive
6. traceroute (tracert on Windows, but only pings) – lists all vertices between your location and the end vertex. Helps discover routing problems.
7. host (nslookup on Windows, but also works on Linux) – resolves the IP address for a specified domain. All the previous commands invoke the DNS resolver.
8. route – shows the routing tables
9. whois – see who is responsible for taking care of some domain (registrar, registrant)
10. make – a means of compiling a set of specified files all at once; this does not recompile files that have not changed
    1. We have to create a makefile, called “Makefile”
    2. We specify the compiling functions in the file:
       1. all: <alias> [<alias>]
          1. The files that the makefile depends upon
       2. <alias>: <filename>\n\t<command>
          1. Specify the files associated with the alias; the instructions on how to build the alias.
          2. On the next line, we tab and then type the compile command
             1. g++ -Wall -g -o server server.c
       3. clean:\n\t rm -f <filename>
          1. Gets rid of any object files or core files so that it can be built from scratch again
          2. -f forces a removal
          3. Filenames
             1. \*core – files that are created when it crashes
             2. \*o – object files that are created
    3. make <makefilename> – makes all the files specified in the makefile.
    4. make clean – cleans all the files specified in the makefile

Implementing protocols

1. Unfortunately, the socket API is not efficient enough for protocols.
2. Models for implementing a protocol
   1. Process model – 2 different approaches
      1. Process-per-protocol – each protocol is implemented as a separate process
         1. We use some method of passing messages (IPC)
         2. Because each protocol is a separate process, we will have a context switch for different protocols.
      2. Process-per-message – all protocols are implemented in a single process, each protocol is a procedure call.
         1. There is no need for passing messages.
         2. We have eliminated the need for a context switch, since we are only calling a series of functions.
   2. Message buffers – the application creates and manages the buffers for messages sent or received.
      1. To move the message from protocol to protocol (between layers), we copy the contents of the buffers (pass by reference array).
3. Bandwidth (throughput) – the number of bits that can be transmitted per unit of time
   1. It can mean:
      1. Frequency – Hertz (measurement of cycles/second of the wave form)
         1. The width of a pulse on the link? (Think in terms of signal duration, such as in an oscilloscope)
            * Bit is 1 microsecond wide on a 1 Mbps link
            * Bit is 0.5 microseconds wide on a 2 Mbps link
      2. Data rate
         1. How long does it take to put a bit on the wire?
            * On a 10Mbps link, it takes 0.1 microseconds per bit
   2. Throughput
      1. Usually indicates the actual performance of the link (how many bits can be crammed through the wire) vs the theoretical capacity (bandwidth) of the link.
   3. Latency – the time it takes for a bit to travel end to end over a link
      1. Measured in time; usually milliseconds
         1. Several years ago, 24ms was a good number for a transcontinental link (U.S.)
      2. RTT (round trip time) – latency in both directions
      3. Three components
         1. Propagation delay = distance / speed of light
            * Speed of light = maximum speed through media

3.0 \* 108 meters/second in a vacuum

2.3 \* 108 meters/second in a cable (copper)

2 \* 108 meters/second in fiber

* + - 1. Transmission time = number of bits/bandwidth
      2. Queuing delay – how long messages spend waiting in a queue to be sent; tends to be unknown
    1. Latency = propagation delay + transmission time + queuing delay
  1. Moving large amount of data at once is usually preferable to moving bits quickly (low latency); however, sometimes its more important to quickly transfer several files.
  2. Example (figure 1):
     1. Link 1
        1. 10 Mbps
        2. Copper cable
        3. Distance – 100 km
     2. Link 2
        1. 10 Mbps
        2. Copper cable
        3. Distance – 100 km
  3. Delay X Bandwidth product
     1. We can think of how much a link can carry at its capacity (volume of the link)
     2. We can think of a link as a pipe, where the delay is the length and the bandwidth is the height
     3. Example
        1. 150 ms latency with 10 Mbps
           + (150 \* 10-3) \* (10 \* 106) = 1.5 \* 106 bps = 183 KBps

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Figure 1* | **Bits** | **Propagation delay(s)** | **Transmission time(s)** | **Latency(s)** |
| **100 Kb** | 819,200 | 0.000435 | Link 1 – 0.82  Link 2 - 0.08 | Link 1 – 0.82  Link 2 – 0.08 |
| **100 Mb** | 838,860,800 | 0.000435 | Link 1 – 838.86  Link 2 – 83.89 | Link 1 – 838.86  Link 2 – 83.89 |

Calculate the round trip for the given info:

128 Kbps link

Distance 55 Gm (to Mars?)

3 \* 108 m/s

Remember, latency = propagation delay + transmission delay + queuing delay (can't determine queuing delay)

55 Gm / 3\*108 m/s = 55 \* 109 m / 3\*108 m/s = 184s one way latency

184 seconds \* 2 = 368 seconds RTT (round-trip time)

How many bits fit in the tube (delay bandwidth product)?

184 seconds \* (128 \* 103) bps = 23.5 Mb

What if we wanted to send a 5MB message; how long would it take?

5MB / (bandwidth) = transmission delay

5MB / 128 Kbps = 5MB / (128 \* 103) bps = (5 \* 220 \* 8) b / (128 \* 103) bps = 328 seconds + time it takes to go across space = 328 + 184 = 512 seconds

Layers

1. Physical layer
   1. Types of common data links
      1. Direct connection
         1. Category 5 cables (twisted pair of cables, Cat5, “ethernet”)
            * 4 pairs of twisted wires)
            * Speed is 10-1000 Mbps
            * Length is good up to 100 meters
         2. Category 6 cables
            * Speeds of 10 Mbps – 10 Gbps
            * Limited to 100 meters
         3. Thin-net (RG58/62)
            * A coaxial cable

Has a copper wire that is completely encompassed by a copper mesh, wrapped by a sheath (insulator).

Essentially a faraday cage, to protect it from radio waves

* + - * + Speed of 10-100 Mbps
        + 200 meters
      1. Thick-net (extinct)
         * 10-100 Mbps
         * 500 meters
      2. Multi-mode fiber
         * 100 Mbps
         * 2 Km
         * Cheaper compared to single mode
         * Uses multiple light rays (modes) carried simultaneously through the wave guide (fiber)

Light will only propegate in the core at discrete angles (light does not get out of the tube)

It suffers from modal dispersion (different light waves arrive at different times at the other end)

* + - 1. Single-mode fiber
         * 0.1 – 10 Gbps
         * 40 Km
         * Cost more than multi-mode and harder to work with
         * Has a much longer distance than multi-mode since there is no dispersion
    1. Indirect connections
       1. If the nodes are too far apart for a direct connection, we'll lease circuits from a common carrier.
          - DS0

1 voice circuit

64 Kbps bandwidth

Delivered on copper

* + - * + DS1 (T1, E1 in Europe)

24 voice circuits

1.544 Mbps

Delivered on copper

* + - * + DS3 (T3)

872 voice circuits in one wire (28 T1)

44.736 Mbps

Delivered on copper

* + - * + STS-1 (OC1 on other continents)

810 voice circuits

51.84 Mbps

Delivered on fiber

* + - * + STS-n

Defined all the way up to OC192/STS-192

n \* 51.84 Mbps

OC192 9.9 Gbps

* + - * + Metro-Etherenet (recently in the last 5 years)

Up to 1 Gbps

* + - * + ATM – Asynchronous Transfer Mode

DSL services

* + - * + DSL – Digital Subscriber Line

Rides on the ATM

* + - * + ISDN – Integrated Services Digital Network

1 and 2 channel versions

Only up to 128 Kbps for 2 channel

* + - * + POTS – Plain Old Telephone Service

300 bps – 56 Kbps

* + - * + CATV (cable television)

1-40 Mbps (should be faster now)

* + 1. Last mile
       1. The last connection from the CO to the consumer location
       2. Central offices (CO)
    2. Optical connections
    3. Copper vs Fiber
       1. Copper carries electrical signals
          - Succeptable to electrical errors
          - Cheaper and easier to install than fiber
          - Lower in terms of speed, distance, and bandwidth than fiber
       2. Fiber carries optical signals
          - Sending pulses of light down pieces of glass
          - We can run fiber
          - Longer distances and bandwidth than coper
          - Much more expensive than copper

Made of many thin stands of optically pure glass

Is much more fragile than fiber.

* + - * + Harder to repair than copper
    1. Twisted pair dominates the networking world
    2. All network links share a common problem with getting bits on the wire
       1. Encoding – representing data in binary form
          - NRZ – non-return to zero (obvious naming)

High signal = 1, low signal = 0

Really easy to understand

Problems

Baseline wander

When signals travel over some distance, the charge decreases.

This means that we can take a measure of the low and high voltages, and take the average of them to find the middle.

This also creates a problem if we get several 0s or 1s, since there is no good way to measure the average, which confuses the receiver; known as **baseline wander**.

Clock recovery

How does the receiving end know when to read the wire for a bit?

We can solve this by using a clock on both ends that is synchronized.

We can send the clock information by assuming that the clock shifts when the signal transitions.

However, this causes a problem with desynchronization if there are too many 0s or 1s in a row.

We could run another wire, but it doubles the cost.

* + - * + NRZI (non-return to zero inverted)

A 1 results in a transition from high to low, while a 0 means that there is no change.

This fixes consecutive 1s, but still does not solve the problem of a long sequence of 0s.

* + - * + Manchester encoding

Start with NRZ, and add XOR

Sender will XOR its clock and the NRZ encoded bits

0 = low to high transition

1 = high to low transition

This requires that the Baud rate (clock rate) be doubled

This also costs half the bit rate

* + - * + 4B/5B

We send 4 bits of data + 1 bit = 5b

5 bit codes to be selected as follows:

Each sequence of bits has no more than one leading 0

Each sequence has no more than two trailing 0s

Therefore, back-to-back 5 bit codes never have more than 3 consecutive 0s

We use NRZI to solve the 1s problem

We need to make up some encoding (pg 82)

* + - 1. Modulation – varying the frequency, amplitude, or the phase of the signal
         * Frequency shift keying (FSK)

Different frequencies (tones) represent different bit sequences

That is, if we use 2 tones, we can use one to represent 0 and the other to represent 1.

1.5 Khz = 0, 2.0 Khz = 1

4 tones

00, 01, 10, 11

Thinking of frequency in wave form, we can shrink/increase the width of the wave in order to increase/decrease the frequency. In other words, we can think of frequency in terms of “cycles”.

* + - * + Phase shift keying (PSK)

Different phases represent different bit sequences

Thinking of the frequency in wave form, as we move along the x-axis of the wave, each new point could be a different set of bits. That is, we can shift the wave to the left or right.

* + - * + Amplitude

Changing the amplitude causes the height of the wave to change.

Different amplitudes can represent different bit sequences.

It's not as good of a choice, since signals lose amplitude as distance increases, which means a possible loss of data (think of amplitude in terms of loudness for a telephone system).

* + - * + Modem – modulator demodulator

It takes bits from a computer and then modulates them into the appropriate signals, and then performs the opposite process to get data.

Most modern modems use a combination of FSK and PSK

A modem's speed is represented by two concepts:

The rate at which we change the signal = baud rate. How fast we are changing either the tones or the phase.

How many bits are being sent per unit of time = bit rate

Usually, bit rate != baud rate

This is a type of device that is referred to as **serial communications**

Serial – bits are put into the wire one at a time

**Parallel communication** – By definition, they will generally be faster, since multiple wires are used in parallel. However, engineering tricks were used to make serial faster than a normal parallel communication (which is more expensive).

* + - * + Example: a modem uses FSK with 16 tones, transmitting 1000 tones per second

Baud rate = 1 KBaud (kilobaud)

16 tones = log2 16 = 4 bits per tone

4 bits \* 1000 tones/sec = 4 Kbps

1. Layer 2 – Data link
   1. Organized bits into frames
   2. Responsible for reliable delivery across the link
   3. Handles the following functions:
      1. Accepts data from higher layers and format data into frames.
      2. Synchronizing frames across the link
      3. Controlling the rate of flow of frames to avoid overwhelming the receiver
      4. Detect and handle frame errors
   4. The data link layer is divided into 2 sublayers. Each sublayer is usually implemented separately
      1. LLC (logical link control) layer – performs data link functions that are independent of the medium used.
         1. Issues
            * How do we mark the start and end of a frame?
            * Can frames have variable length payloads?
            * How do we identify the sender and receiver?
         2. Frame formats
            * Bit oriented protocols – HDLC (high level data link control)

Frames are handled as a sequence of bits.

We use a distinguished bit sequence to mark the start and end of a frame.

0111 1110 (in HDLC) for the beginning and ending

We repeatedly send this on an idle link, in order to help keep the clock in sync

If the same sequence shows up as actual data within the frame, then there will be a problem.

To overcome this, we can “stuff” a 0 in once we detect the fifth 1: 0111 11010

Server inserts a 0 if it sees 5 ones.

Receiver if we see 5 ones, examine the next bit to decide

If the next bit = 0, and discard the zero since it was stuffed by the sender.

If the next bit = 1, then it must be the end of the frame, or there is an error.

If the next bit is a 0, then it is the end of the frame

If the next bit is a 1, then it is an error.

It is possible to lose the next frame if there is framing error.

* + - * + Byte oriented protocols – BISYNC, PPP (point-to-point protocol)

Frames are handled as a sequence of bytes

BISYNC frames start with SYN + SYN + SOH +

SYN – synchronize

SOH – start of header

Payload is delimited with STX data ETX

STX – start of text

ETX – end of text

If ETX shows up in the payload, we escape it with a DLE character

DLE – data link escape

If DLE occurs in the payload, escape it with a DLE character

Sentinel character (escape characters)

Instead of sentinels, we could just include the payload length in the frame header

This does not make them immune to possible errors

* + 1. MAC (media access control) layer – performs data link functions that are dependent on the underlying medium.
  1. Clock-based framing
     1. SONET (synchronous optical networks) (also known as SDH in other parts of the world)
        1. Created by Bellcore (Bell Labs)
        2. Was standardized by ANSI
        3. It deals with framing and encoding
        4. It was focused on data for phone companies
        5. It multiplexes several links over one link
        6. Framing
           + 810 bytes
           + 9 rows x 90 columns
           + The first 3 bytes is control information

The first 2 bytes are synchronization bytes to keep the clock synchronized

We still have the problem of the control info possibly showing up in the data, but this is essentially ignored since it is expected to run on a certain interval.

* + - * + Uses NRZ encoding

The clock problem is (kind of) solved

The baseline problem has to still be solved

This is handled by a 127 bit sequence where XOR causes transitions

Th

* + - * + Generally, SONETs are STS-n

STS-1

Bandwidth of 51.84 Mbps

One frame is sent every 125 microseconds

STS-3

We take 3 SONETs and interleave the frames over one frame.

We essentially split the frame of each SONET into parts and send those parts one at a time, multiplexing each frame.

This prevents the clock from changing while tripling the bandwidth

* 1. Errors
     1. We want our protocol suite to detect errors, and maybe even correct them if possible.
     2. Error detection
        1. A trivial solution is to make a copy of the bits
           + This is a lot of overhead
           + There is no guarantee that this will even work
        2. We need to consider:
           + The number and kind of errors that it will detect
        3. Simple parity
           + Add one bit for every number of bits
           + We have odd and even parity:

In both, we count up the number of 1s (or 0s if desired) and then add a 1 at the end of the bits in order to influence the oddness/evenness.

For example – 1011 010 with an odd parity results in 1011 0101 since a 1 is needed to make it odd.

Another representation is 7e1

* + - * + This is easy to compute and has little overhead
        + It does not protect in the following cases:

It only catches an odd number of flipped bits

The actual parity bit being flipped

* + - 1. 2D parity
         * We use simple parity to add to the end of a string of bits, but we can also apply simple parity to the column of bits from a set of bit strings.
         * For example:

100, 011, 101, 010 results in

1000

0111

1011

0100

1111

* + - * + We can not only detect an error, but we can correct the bit that was a problem by tracking it down according to the associated row/column of detection.

It will catch up to 3 bit errors, and most 4 bit errors.

* + - * + This is stronger than simple parity, and has less overhead than sending all the bits twice.
        + Generally speaking, it is faster to re-transfer lost frames rather than correcting it, unless we have a link with very high latency.
  1. Checksums
     1. These are actually done at a higher layer
     2. We literally add up the sum of all the bytes, and add the result as the last byte.
        1. The receiver checks if the checksum matches the sum of the data that was sent. If not, the frame is resent.
     3. The Internet checksum is on page 94-05
  2. CRC (cyclic redundancy check)
     1. Detects, single bit, double bit, and all n-bit odd bit errors
        1. It is used in ethernet
     2. If we append *r* bits, then we can detect r-1 bit burst errors
     3. Details start on pg 97
     4. The nice thing about this is that we can implement this directly on hardware, making it really fast.
  3. Notes:
     1. The sender just gives a “best effort”
     2. There are no restrictions on
        1. If it all gets delivered
        2. What order it gets delivered
        3. Whether it has errors
     3. So, essentially there are no guarantees, or in other words, there is unreliable delivery.
        1. No guarantees = unreliable delivery = best effort
     4. Questions
        1. What prevents the sender from overrunning the receiver? Nothing
        2. How do we know that all frames are delivered? We can't.
        3. How do you know that the info is in order? We don't.
        4. How do we know that there are no errors? We don't.
     5. We want
        1. Reliable delivery
           + All delivered
           + In order
           + Error free (bit errors)
  4. Flow control (between layers)
     1. Simplex
        1. Essentially a “best effort”
        2. Simply accepts the data from the layer above or below, and passes it to the next layer.
        3. No checks take place; the sender does not know when the receiver gets the message.
        4. Doesn't solve any problems, but it is the easiest to implement.
     2. Simplex + ACK (acknowledgement)
        1. Same as simplex, but adds the ability for the receiver to send an acknowledgement back to the sender that the frame was received.
        2. What happens if a frame is lost? The receiver never gets an acknowledgement.
     3. Simplex + ACK + Timeouts (ARQ – Automatic Repeat Request)
        1. The sender starts a timer when it sends a frame.
        2. The sender expects that it will receive an acknowledgement that the frame was received before the timer runs out.
        3. If the timer runs out, then the frame is sent again and the timer is started again.
        4. Problems occur if the acknowledgement gets lost or if the acknowledgement is received after the timer (timer is too short).
           + The receiver might get the same frame twice.
     4. ARQ + Frame ID (sequence number) (Stop and Wait)
        1. An ID is sent with both the frame and the acknowledge.
        2. This solves the problem of the duplicate frame.
        3. Only one unacknowledged frame can be on the link at a given time.