Week.2.txt

- January 18th, 2021
    - Outline
        - Internet architecture and design principles
        - How internet services are optimized
        - How the internet is future-proofed
            - We have applications and services that the founding fathers could not even imagine
    - Internet Protocols
        - The internet goes beyond the network of networks, routers, switches, etc.
            - The software that runs the internet is quite important
                - i.e. TCP, IP, HTTP, Skype, 802.11, Etc.
        - Protocols control the sending and recieving of messages
            - Protocols are necessary for interoperability, and allow different networks to talk to each other
                - Anybody can design their own internet protocol, but it may not be recognized by other vendors
        - The Internet standard utilizes the Requests For Comments (RFC) standard
            - The Internet Engineering Task Force (IETF) ensures that:
                - There is no host in the protocol
                - Protocols are compliant by the vendors
    - What Is A Protocol
        - Protocols define format, order of messages sent and recieved among network entities, and actions taken on message transmission or receipt
            - Protocols define the content/format of the message
                - i.e. What is in the header, what kind of information can be stored in the payload of the message
            - Protocols dictate which message may follow others and between the communication entity
            - Protocols contain information about what action needs to be taken upon reception of the message
        - Protocols are standardized by the IETF
            - The IETF specifies the format, ordering, and actions that should be followed by all vendors, or those who implement the protocol
                - Standardization allows interoperability
                    - i.e. Windows OS can talk to a server running Linux, as long as they use the same protocol/standard
    - Protocol Layers
        - Networks are complex with many "pieces"
            - i.e.
                - Hosts
                - Routers
                - Links of various media
                - Applications

        - Protocols
        - Hardware
        - Software
    - Protocols need to be optimized to ensure maximum efficiency and future-proof
- Why Layering
    - Without layering, protocols will directly interface with the transmission
      media
        - i.e. Protocols such as SMTP, SSH, FTP, etc., will have their own standard that defines how to send data over transmission media such as coaxial cable, fiber optic, etc.
        - This approach is bad because it is not going to scale
            - Every protocol needs its own implementation of communicating over the transmission media
                - Adding more protocols or media is very tedious; time and resource consuming
            - If there are 'M' different transmission medias, and 'P' different protocols, then there needs to be (M X P) different interfaces
                - As an application developer, you do not want to deal with the underlying complexity, nor do you want to create additional software that isn't relevant to your main application
- Solution: Indirection
    - The solution is to introduce an intermediate layer, instead of exposing the low-level complexity from different transmission media directly to protocols
        - This is "Indirection", coined by David Wheeler, and is frequently used to solve problems in computer science
        - Modifying the intermediate layer does not break com-munication between protocols and the transmission media
            - Adding new protocols and medias is easy, because they need to interface with a "single" intermediate layer
                - i.e. Easy to add new technologies like Wi-Fi and protocols like HTTP or HTTPS
        - The intermediate layer is several layers, but this is hidden, through abstraction, from application developers
        - The organization of the intermediate layer is called the internet architecture
- Network Architecture
    - Note: Architecture is not the implementation itself
    - Architecture is a specification that tells us how to organize implementations
        - i.e. What interfaces are supported?
        - i.e. Where functionality is implemented?
    - Network architecture is the modular design of the network
- Modularity

- Software Modularity
  - Complex systems are broken in modulues
  - Well defined interfaces gives flexibility
    - Can easily change the implementation of modules as long as the interface remains the same
      - i.e. Optimize module for speed without breaking existing apps
    - Can extend functionality of system by adding new modules
      - i.e. Introduce new functions
  - Interfaces hide information
    - Separation of concerns
    - Allows for flexibility
    - May hurt performance
- Network Modularity
  - Like software modularity, but with a twist:
    - Implementation is distributed across routers and hosts
      - But, you must decide:
        - How to break system into modules
        - Where modules are implemented?
          - i.e. Should this module be on the end host, or implemented on the network's router(s)

- Layering
  - Layering is a form of modularization
    - Modularity is not arbitrary
  - The system is broken into a vertical hierarchy of logically distinct entities, called layers
    - This hierarchy is like a stack
    - Makes reuse possible, but may hinder performance in some cases
      - The performance may suffer because the data is passed through extra layers, causing processing overhead
    - i.e.
      End-host layer
      Application
      Network
      Link
      Physical
    - i.e.
      Router layer
      Network
      Link
      Physical
- Key Concepts
  - Service
    - A service says what a layer does
      - Ethernet

            - Is a widely used protocol in local area networks
                - Abbreviated as LAN
            - Unreliable subnet unicast/multi-cast/broadcast datagram service
        - IP
            - Unreliable end-to-end unicast datagram service
                - IP will route packets from source to destination, but makes no attempt to deliver the message reliably
        - TCP
            - Reliable end-to-end bi-directional byte stream service
    - Service Interface
        - A service interface says how to access a service
            - i.e. Socket interface
            - i.e. TCP/IP are modulues, and you need to specify how to utilize the services
    - Protocol
        - A protocol says how the service is implemented
            - This is a set of rules and message formats that govern the communication between two peers
            - i.e. How to implement TCP and make it reliable, so bytes can be continuously streamed
- Internet Protocol Architecture
    - Note: In older textbooks you may see a 7-layer model
    - The TCP/IP protocol suite is the basis for the networks that we call the internet
        - The TCP/IP suite has four layers (from top to bottom):
            - Application Layer
                - i.e. Telnet, FTP, SMTP, HTTP, DNS
            - Transport Layer
                - TCP, UDP
            - Network Layer
                - IP, ICMP, IGMP
            - (Data) Link Layer
                - FDDI, ATM, Ethernet
    - Computers (hosts) implement the entire TCP/IP suite, all four layers, and they implement the physical layer
    - Routers (gateways) only have the bottom two layers; the Network layer and (Data) Link layer, combined with the physical layer
- Layering In Action
    - The application (i.e. Web browser) encapsulates your query/search
        - The message is passed down to the transport layer
            - The transport layer adds a header to the message
                - The header is also referred to as payload
        - After the transport layer, the message is passed to the network layer
            - The network layer adds an additional header

        - The packet is now referred to as a datagram
      - After the network layer, the datagram is passed to the link layer
        - The link layer adds another header to the datagram
          - The datagram is know referred to as a frame
    - The datagram, or message, is sent by the physical layer
      - It travels through a bunch of switches until it reaches the destination router's physical interface (i.e. Wireless, wired, etc.)
        - Once the router's link and network layer have the datagram, they read the header to determine where the packet needs to be sent
          - When the packet reaches the correct destination, it travels through the machine's network layer
            - Each header in the datagram is stripped until the message is left
    - When a message is sent it travels DOWN the TCP/IP stack, and additional information is added in the form of headers
      - When that message has been recieved by the destination's machine, it travels UP the TCP/IP stack, and each header is removed until the message is left
        - Note: Headers are also referred to as payloads

- January 20th, 2021
  - Internet Protocol Architecture
    - The TCP/IP suite has a total of four layers
      - In order, from the top, they are:
        - Application Layer
        - Transport Layer
        - Network Layer
        - (Data) Link Layer
    - Internet architecture is a way to organize the implementation of services and functionality in the internet
  - Layering In Action
    - Information coming from the top (source address) will be sent through the interfaces in between layers
      - The control information is added to the message that will be further sent along the protocol stack
    - When packets arrive at switches and routers, they are first processed at the physical layer, and then the subsequent layers; network and link
      - The packet is processed based on the data encapsulated in the headers
        - i.e. Where should this packet be sent?
    - Once the packet reaches the destination host, the packet travels up the protocol stack until the message is delivered to the application
      - Each layer removes its corresponding header/payload
  - Services Of The Layers
    - Application

        - Collection of application protocols that can be used to support a variety of network applications
            - i.e. FTP, SMTP, HTTP, DNS, Telnet, Etc.
                - These protocols can be utilized by applications to serve you content
                    - i.e. HTTP is used by web browsers to send data back and forth between hosts
                - Applications can utilize more than one protocol
    - Transport
        - Transfers data between processes and the system
        - Transport layer protocols only deal with the data transfer between end system processes
            - i.e. TCP, UDP, Etc.
    - Network
        - The network layer routes datagrams from source to destination
            - It also contains protocols that define error handling and error messages, if there are any failures in the network
            - i.e. IP, ICMP, OSPF, RIP, BGP, Etc.
                - Traceroute uses the ICMP protocol
            - Note: Applications do not have to strictly use the application layer protocols, they can also use protocols in the network layer
    - (Data) Link
        - Transfers data between neighbouring network elements
            - i.e. A laptop and nearby wireless router
            - i.e. Two different routers
        - Examples of (data) link layer protocols:
            - Ethernet, 802.11, Bluetooth
                - Ethernet is for wired connections
                - 802.11 is wireless (WiFi) connections
- Layering In An Example
    - A web browser sending a query to a Google server
        1. Source Host (i.e. Web Browser)
            - At the source host, the web browser application will utilize the HTTP protocol, an application layer, to encapsulate data.
            - Then, the HTTP protocol will use the TCP protocol, from the transport layer, to reliably transfer data to the end host.
                - Also, TCP ensures that the messages arrive in the order that they are sent
            - Next, the TCP protocol will utilize the IP protocol, provided by the network layer.
            - Then, link layer services are used to encapsulate the IP packet/datagram, and send it to the next hub
                - The link layer can be Ethernet, Wi-Fi, etc.
        2. Router(s) (i.e. Hops)
            - When the packet arrives at the next router, it

processes the packet. The router decides which hub the packet should be sent to
  - Achieved with the help of a routing table and forwarding
3. End Host (i.e. Web Server)
  - Once the packet reaches the end host, it travels up the protcol stack.
    - i.e. It may use ethernet for the link layer, and then IP, TCP, & HTTP to process the packet

- Reality
  - Layering is a convenient way to think about networks
    - This is because it is modular and linear
  - Layering architecture is not always followed rigorously
    - i.e. Middle-boxes look at information in multiple layers
      - i.e. Firewalls, network address translation
        - Firewalls inspect information in the IP layer, transport layer, application payload, and they may even perform deep packet inspection
    - i.e. Cross layer optimization
      - In order to optimize performance, you need to take into account the information from multiple layers
- The Internet Design Question
  - When communicating across the internet, services should have the following characteristics:
    - Reliability
      - Packets should not be lost or dropped
    - In-order delivery
      - The first message sent should be the first message arrived
    - Guaranteed bandwidth
      - Smooth and seamless audio/video streaming, downloading, etc.
    - Low latency
      - Communication should be fast, especially for Warzone
    - Accountability
      - Packets arrive at the end host destination
    - Security
      - Information should be encrypted
  - Other important questions to ask:
    - What functionalities should be supported?
      - Where should the functionality be placed?
- Two Design Principles
  1. Hourglass model
    - Focuses on the network core
  2. End-to-end argument
    - Focuses on the end systems' point of view
  - Note: These two design principles are pretty much the same, but viewed from different perspectives. However, both principles answer the question, "Where and what functionalities should be implemented on the network

elements?"
- The Hourglass Model
  - This model mimics an hourglass, where the waist is very narrow, and the opposite ends are quite large
    - The opposite ends, top and bottom, symbolize the TCP/IP layers
      - The top end is the application and transport layer
      - The bottom end is the (data) link layer and physical layer
        - i.e. The physical layer can be copper, fiber, Wi-Fi, Bluetooth, cellular, etc.
      - The top and bottom of the hourglass model are very diverse and contain a vast amount of applications, technologies, etc.
    - The waist is the network layer
      - It only contains essential protocols that are future-proof
        - It is the common denominator for both ends of the hourglass model
          - This ensures that no matter what happens to the other layers, the network layer (waist) can interface with both ends of the stack
      - The waist of the hourglass is kept as small as possible, to easily incorporate new applications and technologies, without breaking the old ones
- Implications Of Hourglass
  - The Hourglass model:
    - Is a single internet layer module
    - Allows different networks to interoperate as long as they support IP
      - In the past, cellular did not support IP, and data sent from cellular had to be translated by a gateway
        - Once companies realized the implications of IP, they made their core network IP based
    - Allows all applications to function on all networks
      - An application that can run on IP can use any network
    - Enables simultaneous developments above and below the IP layer
      - i.e. New cellular technologies like 5G can be developed and easily incorporated into the existing internet network
      - i.e. Developers can make apps and easily deploy them on the internet, as long as those apps use IP
    - Simplifies the router's implementation
      - This is because IP implements the minimum set of functionality
        - In turn, the router can make decisions very quickly, and send packets out as fast as possible

        - In contrast, burdening the router with stuff like security, reliability, etc. complicates the implementation, slows down processing in the router, and negatively affects the source-destination connection
    - Routers have multiple interfaces inside them
        - i.e. Switching fabric, forwarding table, etc.
- The End-2-End Argument
  - The end hosts decide, based on requirements, if certain functionalities should be implemented in the network
    - i.e. If you want to implement reliability in the network, should you implement this on the end host, or the network/router
    - If you implement something in the network, it should consult the following criteria:
      1. Reduces the host implementation complexity
          - The functionality should only be implemented in the network, if it reduces the complexity in the hosts
            - In contrast, the functionality should not be implemented in the network if it adds complexity in the host
            - It is better to have complexity in one part, than both parts
      2. Does (greatly) it increase network complexity?
          - Adding complexity to the network will slow it down
            - The network should be kept as fast as possible
      3. Does implementing in the network impose delay on all apps, regardless of whether or not the apps use the functionality?
          - Apps should only use the protocols that they need
  - The end-2-end argument is similar to the hourglass Model
    - Both aim to make the network as streamlined as possible, and only implement the bare minimum needed by all applications
- The E2E Argument Example (1)
  - Should routers try to fix corrupted packets? In other words, should reliability be implemented in the network?
    - Reliability is a common requirement for applications
        - Reliability states that the message is correctly and eventually delivered from the application process, on the source host, to the destination host
  - To answer this question, the following questions need to be looked at:
    1. Will it reduce host implementation complexity?
        - No. You still have to check the packet once it arrives at the destination host

        2. Does it increase network complexity?
            - Yes. You do not want to check every packet at every router, especially when billions of packets are sent every second
        3. Does it impose delays and overhead on all applications?
            - Yes. Some applications are delay sensitive but loss tolerant, and do not need reliability
                - i.e. Streaming movies/music
                - This functionality should be left to the end system to implement
                    - i.e. Add TCP for reliability
- The E2E Argument Counter Example (1)
    - Reliability should not be implemented in the network layer, and should be left to the applications to check whether packets are recieved
        - However, implementing reliability in a very lossly link can enhance performance in some cases
            - This is ideal for when you are on Wi-Fi, and you are trying to connect to a server across the ocean. The Wi-Fi connection has a low latency, but high bit error rate. And the outer-continental connection has a high latency, but low bit error rate. Thus, it is ideal to check packets before they are sent across the ocean, to make sure that corrupted (useless) packets aren't being sent
                - Reliability is not implemented at the router or inside the network, but at certain access points
                    - i.e. Just before the packet crosses the ocean
- The E2E Argument Example (2)
    - Should data privacy be implemented in the network? In other words, should the network be responsible for securing application data?
        - Data privacy is essential for banking apps and private communication based apps
    - To answer this question, the following questions need to be looked at:
        1. Will it reduce host implementation complexity?
            - No. The packets will still have to be encrypted, BEFORE, they are sent to the router. Otherwise, somebody can perform a MITM attack
        2. Does it increase network complexity?
            - Yes. Packets need to be encrypted at the router, which increases overhead
        3. Does it impose delays and overhead on all applications?
            - Yes. Some applications may not need their data to be encrypted before being sent over the web
                - i.e. Broadcasting the existence of your device so other devices on the network, or nearby, can communicate with your device

        - As a counter point, we should also think about this from a moral and ethical perspective, and not just an engineer's perspective
            - Privacy is a fundamental human right, so maybe the tradeoff is worth it if we can achieve a more secure internet
                - However, it's probably better to just force developers to encrypt and secure data by default

- January 22nd, 2021
    - Take Home Message
        - Chapter 1 summarized:
            - The internet is a network of networks
            - In order to deliever a packet from source to destination host, the packet will travel through multiple intermediate routers
            - Traceroute is a program that lets us view the connectivities of the internet
            - The internet protocol is organized in a modular way
                - This particularly applies to the TCP/IP architecture
            - The TCP/IP protocol is a layered architecture; it consists of 4 layers, and they are: Application, Transport, Network, and (Data) Link layer. The physical layer is below the TCP/IP stack
            - Network hosts implement the entire TCP/IP stack
            - Routers only implement up to the network layer
    - Application Layer
        - Reading: K & R Chapter 2
    - Outline
        - Principles of network applications
            - This is the focus of this lecture
        - Web and HTTP
        - DNS
        - Socket programming
    - Application Layer
        - When discussing the application layer, we need to specify:
            - What kind of services the layer will provide to the layer above?
                - The application layer provides support to network applications
            - What kind of services the layer will use from the layer below?
                - The application layer uses services provided by the transport layer
                    - i.e. TCP, UDP, Etc.
        - The application layer sits on the end systems, not the routers
    - Application Layer Architectures
        - There are different ways you can implement a distributed application

        - The most common way is client-server architecture
        - Less common ways are Peer-To-Peer (P2P), and a hybrid of client-server and P2P
- Client Server Architecture
    - Client-server architecture consists of two types of entities
        - Servers
            - Always-on hosts
            - IP address is permanent
                - So clients can connect to them without hassle
            - Servers are employed in data centers
                - This allows easily scaling if there is a lot of demand
                    - However, this architecture is heavily centralized
                    - Additional resources need to be paid for
        - Clients
            - Clients are typically user devices
                - i.e. Your laptop, phone, etc.
            - The apps on your device will communicate with servers
                - Some of them are intermittently connected
                    - i.e. iCloud and other backup services
            - IP address is dynamic
                - Your local IP address is different for every Wi-Fi you connect to
            - Generally, clients do not communicate directly with one another
                - This is because Clients:
                    - Are not always available
                    - Do not have a permanent IP address
                        - This makes it harder to discover other clients on the internet
                    - Sit behind a firewall
                        - This makes communication harder
- P2P Architecture
    - P2P stands for peer-to-peer
        - It is also called peer-peer
    - This is another way you can implement distributed applications
        - It is the alternative to the client-server model
    - In P2P:
        - The networks hosts are not always on
        - Arbitrary end systems directly communicate with each other
        - Peers can request service from other peers, and provide a service in return to other peers
    - P2P is self-scalable
        - Because new peers bring more than just new service demands, they also bring new service capacity
            - In other words, more people on the network means

            more people's hardware you can use to service other people's needs
    - Peers in a P2P network are intermittently connected, and their IP address is dynamic/changes
        - Managing this is very complex
    - Examples of P2P architecture:
        - Torrents
            - BitTorrent
            - ThePirateBay
            - uTorrent
        - Bitcoin
            - Entirely decentralized
            - Utilizes blockchain technology
- Hybrid
    - The hybrid model combines the client-server model and P2P architecture
    - Consider the application, Skype:
        - Servers provide account authentication and maintain information for clients
            - i.e. Logging into Skype requires authenticating with user information
        - But when you video chat with someone, the data does not flow to the server, but directly to the peer
            - Data communication is done directly between clients/peers
                - i.e. Video-chatting with someone on skype
            - This removes a huge load from the server
                - i.e. 10,000 clients can be video-chatting with each other, and the server won't take a performance hit
- Communicating Processes
    - Processes are the programs that are running within a host
        - Within the same host, two processes communicate using inter-process communication
            - This is supported by the OS
        - Processes in different hosts communicate by exchanging messages, according to the specification of the application layer protocol
            - In a client-server model, there are:
                - Client processes that inititate communication with the server
                - Server processes that wait to be contacted by the client
                    - The process is sitting on a particular port, waiting to be contacted
            - In P2P architecture, applications have both client processes, and server processes
                - The server process handles file retrieval requests from other peers
                - The client process retrieves files from other

peers
- Implementing both processes, simultaneously, requires multi-threading

- Sockets
  - Sockets are the interface between the application layer, and the transport layer
    - Note: Transport layer is below the application layer
  - Sockets allow processes to send or recieve messages between the two layers
    - Processes send/recieve messages to/from its socket
  - Sockets are analogous to a door
    - The sending process shoves the message out of a door
      - It does not need to worry about how the message is delievered
        - The transport layer is responsible for this
    - On the reciever side, the recieving processes are listening on a port, waiting for the message to be delievered from the transport layer
- Addressing Processes
  - In order to determine which application the message needs to be delievered to, each process must have an identifier
    - The identifier includes both IP address and port number; or port numbers
      - The IP address refers to the host and is unique
      - The port number is associated with a process on the host
  - The IP address of the host is set by the router
    - Routers have multiple interfaces
      - Each interface is associated with one IP address
  - Since all applications share the same IP address, additional information is needed to distinguish the incoming messages
    - Port numbers are used to deliever data to the correct apps
      - Example port numbers:
        - HTTP Server: 80
        - SMTP Mail Server: 25
      - An HTTP request to www.cas.mcmaster.ca will include:
        - IP Address: 130.113.68.10
        - Port Number: 80
          - This information is in the transport layer
- Requirements: Common Apps

Application	Data Loss	Throughput	Time Sensitive
File Transfer	No Loss	Elastic	No
E-Mail	No Loss	Elastic	No
Web Documents	No Loss	Elastic	No

Real-Time Audio/Video	Loss Tolerant	Minimum To Maintain	Yes; < 100 msec delay
Stored Audio/Video	Loss Tolerant	Same As Above	No; video can buffer
Online Games	Loss Tol.	Minimum	Yes; small delay
Text Messaging	No Loss	Elastic	Yes, and No; Depends on msg

- Internet Transport Protocol Services
  - As a developer you need to identify the requirements for your app
    - i.e. Is the app sensitive to data loss, is it time sensitive, does it require a minimum throughput, etc.
    - Understanding the requirements for your app helps you identify what internet protocols you need to implement
      - There are sockets that associate with TCP, UDP, and there are raw sockets that directly interface with the IP service
        - i.e. Traceroute uses raw sockets to directly interface IP
  - TCP service:
    - Reliably transports data
      - Packets sent from source are not lost, and corrupted packets are re-sent
      - Destination host sends acknowledgements when packets are recieved
    - Provides flow control
      - The sender does not overwhelm the reciever with lots of data
        - This is done by throttling the sender
        - i.e. If the sender can upload faster than the reciever can download, then TCP will throttle the sender's up-speed
    - Provides congestion control
      - Congestion is caused by queueing delay
        - i.e. Lots of packets are sent to the routers or switches
          - TCP will regulate the sending rate of packets to prevent overloading the network
    - Does not provide:
      - Timing
      - Minimum throughput guarantee
      - Security
    - Is connection oriented
      - TCP will setup the connection required between the client process and server process

            - This enables the source host to send data to the destination host in a reliable, and orderly manner
    - UDP service:
        - The data transfer between sending and recieving processes is unreliable
            - Acknowledgements are not sent from reciever to sender
        - Does not provide:
            - Reliability
            - Flow control
            - Congestion control
            - Timing
            - Throughput guarantee
            - Security
            - Connection setup
        - UDP is faster than TCP because it does not provide reliable transport, flow control, and congestion control
            - UDP is good for streaming services because a few dropped packets won't hurt the performance and overall quality
- Internet Apps: Application, Transport Protocols

Application	Application Layer Protocol	Underlying Transport Protocol
E-Mail	SMTP[RFC 2821]	TCP
Remote Terminal Access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File Transfer	FTP [RFC 959]	TCP
Streaming Multimedia	HTTP (i.e. uTube) RTP [RFC 1889]	TCP or UDP
Internet Telephony	SIP, RTP, Proprietary (i.e. Skype)	TCP or UDP

    - The choice of transport layer protocols is dependent on the application developers
        - i.e. Do they want to take advantage of something that exists?
          OR
          Do they want more flexibility to control the

behavior?
- Google, built on top of HTTP 1.1, and created HTTP 3.0
  - HTTP 3.0 uses UDP instead of TCP
    - Because they want to control the behavior to avoid latency in loading web content
- Each transport layer protocol has its own tradeoffs between the complexity of the implementation and the flexibility to control the behavior of your software
  - i.e. You don't have to use TCP, UDP, etc. strictly