Review for Chapter 2 in the book "A top-Down Approach to Networking"

Principles of Networking (2.1)

Network applications are programs that run on different hosts and communicate with each other over a network. Common examples include web browsers, email clients, file transfer applications, and video conferencing tools. The goal of this section is to explain how these applications function in terms of communication and design

Client Server Paradim

Server: Always on host, permanent IP, often in databases.
Clients: Communicate with server, dynamic IP, do not communicate directly, intermittenly connected.
(Examples: HTTP, IMAP, FTP).

Peer to Peer

No server, arbitrary end systems (hosts) which directly communicate with each other. Peers request service from other peers, in return, it provides its services back to peers. Allows self-scalability meaning as new peers join, new service capabilities and demands. Peers are intermittenly connected and have dynamic IPs.
(Examples: Torrents, P2P File Sharing).

Process Communicating

Process: program running within a host
Inter-Process Communication: Within the same host, two processes communicate with each other (Defined by Operating-System).
Messages: Process in which different hosts communicate with messaging.

Addressing Proccesses

To receive messages, processes must have an identifier, host devices have a unique 32-bit IP Adress.
Indentifier: Includes both IP Address and port numbers, associated with processes on hosts. (Exampls: HTTP runs on Port 80, Mail runs on Port 25).

Application Layer Protocol

Types of messages exchanged (Request / Response)
Message Syntax: What fields in messages and how filds are described.
Message Semantics: (Meaning of information in Fields). Rules for when and how processes send and reponse to messages.

2 Protocols

Open: Everyone has access to protocol defined in RFCS, allows for interoperability (Examples: HTTP, SMTP)
Proprietary: Owned by a company and operations are unkown (Example: Skype, Zoom)

Sockets

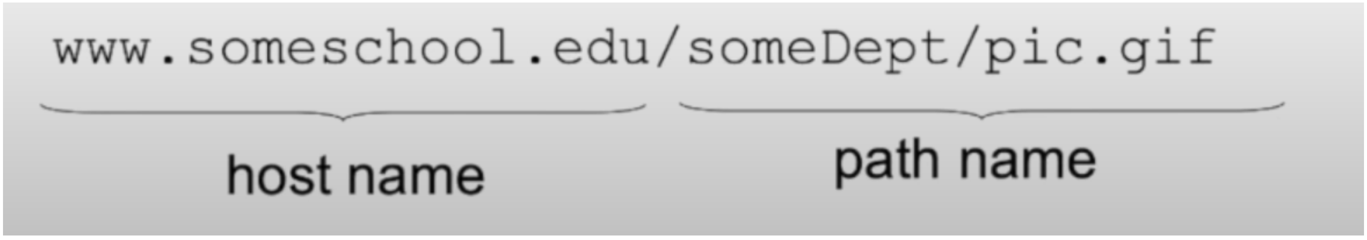
Interface between the application process and the network, acts as an endpoint for sending and receiving messages. (Can be thought of as a door).

Transport Services

Roles of the Transport layer in providing services to the Application layer (Either through TCP or UDP).

The Web and HTTP (2.2)

Web page consists of objects, each of which can be stored on different web servers.
Objects: can be a HTML file, JPEG image, audio file, java applet.
The web page consists of base HTML-files which includes several referenced objects, each addressable by URL.



HTTP also uses a client/server model with the client being the "browser" and the server being the web server.
HTTP also uses TCP which allows the client to connect to the HTTP server using port 80.
HTTP is also considered "stateless" meaning it maintains no infromation about pas client requests and history.
No worrying about rollbacks or cleaning problems, as there is nothing to clean!

Persistent vs Non-Persistent HTTP

Persistent	Non-Persistent
TCP connection remains open for multiple requests and responses	Seperate TCP connection for every request

HTTP Request Message

Contains the HTTP method of either GET or POST, (the URL resource and HTTP version).
Headers: Provide additional information about the request (Such as Domain Name, Acceptable Data Types, User-Agent, etc.)

Request Example

```
GET /index.html HTTP/1.1 (What the client wants)
Host: www.example.com (DNS)
User-Agent: Chrome/95.0 (User's Browser)
Accept: Text/Html (Perferred objects that will be accepted)
If-Modified-Since: Mon, 12 Oct 2023 18:00:00 GMT (Modification records)
```

Other Request Messages

POST	HEAD	PUT	DELETE	GET
Used to send completed form data to the server	Similar to GET but only headers	Uploads a resource to the server	Deletes a resource on the server	used to retrieve a resource only if the resource has been modified since the last time it was requested

HTTP Response Message

Status Line: includes the HTTP version and status code with its phrase accordingly. (e.g, 404 = "Not Found", 200 = "Success")
Header Lines: Metadata about the response (type of content, length, and caching information).
Body: contains the actual content (Images, HtML of web page, etc.)

Response Example

```
HTTP/1.1 200 OK
Date: Mon, 14 Oct 2024 10:30:00 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Type: text/html; charset=UTF-8
Content-Length: 138
```

```
HTTP/1.1: The HTTP version being used.
200 OK: The status code and reason phrase, indicating that the request was successful.
Date: The time and date the response was generated.
Server: Information about the server software handling the request.
Content-Type: Specifies the type of content being returned (text/html in this case).
Content-Length: The length of the message body in bytes (138 bytes).
```

HTTP Status Codes

200	301	400	404	505
Request Successful	Moved to a new location	Message not understood	Request not found on server	HTTP version not supported

Cookies

Used to maintain Information about a User (Solves the "Stateless" problem). Cookies allow users to stay logged in across multiple requests, remembering their preferences, and is also used for analytics / targeted advertising. Uses a [Set-Cookie] Header.

The Web and HTTP (pt2)

Web Caches

Goal: Satisfy the client request without involving the origin server. This allows an improvement in web performance by storing web objects closer to users (On a Proxy Server, CDN or Browser Cache). This reduces latency, loading time, overall load and bandwidth.

Conditional GET

Goal: Don't send objects if the cache already has an up-to-date version. The header request indicates when it last fetched the resource, then checks if it has been modified. If not, it sends a 304 error status code, but if it is, then the server sends the updated resource.

Content Delivery Network (CDN)

Distributed Network of server across the globe used to deliver web content to users more quickly.

HTTP/2 and HTTP/3

Updated versions of HTTP that aim to improve efficiency of data transfer between web clients and servers

EMAIL (2.3)

Three components; user agents, mail servers and SMTP

Simple Mail Transfer Protocol (SMTP)

Used to send emails from a client to a server, or a server to a server (Called the push protocol).

POP3 (Post Office Protocol V3)

Used by email clients to retrive messages from the mail server. Considered a "pull protocol"

Internet Message Access Protocol (IMAP)

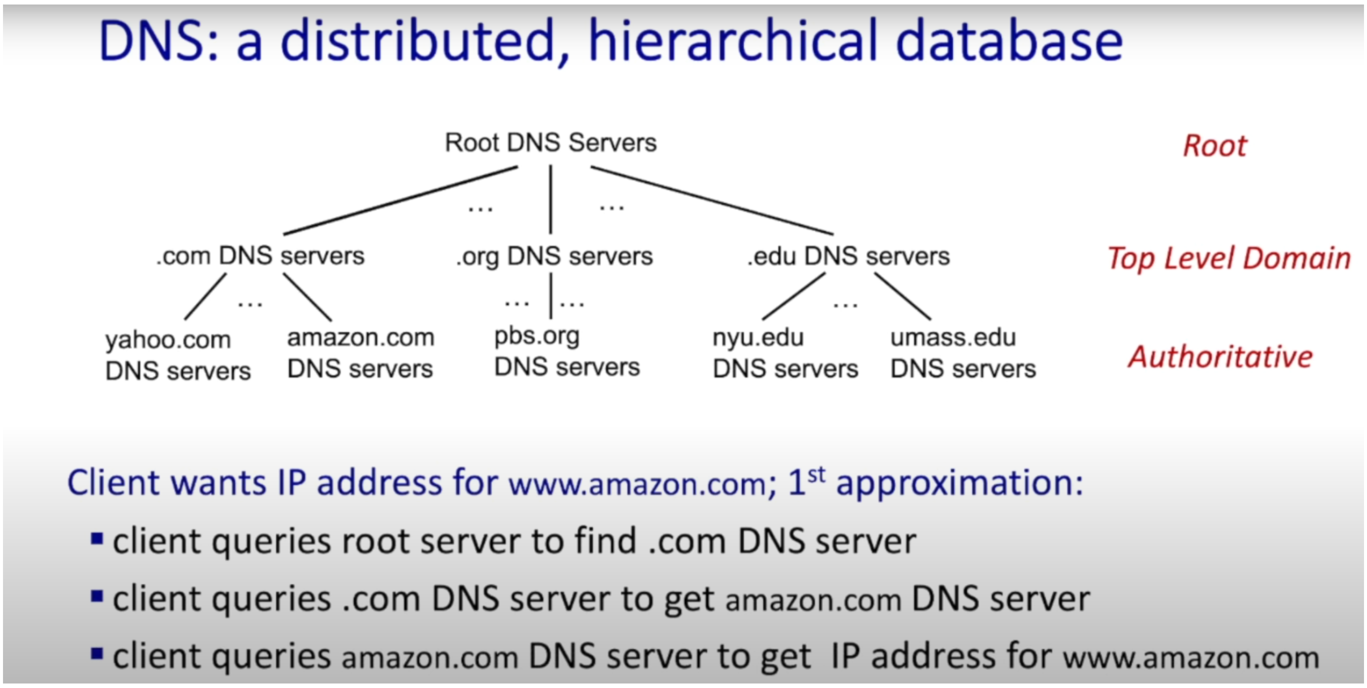
Also another pull protocol, but allows users to keep emails on the server and organize then into folders.

Domain Name System (DNS) (2.4)

DNS

Maps human-readable domain names (<www.example.com>) to machine readable IP addresses (192.168.1.0). (Considered Host-to-IP translation).
This allows people to access websites without remembering complex IPs.

DNS Hierarchy



Authoritative DNS

Servers hold the DNS record for specific domain names, providing the IP associated with a domain.

Root DNS

Servers respond to queries for top-level domains. When a DNS Resolver does not know the IP of a given domain, it queries to the root servers, which reponds with either a .com or .org.

Top Level Domain

Catagorize domain names based on either their purpose (.edu for education or .gov for government) or they can be used for country specific domian names (.us for United States or .fr for France). (Also called TLCD).

Local DNS

Server finds recent name-to-address translation pairs saved from its local cache (each ISP has one).

DNS Queries

There are two main types of queries; Recursive and Interative.

Recursive query

Client asks DNS resolver to resolve the domain name. The DNS client is responsible for doing all the work to return a final answer

Interative query

Rather than the DNS client doing all the work, it instead refers the client to another DNS server that is 'higher' in the DNS hierarchy to resolve the domain name.

DNS Caching

Used to reduce latency and improve efficency, when a DNS resolver, resolves a domain, it caches the result for future requests.

DNS Resource Records

Used to record specfic details about a domain such as its IP, Mail Server, etc.

Examples

```
(A) Records: Maps a domain name to an IPv4 address. www.example.com → 192.0.2.1
(AAAA) Records:  Maps a domain name to an IPv6 address. www.example.com →
2001:db8::1
(MX) Records: Specifies the mail servers responsible for receiving email for the
domain. example.com → mail.example.com
(CNAME) Records: Maps one domain name to another (aliasing). www.example.com →
example.com
```

DNSSEC (Security)

Introduced to protect against attacks by providing data integrity and authetification for DNS queries and responses. This can prevent DNS Spoofing and/or Cache Poisoning.

Peer-To-Peer Applications (2.5 - 2.6)

P2P

P2P Applications are distributed systems where "Peers" directly communicate and share resources without relying on a centralized server. Peers act as both the client and the server.

Key Characteristics

Decentralization: P2P systems distribute tasks amongst peers, making the network more resilient.
Scalability: Bandwidth and storage increases as more peers join, which allows P2P networks to scale effectively according to its demands.
Self Organization: Peers dynamically join and leave the network and the system adjusts accordingly to maintain its functionality without manual intervention.

P2P File Distribution

Rather than uploading an entire file to every client, P2P breaks the file into chuncks, then, some peers have specific chunks which they share with other peers.

BitTorrent Protocol

As discussed above, peers have specific file chunks which they send to other peers. BitTorrent also has a tracker to help file sharing communication amongst peers.
Choking: Regulates which peers it uploads to. (peers that upload data quickly are prioritized for getting data).
Optimistic Choking: Periodically gives slower peers a choice to receive data to prevent them from being left out

Distributed Hash Table (DHT)

Data Structure that allows peers to efficiently locate and retrive files or resources from other peers.
Data is stored across multiple peers, each responsible for the overall data.
DHT provides a scalable way to look up info without needing a central server or tracker

Challenges

Security: Often targeted for abuse, such as spreading malware, DDOS attacks, as their is no central authority.
Free Riding: Peers who consume resources without contributing back

Other Uses

Voice of IP (VoIP): Systems like skype use P2P to route voice and video calls
Content Distribution: CDNs use P2P to share popular content.
Live Streaming: Peers share live video segments with each other

Socket Programming (2.7 - 2.8)

Introduction

The Socket is considered the "door" between the application process and end-end-transport protocol. It is a programming construct that allows a network application to send and recieve messages over the internet.
Sockets can run on either TCP or UDP

```
TCP (Transmission Control Protocol): Provides reliable, connection-oriented
service with congestion control, flow control, and error recovery mechanisms. It
ensures that data arrives in order and without errors.

UDP (User Datagram Protocol): Provides an unreliable, connectionless service. It
doesn’t guarantee message delivery, order, or data integrity, but it has lower
overhead and faster transmission than TCP.
```

Socket Programming with TCP

Socket programming with TCP uses a Client-Server Model. Meaning the server waits for a connection from the clients, and the client initiates communication.
TCP sockets are connection oriented and require a handshake where both parties agree to communicate.
TCP also ensures that the data is delivered reliably, if anything is lost or damaged, TCP will retransmit them.
TCP also ensures all data is in order as it was sent.
TCP allows applications to send a continuous flow of data without dividing it into discrete packets. The data is transmitted as a sequence of bytes, and it’s up to the receiving application to determine where each meaningful message starts and ends.

TCP Example

```
SERVER SIDE
```

```
Creates a socket that will listen for incoming client connections [socket()]
Server binds the socket to a specific IP and port number, alowing it to receive
data sent to this address and port [bind()]
The server socket is put in a listening state, waiting for client connection
requests [listen()]
When a client attempts to connect, the server accepts the connection [accept()]
The server can now receive data and send data to the client through the new socket
[send()] [recv()]
Once the communication is complete, the server can close the socket [close()]

CLIENT SIDE

Creates a socket to connect to the server [socket()]
Client specifies the server's IP and port number and tries to establish a
connection [connect()]
Once connected, the client can send and receive data from the server [send()]
[recv()]
Once complete, the client closes the socket [close()]
```

TCP example in Python

SERVER

```python
import socket  # Import the socket module for network programming

# 1. Create a socket object for the server, AF_INET specifies the address family
for the socket, we are using IPv4, SOCK_STREAM indicates we are using TCP
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2. Bind the socket to a specific address and port, localhost = 127.0.0.1, if you
want to allow any connections use '0.0.0.0'
server_socket.bind(('localhost', 8080))

# 3. Start listening for incoming connections (1 connection at a time)
server_socket.listen(1)

# 4. Accept a connection from a client (blocks until a client connects)
conn, addr = server_socket.accept()

# 5. Receive data from the client (maximum 1024 bytes)
data = conn.recv(1024)

# 6. Send a response back to the client
conn.sendall(b'Hello, Client!')

# 7. Close the connection after communication is done
conn.close()
```

CLIENT

```
import socket  # Import the socket module

# 1. Create a socket object for the client
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2. Connect to the server at localhost on port 8080
client_socket.connect(('localhost', 8080))

# 3. Send a message to the server
client_socket.sendall(b'Hello, Server!')

# 4. Receive a response from the server (maximum 1024 bytes)
data = client_socket.recv(1024)

# 5. Close the client socket
client_socket.close()
```

Socket Programming with UDP

UDP Sockets are used for connectionless, unreliable communication, it DOES NOT establish a connection before sending data.
UDP does not guarantee the delivery of packets, data may be lost, duplicated or received out of order, and not retransmissions for lost data.

UDP Example

```
SERVER SIDE

The server creates a socket for communication [socket()]
Server binds the socket to a specific IP and port number [bind()]
Rather than put on a listening state, the server simply waits for incoming data,
processes it and can respond by sending data back to the client. [recvfrom()]
[sendto()]
After communication is complete, the socket is closed [close()]

CLIENT SIDE

Creates a socket [socket()]
Client can send data directly to server by specifying the server's IP and port
number without needing to establish a connection [recvfrom()]
The client can also receive responses from the server [sendto()]
Socket is closed by client [close()]
```

UDP example in python

SERVER

```python
import socket  # Import the socket module for network programming

# 1. Create a UDP socket object for the server, SOCK_DGRAM specifies we are
creating a UDP socket as apposed to SOCK_STREAM for TCP sockets
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# 2. Bind the socket to an address and port
server_socket.bind(('localhost', 8080))

# 3. Receive data from the client (maximum 1024 bytes) along with the client's
address
data, addr = server_socket.recvfrom(1024)

# 4. Send a response back to the client at the address from which the data was
received
server_socket.sendto(b'Hello, Client!', addr)
```

CLIENT

```python
import socket  # Import the socket module

# 1. Create a UDP socket object for the client
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# 2. Send a message to the server at localhost on port 8080
client_socket.sendto(b'Hello, Server!', ('localhost', 8080))

# 3. Receive a response from the server (maximum 1024 bytes)
data, addr = client_socket.recvfrom(1024)

# 4. Close the client socket
client_socket.close()
```

Key Differences between TCP and UDP

Feature	TCP Socket	UDP Socket
Connection	Connection-oriented (three-way handshake)	Connectionless (no handshake, direct communication)
Reliability	Reliable (guarantees data delivery and order)	Unreliable (no guarantees for delivery or order)
Data Transfer Mode	Stream-oriented (data sent as a stream of bytes)	Message-oriented (each message is a discrete packet)
Flow Control	Yes (handles flow control and congestion)	No (no flow or congestion control)

Feature	TCP Socket	UDP Socket
Overhead	Higher (due to connection management, reliability)	Lower (simpler, with minimal overhead)

Addressing in Socket Programming

Network addresses are used to indeitify endpoints for communication
IP Address: Uniquely identifies a machine on the network
Port Number: Identifies the specific process or application running on the machine
When a client connects to a server, it needs to specify both IP and port number to ensure the data reaches the correct person