CS2105 Cheatsheet

for finals, by Hanming

Introduction: Computer Networks and the Internet

What is the Internet

- The Internet is a computer network that interconnects billions of computing devices throughout the world.
- All devices hooked up to the Internet are called **hosts** or **end systems**.
  - We can further split hosts into **clients** and **servers**.
- End systems are connected together by a network of **communication links** and **packet switches**.
  - **Routers** are packet switches used in the network core.
  - **Link-layer switches** are used in access networks.
- End systems access the Internet through **Internet Service Providers (ISPs)**.
- End systems run **protocols** that define the format and order of messages exchanged and the actions taken after messages are sent or received.

Network Edge

Access Network

The **access network** is the network that physically connects an end system to the first router on a path from that end system to any distant end system.

Examples to be added if there is time.

Network Core

The network core is the mesh of packet switches and links that interconnects the Internet's end systems.

Data is transmitted through the network in two ways: circuit switching and packet switching.

Circuit Switching

In circuit-switched networks, the resources needed along a path (buffers, link transmission rate) to provide for communication between the end systems are **reserved** for the duration of the communication session between the end systems. Commonly used in traditional telephone networks.

- Guaranteed constant rate performance.
- Idle during **silent periods** e.g. when someone stops talking, as there is no sharing allowed.

Packet Switching

End systems break down long messages into smaller chunks known as **packets**. These packets are then sent through communication links and packet switches. The **transmission rate** of a link is the number of bits that can be transmitted over it per second, also known as link capacity or link bandwidth.

- **Store-and-Forward**: Most packet switches use this. They must receive the entire packet from an inbound link before they can transmit the first bit onto an outbound link.
- **Forwarding Tables and Routing Protocols**: Routers use these to determine the link it should forward the packet on to.
- **Benefits**: Message segmentation reduces delay, allow for retransmission of smaller parts in case of corruption, and are easier to buffer + queue behind (more fair for other packets).

A Network of Networks

Hosts are connected to the Internet via an **access ISP**. This access ISP can provide wired or wireless connectivity. The access ISPs themselves must further be interconnected. This results in a network of networks, with a hierarchy of ISPs:

- **Regional ISP**: A ISP that the access ISPs in the region connect to. This may be multilevel.
- **Tier-1 ISP**: A ISP that regional ISPs connect to. They are the highest level of ISP.
- **Internet Exchange Point (IXP)**: When ISPs of the same level peer with each other, so they can skip the upstream ISP. They may thus build a IXP where multiple ISPs can peer together.
- **Content-Provider Networks**: A company's own network, e.g. Google.

Who Runs the Internet?

- Network Information Centre (NIC): IP address and Internet naming
- The Internet Society (ISOC): Internet related standards, education and policy
- The Internet Architecture Board (IAB): Issue and update technical standards regarding Internet protocols
- Internet Engineering Task Force (IETF): Protocol engineering, development and standardization arm of the IAB.

Delay, Loss and Throughput in Packet-Switched Networks

Total Nodal Delay

As a packet travels from the source to the destination, it will suffer from several types of delays at each node (host or router) that it passes along the way. The delays covered subsequently make up the **total nodal delay**:

- **Nodal Processing Delay**: Time required to examine the packet's header, check for bit-level errors and determine where to direct the packet. Typically in microseconds or less. After processing, the router directs the packet to the queue for the link to its next router.
- **Queuing Delay**: Time spent waiting to be transmitted onto the link. Depends on the number of earlier-arriving packets being queued or transmitted. Typically in microseconds to milliseconds.
- **Transmission Delay**: Given a packet of L bits and a link with transmission rate R bits/sec, we will have L/R seconds of transmission delay. Usually microseconds to milliseconds.
- **Propagation Delay**: Once a bit is pushed onto the link, it needs time to propagate to the next router. Depends on propagation speed of the link. Given d distance between the routers and s propagation speed of the link, we have d/s propagation delay. Usually in milliseconds for wide-area networks.

For $N-1$ routers between source host and destination host, we have:

$$d_{\text{end-to-end}} = N(d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}})$$

Packet Loss

As the queue for a router has finite capacity, a router will **drop** a packet when it arrives and the queue is full. The packet is thus **lost**.

Throughput

Another critical measurement of performance is throughput. The **instantaneous throughput** at any instant of time is the rate (in bits/sec) that a host is receiving information over the network. If let's say a file of F bits took T seconds for the host to receive, then the **average throughput** of the file transfer is F/T bits/sec.

Protocol Layers and Their Service Models

To provide structure to the design of network protocols, network designers organise protocols in layers. We are interested in the services that a layer offers to the layer above, i.e. the **service model** of a layer.

Internet Protocol Stack

1. **Application**: Where network applications and their application-layer protocols reside. A packet of information at this layer is called a **message**. Examples:
   (a) HTTP
   (b) SMTP
   (c) FTP
2. **Transport**: Transports application-layer messages between application endpoints. A transport-layer packet is called a **segment**.
   (a) TCP
   (b) UDP
3. **Network**: Moves network-layer packets known as **datagrams** from one host to another. Includes the celebrated IP protocol and other routing protocols.
4. **Link**: Moves a packet from one node to another. A link-layer packet is called a **frame**.
   (a) Ethernet
   (b) WiFi
5. **Physical**: Moves the individual bits within the link-layer frame from one node to another. Link-dependent and transmission medium-dependent.

Application Layer

Principles of Network Applications

Architectures

1. **Client-Server**
   - Server waits for incoming requests and provides required services to client. Can have numerous servers in a data center for scalability.
   - Client initiates contact with server and requests for service.
2. **Peer-To-Peer (P2P)**
   - No dedicated server, instead it relies on direct communication between pairs of intermittently connected hosts called **peers**.
   - Results in **self-scalability**, since each peer generates workload but also adds service capacity by distributing files.

Process Communication

- **Socket**: A software interface that a process uses to send messages into, and receive messages from the network. Generally a combination of an IP address and a port number.
- **IP Address**: A 32-bit quantity that uniquely identifies a host.
- **Port Number**: A 16-bit integer used to identify a receiving process running in a host.

Transport Services

We can broadly classify the services that a transport-layer protocol can offer to applications along four broad dimensions.

1. **Reliable Data Transfer**: Some applications require data to be sent correctly and completely to the receiver, while others are **loss-tolerant applications**.
2. **Throughput**: Some **bandwidth-sensitive applications** may need some guaranteed throughput of r bits/sec, while other **elastic applications** can make use of as much or as little throughput as available.
3. **Timing**: Real-time applications generally require low delays to be effective, while others may not have tight constraints on end-to-end delays.
4. **Security**: Some transport protocols can encrypt all data being sent, etc.

Transport Services for the Internet

There are two main protocols for the Internet:

1. **Transmission Control Protocol (TCP)**
   - Reliable data transfer
   - Connection-oriented service: A handshaking procedure is done before the application-level messages begin to flow
   - Flow control: Sender won't overwhelm receiver
   - Congestion control: Throttle sender when network is overloaded
   - Security: Can be enhanced at the application layer with **Secure Sockets Layer**
   - Does not provide: Timing and throughput guarantee
2. **User Datagram Protocol (UDP)**
   - Unreliable data transfer
   - Connectionless: No handshaking is done
   - No flow control
   - No congestion control
   - Does not provide: Timing and throughput guarantee, security

Application-Layer Protocols

An application-layer protocol defines:

- **Types** of messages exchanged, e.g. request and response messages.
- **Syntax** of message types, e.g. fields and how they are delineated.
- **Semantics** of the fields, i.e. what the information means.
- **Rules** for when and how to send a message and respond to messages.

There are open protocols defined in RFCs (Request for Comments), such as HTTP, while there are some proprietary protocols.

The Web and HTTP

Web Jargon

- **Web Page**: A document consisting of a base HTML file and several referenced objects.

- **Objects**: A file, such as a HTML file, JPEG image, Java applet, etc. that is addressable by a single URL.
- **Hostname**: Example: http://www.comp.nus.edu.sg
- **Path name**: Example: /∼cs2105/img/doge.jpg

Overview

The **HyperText Transfer Protocol** is the Web's application-layer protocol.

- **Client-server model**: Client is the browser that requests, receives and displays Web objects, the server is a Web server that sends objects in response to objects.
- **Stateless**: The server maintains no information about the clients.
- **Over TCP**: Reliable.

Three-Way Handshake

1. Client sends a small TCP segment to the server asking for TCP connection.
2. Server acknowledges and responds with a small TCP segment.
3. Client acknowledges back and sends this acknowledgement along with the request message.

Non-Persistent HTTP: HTTP 1.0

Let's say we want to get http://www.comp.nus.edu.sg/∼cs2105/demo.html.

1. (Client) Initiates TCP connection to server http://www.comp.nus.edu.sg on **port number 80**, which is the default port number for HTTP.
2. (Server) Accepts TCP connection request from client and replies.
3. (Client) Sends an HTTP request message to the server via the socket, the request message having the path name /∼cs2105/demo.html.
4. (Server) Receives request, retrieves /∼cs2105/demo.html from its storage, encapsulates it in a HTTP response message, and sends it to the client via the socket.
5. (Server) Tells TCP to close the TCP connection, but TCP doesn't actually terminate until it knows for sure that the client has received the response message intact.
6. (Client) Receives the response message. TCP connection terminates. The client extracts the file from the response message, examines the HTML file, and finds references to JPEG object.
7. Repeat steps 1 to 5 for each JPEG object. Browsers now tend to open **parallel TCP connections** to fetch referenced objects.

The **round-trip time (RTT)** is the time taken for a **small packet** to travel from server and then back to the client. It does not include the transmission delay!

$$\text{RTT} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{prop}}$$

The first two steps of the handshaking takes one RTT, then the third step + requesting for the object itself takes one RTT + transmission time for the entire file to be received. Thus, non-persistent HTTP takes **two RTTs + file transmission time** per object requested.

Persistent HTTP: HTTP 1.1

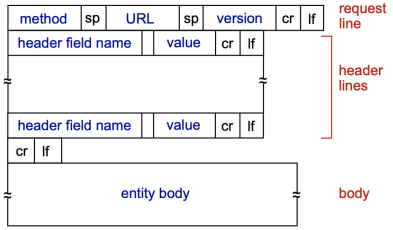
In persistent HTTP, the server leaves the TCP connection open after sending a response. Subsequent requests and responses can be sent over the same connection. The connection is closed only when it isn't used for a certain time.

Browsers also use **pipelining**, i.e. when they encounter multiple objects referenced, they will send requests for those objects back to back, allowing for multiple requests and responses to be interleaved in the same connection.

HTTP Request Format

Here's a sample request message:

```
GET /index.html HTTP/1.1\r\n
Host: www.example.org\r\n
Connection: keep-alive\r\n
...
\r\n
```



- HTTP 1.0 Methods
  - `GET`: Gets an object.
  - `POST`: Posts form data.
  - `HEAD`: Gets the header without the body.
- HTTP 1.1 Methods
  - `GET`, `POST`, `HEAD`
  - `PUT`
  - `DELETE`
- Host: Used by Web proxy caches
- Connection: Value is close if it's the final request
- Blank line at end of header: Indicates end of header lines

HTTP Response Format

Here's a sample response message:

```
HTTP/1.1 200 OK\r\n
Date: Wed, 23 Jan 2019 13:11:15 GMT\r\n
Content-Length: 606\r\n
Content-Type: text/html\r\n
...
\r\n
data data data data data ...
```

- `200 OK`: Request succeeded and object is later in this message.
- `301 Moved Permanently`: Requested object moved and new location is later in this message.
- `403 Forbidden`: Server declines to show the requested object
- `404 Not Found`: Request object not found on this server

Cookies

As HTTP is stateless, cookies help to carry state.

1. Server responds with a `Set-Cookie` header field
2. Cookie is stored on user's end and will be sent via the `Cookie` header field in future messages
3. Server checks the cookie for each request against their backend database

Conditional GET

The web also heavily uses caching (not covered in CS2105), which reduce response times but may have the issue of objects in cache going stale. We thus have a conditional GET, which is when a GET request has a `If-Modified-Since: <date>` header line.

If it's been modified, it will send a `200 OK` and the object, else it will send `304 Not Modified` and no object.

DNS: The Internet's Directory Service

Humans generally identify a host by its **hostname**, e.g. www.facebook.com, but it provides little information about the host's location within the Internet. Thus, hosts are also identified using **IP addresses**, e.g. 157.240.7.35. The **Domain Name System (DNS)** thus helps to translate between the two.

DNS Record

DNS servers store **resource records**:

```
(Name, Value, Type, TTL)
```

1. `Type = A`: `Name` is hostname and `Value` is the IP address.
2. `Type = NS`: `Name` is a domain and `Value` is the hostname of an authoritative DNS server that knows how to obtain the UP addresses for hosts in the domain.
3. `Type = CNAME`: `Value` is a canonical hostname for the alias hostname `Name`.
4. `Type = MX`: `Value` is the canonical name of a mail server that has an alias hostname `Name`.

DNS Hierarchy

DNS servers form a hierarchy to have a scalable way to distribute mappings. There are three classes of DNS servers:

- **Root servers**: Provides IP addresses of the TLD servers.
- **Top-level domain (TLD) servers**: For each top-level domain e.g. com, org, net, and all of the country top-level domains, there is a TLD server that provides the IP addresses for authoritative DNS servers.
- **Authoritative servers**: Every organisation with publicly accessible hosts on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses.

Outside of the hierarchy, we also have **local DNS servers**. These are servers belonging to ISPs (also called default name servers). They speed up DNS queries by providing caching.

DNS Caching

Once a DNS server learns of a mapping, it caches it, and will return this as a **non-authoritative** answer the next time it is queried. This allows local DNS servers to skip querying the root server and the entire hierarchy.

Cached entries may also expire after some time. This is based on the `TTL` (time to live) value (in seconds) of the resource record. Usually it is 2 days.

DNS Name Resolution

Note that DNS runs over **UDP**. There are two ways for the local DNS server to get the required mapping:

1. **Iterative query**: Local DNS server goes back and forth with each server in the hierarchy.
2. **Recursive query**: The server that receives the query will be the one doing the next query, so the chain will look like: Local DNS → Root DNS → TLD DNS → Authoritative DNS.

DNS Cache Poisoning

DNS cache poisoning (a kind of DNS spoofing) is a computer hacking attack, whereby rogue DNS records are introduced into a DNS resolver's cache, causing the name server to return an incorrect IP address, diverting traffic to the attacker's computer (or any other computer). For example, DDoS (Distributed Denial of Service Attack) on a particular machine can be achieved via DNS cache poisoning.

Socket Programming: Creating Network Applications

Using UDP

As mentioned before, UDP is connectionless, so the client needs to explicitly attach the destination IP address and port number to each packet. The receiver is also able to extract the sender IP address and port number from the received packet. Uses `SOCK_DGRAM` sockets.

1. (Server) Create `serverSocket`, `port = x`.
2. (Client) Create `clientSocket`.
3. (Client) Create datagram with server IP and `port = x`, and send datagram via client socket.
4. (Server) Read UDP segment from `serverSocket`.
5. (Server) Write reply to `serverSocket`, specifying client address and port number.
6. (Client) Read datagram from `clientSocket`.
7. (Client) Close `clientSocket`.

Using TCP

TCP is a connection-oriented protocol. The server will need to be ready and have a welcoming socket (think of a main door). Upon receiving the TCP connection request, a new socket (or door) is opened just to communicate with that client. The two hosts can just keep communicating via that connection until one side closes it. Uses `SOCK_STREAM` sockets.

1. (Server) Create `serverSocket`, `port = x`.
2. (Server) Wait for incoming connection request, which gives it a new `connectionSocket` once received, (Client) Create `clientSocket` and connect to server IP, `port = x`.
3. (Client) Send request via `clientSocket`.
4. (Server) Read request from `connectionSocket`.
5. (Server) Write reply to `connectionSocket`.
6. (Client) Read reply from `clientSocket`.
7. (Server) Close `connectionSocket` and wait go back to step 2 to continue waiting for new connection request.
8. (Client) Close `clientSocket`.

Transport Layer

Introduction and Transport-Layer Services

Transport layer senders breaks up application messages into **segments**, then passes them to the network layer. The receivers reassembles segments into messages and pass them to the application layer.

Relationship Between Transport and Network Layers

Transport layer takes care of logical communication between **processes**, while network layer takes care of logical communication between **hosts**. To some extent, the services that a transport protocol can provide are constrained by the service model of the underlying network-layer protocol. However, it is possible to offer certain services that are not provided by the underlying protocol, e.g. reliable data transfer over an unreliable underlying network.

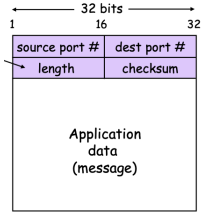
Connectionless Transport: UDP

Connectionless Multiplexing & Demultiplexing

A UDP socket is fully identified by a two-tuple consisting of a destination IP address and a destination port number.

- **Multiplexing**: UDP gathers data from processes, form transport-layer segments that include the application data, source port number and the destination port number, and passes them to network layer.
- **Demultiplexing**: Examines the destination port number of the segment and passes the segment to the socket identified by it.

UDP Segment



The length includes both the header and the body.

UDP Checksum

Allows for error detection, but not correction. It is needed as not all links along the way may provide error-checking, and there may be bit errors when segments are stored in a router's memory.

1. Treat UDP segment as a sequence of 16-bit integers.
2. Apply binary addition on every 16-bit integer (checksum field is currently 0).
3. Carry (if any) from the most significant bit will be added to the result.
4. Compute 1's complement to get UDP checksum.

Principles of Reliable Data Transfer

As mentioned before, the network layer provides host-to-host, best-effort and unreliable communication. We thus need to build a reliable transport layer protocol on top of unreliable communication.

- Packet corruption
- Packet loss
- Packet reordering
- Packet (arbitrarily long) delay

rdt 1.0

Assumption: Underlying channel is perfectly reliable.

1. (Sender) Make packet and send packet to receiver.
2. (Receiver) Extract packet and deliver data to application.

rdt 2.0

Assumption: Underlying channel may flip bits i.e. corruption. We use the **stop and wait** protocol.

1. (Sender) Make packet and send packet to receiver.
2. (Receiver) Check if packet is corrupt. If not corrupt, deliver data to application and send ACK to client, else just send NAK to client.
3. (Sender) If received NAK, resend the same packet, else go to step 1 for next packet.

Problems: If ACK or NACK are corrupted, there is no guaranteed way to recover. If we simply resend the packet, the receiver will not know it's a duplicate.

rdt 2.1

Assumption: Same as rdt 2.0 - corruption. In addition to what is covered in rdt 2.0, we now add a sequence number to the packet. This number alternates between 1 and 0. Duplicates are thus detected using sequence number.

1. (Sender) Sends packet 0.
2. (Receiver) Sends ACK or NAK.
3. (Sender) If NAK or corrupted message received, resend. Else send packet 1.
4. (Receiver) If duplicate packet 0 is received, drop it and reply ACK. Else continue as per usual.

rdt 2.2

Assumption: Same as rdt 2.0 and 2.1 - corruption. In addition to what is covered in rdt 2.1, we now explicitly include the sequence number of the packet being acknowledged, removing the need for a NAK. From the sender's perspective, we basically resend current packet if a duplicate ACK is received.

1. (Sender) Sends packet 0.
2. (Receiver) Sends ACK1 if corrupted, else ACK0.
3. (Sender) So long the ACK is not that of the packet sent earlier, keep resending that packet.
4. (Receiver) Keep sending ACK1 if **corrupted**, else if duplicate or is correct, send ACK0.
5. Repeat for packet 1.

rdt 3.0

Assumption: Corruption, packet loss and packet delays, but no re-ordering (i.e. the order sent is the order received). We need to add sender timeouts to rdt 2.2 to handle packet loss and delays. One important difference is that sender **does not respond to** duplicate ACKs. To detect packet loss, the timer is used.

1. (Sender) Sends packet 0.
2. (Receiver) If received and is correct, send ACK0. Else if received and is corrupted, send ACK1.
3. (Sender) If no response is heard by a certain time (either due to delay or loss on either side), resend packet 0. Else if ACK1 received, also resend packet 0. Only if ACK0 is received do we move on to next packet.
4. (Receiver) Same as step 2, except if packet 1 has been received before and it's a duplicate, we also send ACK0.
5. (Sender) Even if ACK0 is received twice, it will not respond to the second one. It will let the timer started since the first response do the job.

rdt 3.0 Peformance

The utilisation rate of the sender is low. If let's say the RTT is 30ms, and transmission is 0.008ms (8000 bits sent over 1Gbps link rate), we are sending 8000 bits every 30.008ms. Utilisation rate is 0.027%.

We can thus do **pipelining**, which is to send multiple packets without waiting for acknowledgements. The number of packets sent at once is called the **window size**.

- **Sequence numbers**: The range needs to be increased since now there are multiple packets being sent at once.
- **Buffer**: We need some buffering mechanism at both the sender and receiver.

Go-Back-N

- Think of it as a sliding window that slides forward only when an ACK is received for the leftmost packet in the window.
- Requires k bits in packet header for 2^k sequence number.
- Sender keeps a timer for the oldest unACKed packet. Only one timer is kept.
- Receiver only accepts ACK packets that arrive in order. Discards out of order packets and ACK the last in-order sequence number. This is also known as cumulative ACK, i.e. ACK m means all packets up to packet m have been received.

Example:

1. Sender sends packets 0, 1, 2, 3, and waits. Packet 2 is LOST. *Window: [0 1 2 3] 4 5 6 7.*
2. Receiver receives 0, sends ACK0, receives 1, sends ACK1, receives 3, discards and resends ACK1.
3. Sender receives ACK0, sends packet 4. *Window: 0 [1 2 3 4] 5 6 7.*
4. Sender receives ACK1, sends packet 5. *Window: 0 1 [2 3 4 5] 6 7.*
5. Receiver receives packet 4, discards, sends ACK 1.
6. Receiver receives packet 5, discards, sends ACK 1.
7. Packet 2 timeout! Sender resends packets 2, 3, 4, 5.

Selective Repeat

- Receiver individually acknowledges all correctly received packets, but buffers out-of-order packets, which will eventually be sent to application layer.
- Sender maintains a timer for each unACKed packet. When timer expires, retransmit only that unACKed packet. Multiple timers are kept
- Sender marks packets n as received. It toggles window base by sliding to the smallest unACKed packet.
- ACK m simply means packet m has been received, but has no implication on the receipt of other packets.

Example:

1. Sender sends packets 0, 1, 2, 3, and waits. Packet 2 is LOST. *Window: [0 1 2 3] 4 5 6 7.*
2. Receiver receives 0, sends ACK0, receives 1, sends ACK1, receives 3, buffers and sends ACK3.
3. Sender receives ACK0, sends packet 4. *Window: 0 [1 2 3 4] 5 6 7.*
4. Sender receives ACK1, sends packet 5. *Window: 0 1 [2 3 4 5] 6 7.*
5. Sender receives ACK3, but cannot send packet 6 until ACK2 is in.
6. Receiver receives packet 4, buffers, sends ACK 4.
7. Receiver receives packet 5, buffers, sends ACK 5.
8. Packet 2 timeout! Sender resends packet 2.
9. Receiver receives packet 2, sends packet 2 and buffered packets 3, 4, 5 up to application, responds with ACK2.

Connection-Oriented Transport: TCP

Connection-Oriented Multiplexing & Demultiplexing

A TCP socket is fully identified by a four-tuple: (source IP address, source port number, destination IP address, destination port number). The connection is **duplex**, i.e. two-way.

- **Multiplexing**: TCP gathers data from processes, form transport-layer segments that include the application data and the 4-tuple above, and passes them to network layer.
- **Demultiplexing**: When creating the connection socket, the server already has noted the 4-tuple that identifies it. Any subsequent packets will be directed, or demultiplexed, to the appropriate socket using those 4 values.

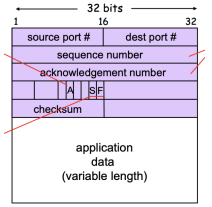
TCP Buffers

During the handshake, TCP will set aside buffers on both sides.

- **Send buffer**: A buffer on the sender side of the connection where application data will be stored. From time to time, TCP will grab chunks from the buffer and pass it to the network layer.
- **Receive buffer**: Like the send buffer, it is specific to a connection, but it's on the receiver side. Received segments will have their data extracted and placed here, and application will read the stream of data from this buffer.

TCP Segment

The **maximum segment size (MSS)** depends on the **maximum transmission unit (MTU)**, the largest link-layer frame size, but generally MSS is **1460 bytes**, as MTU is 1500 bytes for Ethernet and PPP link-layer protocols. The remaining 40 bytes are split half-half for the TCP header and the IP header.



The checksum computation is the same as UDP - 1s complement. Flags:

- `ACK` bit: Indicates whether the acknowledgement field is valid.
- `SYN` bit: Used to signal connection setup.
- `FIN` bit: Used to signal connection teardown.

TCP Sequence Numbers and ACK

- **Sequence Number**: The sequence number is the byte number of the first byte of data in the segment, with respect to the whole file.

  - If we have a file of 500,000 bytes, and MSS is 1,000 bytes, then the sequence numbers will be 0, 1,000, 2,000, and so on.
  - **Random initial sequence number**: To minimise the probability of some segment from a previous connection being mistaken as from the current connection.
- **ACK Number**: This is the sequence number of the next expected packet.
  - Let's say we are responding to the above packet of MSS = 1,000 with sequence number 0. We will send back ACK number = 1,000, since we expect the next packet to have sequence number 1,000.
  - **Cumulative ACK**: The ACK number will be the byte number of the first missing byte in the stream.
  - **Piggy-backing**: We can also send data together with an ACK. The ACK will be processed as long as the ACK bit is 1.
  - **Delayed ACK**: Receiver can wait up till 500ms for the next segment before sending an ACK.

TCP Duplicates

There is no specification on how duplicates should be handled (i.e. discarded or buffered). The approach taken in practice is buffering.

TCP Timeout

TCP also uses timeout, much like rdt 3.0. The recommended TCP timer management procedures use only a single retransmission timer. This timer is associated with the oldest unacknowledged segment. This timeout interval is based on an estimated RTT. α is generally 0.125 and β is generally 0.25.

$$\text{EstimatedRTT} = (1-\alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}$$

$$\text{DevRTT} = (1-\beta) \times \text{DevRTT} + \beta \times |\text{SampleRTT} - \text{EstimatedRTT}|$$

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$$

TCP Fast Retransmission

We can make use of some heuristics. If the sender receives 3 duplicate ACKs, then the sender will perform a **fast retransmit** and retransmit the missing segment even before that segment's timer expires.

TCP Handshake & Goodbye

The connection establishment is as such:

1. (Client) Send a special TCP segment. This segment:
   - Has no application data.
   - Has a random initial sequence number, `client_isn`
   - `SYN` bit 1.
2. (Server) Once received, it extracts the TCP segment from the datagram, allocates the TCP buffers and variables to the connection, and replies with a connection-granted segment (`SYNACK` segment). This segment:
   - Has no application data
   - `SYN` bit 1.
   - ACK number = `client_isn` + 1.
   - `ACK` bit 1.
   - Has a random initial sequence number, `server_isn`.
3. (Client) Once received, client also allocates buffers and variables to the connection. The third segment:
   - Can carry client-to-server data in the payload
   - `SYN` bit 0.
   - ACK number = `server_isn` + 1.
   - `ACK` bit 1.

The disconnection is as such:

1. (Client) Sends a special TCP (shutdown) segment to the server, asking to close the connection:
   - Has no application data.
   - `FIN` bit 1.
   - Sequence number u.
2. (Server) Sends back an ACK segment, and client can no longer send any info over but can receive info:
   - ACK number = $u + 1$
   - `ACK` bit 1.
3. (Server) Sends its own shutdown segment, like step 1.
4. (Client) Acknowledges the shutdown segment, like step 2.

The Network Layer: Data Plane

The role of the network layer is deceptively simple - to move packets from a sending host to a receiving host.

The **data plane** functions are the per-router functions in the network layer that determine how a datagram arriving on one of the router's input links is forwarded to one of the output links. The **control plane** function of the network layer is the network-wide logic that controls how a datagram is routed among routers.

Overview of Network Layer

Forwarding vs Routing

- **Forwarding**: The moving of an incoming packet to the appropriate output link.
- **Routing**: The calculation of a path taken by packets as they flow from a sender to receiver.

Network Services Model

This model defines the characteristics of end-to-end delivery of packets between sending and receiving hosts. Here are some possible services:

- **Guaranteed delivery**: Packets will eventually arrive at the destination.
- **Guaranteed delivery with bounded delay**: Packets will eventually arrive at the destination within some specified host-to-host delay bound.
- **In-order packet delivery**: Packets arrive in the order they were sent.
- **Guaranteed minimal bandwidth**: Emulate the behaviour of a transmission link of a specified bit rate.
- **Security**: Encrypt at source and decrypt at destination.

The Internet's network layer provides a single service, known as **best-effort service**.

What's Inside a Router?

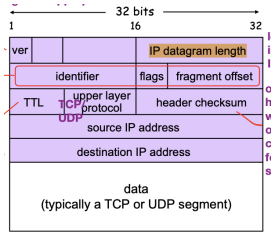
Fortunately (or unfortunately), we don't actually need to know what's inside a router, except that it contains a **forwarding table**. This forwarding table is computed by the routing processor and lets us know how to forward packets.

Destination-based Forwarding

Every packet will stop at a router, let the router know its final destination, and the router will let the packet know which path to take. This is unlike generalised forwarding, which forwards based on e.g. type of packet, source of packet etc.

The Internet Protocol (IP): IPv4 & Addressing

IPv4 Datagram Format



1. **Datagram length**: Length of the IP datagram, including 20 bytes from the header.
2. **Identifier, flags, fragment offset**: To support fragmentation and reassembly
3. **Time to live**: To prevent datagrams from circulating forever.
4. **Upper layer protocol**: Only used at final destination, to determine if it's UDP or TCP (for Internet).
5. **Checksum**: Only for header, while for UDP/TCP it's the entire segment. Also uses 1s complement.

IPv4 Datagram Fragmentation

Different link layer frames can carry different maximum amount of data, also called **maximum transmission unit (MTU)**. This MTU encapsulates the IP datagram and includes the 20 byte header. A large incoming datagram may thus be sent out as multiple smaller datagram.

1. The router will need to decide if the datagram needs to be fragmented.
2. **Identifier**: All of the smaller datagrams will have the same identifier as the original larger datagram.
3. **Fragmentation flag**: Set to 1 for all fragments except the last, which will be 0.
4. **Offset**: The offset of the first byte from the start of the original larger datagram. This offset is in units of 8 bytes, i.e. offset = 60 means the first byte is the 480th byte.
5. **Reassembly**: Will only be done by the destination host. Routers in between will not reassemble!

IP Addresses

An IP address is a 32 bit integer represented by either binary or decimal. It's usually processed in 4 parts of 8 bits. A host can get an IP address either by manual configuration by system admin or by automatic assignment by a **Dynamic Host Configuration Protocol** server.

Each network interface has an IP address. A host usually has one or two network interfaces, while a router may have multiple.

Special IP Addresses

- **0.0.0.0/8**: Non-routable meta-address for special use
- **127.0.0.0/8**: Loopback address, i.e. localhost.
- **10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16**: Private addresses that can be used without any coordination with IANA or an Internet registry.
- **255.255.255.255/32**: Broadcast address.

Dynamic Host Configuration Protocol (DHCP)

DHCP generally assigns hosts a temporary IP address that will be different each time the host connects to a network (unless manually configured by admins to be constant).

1. Host broadcasts a DHCP discover message.
   - Sent within a **UDP** packet.
   - Sent to **port 67**.
   - Source port is **68**.
   - Contains a transaction ID.
   - Encapsulated in IP datagram with destination IP address of 255.255.255.255.
   - MAC address of link layer frame is FF-FF-FF-FF-FF-FF.
   - Also with source IP address of 0.0.0.0.
2. DHCP server responds with a DHCP offer message, broadcasted to all nodes
   - Destination IP is again 255.255.255.255.
   - Destination port is 68.
   - MAC is FF-FF-FF-FF-FF-FF.
   - Source IP is the server's IP
   - Source port is 67
   - Client might be able to actually choose between offers from multiple servers
   - Contains:
     (a) Transaction ID of the received discover message
     (b) Proposed IP address for the client
     (c) Subnet mask
     (d) Address of the first-hop router (default gateway)
     (e) Address of local DNS server
     (f) IP address lease time (how long the IP address will be valid)
3. Host responds to a selected offer with a DHCP request message, echoing back the configuration parameters.
   - Source IP address is still 0.0.0.0.
   - Destination IP address is still 255.255.255.255.
4. DHCP server responds with a DHCP ACK message, confirming the requested parameters.
   - Destination IP address is still 255.255.255.255.

Subnet

An IP address consists of two parts:

$<n$ bits for subnet prefix$> <32-n$ bits for host ID$>$

- **Subnet**: Network of directly interconnected host.
- Same network prefix and can physically reach each other without a router.
- Connect to the outside world through a router.
- Generally, this subnet will be bought or rented by an organisation from an ISP.
- This ISP will allocate from the address block that it has that it obtained from ICANN (Internet Corporation for Assigned Names and Numbers).

Classless Inter-Domain Routing

The two parts above, $a.b.c.d/x$, is called CIDR, and generalises the notion of subnet addressing. An organisation is often assigned a block of contiguous addresses with a common prefix. Only the x leading bits will be considered by routers outside the organisation's network.

Subnet Mask

The subnet mask is obtained by setting the subnet prefix bits to 1 and all other bits to 0. For example, 200.23.16.42/23 becomes 255.255.254.0.

Hierarchical Addressing

We can have an efficient way of routing by structuring the IP addresses as a hierarchy. For example, a ISP may have address block 200.23.16.0/20. It can further split it up to support multiple organisations, e.g. 200.23.16.0/23, 200.23.18.0/23, and so on.

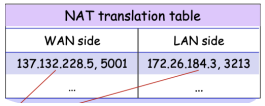
Longest Prefix Match

There may be cases, such as when organisations switch ISPs but wish to maintain their IP address blocks, where we may have seemingly conflicting subnet prefixes. For example, one organisation has 200.23.16.0/20, while the other has 200.23.18.0/23. The first 20 bits for both are the same.

The solution is to choose the one with the longer match. If an IP address matches all 23 bits for the latter organisation, then the packet will be forwarded to that organisation, else it will be forwarded to the latter.

Network Address Translation (NAT)

We unfortunately cannot allocate unlimited contiguous address ranges, and it's neither scalable nor practical to expect e.g. all homeowners to know how to manage IP addresses. The following will be done via a NAT-enabled router.



- **Private Addresses**: As stated before, there are private IP address ranges that will not be used on the public Internet. Address spaces within these private ranges can thus been used for hosts on private networks.
- **Network Address Translation**: Translates datagrams containing these private addresses to use the router's IP instead.
- **Single device to the outside**: To the public Internet, it looks like the all the devices are actually one device, which is the NAT-enabled router itself.
- **NAT Translation Table**: For each host in the private network, a specific port on the router will be used for it. It stores the IP address and port number for the WAN and LAN sides.
- **Security**: Hosts inside a network are not explicitly addressable.
- **Convenience**: Can change ISP without changing addresses of hosts in the local network.

The Network Layer: Control Plane

We have seen how the forwarding table specifies the local data-plane forwarding behaviour of a router. We will now see how this table is computed and maintained.

Routing Algorithms

As learnt before, the Internet is a hierarchy of **autonomous systems (AS)**. Routing is similarly done via a hierarchy.

Intra-AS vs Inter-AS Routing

- **Intra-AS Routing**: Finding a good path between two routers in an AS.
  - Single admin, so no policy decisions.
  - Focused on performance.
  - Can be viewed as a graph, where routers are nodes, physical links are edges, and we assign some cost to the edges, e.g. constant, based on congestion, inverse of bandwidth, etc.
- **Inter-AS Routing**: Handles interfaces between ASes.
  - Admin wants control over who and what is routed.
  - Policy dominates over performance.

Centralised Routing Algorithms

- Compute least-cost path from source to destination based on complete, global knowledge about the network, e.g. using Djikstra's algorithm.
- For example, routers may periodically broadcast link costs to each other.
- These are also called **link-state algorithms**.

Decentralised Routing Algorithms

- Each node begins with only knowledge of its direct links.
- Iteratively exchange information with neighbours and calculate the least cost path to a destination.
- We will be studying **distance-vector algorithms**.

Distance-Vector Algorithms

This algorithm is based on the Bellman-Ford equation:

$$d_x(y) = min_v\{c(x,v) + d_v(y)\}$$

where $c(x,v)$ is the cost of the edge from x to v, and $d_v(y)$ is the minimum cost path from v to y.

1. First initialise the shortest path to all direct neighbours with the cost of the edge, and to all other routers as cost ∞.
2. We then let all our neighbours know of all the shortest paths we are currently aware of.
3. We then wait for our neighbours to let us know of their shortest paths OR if the link cost to any neighbour changes.
   (a) For each shortest path from a neighbour, we check if our cost to that neighbour + that neighbour's shortest path is shorter than our shortest path. If so, update our shortest path and broadcast this change to all neighbours.
   (b) If it's a change to the link cost to our neighbour, similarly check if it results in shorter paths, and if so, broadcast all changed shortest paths.

The final shortest paths or distance vectors will form our forwarding table, i.e. if we can get to y quickest via v, then all packets to y will be forwarded to v.

Routing Information Protocol

- **Hop Count**: Metric used as cost, insensitive to congestion.
- **Periodic exchange**: Exchange routing table every 30 seconds over **UDP port 520**.
- **Self-repair**: If no update from neighbour for 3 minutes, assume neighbour has failed.

Internet Control Message Protocol (ICMP)

This is a protocol used by routers and hosts to communicate network-layer information, typically error reporting.

- **Above IP**: Often considered as part of IP, but architecturally it lies just above IP, as ICMP messages are carried inside IP datagrams. ICMP will be specified as the upper-layer protocol.
- **Demultiplexing**: When the router sees that ICMP is the upper-layer protocol, it will demultiplex into ICMP.
- **Format**: The header is right behind the IP header, and contains the **Type**, **Code**, **Checksum**. The message also contains the header and first 8 bytes of the datagram that caused this error.
- **Traceroute**: Implemented using ICMP. Traceroute spams small packets with unlikely UDP port numbers and observes the ICMP responses.
- **Expired TTL**: When TTL in the IP header reaches zero, the router will discard the IP datagram and send a ICMP error message back to source host.

ICMP Message Types

1. Type 0, Code 0: Echo reply to ping
2. Type 3, Code 1: Destination host unreachable
3. Type 3, Code 3: Destination port unreachable
4. Type 8, Code 0: Echo request (ping)
5. Type 11, Code 0: TTL expired
6. Type 12, Code 0: IP header bad

The Link Layer and LANs

Introduction to the Link Layer

The link layer takes care of how packets are sent across **individual links**. We call all devices that runs a link-layer protocol a **node**, and all communication channels connecting adjacent nodes as **links**. Different links may run different protocols. Over a given link, a transmitting node encapsulates the datagram in a **link-layer frame** and transmits the frame onto the link.

Services Provided

- **Framing**: Encapsulation of a network-layer datagram within a link-layer frame.
- **Link access**: A medium access control protocol serves to coordinate the frame transmissions of many nodes over a single link.
- **Reliable delivery**: Usually used on error-prone links (e.g. wireless) instead of low bit-error links (e.g. fiber). Goal is to correct an error locally instead of forcing an end-to-end retransmission by the transport or application layer protocol.
- **Error detection and correction**: Errors caused by signal attenuation or noise should be caught as soon as possible to prevent unnecessary transmission of incorrect frames. Some protocols can even identify and correct errors.

Network Adapter

A **network adapter**, also called a **network interface card (NIC)**, is a single, special-purpose chip that implements the link-layer services above. Thus, many of the services are implemented in hardware, and is the place in the protocol stack where software meets hardware.

Error-Detection and -Correction Techniques

Generally, we have an error detection and correction (EDC) field to the link layer frame, which protects not just the datagram, but also link-level addressing information, etc. The larger the field, the better the detection and correction, but also the larger the overhead.

Parity Checks

- **Single Bit Parity**: We have 1 parity bit for the data.
  - **Even Parity Scheme**: We choose the bit to make the total number of 1s even.
  - **Odd Parity Scheme**: Same but we make the number odd.

  Two-Dimensional Parity: We divide the data into rows and columns, and repeat the above but have parity bits for each column and each row.
  - Can detect and correct single bit errors in data.
  - Can detect two-bit errors.
- **Multi-Dimensional Parity**: This can go for n dimensions. An n-dimensional parity scheme is only guaranteed to correct up to $n/2$ errors

Cyclic Redundancy Check (CRC)

We can think of this as long division but with division being replaced by a bitwise XOR operation. Generally done by hardware, so very fast.

- D: d-bit data, which is also the dividend.
- G: Generator of $r+1$ bits, which is also the divisor.
- R: r-bit CRC, which is also the remainder.

The resultant $d+r$ bits is "divisible" by G, so the receiver can check for a zero remainder.

Multiple Access Links and Protocols

Types of Network Links

1. **Point-to-point link**: Sender and receiver connected by a dedicated link. No need for multiple access control.
2. **Broadcast link**: Multiple nodes connected to the same shared broadcast channel. When any one node transmits a frame, all other nodes in the channel receives a copy. We need a **Multiple Access Protocol** to prevent frame collisions.

Channel Partitioning Protocols

1. **Time Division Multiple Access**: Each node gets a fixed length time slot, where length = frame transmission time. This repeats in rounds. Unused slots go idle.
2. **Frequency Division Multiple Access**: Channel spectrum is divided into frequency bands, and each node is assigned one band. Bandwidth has thus decreased, thus transmission is slower. Unused transmission time in frequency bands go idle.

Taking-Turns Protocols

1. **Polling**: A master node invites each of the other nodes to transmit in turns. There's minor polling overhead but the greatest concern is a single point of failure, the master node.
2. **Token Passing (Token Ring) / Round Robin**: Control token is passed from one node to the next sequentially. There is overhead for the token and a single point of failure as well, which is the token.
   - Even if only a few of the nodes have data to send, it can still be quite efficient.

Random Access Protocols

Generally these protocols specify how to detect collisions and how to recover from these collisions. The benefit is there there is no need for centralised coordination, thus no single point of failure.

1. **Slotted ALOHA**
   - Assume all frames are of the same size, and we split the time into slots of equal length, where length = time to transmit 1 frame.
   - A node will only transmit at the start of a slot.
   - Each node listens to the channel while transmitting. If a collision occurs, it retransmits in each subsequent slot with probability p until success.
   - p depends on network congestion.
2. **Pure (Unslotted) ALOHA**
   - Like ALOHA but no slots nor synchronisation, just transmit when there's a fresh frame.
   - Chance of collision increases, as now it can collide with frames both in front and behind $(t_0 - 1, t_0 + 1)$.
3. **Carrier Sense Multiple Access (CSMA)**
   - Basically sense if the channel is idle, if so, transmit frame.
   - Still can collide when both nodes sense that a channel is idle.
   - This is especially since there is propagation delay and nodes may be far apart. Transmission also does not stop despite collision being detected.
4. **Carrier Sense Multiple Access / Collision Detection (CSMA/CD)**
   - Basically same as CSMA except the moment a collision is detected, the node stops its transmission.
   - The node then retransmits after some random amount of time.
5. **Minimum Frame Size**: For CSMA and CSMA/CD above, we need some minimum frame size so that collisions can always be detected. Ethernet has a minimum size of **64 bytes**.
6. **Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA)**
   - Collision detection can be hard for wireless LANs, as energy levels drop too quickly.
   - **Hidden Node Problem**: When two nodes cannot detect each other but a node in-between encounters a collision.
   - As such, an ACK is required from the receiver as well.

Bit Times

A common unit used in multiple access protocols is **bit times**. Bit time, also called bit transmision time, is the time it takes to transmit 1 bit. Often, it will be used as such: propagation delay is equals to 800 bit times. This means the propagation delay is equals to the time it takes to transmit 800 bits onto the link.

Exponential Backoff

If we keep encountering collisions, we would naturally want to wait slightly longer before trying again. After the m^{th} collision, we want to choose K at random from $\{0, 1, \cdots, 2^m - 1\}$, then wait $K \times 512$ bit transmission times before retransmitting.

Switched Local Area Networks

MAC Addresses

Every NIC has a unique **MAC address** that is used when sending link layer frames. When an adapter receives a frame, it checks if the destination MAC address of the frame matches its own MAC address.

- **48 bits**: Burned in NIC ROM, e.g. 5C-F9-DD-E8-E3-D2
- **IEEE**: Administers the MAC address allocation. The first three bytes of the MAC identifies the vendor of the adapter.
- If by some chance the MAC is manually configured to be not unique, and the NICs are on the same subnet, transmission will be severely affected.

Address Resolution Protocol (ARP)

All nodes have an ARP table containing mappings of IP addresses and MAC addresses of other **neighbouring nodes** in the same subnet. Here's the entry format:

<IP address; MAC address; TTL>

1. If A and B are in the same subnet, and A knows B's MAC address from its ARP table:
   (a) A can just create a frame with B's MAC address and send it.
   (b) Only B will process this frame, all other hosts that receive it will ignore it.
2. Else if A and B are in the same subnet but A does not know B's address:
   (a) A broadcasts an ARP query packet, containing B's IP address. The destination MAC address is set to FF-FF-FF-FF-FF-FF.
   (b) All other nodes in the same subnet will receive this ARP query packet, but only B will reply it.
   (c) A caches B's IP-to-MAC address mapping in its ARP table (until TTL expires).
3. Else if A and B are in different subnets (assume there's a router R directly connecting the two subnets, and A and B):
   (a) A will need to send a frame with R's MAC address but B's IP address as destination.
   (b) R will realise it needs to forward this frame as the IP doesn't match when MAC matches.
   (c) R will forward the datagram to an outgoing link and construct a new frame with B's MAC address.

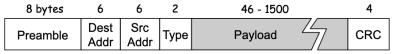
Ethernet Standards

Refer to Lecture 9 slides.

Ethernet Topology

- **Bus Topology**: All nodes are connected and can collide with each other.
- **Star Topology**: There's a switch in the centre and nodes are connected to that switch. They do not collide with each other.

Ethernet Frame



- **Preamble**: Formed by 7 bytes of 10101010 and 1 byte of 10101011.
  - Allows sender and receiver to synchronise clock rates, since the alternating bits form a square wave.
  - Basically, lets the receiver know how long is 1 bit.
- **Destination MAC Address**: If the NIC receives a frame with either matching address or the broadcast address, then it will pass the data in the frame to the network layer protocol. Else it will discard.
- **Type**: Higher layer protocol used, usually IP.
- **Payload**: Minimum size of 46 bytes for collision detection, max size is 1500 (MTU). A maximal payload size makes it easier to implement efficient data buffer management algorithms in the NICs, switches and routers. It also helps to make sure that senders do not "hog" shared links and other senders get a chance to transmit as well.
- **CRC**: For corruption detection.

Ethernet Delivery

- **Connectionless**: No handshaking between sender and receiver.
- **Unreliable**: No ACK or NAK. Data in dropped frames will be recovered only if the initial sender uses higher-layer rdt.
- **Multiple Access Protocol**: CSMA/CD with exponential backoff.

Ethernet CSMA/CD Algorithm

1. NIC receives datagram from the network layer and creates a frame.
2. If NIC senses that the channel is idle, it will start frame transmission. Else it will wait until idle.
3. If NIC transmits the entire frame without detecting another transmission, NIC is done.
4. If another transmission is detected, NIC aborts and sends a **jam signal**, which tells all other nodes that a collision has been detected and NIC will be retransmitting.
5. After aborting, NIC enters exponential (binary) backoff, and repeat from step 2.

Ethernet Switch

A switch is a link-layer device used in LAN that also stores and forwards Ethernet frames.

- **Layer 2**: Unlike routers, which is a layer 3 device i.e. it goes up to the network layer, switches only have 2 layers, i.e. up to link layer.
- **No IP address**: For the reason above, it has no IP address.
- **Transparent to hosts**: Hosts are not aware of the presence of switches. They merely send frames to other hosts, unaware that a switch will receive it and forward it.
- **Collision-free**: Each host has a dedicated connection to the switch, i.e. they have **separate collision domains**. This connection has two channels, i.e. fully **duplex** and frames can be sent two-way simultaneously. Lastly, the switch **buffers** frames if let's say they are currently forwarding another frame to the outgoing link.

Switch Forwarding Table

A switch has multiple interfaces, and it will need to know which nodes are reachable via which interface. This is done via switch forwarding tables, which have the following entry format:

<MAC address of host, interface to reach host, TTL>

- **Self-Learning**: Whenever the switch receives a frame from host A, it will record the interface that reaches A in its switch table, and send all frames for A to that interface.
- **Broadcast**: If the destination host is not found in the switch forwarding table, the switch will broadcast the frame to all outgoing links.

Multimedia Networking

Multimedia Networking Applications

There is a rise in the use of multimedia applications, especially since video are often delivered **Over-the-Top (OTT)**, i.e. a streaming media services are offered directly to viewers via the Internet, bypassing cable, broadcast, and satellite television platforms, which traditionally distribute video content.

There are a few main types of multimedia applications:

- Streaming Stored Audio/Video
- Conversational Voice/Voice-over-IP
- Streaming Live Video

Properties of Video

- **High Bit Rate**: Video streaming consumes a lot of bandwidth, having a bit rate of more than ten times greater than that of photo or music applications.
- **Video Compression**: A video is a sequence of images, generally 24 or 30 images per second. Each image is an array of pixels that represent luminance and colour. We can thus exploit certain redundancies:
  - **Spatial Redundancy**: Instead of sending the same value N times, we just send the value once and N.
  - **Temporal Redundancy**: Only send differences between two consecutive images, as they tend to be very similar.
- **Bit Rate**: We can have the video at constant bit rate, which is easier to buffer for routers and network, or at variable bit rate, which changes based on amount of encoding and gives better quality.

Properties of Audio

- **Sampling**: To convert an audio analog signal into a digital signal, we sample the signal at some fixed rate to get some real number value. Telephone does so at 8,000 samples/sec, CD does 44,100 samples/sec.
- **Quantisation**: Each sample is rounded to one of a finite number of values. The number of quantisation values is typically a power of two, e.g. 256.
- **Concatenation & Decoding**: All the values are represented as bits, and all the samples would then be concatenated together. To decode, we just convert it back to an analog signal. This signal is an approximate of the original, since certain sounds may have been lost in the sampling and encoding.

The process above is called **pulse code modulation**.

- **Speech Encoding**: Often uses PCM, with a sampling rate of 8,000 samples/sec and 8 bits per sample, hence having 64kbps.

- **CD**: Also uses PCM, with sampling rate of 44,100 samples/sec and 16 bits per second. This gives us a rate of 705.6kbps for mono and 1.411 Mbps for stereo.

Compression Ratio

The compression ratio of a media codec (= compressor/decompressor) is measured as the bitrate of an uncompressed media stream divided by the bitrate of the same compressed media stream.

Streaming Stored Video

Has three distinguishing features:

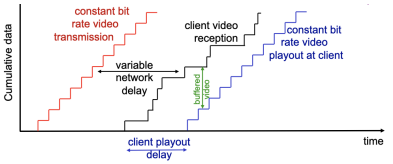
- **Streaming**: Client can play one part while receiving later parts from the server. This avoids having to download the entire video before playout begins.
- **Interactivity**: Because it is prerecorded, users may pause, reposition, fast-forward and so on.
- **Continuous Playout**: Once playout of the video begins, it should proceed according to the timing of the original video, i.e. data must be received from the server in time for its playout.

Client-Side Buffering

There is extensive use of client-side application buffering to mitigate the effects of varying end-to-end delays. When the video first starts to arrive at the client, the client needs not immediately begin playout, but can build up a reserve of video in the buffer. Once several seconds of reserves have been built up, the client can then begin video playout.

- **Absorbs variations in delay**: If a certain piece is delayed, so long it arrives before the reserves are depleted, the delay will not be noticed.
- **Bandwidth changes**: If the bandwidth briefly drops below the video consumption rate, the user can continue to enjoy continuous playback, so long as the buffer is not drained.

If a block/piece does not arrive by the time its playout time, then either the video playout will **stall**, or the block will be **skipped**.



UDP Streaming

- **Push-based Streaming**: Server transmits video at a rate that matches the client's video consumption rate. As UDP does not employ any congestion-control mechanism, the server can push packets without rate-control restrictions of TCP.
  - Let's say the video consumption rate is 2Mbps and each UDP packet carries 8,000 bits of video.
  - The server needs to transmit one UDP packet into its socket every (8000 bits)/(2 Mbps) = 4 msec.
- **RTC**: Uses the Real-Time Transport Protocol covered later, which also has a separate control connection.
- **Small Buffer**: Usually a buffer of less than 1 second of video.
- **Error Recovery**: Must be done at application level, if there is time to do so.
- **Lower Playout Delay**: There may be an initial playout delay of 2-5 seconds.
- **Susceptible to Bandwidth Changes**: UDP streaming does so at constant rate, so if the available bandwidth drops below the consumption and transmission rate, the video will either freeze or skip frames.
- **Complexity**: There is a need for media control, e.g. pause, play etc., which increases complexity. Covered under RTP later.
- **Firewall**: Many firewalls are configured to block UDP traffic, preventing users behind these firewalls from receiving UDP video.

HTTP Streaming

- **Pull-based Streaming**: Put very simply, we try to download the video file from the server, and can potentially download at a rate *higher* than the consumption rate, thus **prefetching** video frames to be consumed in the future.
- **Prefetching**: This occurs naturally as TCP's congestion avoidance mechanism tries to use all available bandwidth. Thus if bandwidth is greater than consumption rate, prefertching will occur.
- **Sending Flow**: Parts of the video file will be brought into the server send buffer → client TCP receive buffer → client TCP application buffer, then finally the application will grab frames from the buffer, decompress it and display them. Potentially, if the application buffer is larger than the video file, we are simply downloading the entire video and playing it.
- **Full Client Application Buffer**: The above flow also means if the application buffer is full, the send rate will be reduced to the video consumption rate.
- **Pausing**: The client may also pause, in which case the sending and buffering will still continue. Once the application buffer is full, then sending will block until video is resumed.
- **Fluctuating Fill Rate**: The fill rate may fluctuate due to TCP congestion control and retransmissions i.e. in-order delivery.
- **Repositioning**: If a user suddenly skips forward in the video, all the buffered frames will be wasted. This is why many video applications use a medium sized client application buffer, to reduce wastage.
- **Larger Playout Delay**: Client application will wait till buffer is filled up to a certain point before playout. Longer than in UDP, which is push-based.
- **Firewall**: HTTP/TCP generally passes through firewalls more easily.

Adaptive HTTP Streaming (DASH)

The above HTTP streaming results in clients receiving the same encoding of the video, despite having differences in the amount of bandwidth across clients and across time for the same client. We thus have **Dynamic Adaptive Streaming over HTTP (DASH)**. Much of the video-on-demand media streaming on the Internet today uses either DASH or Apple's HTTP Live Streaming (HLS).

- **Various Encodings**: The video is encoded into different versions with different bit rates and hence quality levels.
- **Pull-based Streaming**: The client will dynamically request chunks of video segments of **2-10 seconds** in length. They will select chunks from a version that their current bandwidth can support.
- **Manifest and Byte Range**: The HTTP server stores a `.m3u8`/`.mpd` manifest file that provides a URL for each version, along with its bit rate. The client first requests for the manifest file to know the various versions.
- **Adaptive Bitrate Algorithm**: The client runs this algorithm to decide which quality the next chunk should be.
- **Streamlets or Byte Range**: Thereafter, the client will either GET request for a specific streamlet, if the server had split the video into streamlets during pre-processing, or use GET requests with a specified byte range in the header to get the right chunk.
- **Works with Web Caching**: DASH works well with the existing web caching infrastructure that ISPs and Content Delivery Networks (CDN) have built up over in recent years.

Voice-over-IP

Real-time conversational voice over the Internet is often referred to as **Internet telephony**, since it's similar to traditional circuit-switched telephone service. It's also called Voice-over-IP. We will only focus on voice here, instead of voice and video combined.

- **Talk Spurts**: We don't always talk during a conversation. As such, during periods of silence, nothing is sent. However, generally, when a speaker is speaking:
  - We will have 8,000 samples/sec.
  - Each sample will be 8 bits, with 256 quantization levels.
  - The data will thus be 64kbps.
  - We will generate 20 msec chunks, hence 160 bytes per chunk.
- **Special Header**: We will add an application-layer header to each chunk, then all that will be encapsulated into a (generally) UDP segment.

Limitations of Best-Effort IP Service

- **Packet Loss**: UDP segments, which contain a chunk and a special header, are used and may be lost along the way (**network loss**). We cannot use TCP since retransmission mechanisms are often considered unacceptable for conversational real-time audio applications, as they increase end-to-end delay. Generally, packet loss rates between 1% and 10% can be tolerated.
- **End-to-End Delay**: For VoIP, end-to-end delays smaller than 150 msecs are not perceived by a human listener; delays between 150-400 msecs are acceptable but not ideal, and delays beyond that will seriously hinder interactivity in voice conversations. Receivers will generally disregard any packets delayed for more than a certain threshold (**delay loss**).
- **Packet Jitter**: A varying delay is the queuing delay. As such, time taken by a packet can vary from packet to packet. This is called jitter. The receiver cannot simply play the packet the moment it is received, since it will be unintelligible.

Removing Jitter for Audio

- **Timestamp**: Each chunk is prepended with a timestamp. The sender stamps each chunk with the time at which the chunk is generated
- **Playout Delay**: This delay needs to be long enough so that most of the packets are received before their scheduled playout times.
  - **Fixed Playout Delay**: The receiver attempts to play out each chunk at exactly q msecs after the chunk is generated, i.e. a chunk timestamped with time t will be played at time $t + q$. Packets arriving after their scheduled playout time are discarded. If q is large, we have less packet loss but a less interactive experience. The opposite is true if q is small.
  - **Adaptive Playout Delay**: We want the best of both worlds. We estimate the network delay and adjust playout delay at the beginning of each talk spurt, effectively adjusting the silent periods. We can use a exponentially weighted moving average:

$$d_i = (1 - \alpha)d_{i-1} + \alpha(r_i - t_i)$$

where d_i is the delay estimate after the i-th packet, α is a small constant, usually 0.1, r_i is the time received for the i-th packet, and t_i is the time sent for the i-th packet. We will still play it at 20 msec for each chunk during talk spurt.

$$v_i = (1 - \beta)v_{i-1} + \beta|r_i - t_i - d_i|$$

The average deviation is also calculated. Note that all these are calculated for every packet but only used at the start of a talk spurt.

$$\text{playout-time}_i = t_i + d_i + Kv_i$$

where K is usually 4. Remaining packets in talk spurt are played out as per usual.

Recovering from Packet Loss

- **Forward Error Correction (FEC)**: The idea is to add redundant information to the original packet stream that can be used to reconstruct approximations or exact versions of some of the lost packet.
  - **XORed Chunk**: For every n chunks, we create a redundant chunk by XOR-ing those n chunks. We then send these $n + 1$ chunks. This increases bandwidth by a factor of $1/n$, but allows us to reconstruct up to one lost chunk. This increases the playout delay as well, since we need to wait for the entire group of packets before it can begin playout.
  - **Lower-Resolution Audio Stream**: For the n-th chunk, we append a lower quality version of it to the $n + 1$-th chunk. For example, the nominal stream can be PCM encoding at 64kbps, while the **redundant stream** is GSM at 13kbps. So long we do not have consecutive loss of packets, we can conceal the loss. We can extend this further by appending even more low bit-rate chunks, e.g. chunk 3 has low quality chunks 1 and 2.
- **Interleaving**: We resequence units of audio data before transmission, so that originally adjacent units are separated by a certain distance in the transmitted stream. This mitigates the effect of packet losses. For example, we have units of 5 msecs and chunks of 20 msecs. The first chunk may contain units 1, 5, 9 and 13. The second chunk contains 2, 6, 10, and 14. And so on. This improves the perceived quality of an audio stream and has low overhead, but it increases latency and playout delay, since reordering is needed on both sides.
- **Error Concealment**: The simplest way is to just repeat a packet to cover the loss. Another way is interpolation, i.e. we somehow derive the missing

packet based on the packet before and after it. But the latter is more computationally expensive.

Protocols for Real-Time Conversational Applications

The following two protocols are both enjoying widespread implementation in industry products.

Real-Time Transport Protocol (RTP)

RTP defines standards for a packet structure that includes fields for audio/video data, sequence number, timestamps, and other useful information.

- **UDP**: RTP runs on top of UDP. The sending side encapsulates a media chunk within an RTP packet, then encapsulates that packet in a UDP segment, then hands the segment to IP. To some extent, RTP libraries provide **transport-layer** interface that extends UDP.
- **Interoperability**: If two VoIP applications both incorporate RTP, then there's a chance for them to be able to communicate with each other.
- **No Guarantees**: There are no guarantees on timely delivery nor are there other quality-of-service (QoS) guarantees. No guarantee on delivery or prevention of out-of-order packets either.
- **End-Systems**: RTP encapsulation is only seen at the end systems, hence routers do not distinguish between IP datagrams that carry RTP packets and IP datagrams that don't.
- **Streams**: RTP allows each source e.g. a camera to be assigned its own independent RTP stream of packets. For example, we can have one stream for audio, and one for video. However, many popular encoding techniques e.g. MPEG 1 and MPEG 2, bundle the audio and video into a single stream during the encoding process.
- **Sessions**: RTP works for both unicast (one-to-one) applications and multicast trees (one-to-many, many-to-many). For a many-to-many session, all of the session's senders and sources use the same multicast group for sending their RTP streams. These streams belong to an **RTP session**.

Real-Time Control Protocol

This is a sister protocol with RTP. RTCP is a lightweight connection where one packet is sent every few seconds over UDP in both directions, informing the other side about how things are going, e.g. loss rates, congestion levels etc. Basically status information, transmission statistics and quality-of-service (QoS).

Real-Time Streaming Protocol

RTSP is defines control sequences useful in controlling multimedia playback, e.g. play, fast forward, etc. While HTTP is stateless, RTSP has state; an identifier is used when needed to track concurrent sessions. Like HTTP, RTSP uses TCP to maintain an end-to-end connection and, while most RTSP control messages are sent by the client to the server, some commands travel in the other direction (i.e. from server to client).

- **Special-Purpose Server for Media**: The above three protocols require fine-grained packet scheduling and state management, which can be complex.
- **Firewalls**: Since both TCP and UDP are used, UDP transmissions may be blocked by firewalls.
- **Caching**: It is difficult to cache data, since there is no web caching for RTP packets.
- **Short Latency**: End-to-end latency is about 100-150 msecs.

RTP Header

- **Payload Type (7 bits)**: Indicates the type of media encoding currently being used. If the sender changes their encoding during the call, the sender informs the receiver via the payload type field.
  - Payload type 0: PCM mu-law, 64 kbps
  - Payload type 3: GSM, 13 kbps
  - Payload type 7: LPC, 2.4 kbps
  - Payload type 26: Motion JPEG
  - Payload type 31: H.261
  - Payload type 33: MPEG2 video
- **Sequence Number (16 bits)**: Increments by one for each RTP packet sent. Helps to detect packet loss and restore packet sequence.
- **Timestamp (32 bits)**: Sampling instant of the first byte in this RTP data packet. For audio, the timestamp clock increments by one for each sampling period e.g., every 125 µsecs for 8 KHz sampling clock.
  - If the audio application generates chunks of 160 encoded samples, then the timestamp increases by 160 for each RTP packet when the source is active.
  - The timestamp clock continues to increase at a constant rate even if source is inactive.
- **Synchronisation Source Identifier (SSRC) (32 bits)**: Identifies the source of RTP stream. Typically, each stream in an RTP session has a distinct SSRC. It is not the IP address, but is a number that the source randomly assigns when the new stream is started. If the two SSRCs assigned are the same, then the two sources pick a new SSRC value.

WebRTC

WebRTC is an open framework for the web that enables Real-Time Communications (RTC) capabilities in the browser, via simple JavaScript APIs. It makes use of RTP and RTCP .

Session Initiation Protocol (SIP)

The SIP is an open and lightweight protocol that does the following:

- **Call Setup**: It provides mechanisms for establishing calls between a caller and callee over an IP network. It allows the caller notify the callee that it wants to start a call. It allows the participants to agree on media encodings. It allows participants to end calls.
- **IP Address**: It provides mechanisms for the caller to determine the current IP address of the callee, which may be dynamic e.g. due to DHCP, and they may have multiple addresses. It can map a mnemonic identifier to its current IP address.
- **Call Management**: It provides mechanisms for adding new media streams during the call, changing the encoding during the call, inviting new participants during the call, call transfer and call holding.

Here are some other details:

- **Long-Term Vision**: All telephone calls and video conference calls will take place over Internet, where people are identified by names or email addresses, instead of phone numbers. We thus want to be able to reach a callee (if the callee so desires), no matter where this callee roams, and no matter what IP device the callee is currently using.
- **TCP & UDP**: SIP can run over both because it's a very lightweight protocol. In fact, TCP may be better than UDP for mobile devices because NAT table entries in a wireless router or a cell providers' router generally time out much quicker for UDP than for TCP. SIP will need to send `keep-alive` every 30s for UDP, and every 15 minutes for TCP.

Security in Computer Networks

What Is Network Security?

Properties of Security

- **Confidentiality**: Only sender should be able to read and understand the message.
- **Integrity**: Assets can only be modified by authorised parties.
- **Availability**: Services must be available to authorised parties who need it.
- **Authenticity**: Ability for sender and receiver to confirm the identity of each other.
- **Non-repudiation**: Sender cannot convincingly deny having sent something.

Potential Attacks

- Eavesdrop
- Insert or delete messages
- Impersonation or spoofing
- Hijacking connections
- Denial of service

Principles of Cryptography

Terminologies

- **Plaintext**: The original message
- **Ciphertext**: The encrypted message
- **Encryption Key**: Used to encrypt messages
- **Ciphertext Only Attack**: Crack the encryption using only the encrypted text. Generally through brute force or frequency analysis.
- **Known Plaintext Attack**: When you have the ciphertext and the corresponding plaintext.
- **Chosen Plaintext Attack**: Presumes that the attacker can obtain ciphertexts for arbitrary plaintexts.

Symmetric Key Cryptography

When both Alice and Bob share the same key K_S. Challenge is in communicating that shared key securely.

- **Substitution Cipher**: Replace one character for another. The encryption key is a one-to-one mapping of the characters used.
- **Caeser's Cipher**: Left or right rotation of the alphabets or characters used. Encryption key is the shift number.
- **Multiple Substitution Cipher**
  1. Choose n substitutions ciphers.
  2. Choose a random cyclic pattern of numbers 1 to n.
  3. Repeat this cyclic pattern until it's the plaintext length.
  4. For each character in plaintext, find the corresponding number, then use that substitution for that character.

There are two famous symmetric key ciphers, namely Data Encryption Standard (DES) and Advanced Encryption Standard (AES):

	DES	AES
Date designed	1976	1999
Block size	64 bits	128 bits
Key length	56 bits (effective length); up to 112 bits with multiple keys	128, 192, 256 (and possibly more) bits
Operations	16 rounds	10, 12, 14 (depending on key length); can be increased
Encryption primitives	Substitution, permutation	Substitution, shift, bit mixing
Cryptographic primitives	Confusion, diffusion	Confusion, diffusion
Design	Open	Open
Design rationale	Closed	Open
Selection process	Secret	Secret, but open public comments and criticisms invited
Source	IBM, enhanced by NSA	Independent Dutch Cryptographers

Public Key Cryptography

Instead of sharing the same key, public key cryptography involves two keys, one public key and one private key. Both keys can be used to encrypt and decrypt, but generally, when Alice is sending something to Bob, she will use Bob's public key for encryption. This was first proposed by Diffie and Hellman. Public-key cryptography has enabled effective encryption in the SSL (Secure Socket Layer) of the **HTTPS** protocol.

$$m = K_B^-(K_B^+(m)) = K_B^+(K_B^-(m))$$

Rivest-Shamir-Adleman (RSA) Algorithm

1. Choose two distinct prime numbers p and q
   - They should be chosen at random and should be similar in terms of magnitude, but differ in length by a few digits.
   - Can be found efficiently via the primality test.
2. Compute $n = pq$ and $z = (p-1)(q-1)$
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, z) = 1$; that is, e and z are coprime (has no common factors).
4. Determine d such that $e \cdot d - 1$ is divisible by z; that is, d is the modular multiplicative inverse of e modulo z.
5. The public key is (n, e) and the private key is (n, d).

RSA Encryption & Decryption

- We first treat message m as a decimal number based on its bits.
- To encrypt m ($< n$), compute $c = m^e \mod n$.
- To decrypt c, we compute $m = c^d \mod n$.
- To prove:

$$\begin{aligned} c^d \mod n &= (m^e \mod n)^d \mod n \\ &= m^{ed} \mod n \\ &= m^{(ed \mod z)} \mod n \\ &= m^1 \mod n \\ &= m \end{aligned}$$

Combined

Using public key cryptography to encrypt every message is way too slow. For example, DES is at least 100

times faster than RSA. What we can do is to exchange a session (symmetric) key, K_S, using public key cryptography, then use the shared symmetric key to communicate from there on.

Recommended key lengths:

	1982	1995	2002	2010	2020	2030	2040
Sym	56	66	72	78	86	93	101
RSA	417	777	1028	1369	1881	2493	3214

Message Integrity and Digital Signatures

Digital Signatures

We want to have a signature so that we know who is sending the message and that that person cannot refute having sent it, i.e. authenticity and non-repudiation.

The easiest way is to encrypt the entire message using one's private key and send that. But that is computationally expensive.

Hash Function

Here are some properties of hash functions:

- Many to one.
- Produces fixed size **message digest**.
- Given a digest x, it is computationally infeasible to find m such that $x = H(m)$.
- The Internet checksum has some properties of a hash function.
- Common hash functions:
  - **MD5**: Computes 128-bit message digest in 4-step process. Is actually obsolete now as collisions can be found within a minute.
  - **SHA-1**: Another popular standard. Computes a 160-bit message digest.

Signed Message Digests

We want a fixed-length, easy-to-compute digital fingerprint. We can thus first **hash** the message, then encrypt that hash with our private key.

Password Hashing

Passwords are not stored in plaintext but rather hashed and stored. To make it hard to find similar plaintexts by simply checking the hashes, a **salt** i.e. random string of characters is added to the front of the password before being hashed.

Public Key Certification

Now that we have the public and private keys, we need a trustworthy source to get the list of public keys from. This is provided by a **Certificate Authority (CA)**. For details, read my CS2107 notes.

- CAs bind public keys to entities.
- An entity, e.g. a host or a router, needs to register its public key with a CA, often requiring to provide some proof of identity.
- CA creates a certificate binding that entity to its key (along with a lot of other information) and this certificate is digitally signed by the CA.
- A sender will thus get the receiver's certificate from a CA and apply the CA's public key to the certificate to get the receiver's public key.

Network-Layer Security: Virtual Private Networks

Often, institutions want a standalone physical network (routers, links and DNS infrastructure) that is completely separate from the public Internet. Such a disjoint network is called a **private network**.

Virtual Private Networks (VPNs)

But the above method is very costly. Thus, many institutions now create VPNs over the existing public Internet. With a VPN, the inter-office traffic is sent over the Internet but is encrypted beforehand, and is logically separated from other traffic.

How it's achieved is that the gateway router in an office converts the vanilla IPv4 datagram into an IPsec (IP security protocol) datagram. This IPsec datagram has a traditional IPv4 header, so the public routers process it as if it was a normal IPv4 datagram. The payload actually contains a IPsec header, which is used for IPsec processing, and the remaining payload is encrypted.

Operational Security: Firewalls

A **firewall** is a combination of hardware and software that isolates an organisation's internal network from the Internet at large, allowing some packets to pass through and blocking others.

- **DoS Attacks**: Prevents SYN flooding, where attackers establish many bogus TCP connections.
- **Authorised Traffic**: Prevents illegal modification or access of data, and allows only authorised access to different parts of the internal network.
- **Susceptible to IP Spoofing**: Routers cannot really tell if the data really came from where it claimed to be.
- **UDP**: Generally, firewall either filters all or no UDP.
- **Tradeoff**: It's a tradeoff between security and degree of communication with the outside world.

Stateless (Traditional) Packet Filtering

The router filters **packet by packet** based on policy and firewall settings. Filtering decisions are typically based on:

- IP source or destination address.
- TCP or UDP source and destination port.
- Protocol type in IP datagram field: TCP, UDP, ICMP, OSPF and so on.
- ICMP message type.
- TCP SYN and ACK bits.

However, at times, it can be rather heavy-handed, and may let in packets that do not make sense based on current connections.

Policy	*Firewall Setting*
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Stateful Packet Filtering

Stateful packet filtering tracks the status of every TCP connection, such as connection setup, teardown, etc, thus determining if packets make sense. If inactive connections timeout at the firewall, then incoming packets will be rejected.

Here's an updated Access Control List (ACL):

action	source address	dest address	proto	source port	dest port	flag bit	check conxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	X
deny	all	all	all	all	all	all	

Application Gateways

The above two filter on a packet basis. Application gateways allow us provide have certain privileged users with some specific service. Let this service be Telnet.

1. We set up a Telnet application gateway and a packet filter in a router, such that all Telnet connections except those from the application gateway are blocked.
2. All outbound Telnet connections must thus pass through the application gateway.
3. If an internal user wants to Telnet to the outside world, the user must first set up a Telnet session with the application gateway.
4. The application gateway will prompt the user for a user ID and password. Thereafter, it will check if the user has permissions.
5. If the user does, then the gateway:
   (a) Prompts the user for the host name of the external host to which the user wants to connect.
   (b) Set up a Telnet session between the gateway and the external host.
   (c) Acts as the middleman and relays the data between the two hosts.

Internal networks often have multiple application gateways, such as for Telnet, HTTP, FTP, and email. A mail server and Web cache are application gateways as well.

- **Many Application Gateways Needed**: A different gateway is needed for each application.
- **Performance Penalty**: Relaying takes time, especially when multiple users or applications are using the same gateway machine.
- **Client Connection**: The client needs to know how to contact the gateway when the user makes a request, and how to tell the application gateway what external server to connect to.

Physical Layer

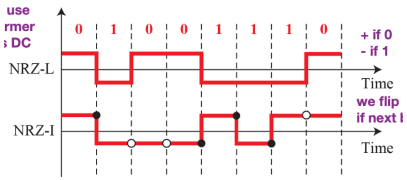
The physical layer moves data in the form of EM signals across transmission medium. 0s and 1s can be transmitted as either **analog signals** or **digital signals**.

Digital Transmission

A digital signal has a fixed number of amplitude values. We generally encode 0s and 1s with different voltages to be transmitted over the wire.

Non-Return-to-Zero

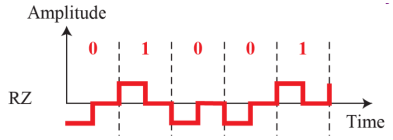
This encoding uses two voltage levels. It has two variations.



- **NRZ-L**: If the voltage is positive, it is 0, else if it is negative, it is 1. The absolute value represents the bit.
- **NRZ-I**: At the start of a bit with value 1, the voltage is inverted i.e. positive to negative and vice versa. Else if it remains the same value, it's a 0. Used by USB.
- **Not Self-Clocking**: The sender and receiver need to have synced clocks, else they are unable to recognise how long is 1 bit.
- **DC Component**: As the average amplitude is not zero, there is a DC component to this signal. This may be undesirable as it cannot pass through transformers or capacitors, requires a physical connection, and may short circuit.

Return-to-Zero

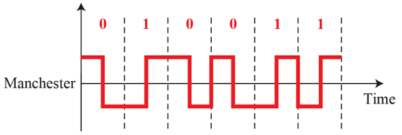
This encoding uses three voltage levels. It always returns the voltage to zero halfway through a bit interval.



- **Self-Clocking**: No need 100% synced clocks for the receiver to recognise 1 bit's length.
- **DC Component**: Same as NRZ.

Manchester

This encoding inverts the signal in the middle of a bit. A positive to negative transition represents 0, while negative to positive represents 1. This is used by Ethernet, RFID and NFC.



- **Self-Clocking**: Same as RZ.
- **AC Signal**: Has no DC component since mean amplitude is always 0.

Analog Transmission

An analog signal is continuous and has infinitely many levels. The most basic signal is the sine wave. Any sort of signal can be decomposed into a sine wave via Fourier Transform.

$$A \sin(2\pi f t + \phi)$$

where A is the peak amplitude, f is the frequency and ϕ is the phase.

Terminology

- **Channel Bandwidth**: A transmission signal only allows signals in a certain frequency range to pass through. The difference between the highest and lowest frequencies is known as the bandwidth.
- **Signal to Noise Ratio**: Often, there is interference from other EM waves. This ratio measures the strength of the signal over such noise.
- **Receiver Comparator**: The signal is sent to this comparator on the receiver side. This comparator sets a minimum threshold for positive voltage, and a maximum threshold for the negative voltage. Anything in between is treated as noise.
- **Modem**: Stands for modulator + demodulator. Basically the device that converts analog signals into digital bit data.
- **Baud Rate**: The number of signal units per second.
- **Bit Rate**: The number of bits received per second.
- **Noise**: For analog signal, interference signals such as electromagnetic fields affect its amplitude, and thus adding noise. Fiber optic cables are the least susceptible to noise as they transmit signals using pulses of light in glass threads.

Shannon Channel Capacity

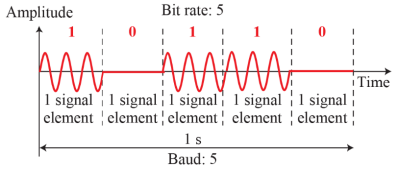
The theoretical maximum bit rate of a noisy channel.

$$C = B \times \log_2(1 + SNR)$$

where B is the channel bandwidth and SNR is the signal to noise ratio of the channel.

Amplitude Shift Keying (ASK)

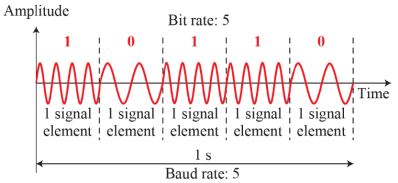
Change the signal's peak amplitude to represent 0s and 1s. If the signal unit's peak amplitude is positive, it represents 1. If the signal has amplitude of 0 i.e. no wave, it represents 0.



- **Susceptible to Noise**: Amplitude is very susceptible, since noise adds to or subtracts from amplitude.

Frequency Shift Keying (FSK)

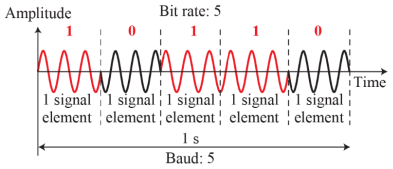
Change the signal's frequency to represent 0s and 1s. If the signal unit's frequency is high, it represents 1. If the frequency is low, it represents 0.



- **Limited by Bandwidth**: The frequencies must be within the frequency range of the channel.

Phase Shift Keying (PSK)

Change the signal's phase to represent 0s and 1s. The phase basically determines how the signal unit begins. We can have 0° for 1, and 180° for 0.

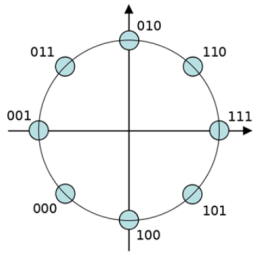


Quadrature Phase Shift Keying (QPSK)

We can transmit even more bits by using more phases. For example, if we use 4 phases, 0°, 90°, 180°, 270°, we can transmit 2 bits per signal unit. We can extend this to **8-PSK** with 8 different phases, thus sending 3 bits per signal unit.

Constellation Diagram

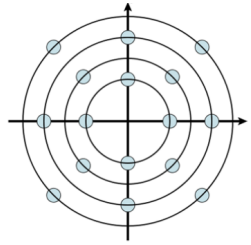
We can use a constellation diagram to represent the phase shift keying.



Quadrature Amplitude Modulation (QAM)

QAM combines ASK and PSK, allowing us to represent k bits in a 2^k-QAM scheme. For example, 16-QAM has 16 different signal elements. Each element differs in either amplitude or phase. Receiver checks both to determine the data carried by a signal.

The sender may switch between different QAMs depending on the noise in the channel. This is used by WiFi (PSK, QPSK, 16-QAM and 64-QAM). This is also used by Singapore TV broadcast (DVB-T). When we switch from 64-QAM to 16-QAM, the maximum bit rate capacity of the channel will drop by a factor of 1.5 since we now transmit 4 bits per signal unit instead of 6.



Commands

`traceroute`

Firstly, it tells you that it's tracing the route to some domain, tells you the IP address of that domain, and what the maximum number of hops will be before it times out. Next it gives information about each router it passes through on the way to its destination. Each of the 3 columns are a response from that router, and how long it took (each hop is tested 3 times).

- 1 is the internet gateway on the network this traceroute was done from.
- The next few routers are likely part of the ISP that the origin computer is connected to.
- The next few are likely global gateways.
- We will then see that we will head towards the destination's local ISP.
- Finally, we will get a router on the network that the domain is hosted on, and lastly the host that the domain is hosted on directly.

`nslookup`

This is usually used to find the IP address that corresponds to a host, or the domain name that corresponds to an IP address (a process called "Reverse DNS Lookup"), by retrieving the relevant address information directly from the DNS cache of name servers.

- The initial server and address are those of our DNS server.
- An authoritative answer comes from a nameserver that is considered authoritative for the domain which it's returning a record for (one of the nameservers in the list for the domain you did a lookup on).
- A non-authoritative answer comes from anywhere else (a nameserver not in the list for the domain you did a lookup on).

`ping`

It sends packets of data to a specific IP address on a network, and then lets you know how long it took to transmit that data and get a response. It uses the echo request and echo reply messages within ICMP. When a `ping` command is issued, an echo request packet is sent to the address specified. When the remote host receives the echo request, it responds with an echo reply packet.

By default, the ping command sends several echo requests, typically four or five. The result of each echo request is displayed, showing whether the request received a successful response, how many bytes were received in response, the Time to Live (TTL), and how long the response took to receive, along with statistics about packet loss and round trip times.

`dig`

Domain Information Groper (dig) allows us to query DNS servers.

- Let's say we do `dig google.com`.
- Lines beginning with `;` are comments not part of the information.
- The first line tell us the version of the `dig` command.
- Next, it shows the header of the response it received from the DNS server.
- Next comes the question section, which simply tells us the query, which in this case is a query for the "A" record of google.com. The IN means this is an Internet lookup (in the Internet class).
- The answer section tells us that google.com has quite a few IP addresses.
- Lastly there are some stats about the query. You can turn off these stats using the +nostats option.

`ifconfig` or `ipconfig`

`ipconfig` stands for Internet Protocol Configuration. This command is used to view all the current TCP/IP network configurations values of the computer, mainly used in Microsoft Windows operating system.

- `ipconfig/registerdns`: Refreshes all DHCP leases and reregisters the DNS names.
- `ipconfig/displaydns`: Displays the information that is stored in the DNS Resolver cache.
- `ipconfig/renew`: Requests a new IP address.
- `ipconfig/flushdns`: Clears the DNS Resolver cache containing previous DNS information.

The `ifconfig` command is mainly used in a Unix-like operating system.

- `ifconfig [interface name]`: This command gives information about the network configuration of the specified interface only.
- `ifconfig {a`: This command gives the network configuration information about all the connected interfaces, whether they are active or not.

`telnet`

Telnet is a computer protocol that was built for interacting with remote computers. It is one of the simplest ways to check connectivity on certain ports. The format is as such: `telnet [domain name or ip] [port]`. If the connection succeeds, a blank screen will show up, meaning that the computer port is open. A failed connection will be accompanied by an error message. It can indicate either a closed port or the fact that the indicated remote server is not listening on the provided port.

`arp`

Displays and modifies entries in the ARP cache, which contains one or more tables that are used to store IP addresses and their resolved Ethernet or Token Ring physical addresses. There is a **separate table** for each Ethernet or Token Ring network adapter installed on your computer. Used without parameters, `arp` displays help.

- `arp -a`: Displays current ARP cache tables for all interfaces.
- `arp -d hostname`: Deletes an entry with the specified `hostname`. To delete all entries, use `arp -d *`, i.e. flush your ARP cache.
- `arp -s hostname ether_addr`: Adds a static entry to the ARP cache that resolves the IP address `hostname` to the physical address `ether_addr`.

`md5sum`

Prints a 32-character (128-bit) checksum of the given file, using the MD5 algorithm.

`curl`

Transfers data to or from a server, using any of the supported protocols (HTTP, FTP, IMAP, POP3, SCP, SFTP, SMTP, TFTP, TELNET, LDAP or FILE). The most basic use is typing the command followed by the URL.

Miscellaneous

- MTU = IP Payload Size + IP Header = MSS + Transport Layer Header + IP Header
- Refer to Tutorial 8 for detailed ARP resolution.