Jin Wei: CS2105 Notes

Internet Protocol Stack

socket

- **Application**
  - Supporting network applications
  - FTP, SMTP, HTTP, DNS, DHCP
- **Transport**
  - Process-process data transfer
  - Logical communication between *processes*
  - Implemented in End System (because need to inject IP / port)
  - TCP, UDP by OS
- **Network**
  - Routing of datagrams from source to destination
  - Logical communication between *hosts*
  - IP, ICMP, routing protocols (BGP, OSPF, RIP)
    - routing protocols are partially Application Layer
- **Link**
  - Data transfer between neighbouring network elements
  - Ethernet, WiFi, PPP, ARP (but partially Network layer too)
- **Physical**
  - Physical bits on wires

Network Paradigms

Packet-Switching Paradigm

- **Packet Transmission Delay** = time needed to transmit L-bit packet into link = L/R
  - **Packets** of length L bits
  - **Transmission rate R** (bits / second) aka link capacity, aka link bandwidth
- **Routing**: Determines *source-destination route* taken by packets (decided by routing algorithms)
- **Forwarding**: Move packets from router's *input* to appropriate *output*

Circuit-Switching Paradigm

- End-end resources allocated to, reserved for "call" between source and destination.
- **No sharing** of resources
  - Circuit-like performance
- Circuit segment idles if not used by call
- Used by traditional telephone networks

Internet Structure

Network of Networks.

- Access Nets
  - Access nets ⇒ regional net ⇒ ISP
  - Access nets ⇒ ISP
- ISP
  - ISP ⇒ IXP (Internet Exchange Point)
  - ISP ⇔ ISP (Peering Link)
- Content Delivery Network covers everything, bypassing tier-I and regional ISPs
- Router
  - For communication to OTHER local area networks
- Switch
  - For local area network
  - Local connections don't go through router
- Hub
  - Switch, but send same message to everyone (like a broadcast)

Packet Delay

- **Nodal Processing** d_{proc}
  - Check bit errors
  - Determine output link
  - Very fast

- **Queueing Delay** d_{queue}
  - Time waiting at output link for transmission
  - Depends on $congestion$ level of router
- **Transmission Delay** d_{trans}
  - Time for a packet to be transferred onto the physical link
  - L: packet length (bits)
  - R: link bandwith (bps)
  - $d_{trans} = L/R$
- **Propagation Delay** d_{prop}
  - Time for a packet to travel along the physical link
  - d: length of physical link
  - s: propagation speed (~$2 * 10^8$ m/sec)
  - $d_{prop} = d/s$

Throughput

Rate (bits / time unit) at which bits are transferred. R_s and R_c are two pipes that the data is flowing through. Doesn't depend on file size.

$$time = \frac{F}{R_s} + \frac{F}{R_c} = \frac{F(R_s + R_c)}{R_s R_c}$$
$$throughput = \frac{F}{time} = \frac{R_s R_c}{R_s + R_c}$$

Message Segmentation

Time = (time for first packet) + (delay between first and second packet) * (n - 1)

- delay between first and second packet is bottlenecked by $slowest$ router's transmission rate
- ignore propagation (B start transmitting when A propagates)

Pros

- Can give way to higher priority packets
- Faster to verify the Checksum
- Easier to identify which part of the packet is corrupted
- Shorter length of message to resend
- Include sequence number in parallel streams

Cons

- Overhead of many segments
- Have to re-arrange and merge messages
- All packet switches doing work

Network Protocols

Client-Server Architecture

- Server
  - Always-on host
  - Permanent IP
  - Data centers for scaling
- Client
  - Communicates with server
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly with each other

P2P Architecture

- No always-on server
- Peers request and provide service to other peers
  - Self scalability! New peers bring service capacity & demand
- Peers are intermittently connected and change IP addresses, very complex management
- **Process**: program running within a host
  - Within same host, two processes communicate using inter-process communication (defined by OS)

  - Processes in different hosts communicate by exchanging messages
  - Client-Server
    - Client process initiates communication
    - Server process waits to be contacted
  - Process can have multiple ports for different purposes!

- **Socket**: process sends/receives messages from *socket*

  - destination IP address + source IP address + port numbers ⇒ identifies socket

TCP Service

- Reliable data transport (correct & ordered)
- Flow control: sender won't overwhelm receiver
- Congestion control: throttle sender when network is overloaded
- Does not provide timing, minimum throughput guarantee, security
- Connection-oriented: requires setup
- Reliable byte stream, connection-oriented

UDP Service

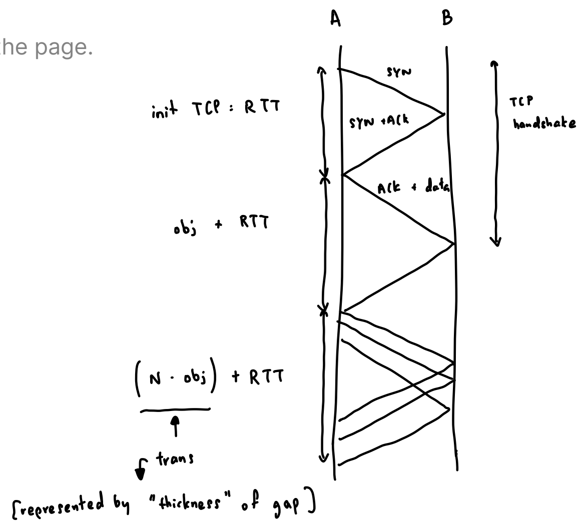
- Unreliable data transfer
- Does not provide: reliability, flow control, congestion control, timing, minimum throughput guarantee, security, connection setup
- Unreliable datagram, connectionless

HTTP

HTTP uses TCP connection. HTTP is stateless, requests are independent.

- Client initiates TCP connection.
- Server accepts TCP connection from client.
- HTTP messages exchanged between browser and Web Server.
- TCP connection is closed.

> Note: need to fetch the HTML file itself first before the other objects / resources on the page.



Non-Persistent HTTP

- At most one object sent over TCP connection, then it's closed
- Downloading multiple objects require multiple connections
- `Connection: close`
- Default for HTTP 1.0

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client, server
  - Can send distinct web pages, as long as from same web server
- Default for HTTP 1.1
- `Connection: keep-alive`

Response Time

- RTT (round trip time): time for a *small packet* to travel from client to server and back
- HTTP response time
  - One RTT to initiate TCP connection
  - - One RTT for HTTP request and first few bytes of HTTP response to return
  - - file transmission time
  - = 2RTT + file transmission time
- If non-persistent: 2 RTT per object
- If persistent: 1 RTT to initiate connection + 1 RTT per object

HTTP Request Message

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n Accept-Language: en-us,en;q=0.5\r\n Accept-Encoding: gzip,deflate\r\n Accept-
Charset: ISO-8859-1,utf-8;q=0.7\r\n Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

```
method sp URL sp version cr lf
header-field-name value cr lf
...
header-field-name value cr lf
cr lf
BODY
```

HTTP Response Message

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n Server: Apache/2.0.52 (CentOS)\r\n Last-Modified: Tue, 30 Oct 2007 17:00:02
    GMT\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
    1\r\n data
\r\n
data data data data data ...
```

- Content-Length only includes the length of the content, not the Header

HTTP Methods

- HTTP/1.0
  - GET
  - POST
  - HEAD
    - Only return header
- HTTP/1.1
  - GET
  - POST
  - HEAD
  - PUT
    - Uploads file in entity body to path specified in URL field
  - DELETE
    - Deletes file specified in URL field

Cookies

- Cookie header line of HTTP *response* message
  - `set-cookie: 1678`
- Cookie header line in HTTP *request* message
  - `cookie: 1678`
- Cookie file kept on user's host, managed by browser
  - Some information is stored on the server (sometimes), but not stored as a cookie
- Backend database of website

Use of Cookies

- Authorisation
- Shopping carts
- Recommendations
- User session state

Web Caches (Proxy Server)

Satisfy client request without involving origin server.

Conditional GET

- In *request* header, include `if-modified-since: <date>`
- If object is not modified before `<date>`, server returns `304 Not Modified` with an empty message body
- Otherwise, returns as normal

DNS: Domain Name System

Mapping between *domain names* (or canonical name) and *IP address*.

Distributed, hierarchical database

- Client wants to find `www.amazon.com`
  - Client queries `root` server to find `.com` server
  - Client queries `.com` server to get `amazon.com` server
  - Client queries `amazon.com` server to get `www.amazon.com` IP address

Root name servers are contacted by local name server that cannot resolve name. Provides IP address of TLD servers.

TLD servers are responsible for `com`, `org`, `net`, `edu`, country domains, etc...

Authoritative DNS servers provide authoritative hostname to IP mapping for organisation's named hosts. Can be maintained by organisation or service provider.

- Local DNS servers are not in global hierarchy of distributed database (because it can be private!!).

Query Resolution

- Iterated Query
  - Server only returns you the IP address of the next server
  - Used in practice
- Recursive Query
  - Server recursively asks other servers until it obtains the final IP address, then returns it
  - Not used in practice because heavy load on upper levels of hierarchy
  - Essentially double the work
- Local DNS server sends / issues *iterative queries* to other DNS servers, *but* responds to / processes **your** query in a *recursive* way.

DNS Caching

- Name server *caches* mapping (including Local DNS server)
  - Cache entries timeout after some TTL (time to live)
  - TLD servers typically cached in local name servers
    - So root name servers are not visited often
- Cached entries may be out of date!
  - hostname ⇒ IP mapping may only be up to date once all TLLs expire

Socket Programming

Port Number

- 16-bit integer (0 to 1023) are reserved for standard use.

Socket Programming with UDP

- No handshaking before sending data
- Sender explicitly attaches (IP destination address) and (port number) with *each packet*
- Receiver extracts (sender IP address) and (port number) from *each packet*
- May be lost or received out-of-order
- Unreliable datagrams

> DNS (port number 53) resolving uses UDP.

UDP Client

```
from socket import *

server_name = "hostname"
server_port = 12000

client_socket = socket(AF_INET, SOCK_DGRAM) # for ipv4, for UDP
message = "blah"
client_socket.sendto(message.encode(), (server_name, server_port)) # encode to byte array
recv_message, server_address = client_socket.recvfrom(2048) # receive up to 2048 bytes
decoded_message = recv_message.decode()
client_socket.close()
```

UDP Server

```
from socket import *

server_port = 12000
server_socket = socket(AF_INET, SOCK_DGRAM)
server_socket.bind(("", server_port)) # empty IP address defaults to localhost
```

```
while True:
    message, client_address = server_socket.recvfrom(2048)
    modified_message = modify(message.decode())
    server_socket.sendto(modified_message.encode(), client_address)
```

Socket Programming with TCP

- Client must perform handshake
  - Server must be running and have a *welcome socket*
- TCP server creates *new socket* for each client
  - Source port numbers / connection are used to distinguish clients
    - No need to explicitly send (IP address) + (port number)

TCP Client

```
...
client_socket = socket(AF_INET, SOCK_STREAM) # for TCP
client_socket.connect((server_name, server_port))
client_socket.send(message.encode())
recv_message = client_socket.recv(1024)
client_socket.close()
```

TCP Server

```
server_socket = socket(AF_INET, SOCK_STREAM)
server_socket.bind(("", server_port))
server_socket.listen(1) # maximum number of connections
while True:
    connection_socket, addr = server_socket.accept() # waits for connection
    connection_socket.recv(1024)
    connection_socket.send(modified_message.encode())
    connection_socket.close() # close connection to THIS client
```

UDP Transport-Layer

UDP: User Datagram Protocol

- On top of IP:
  - Adds connectionless multiplexing / de-multiplexing
  - Adds checksum
- Unreliable, but more performant
  - Often used by streaming multimedia apps (loss tolerant & rate sensitive)
- Application implements error detection and recovery mechanisms to achieve reliable transmission

Why UDP?

- No delay of establishing connection
- Simple: no connection state
- Small header size
- No congestion control

UDP Segment Header

```
⟵       32 bits       ⟶
Source Port # | Dest Port #
Length        | Checksum

PAYLOAD...
```

- Length = payload length (in bytes), *including* header (header is always 8 bytes)

UDP / TCP Checksum

1. Treat UDP segment as sequence of 16-bit integers
   1. Includes Header + Payload + Pseudo-Header (source address / destination address)
2. Apply binary addition on every 16-bit integer
3. Carry (if any) from MSB will be added to result
4. Compute 1's complement (flip every bit) ⇒ UDP Checksum

Creating Reliable Data Transfer

rdt1.0

- Underlying channel perfectly reliable
- No bit errors
- No loss of packets

rdt2.0

- Underlying channel may flip bits (corrupt packet)
  - Use Checksum to detect this
- Receiver explicitly `ACK` and `NAK`
  - Sender retransmits packet on `NAK`
- **Problem**: cannot detect duplicate packets

rdt2.1: rdt2.0 + packet sequence

- Sender adds sequence number (0 or 1) to each packet
  - Receiver discards duplicate packet

rdt2.2: NAK-free

- Instead of NAK, receiver sends `ACK0` or `ACK1` (include sequence number)
  - Sender retransmits current packet if duplicate `ACK`

rdt3.0

- Underlying channel can lose packets (data, ACKs) → in addition to corruption
- Sender waits a "reasonable" amount of time (close to RTT) for ACK
  - Requires countdown timer
  - Retransmits if no ACK received
- Sender ignores duplicate ACKs (already know it's received) & corrupted ACKs (let timer resolve it)
- Receiver must ACK duplicate packet (otherwise will be livelock, because sender will keep retransmitting), receiver can ignore corrupted packets (must send feedback by definition of rdt3.0, but doesn't actually break the correctness of the protocol, just treat it as lost)
- Correct, *if* messages are not reordered (reasonable assumption if sender and receiver are connected by a single wire)
  - To rectify:
    - Use a 32-bit sequence number to lower chance of reuse
    - Use a Time-To-Live (TLL) field that decreases by one each time the datagram arrives at a router
      - Once this field reaches 0, discard the datagram

Pipelined Protocols

Go-Back-N (GBN)

aka Sliding Window Protocol

Re-transmits (up to) everything even if only one packet is dropped!

Go-Back-N Sender

- Use k-bit sequence number $[0, 2^k - 1]$
- "Window" of up to N consecutive un-ACK'd packets are allowed
- Single timer for oldest in-flight packet
- When timeout, retransmit entire window
  - If received ACK of 1: resend $[2, 2 + N - 1]$

Go-Back-N Receiver

- Always ACK for correctly-received packet with highest *in-order* sequence number (cumulative ACK) - **not** expected
  - Only need to remember the expected sequence number
- Out-of-order packet
  - Just discard
  - Still sends ACK for packet with highest *in-order* sequence number

Selective Repeat

- Receiver *individually* acknowledges all correctly received packets
  - Buffers out-of-order packets
  - If in-order
    - Deliver packet + buffered packets to application
    - Advance window to next not-yet-received packet

- Sender maintains timer for *each* unACK'd packet
    - When timer expires, only retransmit that packet
    - IF ACK'd packet is the *smallest* unACK'd packet, advance window base to next unACK'd packet
- Views for Sender and Receiver might not be consistent!

TCP Transport Layer

- Fully duplex data: bi-directional data flow (no sender or receiver)
    - Both sides have sending & receiving buffers
- Reliable, in-order byte stream: no message boundaries
- Maximum Segment Size (MSS): maximum application-layer data a TCP segment can carry (typically 1460 bytes)
    - Limited by Maximum Transmission Unit (MTU) (1500 bytes for Ethernet)
- Retransmissions are triggered by:
    - timeout events
    - 3 duplicate ACKs (total 4 ACKs)
- Still useful even if links are reliable:
    - All *links* are reliable, **but** malicious or full routers can lose packets.
    - Data taking different paths can lead to re-ordering.

TCP Header

```
⟵        32 bits        ⟶
Source Port # | Dest Port #
       Sequence Number
    Acknowledgement Number
head    UAPRSF| rwnd
Checksum      | urgent_data_pointer
  options (variable length)

Application Data (variable length)
```

- Sequence number of *first* byte
    - Initial sequence number is randomly chosen
    - `SYN` consumes an extra sequence number
    - `ACK` **does not** consume sequence numbers
- Acknowledgement number: sequence number of *next expected* byte (from other side)
    - Cumulative ACK
- `head` length of header in 32-bit words (4 bits)
    - minimum size is 5 words, maximum size is 20 words, allows for up to 40 bytes of options
- reserved / unused space (4 bits)
- `U` urgent data (not generally used)
- `A` acknowledgement bit: 1 if valid acknowledgement field, 0 if should ignore (only for first packet of set-up)
- `P` push data now (not generally used)
- `RST`, `SYN`, `FIN` setup, teardown commands
- `rwnd` receive window: # bytes receiver is willing to accept
    - Used for congestion control
- Same checksum as UDP's checksum
- TCP spec doesn't mention how receiver handles out-of-order segments (can choose to buffer)

Connection-Oriented Demux

- Same destination IP address + port ⇒ demultiplexed to *different* sockets
- Identified by (source IP, source port, dest IP, dest port)
    - Handled by OS

TCP 3-way Handshake

- Client sends TCP with `SYN` bit, with `SEQ = x`
- Server sends TCP with `SYN` bit, with `SEQ = y`, `A = 1`, `ACK = x+1` (`SYN` consumes a sequence number)
- Client sends TCP with `A = 1`, `ACK = y+1` (might also send data)

TCP Closing Connection

- Client, Server each close their "sending side"
    - Sends TCP segment with `FIN = 1`
    - Responds to `FIN` with `ACK`, can combine with own `FIN`

  - After closing, can no longer send data (except ACK)
  - BUT, still receives data!
- After sending `ACK` for a `FIN`, must wait 2 * max segment lifetime in case this last `ACK` is lost!

TCP RDT

TCP Sender

- Receive data from application
  - Start **single** timer (if not already running) for $\textit{oldest unACK'd segment}$
  - Send the segment
- Re-transmits $\textit{single}$ segment when:
  - Timeout (then, restart timer)
  - Fast retransmission
- On receiving ACK
  - If ACK acknowledges previously unACK'd segment
  - Update known ACK'd segments
  - Start timer if there are still unACK'd segments

TCP ACK Generation

Event at receiver	**TCP receiver action**
Arrival of in-order segment with expected sequence number. All data up to expected sequence number has been ACK'd.	$\textit{Delay}$ ACK. Wait up to 500ms for next segment. If no segment, send ACK.
Arrival of in-order segment with expected sequence number. One pending ACK'd.	Immediately send single cumulative ACK, ACKing both segments.
Arrival of $\textit{out-of-order}$ segment with $\textit{higher}$ sequence number. Gap detected!	Immediately send duplicate ACK, indicating sequence number of next expected byte.
Arrival of segment that partially or completely fills gap.	Immediately send ACK, provided that segment starts at lower end of gap. (fulfils cumulative ACK)

TCP RTT, Timeout

- Timeout needs to be longer than RTT
  - If too short: premature timeout, unnecessary retransmissions
  - If too long: slow reaction to segment loss

Measure $SampleRTT$, the measured time from segment transmission until ACK receipt (ignoring retransmissions).

- $EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$
  - Exponential weighted moving average
  - Influence of past sample decreases exponentially fast
  - Typically, $\alpha = 1/8$
  - Typically, $\beta = 1/4$
- $DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$
- TimeoutInterval = EstimatedRTT + 4 * DevRTT
  - EstimatedRTT + Safety Margin

TCP Fast Retransmit

- If sender receives 4 ACKS for same data (aka triple duplicate ACKs)
  - Immediately resend unACK'd segment with smallest sequence number

Network Layer

- **Forwarding** (local behaviour): Move packets from router's input to appropriate router output
- **Routing** (global behaviour): Determine route taken by packets from source to destination using routing algorithms
- **Data Plane** (lower layer)
  - local, per-router $\textit{forwarding}$ function
  - determines how datagram arriving on router input port is forwarded to router output port
- **Control Plane** "brain" (upper layer)
  - network-wide logic
  - determines how datagram is routed among routers along end-end path from source host to destination host
  - two approaches: (1) traditional routing algorithms (implemented in routers), (2) software-defined networking (SDN) (implemented in remote servers)

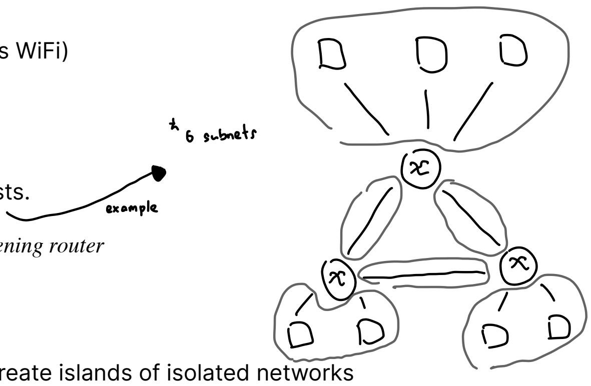
IP Addressing

Interface: connection between host / router and physical link

- routers typically have multiple interfaces
- host typically has one or two interfaces (wired Ethernet, or wireless WiFi)
- an IP address is associated with each interface

Subnets

Subnet is a network formed by a group of "directly" interconnected hosts.



- hosts in same subnet can physically reach each other *without intervening router*
- connect to other subnets through *router*
- hosts in same subnet have same **network prefix** of IP address

Number of subnets = detach each interface from its host or router to create islands of isolated networks

CIDR: Classless InterDomain Routing

- *arbitrary length* for the subnet portion
- `a.b.c.d/x`, `x` is number of bits in subnet portion of address
- ← network part (`x` bits) →← host part (`32 - x` bits) →
- subnet mask = set all network bits to `1` and host bits to `0`, use *bitwise AND* to find network
    - can represent as a single number, or in binary
    - subnet mask of `0` matches everything

Supernetting

- Combining many subnets into one large subnet.
- Decreases the number of unusable (reserved) IP addresses!

Take the longest common prefix. This longest prefix will match all the above.

Subnetting

- Breakup a subnet into smaller subnets
- For organisation of subnets
    - For example, each company has its own subnet
        - Because communication within subnet use switch (not router!)
        - Want to isolate each company's subnet
- Start from the biggest (or smallest) subnet
    - find out how many **bits** are needed
    - assign an appropriate prefix to it
    - size must always be in power of 2

Special IPs

ISP gets blocks of addresses from ICANN (Internet Corporation for Assigned Names and Numbers)

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

Special Addresses	Present Use
0.0.0.0/8	Non-routable meta-address for special use
127.0.0.0/8	Loopback address.Datagram sent to localhost loops back inside the host
10.0.0.0/8 172.16.0.0/12 192.168.0.0/16	Private addresses, can be used without coordination with Internet registry
255.255.255.255/32	Broadcast address. All hosts on same subnet receive datagrams sent with this destination address.

- first IP address is reserved for network address (name of the network)
- last IP address is reserved for broadcast
- gateway router can be *any* IP address - need not be the first usable one! (DHCP will let you know this IP address)

Hierarchical addressing: route aggregation. Just need to announce the *block* of IP addresses for an organisation / ISP.

- can announce multiple (to handle organisations switching ISPs)
- longest prefix matching, i.e. match the one with the higher `x`
- there's a default route, usually to ISP's special website

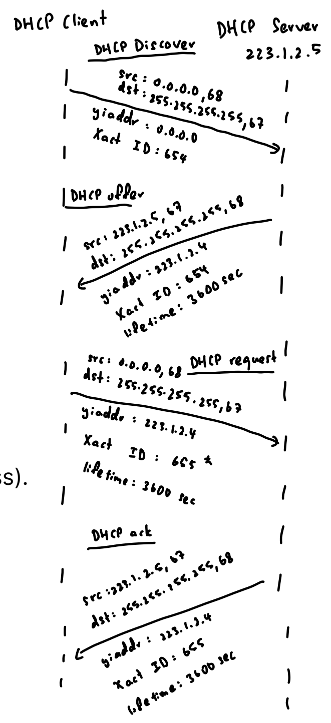
DHCP

2 ways to get IP address:

- *Host*'s IP address is hard-coded by system admin.
- OR **DHCP** (don't need any configuration, dynamically obtain)

* yiaddr: your allocated address

Dynamic Host Configuration Protocol: allows host to *dynamically* obtain IP address from network server when joining a network



- goal:
  - can renew lease on address
  - allows reuse of addresses (only holds address when someone is actively using it)
  - support for mobile users
- overview - **everything** is broadcasted (only difference is 67 or 68)
  - host broadcasts *DHCP discover* message (optional)
    - uses source IP of `0.0.0.0` (because not assigned yet)
  - server responses with *DHCP offer* message - containing *a few* IP addresses (optional)
  - host requests IP address with *DHCP request* message
    - uses source IP of `0.0.0.0` (because not assigned yet)
  - server sends address with *DHCP ack* message
  - after which, done, the host is officially assigned to that IP address
  - also exists: *DHCP renewal* (request to re-use IP address) and *DHCP release* (give up IP address).
- DHCP can also return
  - address of first-hop router for client
  - name and IP address of DNS server
  - network mask
- DHCP runs over UDP
  - DHCP server port number: 67
  - DHCP client port number: 68

Note: there might be multiple DHCP servers (offering at the same time), which is why everything must be broadcasted, so everyone is aware.

Even if a subnet is not using DHCP, router will relay it to a subnet *with* a DHCP server

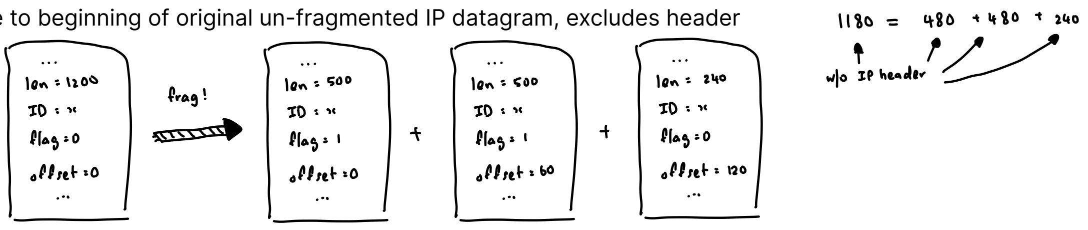
IPv4 Datagram Format

```
⟵        32 bits        ⟶
ver           | IP datagram length
identifier    | flags | fragment offset
TTL | upper_layer | header_checksum
        source IP address
         dest IP address
Data
```

- ver (4 bits)
- IP datagram length (16 bits) - header + data
- identifier (16 bits) - for fragmentation / reassembly
- flags (3 bits)
  - 1bit for frag bit
- fragment offset (13 bits)
- TTL (8 bits) - number of remaining hops (decremented at each router), kills datagram when TTL = 0
- upper layer protocol (8 bits): TCP / UDP (violates upper-lower abstraction!)
- header_checksum (16 bits): only for IP header
- total: 20 bytes

IP Fragmentation

- Different links have different MTU (Max Transfer Unit) - maximum amount of data a link-level frame can carry
- IP datagrams have to be fragmented by routers (if too large)
  - Reassembled only at **final** destination host
- Frag Flag set to 1 if there's a next fragment of same segment; 0 if this is last fragment
- Offset is expressed in units of **8-bytes** (aka 64 bits)
  - offset of **DATA** relative to beginning of original un-fragmented IP datagram, excludes header



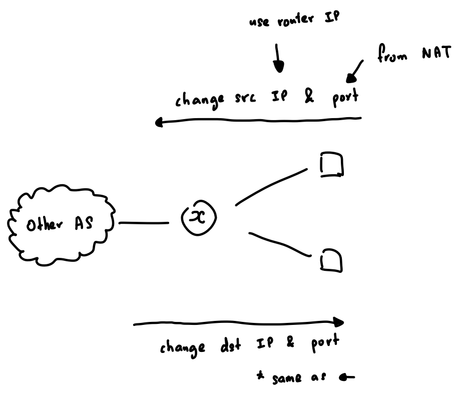
IP + UDP Fragmentation

- Nothing to modify

IP + TCP Fragmentation

- TCP anticipates IP fragmentation, TCP segments itself first such that IP fragmentation will never happen
- So, every segment will contain TCP header
- Happens in Transport Layer (knows the MTU through OS, or some other way)

NAT Table



Network Address Translation

- WAN (wide-area network) - the internet
  - **all** datagrams *leaving* LAN have *same* source NAT ip address (with *unique* port number)
  - Port numbers are from $[0, 2^{16}]$, but $[0, 2^{10} = 1024]$ are reserved.
- LAN (local-area network)
  - within LAN, hosts use *private IP address* to communicate
  - Port numbers are from $[0, 2^{16}]$, none are reserved. So, can use any in that range!

NAT routers must

- outgoing: *replace* (source IP address, port number) of every **outgoing datagram** to (NAT ip address, new port number)
- store: *remember* in NAT table the mapping of the above
- incoming: *replace* (NAT ip address, new port number) in destination fields of every **incoming datagram** with corresponding (source IP address, port number) stored in NAT table

Motivations & Benefits & Cons

- only require one public IP for the NAT router
- all hosts use private IP address, can change these without notifying outside world
- can change ISP without changing addresses of hosts in local network
- hosts inside local network are not explicitly addressable and visible by outside world (protected by NAT router, good for security!)
- bad for peer to peer connections

Intra-AS Routing

AS = Autonomous Systems, normally tied to an ISP (i.e. Singtel, Starhub...)

- finds a good path between two routers within an AS
- single admin, no policy decisions needed
- routing focuses on performance
- commonly used: RIP, OSPF

Bellman-Ford

- $C_{x,y}$ = cost of link between routers x and y (infinity if not *direct* neighbours)
- $D_x(y)$ = least-cost from x to y
- $D_u(z) = min_{a \in N}\{c(u,a) + D_a(z)\}$
  - where N is 1-hop neighbourhood
- At t = 0
  - all nodes have distance estimates to nearest neighbours only
  - then, all nodes send their local distance vector to their neighbours
- At each next time interval
  - all nodes receive distance vectors from neighbours (after some time interval)
  - then, all nodes compute their new local distance vector
  - then, all nodes send their local distance vector to their neighbours

Pros

- iterative, asynchronous: each local iteration caused by
  - local link cost change
  - distance vector update message from neighbour
- distributed, self stopping: each node notifies neighbours only when its own distance vector changes
  - then, only propagate if necessary
  - no new estimate ⇒ no actions taken

Implemented by RIP (Routing Information Protocol)

- uses *hop count* (total number of intermediate routers) as cost metric
  - insensitive to network congestion
- exchanges routing table every 30 seconds over UDP port 520
- *self-repair*: if no update from neighbouring router for 3 minutes, assumes neighbour has failed
- can split paths for *equal-cost* links (**cannot** do more advanced load balancing)

Inter-AS Routing

not covered in-depth in CS2105

- handles interface between ASs
- admin often wants control over how traffic is routed, who routes through net, etc
- security policy over performance
- de facto standard protocol: BGP

ICMP

Internet Control Message Protocol: used by hosts & routers to communicate netowrk-level information

- error reporting: unreachable host / network / port / protocol
- echo request / reply (used by `ping`)

ICMP messages are carried in IP datagrams, ICMP header starts after IP header. It is in the transport layer, but only used for network stuff. (still considered network layer)

ICMP Header = type + code (sub-type) + checksum + others

Type	Code	Description
8	0	echo request (ping)
0	0	echo reply (ping)
3	1	dest host unreachable
3	3	dest port unreachable
11	0	TTL expired
12	0	bad IP header

* some of the ICMP types and codes

- `ping` checks if remote host will respond (i.e. is there a connection?)
- `traceroute` sends small packets, with *different TTL*, to attempt to display routes to get to a remote host
  - might need to run a few times to get all possible routes

Link Layer

- Framing
- Link access control
- Error detection
  - not 100% reliable
- Error correction (without retransmission)
- Reliable delivery (often used on error-prone links like wireless)

Link layer is implemented in "adapter" (aka NIC) or on a chip - Ethernet card, or WiFi adapter. Adapters are semi-autonomous, implementing both link & physical layers.

Different link-layer protocols may be used on different links, each providing different set of services.

Data link layer has responsibility of transferring datagram from one node to *physically adjacent* node over a link.

Error Detection

Parity Checking: Single Bit

For *even parity* scheme, sender includes one additional bit that is such that the total number of `1`s in the `d + 1` bits is **even**. Can only detect *odd number* of single bits errors. Works very well in theory, but errors are clustered together in "bursts" in practice. So, the probability of *undetected errors* approaches 50%, not good enough!

Parity Checking: 2D

Split d bits into i rows and j columns. A parity value is computed for each row and column, and a final $d_{i+1,j+1}$ that is the parity bit for the column parity and row parity bits.

- *detect and correct* single bit errors in data (the row and column bits align)
- *detect* any two bit error (two possibilities)
- *detect* all three bit error, consider two bit, but try to hide it by using one of the corners, but can only hide one of the errors
- *cannot* detect all four bit error

$$\begin{array}{cccc|c} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 \end{array}$$

* correct example

Cyclic Redundancy Check (CRC)

- D: d data bits, viewed in binary
- G: generator of $r+1$ bits, pre-agreed by sender and receiver

- R: the r bit CRC, left-pad with 0 until r bits
- no carries for addition, no borrows for subtraction. both addition and subtraction becomes XOR
- take the D/G to get R. use long division
- sender sends (D, R), receiver performs $(D, R)/G$, if non-zero remainder, there's an error
- performance
  - detects *all odd number* of single bit errors
  - CRC of r bits can detect
    - all burst errors of less than $r+1$ bits
    - all burst errors of greater than r bits with probability $1-0.5^r$

CRC is hardware-optimised, but not efficient for software, so not used in transport and network layers.

Network Links

1. **point-to-point link**
   - sender and receiver connected by **dedicated link**
   - Point-to-Point Protocol (PPP), Serial Line Internet protocol (SLIP), no need for multiple access control
2. **broadcast link** (shared medium)
   - multiple nodes connected to shared broadcast channel
   - when a node transmits a frame, the channel *broadcasts* the frame, and every other node receives a copy.
   - collision if a node receives two or more signals at the same time (cannot distinguish!)

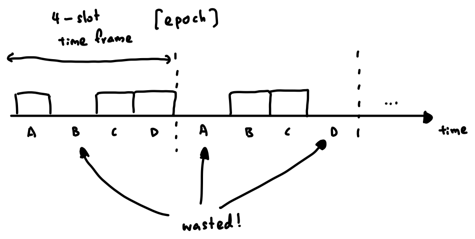
Ideal Access Protocol

- Collision Free
- Efficient: when only one node wants to transmit, it can send at rate R
- Fairness: when M nodes want to transmit, each can send at average rate R/M (equally shared)
- Fully Decentralised: no special node required (no single point of failure)
- Must use channel itself to coordinate (cannot use some out-of-band / external means of channel signalling)

Channel Partitioning Protocols

Time Division Multiple Access (TDMA)

- each node gets fixed length time slots in each round
  - length of time slot = data frame transmission time
  - if unused, time slot is idle
- used in GSM
- Collision Free: yes
- Inefficient
  - unused slots are idle
  - maximum throughput is R/N
- Fairness: perfectly fair
- Decentralised (but, there's a need to synchronise clocks, difficult to implement in practice because of latency)



Frequency Division Multiple Access (FDMA)

- channel spectrum is divided into frequency bands
- each node assigned a fixed frequency band
- if unused, frequency band is idle
- used in radio, satellite systems
- Collision Free: yes
- Inefficient
  - unused slots are idle
  - maximum throughput is R/N
- Fairness: perfectly fair
- Decentralised

Taking Turns Protocols

Polling

- master node polls each node in *round-robin* fashion
  - master informs node i it can transmit up to some maximum number of frames
  - assumes no malicious actor
  - star topology
- used for Bluetooth (phone is master, accessories are slaves)
- Collision Free: yes

- Almost fully efficient
  - overhead of polling, cannot achieve exactly R
- Fairness: perfectly fair
- Decentralised: NO, there exists a master node

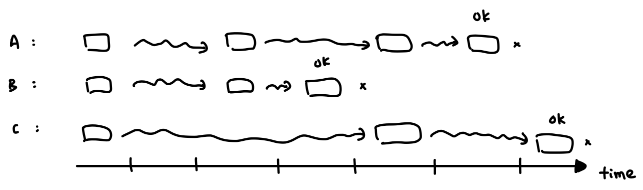
Token Passing

- token is passed from node-to-node sequentially
- when a node receives a token
  - if have frames to transmit: hold on to the token and sends up to a maximum number of frames and then forwards token to next node
  - else: forward token to next node immediately
- used in FFDI (Fiber Distributed Data Interface), Token Ring
- Collision Free: yes
- Almost fully efficient
  - overhead of token passing, cannot achieve exactly R
- Fairness: perfectly fair
- Decentralised: YES
- Downsides
  - token loss can be disruptive (due to data frame loss, system bugs)
  - node failure can break the ring
  - not a simple protocol to implement

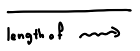
Random Access Protocols

- when a node has data to send
  - transmits at *full channel data* rate R
  - no *a priori* coordination among nodes
- protocol specifies
  - how to *detect* collisions
  - how to *recover* from collisions

Slotted ALOHA

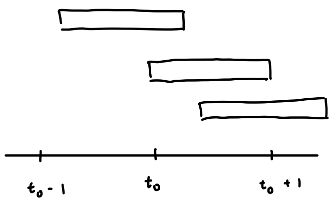


- all frames are of equal size, L bits
- time is divided into slots of equal length
  - length = time to transmit 1 frame = L/R
- nodes only start transmitting at *beginning of slot*
  - time is synchronised at each node
- when a node wants to send
  - wait until beginning of next time slot and transmits the entire frame
    - if no collision: success
    - if collision: retransmit the frame in each subsequent slot with probability p until success



- used in wireless packet switched network (ALOHAnet)
- Collision Free: no
- Efficient?
  - yes, when only one node is active
  - no, when there are many active nodes (because of collision) - maximum efficiency is only 37% with best chosen p
  - slots are wasted because of collision and being empty
- Fairness: perfectly fair
- Decentralised: Yes

Pure (Unslotted) ALOHA



ALOHA, but no time slots, no synchronisation.

- when a node wants to send
  - transmit *immediately*
    - if no collision: success
    - if collision: wait for 1 frame transmission time, retransmit with probability p until success
- chance of collision increases, frame sent at t_0 collides with $(t_0 - 1, t_0 + 1)$, doubles the range of collision
- same as Slotted ALOHA, but maximum efficiency is *worsened* to 18%

Carrier Sense Multiple Access

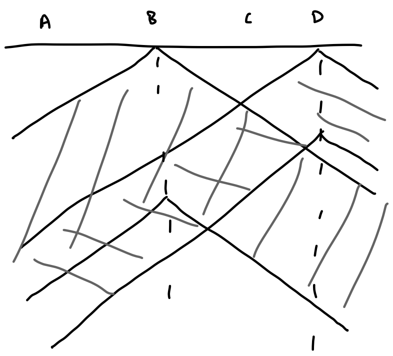
aka CSMA / CA (Collision Avoidance)

- before transmission:
  - if channel *sensed idle*: transmit entire frame
  - if channel *sensed busy*: defer transmission
- can still collide because of propagation delay (because nodes don't hear each other immediately)
- used by WiFi
  - half duplex: cannot listen and send at the same time
- not ideal if most nodes are likely to transmit frequently

CSMA / CD (Collision Detection)

- if channel *sensed idle*: transmit entire frame
- if channel *sensed busy*: defer transmission
- if *collision detected*: abort transmission
  - retransmit after a random delay using Binary Exponential Backoff
  - after mth collision, choose K at random from $[0, 2^m - 1]$
    - waits K frame units, each frame unit is 64 bytes = 512 bits (minimum frame size), take frame unit divide by speed of Ethernet to get frame unit
  - retransmission attempts to estimate current load
  - more collisions imply heavier load, so use a longer back-off interval
- used in (now-obsolete) Ethernet
  - full duplex because wired, can distinguish between parties
- has a $\underline{\text{minimum frame size}}$ so that can detect when collisions occur (Ethernet uses 64 bytes)
- Collision Free: no
- Efficient: yes
- Fairness: yes
- Fully Decentralised: yes

"thickness" of



* both B,D aborts upon detection

MAC (Media Access Control) Address

Every adapter (NIC) has a MAC address (aka physical or LAN address).

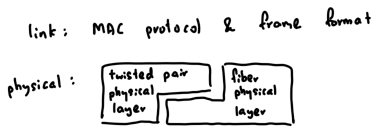
NIC = Network Interface Card; ROM = Read-Only Memory

- used to send and receive link layer frames
- when an adapter receives a frame, checks if destination MAC address of the frame matches its own
  - if yes, adapter extracts the enclosed datagram and passes it to protocol stack
  - if no, discard
- MAC address is typically 48-bits, burned in NIC ROM (sometimes software settable, but most devices don't let you configure it)
  - typically in hexadecimal, i.e. 5C-F9-DD-E8-E3-D2
  - MAC address administered by IEEE (first 3 bytes identifies vendor of an adapter)
  - broadcast address: FF-FF-FF-FF-FF-FF (all `1` s)
- duplicate MAC addresses in the same subnet is problematic and will be severely disrupted (doesn't matter if not in same subnet)

Local Area Network (LAN)

Computer network that interconnects computers within a *geographical area*.

- IBM Token Ring
- Ethernet (dominant) - everything is sent through The Ether
  - series of standards developed over the years with different speeds and physical layer media (cable, fibre optics)
  - MAC protocol and frame format remain unchanged
- WiFi
- etc


link: MAC protocol & frame format
physical:
twisted pair physical layer
fiber physical layer

Ethernet

Ethernet Frame Structure

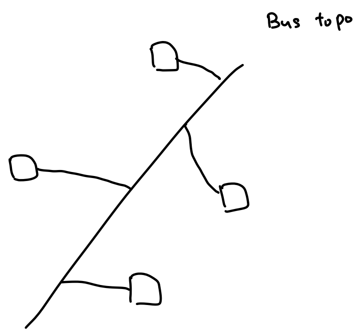
- preamble - 8 bytes
  - 7 bytes with `10101010` + 1 byte with `10101011` (start of frame)
    - only differs by the last bit
  - used to synchronise receiver and sender clock rates (the interval between these bytes provides a square wave pattern)
- destination address - 6 bytes
- source address - 6 bytes
- type - 2 bytes
  - indicates network-layer protocol used (might not be IP)
  - allows Ethernet to multiplex network-layer protocols

    - analogous to protocol field in network-layer datagram, port-number fields in transport-layer segment
- data - 46 to 1500 bytes
    - minimum size to ensure collision will always be detected
- CRC - 4 bytes

Ethernet Reliability

Unreliable because receiving NIC doesn't send ACK or NAK to sending NIC. Requires TCP usage in higher-layer to ensure this.

Ethernet Topology



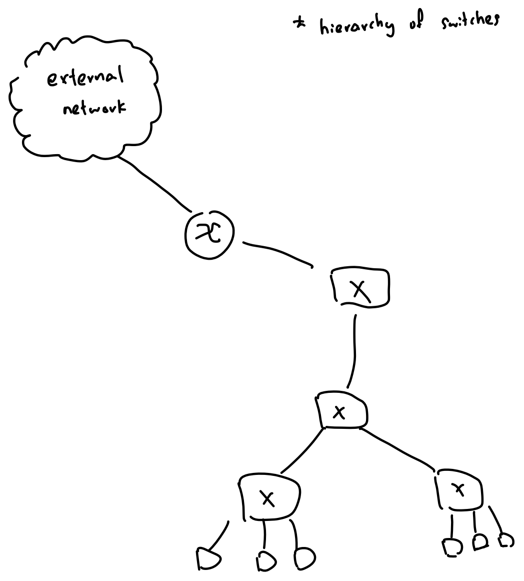
Bus Topology

- original Ethernet LAN used coaxial bus
- broadcast LAN
    - all transmitted frames received by all adapters connected to the bus (all nodes can collide)
- if backbone cable is damaged, entire network fails
- difficult to troubleshoot problems
- very slow and not ideal for larger networks (due to a lot of collisions)

Star Topology

- **Hub** - nodes physically connect to this
    - popular in 1990s, deprecated now
    - *physical-layer* device that acts on *individual bits* rather than frames
        - when a bit arrives: re-creates the bit, boosts its energy strength, transmits bit onto all other interfaces
    - cheap, easy maintenance (due to modular design)
    - very slow and not ideal for larger networks (due to a lot of collisions)
- **Switch** - nodes physically connect to switch
    - popular since 2000s
    - *data-link layer* device, acts on frames
    - no collisions, bona-fide store-and-forward packet switch
    - *selectively forwards* frame to one-or-more outgoing links (based on incoming frame's MAC address)
    - store and forward Ethernet frames
    - uses CSMA / CD to access link
    - transparent: hosts are unaware of presence of switches
    - do not need to be configured!
    - handles multiple simultaneous transmissions:
        - nodes have *dedicated, direct* connection to switch (no collisions!)
        - switches buffer packets
        - Ethernet protocol used on each incoming link (but no collisions)
        - A-to-B and C-to-D can transmit simultaneously
    - switches can be connected in hierarchy
    - unmanaged switches do not require MAC address or IP address

Switch - for LAN



Switch Forwarding Table

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

Self-learning.

- when frame is received, switch *learns* location of sender, records this in the table
- when frame is received
    - record incoming link, MAC address of sending host
    - index switch table using MAC destination address
        - if entry is found
            - if destination on segment from the frame which arrived: drop the frame
            - else: forward frame to interface stored in entry (unicast)
        - else: flood (broadcast) to all interfaces (except arriving interface)
            - if it's a switch, that switch will handle it
            - if it's a host, that's already the host

ARP - Address Resolution Protocol

Find 48-bit MAC address with 32-bit IP address.

Each IP node has an ARP Table, <IP address, MAC address, TTL>

Router, Host all require ARP table. Switch only requires Forwarding Table.

Only stores information for *some* neighbouring hosts in *same subnet* (LAN).

- Same Subnet
    1. If A knows B's MAC address from its ARP table
        1. Create a frame with B's MAC address and send it
        2. Only B will process this frame (others will ignore even if they receive it)
    2. A does not know B's MAC address
        1. A broadcasts an ARP query packet, containing B's IP address
            1. destination MAC address of `FF-FF-FF-FF-FF-FF` (broadcast address)
            2. All other nodes (in subnet) will receive it, but only B will reply
        2. B replies to A with its MAC address (unicast to A's MAC address)
        3. A caches B's IP-to-MAC address mapping in its ARP table (until TTL expires)
        4. **Then**, do as for case 1

   has diff. MAC, IP for each interface
- Different Subnet - Gateway Router R will handle connection between subnets
    1. A creates IP datagram with IP source A, destination B
    2. A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram
        1. If don't know R's MAC address, do ARP for it
    3. R receives frame and passes the datagram to IP
    4. R forwards datagram with IP source A, destination B
    5. R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram (only Ethernet frame changes, IP header and everything above **doesn't** change)

Network Security

Somewhere between Application and Transport layers.

- **confidentiality**: no eavesdropping, i.e. only sender and intended receiver should understand the content
- **authentication**: sender and receiver can *confirm identity* of each other
- **message integrity**: any alteration must be *easily detected*
- **access and availability**: services must be accessible and available to users

Cryptography

- Symmetric Key Cryptography = sender and receiver uses *same key*
    - need to agree on key
- Asymmetric Key Cryptography (aka Public Key Cryptography) = sender and receiver uses *different key*
    - *do not* share secret key, encrypt with public, decrypt with private (or vice versa)
    - slow compared to symmetric
    - in practice, one party creates a symmetry key, and uses RSA to transfer it, then uses that symmetric key as the session key

Ciphers

Cipher text shouldn't reveal *any* information about the original message. Has been shown that the key needs to be at least as long as the original message to leak 0 information.

- Caesar's Cipher: each character `+k` position (wrapped-around), only 25 possible values
- Mono-alphabetic Cipher: mapping of 26 letters to set of 26 letters, only 26! mappings
    - broken through *Statistical Analysis* by using letter frequency
- Poly-alphabetic Cipher: use multiple mappings that cycle
    - define n substitution ciphers and a cycling pattern, e.g. C_1, C_3, C_4, C_3, C_2 (can repeat, and this pattern cycles on infinitely)
    - character 1 uses first, character 2 uses second, etc... (space doesn't increment the pattern)
- Block Cipher: process in K-bit blocks, each block is encrypted independently using a one-to-one mapping, 2^K! number of keys
    - DES: Data Encryption Standard, with 56-bit symmetric key, 64-bit block (broken in less than a day)
    - 3DES: DES but encrypt 3 times with 3 different keys
    - AES: Advanced Encryption Standard, 128, 192, 256 bit keys, 128-bit block

Breaking Encryption

- ciphertext only attack: attacker has ciphertext to analyse
- known plaintext attack: attacker has plaintext corresponding to ciphertext
- chosen plaintext attack: attacker can get ciphertext for any chosen plaintext

RSA: Rivest, Shamir, Adleman Algorithm

1. choose two large prime numbers, p, q (1024, or 2048 bits long)
2. compute $n = pq$, $z = (p-1)(q-1)$

3. choose e ($e < n$) such that it has no common factors with z, aka relatively prime (randomly generate until find a valid e, OR generate a prime number)
4. choose d such that $ed \mod z = 1$ ($ed - 1$ is exactly divisible by z)
5. public key is (n, e), private key is (n, d)

Encryption & Decryption

- to encrypt message m ($m < n$): compute $c = m^e \mod n$
- to decrypt encrypted c: compute $m = c^d \mod n$
- $(m^e \mod n)^d \mod n = m$

Message Integrity

Uses Cryptographic Hash Functions

- $H(.)$ takes large (unbounded size) input m and produces fixed-size message digest / fingerprint
- computationally infeasible to find any two different messages x and y such that $H(x) = H(y)$.
    1. pre-image resistance
        1. given hash output, can't find original message
    2. second pre-image resistance
        1. given hash(m1) = h, very hard to find m2 such that hash(m2) = h as well
    3. collision resistance
        1. hard to find m1 and m2 such that hash(m1) = hash(m2), and m1 != m2
- small change in input should result in large change in hash output
- passwords are hashed using similar techniques, that's why you cannot recover the original password
- common ones
    - MD5 (cryptographically broken), still used for non-critical uses (e.g. websites share MD5 hash for file downloads)
    - SHA-1 (deprecated, cryptographically broken)
    - SHA-2, SHA-3

Message Authentication Code

- Cannot just send $(m, H(m))$ because attacker can replace with $(m', H(m'))$ and it won't be detected
- Send $(m, H(m + s))$ instead, where s is the authentication key

Digital Signature

- digital signature: sign with private, decrypt with public
    - verifiable: receiver can check the signature
    - unforgeable: no one else can generate this message (because private key) - non-repudiation (sender can't claim he didn't send it)
- optimisation: only sign message digest $H(m)$ because signing long stuff is more expensive
    - good for bandwidth & computation; still sufficient

Certificate Authorities (CA)

CA maintains public database of everyone's public key.

CA signs its messages with its own private key. The public key of CAs are hardcoded into Operating System, as a list of *Trusted Root Certificate Authorities*.

- E (person or router) registers its public key with CA
    - E provides a proof of identity to CA
    - CA creates certificate binding E to its public key, this contains E's public key digitally signed by CA

Firewalls

Firewall *isolates* organisation's internal net from larger Internet, allowing some packets, while blocking others.

- prevents DoS (Denial of Service) attacks: SYN flooding with bogus TCP conenctions
- prevents illegal modification / access to internal data
- allow only authorised access to inside network
- internal network connected to Internet via router firewall (both IN and OUT)
- router filters *packet-by-packet*, decides to forward / drop packets based on (network & transport layer)
    - source & destination IP address
    - TCP / UDP source and destination port numbers
    - ICMP message type
    - TCP SYN & ACK bits
- examples
    - no outside web access: drop all outgoing packets to any IP address, port 80

    - prevent 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 network from being tracerouted: drop all outgoing ICMP TTL expired traffic
- limitations
    - IP spoofing: router can't know if data really comes from claimed source
    - can become bottleneck (can use multiple firewalls instead, but costly)
    - tradeoff: degree of communication with outside world VS level of security (e.g. nuclear rocket computers **not** connected to Internet)
    - many highly protected sites still suffer from attacks (might provide a false sense of security)

Access Control Lists (ACL)

Table of rules applied top to bottom.

action	source address	destination address	protocol	source port	destination port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
...	...	...	...	...	...	...
deny	all	all	all	all	all	all

Multimedia Networking

Application Types

- streaming *stored*: can begin playout before downloading entire file
    - stored at server / CDNs, can transmit faster than audio / video will be rendered
    - pre-processing done to optimise this
- conversational (*two-way live*): **interactive**
    - delay more than 400ms is intolerable
- streaming *live* (don't have to be real-time, 10s delay is fine)
    - typically done with CDNs

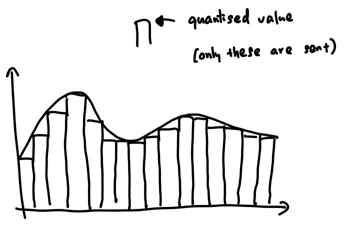
Data is transmitted over-the-top (over the Internet, as compared to cable / broadcast / satellite)

* applicable for both video and audio

Video

Videos are sequences of images displayed at a some constant rate. Digital image: array of pixels, each pixel represented by bits. Videos have high bit rate. To reduce data usage, compress the video by using redundancy.

- reducing redundancy
    - Spatial coding (within image): aggregate pixels of the same colour
    - Temporal coding (between two frames): only send the delta of the frames
    - reduces the size of each frame, number of frames remain the same
- bit rate
    - Constant Bit Rate (CBR): *fixed* video encoding rate (might be wasteful)
        - not responsive to complexity of video
        - bitrate need to be high enough to handle the most complex segments of video
        - well-suited for real-time encoding because of consistency
    - Variable Bit Rate (VBR): encoding rate changes with spatial, temporal coding
        - well-suited for on-demand video because can pre-process data, and can buffer data



Audio

Sampled at constant rate, each sample is quantised (rounded) to a value represented by bits. (i.e. 256 quantised values require 8 bits, so with 8,000 samples / sec ⇒ 8,000 * 8 = 64,000 bps)

Streaming Stored Content

- constraints
    - Continuous Playout Constraint: once client playout begins, playback must match original timing, even with network delay being variable.
    - allow client interactivity: pause, fast-forward, rewind, jumping, etc...
    - video packets may be lost / retransmitted
- application playback buffer to buffer content, let fill rate = $x(t)$, average fill rate = $\bar{x}$, playout rate = r
    - if $\bar{x} < r$: buffer eventually empties (video freezes until buffer fills)
    - if $\bar{x} > r$: buffer will not be empty provided initial playout delay is large enough to absorb variability in $x(t)$
    - initial playout delay tradeoff:

        - buffer starvation less likely with larger delay
        - but larger delay until user can start watching
    - buffers data for smooth playback, client can play any part of this buffer (**NOT** the same as TCP buffer, TCP buffer is for storing out-of-order segments)

RTP vs RTCP

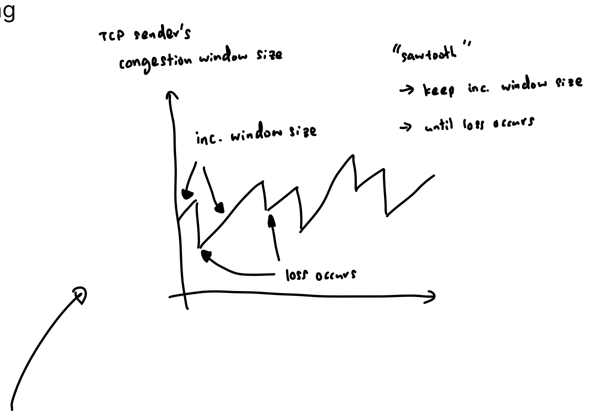
- **RTP** = Real-Time Transport Protocol
    - Delivers interactive data
        - Pure data only
    - Purpose: time-stamp, sequence and mixing (combining video with audio)
    - *UDP* because speed is most important, don't need to be reliable
- **RTCP** = RTP Control Protocol
    - Delivers *statistics and control information* (non-interactive)
        - How many bits, quality of bitstream
    - Delivered over next odd UDP port number that the RTP stream uses
- **RTSP** = Real-Time Streaming Protocol
    - Used to *establish* and *control* media sessions during endpoints.
        - Setup
        - Play / stop
        - Change quality
    - *TCP* because requires reliability, but not necessarily speed
        - Need to make sure the setup is correct

Streaming through UDP

- server sends at (often constant) rate appropriate for client
    - push-based streaming (server push)
- UDP has no congestion control
    - occasional loss of data won't affect future data because no-resending
- only requires short playout delay (2-5 seconds) to remove network jitter
- error recovery depends on application level

Streaming through HTTP

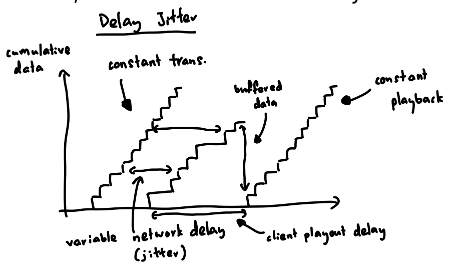
- multimedia file retrieved via HTTP GET
    - pull-based streaming (client pull)
- sends at maximum possible rate under TCP
- advantages
    - HTTP / TCP passes more easily through firewalls
    - network infrastructure (CDNs and Routers) are optimised for TCP
- drawbacks
    - fill rate fluctuates due to TCP congestion control, retransmissions (need to ensure in-order)
    - larger playout delay required



Voice Over IP (VoIP)

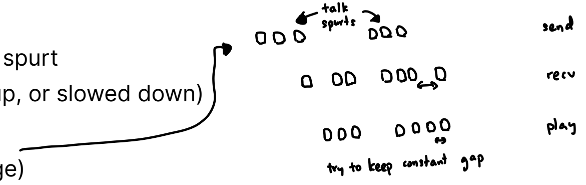
- higher delays are noticeable, impairs interactivity
    - < 150ms: good, > 400ms: bad
    - includes application level delays
        - network loss: IP datagram lost due to network congestion
        - delay loss: IP datagram arrives too late for playout (just discard)
            - typically use UDP
- data loss over 10% makes conversation unintelligible
    - modern networks rarely have data losses > 10%, so this isn't really an issue
- only sends chunks when during *talk spurts* (vs *silent periods*); 20ms chunks at 8Kbytes / second: 160 bytes of data
- application-layer header added to each chunk: chunk + header is then encapsulated into UDP or TCP segment
- packet lengths are ~20ms (fraction of a syllable!); low to minimise impact of packet loss, at a tradeoff for efficiency

Fixed Delay Playout



- receiver attempts to playout each chunk exactly q ms after chunk was generated
    - if chunk has timestamp t, play at $t + q$
    - if chunk arrives after $t + q$: discard
    - no value of q is optimal
        - larger q: less packet loss

    - smaller q: better interactive experience
  - estimate network delay, adjust playout delay at beginning of each talk spurt
    - silent periods compressed and elongated (audio might feel sped up, or slowed down)
    - chunks still played out every 20ms during talk spurt



  - adaptive playout delay: EWMA (Exponentially Weighted Moving Average)
    - delay estimate after ith packet = $d_i = (1-\alpha)d_{i-1} + \alpha(r_i - t_i)$, r_i is time received, t_i is timestamp (time sent), α is small constant (e.g. 0.1)
    - estimate of average deviation of delay after ith packet = $v_i = (1-\beta)v_{i-1} + \beta|r_i - t_i - d_i|$
    - calculated for every received packet, but used only at start of talk spurt: playout-time for ith packet = $t_i + d_i + 4v_i$ (so $q = d_i + 4v_i$)
- every chunk has (1) sequence number & (2) timestamp (implemented in application layer)

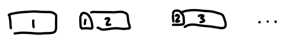
Forward Error Correction (FEC)

Using ACK / NAK is too slow because one RTT is too slow!

Simple FEC

- for every group of n chunks
  - create redundant chunk by XOR-ing n original chunks
  - send $n+1$ chunks (including this redundant chunk)
- can reconstruct original n chunks if at most one lost chunk, lost chunk = XOR of everything that was received
- drawback: increased bandwidth by $1/n$ factor, and receiver has to wait for an extra chunk before playout

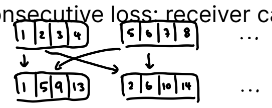
Piggyback Scheme



Send lower resolution audio stream as redundant information, i.e. nominal stream PCM at 64kbps, but redundant stream GSM at 13kbps.

- non-consecutive loss: receiver can conceal loss

Interleaving



- no redundancy overhead, but increases playout delay (need wait for more chunks), even without error

HTTP Streaming

Video on Demand (VoD) video streaming increasingly uses HTTP streaming. Can just GET the whole video file, but wasteful because requires a large client buffer. Also, cannot have different encoding of videos to account for variation in device and network bandwidth.

Most VoD uses either DASH or Apple's HLS (HTTP Live Streaming).

DASH

Dynamic, Adaptive Streaming over HTTP.

- **Server**
  - divides video file into multiple chunks
  - each chunk stored and encoded at different rates
  - *manifest file* (Media Presentation Description - MPD): provides URLs for different encodings
- **Client**
  - periodically measures server-to-client bandwidth
  - consults manifest to request one chunk at a time
    - chooses maximum coding rate sustainable for current bandwidth - Adaptive Bitrate Algorithm (ABR) (user can override this)
    - can change coding rates at different points in time

Client determines

- *when* to request chunk, to avoid buffer starvation & overflow
- *what* encoding rate to request
- *where* to request chunk (picker closest URL server that has the highest availability)

Pros & Cons of DASH

- pros
  - server is simple: no state to maintain
  - no firewall problem because HTTP
  - standard image web caching works
- cons
  - DASH is based on media segment transmissions, typically 2-10 seconds in length
  - by buffering a few segments at the client side, DASH cannot support low latency for interactive two-way applications (Zoom calls)

Content Distribution Networks (CDNs)

1. single large mega-server (doesn't scale!)
    - single point of failure, and network congestion
    - potentially long path to distant clients
    - multiple copies of videos over outgoing link
2. store / serve multiple copies of videos at multiple geographically distributed sites (CDN)
    - enter deep: push CDN servers deep into many access networks (typically at ISPs, close to users)
    - bring home: smaller number (10's) of larger clusters in IXPs near, but not within access networks

Misc

- TCP / UDP Segment = TCP / UDP Header + Data
- IP Datagram = IP Header + Segment
- Link Layer Frame = encapsulation of IP datagrams in header + footer
- Packet = no strict definition (so, it's all of these)

Fin >‿<