CS2105 | Yap Zher Xiang Jason

Chapter 1

Internet

The internet is a network of connecting computing devices. (e.g hosts/end systems)

Network Edge (Access Network)

- Hosts access the Internet through **access network**.
- Wireless access network connects hosts to router
  - Wireless LANs - WiFi
  - Wide-area wireless access - 3G, 4G etc.
- Physical Media — the way hosts connect to access network.
  - Guided media — signals propagate in solid media (wires/fibre optics)
  - Unguided media — signals propagate freely (radio)

Network Core — mesh of interconnected routers

Data is transmitted through the network by

Circuit switching — used by traditional telephone networks

- End to end resources allocated to and reserved.
- Set-up and teardown required.
- Circuit like (guaranteed) performance, as the circuits are reserved per usage.
- Unable to share the resources.

Packet switching

- No set-up.
- Resources not reserved.

Hosting sending function:

- Breaks the application into smaller chunks, known as packets of length *L* bits
- Transmits packets onto the link at transmission rate *R*
  - Link transmission rate == `link capacity` or `link bandwidth`
  - `packet transmission delay` = `time needed to transmit L-bit packets into link`

$$= \frac{L(bits)}{R(bits/sec)}$$

Store and forward:

- Packets passed from one router to the next, across links on path from source to destination.
- Entire packet must arrive at the router before it can be transmitted on the next link.
- `End-to-end delay` = $2 * \frac{L}{R} (assuming\ no\ other\ delay)$

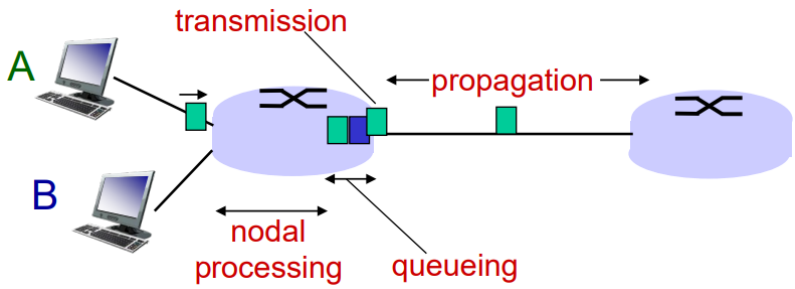
Routing and addressing

- Routers determine the source-destination route taken by using routing algorithms
- Addressing: each packets contains their **source and destination** information.

Internet Structure: Network of Networks

- Organised into **Autonomous Systems (AS)** owned by an organisation.
- AS → Regional ISPs → Access ISPs → End users

Delays



- d_{proc} — Processing Delay
- d_{queue} — Queuing Delay

- check bit errors
- determine output link
- typically < msec

- time waiting in queue for transmission
- depends on congestion level of router

- $d_{trans} = \frac{L}{R}$ — Transmission Delay
  - *L*: packet length (bits)
  - *R*: link bandwidth (bps)
  - Size of packet

- $d_{prop} = \frac{d}{s}$ — Propagation Delay
  - *d*: length of physical link
  - *s*: propagation speed in medium ($2 \times 10^8\ m/sec$)
  - Distance of travel

- $d_{end-to-end}$ = sum(d_{proc}, d_{queue}, d_{trans}, d_{prop})

- $RTT = 2 \times d_{prop}$ — Round Trip Time

Throughput

How many **bits can be transmitted per unit time**

- Measured for end-to-end communication.
- Link capacity (bandwidth) is meant for a specific link.
- `SOURCE(sending F bits)====(bw=R_s bits/sec)===>SWITCH===(bw=R_c bits/sec)==>DESTINATION`
- $d_{trans_1} = \frac{F}{R_s}$
- $d_{trans_2} = \frac{F}{R_c}$

$$Throughput = \frac{F}{d_{trans_1} + d_{trans_2}}$$

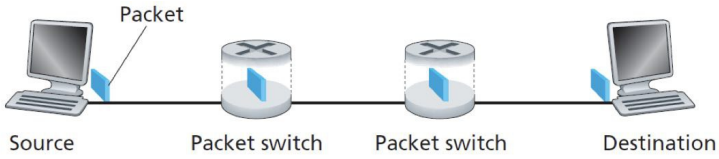
Internet Protocol Stack

1. **application**: supporting network applications (FTP, SMTP, HTTP)
   - moving data to their peer, ensuring data is understood
2. **transport**: process-to-process data transfer (TCP, UDP)
   - handle getting a byte stream form **socket X** to **socket Y**
3. **network**: routing of datagrams from src to dest (IP, routing protocols))
   - handle getting a packet from **point A** to **point B**
4. **link**: data transfer between neighbouring network elements (Ethernet, WiFi)
5. **physical**: bits on the wire

Questions:

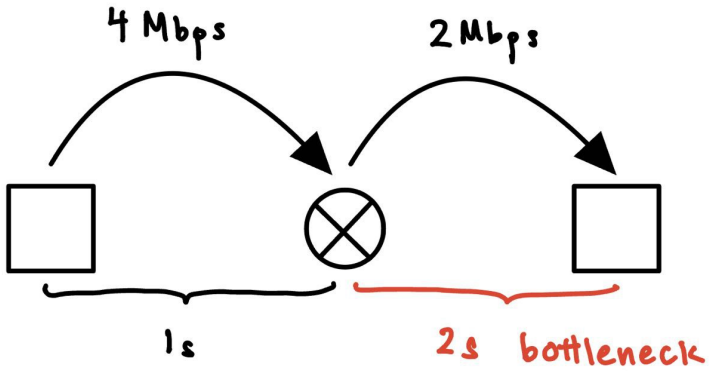
1. It takes 5ms to transmit a packet between each switch. Suppose a total of 800 packets needs to be transmitted to the destination, how long would it take.

$$1st\ packet = 15ms\ (3 * 5ms)$$
$$After,\ every\ 5ms\ one\ more\ packet\ arrives.$$
$$Thus,\ 800^{th}\ packet\ reaches\ at: 15 + (799 * 5) = 4010ms$$



2. Different bandwidth/link rates vs throughput

$$time\ 1st\ pkt\ reaches\ link = \frac{4}{4} = 1s$$
$$time\ 1st\ reaches\ dest = \frac{4}{2} = 2s$$
$$time\ next\ 99\ pkts\ reach\ dest = 99 * 2s\ (bottleneck)$$
$$total\ time = 1 + 2 + (99 * 2) = 201s$$



Chapter 2 & 3: Application Layer

Client-Server Architecture

Client	Server
Initiates contact with Server	Waits for incoming requests
Requests service from server	Provides requested service to client

P2P Architecture

- No always-on server, arbitrary end systems directly communicate.
- Peers request service from other peers, provide service in return to other peers
- Complex to manage as peers are intermittently connected, and may change IP

HTTP - Uses TCP as a transport service

ALL HTML Files **MUST** be downloaded first, before downloading other objects regardless of the version of HTTP.

HTTP 1.0

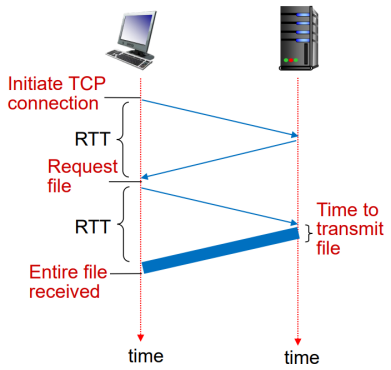
- **Non-persistent**
  - New TCP connection request has to be sent for each web resource/object
    - At most one object sent per TCP connection
    - Downloading multiple objects require multiple connections
  - Requires 2 RTTs per object
- **Sequential**
  - Files have to be loaded 1-by-1
  - Massive overhead

RTT: time for a packet to travel from client to server and go back

HTTP response time:

- one RTT to establish TCP connection
- one RTT for HTTP request and the first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time =

  2 * RTT+ file transmission time



HTTP 1.1

- **Persistent**
  - Server **leaves connection open** after sending response
  - Only requires **1 TCP connection** to load **ALL** required server objects
  - As little as one RTT to *fetch* referenced objects (if with pipeline)
- **Pipelined**
  - New requests can be made even before receiving responses for older requests
  - They still need to be received back in the order of request (head-of-line blocking)

HTTP Request

- `OPTIONS`: request for information about the communication options available on the request/response chain
- `GET`: retrieve whatever information identified by the Request-URI
- `HEAD`: identical to `GET` except that the server *MUST NOT* return a message-body in the response (ie. only headers are sent)

- `POST`: sends data to a specific URI and expects the resource at that URI to handle the request
  - The web server at this point can determine what to do with the data in the context of the specified resource
- `PUT`: puts a file or resource at a specific URI, and exactly at that URI
  - If there's already a file or resource at that URI, `PUT` **replaces** that file or resource
  - If there is no file or resource there, `PUT` creates one
- `DELETE`: requests that the origin server delete the resource identified by the Request-URI
- `TRACE`: invoke a remote, application-layer loop- back of the request message

```
GET /~cs2105/demo.html HTTP/1.1\r\n
HOST: www.comp.nus.edu.sg\r\n
User-Agent: Mozilla/5.0\r\n
Connection: close\r\n
\r\n # Empty line marks end of headers
```

- Line 1 is the request line, `<METHOD, URI, HTTP_VERSION>`
  - `METHOD`: `GET`, `POST`, `PUT`, etc.
  - `URI`: Uniform Resource Identifier
  - `HTTP_VERSION`
- Line 2-4 are the header lines,
  - `HOST`: where the object resides (`HOST` + `URI` = URL)
  - `User-Agent`: the browser type making the request
  - `Connection`: whether to keep the connection persistent or not

HTTP Response

```
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
# Start of content body
DATA DATA ... DATA
```


- Line 1 is the status line
  - Contains response code (`200 OK`, `403 Forbidden`, `404 Not Found`,)
- Lines 2-4 are the header lines,
  - `Date`: refers to date and time at which the message was originated
  - `Content-Length`: Number of bytes of data in the content body (line 7 onwards)

Conditional GET

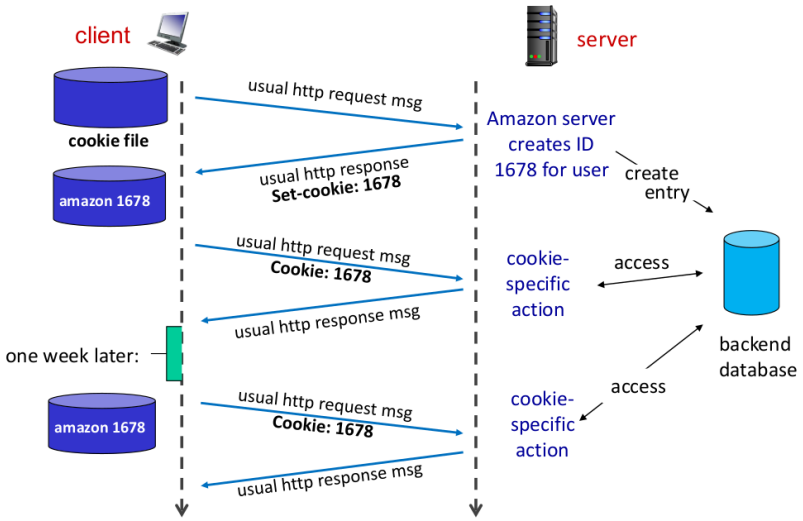
Objects shouldn't be sent if client has an up-to-date cached version

1. Client sends date of cached copy when requesting : `If-modified-since: <DATE>`
2. Server respond with `304 Not Modified` with **empty** content body

Code	Details
200 OK	Request succeeded
201 Created	Resource is created
202 Accepted	Indicates that request has been accepted for processing, but the processing has not been completed
204 No Content	Server declines to send back any status message in response message's body
301 Moved Permanently	Requested object moved, new location specified later in message
304 Not Modified	Sent during a conditional GET
400 Bad Request	Generic error code: syntax errors, invalid parameters etc.
401 Unauthorised	Client tried to operate on a protected resource without providing the proper authorisation
403 Forbidden	Server declines to show the requested webpage
404 Not Found	Requested document not found on this server
500 Internal Server Error	Server's own runtime exception; never the client's fault

Cookies

- HTTP is designed to be stateless (server maintains no info about past client requests)
- Cookies are **used to maintain state**



Domain Name System

- Translates the hostname: `www.example.org` to IP address: `93.184.216.34`
- Client must carry out a DNS query to determine the IP address corresponding to the hostname prior to connection.
- Runs on **UDP**
  - UDP is much faster than TCP since 3-way handshake is needed for TCP
  - Much smaller packet header overhead for UDP (most DNS queries are local, smaller chance of packet lost)
- **Recursive Queries**:
  - DNS client requests information from a DNS server that is set to query subsequent DNS servers until a definitive answer is returned to the client
  - Subsequent DNS queries made from first DNS server are iterative queries
  - Vulnerable to:
    - DDOS attacks
    - DNS cache poisoning
    - Easily hijacked

    - Root name server performance degradation
- **Iterative Queries**:
  - Ask a central hub DNS to search the other DNS (querying each DNS server one by one)
  - DNS server is queried and returns an answer without querying other DNS servers, even if it cannot provide a definitive answer
- **Why not centralize DNS**
  - single point of failure
  - traffic volume
  - distance centralized database (long RTT delays to further places on the planet)
  - maintainence

RR (Resource Record)

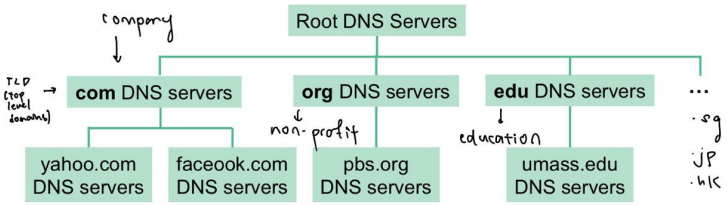
- Mapping between host names and IP addresses
- format: `<NAME, VALUE, TYPE, TTL>`
  - `TTL`: Time to live
  - `CLASS`: Almost always `IN` for Internet
  - eg. `www.google.com. 196 IN A 74.125.68.104`

TYPE	NAME	VALUE
A	hostname	IP Address
NS	domain (eg. nus.edu.sg)	hostname of authoritative name server
CNAME	alias name	canonical (real) name
MX	hostname	name of mail server

Distributed, Hierarchical Database

- DNS stored RR in distributed database implemented in hierarchy of many name servers.
  - Root → TLD → Authoritative servers (organisation's own servers) → Local DNS servers

- Client wants IP address for `www.facebook.com`
  - client queries root server to find .com DNS server
  - client queries .com DNS server to get facebook.com DNS server
  - client queries facebook.com DNS server to get IP address for www.facebook.com



DNS Caching

- Resources cached as RR, stored on local DNS servers
  - Cached entries may be out of date and only expire after `TTL` expires
  - If name host changes IP address, it may not be know until ALL TTLs expire
  - Every DNS query from host is first sent to local DNS server
    1. If server has cached RR, just return the RR
    2. If server does not have cached RR, local DNS server acts as proxy to forward query into hierarchy
- DNS Cache poisoning
  - (a type 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. This diverts traffic to the attacker's computer.
  - e.g DDoS (Distributed Denial of Service Attack) on a particular machine can be achieved via DNS poisoning.

Socket Programming

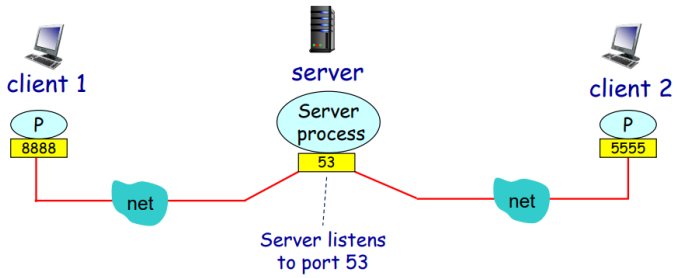
UDP: Client/server socket interaction

UDP is connectionless, client needs to explicitly attach the destination IP address and port number to each packet. The receiver is also able to extra the sender IP address and port number from, the received packet. Uses `SOCK_DGRAM` sockets (Datagram).

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

UDP: no "connection" between client and server

- Sender (client) explicitly attaches destination IP address and port number to each packet.
- Receiver (server) extracts sender IP address and port number from the received packet.



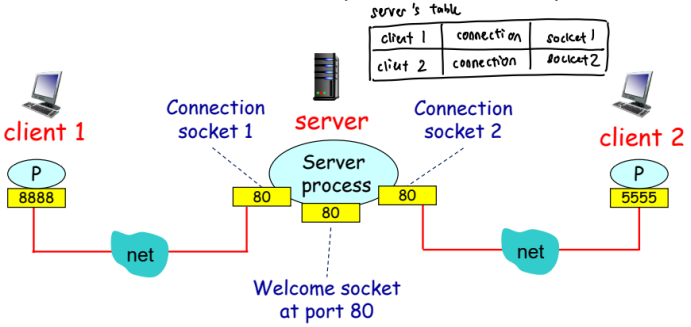
TCP: Client/server socket interaction

TCP is a connection-oriented protocol. 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 to communicate SOLELY with that client. The two hosts can just keep communicating with one another via this connection (door) until either side closes. Uses `SOCK_STREAM` sockets. (byte stream-orientated socket).

1. (Server) Create `serverSocket`, port = `x`.

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

- When client creates socket, client TCP establishes a connection to server TCP.
- When contacted by client, server TCP creates a new socket for server process to communicate with that client.
  - allows server to talk with multiple clients individually.



Chapter 4 & 5: Transport Layer

Transport Layer Services

- Deliver messages between application processes running on different hosts
  - 2 popular protocols: **TCP** and **UDP**
- Transport layer protocols run in hosts
  - **Sender side:** breaks app message into *segments*, passes them to network layer
  - **Receiver side:** reassembles segments into message, passes it to app layer
  - **Packet switches (routers) in between**: only check destination IP address to decide routing
- **IP** datagram contains source and destination IP addresses

Two principle internet protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP**: User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of “best-effort” IP
- services not available:
  - delay gurantees
  - bandwidth gurantees

UDP: User Datagram Protocol

Services on top of IP:

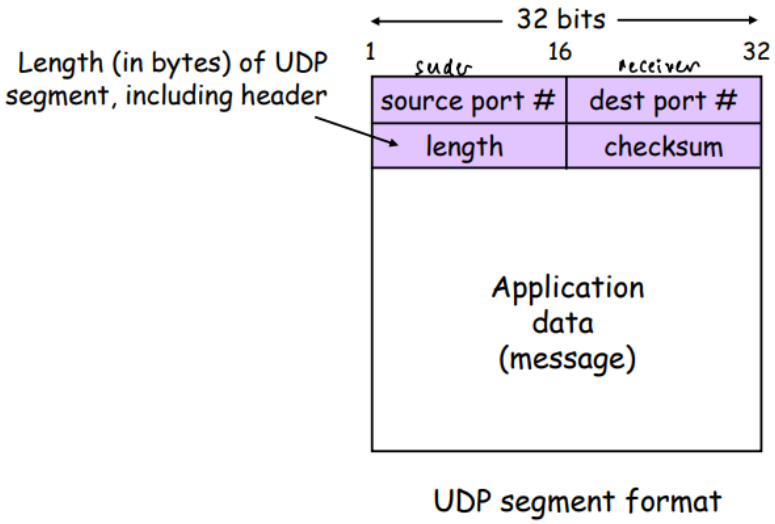
- **Multiplexing:** UDP gathers data from processes, form transport-layer segments that include the application data, source port number and the destination port number and passes them to network layer
- **Demultiplexing:** Examines the destination port number of the segment and passes the segment to the socket identified by it
  - Checks destination port number in segment
  - Directs UDP segment to the socket with that port number
  - IP datagrams (from different sources) with the same destination port number will be directed to the same UDP socket at destination

- **Checksum:** Used to check bit error

UDP Header

Why UDP?

- No connection establishment (which could add to delay)
- Simple: no connectino state at sender, receiver
- Small header size
- No congestion control: UDP can blast away as fast desired



UDP segment format

UDP Checksum

Goal: to detect "error" (i.e., flipped bits) in transmitted segment.

Sender:

- compute checksum value (next page)
- put checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed check sum equals checksum **field value**

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 bti 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

- The network layer provides host-to-host, best-effort and unreliable communication
- The transport layer resides on end hosts and provides process-to-process communication

> 💡 Host to Host communication is the one that usually takes place between one host and another host i.e., the computer. **IP protocol** is used in this method.
>
> The Process to Process communication means the communication between the two specific processes on the hosts. **UDP protocol** is used in this method to convey the given message from the host to the process.

- We thus need to build a **reliable transport layer protocol** on top of **unreliable communication**.

Unreliable channel network may:

1. corrupt packets ⇒ bits might be flipped wrongly
2. drop packets ⇒ dropped if the queue is full
3. re-order packets ⇒ packets take different route to packet switches
4. deliver packets after an arbitrarily long delay

rdt 1.0: Perfectly Reliable Channel

Assumption: Underlying channel is perfectly reliable

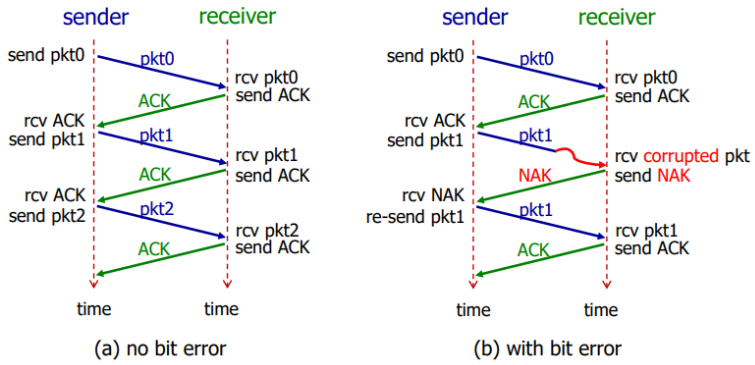
- Separate FSMs for sender, receiver:

- Sender sends data into underlying (perfect) channel
- Reciever reads data from underlying (perfect) channel

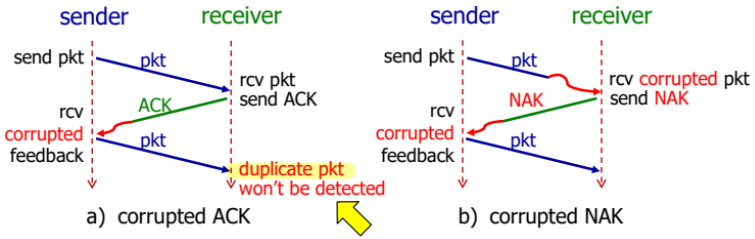
rdt 2.0: Channel with *Bit Errors*

Assumption: Underlying channel may flip bits in packets, other than that channel is perfect. Uses the **stop and wait** protocol.

- How to detect bit errors ⇒ Receiver use *checksum* to detect bit errors
- How to recover from bit errors ⇒
  - **ACKs**: receiver explicitly tells sender that packet received is OK
  - **NAKs**: receiver explicitly tells sender that packet has errors.
    - Sender retransmits packet on receipt of NAK

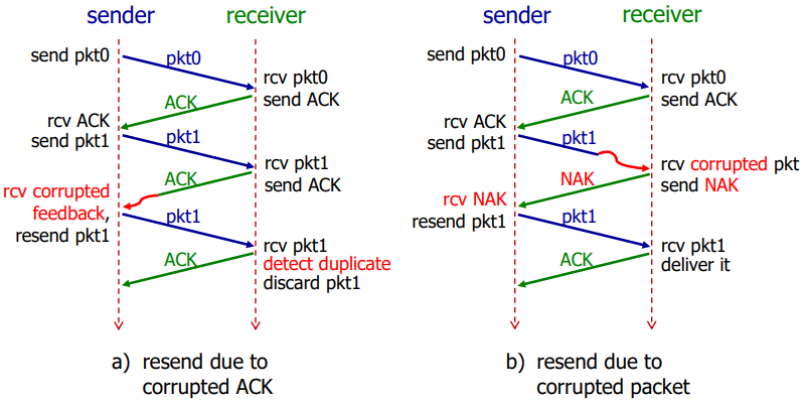


Problems: If ACK or NAK are corrupted, there is no guarantee way to recover data. If we simple resend the packet, the receiver will not know if it is a duplicate or not.



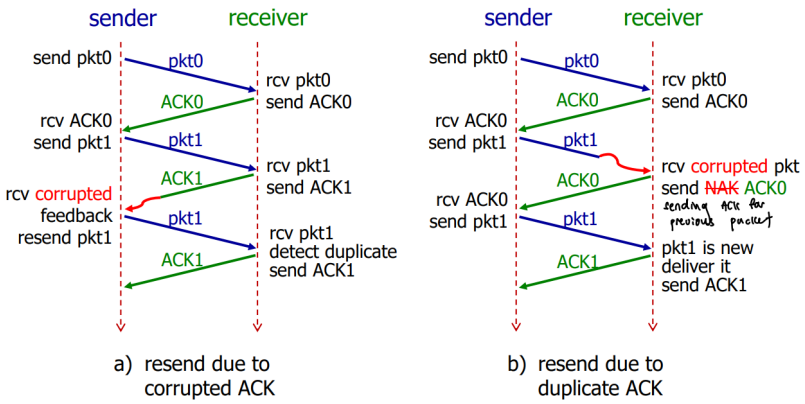
rdt 2.1: rdt 2.0 + Packet Sequence Number

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 as receiver discards (does not deliver up) duplicate packet.



rdt 2.2: a NAK-free Protocol

Assumption: Same as rdt 2.0 and 2.1 - corruption. In addition to what is covered in rdt 2.1, we not explicitly include the sequence number of the packet being ACKed, removing the need for a NAK. From the sender's perspective, we basically resend current packet if a **duplicate ACK is received**.



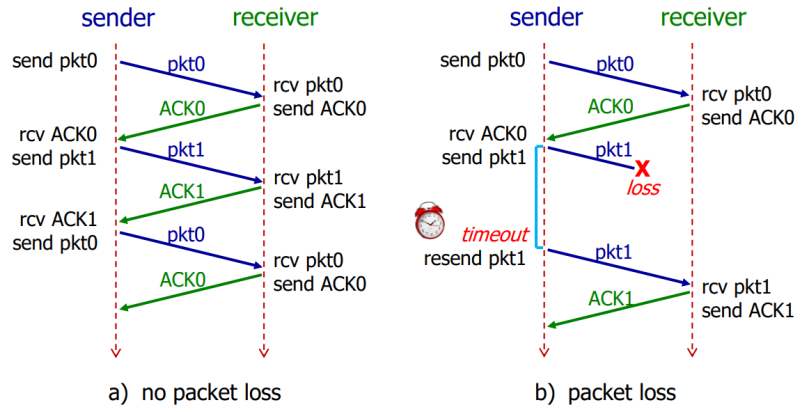
rdt 3.0: Channel with *Errors* and *Loss*

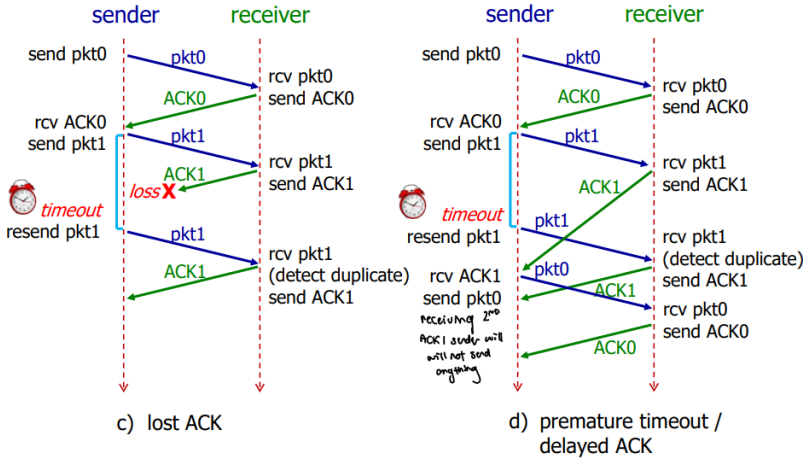
Assumption: Underlying channel:

- ✘ may flip bits in packets (**Corruption**)
- ✘ may lose packets (**Packet loss**)
- ✘ may incur arbitrarily long packet delay
- ✓ would not re-order packets

To handle packet loss:

- Sender waits "reasonable" amount of time for ACK.
- Sender retransmits if no ACK is received till *timeout*.





rdt Summary

rdt Version	Scenario	Features Used
1.0	no error	nothing
2.0	**data Bit Error**	*checksum, ACK/NAK*
2.1	data Bit Error **ACK/NAK Bit Error**	checksum, ACK/NAK, *sequence numbers*
2.2	Same as 2.1	NAK free (uses sequence numbers to detect loss packets)

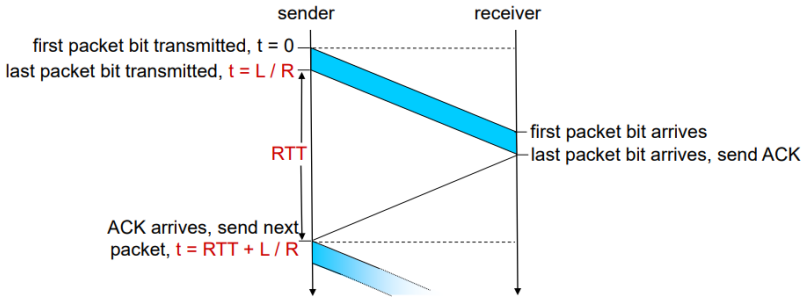
rdt Version	Scenario	Features Used
3.0	data Bit Error ACK/NAK Bit Error **packet loss**	checksum, ACK/NAK, sequence numbers, *timeout/re-transmission*

Performance of rdt 3.0

Example: packet size = `8000 bits`, link rate = `1 Gbps`

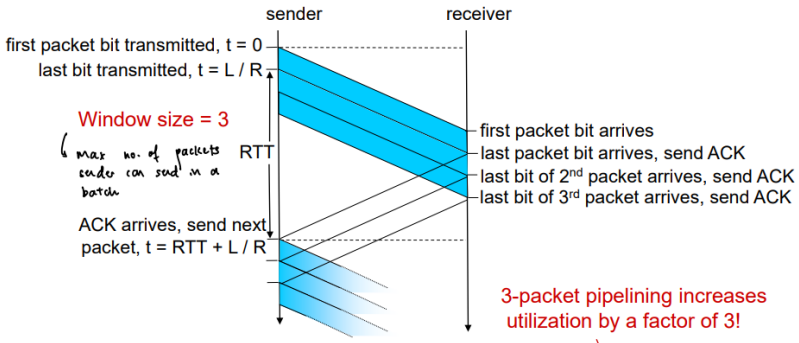
```
d_trans = L/R = 8000 bits / 10^9 bits/sec = 0.008 msec

//If RTT = 30 msec, sender sends 8000 bits every 30.008 msec
throughput = L/(RTT + d_trans) = 8000/30.008 = 267 kbps

//U_sender: utilization is the fraction of time sender is busy sending
U_sender = d_trans/(RTT + d_trans) = 0.008 / (30 + 0.008) = 0.00027
```

- Network protocol limits use of physical resources!



$$U_{sender} = \frac{L / R}{RTT + L/R} = \frac{0.008}{30 + 0.008} = 0.00027$$

Pipelining: Increased Utilization



$$U_{sender} = \frac{3 * L / R}{RTT + L/R} = \frac{0.024}{30 + 0.008} = 0.00081$$

Pipelined Protocols

Pipelining

sender allows multiple, “in-flight”, yet-to-be-acknowledged packets.

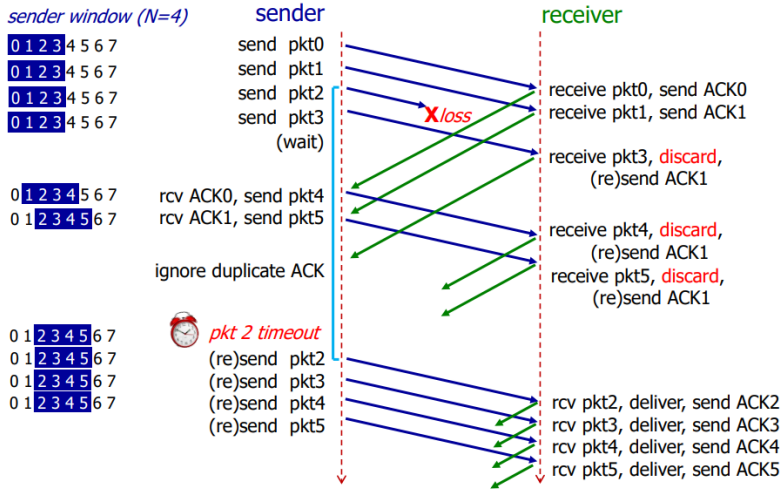
- range of sequence numbers must be increased
- buffering at sender and/or receive

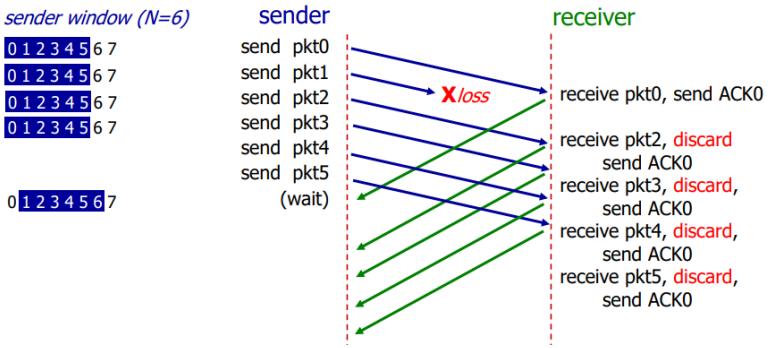
Go-Back-N (GBN)

Think of it as a sliding window that slides forward only when an ACK is received for the leftmost packet in the window.

- **GBN Sender:**
  - can have up to N unACKed packets in pipeline
  - insert k-bits sequence number in packet header
  - use a “sliding window” to keep track of unACKed packets
  - keep a timer for the oldest unACKed packet
  - `timeout(n)`: retransmit packet *n* and all subsequent packets in window
- **GBN Receiver:**

- only ACK packets that arrive IN ORDER
  - simple receiver: need only remember `expectedSeqNum`
- discard out-of-order packets and ACK the last in-order sequence number
  - cumulative ACK: “ACK m” means all packets up to m are received

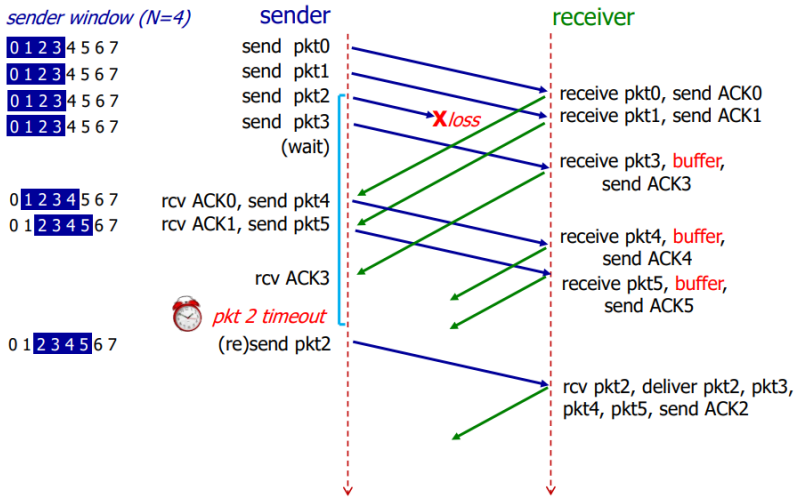




Selective Repeat

- Receiver *individually acknowledges* all correctly received packets

  - Buffers out-of-order packets, as needed, for eventual in-order delivery to upper layer
- Sender maintains timer for **each** unACKed packet
  - When timer expires, retransmit only that unACKed packet
- 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.



packets are buffered and only delivered to the applcation layer IN ORDER, hence only once pkt2 is recevied, pkts 2, 3, 4, 5 will be delivered.

sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase, sendbase+N]

- mark pkt n as received
- if n is smallest unACKed pkt, advance window base to next unACKed seq. #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N, rcvbase-1]

- ACK(n)

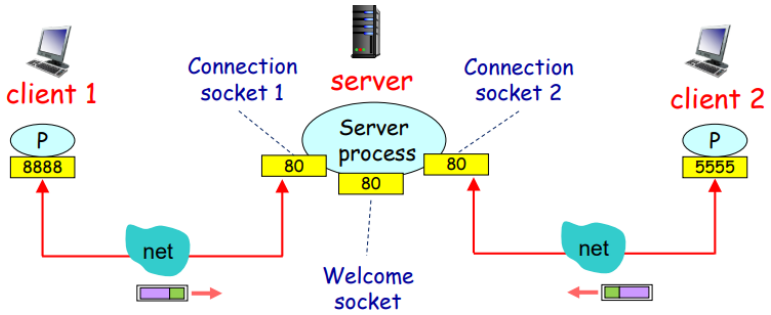
otherwise:

- ignore

Connection-oriented transport: TCP

Connection-oriented De-mux

- A TCP connection (socket) is identified by 4-tuple
  - `srcIPAddr`, `srcPort`, `destIPAddr`, `destPort`
  - Receiver uses all four values to direct a segment to the appropriate socket

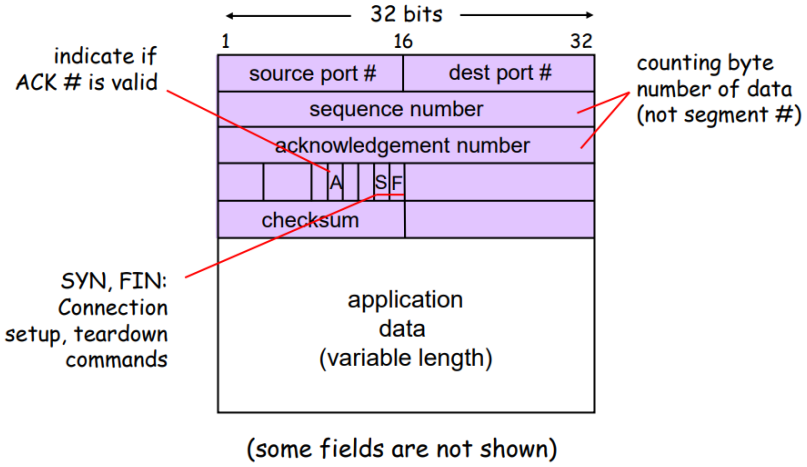


TCP: Buffers and Segments

- TCP send and receive buffers ⇒ 2 buffers created after handshaking at any side
- How much app-layer data a TCP segment can carry?

  - maximum segment size (**MSS**), typically 1460 bytes (+ 20 byte header = 1480 bytes total)
  - app passes data to TCP and TCP forms packets in view of MSS
  - TCP header is **not included** in MSS. (see PYP question 4)

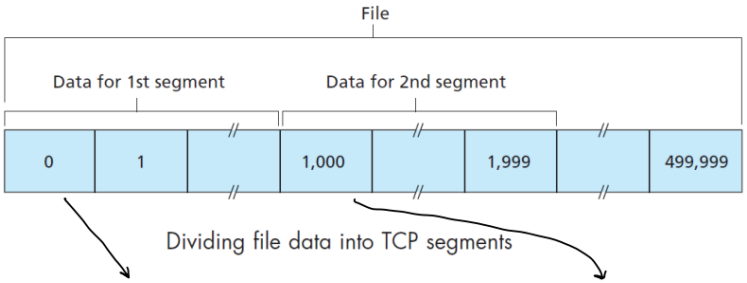
TCP Header



- **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 Number

- **Sequence Number:** Byte number of the first byte of data in a segment, wrt the whole file
- **Random initial sequence number:** To minimise the probability of some segment from a previous connection being mistaken as from the current connection
- Example: send a file of 500,000 bytes; MSS is 1,000 bytes



- Seq. # of 1st TCP segment: 0, 2nd TCP segment: 1,000, 3rd TCP segment: 2,000, 4th TCP segment: 3,000, etc.

TCP ACK Number

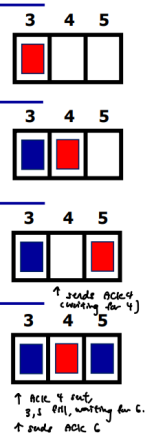
- Sequence number of next byte of data **expected by receiver**

Sequence number of a segment	Amount of data carried	Corresponding ACK number
0	1,000	1,000
1,000	1,000	2,000
2,000	1,000	3,000
3,000	1,000	4,000
...	...	...

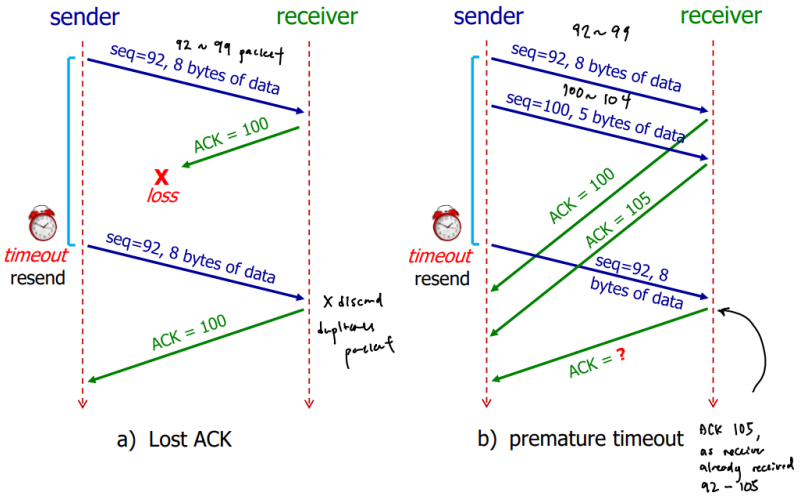
- TCP ACKs up to the first missing byte in the stream (**cumulative ACK**)
  - TCP spec does not say how receiver should handle out-of-order segments, up to implementer. (drop/buffer)
- Initial sequence number is randomly chosen (client/server chooses randomly), may NOT start from 0

TCP ACK Generation

Event at TCP receiver	*TCP receiver action*
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK: wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # (gap detected)	Immediately send *duplicate ACK*, indicating seq. # 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



TCP Timeout / TCP Retransmission



- How does TCP set appropriate timeout value?
  - **too short:** premature timeout and unnecessary retransmission

- **too long**: slow reaction to segment loss ⇒ timeout interval must be longer than RTT, but RTT varies

- TCP computes and keeps updating timeout interval based on estimated RTT

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

  - Typical value of α: 0.125

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

  - Typical value of β: 0.25

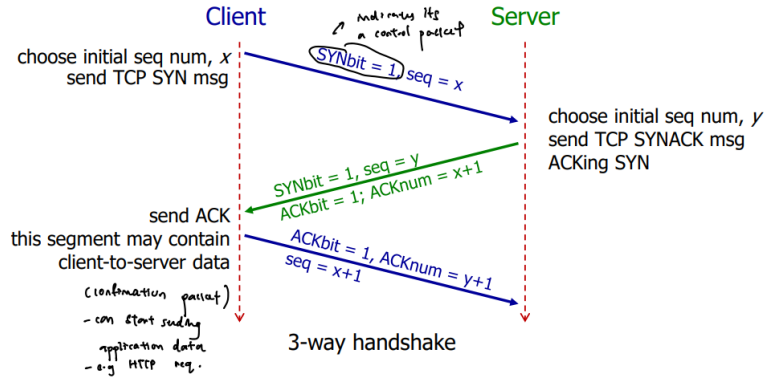
$$TimeoutInterval = EstimatedRTT + 4 * DevRTT$$

  - $DevRTT$ is the "safety margin"

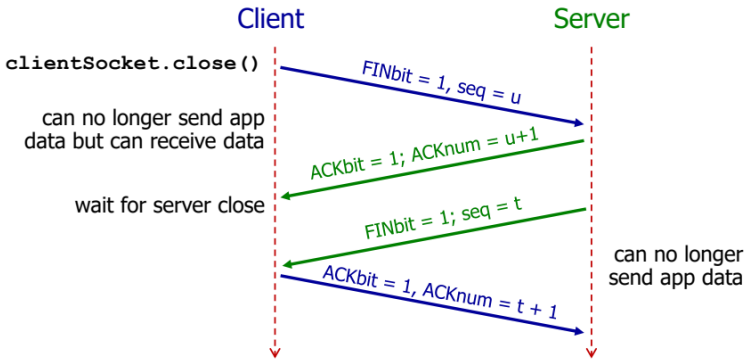
- TCP Fast Retransmission
  - Timeout period is often relatively long ⇒ long delay before resending lost packet
  - Fast retransmission:
    - **Event:** If sender receives *4 ACKs* for the same segment, it supposes that segment is lost
    - **Action:** resend segment (even before timer expires)

Establishing Connection

- Before exchanging app data, TCP sender and receiver "shake hands" ⇒ agree on connection and exchange connection parameters



- Closing Connection
  - Client, server each close their side of connection ⇒ send TCP segment with FIN bit = 1



Questions

1. Consider transferring an enormous file of *L* bytes from Host A to Host B. Assume an MSS of 512 bytes.
   a. Maximum value of L such that TCP sequence numbers are not exhausted. Recall that the TCP sequence number field is 32 bits

$$2^{32}\ bytes$$

Sequence number does not increment by 1 with each segment, instead it increments by the number of bytes of data sent. ⇒ size of MSS is irrelevant, simple just the number of bytes representable, 2^{32}.

b. For the *L* obtained in (a), find how long it takes to transmit the file. Assume a total of 64 bytes of transport, network and data-link header are added to each packet, before resulting packet is sent out over a 155 Mbps link.

$$
\begin{aligned}
Number\_of\_segments &= \frac{2^{32}}{512} \\
Total\_data &= (\frac{2^{32}}{512} \times 64) + 2^{32}\ bytes \\
\therefore Time &= \frac{2^{32}(\frac{64}{512} + 1) \times 8}{155 \times 10^6} \\
&= 249s\ (3sf)
\end{aligned}
$$

Chapter 6 and 7: Network Layer

IP Address

Used to identify a host (or a router)

- A 32-bit integer expressed in either binary or decimal

```
binary: 00000001 00000010 00000011 10000001
decimal: 1.2.3.129
```

How does a host get an IP address?

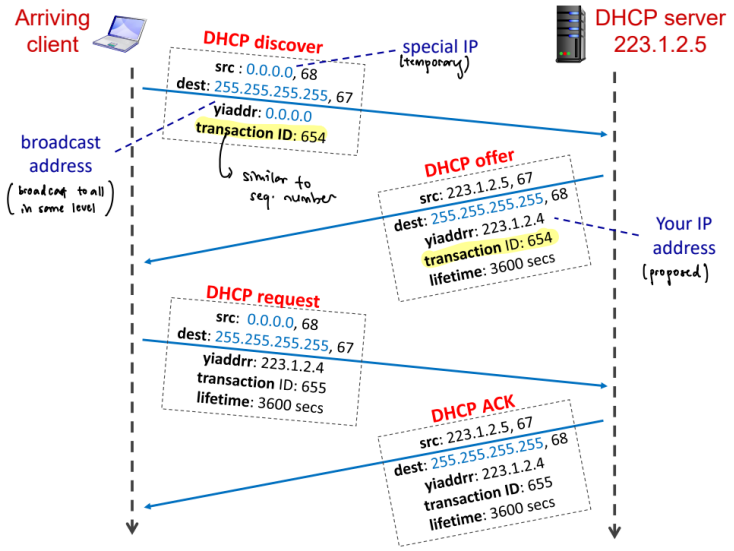
- manually configured by system admind
- auto assigned by DHCP server ⇒ scalable

Dynamic Host Configuration Protocol

- **DHCP** allows host to dynamically obtain its IP address from DHCP server when it joins network
  - IP address is renewable
  - allow reuse of addresses (only hold address while connected)
  - support mobile users who want to join network
- 4 step process
  1. Host broadcasts `DHCP discover` message
     - Sent with 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
     - Source IP address is 0.0.0.0
  2. DHCP server responds with `DHCP offer` message
     - Destination IP is again 255.255.255.255
     - Destination port is **68**
     - Source IP is the server's IP
     - Source port is **67**

     > 💡 In addition to host IP address assignment, DHCP may also provide a host additional network information:
     > - IP address of first-hop router
     > - IP address of local DNS server
     > - Network mask (indicating network prefix versus host ID of an IP address)

  3. Host requests IP address: `DHCP request` message, echoing back configuration parameters
  4. DHCP server sends address `DHCP ACK` message, confirm requested parameters



Some Special IP addresses

Special Addresses	Present Use
0.0.0.0/8	Non-routable meta-address for special use
127.0.0.0/8	Loopback address. A datagram sent to an address within this block loops back inside the host. This is ordinarily implemented using only 127.0.0.1/32
10.0.0.0/8 172.16.0.0/12 192.168.0.0/16	Private addresses, can be used without any coordination with IANA or an Internet registry.
255.255.255.255/32	Broadcast address. All hosts on the same subnet receive a datagram with such a destination address.

IP Address and Network Interface

An IP address is associated with a **network interface**.

- A host usually has one or two network interfaces (e.g. wired Ethernet and WiFi)
- A router typically has multiple interfaces

IP Address and Subnet

- An IP address logically comprises two parts:


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

- **Subnet** is a network formed by a group of “directly” interconnected hsots.
  - Hosts in the same subnet have the same network prefix of IP address
  - Hosts in the same subnet can physically reach each other without intervening router
  - They connect to the “outside” via a router

IP Address: CIDR

- The Internet’s IP address assignment strategy is known as *Classless Inter-domain Routing (CIDR)*
  - Subnet prefix of IP address is of arbitrary length
  - Address format: `a.b.c.d/x`, where `x` is the number of bits in subnet prefix of IP address

```
1100 1000    0001 0111    0001 0000    0010 1010
|----------subnet prefix---------|<---hostID--->
```

  - this subnet contains 2^9 IP addresses
  - subnet prefix: 200.23.16.42/23

Subnet Mask

Used to determine which subnet an IP address belongs to

❖ Example: for IP address 200.23.16.42/23:

	← subnet prefix →			← host ID →
IP address in binary	11001000	00010111	00010000	00101010
Subnet mask	11111111	11111111	11111110	00000000
Subnet mask in decimal	255.255.254.0			

IP Address Allocation

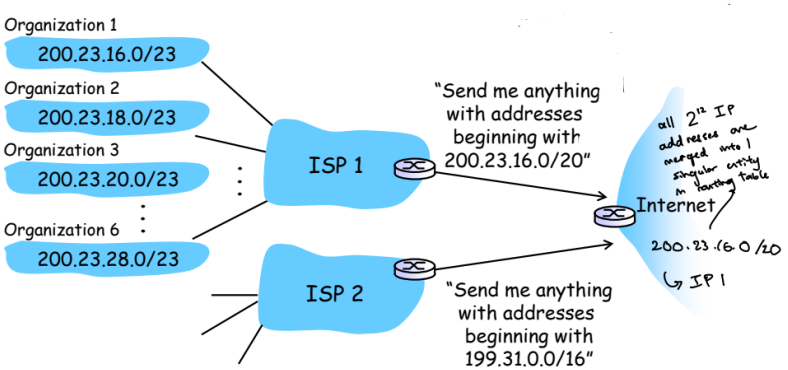
An organization obtains a block of IP addresses by buying from registry or rent from ISP's address space.

	Binary Address	Decimal Address
ISP's block	11001000 00010111 0001 000 0 00000000	200.23.16.0/20
Organization 1	11001000 00010111 0001 000 0 00000000	200.23.16.0/23
Organization 2	11001000 00010111 0001 001 0 00000000	200.23.18.0/23
Organization 3	11001000 00010111 0001 010 0 00000000	200.23.20.0/23
...	...	...
Organization 6	11001000 00010111 0001 101 0 00000000	200.23.28.0/23

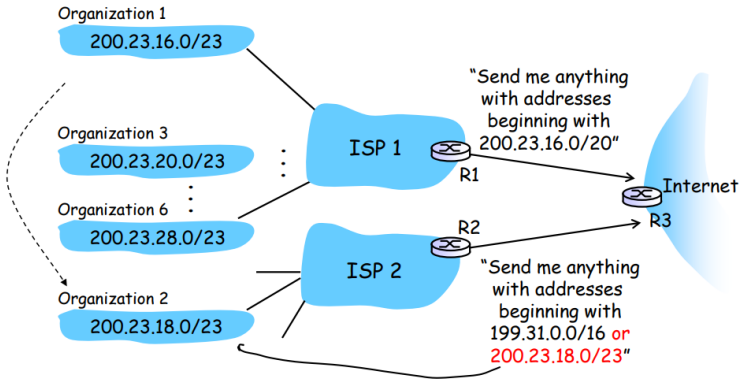
use 3 more bits to differentiate 6 organizations

Hierarchical Addressing

We can have an efficient way of routing by structuring the IP addresses as hierarchy. For example, a ISP may have address block 200.23.16.0/20. It can further split it up to support multiple organizations.



If Organization 2 now switches to ISP 2, but does not want to renumber all its routers and hosts:



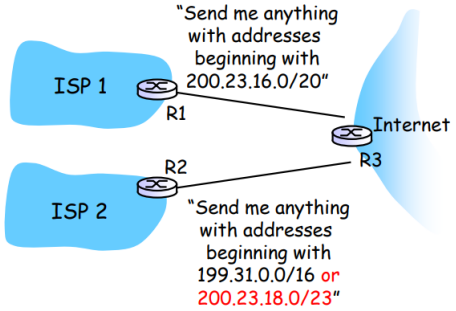
Longest Prefix Match

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.

- Question: which router to deliver to,
  - if a packet has destination IP 200.23.20.2?
  - if a packet has destination IP 200.23.19.3?

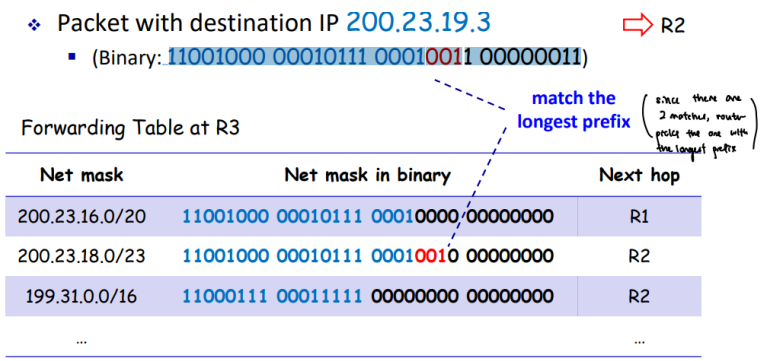
Forwarding Table at R3

Net mask	Next hop
200.23.16.0/20	R1
200.23.18.0/23	R2
199.31.0.0/16	R2
...	...



- 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 organization, else it will be forwarded to the other.
- Packet with destination IP `200.23.20.1` ⇒ **R1**
  - Binary: 11001000 00010111 00010100 00000010

- Packet with destination IP `200.23.19.3` ⇒ **R2**
  - Binary: 11001000 00010111 0001***001***1 00000011



Routing Algorithms

The internet is a hierarchy of Autonomous Systems (AS), routing is similarly done in a hierarchy.

- **Intra-AS Routing:** Finding a good path between 2 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 (constant based on congestion, inverse of bandwidth etc.)
- **Inter-AS Routing:** Handles interfaces between ASes. (NOT COVERED)
  - Admin wants control over who and what is routed.
  - Policy dominates over performance.

Link State algorithms — Centralised Routing algorithm

- Compute least-cost path from source to destination based on complete, global knowledge about the network (e.g Djikstra's algorithm)
- All routers have the complete knowledge of network topology and link cost.
  - Routers periodically broadcast link costs to each other.

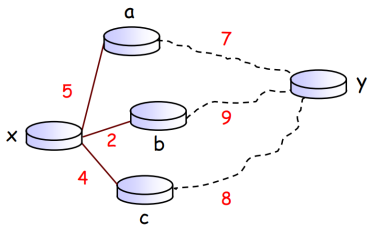
Distance vector algorithm — Decentralised Routing algorithm

- Each node begins with the knowledge of its direct links (physically connected neighbours).
- Nodes exchange "local views" with neighbour and update own "local views" (based on neighbour's view).
- Iteratively exchange information with neighbours and calculate least cost path to destination.
  1. Swap local view with direct neighbours
  2. Update own's local view
  3. Repeat 1-2 till no more change to local view

$$c(x,y) : cost\ link\ between\ routers\ x\ and\ y$$
$$d_x(y) : cost\ of\ the\ least-cost\ path\ from\ x\ to\ y$$

Distance-vector algorithm: Bellman-Ford Equation

$$\begin{aligned} d_x(y) &= min_v\{c(x,v) + d_v(y)\} \\ d_x(y) &= min_v\{c(x,a) + d_a(y), \\ &\quad\ c(x,b) + d_b(y), \\ &\quad\ c(x,c) + d_c(y)\} \\ &= min\{12, 11, 12\} = 11 \end{aligned}$$



1. First initialize the shortest path to all direct neighbours with the cost of the edge, and to all other routers as ∞.
2. We then let all our neighbours know of all the shortest path we are currently aware of.
3. We then wait for our neighbours to let us know their shortest path 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.

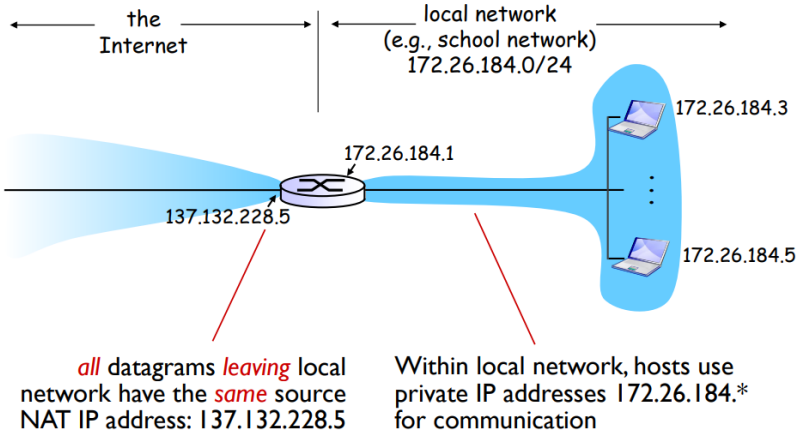
b. If it's a change to the link cost to our neighbour, similarly check if it results in shorter paths, and if so, broadcast all changed shortest paths.

4. The final shortest path or distance vectors will form our forwarding table.

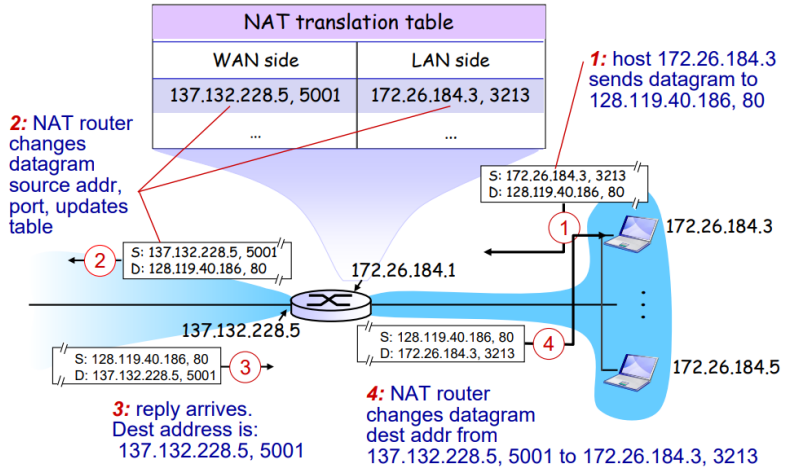
Routing Information Protocol (RIP)

- implements the DV algorithm.
- **Hop Count:** as the cost metric (i.e. insensitive to network congestion):
  - regardless of how congested netowrk is, always use least hop count
- **Periodic Exchange**: Exchange routing table every 30 seconds over UDP port 520
- **Self-repair**: if no update from a neighbour router for 3 minutes, assume neighbour as failed

Network Address Translation (NAT)



We cannot allocate unlimted contiguous address ranges, and it is neither scalable nor practical to expect (all home owners to know how to manage IP addresses) The following will be done via a NAT-enabled router:



NAT: Implementation

NAT Routers must:

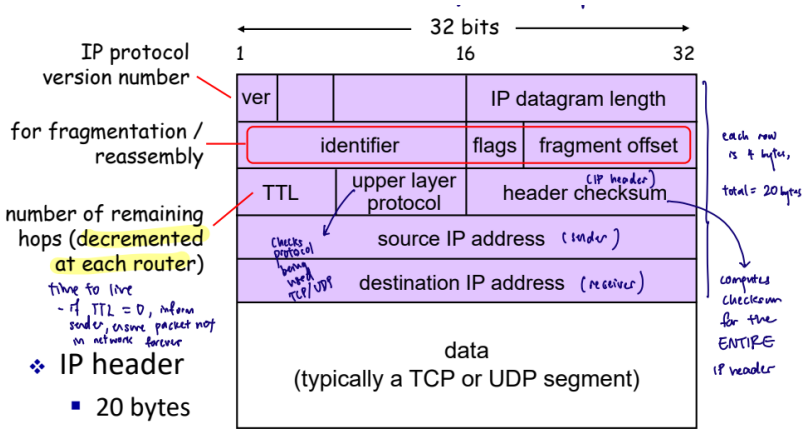
- **Replace** (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
- **Remember** (in NAT translation table) the mapping from (source IP address, port #) to (NAT IP address, new port #)
- **Replace** (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT translation table

NAT Benefits

- No need to rent a range of public IP addresses from ISP: just one public IP for the NAT router.
- All hosts use private IP addresses. Can change addresses of hosts in local network without notifying the 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 (a security plus).

IPv4 & Addressing

IPv4 Datagram Format

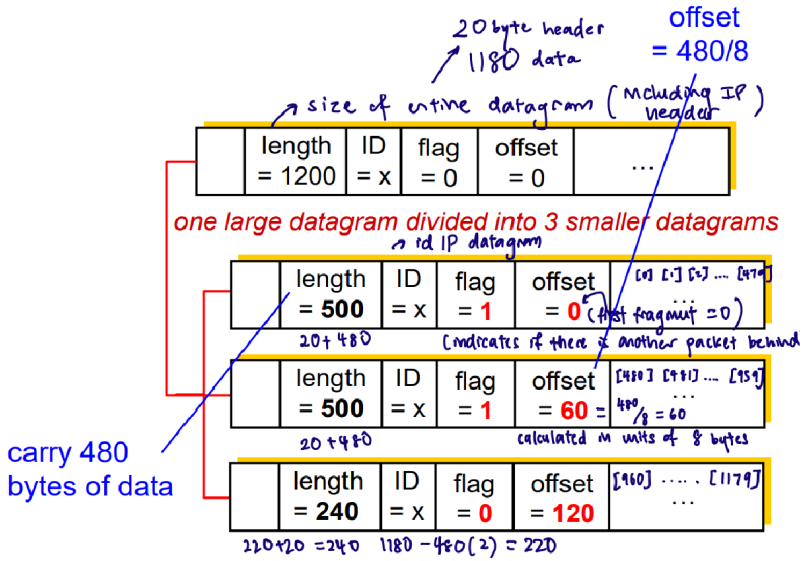


1. **Datagram Length:** length of the IP datagram, including 20 bytes from header
2. **Identifier, flags, fragment offset:** To support fragmentation
3. **Time to live:** To prevent datagrams from circulating forever
4. **Upper layer protocol:** Only used at final destination to determine if it is UDP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

IP Fragmentation and Reassembly

Different links may have different **MTU (Max Transfer Unit)** – the maximum amount of data a link-level frame can carry. 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.

Example: 20 byte IP header, 1,200 byte IP datagram, MTU = 500 byte



1. The router will need to decide if the datagram needs to be fragmented.
2. **Identifier:** All 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 $\frac{480}{8} = 60$
5. **Reassembly:** Will only be done by the destination host. Routers in between will not reassemble.

Internet Control Message Protocol (ICMP)

Used by hosts and routers to communicate network-level information ⇒ architecturally it lies above the IP layer, as ICMP messages are carried inside IP datagrams.

ICMP will be specified as the upper layer-protocol, and when the rotuer sees that ICMP is the upper-layer protocol, it will demultiplex into ICMP.

- 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:
    $Type + Code + Checksum + others$

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

1. `ping`: the command `ping` sees if a remote host will respond to us — do we have a connection?
2. `traceroute`: sends a series of small packets across a network, and attempts to display the route (or path) that the messages would take to get to a remote host

Questions

1. Which of the following fields of an IP datagram header may be changed by an NAT-enabled router?
   i. TTL ⇒ when router receives packet, TTL -= 1 before passing to next router
   ii. Checksum ⇒ if TTL is changed, checksum is computed by the entire IP header, hence checksum is changed as well
   iii. Destination IP address
   iv. Source IP address

   ans: **(i), (ii), (iii), (iv)**

Chapter 8 Link Layer

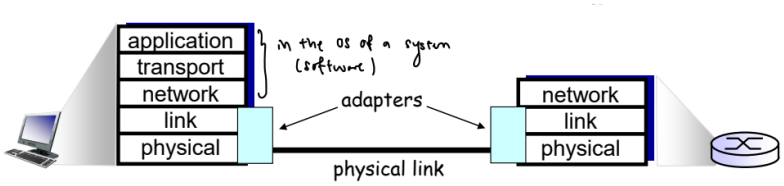
The link layer takes care of how packets are sent across **individual links**. We call all devices that runs a link layer protocol a **node**, and all communication channels connecting (physically) adjacent nodes as **links**. Different links may run different protocols. Over a given link, a transmitting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

Motivation

- aim: send data between N nodes via cable
- solution: interconnect the N nodes via a broadcast link
  - each link needs to be addressed ⇒ Framing
    - **Framing** — encapsulation of a network-layer datagram within a link-layer frame
  - need to define a protocol ⇒ Link access control
    - **Link access control** — coordination of which nodes can send frames at a certain point of time when multiple nodes **share** a singular link
  - need to handle errors ⇒ Detection & Reliability
    - **Reliabile delivery** — Seldom used on low bit-error link (e.g., fiber) but often used on error-prone links (e.g., wireless link).
    - **Error detection** — Errors are usually caused by signal attenuation or noise. Receiver detects presence of errors, may signal sender for retransmission or simply drops frame.
    - **Error correction** — Receiver identifies and corrects bit error(s) without resorting to retransmission.

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 hardward, 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. ⇒ in the below example, the overhead is 50%

$$EDC = D$$
$$|1011|1011|$$

Popular error detection schemes:

1. Checksum (used in TCP/UDP/IP)
2. Parity checking
3. CRC (commonly used in link layer)

Checksum (review)

- treat segment contents as sequence of 16-bit integers
- *checksum*: 1's complement of the sum of segment contents

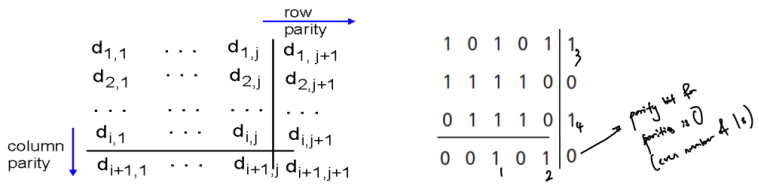
Parity Checking

- **Single Bit:** 1 parity bit for the data

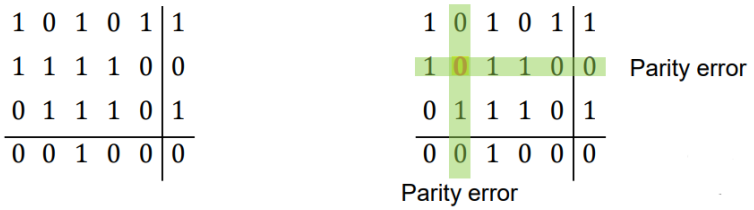
  — **Even Parity Scheme:** We choose the bit to make the total number of 1s even

  — **Odd Parity Scheme:** Same but we make the number odd instead of even.

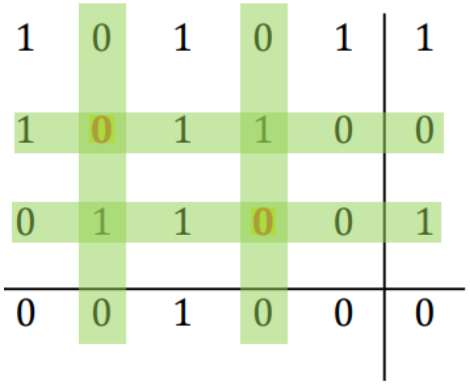
  - Can detect single bit errors in data (if even parity scheme) ⇒ this means that only 50% of errors can be detected
    - can detect odd number of single bit errors
    - CANNOT detect even number of single bit error
  - Works exceptionality well (probabilty of multiple bit errors is low)
  - However, errors are often clustered together in "bursts"
    - Probability of undetected errors in a frame can approach 50%
- **Two-Dimensional Parity:** We divide the data into i rows and j columns, and repeat the above but have parity for each column and each row.
  - The resulting i + j + 1 parity bits comprise the link-layer frame's error-detection bits



- Can detect and correct single bit errors in data



- Can detect (**CANNOT CORRECT**) two-bit error in data ⇒ uncertain which bits it may be



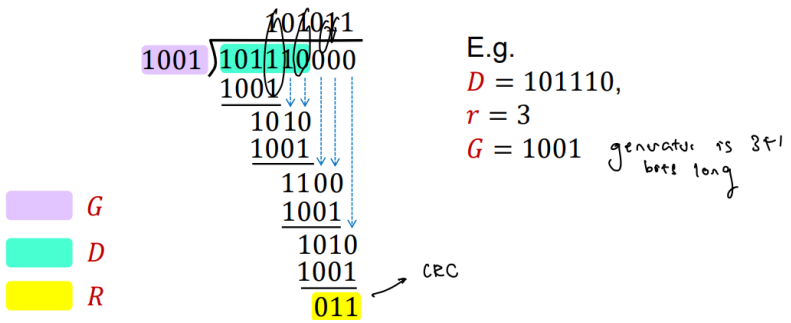
- **Multi-Dimensional Parity:** This can go for n dimensions. An n-dimensional parity scheme is only guaranteed to correct up to $\frac{n}{2}$ errors

Cyclic Redundancy Check

We can think of this as long divison but with divison being replaced by a bitwise XOR operation.

- 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

For performing division, we append r 0's to D. Because of the properties of modulo 2 arithmetic, the remainder directly gives us R. If non-zero remainder, error is detected!



- Easy to implement on hardware
- Powerful error-detection coding that is widely used in practice (e.g. Ethernet, Wifi)
  - Can detect 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 probabilty $1-0.5^r$
  - CRC is also known as polynomial code
    - a k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging form x^{k-1}
    - 110001 $\Rightarrow 1x^5+1x^4+0x^3+0x^2+1x^0=x^5+x^4+1$

Multiple Access Links and Protocols

Types of Network Links

1. **Point-to-point link:** Sender and receiver connnected 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 collision.

An idea multiple access protocol:

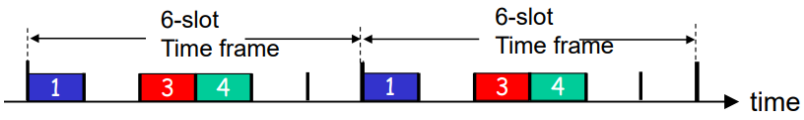
- **Given:** Broadcast channel of R bps
- **Desired Properties:**
  1. Collision free
  2. Efficient: when only one node wants to transmit, it can send at rate R
  3. Fairness: when M nodes want to transmit, each can send at rate $\frac{R}{M}$
  4. Fully decentralised $\Rightarrow$ no special node to coordinate transmissions

  Mandatory Requirement: coordination about channel sharing must use channle itself $\rightarrow$ no out-of-band channel signalling

Channel Partitioning Protocols

TDMA (time division multiple access)

- access to channel in "rounds"
- each node gets fixed length time slots in each round
  - length of time slot == data frame transmission time
- e.g 6 nodes sharing a link $\Rightarrow$ 1, 3, 4 have data to send, slots 2, 5, 6 idle

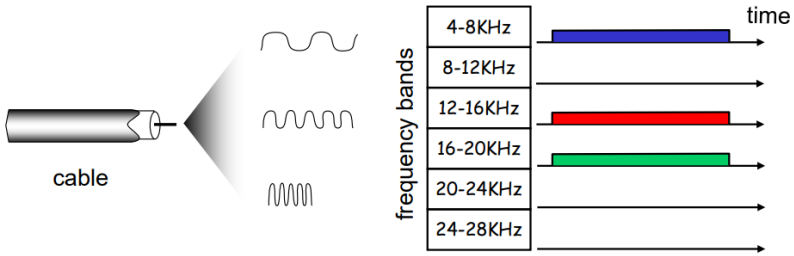


Desired Property	Remarks
Collision Free	Yes
Efficiency	Inefficient, unused slots are idle Maximum throughput for a node is $\frac{R}{N}$
Faireness	Perfectly fair

Desired Property	Remarks
Decentralised	Yes ⇒ only clocks need to be synchronise, no singular node coordinating instructions

FDMA (frequency division multiple access)

- Channel spectrum is divided into frequency bands.
- Each node is assigned a fixed frequency band.
- Unused transmission time in frequency bands go idle.
- Example: 6 nodes, 1, 3, 4 have frames, frequency bands 2, 5, 6 are idle.



Desired Property	Remarks
Collision Free	Yes
Efficiency	Inefficient, unused slots are idle Maximum throughput for a node is $\frac{R}{N}$
Faireness	Perfectly fair
Decentralised	Yes

Multiple Access Protocols

Multiple access protocols can be categorized into three classes

- Channel partitioning
  - divide channel into fixed, smaller “pieces” (e.g time/freq slots)
  - allocate piece to node for exclusive use
- “Taking turns” ⇒ nodes take turn to transmit
- Random Access

  - channel is not divided, collisions are possible
  - “recover” from collision

“Taking turns” Protocol: Polling

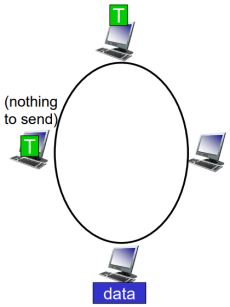
- requires one of the nodes to be designated as master node
- master node polls each of the nodes in a round-robin fasion
  - master node informs 1, it can transmit up to a maximum number of frames
  - after node 1 transmits some frames, the master node tells node 2 it can transmit up to a maximum number of frames and cycle repeats…

Desired Property	Remarks
Collision Free	Yes ⇒ only talk when required
Efficiency	Higher efficiency Overhead of poling ⇒ tends to R (approx.)
Faireness	Perfectly fair ⇒ as long as master node is fair
Decentralised	No ⇒ master node is **single point of failure**

“Taking turns” Protocol: Token Passing

- special frame: token is passed from one node to next, sequentially
- when a node receives a token
  - hold onto the token only if some frames to transmit
    - it sends up to a maximum number of frames and then forwards the token to the next node
  - otherwise, forward the token to the next node

Desired Property	Remarks
Collision Free	Yes ⇒ only transfer when holding token
Efficiency	Higher efficiency Overhead of poling ⇒ tends to R (approx.)
Faireness	Perfectly fair
Decentralised	Yes
Downside	Token loss can be disruptive - data frame loss - system bugs Node failure can break the ring

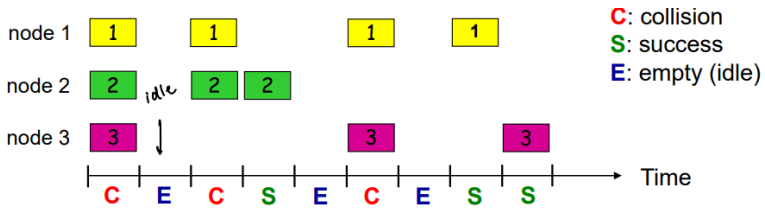


Random Access Protocols

Not a priority to have coordination among nodes, and able to transmit at full channel data rate R if a node has data to send. These protocols specify how to detect and recover from collisions.

Slotted ALOHA

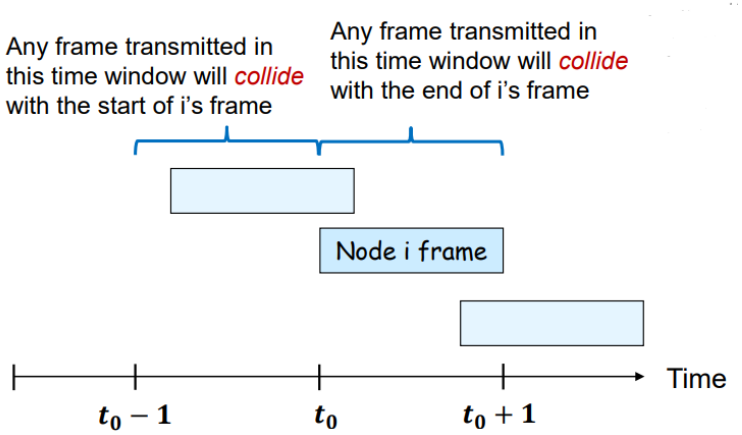
- Design
  - all frames L bits (equal size)
  - time divided into slots of equal length = time taken to transmit 1 frame = $\frac{L}{R}$
  - nodes start to transmit only at the beginning of the slot ⇒ time synchronised
- When sending a fresh frame
  - wait until beginning of the next slot, transmit entire frame in slot
  - if **no collision:** data transmission is a success
  - if **collision:** data transmission is a failure ⇒ retransmit the frame in each subsequent slot with probability p until success



Desired Property	Remarks
Collision Free	No
Efficiency	Yes ⇒ when only one node is active (throughput is R) No ⇒ when there are many active nodes, maximum efficiency is 37% ⇒ slots wasted due to both collision and because of being empty
Faireness	Perfectly fair
Decentralised	Yes

Pure (unslotted) ALOHA

- no time slots, no synchronization (simpler than slotted ALOHA)
- When sending a fresh frame
  - transmits the entire frame immediately
  - if **no collision**: data transmission is a success
  - if **collision**: data transmission is a failure
    - Wait for 1 transmission time
    - retransmit the frame with probabilty p until success
- chance of collision increases: frame sent at t_0 collides with other frames sent in $(t_0 - 1, t_0 + 1)$ ⇒ probabilty of collision doubles , efficiency ↓



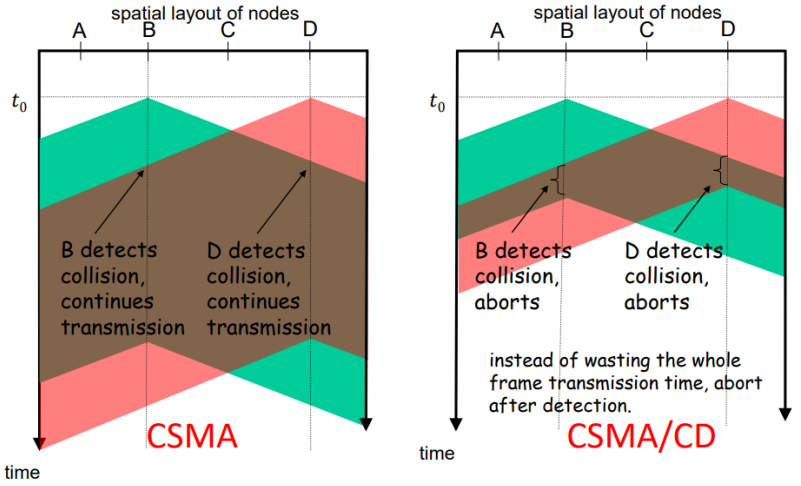
Desired Property	Remarks
Collision Free	No
Efficiency	Yes ⇒ when only one node is active (throughput is R) No ⇒ when there are many active nodes, maximum efficiency is 18% ⇒ slots wasted due to both collision and because of being empty
Faireness	Perfectly fair
Decentralised	Yes

CSMA (Carrier Sense Multiple Access)

- Design flaw in ALOHA ⇒ a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel
- a node pays no attention to whether anotehr node happens to be transmitting when it begins to transmit
- **listen before transmit**
  - if channel **sense idle:** transmit entire frame
  - if channel **sensed busy:** defer transmission
- **collisions can still occur:** propagation delay means 2 nodes may not hear each other's transmission immediately

CSMA/CD (Collision Detection)

- Design flaw in both ALOHA and CSMA ⇒ node does not stop transmitting when collision is detected
  - if channel **sensed idle:** transmit entire frame
  - if channel **sensed busy:** defer transmission
  - if **collision detected:** abort transmission — retransmit after a random delay



CSMA/CD Backoff Algorithm

- **motivation:** ALOHA

  collision: data transmission is a failure, wait for 1 frame transmission time retransmit the frame with probabilty p until success

  drawback: probabilty of collision in all subsequent time slots is the same
- **goal:** adapt retransmission attempts to estimate current load (↑ backoff interval if there is more collisions, as more collisions == heavy load)
- **Binary Exponential backoff**

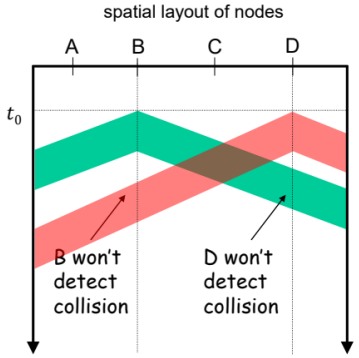
Collision	Explaination	p - probabilty of retransmission
1^{st}	- choose Kat random from $\{0, 1\}$ - wait K time units before retransmission	$\frac{1}{2}$
2^{nd}	- choose K at random from $\{0, 1, 2, 2^2 - 1\}$ - wait K time units before retransmission	$\frac{1}{4}$
...	...	...
m^{th}	- choose K at random from $\{0, 1, 2, ..., 2^m - 1\}$	$\frac{1}{2^m}$

- if frame size is too small, collisions

happen but may not be detected by sending nodes ⇒ no retransmission!

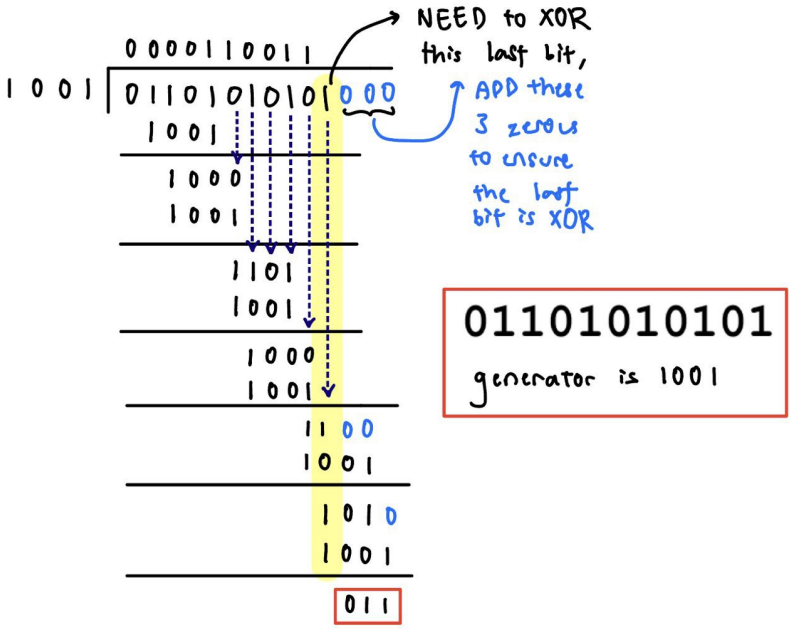
- Ethernet requires a minum frame size of $64\ bytes$.

Desired Property	Remarks
Collision Free	No
Efficiency	Yes
Fairness	Yes
Decentralised	Yes



Questions

1. CRC with the first bit as $'0'$.



01101010101

generator is 1001

2. There are many nodes in a shared medium network and most nodes are likely to transmit frequently. Which of the following multiple access protocol(s) is (are) suitable?
   (1) TDMA; (2) CSMA; (3) Token passing
   - TDMA and token passing are suitable because there is sufficient work to do to utilize the
     "fixed" resources allocated. CSMA is not because many nodes competing for the shared channel can result in lots of collision. Utilization will be low.
3. Exponential backoff: k^{th} collision: $2^k - 1$. If node A tries to send a 128B packet at time t, for the 3^{rd} time unsuccessfully, the next time it will send the packet is $t + 15$ time units.
   - FALSE. Time at which retransmission begins is **RANDOM** thus it is not fixed, may not be $t + 15$ time units. (this will be the maximum time it will take to begin its retransmission).
   - k^{th}collision: $2^k - 1$ gives the **range** of unit time to wait (max time in range)

Chapter 9 Ethernet

Switched Local Area Network

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.

⇒ if yes, adapter extracts the enclosed datagram and passes it to the protocol stack.

⇒ if no, adapter discards the frame without interuptting the host.

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

Link Layer

Link layer sends datagram between adjacent nodes (hots or routers) over a single link ⇒ responsibility of transferring datagram from one node to physically adjacent node over a link

- IP datagrams are encapsulated in link-layer frames for transmission
- Different link-layer protocols may be used on different links.
  - each protocol may provide a different set of services

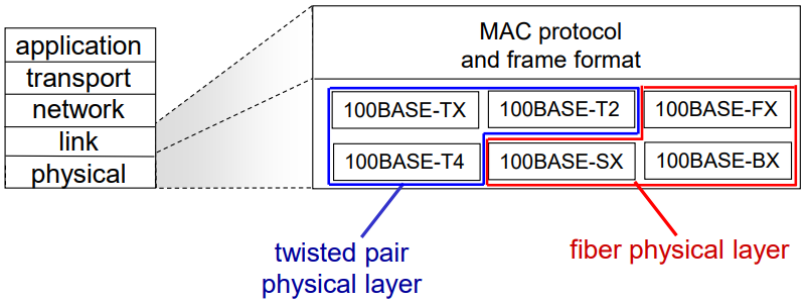
Local Area Network (LAN)

LAN is a computer network that interconnects computers within a geographical area such as office building / university campus

1. IBM Token Ring: IEEE 802.5 standard
2. Ethernet: IEEE 802.3 standard
3. Wi-Fi: IEEE 802.11 standard

Ethernet Standards

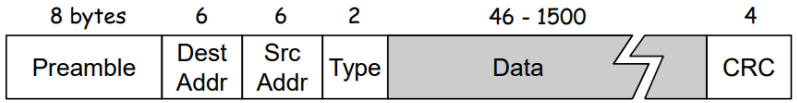
- Different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps, 100 Gbps
- Different physical layer: cable, fiber optics
- **MAC protocol** and **frame format** remain unchained



Ethernet Topology

- **Bus Topology:** All nodes are connected and can collide with each other
  - If backbone cable damaged, entire network fails
  - Difficult to troubleshoot problems
  - Very slow and not ideal for larger networks
- **Star Topology Hub:** Hub in the centre, nodes connected to that hub. Physical layer device that acts on individual bits rather than frames. Hub recreates bit, boost its energy and transmit to all other interfaces
  - Cheap and easy to maintain (due to modular design → just replace the hub)
  - Very slow and not ideal for larger networks → due to collisions
- **Star Topology Switch:** A switch is a layer-2 device in the centre and all nodes connected to this switch. They do not collide with one another.
  - Works acts on frames rather than individual bits. No collisions, a bona-fide store-and-forward packet switch.

Ethernet Frame



- **Preamble**: Formed by 7 bytes of 10101010 (AA_{hex}) → Acts like a "wake-up" and 1 byte of 10101011 (AB_{hex}).
  - Use to synchronise receiver and sender clock rates, since the alternating bits form a square wave. Basically lets the receiver know how long is 1 bit.
- **Source & 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, corrupted frames will be dropped

Ethernet Data Delivery Service

- **Unreliable**: receiving NIC doesn't send ACK or NAK to sending NIC
  - data in dropped frames will be recovered only if initial sender uses higher layer rdt (e.g. TCP); otherwise dropped data is lost.
- **Connectionless:** No handhsaking between sender and receiver
- **Multiple Access Protocol:** CSMA/CD with exponential backoff

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 only goes up to the network layer, switches only have 2 layers, i.e. up to the 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 2 channels, i.e. fully **duplex** and frames can be sent 2-way simultaneously. Lastly, the switch **buffers** frames if they are currently forwarding another frame to the outgoing link. (dedicated buffers for each interface)

> 💡 Since switches have no MAC addresses, the connected devices are all technically a **single hop away**.

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.

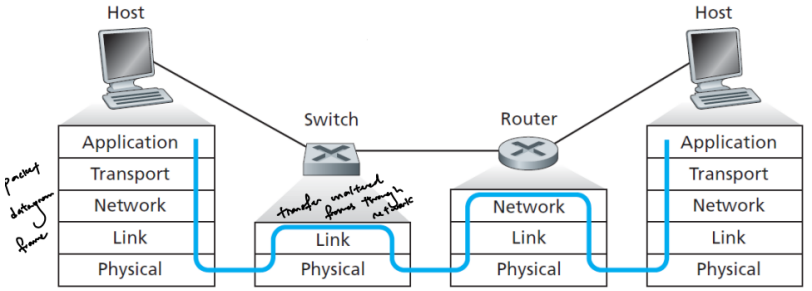
- **Broadcast:** If the destination host is not found in the switch forwarding table, the switch will broadcast the frame to all outgoing links.

Switch: frame filtering/forwarding

When frame received at switch:

1. Record incoming link, MAC address of sending host
2. Index switch table using MAC destination address
3. if entry found for destination
   a. if destination on segment from which frame arrived
      i. drop frame
   b. else forward frame on interface indicated by entry
4. else flood
   a. forward on all interfaces except arriving interface

Routers	Switches
Check IP address	Check MAC address
Store-and-forward	Store-and-forward
Compute routes to destination	Forward frame to outgoing link or broadcast

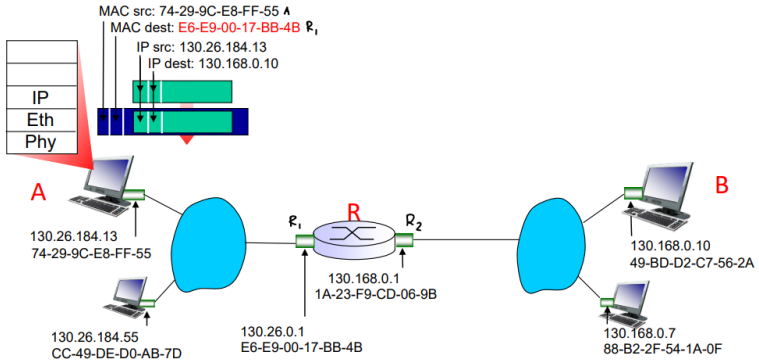


Address Resolution Protocol (ARP)

All nodes have an ARP table containing mappings of IP address and MAC addresses of other **neighbouring nodes** in the same subnet

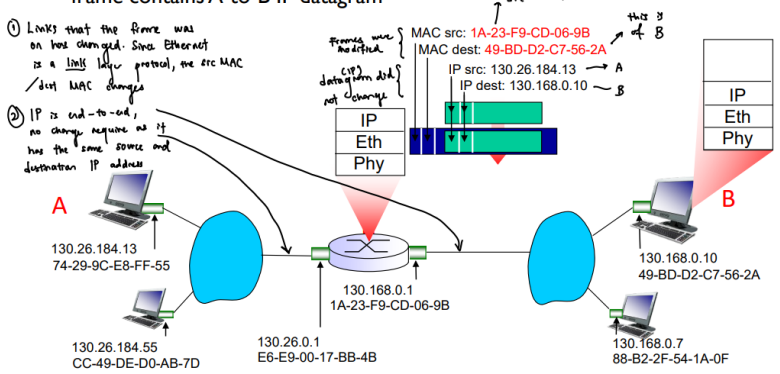
$$< IP\ address;\ MAC\ address;\ TTL >$$

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.
    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 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.



Sending Frame to Another Subnet

- *R* forwards datagram with IP source *A*, destination *B*
- *R* creates link-layer frame with *B*'s MAC address as destination address, frame contains A-to-B IP datagram



IP Address - logical grouping of networks	Mac Address
32 bits in length	48 bits in length
network layer address used to move datagrams from source to dest — end to end	link-layer address used to move frames over every single link
Dynamically assigned; hierarchical (to facilitate routing) — DHCP	Permanent, to identify the hardware (adapter)
"postal address"	"NRIC number"

Questions

1. Enumerate all the steps the host and switch take to move the packet from $\boldsymbol{A}$ to $\boldsymbol{B}$. assuming that ARP table in the sending host is empty, but all other tables are up to date.
    - $\boldsymbol{A}$ broadcasts an ARP query packet, with the destination MAC address as FF-FF-FF-FF-FF-FF.
    - Switch $\boldsymbol{S}$ forwards this ARP query packet to both $\boldsymbol{B}$ and Router $\boldsymbol{R}$ since destination MAC address is a broadcast address.

- R will ignore this ARP query packet but B will reply to A. Switch S forwards the reply frame towards A (interface to A is found in switch table).
- [Now that the ARP table of A is filled for B]:
  - A creates a frame with destination MAC address CC-49-DE-D0-AB-7D (B's address is found in ARP table)
  - This frame travels to switch S and is forwarded towards B (interface to B is found in switch table).

2. Suppose that the following events happened in sequence,
   - i. B sends a frame to D ⇒ broadcast to all links other than 4
   - ii. B replies with a frame to D ⇒ send directly to 4 (switch knows B is 4), switch learns D
   - iii. D sends a frame to A ⇒ send to all other than 3 from which is received - broadcast, possibility of 4 being another switch, hence need to send to 4 also

   The switch table is initially empty. Show the state of the switch table after each event above.

Event	Switch table after event	Link(s) a frame is forwarded to
B sends a frame to D	$(B, 4)$	$1, 2, 3$
B replies with a frame to D	$(B, 4), (D, 3)$	4
D sends a frame to A	$(B, 4), (D, 3)$	$1, 2, 4$

Chapter 10 Network Security

What is network security?

Properties of Network Security?

- **Confidentiality:** Only sender should be able to read and understand the message ⇒ *Substitution Cipher, Symmetric and Public Key*
- **Integrity:** Assets can only be modified by authorised parties ⇒ *Cryptographic Hash*
- **Availability:** Services must be available to authorised parties who need it ⇒ *Firewall*

- **Authenticity:** Ability for sender and receiver to confirm the identity of each other ⇒ *Signature*
- **Non-repudiation:** Sender cannot convincingly deny having sent something

Potential Attacks

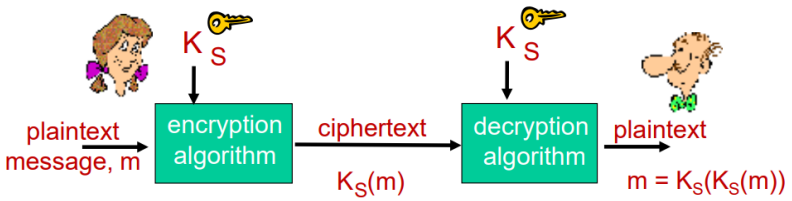
- Eavesdrop
- Insert/delete messages
- Impersonation or spoofing
- Hijacking connection
- Denial of Service

Principles of Cryptography

- **Ciphertext:** 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
- **Plaintext:** original message
- **Known Plaintext Attack:** When you have the ciphertext and the corresponding plaintext
- C**hosen Plaintext Attack:** Presumes that the attacker can obtain ciphertexts from arbitrary plaintext

Symmetric Key Cryptography

When both Alice and Bob share the same key K_s, challenge is communicating that shared key securely.



- Drawback: requires sender, receiver to already know shared secret key
  - How to agree on key in the first place (if they have never met)?

- **Substituition Cipher:** Replace one character for another. The encryption key is a one-to-one mapping of the characters used (**Monoalphabetic cipher)**
  - $26!(\ 10^{26})$ mappings possible are not as much as they seem
  - Can break it with *Statistical Analysis* → e(13%) and t(9%) are the most frequent letters
- **Caeser's Cipher:** Left or right rotation of the alphabets or characters used. Encryption key is the shift number

e.g., right shift by 3:

```
abcdefghijklmnopqrstuvwxyz
defghijklmnopqrstuvwxyzabc

plaintext: the quick brown fox
ciphertext: wkh txlfn eurzq ira
```

- **Polyalphabetic encryption:**
  - Each letter only has one mapping in Monoalphabetic Ciphers

1. Choose n substituition ciphers: $C_1, C_2, ..., C_n$
2. Choose a random cyclic pattern of numbers 1 to n: C_1, C_3, C_4, C_3, C_2 when n = 4
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.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
C_1	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
C_2	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
C_3	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i

Cycling Pattern: $C_1C_3C_3C_2$

e.g.:
```
plaintext: bob, i love you. alice
ciphertext: exk, o oxek bxd. durih
```

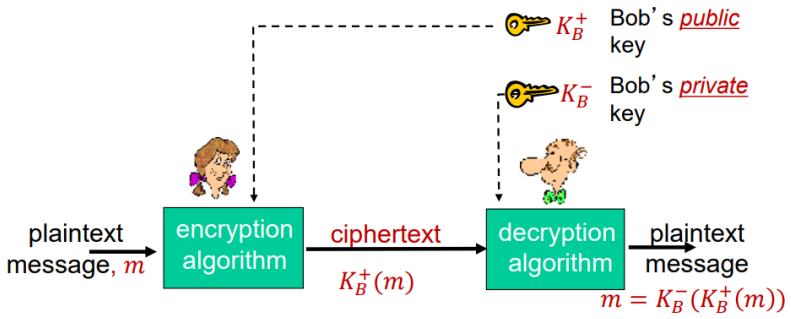
- **Block Ciphers**

	DES	AES (Nov 2001(
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
Solution process	Secret	Secret but open public comments and criticisms invited
Source	IBM enhanced by NSA	Independent Dutch Cryptographers
Broken?	56-key-encrypted phrase decrypted (brute force) in less than a day	Machine capable of Brute force decryption DES in 1 sec on DES, takes 149 trillion years for 128-AES

Public Key cryptography

Instead of sharing the same key, public key crytpography invovles 2 keys, one public and one private. Both keys can be used to encrypto and decrypt, but when Alice is

sending something to Bob, she will use Bob's public key for encryption. This way, only Bob is able to decrypt Alice's message as Bob;s private key is used to decrypt (only known to Bob).



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

Rivest-Shamir-Adleman (RSA) Algorithm

1. Choose 2 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 (no common factors).
4. Determine d such that $e \cdot d - 1$ is divisible by z; that is d is the odular multiplicative inverse of e modulo z
5. The public key is $(n, e) = K_B^+$ and the private key is $(n, d) = K_B^-$.

RSA Encryption & Decryption

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

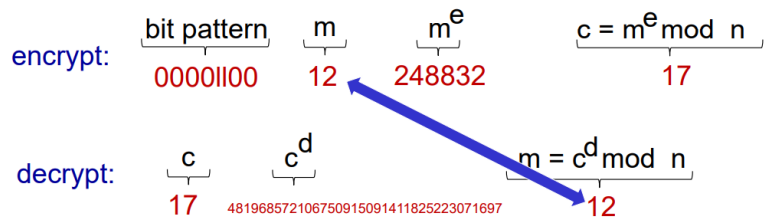
- To decrypt c, we compute $m = c^d \mod n$

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

RSA example: Bob Chooses

1. $p = 5, q = 7$.
   1. $n = pq = 35$,
   2. $z = (p-1)(q-1) = 24$
2. $e = 5$ (with $e < n$ & e and z are "relatively prime").
3. $d = 29$ such that $ed \bmod z = 1$.

encrypting 8-bit messages.



Combined

Using public key cryptography to encrypt every message is way too slow. For example, DES is at least 100 times faster than RSA. Instead, exchange a session (symmetric) key K_s, using public key cryptography, then use the shared symmetric key to communicate from there on.

Message Integrity and Digital Signatures

sender, receiver want to ensure message not altered (in transit, or afterwards) without detection ⇒ "error detection"

- Checksum / Parity / CRC

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, however it is computationally expensive.

Hash Function

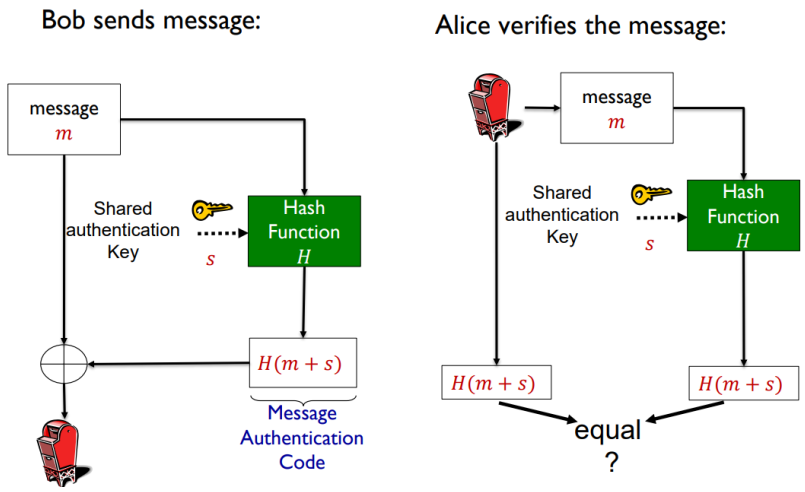
- Many to one
- Produces fixed size **message digest**
- Given a digest x, it is computationally infeasible to find m such that $x = H(m)$
- The Internet checksum has some properties of a hash function
- Common Hash functions:
  - **MD5:** Computes 128-bit message digest in 4-step provess. (Is obsolete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

Message Integrity

❌ Send $(m, H(m))$
→ does not work as sender, receiver want to ensure message not altered (in transit, or afterwards) without detection.
→ If attacker replaces $(m, H(m))$ with $(m', H(m'))$, no way for receiver to detect.

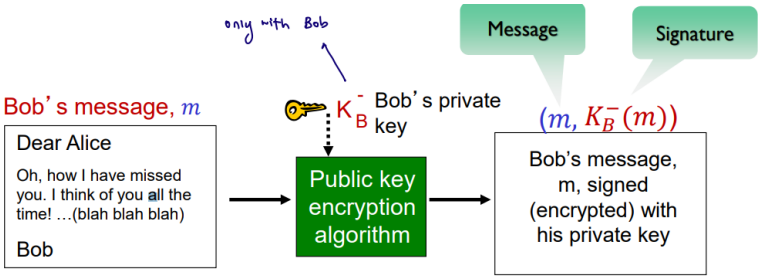
- sender and receiver share a "Authentication key" S
- To ensure Message integrity: Send $(m, H(m+s))$, also known as the Message Authentication Code
  - s is a secret key known to the receiver and no one else
  - receiver can generate the authentication code directly from m and compare with the received code

Message Authentication Code



Digital Signatures

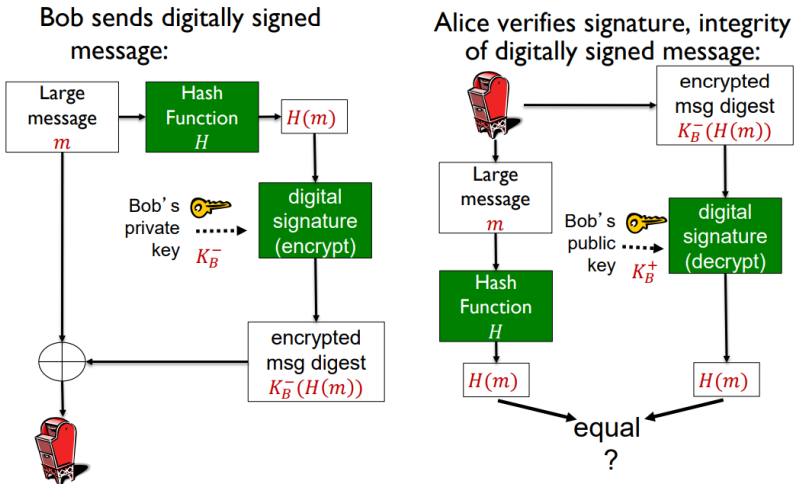
- Cryptographic technique analogous to *hand-written signatures*
- Sender (Bob) digitially *signs* document, establishing they are the document creator
  - **Verifiable:** Recipient (Alice) can check if the signature and the message was generated by Bob.
  - **Unforgeable:** No one, other than Bob should be able to generate the signature and the message.
- **Example:** Bob signs m by encrypting with his private key K_B^- creating the signature $K_B^-(m)$



- Suppose Alice receives msg m, with signature: $m, K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$
- If so, whoever signed m must have used Bob's private key.
  - Alice verified that
    1. Bob signed m
    2. no one else signed m
    3. Bob signed m and not m'
  - non-repudiation
    - Alice can take m and signature $K_B^-(m)$ to court and prove that Bob signed m

- compuotationally expensive to public-key-encrypt long messages
  - Apply hash function H to m, get fixed sized message digests $H(m)$

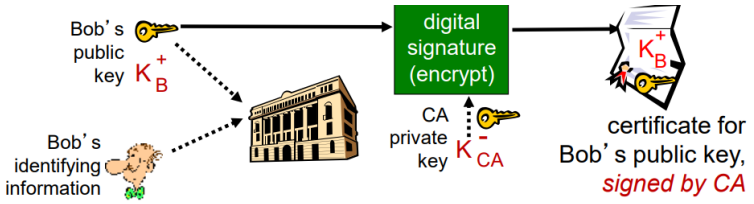
Digital signature = signed message digest



Certification Authority (CA)

Having public and private keys, we need a trustworthy source to get a list of public keys from → anyone can send a message claiming to be Bob, but we have no way of verifying that this person's public/private key pair is indeed Bob.

- Operating system has a list of "Trusted Root Certification Authorities"
- **certification authority (CA)**: binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key is digitally signed by CA



- when Alice wants Bob's public key
  - get Bob's certificate (from Bob or elsewhere)
  - apply CA's public key to Bob's certificate, get Bob's public key

Network-layer security

Virtual Private Networks

Institutions wants a standalone physical network (routers, links, DNS infra) that is completely separate from the public Internet → such a disjoint network is called a **private network.**

Above method is costly, thus many institutions now create VPNs over the existing Internet. With a VPN, the inter-office traffic is sent over the Internet but is encrypted beforehand, logically separated from other traffic.

- Gateway router in an office converts the vanilla IPv4 datagram into 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.
- Payload actually contains a IPsec header, which is used for IPsec processing, remaining payload is encrypted.

Operataional Security: Firewalls

A **firewall** is a combination of hardware and software that isolates an organnisation's internal network fromt he 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 modication or access of data, and allows only authorised access to different parts of the internal network

Limitations of firewalls:

- **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 external network

- **Three types of firewalls:**
  - stateless packet filters
  - stateful packet filters
  - application gateways

Stateless packet filtering

- internal network conneted to Internet via router firewall
- router filters packet-by-packet, decision to forward/drop packet based on:
  - soruce IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - Protocol type in IP datagram field: TCP, UDP, ICMP, OSPF
  - 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 conditions.

Policy	Firewall Setting
No outside Web access	Drop all outgoing packets to any IP address, port 80
No incoming TCP connection 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 outgoingICMP TTL expired traffic
all incoming/outgoing UDP flows blocked	Block incoming and outgoing datagrams with IP protocol field = 17 (UDP protocol)
prevent external clients from making TCP connections with internal clients but allows internal clients to connect to outside	block inbound TCP segments with ACK=0

Access Control Lists

tablel of rules applied top to bottom (sequentially) to incoming packets:

(action, condition) pairs

action	source address	dest address	protocol	source port	dest port	flag bit
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
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

Chapter 11. Multimedia Networking

Mains 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 10 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 redudancies to reduce data usage:
  - **Spatial Redundancy:** Instead of sending the same value (same color) N times, we just send the value (color) once and N (number of repeated times)
  - **Temporal Redundancy:** Only send differences between 2 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 (spatial/temporal) and gives better quality.
  - **Constant Bit Rate**: video encoding fixed, not responsive to complexity of video, and needs to be set high to handle more complex segments of the video. → consistency makes it suitable for *real-time* live streaming
  - **Variable Bit Rate:** changes as the amount of encoding changes → suited for *on-demand video* due to longer time to process data

Properties of Audio

- **Sampling:** To convert an audio analog signal into a digital signal, we sample the signal at somme 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.
  - 8,000 samples/sec, 256 quantized values (8bits) → 64,000 bps
  - receiver converts bits back to analog signal (Digital-to-analog converter) → some quality reduction

Example rates:

- CD: 1.411 Mbps
- MP3: 96, 128, 160 kbps
- Internet telephony: > 5.3 kbps

Streaming Stored Video:

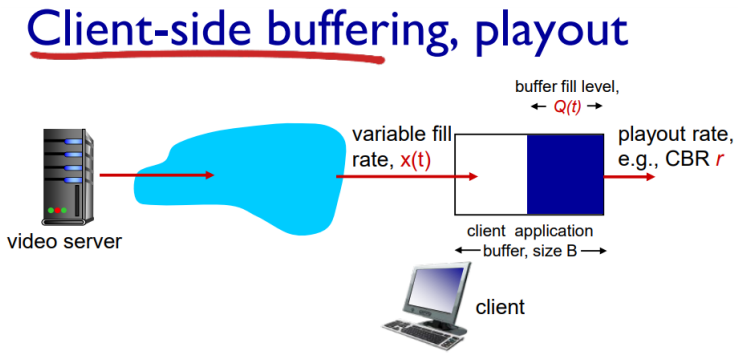
3 distinguishing features

- **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-foward and so on.
  - challenges → have to deal with client's actions

- **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.
  - challenges → network delays are variable, but playout must match
- other challenges → video packets may be lost or retransmitted

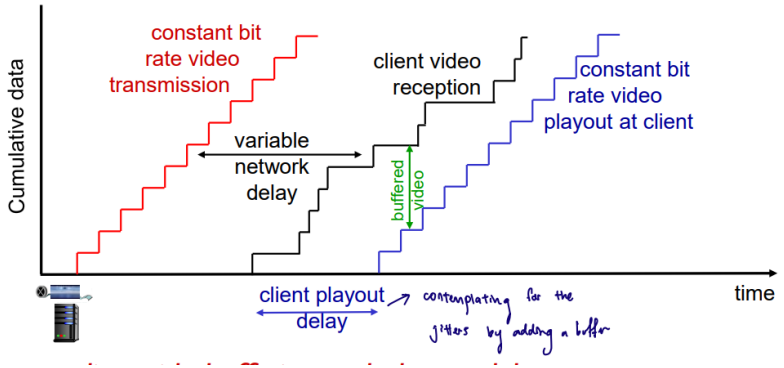
Client-Side Buffering

Extensive usage 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 need not immediately beginn 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 delays:** 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 bandwith briefly drops below the video consumption rate, the user can continue to enoy continuous playback, so long as the buffer is not drained

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



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 with rate-control restrictions of TCP.
  - Video consumption rate is 2Mbps, 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
- **RTP:** Uses Real-Time Transport Protocol, which has a separate control connection. (Video chunks are encapsulated using RTP)
- **Small Buffer:** Usually a small 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 separate media control: e.g. play, pause, skip etc. which increases complexity (RTSP).
  - RTSP used for establishing & controlling media sessions between endpoints
  - Client issue commands such as *play*, *record*, and *pause*

- **Firewall:** Many firewalls are configured to block UDP traffic, preventing users behind these firewalls from receicing UDP video.

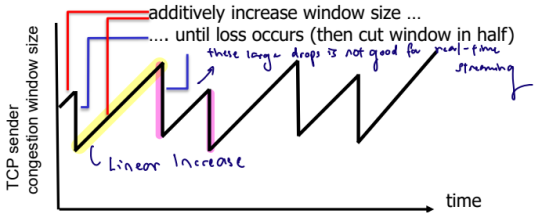
HTTP Streaming

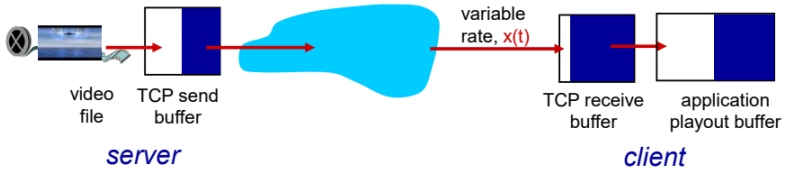
- **Pull-based Streaming:** Put simply, we try to download the video file from the server, and can potentially download at a rate higher than consumption rate, thus **prefetching video frames**.
- **Prefetching:** Occurs naturally as TCP's congestion avoidance mechanism tries to use all available bandwidth. Thus if bandwidth > consumption rate, this occurs.
- **Sending Flow:** Parts of the video file will be brought into server send buffer → client TCP receive buffer → client TCP application buffer, then finally application grabs frames from buffer and decompresses them. (if video file < buffer size, entire video will be downloaded)
- **Full Client Application Buffer:** The above flow 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 buffered frames will be wasted. 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.

Streaming multimedia: HTTP

- Advantages
  - HTTP/TCP passes more easily through *firewalls*
  - Network *infrastructure* (like CDNs and Routers) fine tuned for HTTP/TCP → all are traditionally fine tuned for TCP already
- Drawbacks
  - fill rate *fluctuates* due to TCP congestion control, retransmissions (in-order delivery) L TCP transmitted always tries to send the maximum possible rate, without overloading network.
  - *larger* playout delay: smooth TCP delivery rate

Additive Increase Multiplicative Decrease *saw tooth* behavior: *probing for bandwidth*





Voice Over Internet Protocol

Challenges:

1. IP Layer is a best effort service
   a. No upper bound on *delay*
   b. No upper bound on percentage of *packet loss*

- **Talk Spurts:** We don't always talk during a conversation, during periods of silence, nothing is sent. 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.

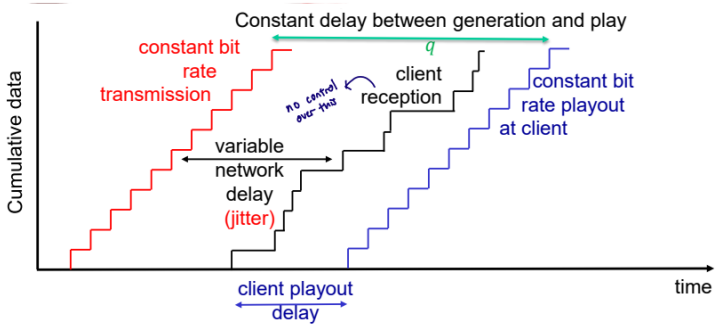
Limitations of VoIP: packet loss, delay

- **Network loss:** IP datagram lost due to network congestion (router buffer overflow, etc.)
- **Delay loss:** IP datagram arrives too late for playout at receiver
  - *delays*: proccessing, queueing in network; end-system (sender, receiver) delays
  - typical maximum tolerable delay: 400ms → discarded after 400ms
  - VoIP Applications typically use UDP to avoid Congestion control → as there is also a tighter delay requirements
    - Cannot use TCP as retransmission mechanisms are often considered unacceptable for real-time audio applications as they increase end-to-end delay
- **Loss tolerance:** depending on voice encoding, loss concealment, packet loss rates between 1% and 10% can be tolerated

Removing Jitter for Audio

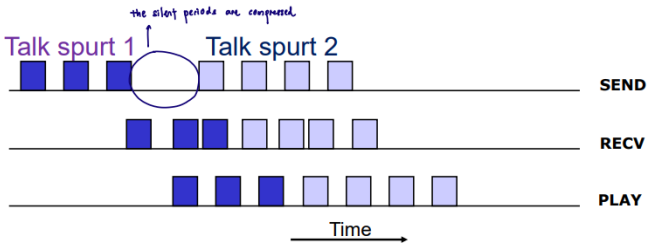
- **Timestamp:** Each chunk is prepended with a timestamp. The sender stamps each chunk with the time at which the chunk is generated.
- **Playout Delay:** This delay needs to be long enough so that most of the packets are received before their scheduled playout times.
  - **Fixed Playout Delay:** receiver attempts to playout each chunk exactly q msecs after chunk was generated.
    - chunk has time stamp t: playout chunk at $t + q$
    - chunk arrives after $t + q$: data arrives too late for playout: data "lost"

    tradeoff in choosing q:
    - large q: less packet loss
    - small q: better interactive experience



- No value of q can guarantee an optimal performance
  - We will eventually have a packet loss, or
  - Waste a lot of playout time

- **Adaptive playout delay:**
  - *goal*: low playout delay, low late loss rate
  - *approach*: adaptive playout delay adjustments
    - *estimate* network delay, *adjust* playout delay at beginning of each talk spurt
    - silent periods *compressed* and *elongated*
      - chunks still played out every 20 msec during talk spurt



Adaptive playout delay

Jargon Alert:
- EWMA: exponentially weighted moving average

- Adaptively estimate packet delay (EWMA):

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

d_i: delay estimate after ith packet; α: small constant, e.g. 0.1; $(r_i - t_i)$: time received - time sent (timestamp) = measured delay of ith packet

estimate of average deviation of delay after ith packet:

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

- Estimates, d_i *and* v_i calculated for every received packet, but used only at start of talk spurt
  - for first packet in talk spurt, playout time is:

$$\textit{playout-time}_i = t_i + (d_i + 4v_i)$$

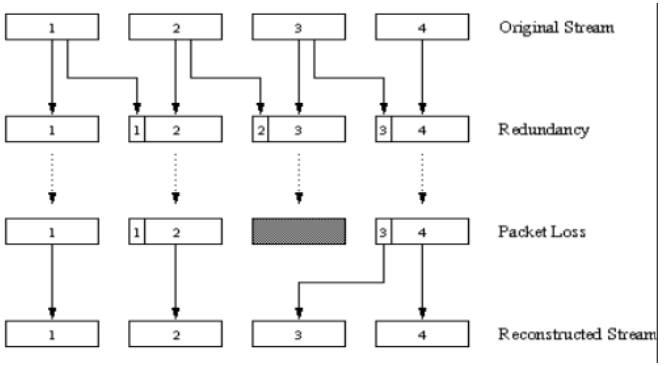
  - remaining packets in talk spurt are played out periodically

Recovery from packet lost

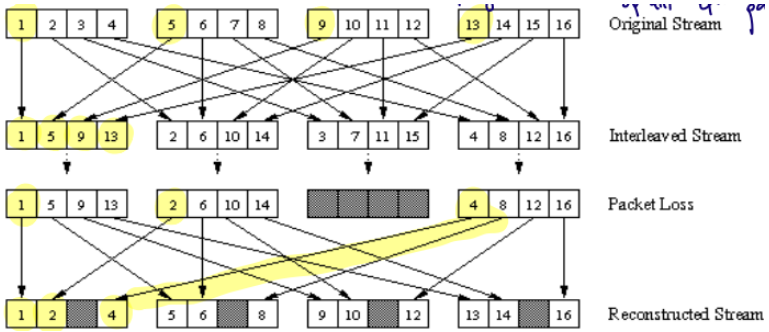
Challenge: recover from packet loss given small tolerable delay between original transmission and playout → using ACK/NAK takes RTT — too slow

- **Forward Error Correction (FEC):** send enough bits to allow recovery without retransmission (recall two-dimensional parity)
  - for every group of n chunks
    - create redundant chunk by XOR-ing n original chunks
    - send $n+1$ chunks
  - can reconstruct original n chunks if at most one lost chunk from $n+1$ chunks, with playout delay
  - Drawback
    - Increasing bandwidth by factor $\frac{1}{n}$
    - Playout delay is increased during packet loss
      - Recceiver waits for $n+1$ chunks before playout

        → cannot afford if playout delay has to be low
  - **FEC scheme (piggyback):** piggyback lower quality stream
    - send lower resolution audio stream as redundant information
      - nominal stream PCM at 64kbps and redundant stream GSM at 13kbps
    - *non-consecutive loss*: receiver can **conceal** loss
    - *generalization*: can also append (n-1)st and (n-2)nd low-bit rate chunk



- **Interleaving to conceal loss:**
  - Audio chunks divided into smaller units, e.g. 4 msec units per 20 msec audio chunk
  - packet contains small units from *different* chunks
  - if packet lost, still have *most* of every original chunk
    - Concealed by packet repetition or interpolation
  - no redundancy overhead, but increases playout delay, even without error → to playout the 1st chunk, have to receive up till the 4th chunk



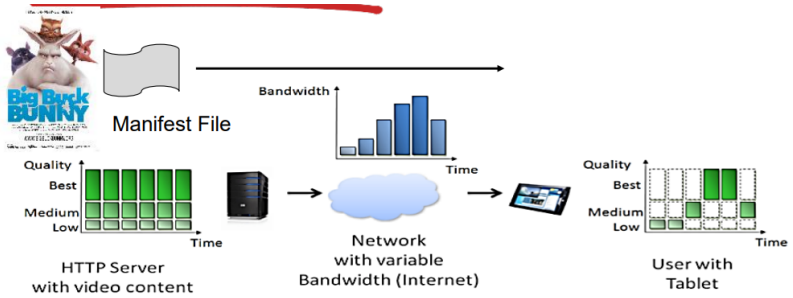
Dynamic Adaptive Streaming over HTTP (DASH)

- **Server:**
  - divides video file into *multiple* chunks
  - each chunk stored, encoded at *different rates (different quality)*
  - **manifest file:** provides URLs for different encoding
- **Client:**
  - periodically *measures* server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate *sustainable* given current bandwidth
    - *can choose* different coding rates at different points in time (depending on available bandwidth at time)

"intelligence" at client: client determines the following:

1. ***when*** to request chunk → buffer starvations/overflow does not occur
2. ***what encoding rate*** → what to request (higher quality when more available bandwidth)
3. ***where*** → where to request chunk (request fro URL server close to client or has the highest available bandwidth)

DASH



- Data is encoded into different qualities and cut into short segments (streamlets, chunks).
- Client first downloads *Manifest File*, which describes the available videos and qualities.
- Client/player executes an adaptive bitrate algorithm (ABR) to determine which segment do download next.

- **Streamlets or Byte Range:** 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 existing web caching infrastructure that ISPs and CDNs have built.

Tradeoffs of DASH

- **Advantages:**
  - Server is simple, i.e., regular web server (no state, proven to be scalable)
  - No firewall problems (use port 80 for HTTP)
  - Standard (image) web caching works
- **Disadvantages:**
  - DASH is based on media segment transmissions, typically 2-10 seconds in length
  - By buffering a few segments at the client side, DASH does not:
    - Provide low latency for interactive, two-way applications (e.g., video conferencing)

Content Distribution Networks (CDN)

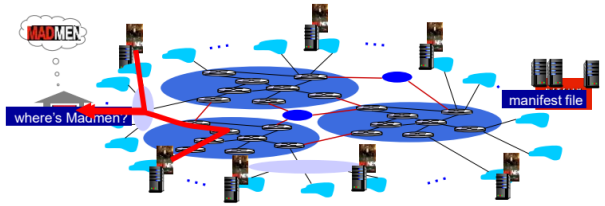
how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

1. single, large “mega-server” ⇒ NON-SCALABLE
   - single point of failure, point of network congestion, long path to distant clients, multiple copies of video sent over outgoing link
2. store/serve multiple copies of videos at multiple geographically distributed sites (CDNs)

   enter deep: push CDN servers deep into many access networks
   - Usually at ISP (Internet Service Provides)
   - close to users
   - used by Akamai, 1700+ locations

   bring home: smaller number (10’s) of larger clusters in IXPs (Internet Exchange Points) near (but not within) access networks
   - used by Limelight

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
- Client requests content
  - service provider returns manifest
- using manifest, client retrieves content at highest supportable rate
- may choose different rate or copy if network path congested



RTP RTCP RTSP - Good to know

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

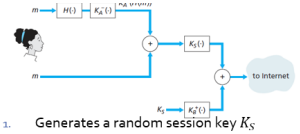
- **Payload Type (7 bits)**: Indicates the type of media encoding currently being used. If the sender changes their encoding during the call, the sender informs the receiver via the payload type field.
  - Payload type 0: PCM mu-law, 64 kbps
  - Payload type 3: GSM, 13 kbps
  - Payload type 7: LPC, 2.4 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.

Questions

1. Using the substitution cipher encode/decode messages

> 💡 If a non-bijective function (one-one) is used on this ciphertext, it would not work. This function has to be one-one to ensure that the conversions are always the same

2. Suppose N people want to communicate with $N-1$ people, communication is visible, but no other person should be able to decode their communication
   a. symmetric key encryption: **need a singular key** with every other person → complete graph $\therefore = \frac{n(n-1)}{2}$ total keys
   b. public key encryption: total N people, everyone has public/private key pair, $\therefore N \times 2 = 2N$ total keys
3. Suppose Alice wants to send a secure email m to Bob, and wants to ensure its confidentiality and integrity. Alice performs the following steps.



1. Generates a random session key K_S
2. Encrypts the session key with Bob's public key K_B^+, obtaining $K_B^+(K_S)$
3. Hashes the message m with a cryptographic hash function H, obtaining message digest $H(m)$
4. Encrypts the hash with Alice's private key K_A^-, obtaining digital signature $K_A^-(H(m))$
5. encrypts the message m, concatenated ($\oplus$) with $K_A^-(H(m))$, using the session key K_S to obtain $K_S(m \oplus K_A^-(H(m)))$
6. finally, sends $K_S(m \oplus K_A^-(H(m))) \oplus K_B^+(K_S)$ to Bob

Bob has to:

1. **recover the session key using Bob's private key by computing**
   $$K_B^-(K_B^+(K_S)) = K_S$$
2. **with K_S, Bob decrypts the message**
   gets m and $K_A^-(H(m))$
3. **use Alice's public key K_A^+ to recover $H(m)$: $K_A^+(K_A^-(H(m))) = H(m)$**
4. **with m, Bob computes $H(m)$ and verifies that it is equal to $H(m)$ from step 3**

- **AUTHENTICITY:** Private key signature → Alice signs the Hash with her private key, only Alice is able to encrypt $\therefore$ only Alice could have written it
- **INTEGRITY:** Hash function → If attacker does not have session key K_s, attacker is unable to hash their "fake" message properly even if they can spoof the packet, hence this ensures integrity
- **CONFIDENTIALITY:** Encryption with public key → K_s (or the session key) is encrypted with Bob's public key, hence only Bob can decrypt, no one else can decrypt it

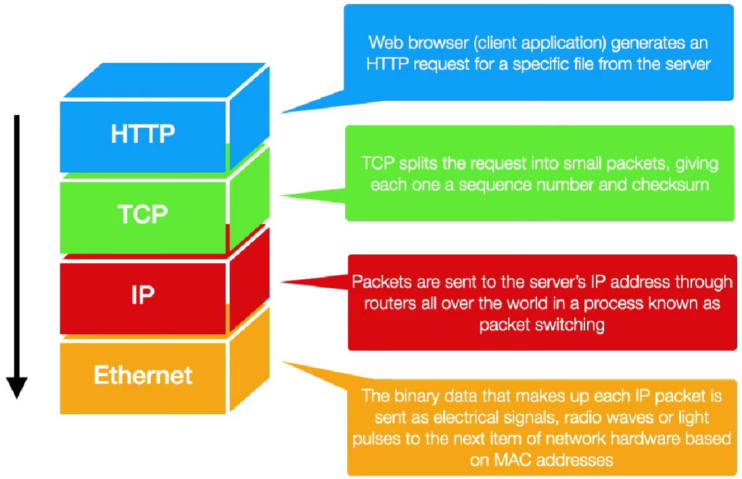
Past Year Paper Questions

1. A web server supports both `HTTP/1.0` and `HTTP/1.1`, so far 120 clients have downloaded a webpage from the server, which contains 1 HTML file and 3 images. Half the clients run `HTTP/1.0` and the other half runs `HTTP/1.1`.

$$HTTP/1.0 = 60 \times 4 = 240\ sockets$$
$$HTTP1.1 = 60 \times 1 = 60\ sockets$$
$$Total = 60 + 240 = 300$$

2. Which of the following statement is TRUE when a packet containing application message is passed from **router *A* to router *B*** in the Internet?
   a. ***Upon arrival, the packet may be discarded by B if B's buffer is full.*** ✓
   b. *A* and *B* must establish a TCP connection before the packet is retransmitted. ✗ - not transport layer
   c. *A* may pass the packet to *B* through a UDP connection. ✗ - not transport layer
   d. Circuits must be reserved before *A* can pass the packets to *B*. ✗ - not circuit switching

> 💡 Routers are communicating in the **network** layer.



3. A Go-Back-N sender just receives an ACK packet with sequence number 14. This ACK number falls within sender's window. Sender's window size is 6. Every packet embeds a k-bit sequence number field (k is an unknown constant). Which of the following definitely CANNOT be the sequence number of the next packet transmitted by the sender?

   a. 4

   b. ***9***

   c. 15 ⇒ if the last packet sent by the sender is 14, the next packet will be 15

   d. d. 19 ⇒ if sender has already sent [14, 15, 16, 17, 18], the next packet will be 19

   e. e. 20 ⇒ if sender has already sent [14, 15, 16, 17, 18, 19], the next packet will be 20

   Suppose k = 4, sequence number space will be 0 until 15.

   If sender has already sent [14, 15, 0, 1, 2, 3], the next packet will be 4. But you cannot find a scenario that next packet will be 9, with the conditions laid out in the question.

4. Consider a Go-Back-N reliable transmission protocol with a k-bit sequence number and sending window of size 3, operating over a channel that can delay, corrupt or lose packets, but not reorder them. **The sender has just sent a new**

packet with sequence number 2.

Which of the following events could have **directly preceded** (came before) this (i.e. no other packets were sent or received in between)?

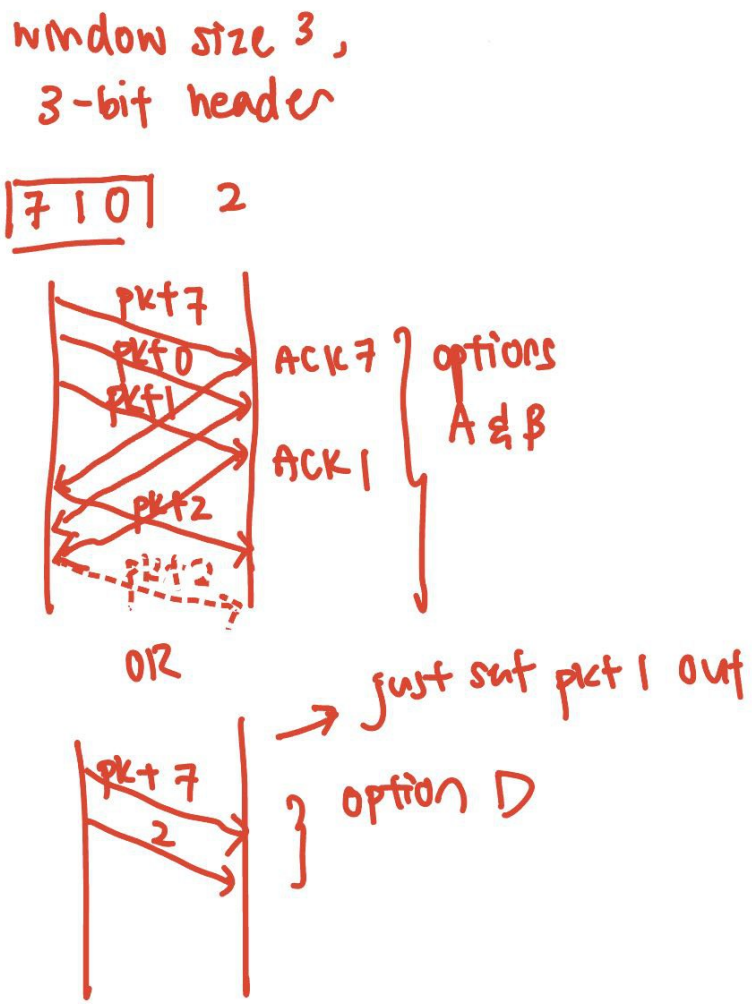
A. Sender received an ACK with sequence number 1

B. Sender received an ACK with sequence number 7

C. Sender sent a packet with sequence number 0 ⇒ cannot be 0 as 1 is between 2 and 0

D. Sender sent a packet with sequence number 1

E. Sender sent a packet with sequence number 4 ⇒ 4 is after 2



4. A file of size 9990 bytes is transferred over a TCP connection. The connection is still open after file transmission. MSS is 1000 bytes and TCP sends as much data as possible in a segment. Assume TCP header is 20 bytes, what is the size of the last TCP segment (including TCP header and file data)?
   - TCP header is not included in MSS.

   - Per segment, number of bytes = $1000 + 20\ byte\ header$
   - Last segment = $9900 - \lfloor \frac{9900}{1000} \rfloor = 900\ bytes$
   - Last segment included TCP header = $\underline{900 + 20 = 1010\ bytes}$
5. In rdt 3.0, what does the sender do if it receives a corrupted ACK and what does the receiver do if it receives a corrupted packet?

   A. Sender does nothing; receiver does nothing.
   B. Sender does nothing; receiver sends ACK for the previous packet.
   C. Sender resends data packet; receiver does nothing.
   D. Sender resends data packet; receiver sends ACK for the previous packet.
   E. None of the rest
6. A huge file is transferred over an existing TCP connection (i.e., 3-way handshake is already done). The connection is still open after transmission. The first and last TCP segments have the sequence numbers 12,345 and 2,105 respectively. MSS is 1,024 bytes and TCP sends as much data as possible in a segment. How many TCP segments are used to transfer the file (i.e. carries file data), assuming the communication channel is perfectly reliable? (Hint: TCP sequence number will wrap up and restart from 0 after reaching the biggest sequence number)

   Sequence number begins at 12345, goes all the way up as more and more bytes are sent, until 2^32-1 is reached, then restarts from 0 until 2105. So the number of bytes transmitted all in packet (except the last one) is 2^32 + 2105 - 12345. 4194294 packets are used to carry all these bytes.

   But don't forget the last packet which carries data from byte 2105 onwards.

   So the total number of packets is 4194294 + 1.
7. Two hosts A and B are connected by a router R as shown in the following diagram.

   For link 1, link transmission rate is 1 Kbps and propagation delay is 100 milliseconds.
   For link 2, link transmission rate is 250 bps and propagation delay is 150 milliseconds.
   Suppose Host A sends 2000 packets to Host B continuously and each packet is 500 bits
   long. Host A starts sending the 1st packet at time t = 0.
   When (in seconds) will host **B** receive the k^{th} packet ($1 \leq k \leq 2000$)?

   The key observation is that link 2 is slower than link 1. Hence packets will queue in R. The easiest calculation is as follows:

Step 1: calculate the time 1st packet reaches R. Now R will start transmitting all packets

continuously onto the second link (this way we avoid calculating queuing delay).

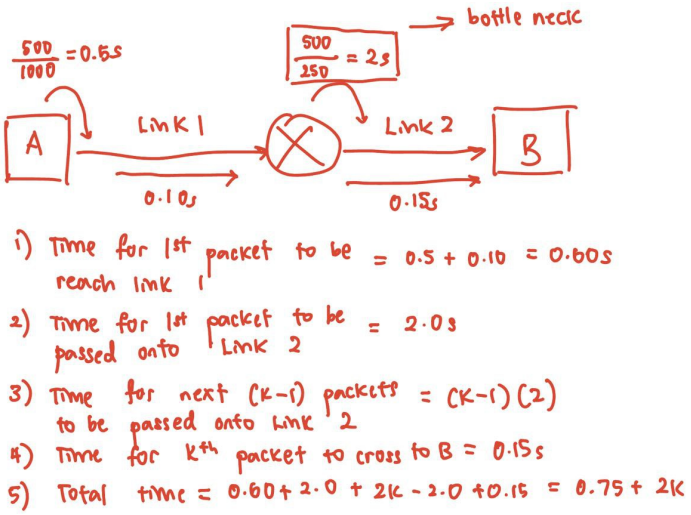
- $\frac{500}{1}$ kbps + 100 milliseconds = 0.6 seconds

Step 2: calculate the time kth packet leaves R (i.e. transmission delay of k packets).

- 0.6 + $\frac{k \times 500}{250}$ bps = $0.6 + 2k$ second

Step 3: calculate the time kth packet propagate to B.

- $0.6 + 2k + 0.15$ seconds = $2k + 0.75 seconds$



9.

```
mySocket = socket(AF_INET, SOCK_STREAM) #SOCK_STREAM is TCP
mySocket.connect(('sunfire.comp.nus.edu.sg', 2105))#port number of server to connect to
```

Suppose no runtime exception is raised, what port number is `mySocket` bound to when above statements finish execution?

A. It depends on the remote host's port that's making the connection.
B. TCP port 2105

C. Cannot say; it's operation system dependent and is usually a randomly chosen port. ⇒

- Client socket, client port number is dynamically allocated.

D. UDP port 2105
E. None of the rest

10. RDT 3.0 protocol sender is correctly implemented, the following is implemetation of receiver:

```
byte[] recv() throws IOException, ClassNotFoundException {
  DataPacket p = udt.recv();
  while (p.isCorrupted || p.seq != seq)
  p = udt.recv();
  udt.send(new AckPacket(p.seq));
  seq = 1 - seq;
  return deliverData(p);
}
```

Here, seq is the expected sequence number (either 0 or 1) at the receiver, and deliverData is
a method that extracts and returns the payload from packet p. The protocol is used over a channel that may lose or corrupt a packet, but always delivers packets in the order that they are sent.
We say that the receiver ***waits forever***, if it is blocked at the call `udt.recv()`, waiting to receive a packet that will never be sent. We say that the ***sender loops*** forever, if it repeatedly retransmits the same packet over and over again. Which of the following statement CORRECTLY describes the behavior of the protocol implemented above?

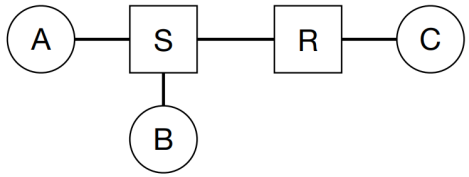
A. A single corrupted data packet is sucient to cause the sender to loop forever.
B. A single corrupted data packet is sucient to cause the receiver to wait forever.
C. A single loss ACK packet is sucient to cause the sender to loop forever.
D. A single loss ACK packet is sucient to cause the receiver to wait forever.
E. A single premature timeout is sucient to cause the sender to loop forever.

💡 A loss ACK would cause the sender to send a duplicate packet to the receiver
after timeout. Since the receiver does not send an ACK on duplicate packet, the sender
keeps timing out and loops forever. Note: if a packet is corrupted, the sender would time
out and resend the packet (A and B are wrong). A premature timeout would cause the
sender to send a duplicate packet as well, but eventually ACK is received by the sender, so
the sender won't loop foever.

11. Which of the following protocols does NOT run on a router?

A. IP	D. ICMP	UDP is in the transport layer and runs only in the end host.
B. UDP	E. ARP	
C. RIP	F. None of the above	

12. How many IP Addresses are used in this network? Which entities do these addresses belong to?



5 → 2 for R, one each for A, B, and C.

(i) A **switch** is a **link-level** device to interconnect between hosts. A **switch interface DOES NOT HAVE A** MAC address, nor does it have an IP address.

(ii) An IP address is associated with a **network interface**. Thus, R has two IP addresses.

⇒ router has a public IP address, a private "management" IP address, and then additional private IP addresses for each device in addition to the router's internal IP, which is your LAN default gateway.

(iii) At the link layer, C only communicates with R, and A only communicates with B and R. Therefore, the ARP table of C does not contain entries for A and B; ARP table of A does not contain entry for C.

13. 1s complement is used as checksum in UDP, TCP and IP.

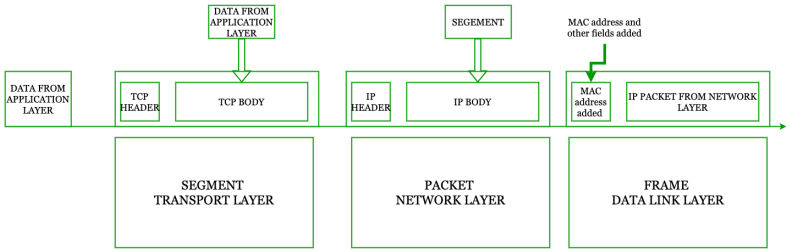
14. Which of the following statement about IP datagram is FALSE?

A. Routing protocols determine the routes that datagrams take between sources and destinations
B. TTL field of IP header prevents a datagram from circulating in the network forever.
C. When a big datagram is fragmented into a series of smaller fragments, transport layer header will be replicated in each fragment
D. On the Internet, datagrams from the same source may take different routes towards the destinations.
E. MTU of the link-layer protocol places a limit on the length of a datagram.

Protocol Data Unit

Layer	Term	Remarks	Header
Transport	**Segment**	The data from the application layer is broken into smaller parts as per the MSS of the network and the TCP header is added to the smaller parts. - TCP Header not included in MSS - Source and Dest port included as it tells this PDU where to be delivered in host	1. Source Port 2. Destination Port 3. Flag bits (like DF, MF, etc) 4. Sequence Number of the Segments 5. Checksum 6. Options Field

Layer	Term	Remarks	Header
Network	**Packet**	The segments received from the Transport layer are further processed to form the Packets, adding the IP header infront. - MTU includes IP Header. - IP header always modified as TTL changes on each hop. - Network layer responsible for fragmentation (done in routers) if MTU is smaller. - **IP Datagram** (meant when IP header appended to transport layer datagram)	1. Source IP Address 2. Destination IP Address 3. TTL(time to live) 4. Identification 5. Protocol type 6. Version (version of protocol) 7. Options
Data-Link	**Frames**	The Packets received from the Network Layer further processed to form the Frames, adding the data link layer. - Data-link header is added - MAC addresses resolved from ARP - Source and Destination MAC would be modified while moving in the network (done by routers).	1. Source Mac Address 2. Destination Mac Address 3. Data - network layer packet 4. Length 5. Checksum (CRC)



Special Ports / Commands in Networks

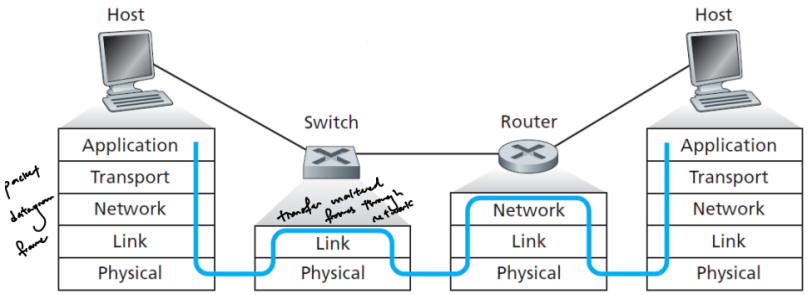
- Ports

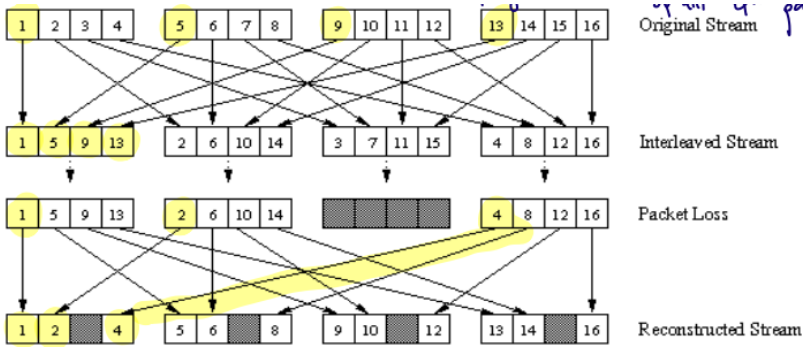
Port number	Use Case
20, 21	File Transfer Protocol (FTP). FTP is for transferring files between a client and a server.
22	Secure Shell (SSH). SSH is one of many tunneling protocols that create secure network connections.
25	Simple Mail Transfer Protocol (SMTP). SMTP is used for email.
53	Domain Name System (DNS). DNS is an essential process for the modern Internet; it matches human-readable domain names to machine-readable IP addresses, enabling users to load websites and applications without memorizing a long list of IP addresses.
80	Hypertext Transfer Protocol (HTTP). HTTP is the protocol that makes the World Wide Web possible.
123	Network Time Protocol (NTP). NTP allows computer clocks to sync with each other, a process that is essential for encryption.
179	Border Gateway Protocol (BGP). BGP is essential for establishing efficient routes between the large networks that make up the Internet (these large networks are called autonomous systems). Autonomous systems use BGP to broadcast which IP addresses they control.
443	HTTP Secure (HTTPS. HTTPS is the secure and encrypted version of HTTP. All HTTPS web traffic goes to port 443. Network services that use HTTPS for encryption, such as DNS over HTTPS, also connect at this port.
500	Internet Security Association and Key Management Protocol (ISAKMP), which is part of the process of setting up secure IPsec connections.
3389	Remote Desktop Protocol (RDP). RDP enables users to remotely connect to their desktop computers from another device.

- Commands

Command	Use Case
`telnet`	Telnet utility allows users to **test connectivity to remote machines and issue commands through the use of a keyboard**
`nslookup`	It is used to find the DNS mapping between hostname and IP address.
`dig`	The dig command in Linux is **used to gather DNS information**. It stands for Domain Information Groper, and it collects data about Domain Name Servers.
`ping`	The ping command is a general utility which is **used for checking whether any network is present and if a host is attainable**.
`netsat`	The netstat provides the statistics and information in the use of the current TCP-IP Connection network about the protocol.

`ipconfig` (Windows)	The command IP config will display basic details about the device's IP address configuration. Just type IP config in the Windows prompt and the IP, subnet mask and default gateway that the current device will be presented.
`traceroute`	tracing the route to some domain, tell us the IP address of that domain and the maximum number of hops before timing out.





Tutorial Questions