Computer Networks

① What are computer networks?

↳ essentially, networks of communicating devices.

hosts / end systems — end-devices that transmit or receive messages

communication links — the different physical media (eg. cables, optical fibers, radio spectrum) that transmit messages between devices

packets — when one end system has data to send to another end system, the sending end system segments the data and adds header (metadata) bytes to the segment. The resulting packages of bytes are known as packets and sent through the network.

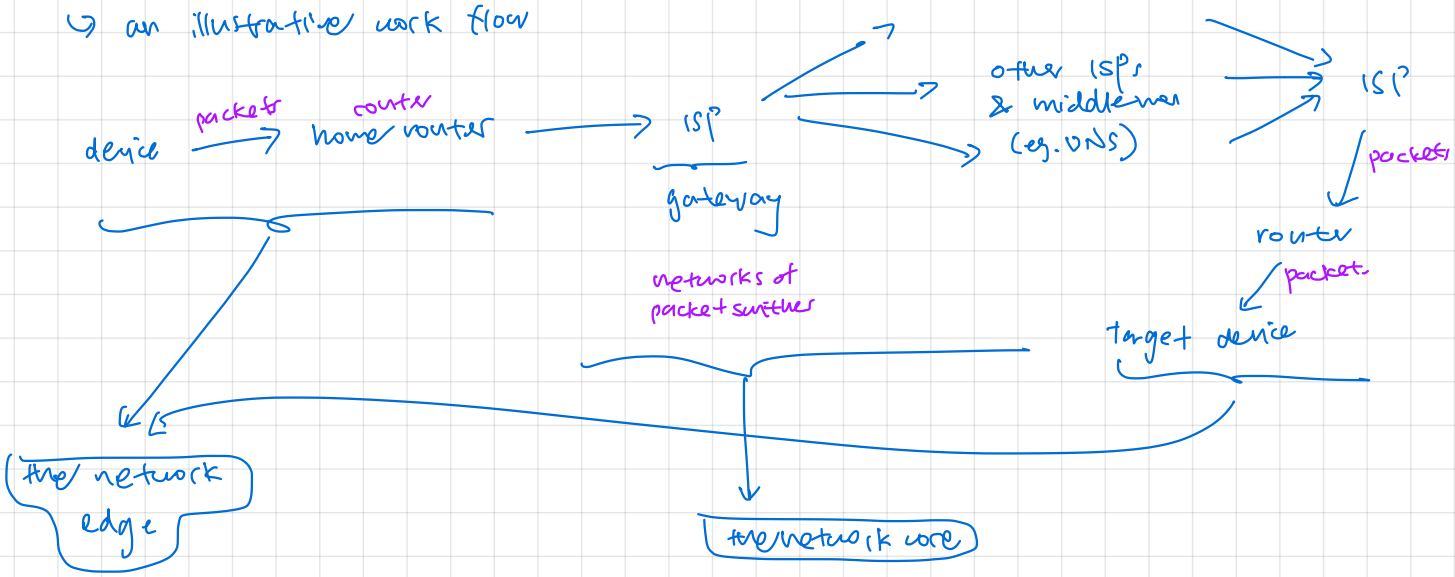
Internet service providers (ISP) — itself a network of packet switches & communication links that provide a variety of types of network access to other end systems. think of them as a gateway / middle men to content providers and other end systems.

2) what is the internet?

↳ conceptually, an infrastructure that provides service to applications

↳ physically, a network of networks, decentralised but coordinated by alliances & regulatory bodies.

↳ an illustrative work flow



---

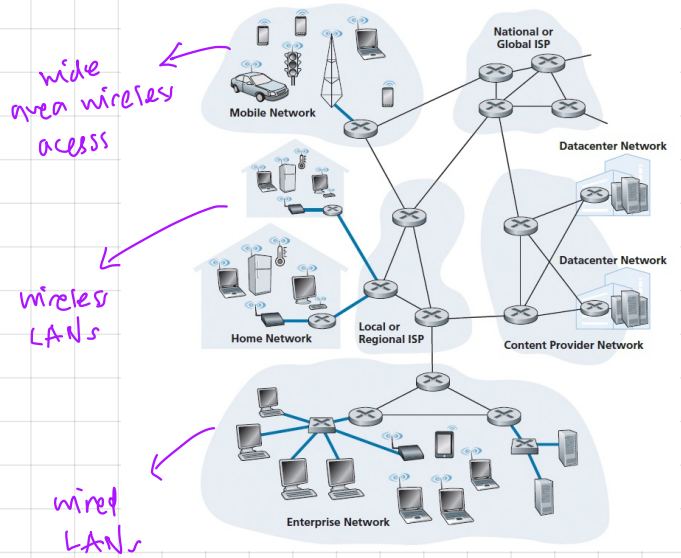
② The network edge

↳ components that exist at the "edge" — near the end systems.

(wireless) access networks

↳ the network that physically connects an end system to the first router

↳ this router is known as an edge router



physical media

↳ end systems connect to the access network via different physical media

- guided media
  eg. cables
- unguided media
  eg. radio signals

③ The network core

↳ the mesh of packet switches and links that interconnect the internet's edge systems

1) how data is transmitted through the network: packet switching

packet switches A device that takes a packet arriving on one of its incoming communication links and forwards that packet on one of its outgoing links.

↳ packet switches come in many forms, 2 most common are

routers
- typically used in network core

link-layer switches
- typically used in access networks

route / path The sequence of communication links & packet switches traversed by a packet from source to destination

store & forward transmission The packet switch must receive the entire packet (all bits) before it can begin to transmit the first bit on an outbound link.

↳ to send a message from one end system to another, the source breaks the message into chunks called packets

↳ each packet travels through the communication links and packet switches, going through store & forward transmission

2) routing & addressing

IP address Every end system has an IP address that uniquely identifies it.

↳ a router takes an arriving packet & forwards it

↳ each packet must have the destination IP in the header, and every router has a forwarding table that maps this address to an outbound link

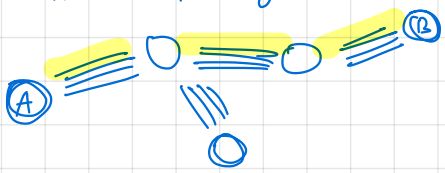
⇒ this process continues for every router until packet reaches correct destination

⇒ but how do we decide where to map to? ⇒ routing algorithms & hierarchical IP addressing

2) circuit switching as an alternative

↳ in packet switching, we can think of packet switches as broker nodes in a graph, directing traffic as needed. These nodes are shared.

↳ an alternative way to transmit is to have reserved connections — that is, for a given link (circuit) between two nodes, we reserve it for messages between two end systems.



e.g.

A to B, we need to reserve a circuit in each link.

3) packet switching vs. circuit switching

↳ people argue for circuit switching for real-time comms, while others argue packet switching has higher utilization & lower cost.

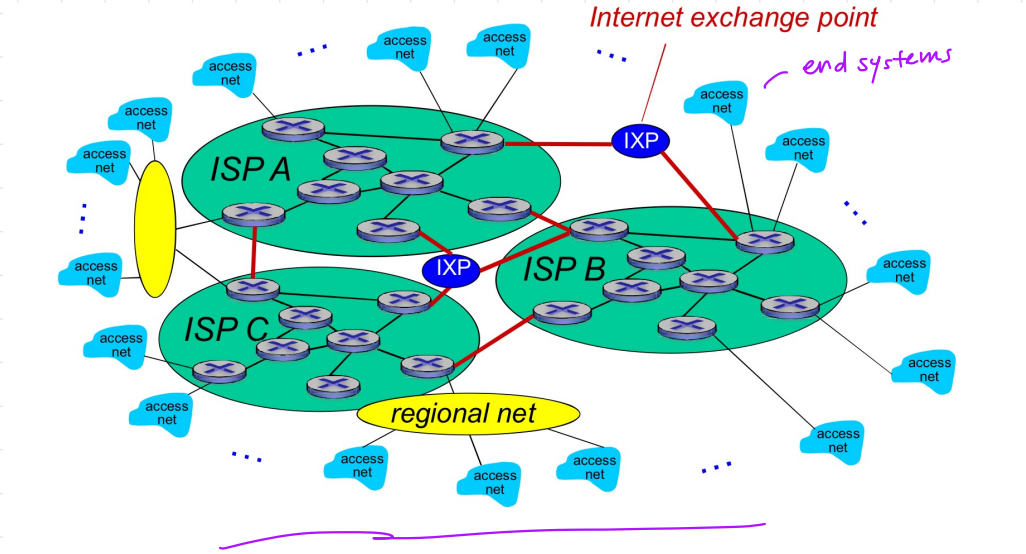
⇒ indeed, packet switching is more efficient as more users can use the same number of physical links at the same time, with less idle time

although, packet switching requires overhead in reassembly + space overhead from header metadata

4) the core as a network of networks

key idea: hosts connect to ISPs ⟶⟵ ISPs are connected to each other

network of networks, each autonomous and owned by an entity / organization



ISPs connected @ different scales

5) who runs the internet?

Who Runs the Internet?

- IP address & Internet Naming administered by Network Information Centre (NIC)
  - Refer to: www.sgnic.net.sg; www.apnic.org
- The Internet Society (ISOC) - Provides leadership in Internet related standards, education, and policy around the world.
- The Internet Architecture Board (IAB) - Authority to issue and update technical standards regarding Internet protocols.
- Internet Engineering Task Force (IETF) - Protocol engineering, development and standardization arm of the IAB.
  - Internet standards are published as RFCs (Request For Comments)
    - Refer to: www.ietf.org; for RFCs: http://www.ietf.org/rfc.html

---

④ Delay, loss & throughput in packet-switched networks

1) how do delays & losses occur?

↳ in a packet switching network, packets traverse by arriving at and being forwarded

↳ a queue of packets to be forwarded forms @ each router ⇒ **queueing delay**

↳ if queue is full, packet nowhere to go ⇒ **packet loss**

↳ at router, need to do some processing ⇒ **nodal processing delay**

↳ after processing, router can push packets into link at a certain rate ⇒ **transmission delay**

$$d_{trans} = \frac{N_{bits}}{R_{transmission}}$$ — bits/sec

↳ as packets traverse link ⇒ **propagation delay**

calculating end-to-end delay

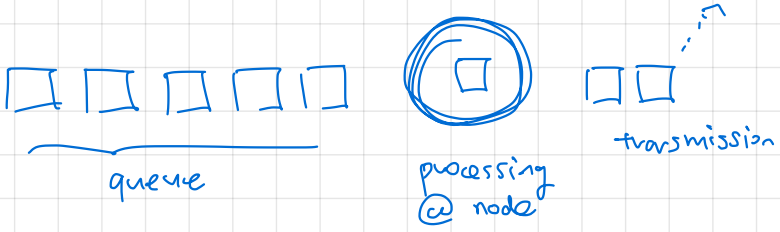
↳ stream of packets

for the first bit:

1. arrival & queuing
2. processing
3. transmission (get into link)
4. propagation

for all other bits:

think of inbetween



2) throughput

↳ describes end-to-end

↳ how many bits per unit time?

consider:
- relative rates @ each router, possible bottlenecks?
- visualize streams

---

⑤ the layered architecture of the internet

network protocol: A defined format & order of messages exchanged and the actions taken before & after messages are sent & received

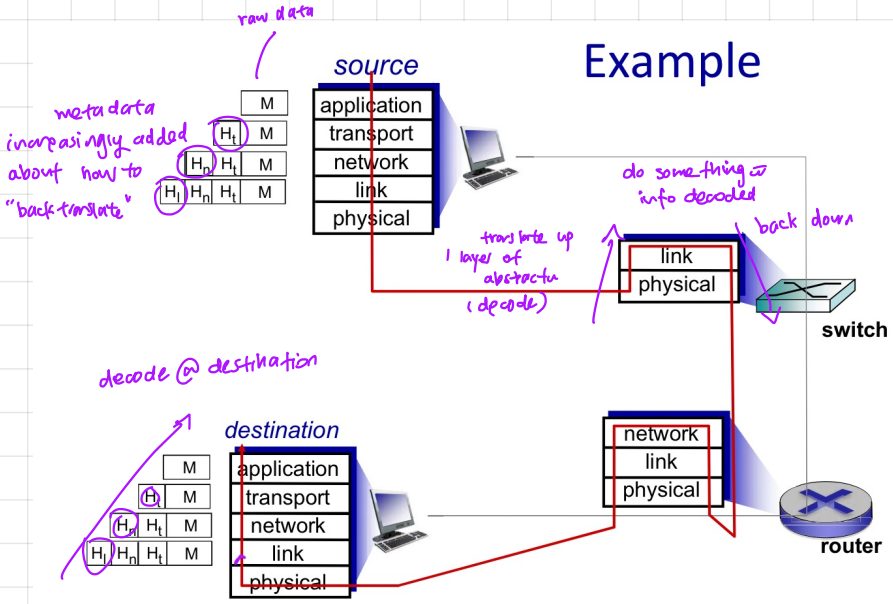
↳ the internet, as a network, involves end devices running processes, and these processes trying to send messages through the network to other processes on other devices.

⇒ a lot of different functionalities & services required

⇒ the internet uses a protocol stack to logically organize communications:

- *application:* supporting network applications → application to application
  - FTP, SMTP, HTTP
- *transport:* process-to-process data transfer → process to process
  - TCP, UDP
- *network:* routing of datagrams from source to destination → node to node (how to traverse)
  - IP, routing protocols
- *link:* data transfer between neighboring network elements → node to link to node (how to transmit)
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits "on the wire" → actual physical media specifications

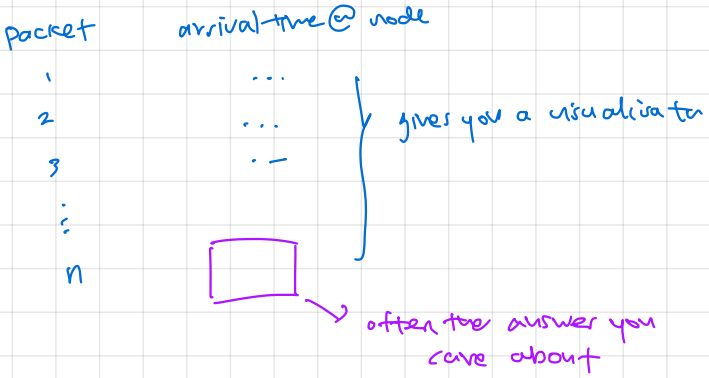
composed as a "protocol stack", what it means is we have increasing layers of abstraction, and define protocols to convert between them, so we can transmit information down & then back up.



organising in this layered way is natural for networks, where there are different layers of abstraction (OS vs. app vs. physical comm. media, etc.)

Notes

1. visualizing packet transmission



2. Be mindful of bytes vs. bits!

Application layer

① applications, process and sockets

1) process & sockets as an interface to transport layer

↳ in OS terms, it is processes on different end systems that communicate with one another

↳ end systems are identified by IP address. How about processes on those systems?

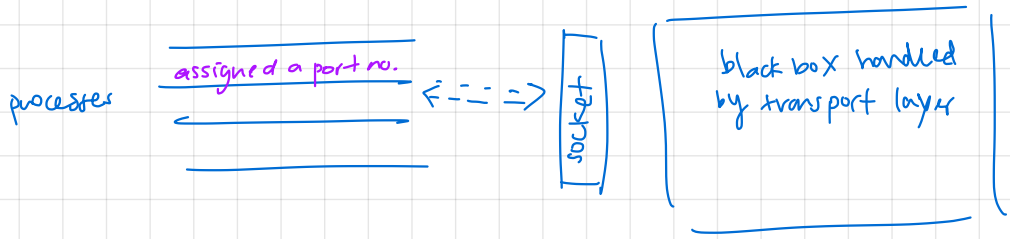
↳ A (port number) serves this purpose. So, IP + port number uniquely addresses a process.

↳ a process sends messages (bytes) to, and receives (bytes) from the underlying network through a software interface called a (socket). The is the interface between the application & transport layer within a host.

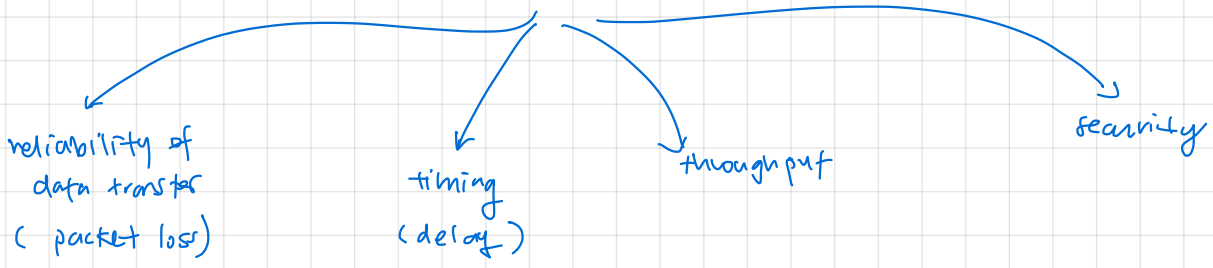
⇓

the developer has complete control @ application layer, but only choice of transport protocol below that

↳ to the app, it looks like this:



2) a high level overview of transport layer protocols



eg. TCP, UDP

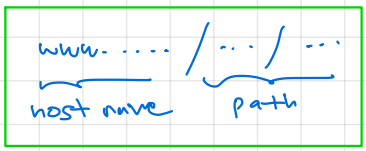
(2) webresources & addressing URLs

1) web resources

↳ we can think of all message exchanges as the request / return of resources

e.g. a typical web page consists of an index.html file and some objects

↳ every resource or its handler should be identifiable by a unique address



URL
- user-friendly, readable
- not hierarchical or easy to manipulate

IP address
- easy to manipulate
- not readable easily

we need a mapping between the two

⇓

domain name system (DNS)

2) DNS as a distributed, hierarchical database of mappings

DNS translates from url to ip address and commonly employed by various application-layer services (eg. HTTP, SMTP) to translate user-supplied hostnames to IP addresses runs over UDP

↳ the DNS uses a large number of servers organised in a hierarchical & regional fashion to avoid a single point of failure

Root DNS servers There are >1000 root server instances globally, that are copies of 13 different root servers, managed by 12 organisations & coordinated by the IANA. They provide the IP addresses of top-level domain (TLD) servers.

Top-level domain servers for each of the top-level domains (.com, .org, .edu) and all country top-level domains (.fr, .jp) there is a TLD server, each maintained by some organisation. These provide the IP of authoratative DNS servers

authoratative servers Every organisation with publicly accessible hosts on the internet must provide publicly accessible DNS records. An organisation can implement its own authoratative DNS server to hold these records or pay to have them stored in one of a service provider.

local DNS server Each ISP has a local DNS server

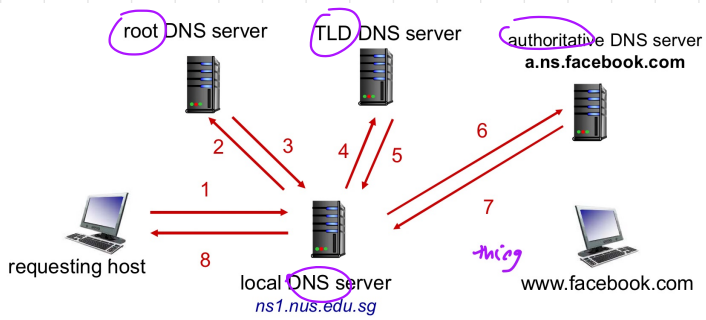
DNS resource records mapping between host names and IP addresses.

(name, value, type, ttl)

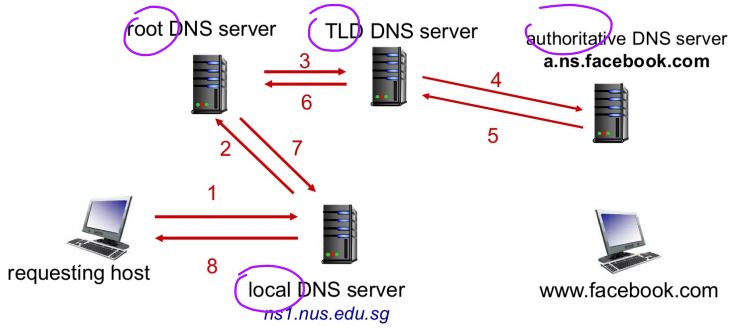
eg. www.nus.edu

- **A**: name → hostname; value → IP
- **CNAME**: name → alias name; value → canonical, real name of server
- **NS**: name → domain; value → hostname of authoritative server
- **MX**: value → name of mail servers associated w name

how DNS name resolution works



- This is known as *iterative query*. to & fro



- This is known as *recursive query*. ie. others do it for you
  - rarely used in practice

↳ heirarchically query until authoritative. local DNS is middleman.

DNS caching

↳ for speed, DNS server tends to cache entries. They expire after some time (TTL).

③ Application layer protocols as an abstraction

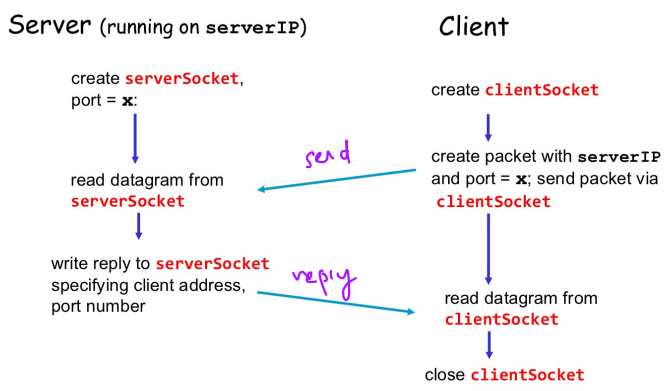
1) raw socket programming

↳ recall that sockets are the interface between transport & application layer

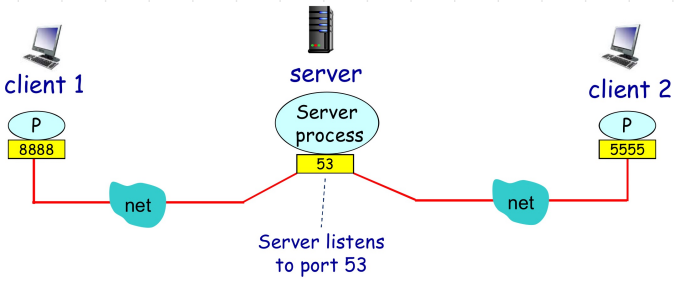
↳ when process wants to communicate w another, we simply need to specify IP + port number, send it into socket w specified transport layer protocol

↳ lets consider two typical transport protocols: TCP & UDP

- transport protocol with no "connection" between client & server (one way)
- client has to explicitly attach destination IP & port in every packet
- receiving server has to extract

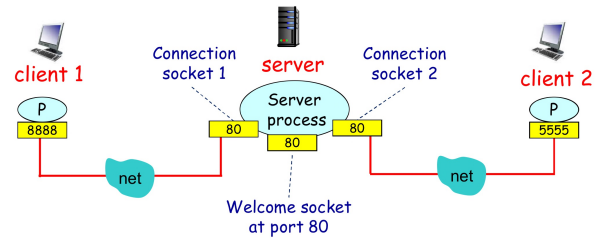


every message is explicit



conceptually like a specific line of communication

- when contacted by client, server TCP creates a new socket for server process to connect w new client
- allows talking w multiple clients individually & stable



conceptually like a butler - welcome port creates new sockets & directs traffic to those new connections

Example: TCP Echo Server

```
from socket import *

serverPort = 2105

serverSocket = socket(AF_INET, SOCK_STREAM)

serverSocket.bind(('', serverPort))

serverSocket.listen()
print('Server is ready to receive message')

connectionSocket, clientAddr = serverSocket.accept()
message = connectionSocket.recv(2048)

connectionSocket.send(message)

connectionSocket.close()
```

TCP socket

listens for incoming TCP request (not available in UDP socket)

returns a new socket to communicate with client socket

details under the hood

Example: UDP Echo Server

```
from socket import *                    ← include Python's socket library

serverPort = 2105                       explicit

# create a socket                       IPv4   UDP socket
serverSocket = socket(AF_INET, SOCK_DGRAM)

# bind socket to local port number 2105
serverSocket.bind(('', serverPort))
print('Server is ready to receive message')

# extract client address from received packet     receive datagram buffer size: 2048B
message, clientAddress = serverSocket.recvfrom(2048)

serverSocket.sendto(message, clientAddress)
                                        respond
serverSocket.close()
```

Example: UDP Echo Client

```
from socket import *

serverName = 'localhost' # client, server on the same host
serverPort = 2105

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input('Enter a message: ')

# send msg to server address            convert message from string to bytes and send it
clientSocket.sendto(message.encode(), (serverName, serverPort))

receivedMsg, serverAddress = clientSocket.recvfrom(2048)

print('from server:', receivedMsg.decode())
                                        convert from bytes to string
clientSocket.close()
```

Example: TCP Echo Client

```
from socket import *

serverName = 'localhost'
serverPort = 2105

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))      ← establish a connection

message = input('Enter a message: ')

clientSocket.send(message.encode())     ← no need to attach server name, port
                                        handled under the hood
receivedMsg = clientSocket.recv(2048)

print('from server:', receivedMsg.decode())

clientSocket.close()
```

- in TCP, processes communicate as if there is a pipe between them, remaining in place until we close it. So long as pipe is established, we don't need to keep specifying IP & port
- UDP, more like one-way messaging, need to attach destination details in every message

1. different transport protocols, different needs when transporting data
2. format for encoding, decoding? might be ambiguous

⇓

3) AL protocols

↳ wrap the format of communication to- and- from socket + choice of transport layer protocol into another protocol

- choice of transport layer protocol
- types of messages exchanged (eg. request, response)
- message syntax & semantics (how to write messages & what they mean)
- rules for when & how to respond

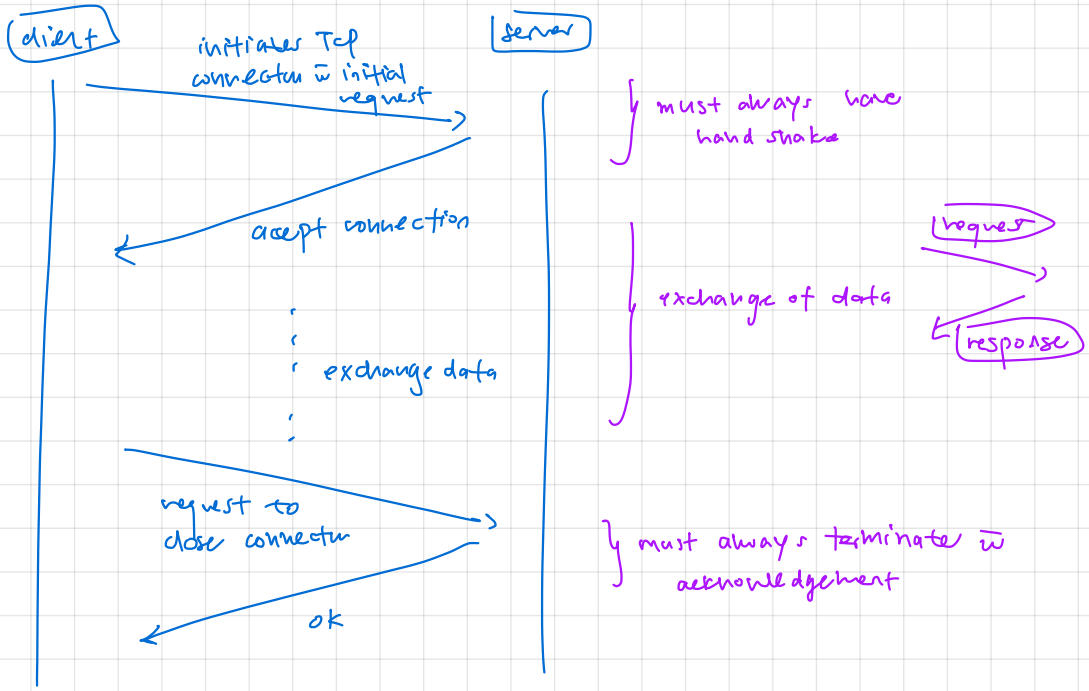
(4) HTTP

1) what is HTTP?

↳ hypertext transfer protocol

↳ based on client-server model & runs over TCP transport protocol

2) how HTTP works

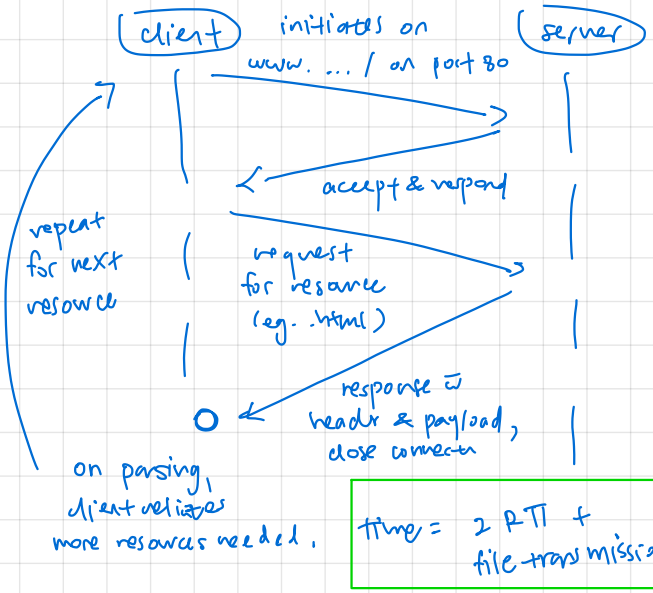


3) persistence of HTTP connections

non-persistent HTTP

↳ handshake, then at most 1 object sent over TCP connection, then connection is closed. Subsequent over diff. connection

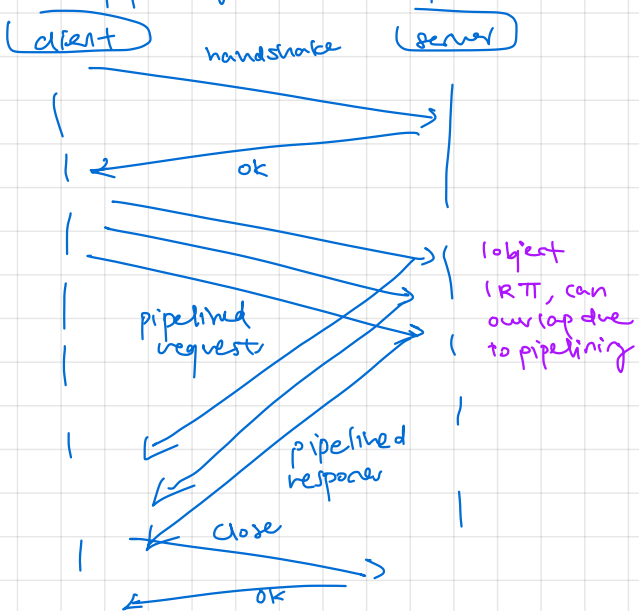
↳ downloading multiple items requires multiple connections



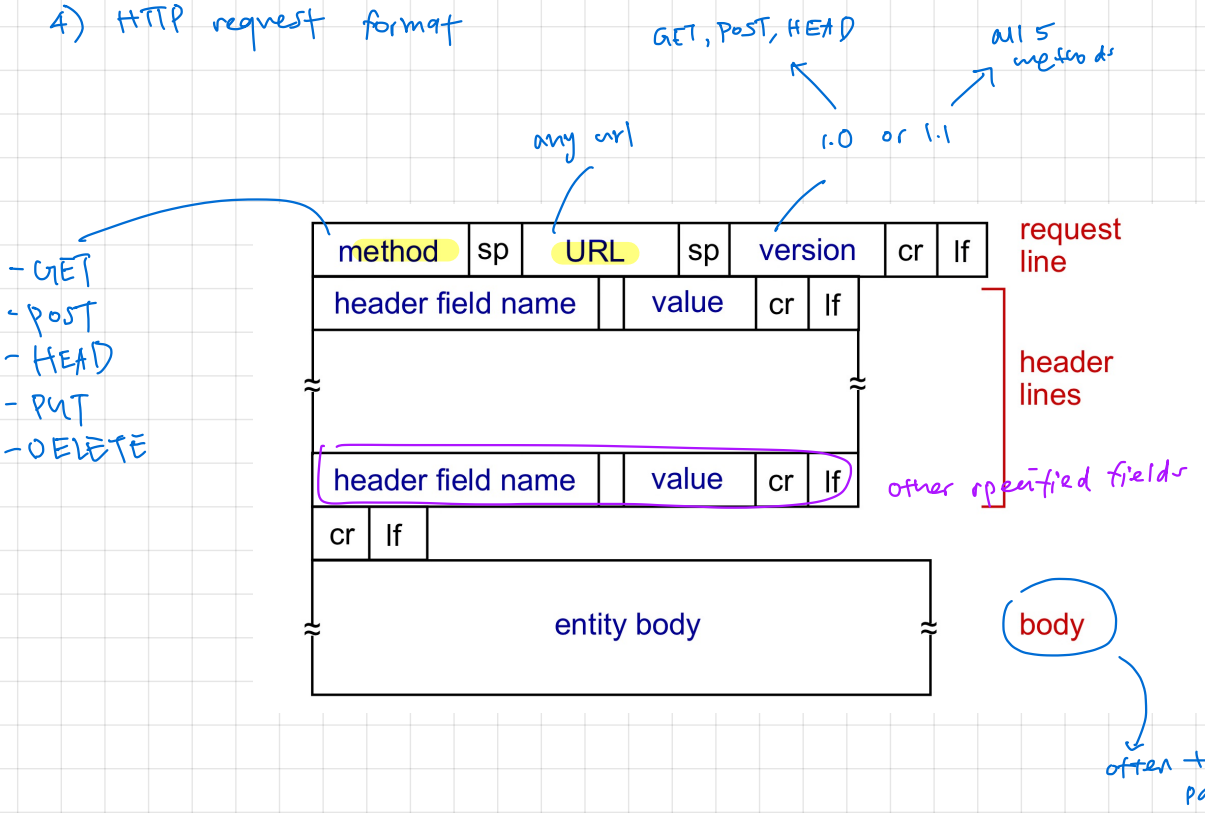
persistent HTTP

↳ handshake, server leaves connection open after sending response. Subsequent messages sent over the same TCP connection

↳ can do pipelining w/ mult. requests

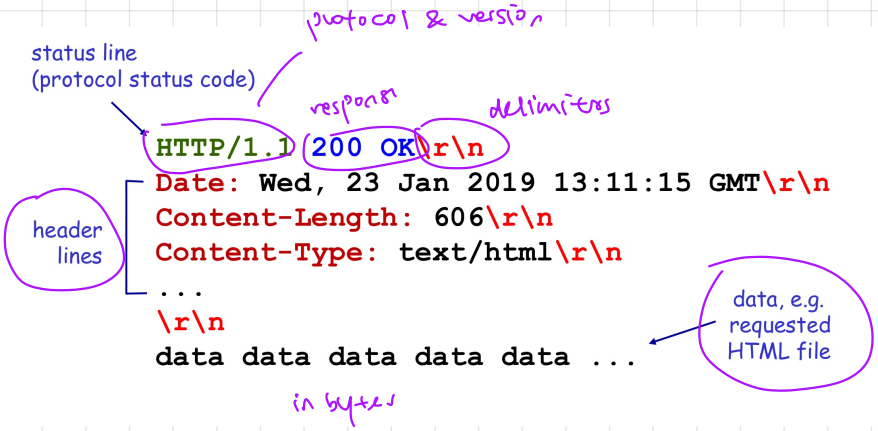


4) HTTP request format



often to store bytes of payload

5) HTTP response format



200 OK
- request succeeded, requested object later in this msg

301 Moved Permanently
- requested object moved, new location specified later in this msg (Location:)

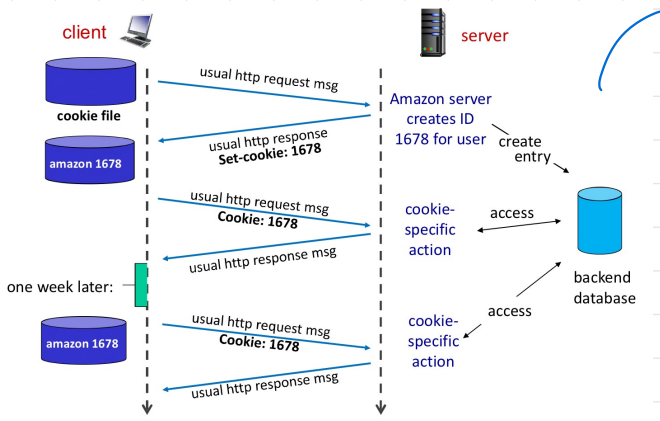
403 Forbidden
- server declines to show the requested webpage

404 Not Found
- requested document not found on this server

For a full list of status codes, check
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

6) cookies

↳ HTTP is designed to be stateless — so we use an identifier (cookies) to maintain state in the form of sender identity



→ how cookies work

- server responds w/ set-cookie : id
- after, client always sends with cookie

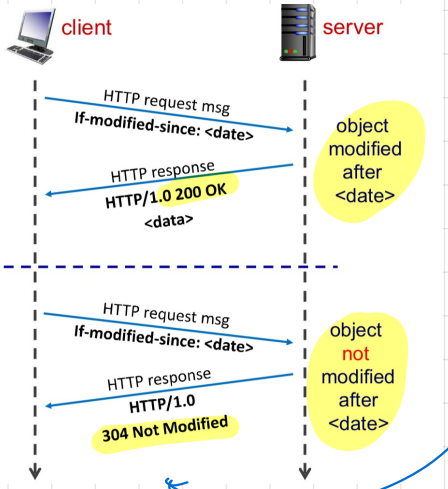
↳ conditional GET with if-modified-since header field

↳ field indicates to use cached value if not modifies since date ⇒ 304 code

- *Goal:* don't send object if (client) cache has up-to-date cached version
- *cache:* specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- *server:* response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



↳ returns w payload only if need to

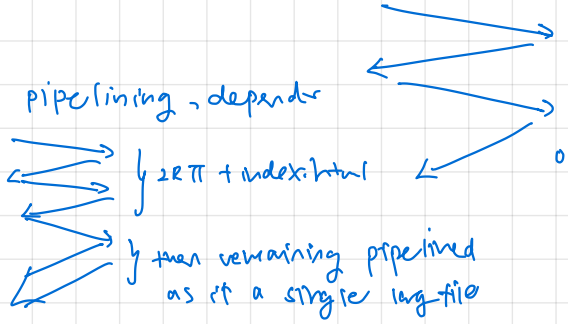
no payload

Notes

1. HTTP always 2 RTT : handshake + actual request for data (minimally) + file transfer

↳ if persistent connection + pipelining, depends

⇒ if pipelining, still need



2. HTTP 1.0 no pipelining. HTTP/1.1 has.

↳ new socket for every resource.

↳ or pipeline. 1 socket.

Transport Layer

① The transport layer

i) responsibilities of the transport layer

↳ the transport layer protocol provides for logical communication between application processes running on different hosts (i.e. from an applications perspective, it is as if the hosts running the processes were directly connected)

sender breaks message into segments, pass into network layer

receiver reassembles segments into message, passes it up to application layer

↳ in a computer network, several transport protocols may be available, with each offering a different service model to applications

ii) multiplexing & demultiplexing

↳ recall that as we go down the protocol stack, we add metadata (headers) as needed

↳ each transport-layer segment also has a set of fields — and at the receiving end, the transport layer examines these fields to identify the receiving socket and then direct the segment to that socket. This is **demultiplexing**

↳ the reverse — encapsulating each data chunk w/ header metadata and passing those into the network layer is **multiplexing**

---

② UDP : user datagram protocol

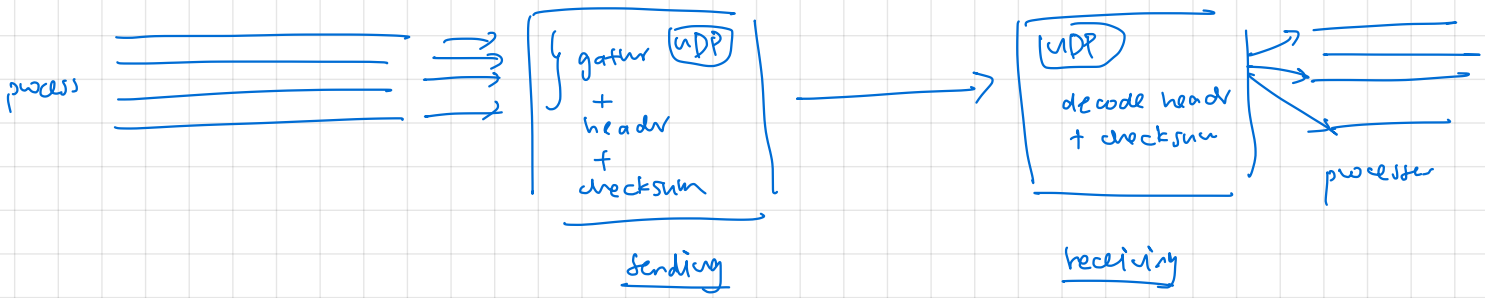
i) how UDP works

↳ suppose you wanted to design a no-frills, bare bones transport protocol. You would minimally need to provide multiplexing & demultiplexing to the application layer above.

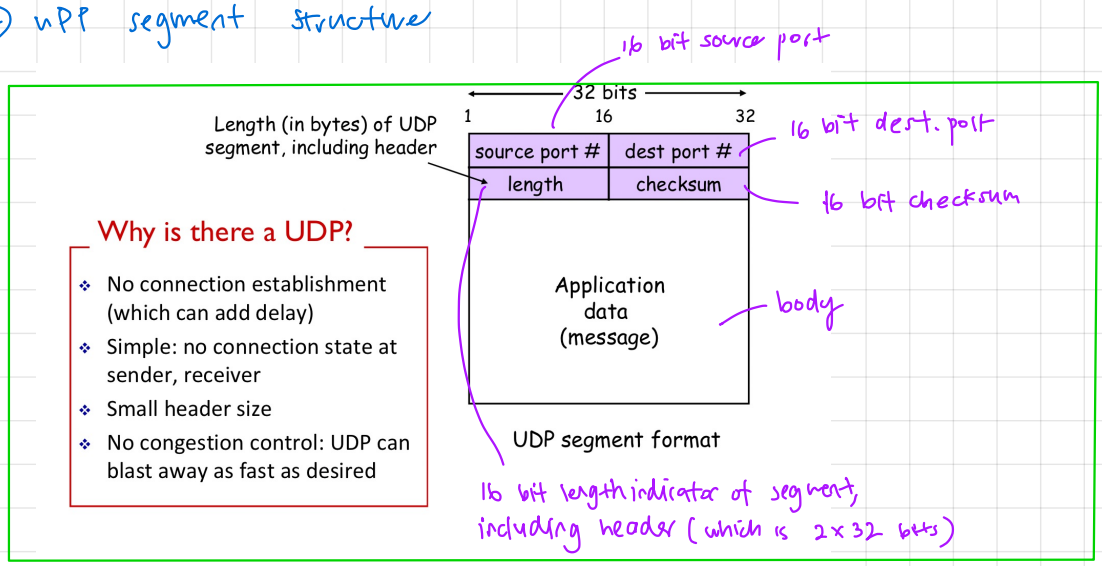
⇒ UDP is that:

- multiplexing : gathers data from processes, forms packets w/ headers, sends to network layer
- demultiplexing : receives packets from network layer and dispatches them to the right processes for access
- checksum : basic bit-error checking

↳ UDP is connectionless as there is no handshaking between sending & receiving transport layer entities



2) UDP segment structure



Why is there a UDP?

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

3) UDP checksum

↳ goal: detect bit flip errors

↳ how: pass all data through a deterministic algorithm, see if it matches

⇒ all 16-bit fields in UDP are added (with any most significant bits carried over back to the 0th bit). After 3 fields (source #, dest #, length) are added, all bits are flipped. 16 bit checksum.

1. Add source + dest, carrying if needed
2. Add result to length, carrying if needed
3. Flip bits (1s complement)

→ if fail, nothing is done to recover from error. Segment simply discarded.

4) notes about UDP

↳ UDP is very barebones but also very lightweight due to small header size, no congestion control, no need for handshake ⟶ used in DNS lookups

↳ UDP is unreliable

→ best-effort delivery; no guarantee of arrival at destination

→ checksum is not a 1-1 mapping

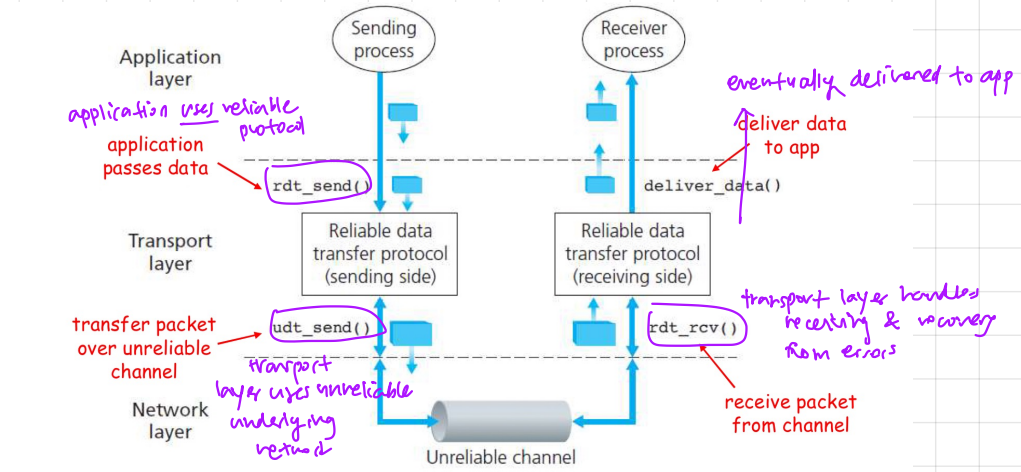
③ Techniques for reliable data transfer

1) problems with data transfer

↳ the underlying network can lead to → corrupt packets (bit flips)
→ dropped packets (full queues)
→ reordered packets
→ deliver packets after an arbitrarily long delay

2) reliable data transfer protocols : protective interface

↳ we want the transport layer to be a layer of abstraction over the unreliable network and provide services for error detection & recovery

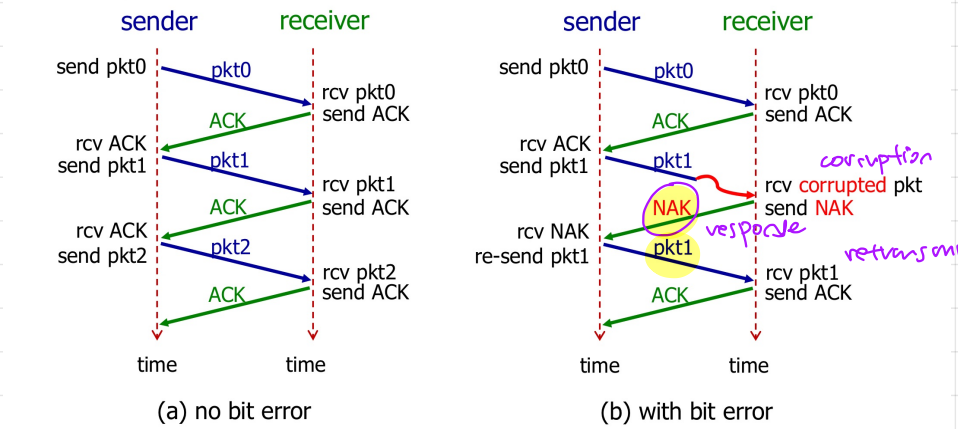


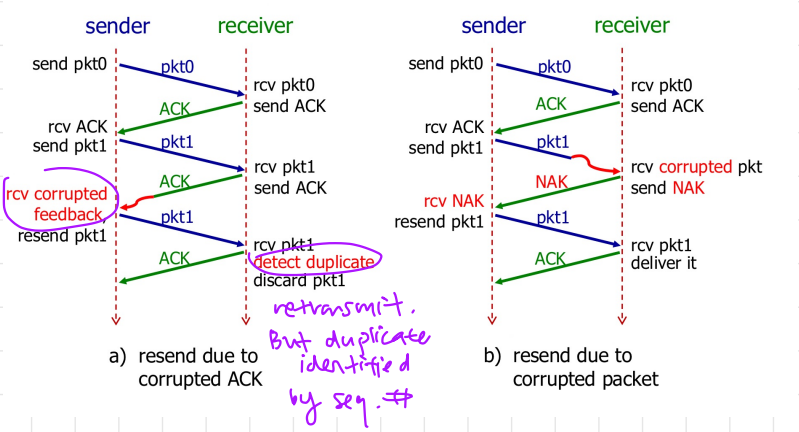
3) detecting & recovering from bit errors : checksum

↳ simple checksum like that we saw in UDP can be used to detect bit errors in transmission ⇒ but what to do after ?

↳ receiver should reply w ACK, retransmit if NACK

↳ but ACK itself can be corrupted ! ⇒ we identify packets by a sequence no. and retransmit if garbled or NACK. If receiver already seen sequence no., discard



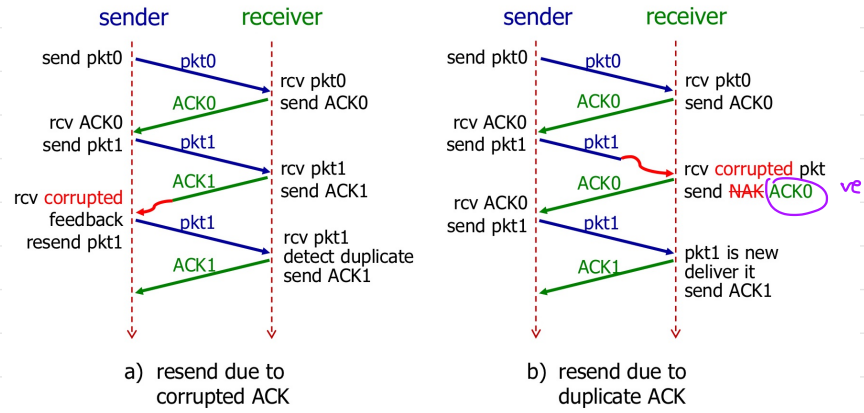


garbled feedback

retransmit, but duplicate identified by seq. #

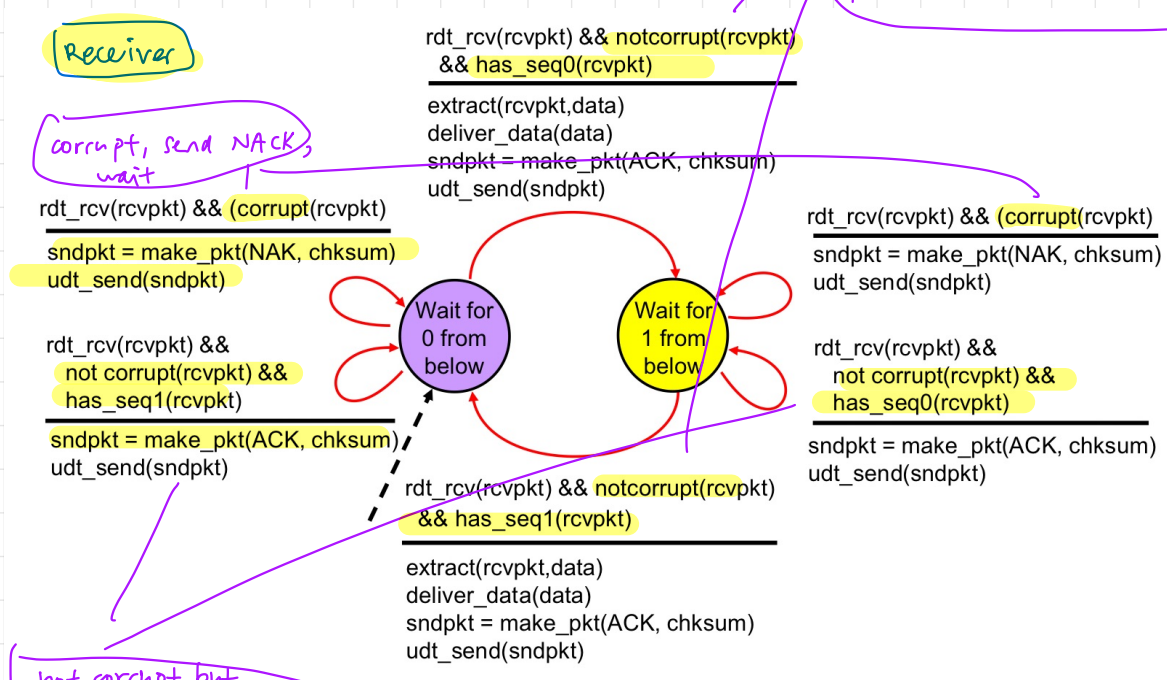
to simplify, instead of ACK & NACK, we can simply have receiver always send ACK + seq. no. of last ok packet

⇒ save amt. of info, unified format, allows sender to know exactly what to retransmit

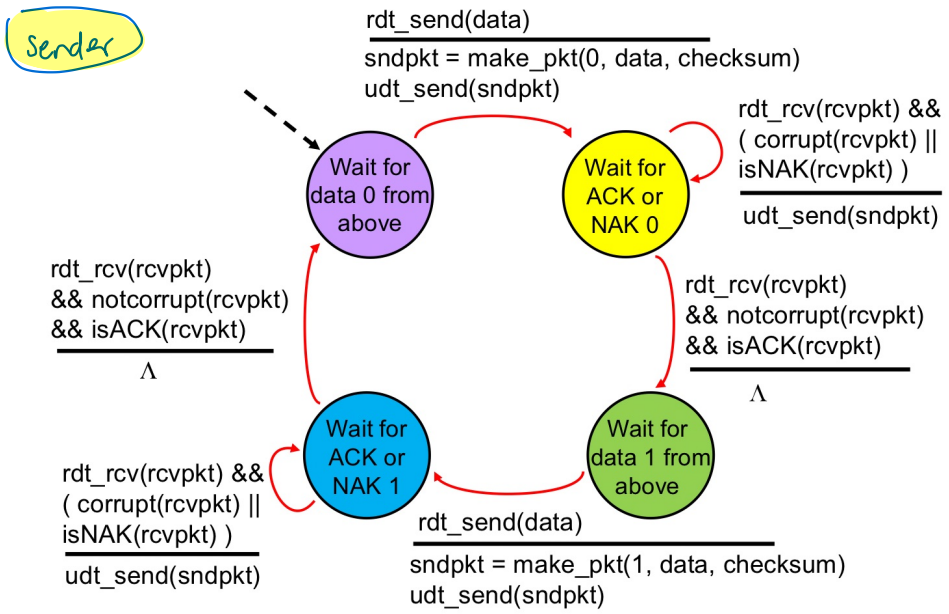


reply w last good seq. no.

not corrupt, correct expectant packet, change expectant packet no. & deliver data up



not corrupt, but duplicate. discard and wait, respond w ACK + seq. no.

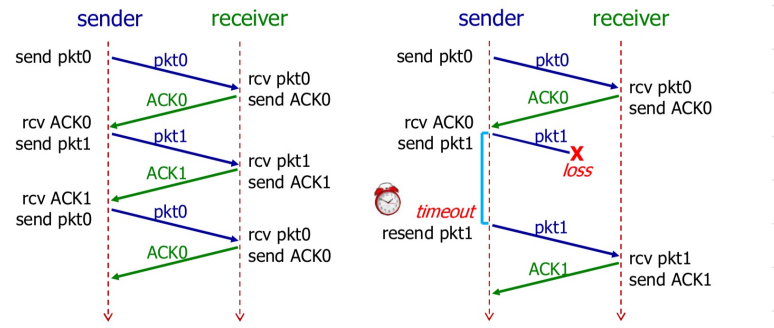


4) ensuring arrival : timeouts

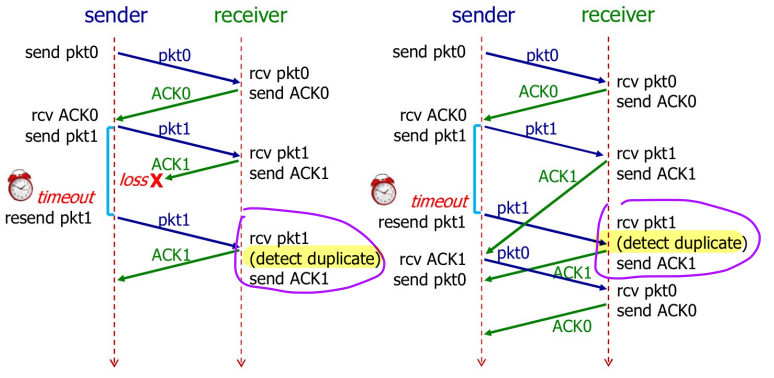
↳ to server, whether a packet is lost (didn't reach client), response is last cACK lost) or either are delayed is the same

⇒ we retransmit after some timeout

⇒ we use seq. number to identify duplicates / what is expected next



lost in sending. timeout. send pkt1 since last ACK was 0.



ACK1 lost, resend packet1 since last ACK was 0.
⇒ duplicate detected

timeout. resend packet 1 since last ACK @ retransmission is 0.
eventually ACK 1.
- when repeat arrives, duplicate detected.

(4) pipelined protocols

1) motivation

↳ easy to have reliable data transfer w stop & wait protocol (ping-pong)
↳ but very low hardware utilisation — mostly idle time waiting for response

↳ stop & wait follows pattern of

transmission + 2RTT + data + transmission + 2RTT + data ...
(transmit) (wait) (transmit) (wait)

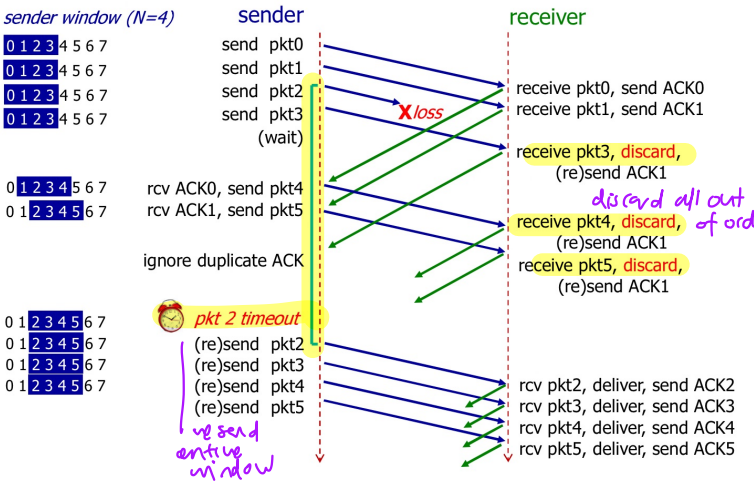
↳ actual hardware utilisation

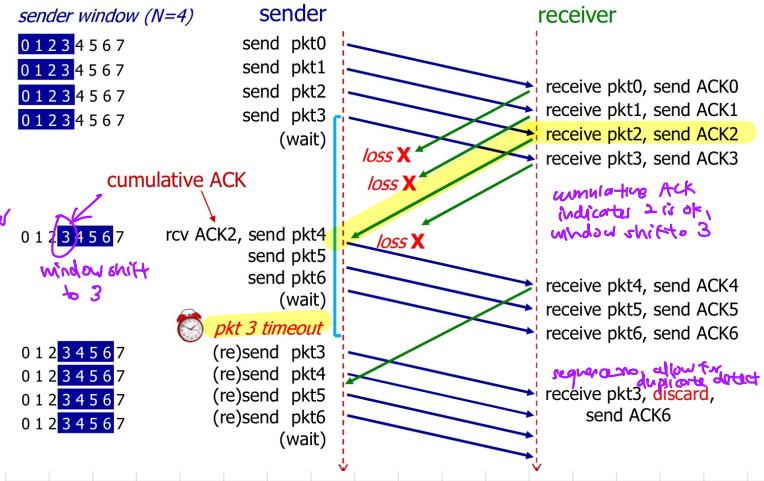
if 2RTT + data >> transmission (typical), low utilisation!

pipelinined protocols sending multiple "inflight", yet-to-ACK packets

↳ buffering/queuing on receive & sender needed
↳ seq # need to be > 0/1 since need to identify

2) **go-back-N**





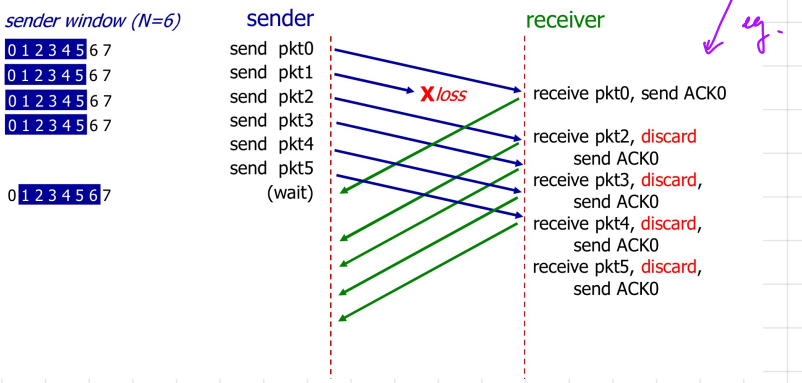
↳ key idea: we track a window of N packets we will send at a time, where the lowest tracked in the window is the lowest non-ACK-ed packet. If the lowest timeouts / any are corrupted we resend the whole worth of packets.

- k bit sequence no. in packet header } sender
- timer for oldest un-ACK-ed packet
- incrementing ACK every time allows cumulative ACK ⇒ last good packet } receiver
- discard out of order packets and ACK the last in sequence

if ACK 0 lost but ACK 1 received, sender knows all up to 1 received, no need retransmit 0

⇒ ie. receiver ignores everything except the one it's expecting

↳ window retransmission + discarding out of order wasted, inefficient



eg.

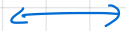
GBN must respond to :

- invocation from above .
  1. check if window is full
  2. if not full, create packet, insert
- receipt of ACK
  1. sequence no. in Ack indicates cumulative acknowledgment
  2. update window & shift
- timeout
  1. resend all in window
  2. if window is empty, stop (terminates)

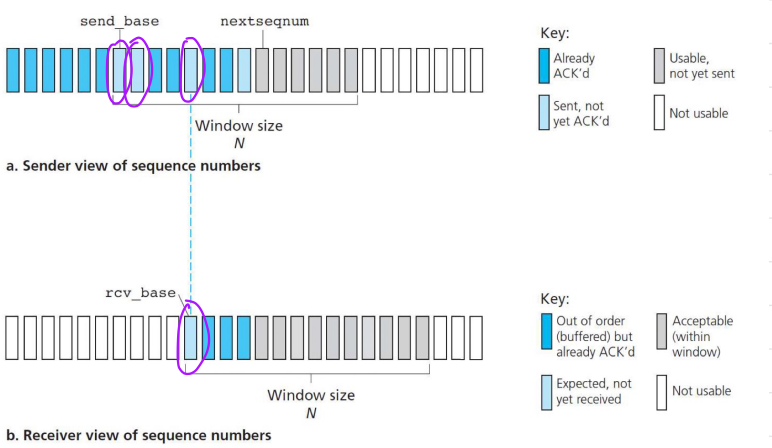
3) selective repeat

↳ key idea : a more refined version of go-back-N. instead of tracking up to last un-acked packet and resend window + receiver discarding out of order, we use a hash set to track

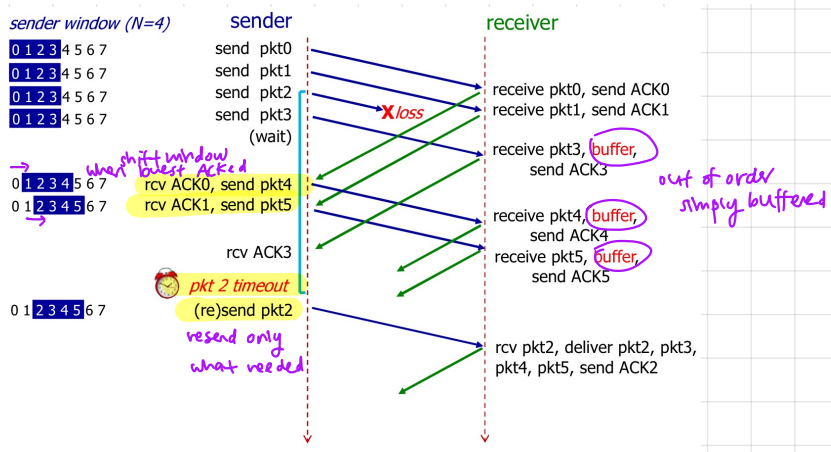
sender tracks what has been ACKed, retransmitting (only) un-ACKed when timeout



receiver tracks what has been received, creating gaps as needed & sending ACK corr. to sender to track



both views are needed to understand what is going on

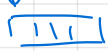


sender

- data received from above
  - ↳ check next available seq_number. If within window (window not full), packet & put into window
- timeout (for each in window)
  - ↳ retransmit specific if not ACK-ed
- ACK received
  - ↳ mark packet in window. If is lowest, shift window.
  - ↳ if window shifted & now untransmitted in window, transmit those

receiver

- packet received
  - ↳ check seq. no. if not previously received → buffered.
  - ↳ if packet is at base of window, it and all later buffered, contiguous packets sent up
  - ↳ send ACK



---

⑤ TCP

1) Key features of TCP

- point-to-point
  - one sender, one receiver
- connection-oriented
  - handshake required
- duplex service
  - bidirectional data flow in same connection
  - client can be client & either can be server @ the same time
- reliable, in-order byte stream
  - guaranteed delivery in order

TCP socket

↳ a TCP socket connection is identified by (source IP, source Port, dest. IP, dest. port)

A receiver uses all four values to direct the incoming @ the welcome port to the appropriate actual socket

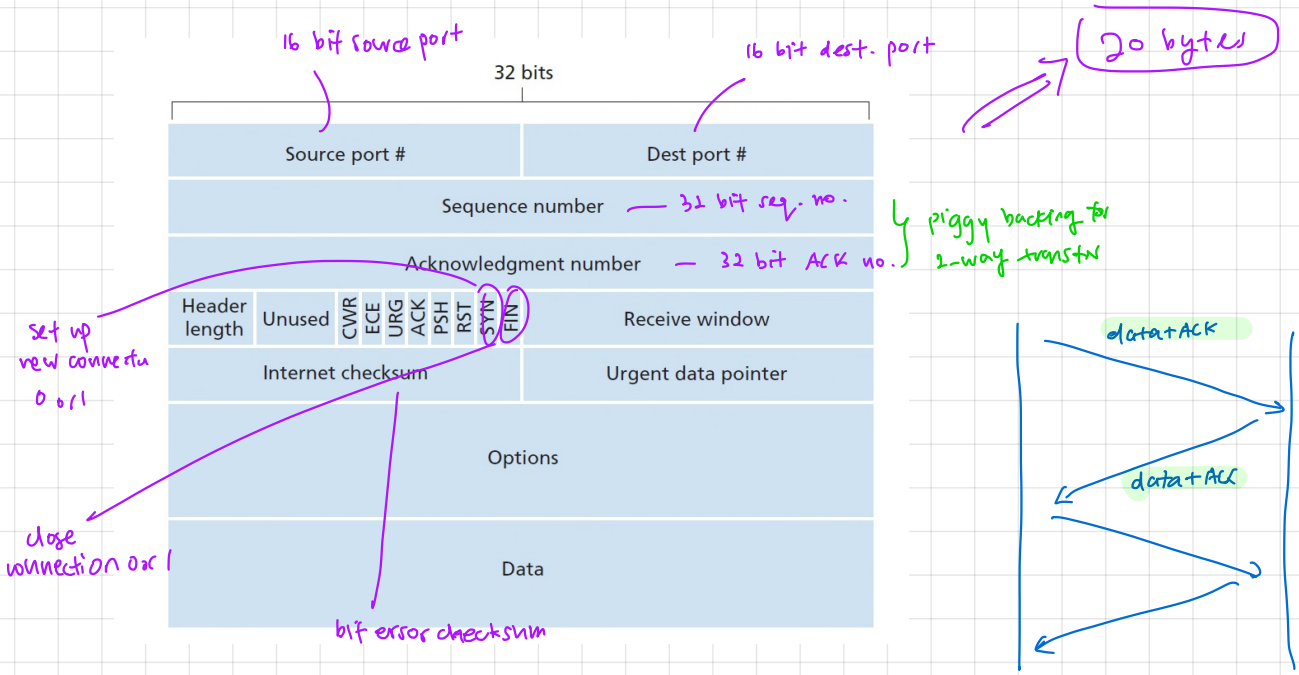
TCP buffers

↳ two buffers created after handshaking @ either side (receive & send)

2) TCP segments

↳ maximum segment (body) size typically 1460 bytes

↳ then headers added on before transmit, 20 bytes



sequence and ACK — TCP uses a variant of (selective repeat) but w cumulative ack.

↳ in TCP, seq. no. indicates first byte of data in a segment. Is not the datagram no.

↳ ACK no. indicates first byte of next segment to be received.

seq. sent	data size	ACK no.
1000	1000	2000
2000	1000	3000
⋮	⋮	⋮

} technically cumulative ACK but can also use as specific ACK

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	3 4 5
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments	3 4 5
Arrival of out-of-order segment higher-than-expect seq. # (gap detected)	Immediately send *duplicate ACK*, indicating seq. # of next expected byte	3 4 5
Arrival of segment that partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap	3 4 5

} ACK only after some time, so we can use cumulative & reduce load

} use of cumulative ACK

} out of order is cached, immediate reply to request expected

} segment fills gap, also immediate reply

immediate reply when out of order, since cannot gain from cumulative ACK

3) dynamically updating timeout

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

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT → exponential average of past RTT's

(typical value of α : 0.125)

DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT| → calculate st. dev of estimated vs. historical actual

(typical value of β : 0.25)

TimeoutInterval = EstimatedRTT + 4*DevRTT

"safety margin"

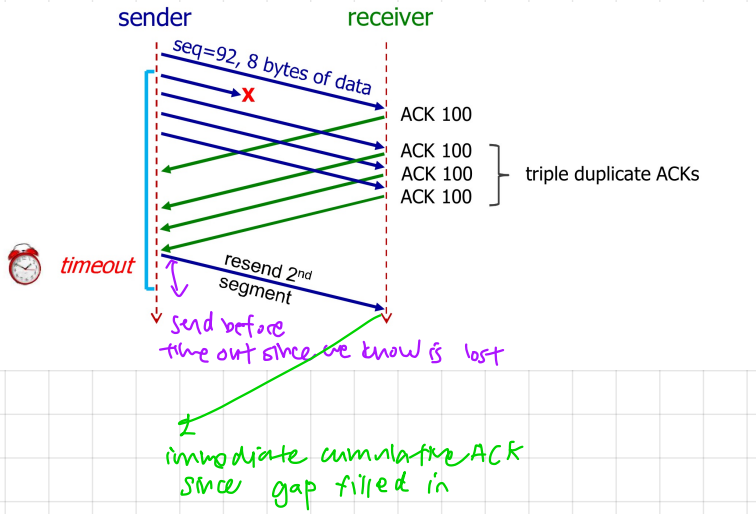
use variance as safety margin

4) fast retransmission

↳ recall that if out of order received, immediate ACK transmitted

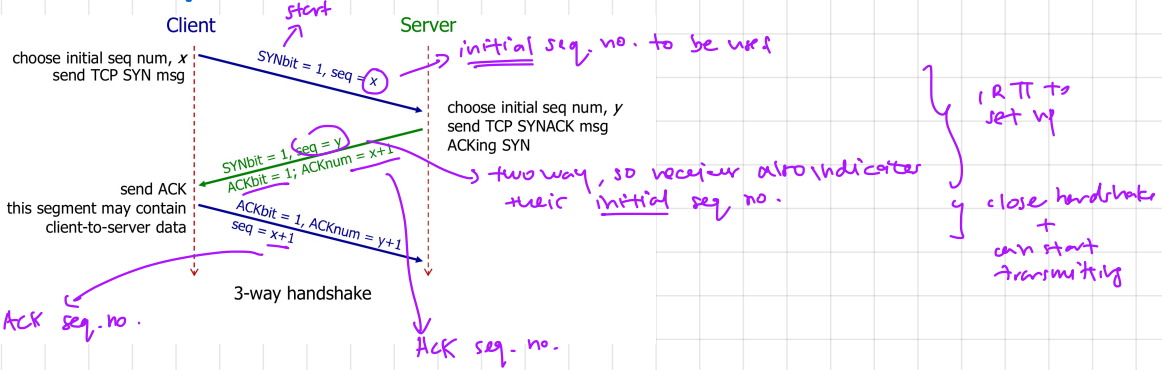
⇒ consecutive ACKs of same ACK no. indicate packet did not reach

↳ for TCP, on 4 ACKs w same no., that packet is resent even if timeout has not expired

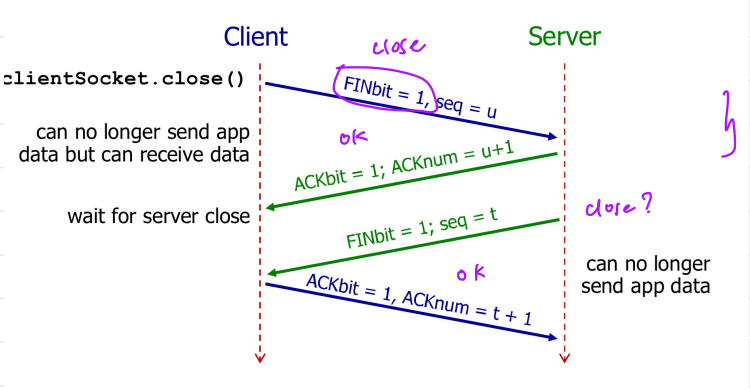


5) establishing & closing connection

establishing need to agree on connection params



Closing



} indicate want to close

} confirm closure

6) how much data can TCP carry?

↳ recall — 2^{32} bit seq. no., and each segment seq. no. grows by payload bytes ⇒ exactly 2^{32} bits can be transmitted over a single TCP

↳ how many total bytes are transmitted?

↳ no. of packets × header size + 2^{32}

because header overhead is packet-specific

7) ensuring seq. no. is not reused

↳ TCP uses 32 bit seq. no, which makes it unlikely

↳ IP protocol specifies TTL field in packet header, which decreases by 1 everytime it is redirected. Discard when hits 0 ⇒ packet cannot live forever

⇒ typically about ~3 min for packet to leave network, so after that ok to use

Notes

1. MSS typically refers to payload but can mean whole

2. TCP ACK is first byte of next — notion of "sequence no." isn't quite precise

3. TCP uses GBN - SR hybrid — one timer but send back one @ a time + immediate ACK

4. Stop-and wait 

5. HTTP / TCP 3 way :  1.5 RTT then ready to start reading data + close

6. Is complement checksum used in both UDP & TCP

7. pipelined window size: $\text{utilization rate} = \frac{\text{window} \times \text{transmission}}{\text{transmission} + \text{RTT}}$

Network layer

① Responsibilities of the network layer

↳ recall : the application layer handles application-level communication between applications. transport layer is a layer below that, handling interactions between machines

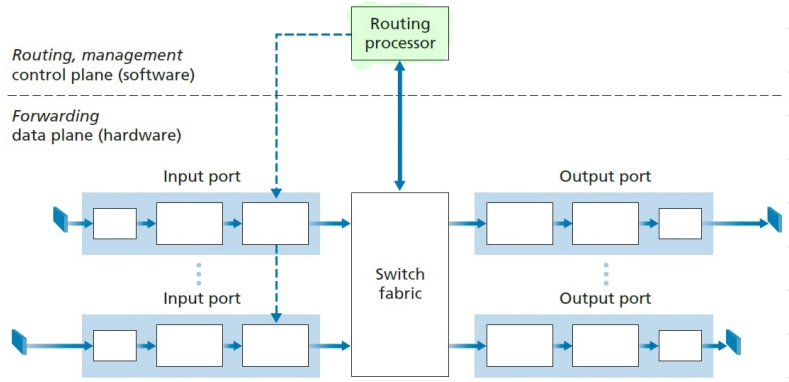
→ the network layer is a set of protocols determining how datagrams move through the network to reach their destination

forwarding action at the router/node level. when a packet arrives at an input link, the router must move the packet to an appropriate output link, so that it will arrive @ its destination

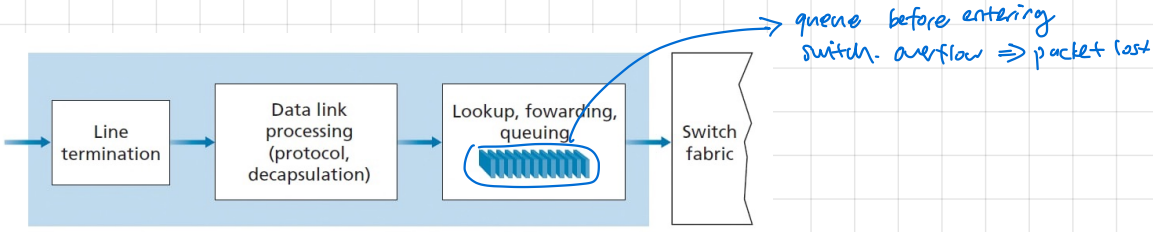
routing behavior/coordination at the network level so that packets reach their destination

---

② routers

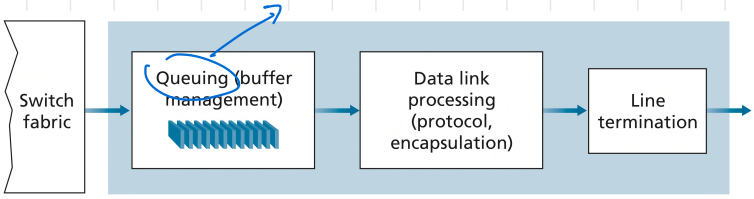


input ports performs the physical-layer function of terminating an incoming physical link and link-layer functions to translate the datagram back up to the network layer. Then the last step is to do some preprocessing before entering the switch fabric for direction to the output port.



output ports inverse of the input port, handles conversion down and sends it out.

queue before sending. Full ⇒ packet lost

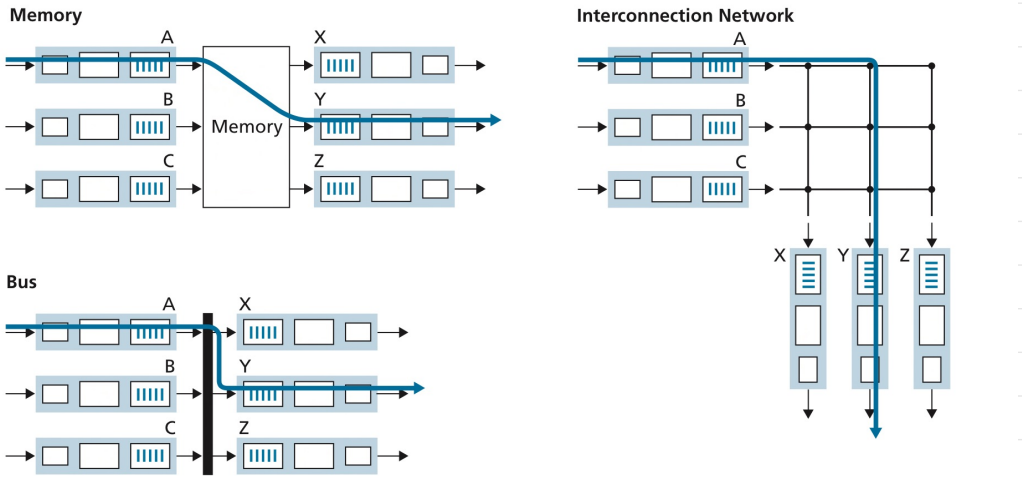


switching fabric connects the router's input ports to its output ports.

1. switching in memory: routing via a CPU. ports are I/O, read to RAM, compute.

2. switching via a bus: input → output via a shared bus, without CPU intervention. All output ports receive the input, but only matching port keeps it. only one packet can cross bus at a time → need for input queue

3. switching via interconnection network: essentially a bigger bus that allows more signals to move.



routing processor Executes routing protocols, maintains routing tables, computer forwarding table. In general, handles computation for forwarding & transmission, and dependent on network-layer protocol. (eg. packet scheduling)

---

② IP addressing

i) IP addresses

↳ IP addresses are used to identify a host/router (some device) — specifically, network interfaces. PC has 2 (ethernet, wifi), routers typically have multiple

special 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.

→ same device (local host)

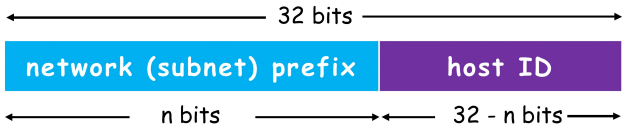
→ use behind NAT. Different so that devices can identify local vs. global devices

→ used in DHCP before assignment

2) IP addresses for heirarchical addressing

↳ recall that IP addresses uniquely identify devices, and that the internet is a network of networks

An IP address logically comprises two parts:



IP identifies host within a specific subnet (heirarchically & recursively)

↳ to traverse the net, routers need to know which subnets an IP address belongs to, which can be identified by a subnet mask

Classless Inter-domain routing — CIDR: the internet's IP address assignment strategy. We have a subnet prefix of arbitrary length.

eg. a.b.c.d/x — x = no. of bits of x determining the subnet } same prefix, same subnet.

subnet mask — 1's and 0's indicating which bits belong (1) to prefix and which don't (0)

- 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			

routers use the IP addresses heirarchically to map packets to other routers in the correct subnet, thus traversing the network

3) subnet resolution

↳ if 2 prefixes, but 1 is superset of other, and could match multiple subnets → ambiguity ⇒ we map to the most specific subnet

- Packet with destination IP 200.23.20.2 ⇨ R1
  - (Binary: 11001000 00010111 00010100 00000010)
- Packet with destination IP 200.23.19.3 ⇨ R2
  - (Binary: 11001000 00010111 00010011 00000011)

both match R1, but for second one R2 is the more specific match

match the longest prefix

Forwarding Table at R3

Net mask	Net mask in binary	Next hop
200.23.16.0/20	11001000 00010111 00010000 00000000	R1
200.23.18.0/23	11001000 00010111 00010010 00000000	R2
199.31.0.0/16	11000111 00011111 00000000 00000000	R2
...		...

4) IP address allocation (DHCP, ISP)

↳ visualize allocation at different hierarchies.

1. ICANN allocates addresses, manages DNS, resolves disputes
2. ISP's get a block of addresses (ie. ü some prefix to identify its subnet) from ICANN
3. organisations then buy a block from ISPs (more specific prefix to identify their subnet within ISP subnet)
4. organisations (as small as home router — 1 address or company — many) then give their end devices IP addresses they've been allocated by

manual administration

using a dynamic Host configuration Protocol server

DHCP

- DHCP allows a 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.

- 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)
- DHCP runs over UDP
  - DHCP server port number: 67
  - DHCP client port number: 68

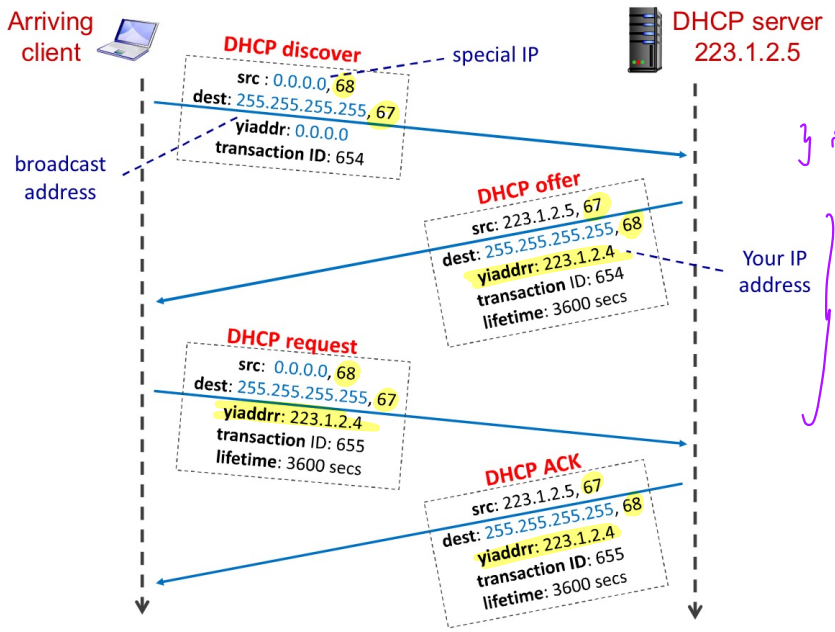
- DHCP: 4-step process:
  1) Host broadcasts "DHCP discover" message
  2) DHCP server responds with "DHCP offer" message
  3) Host requests IP address: "DHCP request" message
  4) DHCP server sends address: "DHCP ACK" message

reserved source IP to agreed broadcast @ 255.255.255.255, 67

reply on broadcast IP

client makes handshake offer on broadcast

DHCP server replies on assigned IP



3 initial request

offer, response, ok 3 way handshake

all on public channel 255.255.255.255 over UDP, identified by transaction ID

③ Network address translation

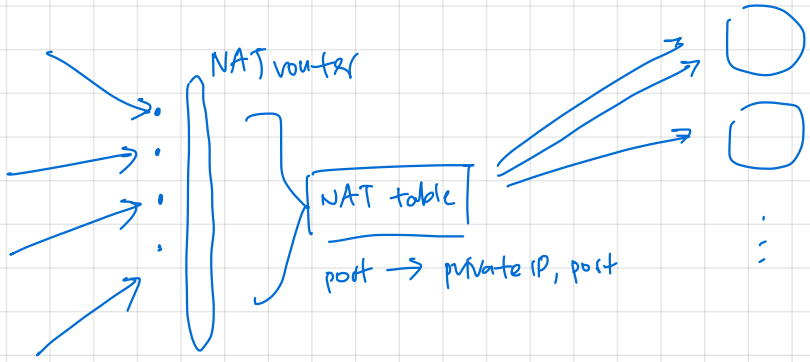
1) What is NAT

motivation IP addresses have a limited number (eg. IPv4 → 32 bits → 2^{32}). If we uniquely address every device, we will run out of space.

key idea we group devices into a single virtual device, and within that group, we use private IP addresses (not used anywhere else, though).

↳ NAT table translates virtual port no. → private IP, port no.

↳ private IP is never used outside, since otherwise we would be unable to distinguish between inside & outside network traffic



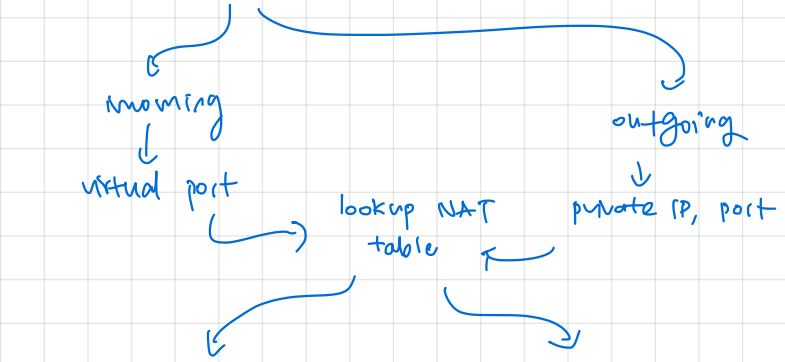
16 bit port no., 1 IP address ⇒ 2^{16} things it can address

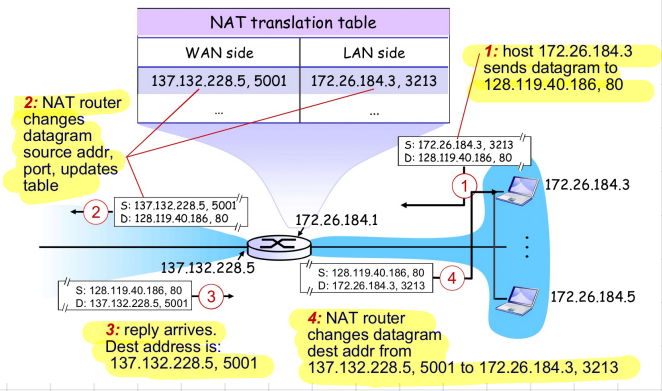
2) How do we get the mapping?

↳ DHCP

↳ just as how public devices get IP from ISP's via DHCP, devices do the same — the NAT router runs a DHCP server.

3) NAT process





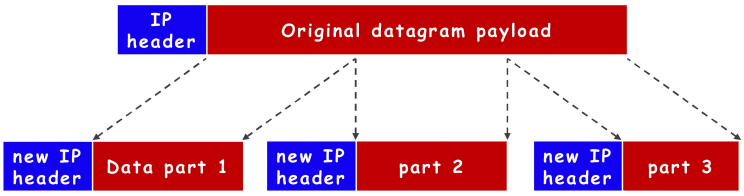
④ IPV4

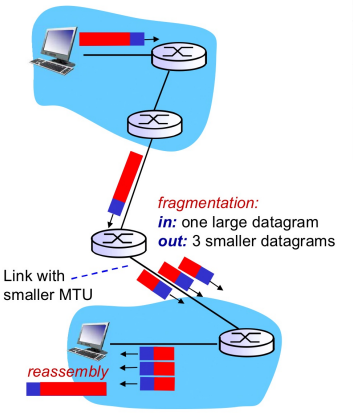
1) data fragmentation

↳ different physical links may have different maximum transfer unit (MTU)

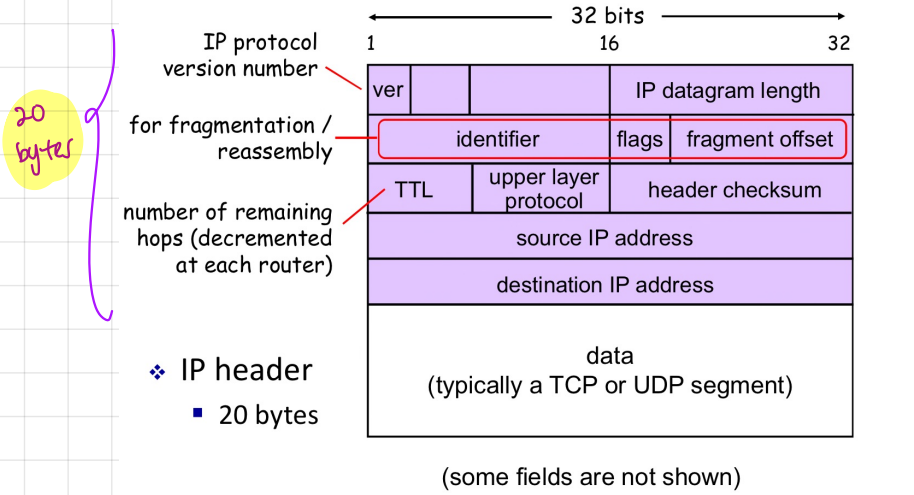
↳ "too large" IP datagrams may be fragmented by routers

↗ the maximum amount of data a link layer frame can carry





2) datagram format



3) reassembly after fragmentation

take note !

- Flag (frag flag) is set to
  - 1 if there is next fragment from the same segment.
  - 0 if this is the last fragment.
- Offset is in expressed in unit of 8-bytes.
- Example
  - 20 bytes of IP header
  - 1,200 byte IP datagram
  - MTU = 500 bytes

offset = 480/8

length = 1200	ID = x	flag = 0	offset = 0	…

one large datagram divided into 3 smaller datagrams

length = 500	ID = x	flag = 1	offset = 0	…
length = 500	ID = x	flag = 1	offset = 60	…
length = 240	ID = x	flag = 0	offset = 120	…

carry 480 bytes of data

			length
identifier		flags	offset
source IP address			
destination IP address			

in data fragmentation, we use the flag to indicate if there is another segment, and offset to determine location in the datagram.

note that IP header is 20 bytes → we can use length, flag, offset to rebuild the original data gram.

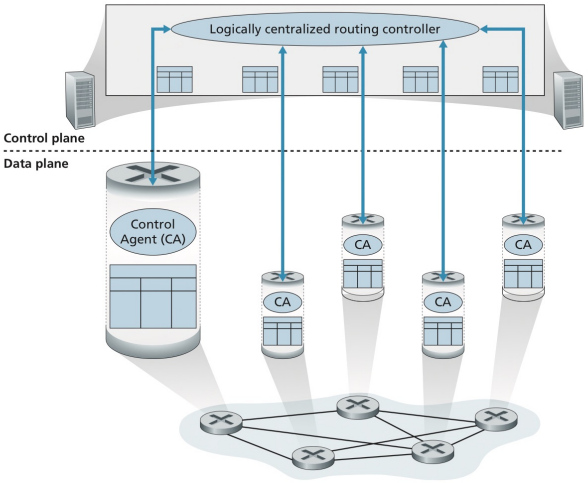
⑤ Routing algorithms

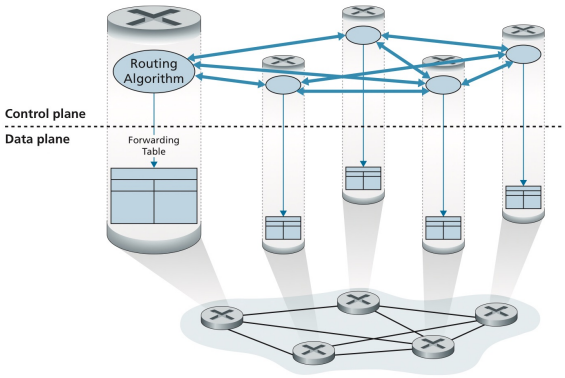
1) routing and types of routing algorithms

↳ routers as nodes, links as edges

centralised routing algorithm
- compute least cost path using complete, global knowledge of the network
  eg. Dijkstra's
- computed by central node

decentralised routing algorithms
- calculation of least cost path done in an iterative, distributed manner
- no node has complete information
- through iterative information exchange, a node gradually learns the least cost path to direct a datagram to





2) the distance vector algorithm

↳ distributed: each node receives some information from one or more of its neighbours, does a calculation, and distributes its results back to its neighbours

↳ iterative: process continues until no more information is exchanged (self-terminating)

↳ asynchronous: does not require all nodes to operate in lockstep with one another

Distance-Vector (DV) Algorithm

At each node, x:

```
1  Initialization:
2    for all destinations y in N:
3       D_x(y) = c(x,y) /* if y is not a neighbor then c(x,y) = ∞ */
4    for each neighbor w
5       D_w(y) = ? for all destinations y in N
6    for each neighbor w
7       send distance vector  D_x = [D_x(y): y in N] to w
8
9  loop
10    wait  (until I see a link cost change to some neighbor w or
11             until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       D_x(y) = min_v{c(x,v) + D_v(y)}  // all neigbours v of x , Bellman ford
15
16 if D_x(y) changed for any destination y
17       send distance vector D_x = [D_x(y): y in N] to all neighbors
18
19 forever
```

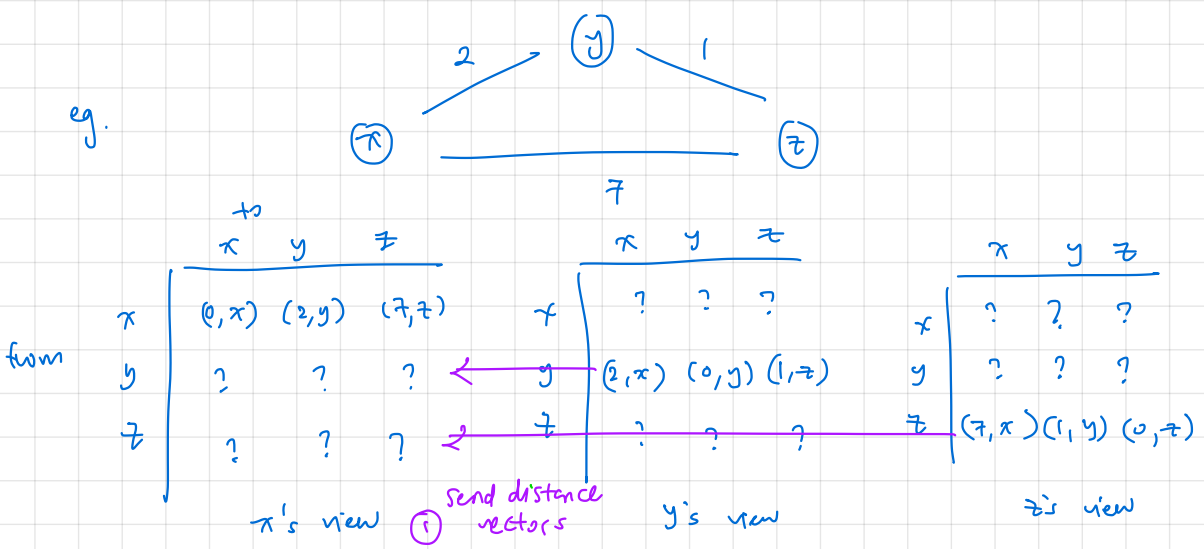
intuition The DV algorithm maintains a set of vectors: D_x, which holds the least cost path to y from x (this), $\{D_v\}_{v=0\ldots N}$, which holds all the least cost paths from neigbours to y, and $\{c(x,v)\}$, which holds the cost to neigbours.

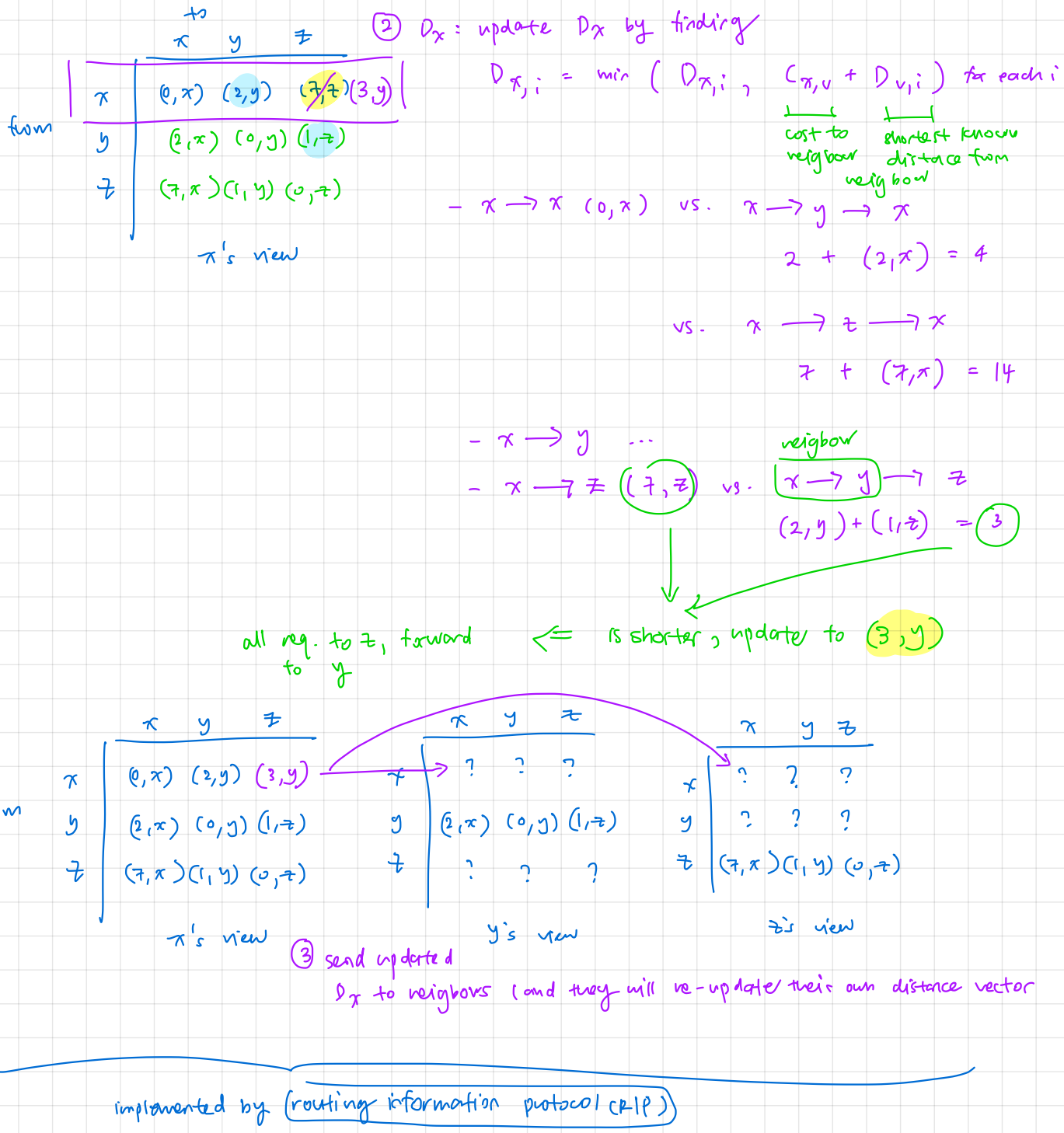
At initialisation, all neigbours have cost $c(x,v)$, all other distances initialised to ∞.

Whenever our D_x changes (including @ initialisation), we have new knowledge, so we propagate it outwards. Likewise, when we receive a distance vector, we use bellman-ford equation to update our own distance vector D_x if there is a difference

⇒ then when routing, we route using these distances (think of it as storing a (distance, to-who) pair.

eg.





implemented by (routing information protocol (RIP))

RIP

- RIP (Routing Information Protocol) implements the DV algorithm. It uses hop count as the cost metric (i.e., insensitive to network congestion).
- Exchange routing table every 30 seconds over UDP port 520.
- "Self-repair": if no update from a neighbour router for 3 minutes, assume neighbour has failed.

(6) Internet control message protocol (ICMP)

1) ICMP used by hosts & routers to communicate network-level information (e.g. error reporting, basic responses)

2) just above IP

   a) ICMP is carried by IP datagrams (so technically above network level), and the ICMP header (part of payload) starts after IP header

ICMP Type and Code

- ICMP 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

Selected ICMP Type and subtype (Code)

Notes

Link layer

① Responsibilities of the link layer

1) the link layer & how it fits in

↳ application layer : protocol between applications. Transport layer : protocol between machines(os)
Network layer : protocol to traverse the network (eg. IP & routing)

↳ link layer : protocols (ie. set of behaviors, formats) to move a datagram from one node to an adjacent one over a communication link

Some services provided by link layer protocols:

Framing — Link layer transmissions require their own metadata. A link-layer frame encapsulates a network-layer data gram.

Link access protocols — Depending on the type of link (point-to-point) vs. broadcast, protocols / conventions need to be defined for how and when transmission takes place.

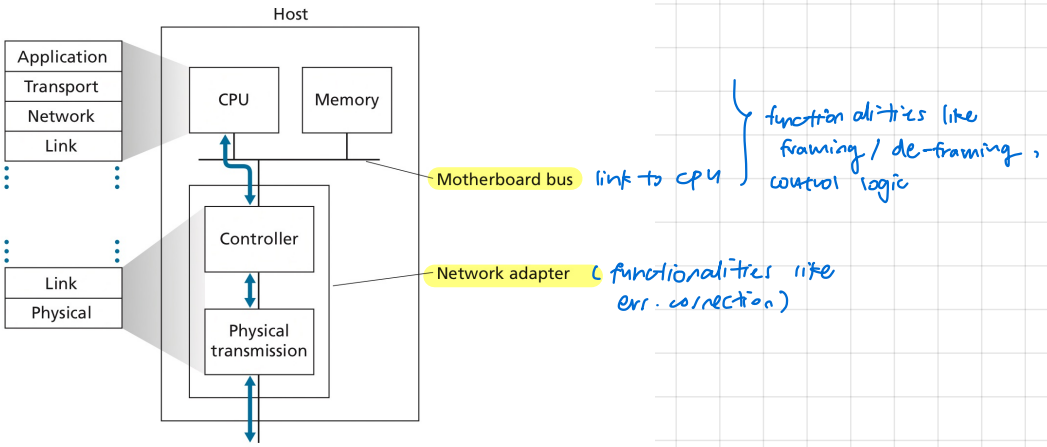
Reliable delivery — Done via error-correction codes or retransmissions. Typically found in unreliable media (eg. wireless) but not reliable (eg. wired)

error detection & correction — Bit errors can be introduced via signal attenuation and electromagnetic noise. Nodes typically include error detection bits in the frame, and receiving nodes do a check (no point forwarding corrupted). Error correction also sometimes supported.

Typically implemented in hardware.

2) where is the link layer implemented?

↳ implemented in both software & hardware, but primarily in hardware in the network adapter / network interface component



② network links

1) types of communication links

↳ the link layer is really about the intricacies of how to get info from A → B, as determined by the network layer.

↳ Ideally, all nodes are fully connected graph ⇒ then we just execute at the network layer. But this is expensive & challenging.

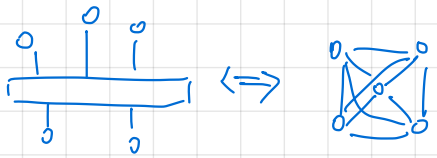
⇒ instead, we devise ways to connect devices so that they behave as if they are fully connected to higher layers.

broadcast network

↳ nodes share a spine, a broadcast channel

↳ mental model: every host broadcasts (shouts) to all hosts, they coordinate how to shout

↳ since many hosts are connected to the same broadcast channel, an access-protocol is needed to coordinate frame transmission over the same shared channel. can be done by central controller / in a distributed way by individual hosts



switched network

↳ nodes are connected hierarchically through link-layer switches

↳ mental model: every host addresses a target, switches store & forward until reach

both require addressing their target

---

③ addressing at the link layer

1) MAC addresses

↳ technically speaking, it is not hosts/end devices and routers that have link-layer addresses, but rather their network adapters (aka. NIC) — a device w multiple NICs would thus have multiple MACs & IPs

↳ a link-layer address ⟺ LAN address ⟺ physical address ⟺ MAC address

↳ uniquely identifies a network interface, 48 bit string.

uniqueness

↳ allocation & ensuring uniqueness managed by IEEE

↳ flat structure (as opposed to hierarchical, in IP), and does not change regardless of where adapter is

broadcast address most of the time, MAC uniquely identifies so that when NIC receives (broadcast to all) check whether match, if no, drop. But 0xFFFF-FFFF-FFFF is reserved for broadcast

2) the need for ARP

↳ recall that at the application layer, the need is human-readable addressing. But at the network layer, we need IP for routing. So we have DNS to convert between the two.

↳ Now, we need IP for routing & addressing, but the underlying connection the network layer sees may not be true, and we need MAC addresses for that (eg. switching via switch network vs. p2p communication links)

⇒ we use ARP to convert between MAC and IP

3) address resolution protocol and how it works

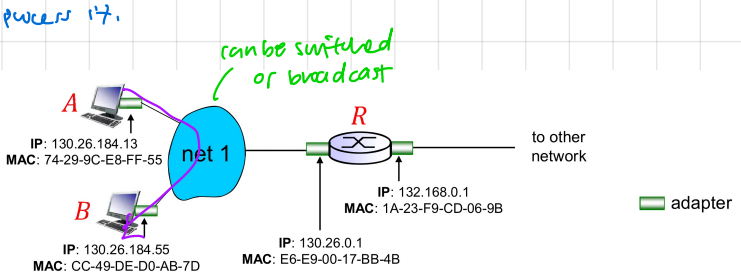
main idea: Each host and router (notice: at the network level) has an ARP table in memory, which maps IP ⟷ MAC addresses and some metadata (eg. TTL)

how it works: A → B.

A knows B's MAC from its ARP table.
1. create link layer frame w B's MAC.
2. send.
3. on broadcast networks, all will receive, only B will process it. on switched networks, only B will receive & process it.

all devices @ network layer - routers and/or hosts

A does not have B in its ARP
1. broadcast ARP query packet on broadcast MAC containing B's IP
2. All on broadcast/switch network will receive. B will reply directly to A its MAC, w AS MAC.
3. A will save to ARP. Then send actual message addressed to B.

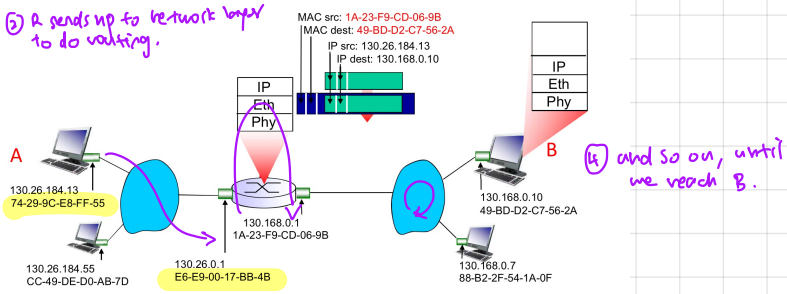


4) within and between subnets

↳ recall that the network layer handles routing, and operates on the abstraction the link layer provides: network layer — hosts, routers; link layer — hosts, routers, switches/broadcast media.

↳ for that reason, link layer addressing on ARP only contains IP ⟷ MAC mapping for devices within the same subnet — since IP handles actual routing — it's not the link layer's concern.

⇒ when we want to IP out of the subnet ⟶ we go to MAC of the router



5) is ARP a link layer protocol?

↳ An ARP packet is a link layer frame, so is technically above the link layer.
↳ But also, it has fields containing link layer addresses, so is arguably link layer
↳ But also contains network layer addresses, so is arguably network layer

⇒ best considered straddling the line

6) IP vs. MAC

- IP address
  - 32 bits in length
  - network-layer address used to move datagrams from source to dest.
  - Dynamically assigned; hierarchical (to facilitate routing)
  - Analogy: postal address
- MAC address
  - 48 bits in length
  - link-layer address used to move frames over every single link.
  - Permanent, to identify the hardware (adapter)
  - Analogy: NRIC number

---

③ broadcast network links : multiple access protocols

1) the multiple access problem

↳ in broadcast channels, when any one node transmits a frame, the channel broadcasts the frame and every other node receives a copy

⇒ because of the shared channel, nodes may talk over, interrupt etc. one another

transmission collisions in broadcast channels, when two or more nodes transmit simultaneously, none of the receiving nodes can decode the frames as the signals of the colliding frames are inextricably tied together

2) multiple access protocols

↳ coordinate transmissions by determining who, when, how and how long a node gets to transmit

- Desired Conversational Characteristics: *etiquettes*
  - Give everyone a chance to speak.
  - Don't speak until you are spoken to.
  - Don't monopolize the conversation.
  - Raise your hand if you have a question.
  - Don't interrupt when someone is speaking.
  - Don't fall asleep when someone is talking.

⇒

Given: Broadcast channel of rate 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 average rate R/M
4. fully decentralized:
   - no special node to coordinate transmissions

Mandatory Requirement: coordination about channel sharing must use channel itself!: no out-of-band channel signaling

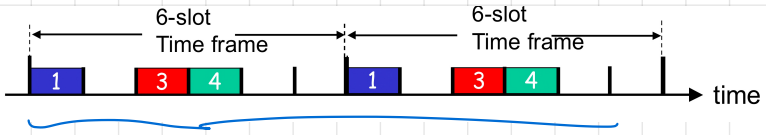
important note

3) channel-partitioning protocols

time division multiple access

main idea suppose we have N nodes. Then each node gets a fixed amount of time t to broadcast in a round-robin fashion.

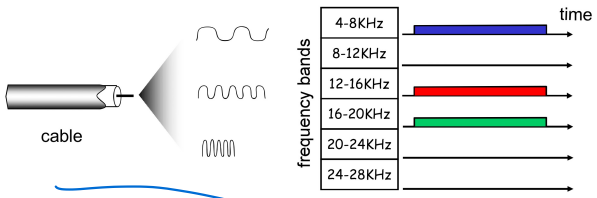
A group of N time transmissions is called a **time frame**



6 nodes, only 1, 3, 4 transmitting

frequency division multiple access

main idea we divide the frequencies we can transmit over into N ranges, where each node transmits only over its assigned range of frequencies.



6 nodes, only 1, 3, 4 transmitting

- Collision Free: Yes
- Efficiency
  - Inefficient
  - Unused slots go idle.
  - The maximum throughput for a node is R/N

} main issue: low efficiency due to idle time

- Fairness: Perfectly Fair
- Decentralized: Yes

4) turn-taking protocols

polling

main idea a master node polls N nodes in a round-robin fashion, letting them know they can transmit for up to some fixed time t. If node does not want to transmit, it is skipped

↳ higher efficiency than FDMA / TDMA as idle nodes are skipped

- Collision Free: Yes
- Efficiency
  - Higher efficiency.
  - Overhead of polling.
- Fairness: Perfectly Fair
- Decentralized:
  - No
  - Master node is a single point of Failure

token passing ← decentralize round robin

main idea instead of having a master node coordinate, we let the nodes tell each other when they are done. When a node is done, transmits to next node to start. When that node is done / time's up / don't want to transmit, it lets next in the chain know, and so on.

- Collision Free: Yes
- Efficiency
  - Higher efficiency.
  - Overhead of token passing
- Fairness: Perfectly Fair
- Decentralized: Yes

- *Downside*
  - Token loss can be disruptive
    - data frame loss
    - System bugs
  - Node failure can break the ring

5) random access protocol

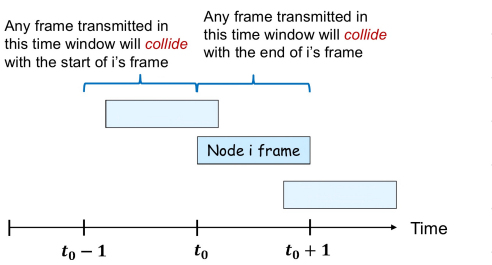
↳ to reduce the communication overhead / failure risks of round robin, instead of explicit turn taking, we can do it implicitly. That is, a transmitting node always transmits at the full rate, but when there is a collision, it picks a random delay, then retransmits (and again if necessary) ⇒ we leave it to probability & independence to coordinate

unslotted ALOHA

main idea: when a node has a frame to send, it transmits its entire dataframe immediately. if there is no collision detected, success. If collision detected, then wait for 1 frame transmission time (locally defined), then retransmit the after with probability p until success. No coordination w any other node.

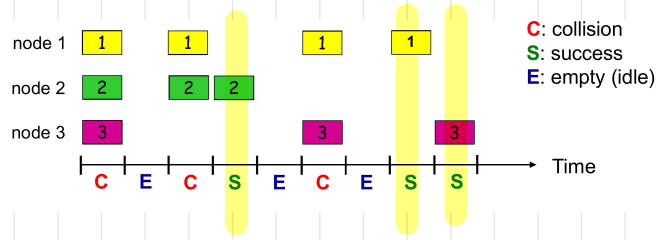
- Collision Free: No
- Efficiency
  - Yes, when only one node is active, it gets a throughput of R
  - No, when there are many active nodes the maximum efficiency is only *18%*
    - Slots are wasted due to both *collision* and because of being *empty*
    - 100 Mbps system will give only 18 Mbps
- Fairness: Perfectly Fair
- Decentralized: Yes

lack of time synchronization ⇒ unnecessary amount of extra collisions due to poor alignment



Slotted ALOHA

main idea: all data frames are split / padded to be of equal size, at L bits. Time is divided into slots of equal length, $t = \frac{\text{bits, } L}{\text{transmission rate, } R}$. when a node has a frame to send, wait until beginning of next slot and transmit the entire frame. If no collision detected, success. If collision detected, retransmit in each subsequent slot w probability p until success.



but, efficiency loss due to idle time & collisions

- Collision Free: No
- Efficiency
  - Yes, when only one node is active, it gets a throughput of R
  - No, when there are many active nodes the maximum efficiency is only *37%*
    - Slots are wasted due to both *collision* and because of being *empty*
    - 100 Mbps system will give only 37 Mbps
- Fairness: Perfectly Fair
- Decentralized: Yes

Efficiency analysis

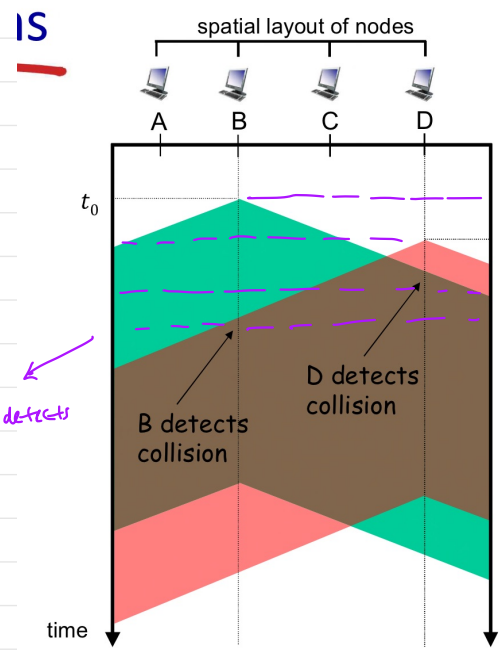
1. when there are multiple active nodes, some periods of time will have colliding frames, and others won't.
   - 1.1 collided time periods are wasted
   - 1.2 some time periods are <u>empty</u> as some refrain from transmitting due to probability p
2. probability of successful transmission at a given dt. is $p(1-p)^{N-1}$ (1 transmit x all others don't)
   - 2.1 probability that any one node succeeds is $N \cdot p \cdot (1-p)^{N-1}$
3. To find max. efficiency, we find p^* s.t. $N \cdot p^* (1-p^*)^{N-1}$ is maximized.
4. Then as $N \to \infty$, efficiency $\to k$.
   - 4.1 For ALOHA, $k \to 1/2e \approx 18\%$
   - 4.2 For slotted ALOHA, $k \to 1/e \approx 37\%$

Carrier sense multiple access

motivation #1 slotted ALOHA has low efficiency due to wasted frame from collision.

main idea #1 Listen before you speak. If channel sensed idle, transmit. Else defer.

motivation #2 Even if we try to listen for collisions, they can still occur due to propagation delay — it takes time for signal to transmit.

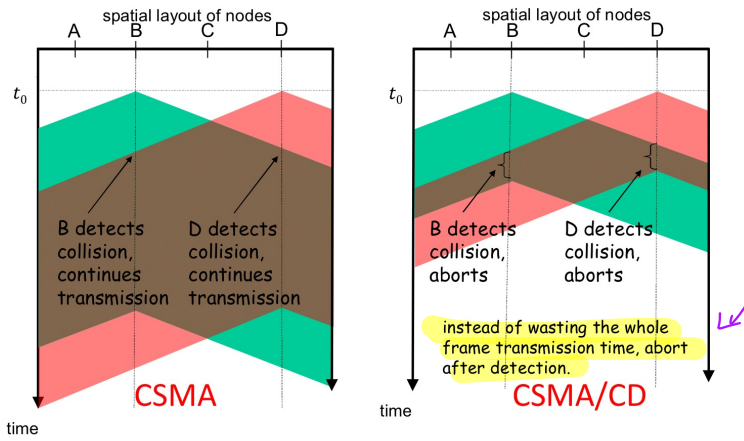


t_1 → ① at t_1, signal from B not reached. transmit.

② at $t_0 + \frac{D_{BD}}{c}$, signal from B reaches D.

③ at $t_1 + \frac{D_{BD}}{c}$, B detects collision

main idea #2 we stop transmission the moment a collision is detected. Because the time period $t = \frac{1}{R}$ is fixed, we can then re-synchronize our clocks for when the next frame starts. ⇒ reduced efficiency loss from wasted frame since we adaptively set timeline



motivation #3 ALOHA takes an efficiency penalty due to empty frames from collision and re-collision as every node's sampling distribution is the same.

main idea #3 CSMA includes a backoff algorithm to adapt retransmission attempts based on estimated current load.

CSMA/CD Backoff Algorithm

For Ethernet 1 time unit is set as 512 bit transmission times

important a.k.a. 1 time unit <=> how much time to transmit 512 bits

Binary Exponential backoff:

- After 1st collision:
  - choose K at random from {0, 1}; $p = 1/2$
  - wait K time units before retransmission.
- After 2nd collision:
  - choose K from {0, 1,2, 2^2-1}. $p = 1/4$
  - wait K time units before retransmission.
- After m^{th} collision $p = 1/2^m$
  - choose K at random from {0, 1, …, 2^m-1}
- *Property*: retransmission attempts to estimates current load
  - More collisions implies heavier load.
  - longer back-off interval with more collisions.

based on the heuristic that more collisions = higher load → reduce probability so that expected no. of tries for another transmission attempt will be longer.

⇒ is a negative binomial. $E(n) = 1/p$.

- Collision Free: NO
- Efficiency: Yes
- Fairness: Yes
- Decentralized: Yes

⑤ link layer switched networks

switch essentially a router that operates at the link layer. But unlike a router, end devices do not address the switch, and behave as if the devices are directly connected. The switch simply forwards and filters link-layer frames.

↳ nodes have dedicated, direct connection to switch
↳ A→B and C→D can transmit simultaneously without collisions, due to buffers.

forward and filter

forwarding: switch function determining which interface a frame should be directed to.

filtering: switch function determining if a frame should be forwarded.

↳ both done by a **switch table**

switch table contains entries for some, not necessarily all, hosts & routers (network layer devices) on a network. contains MAC, interface out, time/TTL.

how forwarding and filtering works

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
   1. if destination on segment from which frame arrived
      1. drop frame
   2. else forward frame on interface indicated by entry
4. else flood
   1. forward on all interfaces except arriving interface

self learning

- Switch *learns* which hosts can be reached through which interfaces.
  - when frame received, switch "*learns*" location of sender
  - *records* sender/location pair in switch table
    - When receiving a frame from A, note down the location of A in switch table.

1. The switch table is initially empty.
2. For each incoming frame received on an interface, the switch stores in its table (1) the MAC address in the frame's *source address field*, (2) the interface from which the frame arrived, and (3) the current time. In this manner, the switch records in its table the LAN segment on which the sender resides. If every host in the LAN eventually sends a frame, then every host will eventually get recorded in the table.
3. The switch deletes an address in the table if no frames are received with that address as the source address after some period of time (the **aging time**). In this manner, if a PC is replaced by another PC (with a different adapter), the MAC address of the original PC will eventually be purged from the switch table.

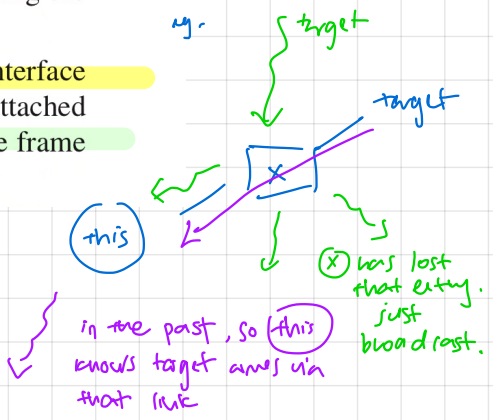
why does this work?

To understand how switch filtering and forwarding work, suppose a frame with destination address DD-DD-DD-DD-DD-DD arrives at the switch on interface x. The switch indexes its table with the MAC address DD-DD-DD-DD-DD-DD. There are three possible cases:

- There is no entry in the table for DD-DD-DD-DD-DD-DD. In this case, the switch forwards copies of the frame to the output buffers preceding *all* interfaces except for interface x. In other words, if there is no entry for the destination address, the switch broadcasts the frame.
- There is an entry in the table, associating DD-DD-DD-DD-DD-DD with interface x. In this case, the frame is coming from a LAN segment that contains adapter DD-DD-DD-DD-DD-DD. There being no need to forward the frame to any of the other interfaces, the switch performs the filtering function by discarding the frame. note : if MAC = 0x FFFFFF ⇒ broadcast to all
- There is an entry in the table, associating DD-DD-DD-DD-DD-DD with interface $y \neq x$. In this case, the frame needs to be forwarded to the LAN segment attached to interface y. The switch performs its forwarding function by putting the frame in an output buffer that precedes interface y.

occurs when neighbouring switch broadcast, but target was come here before

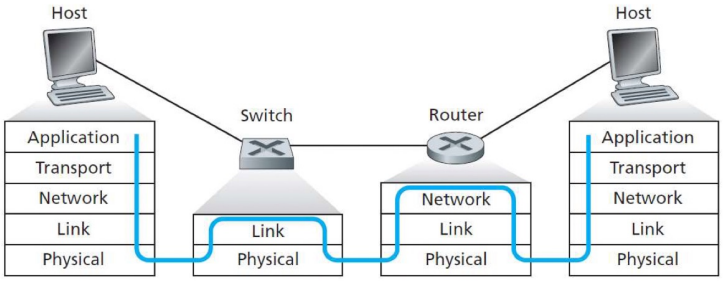
eg.



switches vs. routers

Switches vs. Routers

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



⑤ ethernet : a wired broadcast channel protocol

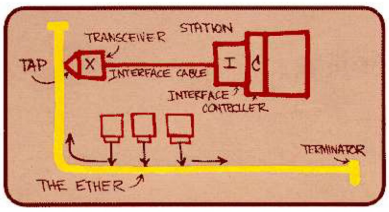
1) LAN technologies

LAN a computer network that interconnects devices within a geographical area

eg. IBM token ring, WiFi, ethernet

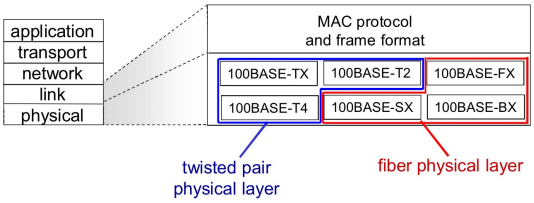
2) Ethernet : the dominant wired LAN technology

- Developed in mid 1970s
- Standardized by Xerox, DEC, and Intel in 1978
- Simpler and cheaper than token ring and ATM

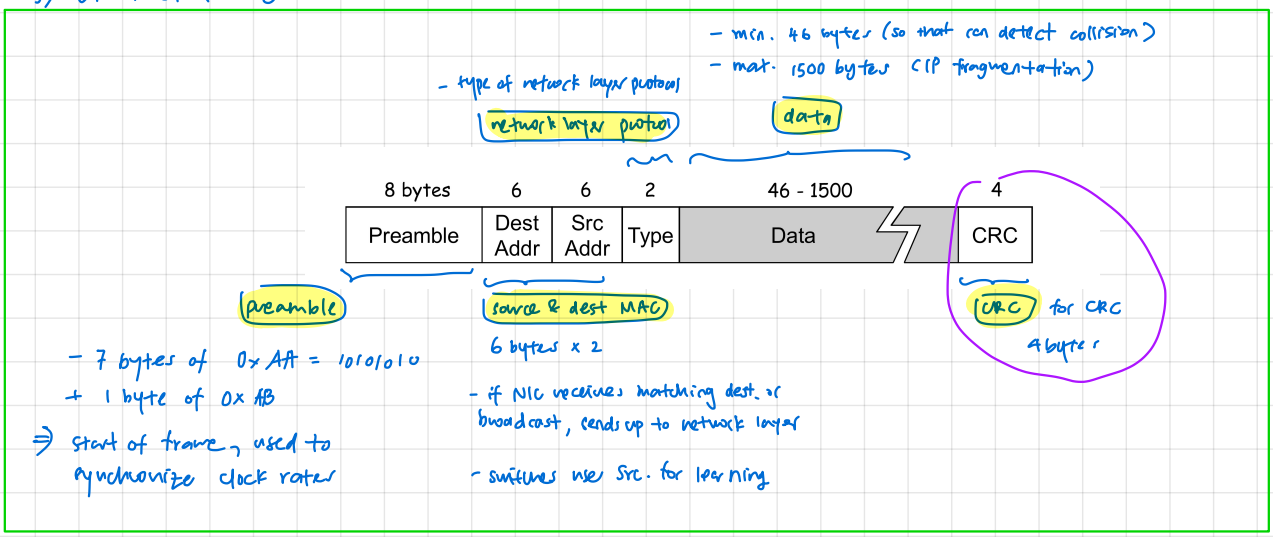


A series of Ethernet standards have been developed over the years.

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



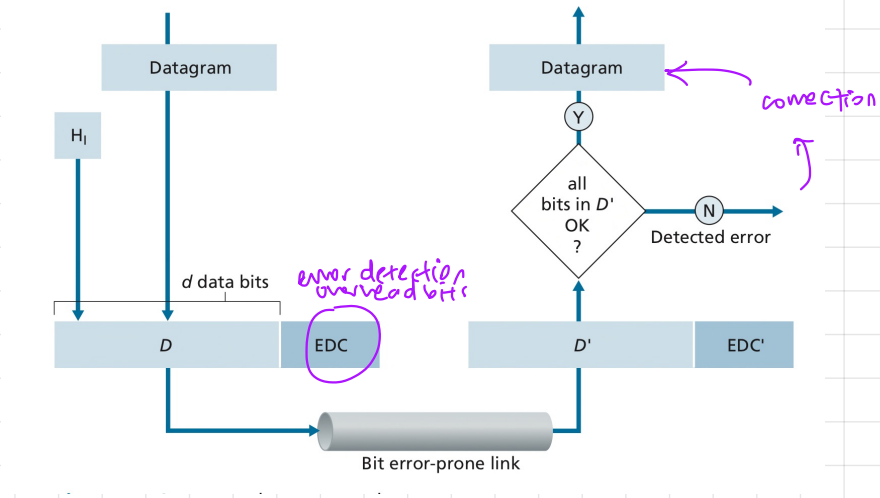
3) Ethernet frame structure



4) 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.
- Ethernet's multiple access protocol:
  - *CSMA/CD* with binary (exponential) backoff.

⑥ error detection and correction techniques



checksum methods

main idea d-bits of data treated as k bit integer. Perform some function to get another number. Send both. Check that they agree on arrival. often d_0 is complement to implement in hardware
eg. UDP checksum

↳ low packet overhead
↳ relatively weak protection

} typically, we use low-overhead methods at higher layers, because at lower layers, we can implement in hardware ⟹ much faster to do complex checks/recovery

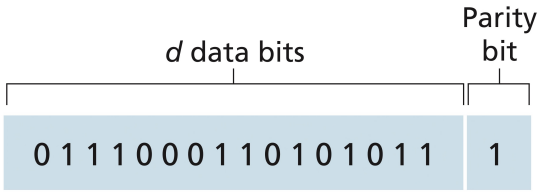
---

parity checks

main idea by including additional bits, choosing their values such that some subset of the original bits (eg. a row) has an even/odd/whatever no. of "1" bits, when the received datagram does not fit ⟶ we can detect (and sometimes fix) these errors.

eg. for even parity scheme

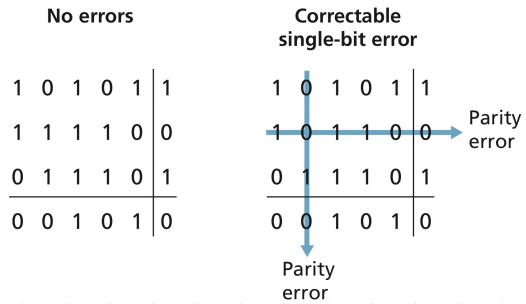
1) parity bit



↳ can detect odd no. of bit flips
↳ cannot detect even no. of bit flips

errors are often clustered in bursts, so probability of undetected errors can approach 0.5

2D parity bit



↳ intersection allows for detection, triangulation & correction of single bit flip
↳ can detect two-bit flips

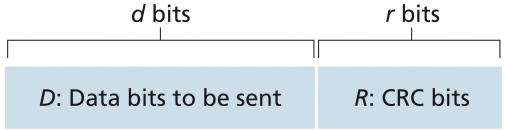
cyclic redundancy checks (polynomial codes)

motivation

↳ checksum methods are weak in detecting errors due to many-many mapping
↳ parity schemes typically cannot detect even/odd bit flips
⇒ we constrain the checking conditions (more stringent check)

main idea sender and receiver agree on a r+1 bit number G. G must lead with a 1. For a given data D, we find a r bit string such that the (d+r) bit pattern is exactly divisible by G. All calculations are done in mod-2 i.e. no carries or borrow ⟶ addition & subtraction are equivalent to XOR.

Deriving R mathematically



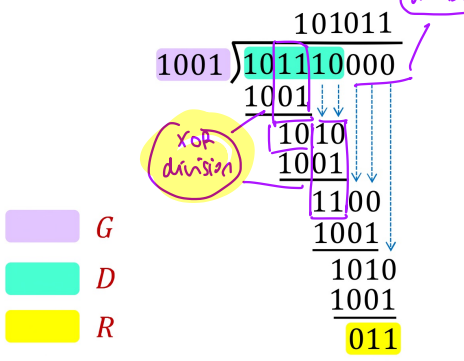
Bit pattern

$D \cdot 2^r$ XOR R — Mathematical formula

$D \cdot 2^r$: bit shift, since we append

addition mod 2

0. Given G.
1. r bit shift D
2. Do binary division



E.g.
$D = 101110$,
$r = 3$
$G = 1001$

1. Bit pattern of length d+r is equal to $(D \cdot 2^r)$ XOR R
   (bit shift; add modulo 2)

2. We want to find R such that bit pattern divisible by G.

$(D \cdot 2^r)$ XOR $R = nG$, $n \in \mathbb{Z}$

⇔ $(D \cdot 2^r) = nG$ XOR R since addition & subtraction mod 2 are both XOR

⇔ $D \cdot 2^r = nG$ + mod 2 remainder R
(nG: exact terms of G)

⇔ R = remainder of $\frac{D \cdot 2^r}{n}$

intuitively, R "back fills" D so that (d+r) bits are divisible by G.

Running the check

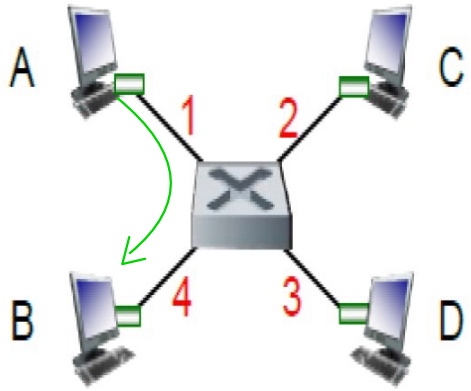
1. we send d+r bits. G known to both sides.
2. Do mod 2 division of (D,R) ÷ G.
   If == 0, OK. Else, error.

properties of CRC

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

Notes

1. putting ARP & packet switching + self learning together



Consider sending an IP datagram from Host A to Host B. Suppose all of the ARP tables and switch table are up to date. Enumerate all the steps the host and switch take to move the packet from A to B.

1) A creates a frame with destination MAC address CC-49-DE-D0-AB-7D (B's address is found in ARP table).
2) This frame travels to switch S and is forwarded towards B (interface to B is found in switch table).

Repeat the problem in a), assuming that all tables in all nodes are empty.

1) A needs to issue an ARP query to know the MAC address of B.
2) The query packet travels to switch S and is forwarded to both B and R. Switch S learns A is reachable via the interface query packet arrives at.
3) 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). Switch S learns B is reachable via the interface reply frame arrives at.
4) Subsequently A can send IP datagram to B as in part a).

Suppose A sends an IP datagram to a host in another subnet. All of the ARP tables and switch table are up to date. Enumerate all the steps the host, switch and router take to move the packet to another subnet.

1) A creates a frame with destination MAC address E6-E9-00-17-BB-4B (R's address is found in ARP table).
2) This frame travels to switch S and is forwarded towards R (interface to R is found in switch table).
3) R checks the destination IP of the datagram and decides to forward it towards external network. It encapsulates the IP datagram in a new frame with source MAC address 1A-23-F9-CD-06-9B (dest MAC address not mentioned in question) and sends it through the interface towards external network.

Notes

1. don't confuse ARP table (IP ↔ MAC) with switch table (MAC ↔ link).
Both self learning.

↓ learn via ARP queries

↓ learn by recording inbound, if don't have, broadcast all

Network Security

① Network security

1) Motivation and aspects of security

Q: What can a "bad guy" do?

A: A lot!

- *eavesdrop:* intercept messages
- actively *insert* messages into connection
- *impersonation:* can fake (spoof) source address in packet (or any field in packet)
- *hijacking:* "take over" ongoing connection by removing sender or receiver, inserting oneself in place (Man in the middle)
- *denial of service:* prevent service from being used by others (e.g., by overloading resources)

- *Confidentiality:*
  - only sender, intended receiver should *"understand"* message contents
- *Authentication:*
  - sender, receiver want to *confirm identity* of each other
- *Message integrity:*
  - sender, receiver want to ensure message *not altered* (in transit, or afterwards) *without detection*
- *Access and availability:*
  - services must be accessible and available to users

---

② cryptography and confidentiality

1) key concepts

plaintext message in the original form

ciphertext encrypted message

key a string provided as input to the encryption / decryption algorithm

symmetric key cryptography Sender and receiver use the same key. $k_A = k_B$.

asymmetric key cryptography sender and receiver use different keys. $k_A \neq k_B$

ciphertext-only attack Attacker only has ciphertext

known-plaintext attack Attacker has plaintext – ciphertext pair

chosen-plaintext attack Trudy can get ciphertext for chosen plaintext

2) symmetric key cryptography

main idea sender and receiver share the same key, decided upon prior to communication.

$encrypt_{shared}(K_{shared}, message) \rightarrow ciphertext \rightarrow decrypt_{shared}(K_{shared}, ciphertext)$

substitution ciphers substitute one character for another

Caesar's cipher

↳ rotation. n character alphabet, n-1 possible substitution schemes.

- e.g., right shift by 3:

```
abcdefghijklmnopqrstuvwxyz
defghijklmnopqrstuvwxyzabc
```

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

monoalphabetic cipher

↳ permutation. n character alphabet, n! possible schemes.

```
abcdefghijklmnopqrstuvwxyz
mnbvcxzasdfghjklpoiuytrewq
```

e.g.:
```
Plaintext: bob, i love you. alice
ciphertext: nkn, s gktc wky. mgsbc
```

because of 1-1 mapping, substitution techniques are vulnerable to **statistical analysis**

- *letters e (13%) and t (9%) are the most frequent letters*
- *knowing that particular two-and three-letter occurrences of letters appear quite often together (for example, "in," "it," "the," "ion," "ing,"*
- *If the intruder has some knowledge about the possible contents of the message, then it is even easier to break the code.*
  - *if Trudy the might suspect that the names "bob" and "alice" appear in the text.*
  - *and had a copy of the example ciphertext message above, then she could immediately determine seven of the 26 letter pairings*

polyalphabetic encryption use n monoalphatic substitution ciphers $C_1, C_2 \ldots C_n$. use some **cycling pattern** (a formula) to determine which cipher to use for the i^{th} character. eg. n - (i % n) cipher

⇒ key: {$C_1 \ldots C_n$, cycling pattern}

---

block ciphers message broken into blocks of k bits. Then we encrypt each of the K blocks individually and then concatenate the outputs.

- E.g.: $K = 3$
  - Input: 010110001111
  - Encrypted output: 101000111001

use a lookup table of formula to get 1-1 mapping

Input	Output
000	110
001	111
010	101
011	100
100	011
101	010
110	000
111	001

- Number of keys: $2^K!$
  - $2^{64}!$ Is an astronomical value.

↳ for binary, there are 2^k possible strings of length k. Suppose we fix inputs. Then there are $(2^k)!$ ways to permute the output strings.

DES: Data Encryption Standard
- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit block
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in *less than a day*
- Making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

AES: Advanced Encryption Standard
- Symmetric-key NIST standard, replaced DES (Nov 2001)
- 128 bit blocks; 128, 192, or 256 bit keys
- How secure is AES?
  - Machine capable of Brute force decryption DES in 1 sec on DES, takes *149 trillion years* for 128-AES

↳ key = {key to encryption scheme, block size k}

---

2) asymmetric key cryptography

↳ symmetric key cryptography requires sender and receiver to share key, with N nodes ⇒ NC_2 keys!

public key cryptography

1. Suppose we have some function f and public key K^+ and private key K^-.

1.1 suppose we choose f, K^+ and K^- such that $m = f(K^-, f(K^+, m))$ (encrypted; decrypted back)

1.2 Suppose also that if one knows K^+ and f, there is no way to get K^- in a feasible amount of time.

2. Then we have an easy way for a sender to use a public key to encrypt a message, and only the reciver, with his private key, can decrypt it!

---

③ Ensuring message integrity

↳ just as in error detection, we simply want to detect if the message from sender has been modified

↳ checksum, parity, CRC all work, but have their flaws from trading speed for security

checksum → weak

parity → weak

CRC → minor changes in input lead to minor changes in output ⇒ bias to the input may make it hard to detect changes + errors

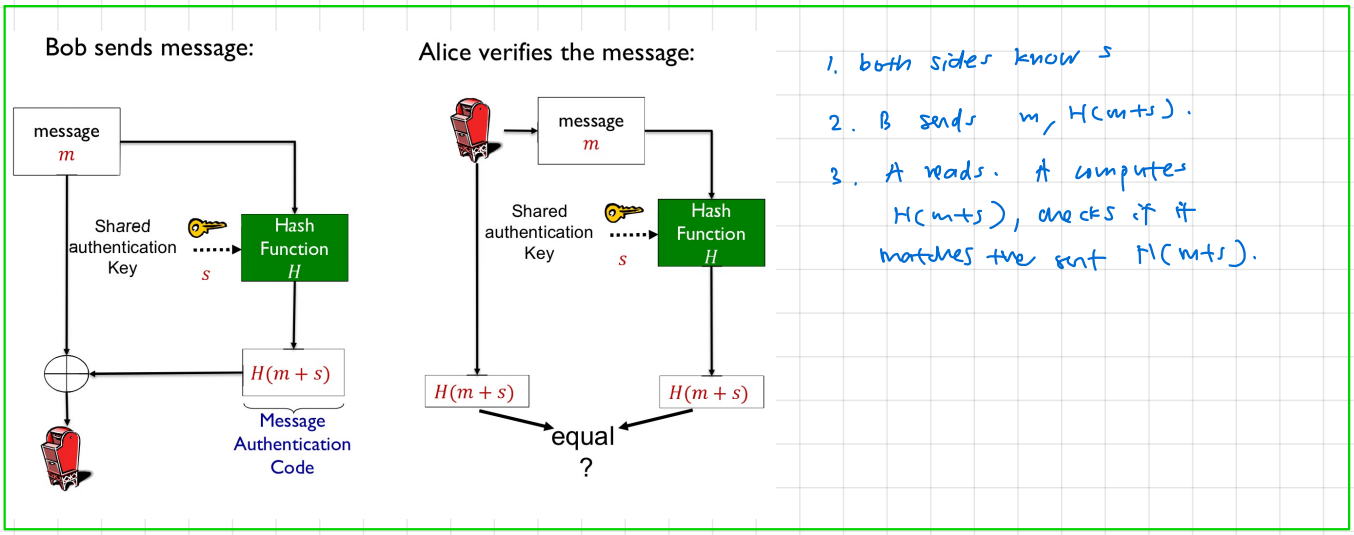
cryptographic hash functions: hash functions such that it is computationally infeasible to find x and y, $x \neq y$ s.t. $H(x) = H(y)$

⇒ computationally infeasible to have malicious message y s.t. $H(x) = H(y)$

using cryptographic hash functions

↳ naively, we send m, $H(m)$. But an intruder could send m', $H(m')$ and we'd be none the wiser.

↳ we send an authentication key s together: m, $H(m+s)$.



1. both sides know s
2. B sends m, $H(m+s)$.
3. A reads. A computes $H(m+s)$, checks if it matches the sent $H(m+s)$.

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest.
- SHA-1 is also used
  - US standard [NIST]
  - 160-bit message digest
- Both SHA-1 and MD5 are cryptographically broken
  - NIST formally deprecated use of SHA-1 in 2011
  - Replaced by SHA-2, SHA-3

---

④ Authentication

↳ when we receive messages, aside from making sure message is encrypted and not tampered with, we also want to know that it really is from the sender

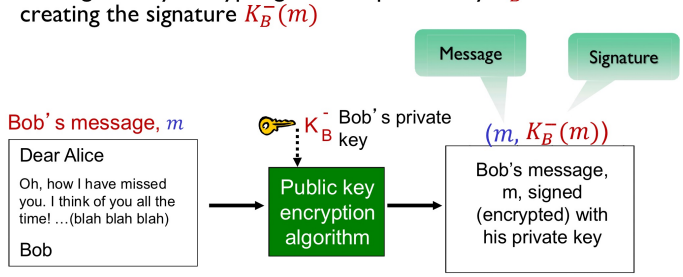
1) **digital signatures** — cryptographic techniques analogous to hand-written signatures; must be verifiable and (practically) unforgeable

using asymmetric cryptography — the central reason why asymmetric key cryptography works is that a function, applied once w/ public key and then private key, gets the same input. suppose now we extend this by choosing such that private first also works.

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

↳ with this property, if sender signs w/ private key, then receiver can decode w/ public. then because it is computationally infeasible to get private key from public key & algo ⇒ we can verify and be sure it is not forged

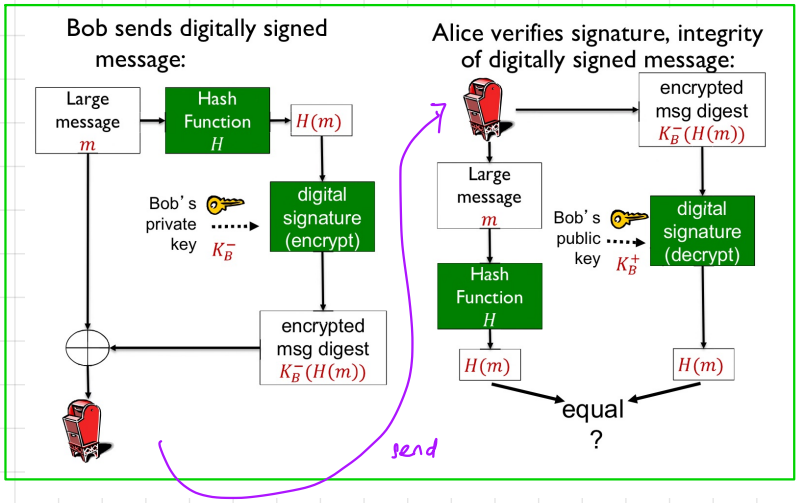
- Bob signs m by encrypting with his private key K_B^- creating the signature $K_B^-(m)$



but applying these on unbounded message lengths can be challenging.

⇓

we use cryptographic hash on message to cut it to a fix length, then do the authentication pureme in this way.



note: this does not take encryption w receiver's public-private key pair into account

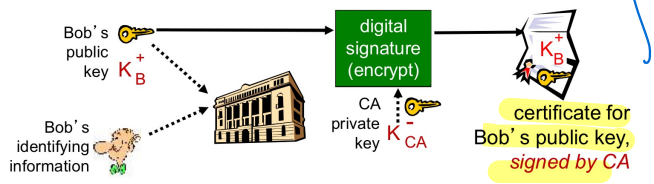
2) public-key certification

↳ how can we be sure someone's public key is actually theirs?

↳ if we have an online certification system, then it can be intercepted → we have a chicken-egg problem

⇒ we maintain a list of certification authorities built into the OS

- *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 digitally signed by CA
    - CA says "this is *E*'s public key"



→ then at run time, we get this, and because CA's public key is baked into the OS, we can verify that it has been verified by the CA.

⑤ Firewall

↳ typically an ingress controller aimed at → preventing DDOS, → prevent illegal access, → prevent unauthorised access

control traffic in and out

stateless packet filter

- internal network connected to Internet via *router firewall*
- router *filters packet-by-packet,* decision to forward/drop packet based on:
  - source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits

determined by a policy

↓ eg.

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

implemented via ACL

ACL: table of rules, applied top to bottom 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

stateful packet filter

application gateway

weaknesses

- *IP spoofing:* router can't know if data "really" comes from claimed source
- Can become a bottleneck
- *tradeoff:* degree of communication with outside world, level of security
- Many highly protected sites still suffer from attacks

⑥ RSA

Rivest, Shamic, Adleman algorithm — a public key cryptographic algorithm used for message encryption and authentication.

1) message encryption

RSA: Creating public/private key pair

1. choose two large prime numbers p, q.
2. compute $n = pq, z = (p-1)(q-1)$
3. choose e (with $e < n$) that has no common factors with z (i.e., e and z are "relatively prime").
4. choose d such that $ed - 1$ is exactly divisible by z. (in other words: $ed \bmod z = 1$).
5. *public* key is $\underbrace{(n, e)}_{K_B^+}$. *private* key is $\underbrace{(n, d)}_{K_B^-}$.

RSA: encryption, decryption

0. given (n, e) *and* (n, d) as computed above
1. to encrypt message m (*Note:* $m < n$), compute
$$c = m^e \bmod n$$
2. to decrypt received bit pattern, c, compute
$$c^d \bmod n$$

encryption / decryption

magic happens! $(\underbrace{m^e \bmod n}_{\text{ciphertext}})^d \bmod n = m$

Why does RSA work? The main "magic" comes in how we choose e, d and n such that the desired reversible property is achieved. out of scope of the class. see page 634 of the tb.

RSA encryption in practice — exponentiation for encryption / decryption is computationally expensive. Instead, we often use RSA in a "handshake" to encrypt a symmetric session key and use that for all subsequent communication. (eg. DES)

2) message authentication

a useful property: $K_B^-(K_B^+(m)) = K_B^+(K_B^-(m))$. So we use RSA for digital signatures.

1. $K_B^- = (n, d)$, $K_B^+ = (n, e)$

1.1 LHS = $\left[(m)^e \bmod n\right]^d \bmod n$

1.2 RHS = $\left[(m)^d \bmod n\right]^e \bmod n$

2. Recall: $\left[a \bmod n \times b \bmod n\right] \bmod n = (a \times b) \bmod n$

3. Then from 1.1:

3.1 $\left[(m)^e \bmod n\right]^d \bmod n = \left[\underbrace{m \bmod n \times m \bmod n \ldots}_{e \text{ times}}\right]^d \bmod n$

$= (\underbrace{m \bmod n \times m \bmod n \ldots}_{e \text{ times}}) \bmod n \times \ldots$

$\underbrace{\qquad\qquad\qquad}_{d \text{ times}}$

3-2 $= (m \mod n)^{d \cdot e} \mod n$

$= (m^d \mod n)^e \mod n$ by the same logic.

multimedia networking

① basics of multimedia networking

1) application types

- **Streaming stored** audio, video
  - *Streaming:* can begin playout before downloading entire file
  - *Stored (at server / CDNs):* can transmit faster than audio/video will be rendered (implies storing/buffering at client)
  - e.g., YouTube, Netflix, Hulu
- **Conversational ("two-way live")** voice/video over IP
  - interactive nature of human-to-human conversation limits delay tolerance
    - Delay more than 400 milli seconds, intolerable
  - e.g., Skype, Zoom, WhatsApp
- **Streaming live ("one-way live")** audio, video
  - Typically done with CDNs
  - e.g., live sporting event (soccer, football)

Jargon Alert:
- CDN: Content Distribution Network

2) types of multimedia

video a sequence of images displayed at a constant rate; each image is an array of pixels, and each pixel is (typically) a byte

↳ we typically use redundancy within and between the images to reduce no. of bits

spatial coding lazy / sparse representation of frame. eg. instead of 256 x 256, encode as (colour, number) pairs in row-col order → reduce repeated info

temporal coding sparse representation of sequence. eg. instead of all frames, send bare frames then diff.

↳ characterised by its high **bit rate** : high amounts of bits needed to encode the video per second

CBR: (constant bit rate): video encoding rate fixed
- Not responsive to the complexity of the video.
- Need to set your bitrate relatively high to handle more complex segments of video.
- The consistency of CBR makes it well-suited for real-time encoding.
  - For *real-time live streaming*

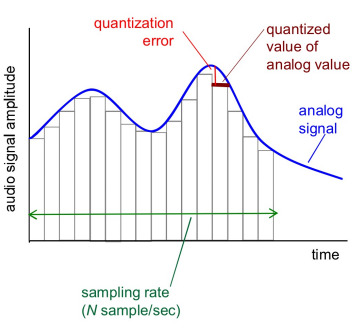
VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- VBR best suited for *on-demand video* due to longer time to process the data.

---

audio

↳ an analog signal we quantize at a constant sampling rate

↳ sampling causes loss due to quantization error



② streaming stored video

streaming: begin playout before downloading the entire file

1) continuous playout amidst jitter

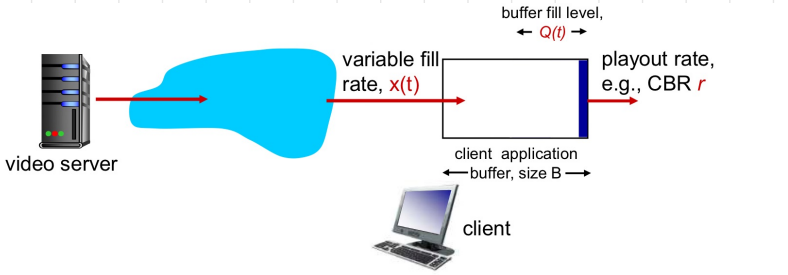
continuous playout challenge: once client begins playout, playback must match original timing. But network delays are variable, client can interact (e.g. fast forward), video packets may be lost, etc.

we use **client-side buffering and playout delay**

↳ given sufficient buffer b, variable fill rate $\bar{x}$, fixed playout rate r:

- $\bar{x} < r$: buffer eventually empties
- $\bar{x} \geq r$: buffer will not empty, provided b able to absorb random variability in $x(t)$

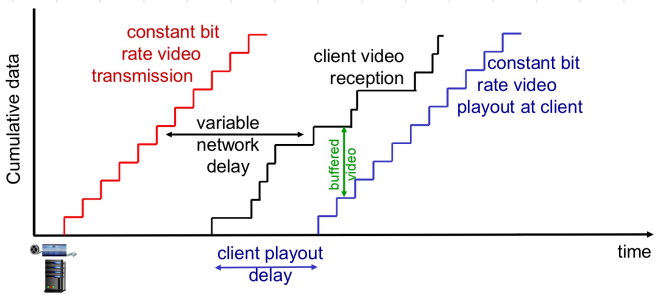
↳ aka we load a buffer to b, then start playback



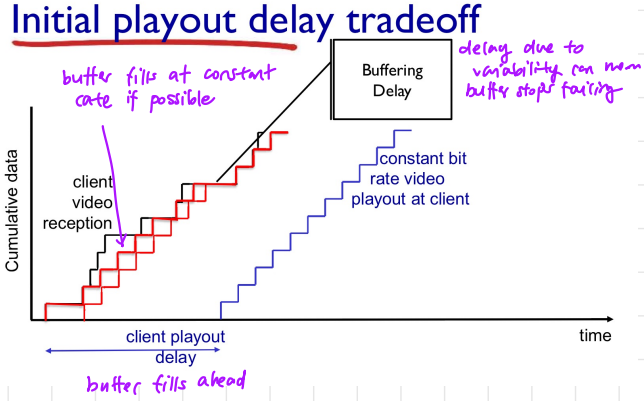
1. Initial fill of buffer until playout begins at t_p
2. playout begins at t_p,
3. buffer fill level varies over time as fill rate x(t) varies and playout rate r is constant

initial playout delay tradeoff

↳ buffer starvation less likely w/ larger delay, but longer delay until user can watch



- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter



2) streaming media

push-based streaming

- server sends at rate appropriate for client
  - often: send rate = encoding rate = constant rate,
    - ☞ push-based streaming (*server push*)
  - UDP has no congestion control
    - Hence transmission without rate control restrictions
- short playout delay (2-5 seconds) to remove network jitter
- Error recovery: application-level, time permitting

real time protocol, application layer, uses UDP

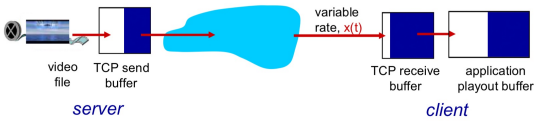
- Video chucks encapsulated using RTP
- Control Connection is maintained *separately* using RTSP

real time streaming protocol, application layer, uses TCP

  - Is used for establishing and controlling media sessions between endpoints.
  - Clients issue commands such as *play*, *record* and *pause*
- Drawbacks
  - Need for a separate media control server like RTSP, increases cost and complexity
  - UDP may *not* go through firewalls

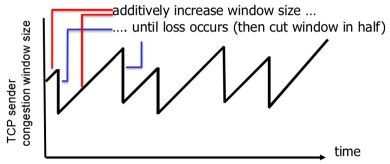
pull-based streaming

- multimedia file retrieved via HTTP GET,
  ☞ pull-based streaming (*client pull*)
- send at maximum possible rate under TCP



- Advantages
  - HTTP/TCP passes more easily through *firewalls*
  - Network *infrastructure* (like CDNs and Routers) fine tuned for HTTP/TCP
- Drawbacks
  - fill rate *fluctuates* due to TCP congestion control, retransmissions (in-order delivery)
  - *larger* playout delay: smooth TCP delivery rate

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



3) conversational multimedia

the needs to maintain the conversational aspect, we need delays < 150ms. Any delay > 400ms is bad from end-to-end. Also, data loss over 10% is about unintelligible.

how we record data in chunks then send them at each talk spurt (ie. talk → record → start sending) instead of send all (emsilence)

the challenge most use cases use the internet at the network layer, which is a best-effort service with no upper bound on delay, variability in delay and packet loss

- delay loss: IP datagram arrives too late for playout at receiver
- network loss: packet lost due to network congestion

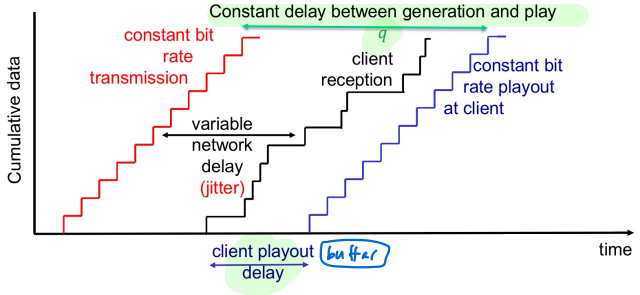
1) handling delay loss with a delay

fixed playout delay just as in streaming, we have a buffer and initial delay. But when our buffer is empty & packet not arrived, we discard it. i.e. buffer, start playing, ignore late chunks.

- receiver attempts to playout each chunk exactly q msecs after chunk was generated.
  - chunk has time stamp t: play out chunk at $t + q$
  - chunk arrives after $t + q$: data arrives too late for playout: data "lost"

Every Chunk will have
- Sequence Number
- Timestamp

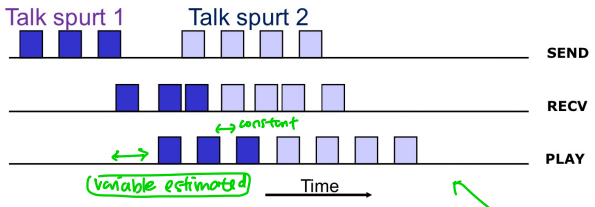
- tradeoff in choosing q :
  - *large* q : less packet loss
  - *small* q : better interactive experience



- 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 fixed delay is inefficient — if packet arrives early, we wait unnecessarily. if packet arrives just a little late (tolerable), we throw it away. Instead, we can estimate network delay and use that to decide our buffer time adaptively.

- *goal:* low playout delay, low late loss rate
- *approach:* adaptive playout delay adjustment
  - *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



- 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)$_i$ = measured delay of ith packet

v_i: 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:

$$\text{playout-time}_i = t_i + d_i + 4v_i$$

  - remaining packets in talk spurt are played out periodically

3) recovery from chunk network loss

idea conventional ACK/NACK techniques too slow. Instead we use self-correcting schemes or failover schemes (eg. send lower quality version tgt to mix in)

forward error correction send additional data to help reconstruct lost chunk

- 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

why does this work? Because XOR is commutative & associative - if one chunk lost, XOR all → get missing chunk.

```
1 0 1 0 1 1
1 1 0 1 0 1
1 0 1 0 0 0
1 1 0 1 1 0
```

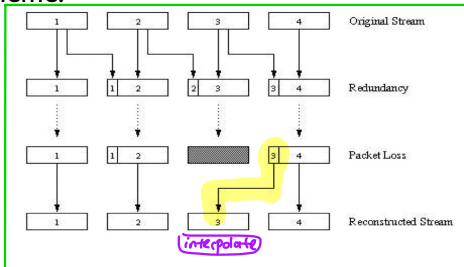
- Drawback
  - Increasing bandwidth by factor $1/n$
  - Playout delay is increased during packet loss
    - Receiver waits for $n+1$ chunks before playout

---

piggyback concealment send lower resolution file of previous chunk for a given chunk. If previous chunk lost → can conceal

Another cool FEC scheme:

- "piggyback lower quality stream"
- send lower resolution audio stream as redundant information
  - e.g., nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps



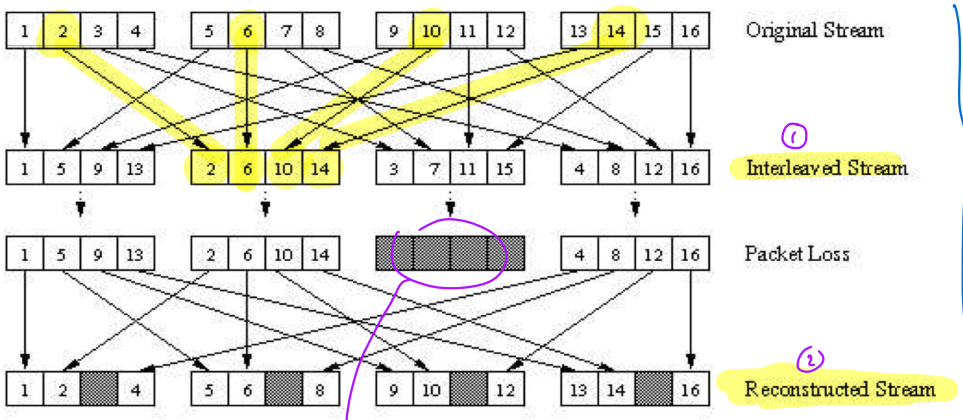
- *non-consecutive* loss: receiver can *conceal* loss
- generalization: can also append (n-1)st and (n-2)nd low-bit rate chunk

drawbacks higher overhead per packet

no redundancy scheme?

interleaved concealment — if we split a chunk into n chunks, then mix n chunks of n sub chunks together → if we lose one or a few packets, most of every chunk still exists → can reconstruct by interpolation. Essentially, each chunk is a sample of n chunks.

① 2nd chunk is 2nd sub chunk of every chunk



② even if lost, can reassemble w slight missing (distribute loss out)

drawback: higher playout delay like FEC, as need to wait for n chunks to arrive & reconstruct them before can do playout

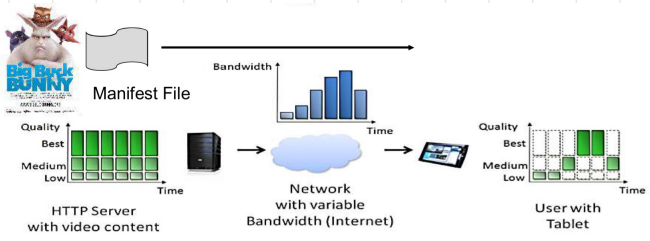
---

⑭ HTTP streaming

1) HTTP streaming and the variability problem
   - with HTTP GET requests, we GET the whole file
   - even if client side buffers and allows streaming, we are not able to adapt to the large amount of variations in bandwidth available, because the file bit rate was fixed the moment we made the GET request

2) dynamic adaptive streaming over HTTP (DASH)

main idea: we don't GET the whole video. Instead, the server stores a URL for a file w different bit rates, and the client — depending on bandwidth estimation — specifies the bit rate and chunk/byte range (re-estimating bandwidth as it gets this, for the next request). This server-side scheme allows the client to adaptively select and switch between quality levels.



- Data is encoded into different qualities and cut into short segments (streamlets, chunks). ① before hand
- Client first downloads *Manifest File*, which describes the available videos and qualities. ② at first request
- Client/player executes an adaptive bitrate algorithm (ABR) to determine which segment do download next. ③ while playing back

- **Advantages of DASH**
  - 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)

a.k.a. we intentionally don't buffer as much, so interactivity could suffer

⑤ content distribution networks

main idea storing large media at a single / few servers can be slow due to network congestion, long distance from end users, etc.

⇒ instead, we can have dedicated servers storing media at geographically distributed sites

- *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 near (but not within) access networks
  - used by Limelight

Jargon Alert:

- IXP: *Internet* exchange point