COMPUTER NETWORKS

Chapter 1

Everything in a Network (Internet)

→ Host / end systems
→ Communication link and packet switches
  ↳ has different transmission rates calculated in bits/sec
→ Packets ← the segments of data sent over from one end system to another
→ Router and link-layer switches (packet switches)
→ The journey made by a packet through communication link, packet switches is called "Route".
→ Internet Service providers (ISPs)
→ Protocols ⟶ TCP
  → IP
→ Internet Standards
  ↳ Request for comments (RFCs)

Types of hosts / end systems → on edge
  → on core
  ↳ access media

* network edges include hosts i.e clients & servers
* Access network, physical media includes wired or wireless communication links

*Network core includes: Interconnected routers, network of networks (large ones with TBs of bandwidth)

→ edge routed: routers connected to network edges.

↳ Shared Access
↳ FDM → frequency division multiplexing
TDM → time division multiplexing

---

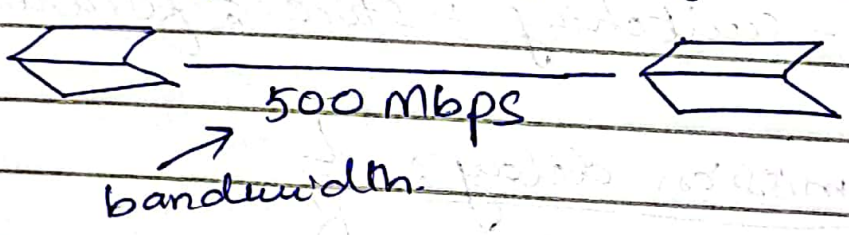
data packet goes to ↳ TCP ⇒ becomes "segment" goes to ↳ "datagram" becomes ⇐ IP goes to ↳ MAC ⇒ becomes "frame"

packet transmission delay = time needed to transmit L-bit packet into link

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

example:

file size = 1 Giga Bytes



packet transmission delay $= \frac{L}{R} \Rightarrow \frac{1 \times 8 \times 10^9 \text{ bits}}{500 \times 10^6 \text{ bits}}$

= 16 sec

physical media:

* half-duplex
* full-duplex

network core

↓

"packet switch"

router switch

Two-key network core functions

routing → determine the path

forwarding → choose a path & move

Packet-switching : store & forward

packet transmission delay : $\frac{L}{R}$

$\frac{10 \times 10^3 \text{ bits}}{100 \times 10^6 \text{ bits}} \Rightarrow 0.0001 \text{ sec.}$

$\Rightarrow 0.1 \text{ msec}$

store & forward :

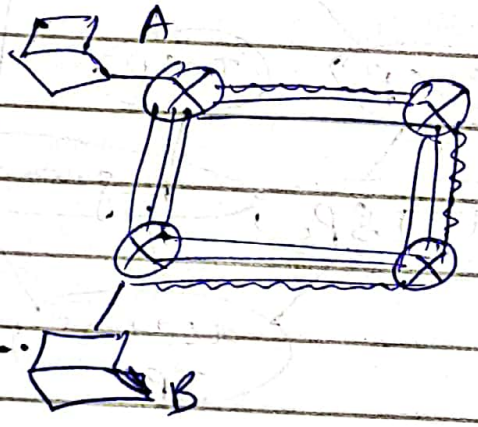
entire packet must arrive at router before it can be transmitted next.

queing : queues of packets are formed at routers / buffer.

⇒ queing occurs when work arrives faster than it can be serviced.

Packet queing & loss: packets will be dropped if when reached a buffer that is full.

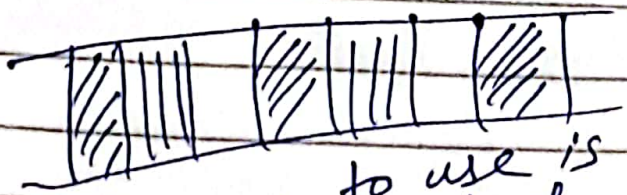
circuit switching: multiple lines connect to routers and each connection is specified for end users.



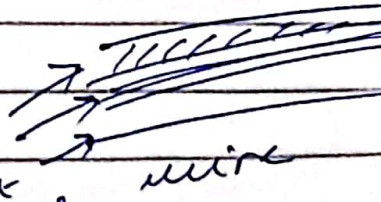
specified for end users A & B's communication

FDM → frequency e.g Radio → total bandwidth is divided

TDM → time → the time is divided & everyone gets full bandwidth.



time to use is divided.

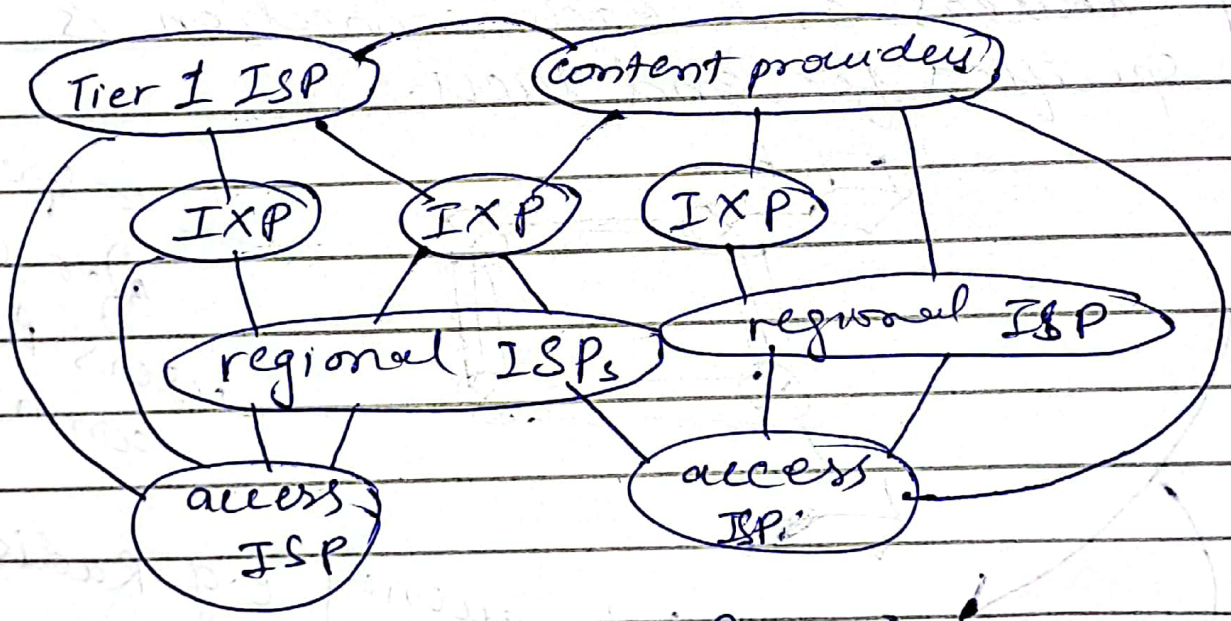


3 freq ranges

wire

→ packet switching is preferred for bursty data i.e data that is not normalized, fluctuates.

IXP : internet exchange points.

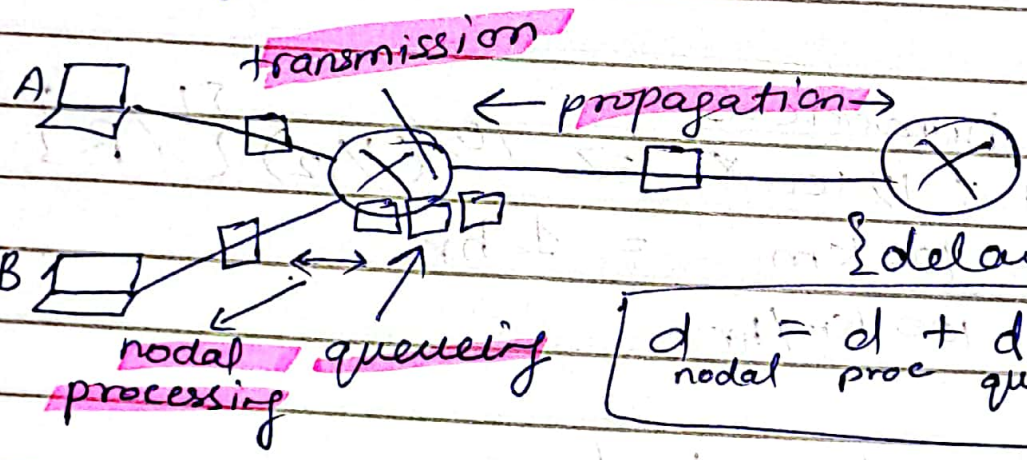


Internet: network of networks

How do packet delay & loss occur?

→ packets queue in router buffers.
→ packet loss occurs when buffers are full

Packet delay: four sources



1 hop = router to before entering next router.

nodal processing: some tasks are performed on packets when they enter router
↳ check bit errors, determine output link.

queuing: buffer delay. $d_{queue} = \frac{L \cdot a}{R}$ (packet length, average packet arrival rate; link bandwidth)
a = arrival rate of bits
service rate of bits

transmission: time a packet takes to be placed on line from router.
$d_{trans} = L/R$

Propagation delay: the time taken ~~from~~ by packet from one router to the next.

$d_{prop} = \frac{d}{S}$ → length of physical link
S → propagation speed 2×10^8 m/s

Caravan numericals:

d_{tran} = 10 bit × 12 sec ⇒ $10/(1/12)$ {L/R}

$d_{prop} = \frac{100 \text{ km}}{100 \text{ km/hr}} = 1 \text{ hr}$

delay = $d_{trans} + d_{prop}$

= 120 sec + 1 hr

= 2 min + 60 min

= 62 min.

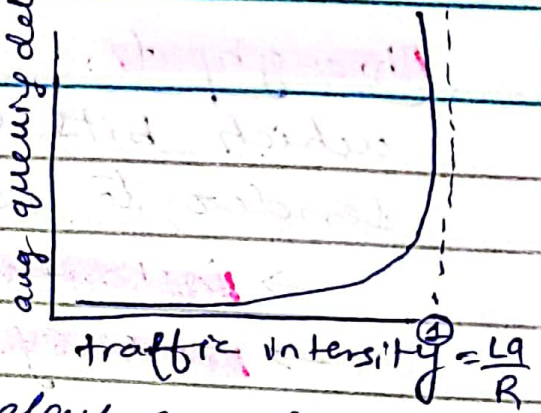
---

d_{trans} = 1 min × 10 ⇒ 600 sec.

$d_{prop} = \frac{1000 \text{ km}}{1000 \text{ km/hr}} \Rightarrow \frac{1}{10} \text{ hr} \Rightarrow$ ~~3600 sec~~ 360 sec.

↙ is variable because of unpredictable traffic

packet queuing delay



"traffic intensity"

$\frac{L \cdot a}{R} \sim 0$: avg queuing delay small.

$\frac{L \cdot a}{R} \to 1$: avg queuing delay large

$\frac{L \cdot a}{R} > 1$: more "work" arriving is more than can be serviced - avg delay infinite.

---

open command prompt & ~~search~~ enter traceroute

e.g traceroute (tracert) google.com

↑ gives you all names & delays of packets and links & losses b/w your system & google.

* probe packet.

* TTL : time to live.

→ first you send 3 probe packets to get ip addresses of each router.

Throughput: rate (bits/time unit) at which bits are being sent from sender to receiver.

→ **instantaneous**: at a point time

→ **average**: over long period of time.

Bottleneck link: the link with the least bandwidth in an end to end path.

per connection end-to-end through-put = $\min(R_s, R_c, \frac{R}{n})$

where n is the number of connections in network.

layers:

application: supporting network applications {HTTP, IMAP} etc

transport: process-process data transfer

network: routing of datagrams from source to destination

link: data transfer b/w neighboring network elements

Encapsulation: example of matryoshka dolls

message → segment → datagram → frame

application layer | transport layer | network layer | link layer

ARP: Address Resolution Protocol

↳ A procedure that connects an ever-changing IP address to a fixed physical machine address called media access control (**MAC**).

Chapter 2
Application Layer

* applications run on end-systems, and not on switches or routers (network core)
* **client-server paradigm**
  - → servers have permanent IP address
  - → clients have normally dynamic IPs

  e.g HTTP, IMAP, FTP
* **peer-to-peer paradigm**
  - → no always-on centralized server
  - → end systems communicate directly
  - → self scalability: every new system would use services but also bring service.
  - → intermitently connected, dynamic IPs.

  e.g P2P file sharing (BitTorrent)

Processes communiting

→ process : program running within a host

→ process communication within same host inter-process communication

→ process communication among host are messages.

Sockets

→ a process sends/receives messages to/from its socket.

→ the socket is present in the connection b/w application layer & transport layer

→ one socket is on client side and the other on server side

* to receive messages, process must have an identifier which is a unique 32-bit IP address

Network layer header contains:

* source IP address * soure port
* destination IP address * destination port
* TTL

* identifier includes both **IP address** and **port numbers** associated with process on host

e.g HTTP server : 80
mail server : 25

* types of messages exchanged
  → request
  → response

* **open protocols**: defined in RFCs, accessible to public
* **proprietary protocols**: not openly public e.g Skype, Zoom.

Transport services that an app needs:
→ data integrity, throughput, timing, security

Internet transport protocols services

TCP service
→ reliable transport
→ flow control: sender won't overwhelm receiver
→ congestion control: throttle send when network overloaded
→ connection-oriented: setup required b/w sender/receiver

→ does not provide : timing, minimum throughput, security

UDP service :

→ unreliable data transfer

→ does not provide : reliability, flow control, congestion control, timing, (all qualities of TCP)

HTTP : hypertext transfer protocol

HTTP uses TCP, port 80

HTTP is stateless, server maintains no info about user activity except for "cookies"

* Non-persistent HTTP
  - → TCP connection opened
  - → At most one object sent over TCP connection
  - → TCP connection closed

* Persistent HTTP
  - → TCP connection opened on server
  - → multiple objects can be sent over a single TCP connection b/w client & server
  - → TCP connection closed

non-persistent HTTP : response time

RTT (round-trip time) : time for a small packet to travel from client to server & back
→ one RTT to initiate TCP connection
→ one RTT for HTTP request & first few bytes of HTTP response return
→ object/file transmission time.

non-persistent response time = 2RTT + transmission time

disadvantages of non-persistent HTTP:
→ require 2RTTs per object
→ OS overhead per TCP connection
→ browser often open multiple parallel TCP connections to fetch objects in parallel

persistent HTTP plus points:
→ as little as 1 RTT for all reference objects
→ leaves port open
→ back to back ~~response~~ request ~~from~~ server
~~objects~~

HTTP request message

POST method : user input sent from client to server
GET method : send data to server
HEAD method : request headers only
PUT method : upload file to server

* netcat
* **HTTP is stateless** : does not maintain any data or 'state' of users of a site.
* **cookies** : some transactions btw user & website is maintained

Cookie → unique ID

four components

→ cookie header line of HTTP response
→ cookie header line in next HTTP request message
→ cookie file kept on user's host, managed by user's browser
→ back end database at web site

(proxy servers)
Web Caches → to check if copies in cache are fresh, web cache pings the website server to get updation timestamps.

It is in between the client and target website server. All requests go to web cache and if it already has the copy of the web page requested, it sends user the page and doesn't have to request the actual page server.

because its a round trip → $2 \times d_{prop}$

usually negligible

$RTT = delay_{prop} + delay_{queue} + delay_{proc}$

* **caching** example:

avg data rate = 100 Kbits x 15/s

access link utilization = $\frac{1.5}{1.54} = .97$

LAN utilization = $\frac{1.5}{1\,Gbps} = 0.0015$

end-to-end delay = Internet delay + access link delay + LAN delay = 2sec + minutes + usec

data requested/s = $15/s \times \frac{100K}{bits} \Rightarrow 1.5$ Mbps

* **Conditional GET:**

client: if-modified-since

server: 304 Not Modified

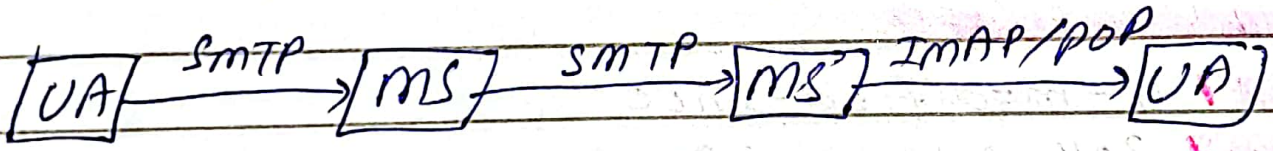
* **HTTP 1.1:** introduced multiple, pipelined GETS
→ **HOL:** head-of-line blocking: small objects have to wait behind large objects
* **HTTP/2** → uses single TCP connection
→ **push** unrequested objects to client that may be needed.

→ solves HOL blocking by dividing large objects into frames and service them in a round robin manner.

* HTTP/3: adds security, congestion control (more pipelining)

QUIC: quick UDP internet Connection

* Email
  → user agents
  → mail servers
  → simple mail Transfer protocol: SMTP

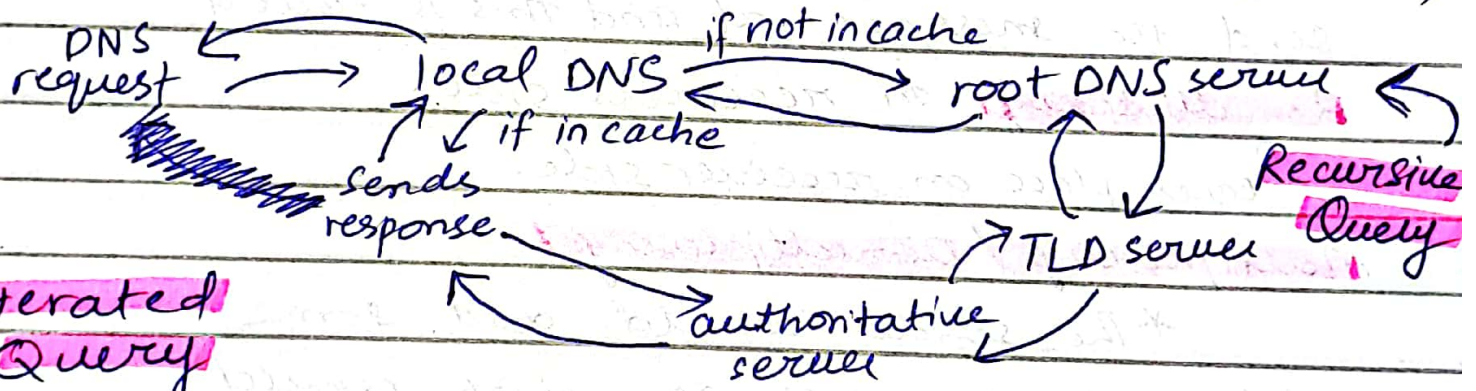


* SMTP
  → TCP 3-way-handshake
  → SMTP's own handshake

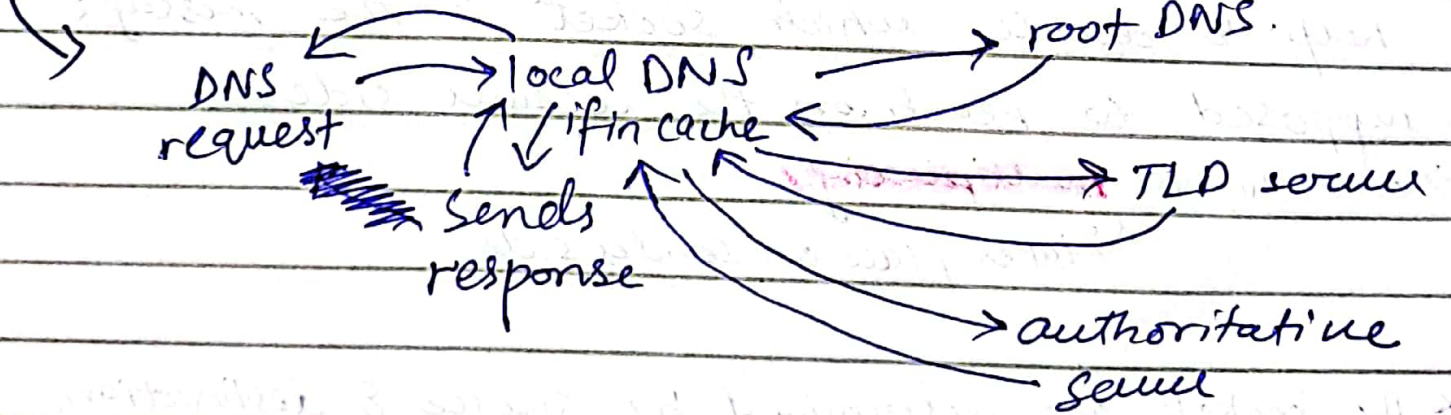
DNS : Domain Name System

* maps IP addresses and names of websites, host aliasing, mail server aliasing, load distribution
  → Distributed Database
  → Application Layer protocol
* decentralized
  → 13 DNS root servers in world.

Root servers
↓
Top level domain servers (.com, .org, .edu) etc
↓
authoritative servers (yahoo.com, amazon.com) etc



Iterated Query



DNS records are stored as RR format

RR format: (name, value, type, ttl)
- name → hostname
- value → IP
- type → A, CNAME, NS, MX

Transport layer actions.

There are many sockets b/w application layer and transport layer. When a segment reaches transport layer of receiver, the transport layer reads the transport layer header and decides which socket to send the message at and this is called demultiplexing on receivers side.

↑ takes place on receiver side.

Multiplexing / Demultiplexing:

* The sender would add some meta data with the message which would help to decide which socket is the message supposed to reach on the receiver side.
  ↳ and this is multiplexing
  ↳ takes place on sender side

* The sockets are determined by source & destination port data in transport layer header.

Connectionless demultiplexing

when it comes to UDP, the sender only needs to send dest port # and dest. IP to receiver along with payload as there's just one-way sending & no acknowledgements. sockets are one-to-one mapped

Connection-oriented demultiplexing

it takes 4 pieces of information in TCPs case the source and destination IP and port#

{Chapter 3}

* SNMP

* UDP segment header
  - → source, dest port
  - → Length of total packet, checksum ← to verify if any errors have occured

* checksum adds all UDP segment as a sequence of 16-bit integers verifies sum when received on the other end.
  - → sum the 16 bit strings, if the answer increases 16 bit range wrap around first bit

NIC → network interface card

to the whole answer, then take 1's compliment of the string (flip all bits)

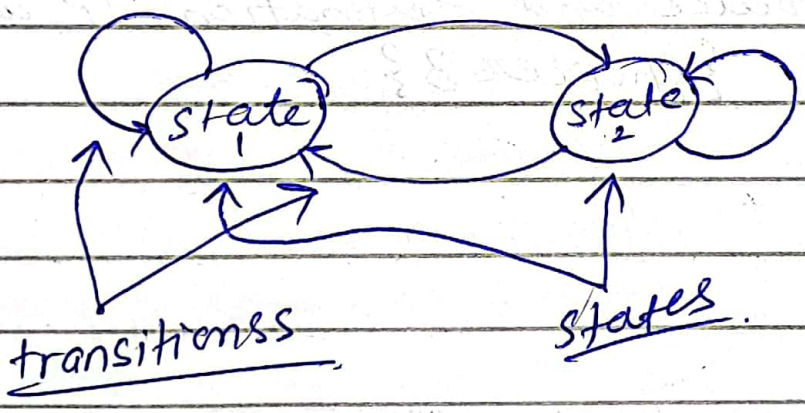
* principles of reliable data transfer (RDT).

→ interfaces

rdt_send() → udt_send()

deliver_data() ← rdt_rcv()

* Finite State Machine (FSM):



* rdt 1.0:

→ assume channel to send is reliable

→ simply send & receive

* rdt 2.0:

→ add checksum in sender side

→ send ack or neg ack from receiver side

→ send receives ack or nak ~~packet~~ &

send retransmissed the flawed data again.

→ The flaw is 'what if ack or nak are flawed?' now we add a sequence number to track duplicates.

* **rdt 2.1**:
  → added a sequence number
  → now there's a checksum for even the ACK & NAK.

* **rdt 2.2**
  only ACK is used. when a packet is corrupt. The receiver will send ACK with the sequence number of the last correct packet received.

* **rdt 3.0** (stop and wait mechanism)
  → packets will be loss in channel
  → Timeout comes to play
  → sender waits for ACK for a reasonable amount of time, if not received, assumes packet is dropped and retransmits.
  → Timeout mechanism is only on sender side.

$$\text{Utilization of Sender} = \frac{L/R}{RTT + L/R}$$

* To improve utilization we increase the window size i.e the number of packets being sent while you wait for the acks, this is applied by pipelining.

Go-Back-N: if receiver sends an ACK(5) to sender it means "all packets upto serial number 5 have been received". This is called cummulative ACKs.

Selective Repeat: if receiver sends an ACK(5) to sender it means "only packet 5 has been received" its called Selective Acks.

* TCP follows order
* window
* when timeout occurs in Go-Back-N, all packets from 0 to n are retransmitted in a window.

* Selective Repeat definitely stores packets in buffer.
* In selective repeat we maintains a timeout for each in-flight packet.

TCP

* point-to-point : one sender, one receiver
* reliable
* in-order (byte stream)
* full duplex
* MSS : maximum segment size
  ↳ window size unit
* cummulative ACKs
* pipelining : congestion & flow control ← limit of sending data
  window size is defined by these
* connection oriented

TCP packet structure

RST : reset.
SYN : start connection
FIN : end connection
Receive window : number of bytes receivable.
C, E : congestion notification bits.
sequence number : byte number
ACK number : ~~last~~ add seq # & size of payload in bytes.

* In ACK packets, the ack bit is turned on.
* Timeout is estimated using RTT by TCP.
  ↳ thus timeout keeps changing based on the RTT time.
* SampleRTT is calculated each time starting from the first connection made.

* EWMA : exponentially weighted moving avg.

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

typical value = $\alpha = 0.125$

avg of all previous

current situation

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

↳ use latest computed ERTT

typical value = $\beta = 0.25$

Timeout Interval = EstimatedRTT + 4 × DevRTT

Delayed ACK : (using cummulative ACKs)
after receiving a packet in order, the receiver waits for a certain time for the next ordered packet so that it sends the lastest ACK & doesn't have to send ~~one~~ one ACK for each received in-order packet.

modifiable value.

TCP fastretransmit : if we receive ACK for a specific packet 4 times, we assume the next packet has dropped.
(receive 3 additional ACKs for same packet)

Flow control: During the TCP 3-way handshake a sending limit is defined by the receiver.
In TCP header → receiver window (rwnd)
↑ used to specify flow control

* **RCV buffer** (typical size 4096 bytes)

* sender will only send (in-flight) packets that don't cross the RCV buffer.

* In a **2-way handshake**, a half-connection problem could occur.

* In **3-way connection**, the first message sent to establish connection contains a SynBit = 1 (ON) to indicate starting connection, in return the receiver sends an ON Synbit and ON Ackbit along with sequence & ACKnum = x+1.

* opening a TCP connection Synbit is used
* to close a TCP connection FINbit is used

Congestion Control:

* window size in TCP is not constant.
* end-to-end congestion control
  ↳ by default mechanism of TCP.
in end-to-end, the congestion is only inferred by observed loss & delay i.e the transactions b/w the sender & receiver & no intermediate entities like switch or router is used.

* network-assisted congestion control:
  C,E: congestion control bits in TCP header
  ECN: explicit congestion notification
  ↳ turned on by router when its buffer gets filled.

following are some protocols for congestion control.

TCP ECN, ATM,

1 MSS = 1 max segment size.

TCP congestion control : AIMD
- additive increase
- multiplicative decrease

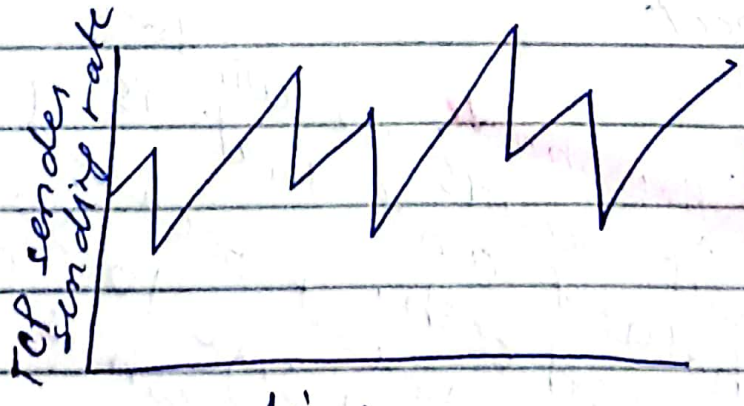
* if timeout occurs : the window size at sender side becomes 1 MSS. (TCP Tahoe)
* if 3 ACKs arrive ; the window size at sender side becomes half. (TCP reno)

* send base : the first ~~seq~~ packet in window's segment number.

---

20 bytes all segments size
send_base = 95
window size = 100 bytes

{window size} {□ □ □ □ □}
seq #'s 95 115 135 155 175



TCP saw tooth graph

* In TCP, window is called congestion window i.e cwnd.

TCP sender limits transmission:

LastByteSent − LastByteAcked ≤ cwnd

$$\text{TCP rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec.}$$

* TCP slow start ← multiplicative increase
* congestion avoidance ← additive increase

Initially increase cwnd by 1MSS
double cwnd every RTT
i.e increase cwnd on every ACK too

* Switch to congestion avoidance when cwnd gets to 1/2 of its value before timeout

its called ssthresh / slow start threshold.

* when timeout occurs the window size is halved and that is your ssthresh. e.g timeout occured at cwnd size 16 so your ssthresh is 8.

not halved. i mean the half of wnd size is the ssthresh

* whenever timeout occurs, the cwnd changes to 1 MSS & the SSthresh is recalculated.
* in 3-dup received the cwnd changes to half the size of cwnd before 3 dup & that's also the new ssthresh point.

{Chapter 4}
Network Layer : Data Plane

SDN: Software defined networking.
* in traditional routers, each router used to populate its own routing table
* in SDN, there is a central brain that does the population of tables for every router, that brain is called control brain or SDN brain.

NAT: assigned "locally" unique IPs.
PXCP protocol for NAT

* network layer takes actions on routers and end hosts.
* Network layer service model.

* Forwarding table

Address Range	Link Interface

* Longest Prefix match.
* TCAM: Ternary content addressable memory
  ↳ finds longest prefix match in 1 clock cycle
→ Cysco Catalyst : ~1M routing table entries allowed

Network layer:

→ Path-selection algos
→ IP protocol
→ ICMP protocol

} all these are run in network layer.

IP datagram format:

* 20 bytes for TCP header
* 20 bytes for IP header.

* routers have multiple interfaces
* hosts have 1 or 2 interfaces.

* dotted decimal IP address notation

Subnets: (v. imp topic)

→ divide the interface into pools which will be called islands.

CIDR: Classless Inter Domain Routing

address format: a.b.c.d /X where X is # bits in subnet portion

200. 23 . 16 . 0 /23 → 23 bits are subnet mask

11001000 . 10111 . 10000 . 0

DHCP: dynamic host config protocol

DHCP discover message

→ dynamically allocates new IP to host when it joins a network.

FLSM: Fix length Subnet Mask

VLSM: variable length Subnet mask. (imp)

Hierarchical addressing: route aggregation

ICANN: allocates IP addresses through 5 regional registries.

IPv4 classes.

The address space is divided into five classes: A to E

		binary notation
Class A	0 - 127	0
Class B	128 - 191	10
Class C	192 - 223	110
Class D	224 - 239	1110
Class E	240 - 255	1111

dotted decimal notation.

Finding the class

if 1st bit is 0 class A, else 1 then class B, ...

Slides have a diagram check that

* host id, net id
  ↳ all zeroes of your ip.

class A : first bit always 0.

* network address have all zeroes after assigned subnet mask

representation of subnet mask is like this 192.0.0.0/8
or 255.255.128.0/17 ← number of initial bits that are 1.

* fixed net subnet masking
* variable net subnet masking

Subnetting

167.199.170.82 /27.
10100111.. 1010010

NAT : network address translation

↳ you buy one globally unique ip address but be able to locally have ips that are locally unique but may not be globally unique. NAT helps in maintaining this difference within & out of subnet.

* Mapping done by storing ip of local machine against ip of gateway router & same port numbers.

IPv6

→ no checksum

* ~~[illegible] tunneling~~

* Tunneling: IPv6 datagram carried as payload in IPv4 ~~[illegible]~~ datagram among IPv4 routers. ("packet within packet").

* encapsulate IPV6 in IPv4 to tunnel through IPv4 routers.

SDN: software defining network.

* a central brain deal with data plane part of routing (i.e population of routing tables of routers)
* the forwarding or "control plane" part is still done on the routers themselves.

Destination based forwarding	Generalized forwarding
forward based on destination IP address	many other headers are used to forward a packet.

Generalized forwarding:

Flow Table: match, action.

- match: pattern value in headers (conditions to match)
- action: action on match condition.

SDN controller

OpenFlow: match, action, stats

- Flow table standard for generalized forwarding (v. imp)
- stats: packet + byte counter

ingress: which port is receiving a packet

egress: which port is sending a packet

Router, switch, firewall, NAT → openflow abstraction.

Middleboxes: any intermediary box (device) performing functions apart from normal, standard function of an IP router, they are called middleboxes e.g NAT, Firewalls, IDS, Load Balancers, application specific middleboxes like service providers, institutional, CDN, caches
↳ content distribution network
e.g Akamai

* initially middleboxes were proprietry but later moved towards "whitebox" implentry open API.
* SDN
* NFV: network functions virtualization

* **The IP hourglass**: There's only one protocol for network layer & that is IP.
→ Later new functionalities have added.

* **Architectural principles of internet**
There main points.
(1) simple connecticity
(2) IP protocol
(3) Intelligence

* The end-end argument
  ↳ data is reliably sent from end to end.
* hop by hop → data is reliably sent from each hop to another

IP fragmentation / reassembly:

MTU: max transfer unit
* network links have different MTUs based on their types
* large IP datagrams are divided into fragments when a link is not capable of transmitting that datagram.
* The fragments all have the copy of IP header and are all reassembled at the destination.

example:

datagram size = 4000 bytes
MTU = 1500 bytes
IP header size = 20 bytes

the original datagram:

data	IP header
3980 B	20 B

$3980 > MTU = 1500$

so we fragment the datagram

* check how many fragments we need:

no. of frags $= \frac{3980}{1500-20} = 2.6 \approx 3$

① | 1480 B | 20 B | $3980 - 1480 = 2500$

② | 1480 B | 20 B | $2500 - 1480 = 1020$

③ | 1020 B | 20 B |

IP Header

original datagram | length = 4000 | ID = x | frag flag = 0 | offset = 0 |

after fragmentation

① | length = 1500 | ID = x | frag flag = 1 | offset = 0 |

② | length = 1500 | ID = x | frag flag = 1 | offset = $\frac{1480}{8}$ = 185 |

③ | length = 1040 | ID = x | frag flag = 0 | offset = 370 |

* if a fragment has another fragment ahead of it, the **frag flag = 1** else if it is the last fragment **frag flag = 0**.
* The **offset** is defined in **blocks of 8 bytes**, so divide the data behind a fragment by 8 to decide its offset from 0.

---

mid 2 topics. (imp)

Destination based forwarding
→ longest prefix match
→ subnet masking < fixed / variable

TCP
→ RDT (versions).
→ congestion control ← flags: SS, CA, FR

slow start, fast recovery,
↳ on 3ACK, come to cwnd half.

* Reno & Tahoe

Open flow → Generalized Forwarding
→ SDN

*

{ Chapter 5 }

* routing protocols ← link state e.g dijkstra
  distance vector e.g Bellman Ford
* 2 approaches to structure network control plane:
  - per-router control (traditional
  - logically centralized control (SDN)

Per-router control

* every router runs a routing algorithm

logically centralized control

* A remote controller computes, installs forwarding table in routers. (algo runs on a central controller)

* Routing protocol goal: determine good paths from sender to receiver
* good means something like "cost" "least "fastest", "least congested"

Graph abstraction : link costs

$c_{w,z} = 5$ (direct link cost b/w w & z)

$c_{v,z} = \infty$ (no direct link b/w v & z)

small c means direct link cost

* cost defined by network operator e.g could all be 1, inversely related to bandwidth or to congestion.

* if a router has all network topology & costs of all links it falls into "link state" algos.

{global} algos ↗ ↘ {decentralized} algos

* iterative process of computation, exchange of info with neighbours : "distance vector"

* "static"
* "dynamic"

1 billionth after

hash

* Dijkstra's link-state routing algo

→ **Centralized**: network topology & link costs known
accomplished via "link state broadcast"

Notation

$c_{x \to y}$: direct link cost from x to y

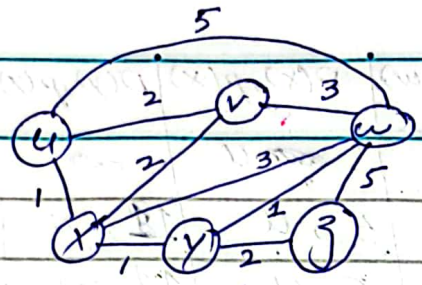
distance
↳ $D(v)$: current estimate of cost of least-cost-path from source to dest 'v'.

$p(v)$: predecessor node along path from source to 'v'.

N': set of nodes whose least-cost-path definitively known.

$$D(v) = \min\left(D(v), D(w) + c_{w,v}\right)$$

Date: _____

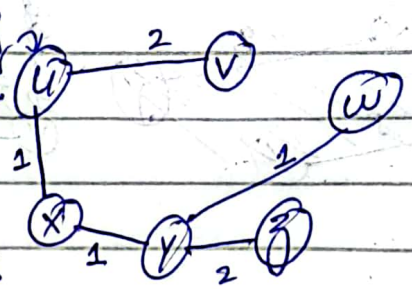


Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1, u	∞	∞
1	u, x	2, u	4, x	//	2, x	∞
2	u, x, y	2, u	3, y	//	//	4, y
3	u, x, y, v	2, u	3, y	//	//	4, y
4	u, x, y, w, v	2, u	3, y	//	//	4, y
5						

$D(v) = \min(2, 1+2) = 2$

$D(w) = \min(5, 1+3) = 4$

$D(y) = \min(\text{inf}, 1+1) = 2$



forwarding table for u

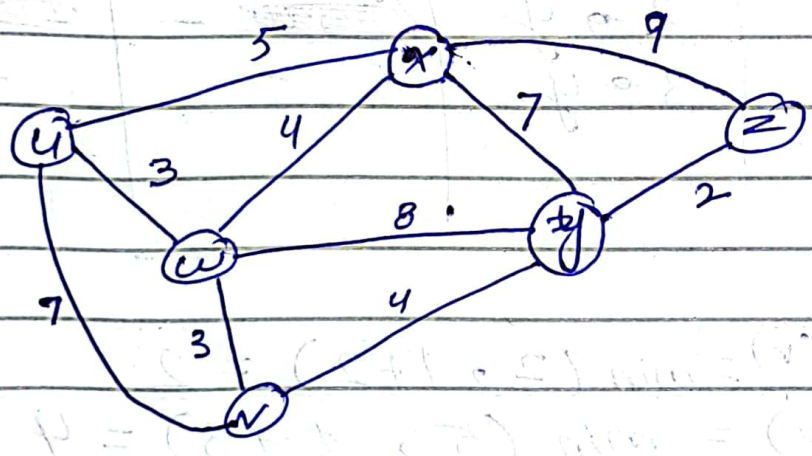
dest	outgoing link
v	(u, v)
x	(u, x)
y	(u, x)
w	(u, x)
z	(u, x)

shortest path tree formed after dijkstra

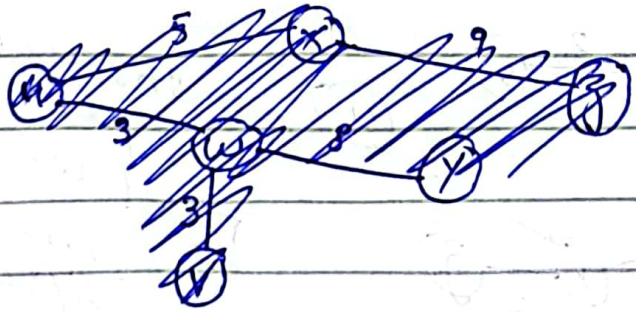
gigame... 1 billion' ether ... has

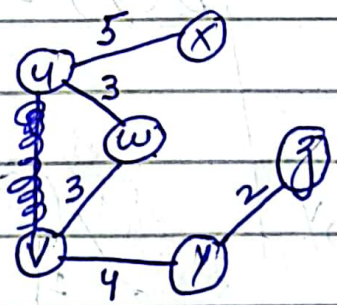
i = ...

Steps	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	7, u	3, u	5, u	∞	∞
1	u, w	6, w	3, u	5, u	11, w	∞
2	u, w, x	6, w	3, u	5, u	11, w	14, x
3	u, w, x, v	6, w	3, u	5, u	10, v	14, x
4	u, w, x, v, y	6, w	3, u	5, u	10, v	12, y
5	u, w, x, v, y, z	6, w	3, u	5, u	10, v	12, y



least-cost path tree:





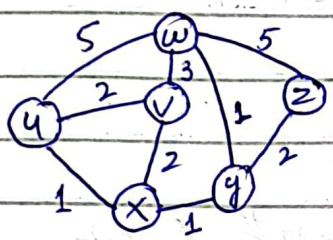
* Distance Vector

→ Bellman-Ford (BF)

→ Let $D_x(y)$: cost of least-cost path from x to y.

Then : $D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$

($c_{x,v}$: direct cost of link from x to v; $D_v(y)$: v's estimated least-cost path to y)



good news → travels fast

bad news → travels slow

* count to infinity problem, black holing

$D_x(y)$ → estimated cost of least-cost path from x to y

$d_x(y)$ → actual cost of least-cost path from x to y

* OSPF (intra-Autonomous System routing)

* BGP ("inter"-Autonomous system routing)

cont'd

hash

1 billionth ether

* OSPF : routing within a domain (AS) / network

* BGP : inter-AS / among networks

↳ algorithms

① RIP : Routing Info Protocol
↳ uses classic D.V

② EIGRP
↳ uses DV

③ OSPF : open shortest path first
↳ uses dijkstra
* floods network picture.

* BGP : Border Gateway Protocol.
↳ eBGP : obtain dest reachability from neighbor ASs
↳ iBGP : propagate reachability to all AS-internal routers.

* BGP session : a semi-permanent TCP connection established btw two peer BGP routers

* path vectors.

* BGP protocol messages: OPEN, UPDATE, KEEPALIVE, NOTIFICATION.

{Chapter 6}

* Link layer wraps datagram into a **frame**
* Services provided by link layer:
  → **Framing** → **Link Access (MAC)** → **Reliable delivery** → **Error detection & correction**
* **MAC**: Medium Access Control
  ↳ makes possible for multiple nodes accessing one link to access without collisions.
* Link layer is implemented majorly in **hardware**, specifically the **network adapter** i.e **NIC** (network interface controller). part of link layer runs on **CPU** (i.e **software**)
* Error detection & Correction techniques.
  → **Parity Checks**: **even** parity, **odd** parity
  single bit

* in **even parity** you add one bit (0 or 1) such that if you count all the 1s in the bit stream along with the parity bit, the count is even.
* in **odd parity** you add one bit (0 or 1) such that if you count all the 1s in the bit stream along with the parity bit, the count is odd.

e.g 1 0 1 0 1 | 1 (even parity)
parity bit

1 0 1 0 1 0 (odd parity)
parity bit

* 2-D parity check

1 0 1 0 1	1
1 1 1 1 0	0
0 1 1 1 0	1
0 0 1 0 1	0

even parity
2-D

Here we'll check parity row-wise & column wise. This method makes error detection & correction much easier and specific. e.g

1 0 1 0 1	1	✓
1 0 1 1 0	0	✗
0 1 1 1 0	1	✓
0 0 1 0 1	0	
✓ ✗ ✓ ✓ ✓		

wrong bit

So here we came to know exactly which bit is wrong because of 2-D structure.

* Forward Error Correction (FEC):

The ability of the receiver to both detect & correct error is called FEC.

* Checksumming Methods:

→ Internet Checksum

- low overhead but weak protection against errors.

* Cyclic Redundancy Check (CRC)

→ The sender & receiver agree upon a bit pattern 'r+1' where the leftmost bit is 1, it is called a generator.

→ Now, if you add this generator to the data, and divide that by G, the answer should be ~~about~~ zero

→ receiver will identify errors based on the resulting non-zero value of the mathematical formula.

→ In general binary arithmatic, multiplication by 2^k left shifts a bit pattern by k places.

so, $D \cdot 2^r$ XOR R yields d+r

($D \cdot 2^r$: data bits, bit pattern; R: additional bits.)

→ we want to compute R such that:

$$D \cdot 2^r \text{ XOR } R = nG$$

$$D \cdot 2^r = nG \text{ XOR } R$$

$$R = \text{remainder } \frac{D \cdot 2^r}{G}$$

* see solved example in book pg 460, figure 6.7.

* CRC standards can detect burst errors of fewer than r+1 bits. (mcq)

cont'd

1 billionth after

hash is

* **multiple access links & protocols**

→ protocols for point-to-point links: ① PPP (point-to-point-protocol) ② high-level data-link control (HDLC)

→ broadcast links: ① Ethernet ② wireless LAN

* **multiple access problem**: how to coordinate the access of multiple sending & receiving nodes to a shared broadcast channel.

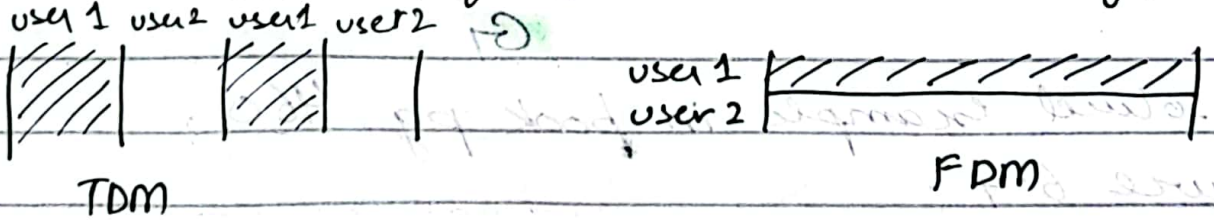
→ to solve this problem we have multiple access protocols, following are some categories of multiple access protocols.

① channel partitioning protocols
② random access protocols
③ taking turns protocols.

* **Channel Partitioning Protocols**

→ TDM (Time division multiplexing) or TDMA

→ FDM (frequency division multiplexing) or FDMA



* **Slotted ALOHA**

→ efficiency = $Np(1-p)^{N-1}$

where p is the probability of success and N is the number of nodes.

→ the efficiency of slotted ALOHA is $\frac{1}{e} = 0.37$

37 % and 26 % are collisions

→ To obtain maximum efficiency for N active nodes, we find p^* that maximizes the expression.

max efficiency = $Np^*(1-p^*)^{N-1}$

* **Pure ALOHA**

→ no syncronization

→ maximum efficiency is 18 % : 1/2e

→ probability of a single node success = $p(1-p)^{2(N-1)}$

* **CSMA : carrier sense multiple access**

→ keeps checking other nodes & transmits data based on that

→ CSMA/CD : collision detection

when collision is detected, the colliding nodes stop transmitting & wait random time before retransmitting.

→ If a node has experienced 'n' collisions the 'K' i.e interval would be $\{0, 1, 2, \cdots, 2^n - 1\}$ wait K·512 bit times

* more collisions: larger backoff interval
* less collisions; shortest backoff interval

CSMA/CD efficiency $= 1/(1+\{5d_{prop}/d_{trans}\})$

* **Taking Turns:**
  - → **Polling**: round robin, a master node allots certain number of frames to other nodes one by one in a round-robin fashion.
  - → **token-passing**: A small frame called a **token** is passed along the network. The node which needs to send frames keeps the token & sends the maximum fixed number of frames & then forwards it to another node in network.

LANS:

* **MAC addresses**
  - → 32-bit, 6 bytes long, 2^{48} possible addresses
  - → MAC broadcast address: FF-FF-FF-FF-FF-FF
* **Address Resolution Protocol (ARP):**
  - → converts IP into corresponding MAC address within a single subnet: ARP table has IP, MAC, TTL.
  - → when TTL ends, the entry is removed from table.
  - → ARP protocol makes a packet "ARP packet" which is sent to all nodes in network to find the MAC of specified IP. (ARP query)

has indeed been one enduring constant that has remained unchanged over 30 years—Ethernet's frame format. Perhaps this then is the one true and timeless centerpiece of the Ethernet standard.

6.4.3 Link-Layer Switches

Up until this point, we have been purposefully vague about what a switch actually do... and how it works. The role of the switch is to receive incoming link-layer frames and forward them onto outgoing links; we'll study this forwarding function in detail in this subsection. We'll see that the switch itself is **transparent** to the hosts and routers in the subnet; that is, a host/router addresses a frame to another host/router (rather than addressing the frame to the switch) and happily sends the frame into the LAN, unaware that a switch will be receiving the frame and forwarding it. The rate at which frames arrive to any one of the switch's output interfaces may temporarily exceed the link capacity of that interface. To accommodate this problem, switch output interfaces have buffers, in much the same way that router output interfaces have buffers for datagrams. Let's now take a closer look at how switches operate.

Forwarding and Filtering

Filtering is the switch function that determines whether a frame should be forwarded to some interface or should just be dropped. **Forwarding** is the switch function that determines the interfaces to which a frame should be directed, and then moves the frame to those interfaces. Switch filtering and forwarding are done with a **switch table**. The switch table contains entries for some, but not necessarily all, of the hosts and routers on a LAN. An entry in the switch table contains (1) a MAC address, (2) the switch interface that leads toward that MAC address, and (3) the time at which the entry was placed in the table. An example switch table for the uppermost switch in Figure 6.15 is shown in Figure 6.22. This description of frame forwarding may sound similar to our discussion of datagram forwarding

Address	Interface	Time
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Figure 6.22 ♦ Portion of a switch table for the uppermost switch in Figure 6.15

in Chapter 4. Indeed, in our discussion of generalized forwarding in Section 4.4, we learned that many modern packet switches can be configured to forward on the basis of layer-2 destination MAC addresses (i.e., function as a layer-2 switch) or layer-3 IP destination addresses (i.e., function as a layer-3 router). Nonetheless, we'll make the important distinction that switches forward packets based on MAC addresses rather than on IP addresses. We will also see that a traditional (i.e., in a non-SDN context) switch table is constructed in a very different manner from a router's forwarding table.

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.
- 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.

Let's walk through these rules for the uppermost switch in Figure 6.15 and its switch table in Figure 6.22. Suppose that a frame with destination address 62-FE-F7-11-89-A3 arrives at the switch from interface 1. The switch examines its table and sees that the destination is on the LAN segment connected to interface 1 (that is, Electrical Engineering). This means that the frame has already been broadcast on the LAN segment that contains the destination. The switch therefore filters (that is, discards) the frame. Now suppose a frame with the same destination address arrives from interface 2. The switch again examines its table and sees that the destination is in the direction of interface 1; it therefore forwards the frame to the output buffer preceding interface 1. It should be clear from this example that as long as the switch table is complete and accurate, the switch forwards frames toward destinations without any broadcasting.

In this sense, a switch is "smarter" than a hub. But how does this switch table get configured in the first place? Are there link-layer equivalents to network-layer routing protocols? Or must an overworked manager manually configure the switch table?

Self-Learning

A switch has the wonderful property (particularly for the already-overworked network administrator) that its table is built automatically, dynamically, and autonomously—without any intervention from a network administrator or from a configuration protocol. In other words, switches are **self-learning**. This capability is accomplished as follows:

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.

Let's walk through the self-learning property for the uppermost switch in Figure 6.15 and its corresponding switch table in Figure 6.22. Suppose at time 9:39 a frame with source address 01-12-23-34-45-56 arrives from interface 2. Suppose that this address is not in the switch table. Then the switch adds a new entry to the table, as shown in Figure 6.23.

Continuing with this same example, suppose that the aging time for this switch is 60 minutes, and no frames with source address 62-FE-F7-11-89-A3 arrive to the switch between 9:32 and 10:32. Then at time 10:32, the switch removes this address from its table.

Address	Interface	Time
01-12-23-34-45-56	2	9:39
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Figure 6.23 ♦ Switch learns about the location of an adapter with address 01-12-23-34-45-56

Switches are **plug-and-play devices** because they require no intervention from a network administrator or user. A network administrator wanting to install a switch need do nothing more than connect the LAN segments to the switch interfaces. The administrator need not configure the switch tables at the time of installation or when a host is removed from one of the LAN segments. Switches are also full-duplex, meaning any switch interface can send and receive at the same time.

Properties of Link-Layer Switching

Having described the basic operation of a link-layer switch, let's now consider their features and properties. We can identify several advantages of using switches, rather than broadcast links such as buses or hub-based star topologies:

- *Elimination of collisions.* In a LAN built from switches (and without hubs), there is no wasted bandwidth due to collisions! The switches buffer frames and never transmit more than one frame on a segment at any one time. As with a router, the maximum aggregate throughput of a switch is the sum of all the switch interface rates. Thus, switches provide a significant performance improvement over LANs with broadcast links.
- *Heterogeneous links.* Because a switch isolates one link from another, the different links in the LAN can operate at different speeds and can run over different media. For example, the uppermost switch in Figure 6.15 might have three1 Gbps 1000BASE-T copper links, two 100 Mbps 100BASE-FX fiber links, and one 100BASE-T copper link. Thus, a switch is ideal for mixing legacy equipment with new equipment.
- *Management.* In addition to providing enhanced security (see sidebar on Focus on Security), a switch also eases network management. For example, if an adapter malfunctions and continually sends Ethernet frames (called a jabbering adapter), a switch can detect the problem and internally disconnect the malfunctioning adapter. With this feature, the network administrator need not get out of bed and drive back to work in order to correct the problem. Similarly, a cable cut disconnects only that host that was using the cut cable to connect to the switch. In the days of coaxial cable, many a network manager spent hours "walking the line" (or more accurately, "crawling the floor") to find the cable break that brought down the entire network. Switches also gather statistics on bandwidth usage, collision rates, and traffic types, and make this information available to the network manager. This information can be used to debug and correct problems, and to plan how the LAN should evolve in the future. Researchers are exploring adding yet more management functionality into Ethernet LANs in prototype deployments [Casado 2007; Koponen 2011].

FOCUS ON SECURITY

SNIFFING A SWITCHED LAN: SWITCH POISONING

When a host is connected to a switch, it typically only receives frames that are intended for it. For example, consider a switched LAN in Figure 6.17. When host A sends a frame to host B, and there is an entry for host B in the switch table, then the switch will forward the frame *only* to host B. If host C happens to be running a sniffer, host C will not be able to sniff this A-to-B frame. Thus, in a switched-LAN environment (in contrast to a broadcast link environment such as 802.11 LANs or hub–based Ethernet LANs), it is more difficult for an attacker to sniff frames. *However*, because the switch broadcasts frames that have destination addresses that are not in the switch table, the sniffer at C can still sniff some frames that are not intended for C. Furthermore, a sniffer will be able sniff all Ethernet broadcast frames with broadcast destination address FF–FF–FF–FF–FF–FF. A well-known attack against a switch, called **switch poisoning**, is to send tons of packets to the switch with many different bogus source MAC addresses, thereby filling the switch table with bogus entries and leaving no room for the MAC addresses of the legitimate hosts. This causes the switch to broadcast most frames, which can then be picked up by the sniffer [Skoudis 2006]. As this attack is rather involved even for a sophisticated attacker, switches are significantly less vulnerable to sniffing than are hubs and wireless LANs.

Switches Versus Routers

As we learned in Chapter 4, routers are store-and-forward packet switches that forward packets using network-layer addresses. Although a switch is also a store-and-forward packet switch, it is fundamentally different from a router in that it forwards packets using MAC addresses. Whereas a router is a layer-3 packet switch, a switch is a layer-2 packet switch. Recall, however, that we learned in Section 4.4 that modern switches using the "match plus action" operation can be used to forward a layer-2 frame based on the frame's destination MAC address, as well as a layer-3 datagram using the datagram's destination IP address. Indeed, we saw that switches using the OpenFlow standard can perform generalized packet forwarding based on any of eleven different frame, datagram, and transport-layer header fields.

Even though switches and routers are fundamentally different, network administrators must often choose between them when installing an interconnection device. For example, for the network in Figure 6.15, the network administrator could just as easily have used a router instead of a switch to connect the department LANs, servers, and internet gateway router. Indeed, a router would permit interdepartmental communication without creating collisions. Given that both switches and routers are candidates for interconnection devices, what are the pros and cons of the two approaches?