CS2105 Cheatsheet AY24/25 -@Jin Hang

Internet: Network of connected hosts/end systems.

- Network edge: End hosts, servers, etc.
- Service: email, ssh, ftp, rdp, ndp, www
- Internet is a **Network of Networks**
  - Hosts connect to Internet via access ISPs (Service Providers)
  - Access ISPs in turn must be interconnected.

Protocols: Organized into "layers", defines

- Simple interfaces between layers, hide details from each other
- format and order of messages exchanged among network entities
- actions taken upon receiving or sending the messages

Access Networks

- Hosts access the Internet through access network
- Enterprise: Hosts typically connect to Ethernet switch
- Wireless access network connects hosts to router

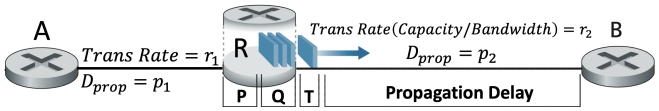
Network Core Mesh of interconnected routers (ISPs, Routers...)

Circuit Switching: End-end resources allocated to and reserved for "call" between source & dest (traditional telephone networks)

- call setup required
- circuit-like (guaranteed) performance
- circuit segment idle if not used by call (no sharing)

Packet Switching: breaks message into chunks (packets) of length L bits $\rightarrow$ transmits at rate $R \rightarrow$ Receiver reassembles them

- Entire packet must arrive before transmitted on the next link
- Excessive congestion is possible
- **Processing delay:** Check bit errors; determine output link
- **Queuing delay:** Waiting in queue for transmission
- **Transmission delay:** Time taken to push bits onto link (L/R)
- **Propagation delay:** Time for bits to travel in link (d/s)
- **End-to-end packet delay:** Time to travel from src to dst
- **Throughput:** Bits transmittable per unit time for end-to-end communication



Sends m packets from Host A to B, each packet is b bits long

- Consider only Transmission Delay $t_k = k \max\{\frac{b}{r_1}, \frac{b}{r_2}\} + \min\{.\}$
- Consider Trans and Prop. Try to use mathematics induction

Packet

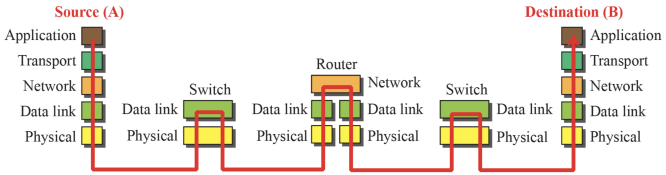
- Each packet needs to carry source and destination information
- Users' packets share network resources

Packet Loss: Packets queue in router buffers, arrive full $\rightarrow$ drop

- Lost packet may be retransmitted by previous node, by source host, or not at all.

Layering $[H_n|H_t|msg]$

- Application - message
- Transport - segment $\mathbf{H_t}$ contains src and dest port
- Network - datagram $\mathbf{H_n}$ contains src and dest IP address
- Link - frame



Application Layer

App. Protocol	Tpt. Protocol	Port
HTTP	TCP	80 (default)
HTTPS	TCP	443 (default)
DNS	UDP	53
SMTP	TCP	25
DHCP	UDP	67 (svr) 68 (client)
RIP	UDP	520

- Protocol used by every Internet Application
- Applications run on hosts and contains communicating processes

Architecture: Client-Server, Peer-to-Peer(e.g. chat), Hybrid

IP Address: Globally Unique, identifies the host

- IPv4(32-bit): dotted decimal notation
- IPv6(128-bit): Hexadecimal notation

Port Number: Locally Unique, identifies the process

Web and HTTP

1. Uniform Resource Locator(URL) - locate resources
   - Web objects are addressable by a url (host N + path N)
2. HyperText Transfer Protocol(HTTP) - access those resource
   - Web resources are requested and received using HTTP
   - HTTP uses TCP as transport service
3. Web Page: HTML + other objects(Transmit after HTML)
4. Client/server model
   - client: usually browser requests, receives and displays obj.
   - server: Web server sends objects in response to requests

HTTP 1.0 closes connection after transmitting single object (RFC1945)

- OS overhead for each TCP connection
- browsers often open parallel TCP conn. to fetch referenced obj.

HTTP 1.1 uses persistent connection by default (possibly with pipelining) - (RFC2616)

- Subsequent HTTP msg. between same client/server sent over the same TCP connection
- client sends requests as soon as it encounters a referenced object (persistent with pipelining)
- Pipelining: RTT + RTT + T(HTML) + RTT + N * T(obj)

HTTP 2.0 Multiplexing, response can in any order, even partially.

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

HTTP response time: (NP) 2 * RTT+T(file)

- 1 RTT to establish TCP connection (Transport layer)
- 1 RTT for HTTP request and first few bytes of HTTP response

HTTP Status Code

- 200 Ok
- 301 Moved Permanently
- 304 Not Modified
- 403 Forbidden
- 404 Not Found
- 500 Internet Server Error

HTTP Request Header

```
GET /cs2105/demo.html HTTP/1.1 /*Request Type:  GET*/
Host:  www.comp.nus.edu.sg
User-Agent:  Mozilla/5.0
Connection:  close
Cookie:  name=value; name2=value2; name3=value3
```

HTTP Response Header

```
HTTP/1.1 200 OK \r\n
Date: Wed, 01 Jul 2015 08:47:52 GMT \r\n
Server: Apache/2.4.6 (Unix) OpenSSL/1.0.1m \r\n
Accept-Ranges: bytes \r\n
Connection: Keep-Alive \r\n
Content-Length: 73 \r\n
Content-Type: text/html \r\n
Keep-Alive: timeout=5, max=100 \r\n
\r\n
```

Cookie: http messages carry "state"

- HTTP Server maintains no info. about past client requests.
- Cookie file kept on user's host, managed by user's browser

Conditional GET: Don't send object (response contains no object) if (client) cache has up-to-date cached version

`If-Modified-Since:  Thu, 15 Jan 2018 13:02:41 GMT`

Server may reply with `HTTP/1.0 304 Not Modified`

Domain Name System Internet's primary directory service

- Map between host names and IP
- A client must carry out a DNS query to determine the IP address corresponding to the server name before the connection
- Stores Resouce Records (RR) `<name, value, type, TTL>` in distributed DB implemented in a hierarchy of name servers

Type	Name	Value
A	Hostname	IP Address
NS	Domain (`nus.edu.sg`)	hostname of authoritative NS
CNAME	Alias for real name (`www.comp.nus.edu.sg`)	Real name
MX	Domain of email address (`xyz@gmail.com`)	Name of mail server managing the domain

- 13 root servers globally that answer NS queries for TLDs

Authoritative servers: Organization's own DNS server(s) provides authoritative hostname to IP mappings for organization's named hosts (e.g. Web, mail)

Top-level domain (TLD) servers: .com, .org, .net, .edu & all top-level country domains

Local DNS Server: Not strictly hierarchical, Each ISP has one

DNS query: (Recursive/Iterative) Query $\rightarrow$ local DNS server $\rightarrow$ Retrieve name-to-address translation from local cache $\rightarrow$ if not found locally $\rightarrow$ acts as proxy and forwards query into hierarchy.

DNS cache: Once a name server learns a mapping, it caches it

- Not permanent, DNS servers discard cached info. after a period
- If a pair is cached $\rightarrow$ another query arrives to the DNS server for the same hostname $\rightarrow$ provide the desired IP address, even if it is not authoritative for the hostname
- Update/notify mechanisms proposed: RFC 2136

DNS Cache Poisoning: Rogue DNS records are introduced into a DNS resolver's cache, causing name server return an incorrect IP, diverting traffic to the attacker/other's computer.

Transport Layer TCP uses service provided by IP

- Processes in different hosts communicate by exchanging messages according to protocols
- **Sender** breaks app message into segments (as needed), passes them to network layer (aka IP layer).
- **Receiver** reassembles segments into msg, pass it to app layer.
- **Packet switches** (routers) in between: only check destination IP address to decide routing.

Goal: Deliver packet to right host and to right process

Multiplexing: Data from multiple sockets to be sent using only one "transmission channel"

Socket: Abstraction interface between processes and transport layer protocols.

- Applications/Process sends/receives messages through socket.
- Programming-wise: a set of API calls

UDP (User Datagram Protocol) RFC768

- Datagram abstraction
- Connection-less
- Unreliable: data may be lost, corrupted or received out-of-order $\Rightarrow$ App implements error detection and recovery mechanisms
- Usually used in small # of pkt(DNS, SNMP), Video, Audio..

De-multiplexing: IP datagrams (from different src) with the same dest port# will be directed to same UDP socket at dest

UDP Socket (Datagram Socket) Only one socket is needed

- Communication with anyone through same socket
- Application creates each packet specifies recipient (IP+Port)
- OS will attach return info (source IP + port)
- Receiver **identifies** sender by source IP+port from packet
- Run `Client.py` before `Server.py` ⇒ Work but no data

```
import socket                                   # Python's socket library

address = ('localhost', 8888)

# Create a socket                               # IPv4, UDP Socket
sock = socket.socket(socket.AF_INET,
                     socket.SOCK_DGRAM)
# bind socket to local port
sock.bind(address)

while True:                                     # receive datagram buffer size (bytes)
    # read packet from client
    data, addr = sock.recvfrom(4096)

    print(f'{data} from {addr}')

    # sends data to client
    sock.sendto(data, addr)
```

```
import socket

address = ('localhost', 8888)

# Create a socket
sock = socket.socket(socket.AF_INET,
                     socket.SOCK_DGRAM)
# no binding
text = input('Enter message:')
# sends data to client                          # convert str to bytes
sock.sendto(text.encode(), address)

# read packet from server
data, addr = sock.recvfrom(4096)

print(f'{data.decode()} from {addr}')           # convert bytes to str
sock.close()
```

TCP (Transmission Control Protocol)

- Stream abstraction
- Connection-oriented
- De-mux: TCP con./socket is identified by 4-tuple, receiver uses all 4 values to direct a segment to appropriate socket.
- Reliable, in order byte stream: Pass data to TCP → Forms segments in view of MSS (Maximum Segment Size)
- Flow control and congestion control: Prevent unnecessary losses Receiver buffers data to application ⇒ Tells sender how much data it can send (by Receive Window, `rwnd`)
- **Fast Retransmission** [RFC 2001] 3 Duplicate ACKs ⇒ resend

TCP Socket (Stream Socket)

- A process establishes a connection to another process.
- Data stream flows between processes during connection
- Run `Client.py` before `Server.py` ⇒ Exception

Server	Client

from socket import socket
address = ("localhost", 8888)
# create a socket to listen 	
# default (AF_INET, SOCK_STREAM) 	
welcome_socket = socket()
welcome_socket.bind(address)
# set max backlog to 1 	
welcome_socket.listen(1)
while True:
  # accept incoming client
  conn, addr = welcome_socket.accept()
  print(f"Client connected from {addr}")
  # create wrapper on input stream
  in_file = conn.makefile('r')
  text = []
  # read a line until blank line
  while (data := in_file.readline()) != '\n':
  text.append(data)
  print(repr(data))
 	
  # write text to socket
  conn.sendall(''.join(text).encode())
  # close the socket
  conn.shutdown(socket.SHUT_RDWR)
  conn.close()
	import socket
	# create a socket to connect
	# default (AF_INET, SOCK_STREAM)
	client_socket = socket.socket()
	client_socket.connect(("localhost", 8888))
	
	# get input for user
	while text := input():
	  # write text to socket
	  client_socket.send(text.encode())
	  client_socket.send(b'\r\n')
	  # flush the socket with sendall
	  client_socket.sendall(b'\r\n')
	  print('Done. Ready to read?')
	  input()
	
	while data := client_socket.recv(10):
	  print(repr(data))
	
	# close the socket
	client_socket.close()

TCP Server: Creates a welcome or listening socket. When contacted by client, forks **a new socket** for server process to communicate with client. ⇒ n Clients → $n+1$ Sockets

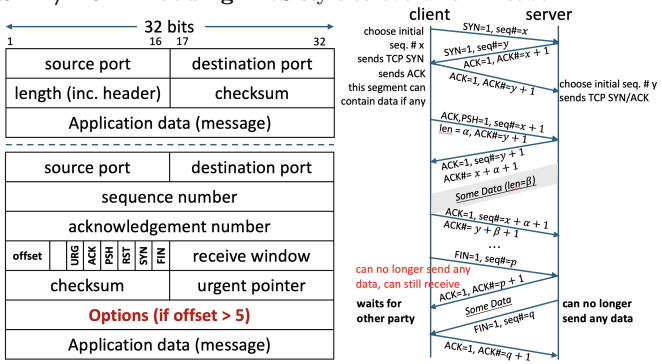
TCP Client: Creates a socket to establish connection with server. Every connection has its own socket instance → Multiple sockets

TCP Receiver:

- **Delay ACK:** Wait up to 500ms for next segment. ACK both together(Maximal 2)
- Segment partially or completely fills gap → Send cumulative ACK immediately

SYN Flooding: DoS style attack by sending SYN

SYN/ACK Flooding: DoS style to overwhelm network



- ACK: Indicate if ACK# is valid
- Seq#: **Byte number** of the first byte of data in the segment
- Ack.#: Seq.# of the next byte of data expected
  TCP only ACKs up to the missing byte (cumulative)

Estimated RTT: $R_\varepsilon = (1-\alpha)R_\varepsilon + \alpha R_s$, Typically $\alpha = \frac{1}{8}$

Deviation of RTT $R_{dev} = (1-\beta)R_{dev} + \beta|R_s - R_\varepsilon|$ $\beta_t = \frac{1}{4}$

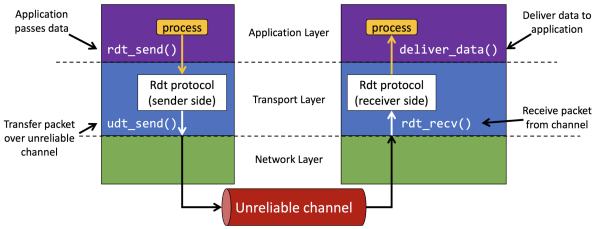
Retransmisson Time Out is set to $R_\varepsilon + 4R_{dev}$

MSS and MTU

- # pkt = Len(File) / MSS
- MTU = MSS + IP header + ... + TCP header
- Total = # pkt * MTU
- Transmit Delay = Total(bits) / Transmission Rate

UDP/TCP Checksum (RFC 768) Split segment into 16-bit integers → Add (wrap around carry) → Compute 1's complement

Reliable Delivery Transfer (rdt)



rdt 1.0 Assume underlying channel is reliable

rdt 2.0 Assume underlying channel may flip bits in packets

- Receiver may use **checksum** to detect bit errors
- Acknowledgements (ACKs) ⇒ Packet received is OK
  Negative ack. (NAKs) ⇒ Sender retransmits
- If ACK is corrupted ⇒ Duplicate packet

rdt 2.1 Add a sequence number to each packet

- Receiver discards (doesn't deliver up) duplicate packet.

rdt 2.2 Replace NAK with ACK of last correctly received packet

- Receiver must explicitly include seq. # of packet being ACKed
- Duplicate ACKs at sender results ⇒ retransmit current pkt
- Duplicate pkt → discard, send ACK

rdt 3.0 Assume packet can be lost or corrupted

- Sender waits "reasonable" amount of time for ACK → retransmits if no ACK received and timeout
- checksum, ACK/NAK, seq num, timeout/re-transmit

Utilization: $U_{\text{sender}} = \frac{\text{Sending Time}}{\text{Total Time}} = \frac{d_{\text{trans}}}{d_{\text{trans}}+RTT}$

Stop-n-Wait: (d_{trans} + RTT) for each packet. Win-Size=1

Pipelined Protocols: Rrange of sequence numbers must be increased; Buffering at sender and/or receiver

Go-Back-N: ACK n means all packets $\leq n$ have been received

- Sliding Window of size n keep track of n unACKed packets → Time Out → Retransmit all packets
- **GBN Sender** up to N unACKed packets; insert k-bits seq.# in packet header; keep **a** timer for the **oldest** unACKed packet.
- **GBN Receiver** only ACK packets that arrive in order; discard out-of-order packets and ACK the **last in-order** seq.#.

Selective Repeat: Only retransmit the unACK pkt not all pkt

- Receiver individually ACK all correctly received packets
- Buffers **out-of-order** packets for eventual in-order delivery
  Sliding Window [i|ACK|..] → Not move until packet i received.

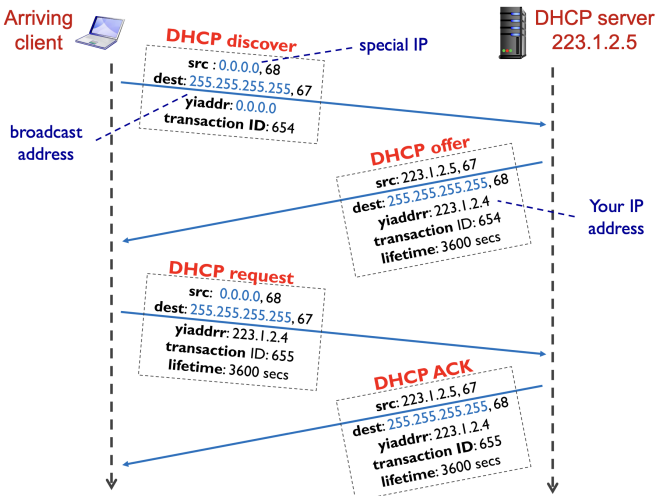
Next Seq# Problem

Assume Every packet k bits long, send pkt with seq# N, Sliding Window Size is S ⇒ Induction: $0 \leq N \leq 2^k - 1$

- GBN Next Seq# Range: $[N, (N+S)\%2^k]$
- SR Next Seq# Range: $[(2^k + N - S + 1)\%2^k, (N+S)\%2^k]$

Dynamic Host Configuration Protocol

- IP address is renewable and allow reuse of addresses
- Provide additional network info (IP address of local DNS server, Network mask, IP address of first-hop router)



Network Layer

- **Forward:** move packets from appropriate next hop. [Data] Use **longest** address prefix that matches dest addr in forward table (Ternary Content Addressable Mem)
- **Route:** Determining src-dest route with min cost. [Control]
  - Intra-AS routing: RIP, OSPF; Inter-AS routing: BGP
  - Traditional routing algorithms: implemented in routers
  - Software-Defined Networking: in (remote) servers

Bellman-Ford Equation: $D_\alpha(z) = \min_{a \in N}\{c(\alpha, a) + D_a(z)\}$

- $C_{x,y}$: cost of link between routers x and y
- $D_x(y)$: cost of the least-cost path from x to y
- N : 1-hop Neighborhood
- Each neighbour a sends its distance vector (z, k) to α

Distance vector algorithm: Loop

1. wait for (change in local link cost or msg from neighbor)
2. recompute DV estimates using DV received from neighbor
3. if DV to any destination has changed, notify neighbors

Routing Information Protocol (RIP)

- Exchange routing table every 30 seconds over UDP port 520

- If no update from a neighbour for 3 min, assume neighbour fail
- Routers periodically broadcast link costs to each other.

IP address: Globally Unique 32-bit identifier for Host/Router
- Associated with every interface of host or router
- Get by manually configured or from DHCP
- x-bits Prefix, $32-x$-bits host ID $\Rightarrow 2^{32-x}$ subnet IP $\Rightarrow$ $2^{32-x}-2$ Host (1st for network addr, last for broadcast)
- Special Addresses

0.0.0.0/8	Local subnet (non-routable)		
127.0.0.0/8	Loopback inside the host (localhost)		
255.255.255.255/32	Broadcast (within subnet)		
10.0.0.0/8	172.16.0.0/12	192.168.0.0/16	Private

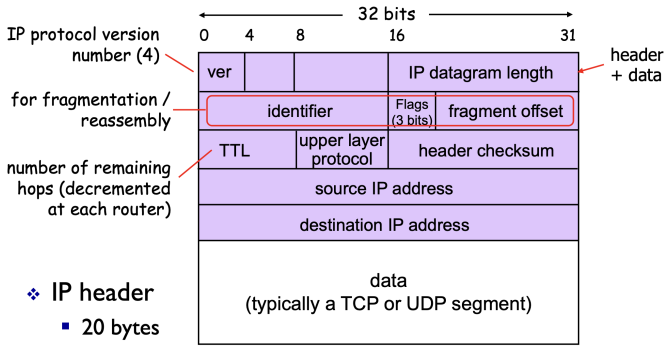
Private IP addresses: Not globally unique used in organization.
- Un-routable on backbone Internet; Routable within organization

Address Aggregation: Reduce the size of the forwarding table by a systematic IP address allocation.

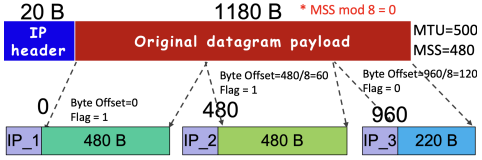
Subnet: Network formed by a group of **directly** interconnected hosts. Same Subnet $\Leftrightarrow$ Same Network Prefix
- Hosts in same subnet can reach each other without router.
- Connect to the outside world through a router.
- # of subnets: detach each interface from its host/router $\rightarrow$ create islands of isolated networks

Subnet mask: Set subnet prefix bits to 1, host ID bits to 0

IPv4 Datagram Format

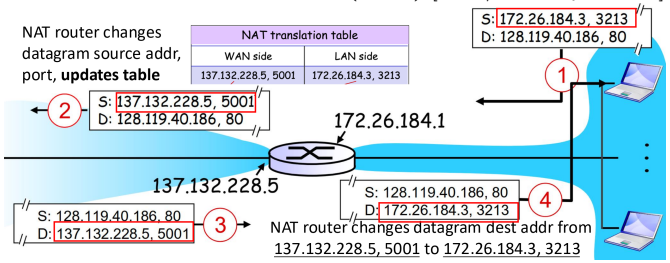


- Different links have different MTU $\rightarrow$ fragment IP datagrams



  - frag. flag set to 1 if next frag. from same segment
  - Offset is in expressed in units of 8-bytes $\Rightarrow$ MSS mod 8 = 0
- Total data transferred increases due to extra IP headers

Network Address Translation (NAT) [NAT/Gateway Router]



- Change ISP without changing addr of hosts in local network.
- Simultaneous Connections: 2^{16} port number
- **Internal** Device can connect to External Device. **External** Device cannot connect to Internal without port# forwarding.

Internet Control Message Protocol (ICMP)
- Error reporting: unreachable host/network/port/protocol
- Echo request/reply (used by ping)
- Messages carried in IP datagrams; Header starts after IP header
- Header: Type + Code(Sub type) + Checksum + others

Service model: "best effort", no guarantees successful delivery/time/order/bandwidth available to end-end flow.

Link Layer: sends datagram between adjacent nodes.
- Encapsulate IP datagram into frame, adding header and trailer.
- Different link-layer protocols may be used on different links.
- Implemented in "adapter" (Network Interface Card) or on a chip (e.g., Ethernet card, Wi-Fi adapter)
  Adapters: semi-autonomous, implement link & physical layers
- **Point-to-Point:** : Point-to-Point Protocol (PPP), Serial Line Internet Protocol (SLIP). No need for multiple access control.
- **Broadcast:** When a node transmits a frame, broadcasts the frame and every other node receives a copy via shared medium.

Possible Service: Error detection/correction/Reliable delivery
- **Reliable delivery:** Seldom used on low bit-error link (e.g., fiber) but often used on error-prone links (e.g., wireless link).
- **Detection:** May retransmission/**Correction:** No need

Multiple Access Protocols: Coordination about channel sharing must use channel itself. No out-of-band channel signal.

Decentralized: No special node to coordinate transmissions. Node failure will not lead to system failure.

Random Access Protocols
- **Slotted ALOHA**
  - All frames of equal size L with transmit rate R
  - Time divided into slots of length L/R (1 slot = 1 frame)
  - Nodes start to transmit only at the beginning of a slot
  - Time is synchronized at each node.

  Operations:
  - Listens to the channel while transmitting (detect collision)
  - If collision, re-transmit frame in each subsequent slot with prob p until success. (N nodes, P(Success)=$p(1-p)^{N-1}$)
- **Pure (unslotted) ALOHA:** No time slots/Synchronization
  - Transmits the entire frame immediately.
  - If collision, wait for 1 frame transmission time and retransmit the frame with probability p until success.
  - P(collision) $\uparrow$: Frame t_0 collides with others (t_0-1, t_0+1) Given a node, P(Success)=$p(1-p)^{2(N-1)}$
- **Carrier Sense Multiple Access:** listen before transmit
  - Sensed idle: transmit entire frame
  - Sensed busy: defer transmission, wait till channel is idle
  - Collisions can still occur due to **propagation delay** (two nodes may not hear each other's transmission immediately)
- **CSMA/CD (Collision Detection):** Abort transmission when collision is detected. Retransmit after a random delay.
  - **Minimum frame size:** Collision may not detected for small frames due to propagation delay. $[2\max(d_{prop}) \le d_{trans}]$
  - Restrict on "MFS" to ensure collision always detected. *Ethernet requires a minimum frame size of 64 B(bytes).*

  Backoff Algorithm: After m-th collision,
  - Choose K at random from $\{0, \ldots, 2^m-1\}$, $p = 1/2^m$
  - Wait K time units before retransmission. For Ethernet 1 time unit is 512 bit transmission times
  - Give up transmit after 16 times failure.
- **CSMA/CA:** Check Collision using **ACK**

Tanking Turns Protocols
- **Polling:** Round Robin; master node [inform slave nodes maximum # of frames can transmit $\rightarrow$ slave transmit] in turn (polling overhead; single point of failure of master node)
- **Token passing:** Control token is passed from one node to next sequentially (token overhead; single point of failure (lost token))
  - Sends up to a max # of frames and then forwards the token
  - Token loss can be disruptive. Node failure can break the ring

Channel Partitioning Protocols
- **TDMA (time division multiple access)**
  - Each node gets fixed length time slots in each round. Length of time slot = Data Frame Transmission Time
  - Frame in TDMA: Collection of N time slots
- **FDMA (frequency division multiple access)**
  - Channel partitioning by frequency band. Time is not limit.

Protocol	Collision	Efficient	Fair	D.C
S-ALO	non-Free	✓ $N\uparrow$ ✗	✓	✓
P-ALO	non-Free	✓ $N\uparrow$ ✗	✓	✓
CSMA(/CD)	non-Free	✓	✓	✓
Polling	Free	High	✓	✗
Token	Free	High	✓	✓
[T/F]DMA	Free	✗	✓	✓

- Throughput: Slot ALOHA > Pure, CSMA/CD > CSMA
- Frequently Trans: Token passing/TDMA > CSMA

Error detection: Checksum/Parity Checking/CRC

Error Detection/Correction Bits: Larger EDC field yields better detection (even correction)

Parity Checking
- Errors often clustered tgt. in "bursts", P(undetected) $\rightarrow$ 50%
- Single: Suppose that the information to be sent, D, has d bits.
  - Sender simply includes one additional bit (chooses its value s.t. # of 1s in the $d+1$ bits is even)
  - can detect odd # of error but not even #
- 2-demensional: d bits divided into i rows and j columns. Compute parity value each row/column. (Odd-1, Even-0)
  - Could detect and correct 1 bit error in the same group
  - Only detect (not correct) 2/Odd bit error in the same group
  - May not detect ≥ 4 Even# bit error.

Cyclic Redundancy Check: $D-d$ bits data, $R-r$ bits CRC
1. Append r 0's to RHS of D. Use XOR for +/− during division
2. $D\%G = R \rightarrow$ Sender sends (D, R) [Append R to D]
3. Receiver knows G, divides (D, R) by $G \rightarrow$ non-zero for error. (either in the data bits D and/or in the CRC bits R.)

- Easy to implement on hardware
- Powerful error-detection coding (e.g., Ethernet, Wi-Fi)
- Detect all odd # of single bit errors
  - CRC of r bits can detect all burst errors of up to r bits.
  - All burst errors of greater than r bits with $p = 1-0.5^r$
  - **Polynomial code:** $\sum b_{k-1} \cdot x^{k-1}$

Local Area Network (LAN) Network interconnects computers within a geographical area.
- IBM Token Ring: IEEE 802.5
- Wi-Fi: IEEE 802.11
- Ethernet: IEEE 802.3
- Async. Transfer Mode(ATM)

Ethernet: "Dominant" wired LAN technology (Simpler/Cheaper)
- MAC protocol and frame format remain unchanged

- Sending NIC encapsulates IP datagram in **Ethernet Frame**.

8 bytes	6	6	2	46 - 1500	4
Preamble	Dest Addr	Src Addr	Type	Data	CRC

- NIC receives a frame with matching dest. addr. or broadcast addr.(FF:FF..) → passes frame data to network layer protocol
- Type: Higher layer protocol (multiplex networklayer protocol)
- Preamble: `AA AA AA ... AB` - Sync receiver/sender clock rate

Ethernet Data Delivery: 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.
- Multiple ccess protocol: CSMA/CD with binary backoff

Topology
- Bus: all nodes can collide with each other
- Star: Nodes are directly connected to a hub/switch.

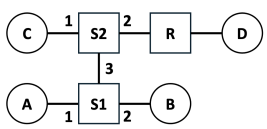
Hub: physical-layer device
- acts on individual bits rather than frames.
- Very slow and not ideal for larger networks (Collisions)

Ethernet Switch: Link layer device used in LAN
- Examine incoming **frame**'s MAC address
- Selectively forward frame to one-or-more outgoing links.
- Use CSMA/CD to access each incoming link (**no collisions**)
- Hosts are unaware of presence of switches
- Buffer packets and transmit simultaneously
- Switch table: `<Host MAC addr,interface to reach host,TTL>`

Plug-and-play (Self-Learning): do not need to be configured
- Records sender/location pair in switch table when receive frame
- When receive frame, index in switch table
  - Find entry → Drop if dest addr = its src addr.
  - Not find → Forward on all interfaces except arriving interface

MAC Address: Adapter(NIC) use it to send/receive frames.
- 48 bits, burned in NIC read-only-memory, **Permanent**
- Allocated by IEEE, first 3 bytes identifies vendor
- Find using `ifconfig eth0` → `ether XX:XX...`

Address Resolution Protocol (ARP): [RFC 826]
- Each IP node has an ARP table `<IP addr; MAC addr; TTL>`



Switch				ARP		
S1		S2		A	C	D
A	1	A	3	B	A	R
B	2	B	3	C	B	
C	3	C	1	R	R	
R	3	R	2			

Suppose NAT-enabled: $C \to D$
Send (ipC, MAC-C, ipD, MAC-D)
Reply (ipD, MAC-D, ipR, MAC-R)

Empty ARP/Switch Table $A \to C$
Without ARP Query
$A \to S1 \to ((S2 \to (C,R)),B)$

Sending Frame in the Same Subnet
- Source issue an ARP query → Switch → Broadcast to Subnet
- Node with correct IP address will respond with its MAC address
- Reply sent back to src MAC address. Source update ARP Table

Sending Frame to Other Subnet
- Frame with dest. router MAC addr → Switch → Router
- Router checks dest. IP of **datagram** and decides to forward.
- Encapsulates IP datagram in a new frame with new src(router) /dest MAC addr (from ARP) and send to external subnet

Network Security
- Confidentiality: Intended receiver should "understand" message
- Message integrity: Message not altered without detection
- Authentication: Sender, Receiver confirm identity of each other

Trade-off: Absolute security is easy but impossible to achieve

Cryptography
- Plaintext message m • Ciphertext $K_A(m)$: Encrypted Message
- Encryption algorithm $K_A(\cdot)$ with key K_A
- Decryption algorithm $K_B(\cdot)$ with key $K_B \Rightarrow K_B(K_A(m)) = m$
- Key: A string of numbers or characters, provided as input parameter to the encryption/decryption algorithm

Breaking an encryption scheme
- Ciphertext only attack: Only get ciphertext to analyze
- Known-plaintext attack: Get m corresponding to $K_A(m)$
- Chosen-plaintext attack: Get $K_A(m)$ for chosen m

Monoalphabetic Cipher: Substitute one letter for another (26!)

Polyalphabetic Encryption: Use n substitution ciphers, $C_1, C_2, \ldots, C_n$ and cycling pattern

Block Ciphers: m to be encrypted is processed in blocks of K bits. Number of keys 2^K!
- DES: Data Encryption Standard, US encryption standard [NIST 1993]. 56-bit symmetric key, 64-bit block
  - Faster than RSA, but needs prior knowledge of Key K_S
- AES: Advanced Encryption Standard, Symmetric-key NIST standard. 128 bit blocks; 128, 192, or 256 bit keys

1. Symmetric Key Cryptography: $K_A = K_B$
- The key K_S is unique to a pair of individuals.
- Requires sender, receiver know shared secret key

2. Asymmetric(Public) Key Cryptography: $K_A \neq K_B$
- Sender uses a public encryption key known to all
- Receiver uses a private decryption key known only to receiver

RSA Algorithm: $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$
- Choose two large prime numbers p, q. $n = pq, z = (p-1)(q-1)$
- Choose $e < n$ has no common factors with z
- Choose d such that $ed \bmod z = 1$
- $K_B^- = (n, e), K_B^+ = (n, d) \Rightarrow (m^e \bmod n)^d \bmod n = m$
- Encrypt message: $c = m^e \bmod n$
- Decrypt received bit pattern: compute $c^d \bmod n$

Session Key:
- Select a Key K_S and use RSA to transfer $K_S \to K_B^+(K_S)$
- Receiver decrypt → $K_B^-(K_B^+(K_S)) = K_S$
- Use K_S as symmetric key in DES for encrypting data for this session

Message Integrity: Cryptographic Hash Function (SHA-2/3)
- Send $(m, H(m))$: × ← Replace by $(m', H(m'))$
- Send $(m, H(m+s))$: Sender/Receiver share Authentication key $s \Rightarrow$ Receiver generate $H(m+s)$ and compare with Sender's Message Authentication Code: $H(m+s)$
- Example: Timestamping: Proof of work without revealing work Password hashing: Passwords are not stored in plaintext but hashed and stored ⇒ cannot be recovered but only reset

Digital Signature: Send sign doc ⇒ Indicate he is owner
- Verifiable: Receiver can check if signature and message was generated by Sender.
- Unforgeable: No one, other than sender should be able to generate the signature and the message.

1. Simple Signature: Send $(m, K_B^-(m))$ → Receiver compare m and $K_B^+(K_B^-(m))$. Expensive for long message.
2. Optimization: Send $(m, K_B^-(H(m)))$ → Receiver generate $H(m)$ and compare with $K_B^+(K_B^-(H(m)))$.

Public Key Infrastructure (PKI): share PK securely
- **Certificate:** Digital doc contains identity of an owner, PK of owner, time window certificate is valid and signature of CA
- **Certification Authority (CA):** Issues and signs digital certificates to websites. Maintains a directory of public keys.
- CA binds public key to particular entity E. E registers its public key with CA. ⇒ Receiver apply Sender's CA and apply CA public key to get Sender's PK

Secure e-mail: Send $((m, K_S(K_A^-(H(m)))), K_B^+(K_S)) \to$ Receiver get $K_S \to H(m)$ and generate $H(m) \to$ Compare $H(m)$

Firewalls: Isolates organization's internal net from larger Internet, allowing some packets to pass, blocking others
1. Internal network connected to Internet via router firewall
2. Router filters packet-by-packet by src/dest IP addr, TCP/UDP src/dest port#, ICMP message type, TCP SYN and ACK bits.

IP spoofing: Router can't know if data "really" comes from claimed source ⇒ Can become a bottleneck.

Access Control Lists (ACL): Table of rules applied top to bottom to packets, (action, condition) pairs.

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

Common Command
- `dig`: Domain Information Groper
- `nslookup`: Get information from DNS server
- `telnet` connect to remote systems over a TCP/IP network check if `comp.nus.edu.sg` is listening on port 80
- `ping` check network connectivity between host and server
- `traceroute` sends a series of ICMP pkt across a network with different TTL to display **route** of messages to a remote host.
- `curl`: Retrieve the HTML content of the specified URL and display it in the terminal.

-*-*-*-*-*-*- PLEASE DELETE THIS PAGE! -*-*-*-*-*-*-

Information
Course:
Type: Cheat Sheet
Date: May 22, 2025
Author: QIU JINHANG
Link: https://github.com/jhqiu21/Notes

-*-*-*-*-*-*- PLEASE DELETE THIS PAGE! -*-*-*-*-*-*-