

Internet Architecture

Entities

- Hosts: end systems (e.g. laptops, smartphones)
- Communication links (e.g. radio satellite, fiber, copper)
- Packet switches (e.g. routers, switches)
- Network edge: Hosts
- Network Core: interconnected routers

Delays

- Store-and-forward: entire packet must arrive at a router before it can be transmitted on next link
- **End-to-End** delay: total time taken to send one packet from src to dest
 $d_{\text{node}} = d_{\text{prop}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{proc}}$
- **Transmission** delay = L/R
- **Queuing** delay: Total = $n(n-1)(L/(2R))$

Latency (s)	Bandwidth (bps)	Throughput (bps)
Time for 1 bit of data to travel from one endpoint to another	Maximum (capacity) number of bits that can be transmitted per unit time	Number of bits transmitted per unit time

- End-to-end throughput = transmission rate of bottleneck link
- Average and instantaneous throughput is independent of file size

Application Layer

- Resides in end-systems
- IP addresses: uniquely identifies host
- Port: identifies a process/service running on the host
- Socket: software interface to allow applications to send and receive messages

HTTP Protocol

- Hypertext Transfer Protocol
- Uses TCP, port **80**
- Stateless

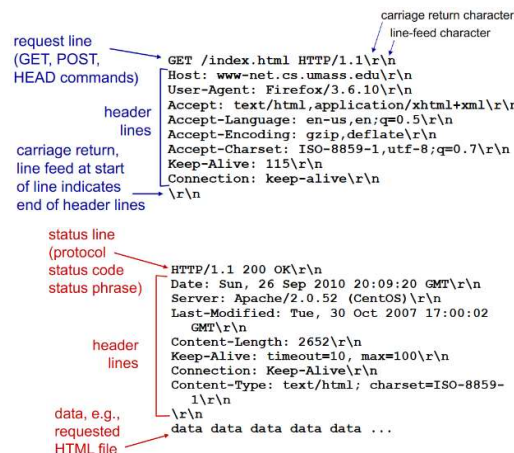
Non-Persistent HTTP

- Non-pipelined:
 1. Initiate TCP connection (1 RTT)
 2. GET HTML file (1 RTT + trans)
 3. Connection closes
 4. Repeat for each n additional objects
- Pipelined (m files at once):
 1. Initiate TCP connection (1 RTT)
 2. GET HTML file (1 RTT + trans)
 3. Connection closes
 4. Repeat for $\lceil n/m \rceil$ times:
 - a. Initiate TCP connection (1 RTT)
 - b. GET m objects (1 RTT + m trans)
 - c. Connection closes

Persistent HTTP

- Non-pipelined:
 1. Initiate TCP connection (1 RTT)
 2. GET HTML file (1 RTT + trans)
 3. Repeat for n times:
 - 3.1. GET object (1 RTT + trans)
 4. Connection closes
- Pipelined (m files at once):
 1. Initiate TCP connection (1 RTT)
 2. GET HTML file (1 RTT + trans)
 3. Repeat for $\lceil n/m \rceil$ times:
 - 3.1. GET m objects (1 RTT + m trans)
 4. Connection closes

HTTP Request and Response



Cookies

1. Client sends HTTP request to server for the first time
2. Server responds with cookie-ID in response header
3. Client accepts cookie?
 - 3.1. Yes: Cookie saved in client until TTL expires. Server creates entry in backend database
4. [If accepted] Client sends next HTTP request, with cookie-ID in request header
5. [If accepted] Server stores and/or retrieves user-specific information from database using the cookie-ID and sends user-specific response

Web proxies

1. Client sends request to proxy server
2. Proxy server checks if response is cached:
 - 2.2. Cached: response with cached response. Done
 - 2.3. Step 3
3. Proxy server forwards original request to origin server
4. Proxy server caches the response from origin server and responds back to client

- **Conditional GET**: Allows proxy server to query origin server to check if the cached response data has changed since it was last cached
 1. Client sends GET request to proxy server
 2. Proxy server sends Conditional GET request to origin server with "If-Modified-Since: <date>" in header, which specifies the datetime the proxy last received (last cached) the requested resource
 3. Origin server responds with:
 - 3.1. **304 Not Modified** if If-Modified-Since is before Last-Modified. **Empty response body**
 - 3.2. **200 OK** if If-Modified-Since is after Last-Modified
 4. Proxy server updates the new response (updates If-Modified-Since and data, if modified) and responds back to client

DNS

- Domain Name System
- Maps host name to unique IP address
- Uses UDP, port **53**
- Local DNS servers do not belong to the global hierarchy of distributed servers
- Root DNS servers: top most in the hierarchy
- Top-level domain (TLD) servers: .com, .net, .edu etc.
- Authoritative DNS servers: organisations' servers e.g. .nus
- DNS resolution:
 1. Host makes request to Local DNS server
 2. Cached?
 - 2.2. Yes: Responds host. Done
 - 2.3. No: Step 3
 3. Local DNS server forwards query to root DNS server
 4. Root DNS server responds local DNS server with the next lower level DNS server X it should query next (e.g. .com)
 5. Local DNS server queries DNS server X...
 6. Repeat iteratively until host name is resolved
- Local DNS servers process DNS queries in a recursive manner, where as the distributed servers do so in an iterative manner
- `dig t a`` command to query local DNS server

Socket Programming

- UDP: `SOCK_DGRAM`; TCP: `SOCK_STREAM`
- TCP: client calls `.connect((server_ip, port))` to establish handshake
 - Server not running: Exception thrown
- UDP: uses `sendto()` with destination addr and port as parameters
- TCP: uses `send()`
- Client's port is automatically chosen by OS; Server needs to bind to correct port with `.bind()`
- Client can call `.bind()` to manually choose which port to use

Transport Layer

- Provides process-to-process communication between hosts
- Resides in end-systems

UDP Protocol

- User Datagram Protocol
- Unreliable data transfer; connection-less
- Not full duplex
- UDP socket is identified by (dest IP, dest Port)

UDP header	
Source Port (2 bytes)	Dest Port (2 bytes)
Length (2 bytes)	Checksum (2 bytes)

- Both length and checksum includes header (in bytes)
- Checksum:
 - Sender: 1’s complement addition with every 16-bit integers (i.e. includes carry-out, inverting bits after sum)
 - Receiver: sum every 16-bit integer, then add the sum to checksum. If all bits of result is 1, no error

Principles of Reliable Data Transfer

- Utilisation: fraction of time sender busy sending (window size of n)
$$U = \frac{nL/R}{RTT + L/R}$$

Stop-and-Wait Protocols

- Low utilisation

rdt1.0

- Assumptions: underlying channel is perfectly reliable
- **Sender:**
 - Sends packet
- **Receiver:**
 - Receives packet

rdt2.0 (Checksum, ACK/NAK)

- Assumptions: packets may be corrupted
- Checksum: detect errors
- ACK/NAK: recover from errors
- **Sender:**
 - Sends packet
 - ACK/NAK is corrupted or NAK: retransmit
- **Receiver:**
 - Packet is not corrupted: Responds with ACK
 - Packet is corrupted: Responds with NAK
- Flaw: Does not deal with duplicate packets received at the receiver

rdt2.1 (Checksum, ACK/NAK, seq #)

- seq #: receiver can detect duplicates from retransmission by sender
- **Sender:**
 - Sends packet
 - ACK/NAK is corrupted or NAK: retransmit
- **Receiver:**
 - Packet is not corrupted: Responds with ACK
 - Packet is corrupted: Responds with NAK
 - Packet is a duplicate: Discards packet and responds with ACK

rdt2.2 (Checksum, ACK, seq #)

- **Sender:**
 - Sends packet
 - ACK is corrupted or duplicate: retransmit
- **Receiver:**
 - Responds with ACK # = last packet received
 - Packet is a duplicate: Discards packet and responds with ACK # = last packet received

rdt3.0 (Checksum, ACK, seq #, Timer)

- Assumptions: bits may be flipped and packets can be lost
- **Sender:**
 - Sends packet
 - Timeout: retransmit
 - ACK is corrupted or duplicate: ignore, do nothing
- **Receiver:**
 - Responds with ACK # = last packet received
 - Packet is a duplicate: Responds with ACK # = last packet received
 - Packet is corrupted: Do nothing (rely on timeout at sender)
- May introduce packet reordering; solution: use larger range of sequence numbers

Pipelined Protocols

Go-Back-N

- Uses cumulative ACK
- Single timer; timer is for the oldest unACKed packet (packet at base)
- **Sender**
 - Window will always contain only unACKed packets
 - ACK $n \rightarrow$ advance window base to $n + 1$, then sends packets newly added into the window
 - ACK is corrupted or duplicate: ignore, do nothing
 - Timeout: retransmit all packets in the window
- **Receiver**
 - Window size = 1
 - Responds with ACK # = last correctly received, in-order packet
 - Packet is corrupted, duplicate or out-of-order: discard, responds with appropriate ACK

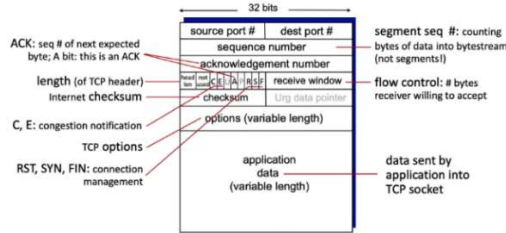
Selective Repeat

- Uses individual ACKs
- Uses individual timers
- **Sender**
 - Window can contain both unACKed and ACKed packets
 - Window base is always at the oldest unACKed packet
 - ACK $base\# \rightarrow$ advance window base to # of oldest unACKed packet
 - Sends packets newly added into the window
 - ACK not in window \rightarrow ignore, do nothing
 - ACK is corrupted or duplicate: ignore, do nothing
 - Timeout: retransmit single packet
- **Receiver**
 - Packet is uncorrupted: Responds with ACK # = # of packet
 - Packet is corrupted: discard, no ACK is sent

- Packet is out-of-order but uncorrupted: buffer, then ACK
- Packet is in-order (pkt# = rcv_base): delivers all the x buffered, consecutive packets beginning with rcv_base and advance window base to rcv_base + x
- Packet is duplicate/outside of window: discard, then ACK

TCP Protocol

- Transmission Control Protocol
- Reliable data transfer; connection-oriented
- TCP socket is identified by (src IP, src Port, dest IP, dest Port)
- Full duplex, point-to-point



Data transfer

- MSS = maximum size of payload
- Seq # = i^{th} position of the first byte in payload in full data; need not start at 0
- ACK # = expected next seq #
- Uses cumulative ACK
- Single timer; timer is for the oldest unACKed packet (in the window)
- **Sender**
 - Window will always contain only unACKed packets
 - ACK $n \rightarrow$ advance window base to $n + 1$, then sends packets newly added into the window
 - Timeout: retransmit single packet
- **Receiver**
 - Packet is out-of-order: buffer or discard, then respond with duplicate ACK
 - Packet is in-order: responds with appropriate ACK
 - Packet is corrupted: discard, then respond with duplicate ACK
- Congestion control: timeout interval doubles after each timeout for each packet
- Fast retransmit: if sender receives 4 consecutive duplicate ACKs, retransmit the corresponding packet

Receiver	Action	
Arrival of in-order packet with expected seq #	Delayed ACK (500ms) and wait for next packet. If no packet, send ACK	
Arrival of in-order packet with expected seq #, the one before has pending ACK	Immediately send single cumulative ACK, essentially ACK-ing both in-order packets with one ACK	
Arrival of out-of-order packet higher than expected seq #	Immediately send duplicate ACK	
Arrival of packet that partially/completely fills gap	Immediately send ACK	

Timeout Estimation

$TimeoutInterval = EstimatedRTT + 4 \times DevRTT$

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

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

- Typically, $\alpha = 0.125, \beta = 0.25$

Three-way Handshake

1. Client sends SYN segment with SYN = 1 (seq # = client_isn)
2. Server sends SYNACK segment with SYN = 1 (seq # = server_isn, ACK # = client_isn + 1)
3. Client sends segment with SYN = 0 (seq # = client_isn + 1, ACK # = server_isn + 1)

Connection Termination

1. Client sends segment with FIN = 1 (seq # = x)
 2. Server responds with segment with ACKbit = 1 (ACK # = x + 1)
 3. Server sends segment with FIN = 1 (seq # = y)
 4. Client responds with segment with ACKbit = 1 (ACK # = y + 1)
 5. Connection is terminated after time wait of $2 \times \text{max segment lifetime}$
- Either client or server can initiate the connection termination
 - Once a FIN segment is send, the host can no longer send application data except for retransmitting segments sent before the FIN segment

Flow Control

- **Receiver**
 - Maintains LastByteRcvd, LastByteRead, RcvBuffer
 - Amount of data in buffer = LastByteRcvd - LastByteRead
 - Attaches rwnd = RcvBuffer - (LastByteRcvd - LastByteRead) to ACK packet
- **Sender**
 - Maintains rwnd, LastByteSent, LastByteAcked
 - Amount of data sent = LastByteSent - LastByteAcked
 - Ensures that LastByteSent - LastByteAcked \leq rwnd whenever it wants to send a segment
- When rwnd = 0, sender continues to send 1 byte to receiver
 - Receive buffer is full: byte is dropped
 - Receive buffer is not full: Receiver sends ACK segment, letting sender know that buffer is not full

Network Layer

- Provides host-to-host communication
- Resides in network core
- **Forwarding:** move packets to appropriate router output port
- **Routing:** determine route taken from source to dest

IP address

- Per interface
- 32-bits
- a.b.c.d/x, x = subnet prefix
- Subnet mask: x MSB 1s, 32-x 0s

- 0.0.0.0/8: non-routable meta-address
- 255.255.255.255/32: broadcast address

Subnets

- Hosts can connect to one another without a router
- Hosts have same network prefix
- First and last IP address in subnet are reserved

DHCP Protocol

- Dynamic Host Configuration Protocol
 - Application layer protocol
 - Dynamically obtain IP address when host joins a network
 - Also returns address of first-hop router, name and IP address of DNS server, and network mask
 - Uses UDP, port **68** for client, port **67** for server
 - ID identifies the client
1. DHCPDISCOVER: client A sends
<srcIP: **0.0.0.0#68**, destIP: **255.255.255,255#67**, srcMAC: A, destMAC: FF-FF-FF-FF-FF-FF, yiaddr: **0.0.0.0**, ID>
 2. DHCPOFFER: server B responds
<src: X.X.X.X#67, dest: **255.255.255,255#68**, srcMAC: B, destMAC: A, yiaddr: Y.Y.Y.Y, ID, lease time>
 3. DHCPREQUEST: client sends
<src: **0.0.0.0#68**, dest: **255.255.255,255#67**, srcMAC: A, destMAC: FF-FF-FF-FF-FF-FF, yiaddr: Y.Y.Y.Y, ID + 1, lease time>
 4. DHCPACK: server responds
<src: X.X.X.X#67, dest: **255.255.255,255#68**, srcMAC: B, destMAC: A, yiaddr: Y.Y.Y.Y, ID + 1, lease time>
- DHCP discover and offer are not mandatory
 - Also returns address of first-hop router, name and IP of DNS server and network mask
 - Client chooses any offer should there be multiple servers repounding

IP Protocol

- Internet
-
- IP datagram structure diagram showing fields: ver, IHL, identifier, flags, fragment offset, TTL, source IP address, destination IP address, and data. Annotations include: IP protocol version number (4), for fragmentation / reassembly, upper layer protocol, header checksum, and data (typically a TCP or UDP segment).

Fragmentation

- Max Fragment size = MTU of link-level frame
- Max payload size in fragment = MTU - 20 bytes (copied IP header)
- # fragments = $\lceil \text{IP payload size} / \text{MTU} \rceil$
- **Flag:** 0 if is the last fragment, else 1
- **Offset:** Units of 8-bytes
- **ID:** ID of original datagram
- Recalculates checksum for each fragment header
- Reassembled at destination host

Intra-Autonomous Systems Routing

- Autonomous system: a group of routers under a common admin domain
- Forwarding table stores <Dest IP addr range, Next Hop> entries
- Longest prefix matching
- **Distance Vector Routing Algorithm**
 - Each router X knows the cost $C_{X,Y}$ of one hop
 - Each router X stores a routing table containing distance vectors $D_X(W)$ to all other routers in the AS network
 - Initial $D_X(W) = C_{X,W}$ if W is one-hop away, else $D_X(W) = \infty$
 - When any $D_X(\cdot)$ changes in router X, it updates its distance vectors, then notifies each neighbour Y
 - Update at node X:
 - For each node Y in X's routing table, compute:
 $D_X(Y) = \min\{C_{X,Y} + D_Y(Y)\}$, where V = X's neighbour
 - Link cost increase can cause DV algorithm to run many more times than link cost decrease
- **Routing Information Protocol (RIP):**
 - DV with $C_{X,Y} = \# \text{ hops from X} \rightarrow Y$
 - Uses UDP, port **520**
 - Routing table exchanged every 30s
 - No update from neighbour for 3 minutes \rightarrow delete entry

ICMP Protocol

- Internet Control Message Protocol
- For error reporting or router signaling
- ICMP header: Type, Code, Checksum
- **ping**
 - Type = 8, Code = 0
 - Checks if a host is reachable on an IP network
- TTL expired: Type = 11, Code = 0
- **Traceroute**
 - Returns route taken by a packer from one host to another by setting TTL = # hops

NAT

- Network Address Translation
 - One NAT table per router (not per interface)
1. Host A (private IP_A, port) sends datagram to NAT router
 2. NAT router determines output interface (e.g. C) using the dest. address, then modifies (private IP_A, port) \rightarrow (NAT IP_C, unique port) and sends the datagram
 3. NAT saves (NAT IP_C, unique port) \rightarrow (private IP, port) mapping in its NAT translation table
 4. Server C responds back with src = IP_C, dest = (NAT IP_C, unique port)
 5. NAT translates (NAT IP_C, unique port) \rightarrow (private IP, port) and sends datagram with src = IP_C, dest = (private IP, port) back to Host A

Link Layer

- Resides in NIC card
- Involved adjacent nodes over a single link

Error Detection & Correction

1-D Parity Checking

- Detects all odd number of single bit errors, but cannot correct
- Poor performance with bursts or errors
- Even parity: 1 if data has odd # of 1s, 0 otherwise
- Sender sends parity bit; receiver count number of 1 bits in data and parity bit

2-D Parity Checking

- Detects all single or 2 bit errors
- Corrects all single bit errors

Cyclic Redundancy Check (CRC)

- Length of CRC = r
- Length of generator $G = r + 1$. MSB of G must be 1
- Sender:
 1. Append r number of 0s to data D
 2. Divide D by G (XOR)
 3. Remainder = CRC
- Receiver:
 1. Divide data $D \oplus \text{CRC}$ by G
 2. If remainder = 0, no errors
- Detects all odd number of single bit errors as well as all burst (consecutive bit) errors of $\leq r$ bits
- Detects burst errors of $> r$ bits with $P = 1 - 0.5^r$

Multiple Access Protocols

- Ideal properties:
 1. Collision free
 2. Efficient: a transmitting node should transmit at max capacity
 3. Fairness: when n nodes want to transmit, each node transmits at a rate of R/n
 4. Fully decentralised: no external nodes

Channel Partitioning Protocols

Time Division Multiple Access (TDMA)

Collision Free	Efficient	Fairness	Decentralised
✓		✓	✓

- Divide into time frames and divide each time frame by # nodes
- Each node gets fixed length time slot to transmit at max throughput = R/n

Frequency Division Multiple Access (FDMA)

Collision Free	Efficient	Fairness	Decentralised
✓		✓	✓

- Divide channel spectrum into n frequency bands and each node transmits at one band at max throughput = R/n

Taking Turns Protocols

Polling Protocol

Collision Free	Efficient	Fairness	Decentralised
✓	(polling overhead)	✓	

- Master node polls which node to transmit next (and how many frames to transmit) in round-robin fashion

Token Passing Protocol

Collision Free	Efficient	Fairness	Decentralised
✓	(token passing overhead)	✓	✓

- Token (special frame) passed from one node to the next sequentially
- Only the node holding onto the token can transmit X # of frames
- If a node has nothing to transmit, it node immediately passes the token on to the next node
- Token loss possible → need to implement token recovery

Random Access Protocols

Slotted ALOHA

Collision Free	Efficient	Fairness	Decentralised
	✓	✓	✓

- Divide time into equal slots (of length \geq time to transmit 1 frame); each node attempts to transmit at the beginning of a slot
- Collision: with probability p , transmit at next slot
- Efficiency $\approx 37\%$

Pure ALOHA

Collision Free	Efficient	Fairness	Decentralised
	✓	✓	✓

- Nodes attempt to transmit at any given time
- Collision: wait for 1 frame transmission time, then with probability p , transmit at next slot or wait for another frame transmission time with probability $1 - p$
- Efficiency $\approx 18\%$

Carrier Sense Multiple Access (CSMA)

Collision Free	Efficient	Fairness	Decentralised
	✓	✓	✓

- If channel is busy: wait for short while before sensing again
- If channel is idle: transmit
- Collision: Possible due to propagation delay; does not abort transmission
- No collision detection → no retransmission → up to upper layer to detect the corrupted data and request retransmission again

Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

Collision Free	Efficient	Fairness	Decentralised
	✓	✓	✓

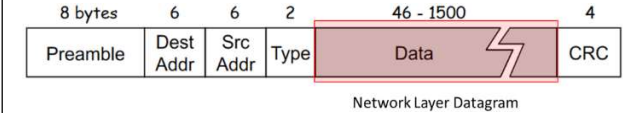
- If channel is busy: wait for short while before sensing again
- If channel is idle: transmit
- Collision: Possible due to propagation delay; abort transmission and retransmit entire frame again after a some delay
- Exponential backoff: For each frame,
 - After the n^{th} collision, next delay is random $K \in \{0, 1, 2, \dots, 2^n - 1\}$ time units, up to $n = 10$
 - Time unit = 512 bit time (time taken to transmit 512 bits)
 - Actual delay = $512K/R$
- Ethernet minimum frame size = 64 bytes to ensure $d_{\text{trans}} > d_{\text{prop}}$ and collisions are detected

Ethernet

MAC Address

- Burnt into NIC card (i.e. fixed and globally unique)
- A host can have multiple MAC addresses as long as it has multiple NIC
- Switches in LAN have no MAC addresses (transparency)
- 48-bit hexadecimal e.g. 5C-F9-DD-E8-E3-D2
- Broadcast address = FF-FF-FF-FF-FF-FF
- **LAN**: network that interconnects hosts within geographical area

Ethernet Frame



- **MTU**: Maximum size of data frame can carry; excludes ethernet frame headers and trailers
- src & dest addr: NIC only delivers packets up if dest MAC of received packet is broadcast address or own address
- Preamble: 7 bytes of 10101010 (AA) and 1 byte of 10101011 (AB) or "start-of-frame"
 - Synchronise sender and receiver clock rates
- Type: indicates the higher layer protocol (e.g. IP)

Physical Topology

- **Bus**:
 - Broadcast LAN, all devices connect to it
 - Bus is single point of failure, hard to troubleshoot and slow due to collisions

- **Star**:

Hub	Switch
Physical layer device	Link layer device
Acts on individual bits	Acts on frames (store-and-forward)
Collisions possible	Simultaneous transmissions without collisions
Bits are copied, boosted and broadcasted to all interfaces	Frames only transmitted to one interface
Cheap	Expensive

Ethernet Switch

- Uses CSMA/CD
- Transparent, plug-and-play (self-learning)
- Buffers frames
- Frame filtering/forwarding (Host A → Host B, $A \neq B$)
 1. Switch receives frame from interface X occupied by Host A
 2. Switch stores $\langle \text{MAC_A}, \text{Interface_X}, \text{TTL} \rangle$ in switch table
 3. Does switch table contain MAC_B ?
 - Yes: Forward to table[MAC_B].Interface
 - No: Broadcast to all interfaces except source interface
- If src MAC = dest MAC, frame is filtered

ARP Protocol

- Address Resolution Protocol
- Maps IP address to MAC address
- ARP tables reside in each interface
- ARP table full → need to wait for an entry to expire

Same Subnet (Host A → Host B)

1. Host A's ARP table contains IP_B?
 - Yes: Transmit ethernet frame with dest MAC = MAC_B
 - No: Transmit **ARP query** packet with dest MAC = FF-FF-FF-FF-FF-FF
2. Switch receives frame from Host A
 - Switch keeps a record <MAC_A, Interface_X, TTL> if there isn't one
 - Not ARP query?
 - Switch table contains MAC_B?
 - Yes: Forward frame to table[MAC_B].Interface. **DONE**
 - No: Broadcast frame to all interfaces. **DONE**
 - ARP query?
 - Broadcast frame to all interfaces
3. Host B receives the frame and responds with its MAC address
 - Host B keeps a record of <IP_A, MAC_A> in its ARP table if there isn't one
4. Switch receives frame from Host B
 - Switch keeps a record <MAC_B, Interface_Y, TTL> if there isn't one
5. Switch forwards frame to Interface_X (Host A)
6. Host A keeps a record of <IP_B, MAC_B, TTL>. Repeat from 1 with IP datagram containing the data to be sent

Different Subnets (Host A → Host B)

- Whether or not the receiver is in the same subnet or not can be determined using the subnet network name + mask via DHCP
1. Host A knows router's IP and MAC?
 - Yes: Host A creates and transmits ethernet frame with src MAC = MAC_A and dest MAC = MAC_Router to router. Router then forwards the frame to Host B, possibly running ARP Protocol again if it does not know B's MAC address. **DONE**
 - No: Host A creates an ARP query for IP_B with src MAC = MAC_A and dest MAC = FF-FF-FF-FF-FF-FF
 2. Host B is not reachable but the router receives the frame, extracts the IP datagram and realises that it can reach Host B. Router then responds to Host A that it can do so
 3. Host A now knows that it can reach Host B via the router. Host A then sends IP datagram with dest IP = IP_B, encapsulated within ethernet frame with dest MAC = MAC_router
 4. Router receives and modifies the frame and forwards the frame with src MAC = MAC_Router and dest = MAC_B to Host B

Network Security

- **Confidentiality:** only sender and intended receivers can understand message contents
- **Authentication:** sender and receiver need to confirm each other's identity
- **Integrity:** messages are not altered without detection
- **Access & Availability**

Confidentiality

- **Symmetric** Key Cryptography
 - Sender and receiver use the same encryption key (K_s)
 - Sender and receiver need to agree on same key
- **Public** Key Cryptography
 - Sender uses public key; receiver uses its own private key
 - Requirements
 - $m = K^*(K^*(m))$
 - Given K^* , it should be impossible to find K

Caesar's cipher (Symmetric)

- A substitution cipher using fixed shift of alphabet
- $K_s = \#$ positions to shift (26)

Monoalphabetic cipher (Symmetric)

- A substitution cipher by substituting one letter for another
- $K_s =$ substitution cipher (26!)
- Can be cracked with Statistical Analysis

Polyalphabetic cipher (Symmetric)

- Using n substitution ciphers and a cycling pattern, substitute the i^{th} symbol in plaintext using the i^{th} cipher in the cyclic pattern
- $K_s = n$ substitution ciphers + cycling pattern

Block cipher (Symmetric)

- Message is processed in blocks of X bits and each block is encoded in a one-to-one mapping
- $K_s =$ block cipher (2^X)
- DES (Data Encryption Standard)
 - 56-bit symmetric key, 64-bit block
- AES (Advanced Encryption Standard)
 - 128/192/256-bit symmetric key, 128-bit block

RSA Encryption (Public)

- Convert message to bit pattern then to integer (e.g. 00001100 = 12)
- $K^*(K^*(m)) = m = K^*(K^*(m))$
- 1. Choose two large prime numbers p, q
- 2. Compute $n = pq, z = (p-1)(q-1)$
- 3. Choose e s.t. $e < n$ and has no common factors with z (relatively prime)
- 4. Choose d s.t. $ed-1$ is exactly divisible by z ($ed \bmod z = 1$)
- 5. Receiver's $K^* = (n, e)$, Receiver's $K = (n, d)$
- 6. Sender Encrypt: $m^e \bmod n = c$, Receiver Decrypt: $c^d \bmod n = m$
- RSA + DES/AES:
 1. Select session key K_s
 2. Use RSA to transfer K_s
 3. K_s is used as the symmetric key in DES/AES for encryption for the session
- RSA is computationally more expensive than DES/AES

Message Integrity

- **md5:** generates short, 128-bit digests
- Authentication key s : shared only between sender and receiver
 - $m \oplus H(m)$ is not enough. Attacker can change to $m' \oplus H(m')$
- Sender sends Message Authentication Code, $H(m + s)$, along with the message i.e. $m \oplus H(m + s)$
- Receiver calculates $H(m + s)$ and compares it with the one it receives

Authentication

- Sender attaches digital signature which must be
 1. Verifiable by receiver
 2. Unforgeable
- Signature: Sender encrypts hashed message with its private key K i.e. $m \oplus K^*(H(m))$
- Receiver uses sender's private key K^* to decrypt the signature and checks if $K^*(K^*(H(m))) = H(m)$

Certification Authorities

- Public database containing a list of public keys K^*_x , each signed by and certified by the CA
- CA signs public keys using its own private key i.e. $K(K^*_x)$
- A list of trusted CA's public keys are hardcoded in the OS
- To get sender's private key K^*_x , receiver gets sender's certificate $K(K^*_x)$ and decrypts it using CA's public key ($K^*(K(K^*_x)) = K^*_x$)

Multimedia Networking

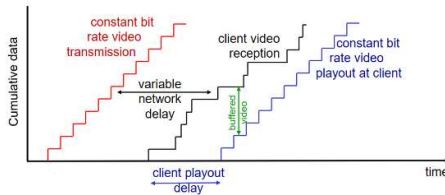
- **Streaming stored** media: playout begins before entire file is downloaded
- **Streaming live** media
- **VoIP:** Voice-over IP; delay must be < 400ms
- **Videos**
 - Spatial Redundancy (per image): Instead of sending the same colour value N times, send the value once along with N
 - Temporal Redundancy (across images): Only send the differences between two consecutive frames
 - High Bit-Rate: large number of bits transmitted per unit time
 - Constant Bit-Rate (CBR): fixed video encoding rate. Minimises lag
 - Variable Bit-Rate (VBR): video encoding rate changes as amount of spatial and temporal coding changes. Higher quality videos
- **Audio**
 - Analog signal is sampled at a fixed rate if N times per second
 - Each sample is quantised into one of 2^m levels
 - Bit rate = Nm bits per second

Streaming Stored Video

- Features
 1. **Streaming:** Client should be able to play video while video file is still being downloaded
 2. **Interactivity:** Client should be able to skip to certain parts of the video, pause, fast-forward etc.
 3. **Continuous Playout:** Video should playout uninterrupted i.e. video should not freeze
- Jitter: variation in network delay, causing packets to arrive at client at variable time; can affect continuous playout

Client-side Buffering

- Client introduces playout delay to compensate for jitter
- Once client starts receiving the video file, it waits for a while to fill up a portion of its buffer before beginning playout
- If average fill rate < playout rate: buffer eventually empties → video freezes
- If average fill rate > playout rate: buffer will never be empty if initial playout delay is large enough to absorb variability in fill rate
- If buffer is full, fill rate reduces to playout rate



UDP Streaming

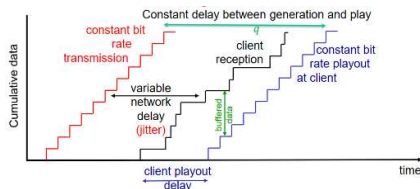
- **Push-based:** server transmits video at rate equals to the video consumption rate at the client
 - If client consumes at 2Mbps and one UDP packet carries 1B of video, server transmits 1 packet every $1B/2Mbps = 4ms$
- Not rate-controlled by congestion control mechanism \rightarrow fast \rightarrow requires smaller playout delay (smaller initial buffer)
- RTP (Realtime Transport Protocol)
 - Needed due to lack of error recovery in UDP
 - Application level protocol
 - Header: <seq #, encoding scheme, timestamp>
- Likely to be blocked by firewalls

HTTP Streaming

- **Pull-based:** client sends GET request for each video frame
- Fill rate fluctuates due to TCP congestion control and retransmissions
- Requires larger playout delay due to fill rate fluctuation
- Less likely to be blocked by firewalls

VoIP

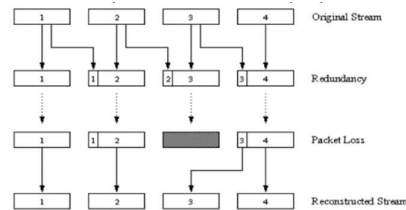
- Maximum end-to-end delay: 400ms
- Packets are generated only during talk spurts
 - Each chunk represents 20ms of audio, encoded at 8KB/s \rightarrow each chunk is 160B
 - chunk + application-level header encapsulated into TCP/UDP segment
- Network jitter: Due to variation in delays, if packets are played the moment they are received, voice can sound intelligible
- **Fixed Playout delay, q :**
 - Each packet comes with timestamp t in its header (time the packet was transmitted by the sender)
 - Packet is played out at $t + q$
 - If arrival time $> t + q$, the packet is discarded
 - Small q : more packet loss, less playout delay
 - Large q : less packet loss, more playout delay



- **Variable Playout delay, d :**
 - Estimates the network delay and adjust the playout delay at the beginning of each talk spurt, compressing silent periods
- $$\text{delay}(d) = (1 - \alpha) \times d_{\text{prev}} + \alpha \times (\text{received}(r) - \text{timestamp}(t))$$
- $$\text{variance}(v) = (1 - \beta) \times V + \beta \times |r - t - d|$$
- $$\text{playout time} = t + d + 4v$$

Forward Error Correction (FEC)

- **XOR-based FEC**
 - Group n chunks and XOR them to get the $n + 1^{\text{th}}$ chunk
 - Send the $n + 1$ chunks to client
 - If at most 1 chunk is lost, the lost chunk can be reconstructed from the remaining n chunks
 - Increases playout delay if a chunk is lost since need to wait for the entire group of chunks to arrive before recovering the lost chunk. If no chunks are lost, playout delay is unaffected
 - Increases bandwidth by factor $1/n$
- **Piggybacking Scheme**
 - For each chunk n , also send lower resolution version of chunk $n - 1$ (and $n - 2, \dots$)



- Incurs extra overhead in each chunk
 - **Interleaving Scheme**
 - Each chunk is further divided into smaller units, then interleave each unit among chunks
-
- If one chunk is lost, only a fraction of the original chunk is lost
 - Increases playout delay due to reordering

DASH

- **Dynamic Adaptive Streaming over HTTP**
- **Server:**
 - Instead of sending entire video encoded by a single encoding scheme, divide video into smaller chunks. Each chunk that is stored will have multiple copies of itself, each encoded at different rates
 - Manifest file provides the URLs for different encodings

- **Client**
 - Periodically measure bandwidth
 - Requests a chunk (at an appropriate time, according to buffer size) and executes an adaptive bitrate algorithm (ABR) to find the maximum encoding rate that is sustainable at the current bandwidth to determine which quality of the file to download next
 - Uses the manifest file to locate the video chunk encoded at a specific rate
- Each video chunk is ~ 2 to 10s long \rightarrow not ideal for livestreaming
- Typically used for Video on Demand (VoD) or streaming stored media
- **Content Distribution Networks (CDNs)**
 - CDNs are a network of geographically distributed servers
 - Store/serve multiple copies of videos at CDNs
 - When a user requests content from a website that uses CDN, the CDN selects the server that is geographically closest to the user and delivers the content to the server \rightarrow reduces the latency