Network Layer

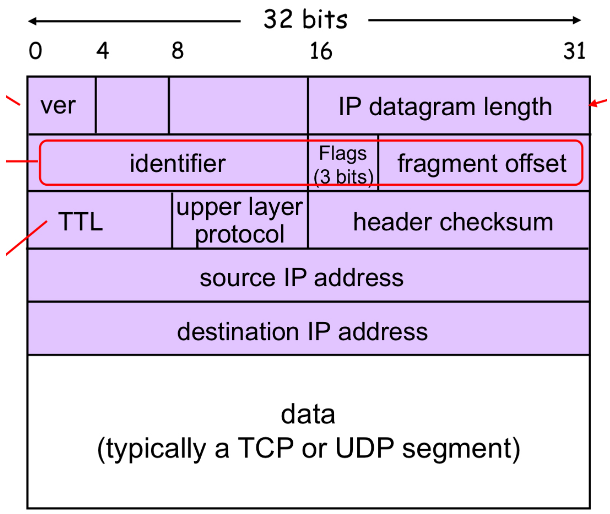
> Delivers packets to receiving hosts

Internet Protocol (IP)

> Dictates addressing conventions, datagram format, and packet handling conventions

IPv4 Datagram Format

- Total size: 20 bytes



- ver: IP protocol version number (e.g. IPv4, IPv6)
- IP datagram length: Header + data
- Identifier, flags, fragment offset: For fragmentation
- TTL: Number of remaining hops before packet dies
- Header checksum: Only for this header, not TCP

IP Fragmentation

- Motivation: Different links may have different Max Transfer Unit
- Solution:
  - Routers fragment IP datagrams
  - Reassemble **only at final destination**
    - TCP does checksum, so reassemble at destination host and then checksum
- How to identify fragments and their relative order? Header fields
  - Identifier: Same for the same segment
  - Flag: 1, if there are more fragments from same segment. 0, if this is the last one.
  - Offset: Offset of data in **units of 8-bytes**
    - What if data size is not divisible by 8? Make data size divisible by 8

Network Address Translation (NAT)

- Motivation:
  - Recall: IP address is 32 bits → How has it not run out?
  - Internet has 2 types of IP addresses:
    - Public: Globally unique and routable
    - Private: Not routable on Internet, Routable within organization, Not globally unique
  - How do private IPs access public IPs? Since private IPs can duplicate and public IPs cannot uniquely identify them
- Solution: Connect to **intermediary router with public IP**
  - All **outgoing** datagrams have **same source NAT IP address** (WAN)
  - **Within local network** (LAN), hosts use private IP addresses for communication
  - NAT translation table: Maps **port numbers in WAN** side to map to **private IP addresses in LAN** side
    - Possible mappings: $2^{16} - 1$
- Procedure:
  1. Host inside local network sends datagram to router
  2. NAT router changes datagram source address and port, and updates table
  3. Reply arrives
  4. NAT router changes datagram destination address and port
- Pros:
  - Only need to rent 1 public IP for NAT router
  - Can change addresses of local network hosts without notifying outside world
  - Hosts inside local network are not visible by outside world

Routing Algorithm

- Motivation: How do we build forwarding table?
- Routing in the Internet: Internet is a hierarchy of Autonomous Systems (AS)
  - Intra-AS routing: Find good path between 2 routers inside an AS
    - Single admin → No policy decisions needed
    - Focused on performance
    - Commonly used protocols: RIP, OSPF
  - Inter-AS routing: Handles interfaces between ASs
    - Policy > Performance
    - Standard protocol: BGP
- Goal: Find **least cost path** between 2 vertices in a graph
- Notation:
  - Vertices = Routers, Edges with cost = Links between routers
  - $C(x, y)$: Cost of link between routers x and y
  - $D_x(y)$: Cost of least-cost path from x to y
  - N: 1-hop neighborhood
- Bellman-Ford Equation:

$$D_u(z) = \min_{a \in N}(C(u, a) + D_a(z))$$

- Procedure:
  1. All nodes start with distances to adjacent neighbors only
  2. All nodes wait for update from neighbor
  3. All nodes computes distance vector using distance vector from neighbors and equation above
     - Computation **does not** include previously stored value. Only use DVs from neighbors.
  4. **If distance vector changed, update neighbors**
  5. Repeat for $n-1$ times
- Insight: For the k-th hop, the shortest paths with k hops are correct

Routing Information Protocol (RIP)

- Use hope count as cost metric
- Self repair: If no update from neighbors for 3 minutes, assume neighbor has failed

Link State Algorithms

- What if routers have global view of network topology and link costs? Dijkstra

Internet Control Message Protocol (ICMP)

> Used by hosts & routers to communicate network-level information, like error reporting and echo request

- ICMP messages carried in IP datagrams
  - ICMP header starts after IP header → Located in transport layer, but used for network layer
- `ping`: Check if remote host will respond to us
- `traceroute`: Display route that messages take to get to a remote host
  - Send series of small packets across network with increasing TTL
  - ICMP error message sent to datagram's source address

Link Layer

> Sends datagram between **adjacent nodes** over a single link

- IP datagrams encapsulated in link-layer data frames
- Aim: Send data between N nodes via cable
- Solution: Interconnect N nodes with a shared broadcast link
  - Each link needs to be **addressed** → Framing
  - Need a **protocol** → Link access control
  - Need to handle **errors** → Reliability, Detection
- **Physically** implemented on adapter (aka Network Interface Card (NIC))
  - Different from higher layers, which are done in operating systems

Error Detection and Correction

> Add Error Detection and Correction bits (EDC) to data

- Not 100% reliable
- Typically, more EDC yields better detection
- Common error detection schemes: Checksum, Parity checking, CRC

Single Bit Parity Checking

> In even parity scheme, sender adds 1 additional bit to make number of 1s even

- Can detect **odd number of single bit** errors
  - Cannot detect even number of single bit errors
  - Thus, can detect 50% of errors

2D Parity Checking

> Data divided into i rows and j columns. Parity computed for each row and each column.

- One more parity bit for column and row parity bits
- Can **detect and correct single bit errors**
- Can **detect any two-bit error**
  - Cannot fix, since not sure which 2 intersections
- Adds more overhead

Cyclic Redundancy Check (CRC)

- D: Data bits (Binary)
- R: r bit CRC
- G: Generator of r + 1 bits, which is agreed by sender and receiver
- Calculations done in **modulo 2 arithmetic**
  - Performed digit by digit on binary
  - No carries for addition and no borrows for subtraction
  - **Addition and subtraction are same as XOR**

1. Append r 0s to D
2. Divide number with G
   - Long division using modulo 2 arithmetic!
   - Note: You can subtract a bigger number from a smaller number in modulo 2, as long as same number of bits
3. Remainder gives R
4. Sender sends (D, R)
5. Receiver knows G, divides (D, R) by G
   - If non-zero remainder, error is detected
   - Else, no error

- CRC of r bits can detect:
  - All odd number of single bit errors
  - All burst errors of less than r + 1 bits
    - Since we check by $\mod G$, which is $r+1$ bits long
  - All burst errors of greater than r bits with probability of $1 - 0.5^r$
- CRC aka **polynomial code**
  - E.g. $x^5 + x^4 + 1 \rightarrow 110001$

Multiple Access Links and Protocols

- 2 types of network links
  1. Point-to-point link: Sender and receiver connected by **dedicated link**
     - No need for multiple access control
  2. **Broadcast link**: Multiple nodes connect via **shared broadcast channel**
     - When node transmits a frame, every node gets a copy

> Motivation: Follows human etiquettes in group discussion

- Given broadcast channel of R bps, ideal multiple access protocol:
  1. Collision free: Node does not receive 2+ signals at same time
  2. Efficient: When 1 node wants to transmit, it can send at rate R
  3. Fairness: When m nodes want to transmit, each can send at average rate of R/M
  4. Fully decentralized: No special node for coordination

Channel Partitioning

- Divide channel into fixed, smaller pieces
- Allocate piece to node for exclusive use

Time Division Multiple Access (TDMA)

> Like presidential debate: Each node gets fixed length time slots in each round

- Time frame: Collection of N time slots
- Cons: If nodes have nothing to send, their slots are idle and wasted

Collision Free	Efficiency	Fairness	Decentralized
Yes	No. Throughput: R/N	Yes	Yes

Frequency Division Multiple Access (FDMA)

- Same idea as TDMA, except each node is assigned to fixed frequency

Taking Turns

Polling

> Master node polls from each node in round-robin fashion

- Master informs node 1 that it can transmit up to some number of frames
- Node 1 transmits some frames
- Master informs node 2 that it can transmit up to some number of frames
- Node 2 transmits nothing
- Repeat

Collision Free	Efficiency	Fairness	Decentralized
Yes	Better. Throughput: Almost R (Polling overhead)	Yes	No. Master is single point of failure

Token Passing

> Special frame, token, passed frame one node to next. Send only if have token.

- If need to transmit some frames, hold onto token and send up to maximum number of frames
- Else, forward the token

Collision Free	Efficiency	Fairness	Decentralized
Yes	Better. Throughput: Almost R (Token overhead)	Yes	Yes

- Cons:
  - Token loss can be disruptive
  - Node failure can break the ring

Random Access

> How do we detect and recover from collisions?

Slotted ALOHA

- Design:
  - All data frames have equal size of L bits
  - Time divided into slots of equal length
    - Length = Time to transmit 1 data frame = L/R
- Operation:
  - Nodes only transmit at **beginning of slot**
  - If no collision, success
  - If collision, retransmit frame in next slot with probability p. Repeat until success
    - Possible for all nodes to send nothing

Collision Free	Efficiency	Fairness	Decentralized

Collision Free	Efficiency	Fairness	Decentralized
No	Yes, when only 1 node. No, when many nodes need to send efficiency is 37%	Yes	Yes

Pure (unslotted) ALOHA

- Design:
  - Simpler than slotted ALOHA
  - No time slots
  - No synchronization
- Operation:
  - Nodes transmit frame **immediately**
  - If no collision, success
  - If collision, wait for 1 frame transmission time and retransmit frame with probability p until success
- Chance of collisions increases
  - Since frame sent at t collides with other frames sent in $(t-1, t+1)$

Collision Free	Efficiency	Fairness	Decentralized
No	Yes, when only 1 node. No, when many nodes need to send efficiency is 18%	Yes	Yes

Carrier Sense Multiple Access (CSMA)

> Listen before transmit

- Addresses design flaw in ALOHA: Node pays no attention to other nodes
- Operation:
  - If channel sensed idle, transmit
  - If channel sensed busy, defer transmission
- Collisions can still occur
  - Propagation delay: 2 nodes may not head each other's transmission immediately

CSMA/CD (Collision Detection)

> Stop once collision detected

- Addresses design flaw in ALOHA and CSMA: Node does not stop transmitting even if collision detected
- Operations:
  - Same as CSMA
  - If collision detected, abort transmission
- Backoff Algorithm
  - Motivation: Probability of collision in all future time slots are the same

    - Goal: Since more collisions imply heavier load, increase back-off interval with more collisions
    - Binary Exponential Backoff:
        - After m-th collision, choose K from $(0, 1, \ldots, 2^m - 1)$ and wait for K time units before retransmission
        - Thus, $p = 1/2^m$
        - Ethernet: 1 time unit is set to 512 bit transmission times
- What if frame size is too small?
    - Collisions happen, but may not be detected by sending nodes → No retransmissions
    - Solution: Set minimum frame size limit
        - Ethernet: 64 bytes

Collision Free	Efficiency	Fairness	Decentralized
No	Yes	Yes	Yes

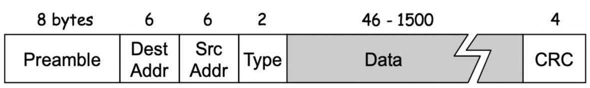
Addressing

- Every NIC has a Media Access Control (MAC) address
    - 48 bits: 5C-F9-DD-E8-E3-D2
    - Allocated by IEEE
    - First 3 bytes identifies vendor of adaptor
    - Broadcast address: FF-FF-FF-FF-FF-FF
- Differences with IP address:
    - MAC address: Flat, globally unique, Like NRIC
    - IP address: Hierarchical, Like postal code

Ethernet

- Local Area Network (LAN): Computer network that interconnects computers within **geographical area**
    - Popular LAN technologies: IBM token ring, Ethernet, Wi-Fi

Ethernet Frame Structure



- Preamble:
    - 7 bytes of 10101010
    - Followed by 1 byte of 10101011 (aka "start of frame")
    - Used to synchronize receiver and sender clock rates (Width of bit)
        - E.g. How to tell between 19 or 20 zeros without knowing width of bit?
- Destination and source MAC address:
    - If NIC receives frame with matching destination address, or with broadcast address, pass data in frame to network layer

    - Else, discard frame
- Type: Indicates higher layer protocol
    - Motivation: Hosts can use other network-layer protocol besides IP
    - Allows Ethernet to multiplex into correct network-layer protocol
- Data:
    - Maximum size: 1500 bytes (Determined by link MTU)
    - Minimum size: 46 bytes (So all collisions can be detected)
- CRC: To check for error

Ethernet Data Delivery Service

- Unreliable: Receiving NIC does not send ACK or NAK to sending NIC
    - Data in dropped frames will be recovered only if sender uses higher layer RDT (e.g. TCP)
- Multiple access protocol: CSMA/CD

Switches

- Motivation: How do we interconnect nodes to create this shared link?
    - Bus: Broadcast LAN
        - Cons: Back bone cable is single point of failure, Slow due to collisions
    - Hub: Physical-layer that acts on individual bits
        - Pros: Cheap, Maintainable due to modular design
        - Cons: Slow due to collisions
- Switch: Link-layer device that acts on frames, which is an upgrade from hub
    - No collisions
    - Can store-and-forward
    - Uses CSMA/CD to access link (For backwards compatibility)
    - **Transparent**: Hosts are unaware of presence of switches
    - Plug-and-play: **Self-learning**
    - Nodes have dedicated, direct connection to switch
    - Buffer packets
    - Interconnecting switches: Can be connected in a hierarchy

Switch Forwarding Table

- Motivation: How does switch know A is reachable via interface 1?
- Format of entry: (MAC address of host, Interface to reach host, TTL)
- How are entries created and maintained?
    - Self-learning: When frame received at switch
        1. Record incoming link, MAC source address of sending host
        2. Query table using MAC destination address
        3. If entry found in table:
            1. If found interface is where the frame was sent from, drop frame

    2. Else, forward frame to interface indicated entry
4. Else, flood and forward to all interfaces, except for arriving interface

Address Resolution Protocol (ARP)

- Motivation: When sending packet to another host, network layer tracks destination IP. But, data frame in link layer does not.
- Purpose: How do we get MAC address of destination host from its IP address?
- Each IP node (Routers and hosts) has ARP table: Stores mappings of IP address and MAC address in the **same subnet**
    - Format of entry: (IP address, MAC address, TTL)

Sending Frame in the Same Subnet

- Suppose A and B are in the same subnet. A wants to send data to B:
    1. If A knows B's MAC address in its ARP table:
        1. Create frame with B's MAC address and send it
        2. Only B will process frame
        3. Other nodes may get it, but will ignore it
    2. If A does not know B's MAC address:
        1. Broadcast ARP query packet with B's IP address to FF-FF-FF-FF-FF-FF
        2. Only B replies to A with its MAC address
        3. A caches B's MAC address in ARP table

Sending Frame to Another Subnet

> ARP table only stores MAC addresses in same subnet

- Suppose A and B are in different subnets connected by router R. A wants to send data to B:
    1. A creates frame with **R's MAC address** as destination address and sends it
    2. R creates frame with B's MAC address as destination address
- Source and destination MAC address changes at different points
- Source and destination IP does not change

Network Security

- Objectives:
    - Confidentiality: Only sender and intended receiver should **understand** message
    - Message integrity: Sender and receiver can ensure message **not altered without detection**
    - Authentication: Sender and receiver can **confirm identity** of each other
- Cryptography:
    - Allow sender to **disguise** data, so intruders cannot understand it
    - Allow receiver to **recover** original data from disguised data
- Notation:

    - Plaintext: Original message
    - Ciphertext: Encrypted message
    - Key: Input parameter to encryption/description algorithm
    - m: Plaintext message
    - $K_A()$: Encryption algorithm with key K_A
        - $K_A(m)$: Cyphertext
    - $K_B()$: Encryption algorithm with key K_B
        - $K_B(K_A(m)) = m$
- Types of cryptography:
    - Symmetric Key Cryptography: Sender and receiver use **same key**
    - Asymmetric Key Cryptography: Sender and receiver use **different key**
        - Aka public key cryptography
- How do sender and receiver agree on key value?
    - Need to decide before communication (e.g. F2F meeting)

Confidentiality

> How to ensure only sender and receiver can **understand** message?

Breaking an Encryption Scheme

- Ciphertext-only Attack: Intruder only has ciphertext
- Known-plaintext Attack: Intruder has plaintext corresponding to ciphertext
- Chosen-plaintext Attack: Intruder can get cipertext for chosen plaintext

Caesar's Cipher

> Fixed shift of alphabet

- Form of substitution cipher: Substituting one thing for another
- Encryption key: Shift number (25 possible values)
- Weakness: Easy to break

Monoalphabetic Cipher

> Map letter to another letter

- Form of substitution cipher
- Encryption key: Mapping from set of 26 letters to set of 26 letters ($26!$ possible values)
- Weakness: Only 1 mapping. Still can be broken using statistical analysis

Polyalphabetic Encryption

> Monoalphabetic Cipher, with many mappings

1. Define n substitution ciphers $C_1, C_2, \ldots, C_n$
2. Define cycling pattern

- E.g. $n = 4, C_3, C_3, C_4, C_2$

3. For each new plaintext symbol, cycle through cycling pattern

- Encryption key: n substitution ciphers and cycling pattern

Block Cipher

> Message to encrypt processed in blocks of K bits

- Each block has one-to-one mapping to another block
- Number of possible keys: $2^K!$
- Popular block ciphers:
  - Data Encryption Standard (DES): 64-bit block
  - Advanced Encryption Standard (AES): 128-bit block

RSA

- Motivation:
  - Previous encryptions are all **symmetric**, which requires sender and receiver to know the key
  - How do they agree on this key if they never met?
- Solution: Public key cryptography
  - Sender and receiver do not share secret key
  - Sender uses **public encryption key** (K_B^+) known to all
  - Receiver uses **private decryption key** (K_B^-) known to receiver
- Requirements:
  - Need K_B^+ and K_B^- such that $m = K_B^-(K_B^+(m))$
  - Given public key, impossible get private key
- Modular arithmetic:
  - $((a \mod n) \pm (b \mod n)) \mod n = (a \pm b) \mod n$
  - $((a \mod n)(b \mod n)) \mod n = (ab) \mod n$
  - $(a \mod n)^d \mod n = a^d \mod n$

Creating public/private key pair:

1. Choose 2 large prime numbers p and q
2. Compute $n = pq$ and $z = (p-1)(q-1)$
3. Choose e such that $e < n$ and e has no common factors with z
4. Choose d such that $ed \mod z = 1$
5. Public key: (n, e), Private key: (n, d)

Encryption and decryption:

1. To encrypt m, compute $c = m^e \mod n$
2. To decrypt c, compute $c^d \mod n$

> Magic! $c^d \mod n = (m^e \mod n)^d \mod n = m$

- Order of using public/private key does not matter: $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$

Session Key

- Motivation:
  - In practice, exponentiation in RSA is expensive
  - DES much faster, but needs to share symmetric key K_S
- Solution:
  1. Choose some key K_S
  2. Use RSA to transfer K_S
  3. Use K_S as symmetric key in DES for encrypting the actual data
- Session key: K_S

Message Integrity

> How to **detect alterations**?

- Can we just use checksum, parity, CRC?
  - No, since easy to find another message with same checksum value
  - Designed for accidental errors, not attacks
- Solution: Cryptographic hash function
  - Hash function: $H()$ that takes in input m and produces **fixed-size** message digest (fingerprint)
    - Many-to-1
  - Cryptographic hash function: Hash function such that you cannot find 2 different message x and y such that $H(x) = H(y)$
    - 1-to-1
    - Popular functions: MD5, SHA-1

Message Authentication Code (MAC)

> Combine authentication key s with message inside hash

- To ensure message integrity, can we send $(m, H(m))$?
  - No, what if replaced with $(m', H(m'))$? Cannot be detected
- Procedure:
  1. Sender and receiver share a authentication key s
  2. Sender sends $(m, H(m+s))$
     - $H(m+s)$ is message authentication code
  3. Receiver uses s to calculate $H(m+s)$
  4. Compare both hashes

Password Hashing

- Passwords not stored in plaintext, but hashed

    - Thus, passwords cannot be recovered, only reset
- Compare using generated hash

Authentication

> How to **confirm identity** of sender?

- Digital signatures: Like a hand-written signature. Establishes that sender is document creator
    - Verifiable: Recipients can check that message was generated by correct sender
    - Unforgeable: Only the sender can generate the signature
- Procedure:
    1. Sender signs m by encrypting with **sender's private key** ($K_B^-(m)$)
        - Unforgeable
    2. Sender sends $(m, K_B^-(m))$
    3. Receiver verifies m by using **sender's public key** ($K_B^+(K_B^-(m))$)
    4. If $K_B^+(K_B^-(m)) = m$, then receiver verifies that:
        - Sender signed m
        - No one else signed m
        - Sender signed m and not m'
- Optimization:
    - Motivation: RSA is expensive on huge data
    - Solution: Hashing produces **fixed-length** fingerprint
        - Sender hashes m first, then apply encryption with sender's private key ($K_B^-(H(m))$)
        - Receiver hashes m and compares with decrypted $H(m)$
- Why not just use message authentication code for authentication?
    - Does not ensure that message is from Bob, since Alice also has the authentication code

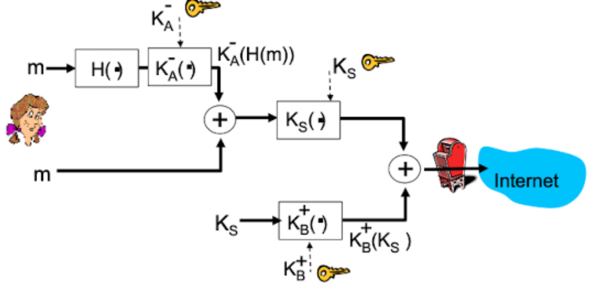
Public-key Certification

- Problem: To decrypt, we need sender's public key. **How do we ensure public key is actually the sender's?**
- Solution:
    - Certification Authority (CA): Maintains public database of everyone's public key
    - Everyone accesses this database to get someone's legitimate public key
- Problem: What if intruder intercepts communication between CA?
- Solution: CA signs its messages
- Problem: But we do not know CA's public key
- Solution: Make CA's public key a **universal knowledge** → OS has list of trusted CAs

Summary

> How do we ensure confidentiality, integrity, and authentication?

- Combine usage of 3 keys: Sender's private key, Receiver's public key, Session key



Firewall

> Isolates organization's internal net from larger Internet, allowing some packets to pass while blocking others

- Purpose:
  - Prevent Denial of Service (DoS) attacks: Spam TCP connections to waste resources and deny legitimate connections
  - Prevent illegal access of internal data
  - Allow only authorized access to inside network
- Types:
  - Stateless packet filter
  - Stateful packet filter
  - Application gateways
- Router **filters packet-by-packet**, based on:
  - Source/destination IP address
  - TCP/UDP source/destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits
- Access Control Lists (ACL): Table of rules, applied top to bottom to incoming packets
  - Essentially lots of switch statements
- Limitations:
  - IP spoofing: Router cannot know if data is really from claimed source
  - Can be bottleneck
  - Degree of communication vs. Level of security

Multimedia Networking

- Delivered as Over-The-Top (OTT), which only uses existing network infrastructure
- 3 application types:
  - **Streamed stored** audio, video (E.g. YouTube)
  - Conversational ("2-way live") voice/video over IP (E.g. Zoom)
  - Streaming live ("1-way live") audio, video
    - Delay can be longer
    - Usually done with CDNs

Video

- Video: Sequence of images displayed at constant rate
  - Image: Array of pixels
- Problem: Videos have **high bit rate**. How can we optimize?
  - Spatial Coding: Instead of sending N values of same repeating colors, send: 1 color and number N
  - Temporal Coding: Instead of sending next complete frame, send only differences from previous frame
- Bit rate:
  - Constant Bit Rate (CBR): Video encoding rate is fixed
    - Not responsive to complexity of video
    - Good for real-time encoding (Streaming): Consistent bit rate
  - Variable Bit Rate (VBR): Video encoding rate changes as amount of spatial and temporal coding changes
    - Good for on-demand videos: Longer time to process data

Audio

> Wave sampled at some rate

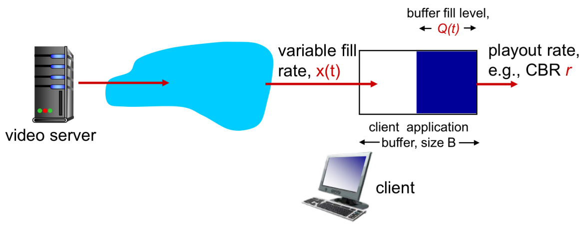
- Analog audio signal is in the form of a continuous wave
  - **Sampled at constant rate** (Analog-to-digital Converter (ADC))
  - Each sample quantized
    - Analogy: Estimating integral under a curve
    - Lower sampling rate → More quantization error (Analogy: Wider rectangles under curve leads to worse integral estimate)
  - Each quantized value represented by bits
  - E.g. 8,000 samples / second, 256 quantized values (8 bits) → 64,000 bps
- Receiver converts bits back to analog signal (Digital-to-analog converter (DAC))
  - Some quality reduction

Streaming Stored Video

- Requirements:
  - Streaming: Can begin playing **before** downloading entire file
  - Stored (at server/CDN): Can **transmit faster** than audio/video can be rendered
- Process:
  1. Video recorded in constant chunks (Like a step function)
  2. Server sends video over network
  3. Client plays video
- Problems:
  - **Network delays are variable** (Jitter)
  - Client interactivity: Can pause, fast-forward, etc.
  - Video packets may be lost
- Solution:

- Client-side buffering
- Playout delay: Delay a bit before playing video, so that some of the video is buffered

Client-side Buffering



1. Before playout of video at t_p, fill buffer to some level
2. Playout of video begins at t_p
3. Buffer fill level varies over time, since fill rate $x(t)$ varies and playout rate r is constant
    - If buffer is full, fill rate $x(t)$ will equal to playout rate r

> 💡 Tip: Can break down units into chunks

- Given average fill rate $\bar{x}$ and playout rate r:
    - $\bar{x} < r$: Buffer eventually starves and empties → Video freeze
    - $\bar{x} > r$: Buffer will not empty, given that playout delay can absorb variablility in $x(t)$
- Tradeoff: More playout delay → Less buffering delay

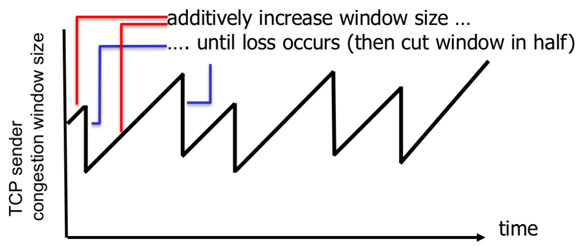
Streaming using UDP

- Server sends at **constant rate**
    - **Pushed-based** streaming (Server push)
- No congestion control → No rate control restrictions → Constant rate, despite packet drops or congestion
- Video chunks encapsulated using Real-time Transport Protocol (RTP)
    - RTP: Sequence number, time stamp, video encoding
- Control connection maintained **separately** using Real-time Streaming Protocol (RTSP)
    - Establishes and controls media sessions between endpoints
    - Handles client commands
- Cons:
    - No connection → Need for separate media control server (e.g. RTSP) → Higher cost and complexity
    - May be blocked by firewalls

Streaming using HTTP/TCP

- File retrieved using **HTTP GET**
    - **Pull-based** streaming (Client pull)
- Send at maximum possible rate under TCP
- Pros:
    - Passes more easily through firewalls

- Network infrastructure tuned for HTTP/TCP
- Cons:
  - TCP congestion control → Fill rate varies
  - Larger playout delay needed to ensure smooth TCP delivery
- Sawtooth behavior: Due to TCP probing for bandwidth

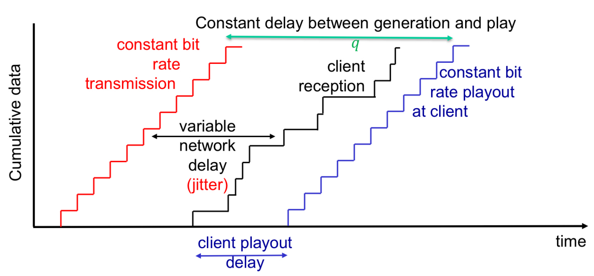


Voice-over-IP (VoIP)

- Requirements:
  - Minimize delay (<400 ms)
  - Minimize data loss
- Challenges: Internet (IP layer) is a **best-effort** service
  - **No guarantee** and upper bound on delay and packet loss
- Design:
  - Audio: Alternating talk spurts and silent periods
    - Packets generated only during **talk spurts**
  - Application layer header added to each chunk
    - Chunk and header encapsulated into UDP or TCP segment
- Packet loss and delay
  - Network loss: IP datagram lost due to network congestion
  - Delay loss: IP datagram arrives too late for playout

Fixed Playout Delay

> Constant delay of q msec after chunk is generated



- Different from client playout delay: q is after chunk is generated, not after receiving first chunk
- Given chunk has timestamp t:
  - Playout chunk at $t + q$ (Time buffer)
  - Chunks that arrive after $t + q$ are lost
- Why not just fix client playout delay? VoIP is live and conversational
- Tradeoff:

- Large q: Less packet loss
- Small q: Better interactive experience
- Solution: **Adaptive playout delay** adjustment
  - Compress/elongate silent periods (Why calls suddenly speed up when laggy)
  - Estimate network delay and adjust playout delay
- Delay Estimate after i-th packet:
  - r_i: Time received
  - t_i: Time send

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

- Estimate of average deviation of delay after i-th packet:

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

- d_i and v_i calculated for every received packet, but **used only at start of talk spurts**
  - 1st packet in talk spurt: Playtime time is $t_i + d_i + 4v_i$
  - Remaining packets in talk spurt played periodically

Recovery from Packet Loss

- Problem: TCP uses ACK/NAK → Each of them takes 1 RTT → Too slow
- Solution: Forward Error Correction (FEC)
  - Send enough bits to allow recovery without retransmission
  1. Simple FEC
     1. For every group of n chunks:
        - Create another chunk by XOR-ing the n original chunks and send $n + 1$ chunks
     2. If at most 1 chunk lost, it can be reconstructed by XOR-ing the rest
        - Why? XOR is commutative and associative
     - Cons:
        - Increase bandwidth by $1/n$
        - Need to wait for $n + 1$ chunks → Playout delay increased
  2. "Piggyback lower quality stream"
     1. Append lower resolution stream as redundant information of next chunk
     - Non-consecutive loss: Receiver can conceal loss by reconstructing with lower quality stream
  3. Interleaving
     1. Chunks divided into smaller units (e.g. 1 chunk = 4 units)
     2. 1 packet interleaves small units from different chunks
     3. Receiver reconstructs original chunks after receiving enough of their respective units (e.g. 4 chunks)
     4. If 1 packet lost, still have most of the original chunk
     - Pros: No overhead needed

- Cons: Need to wait for all respective units → Increased playout delay → Cannot use of live voice

Dynamic Adaptive Streaming over HTTP (DASH)

- Problem: Video-on-Demand (VoD) streaming uses HTTP streaming
  - Needs large client buffer → Wasteful
  - All clients, despite device and bandwidth, receive **same encoding of video**
- Solution: **Dynamic Adaptive** Streaming over HTTP
  - Server:
    - Divides file into many chunks
    - Each chunk **encoded at different rates**
    - Manifest file: Provide URLs for different encodings
  - Client:
    - Periodically measures bandwidth
    - Consults manifest and requests **one chunk at a time** depending on bandwidth
    - Needs to decide **when, what encoding rate, and where** to request chunk
  - Procedure:
    1. Server encodes video into different qualities and cuts into chunks
    2. Client downloads manifest file first
    3. Client executes adaptive bitrate algorithm to determine which segment to download
  - Pros: No firewall problems, Web caching works, Server is simple
  - Cons: Cannot provide low latency

Content Distribution Networks (CDN)

> Store/serve **multiple copies of videos at many geographically distributed sites** to bring videos closer to user

- Motivation: How to stream content to millions of simultaneous users?
- Option 1: 1 big server
  - Nope: Not scalable
- Solution: Content Distribution Networks
  - "Enter deep": Push CDN servers deep into access networks (ISPs)
  - "Bring home": Smaller number of larger clusters in IXPs