|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internet | Network of connected computing devices; infrastructure that connects hosts/end systems tgt - Allows hosts to communicate w each other - Packet switching network | | | Network edge: End hosts, servers, ...  Network core: ISPs, Routers, ... |
| Network Edge | Hosts run network applications -> apps communicate using protocols  Protocols define: 1) format & order of messages exchanged among network entities  2) actions taken upon receiving/sending the messages | | | Protocols e.g.: HTTP, FTP, SMTP, TCP, RTP  1 byte = 8 bits |
| Hosts access Internet through access network (residential/mobile/institutional access networks)  Home networks: hosts -> wireless access point/ethernet -> router -> modem  Enterprise networks: hosts -> wireless/ethernet switch -> institutional router -> Link to ISP (Internet) OR -> Institutional mail/web servers  Typically used in companies, universities. 10Mbps, ... 10 Gbps transmission rates. Hosts typically connect to ethernet switch  Wireless access networks connects hosts to router via base station/access point. Wireless LANs: Wi-Fi. Wide-area wireless access: cellular  Physical media: guided media: signals propagate in solid media: unshielded twisted pair, coaxial cable, fiber optics  Unguided media: signals propagate freely (Wi-Fi, cellular): radio waves | | | |
| Network Core | Mesh of interconnected routers. Data transmitted through network by:  1. Circuit switching: End-end resources allocated and reserved for "call" btw src & dest: - call setup required, - guaranteed performance  - circuit segment idle if not used by call (no sharing), - divide link bandwidth into "pieces" (freq or time division)  2. Packet switching. Host sending fn: - breaks message into smaller chunks, aka packets of length L bits  - transmits packets onto link at transmission rate R = link capacity = link bandwidth  - packet transmission delay = time to transmit L-bit packet into link = L (bits) / R (bits/sec)  2a. Store & forward: - Packets passed from 1 router to next using links on path from src to dest.  - entire packet must arrive at a router before it can be transmitted to the next link  2b. Routing & Addressing: - Routers determine src-dest route taken by packets (routing algos) - Each packet needs to carry src and dest info   |  |  |  |  | | --- | --- | --- | --- | |  | Setup/teardown | Resources | Service | | Circuit | required | Reserved | Guaranteed | | Packet | Not required | Shared on demand | Best effort service | | | | |
| Connecting the Internet | | A diagram of a network  Description automatically generatedHost connect to Internet via access ISPs  "Tier 3" Internet Service Providers (ISPs)/Access net: Singtel, M1, ...  Connecting all access ISP to e.o directly doesn't scale: O(N2) connections. Instead connect each access ISP to a global ISP  Can have multiple Global ISP/tier 1, which can be connected either through an Internet exchange point (IXP) or directly by a peering link  Content provider (Google/Akamai) can run their own network  Internet is a network of networks, organized into autonomous systems (AS), each owned by an organization  Internet standards are published as RFCs (Request For Comments) | | |
| Packet Loss | Packets queue in router buffers. Queue/buffer of a router has finite capacity. If queue is full -> packet is dropped/lost  Packet delay can be due to: 1) Nodal processing = dproc = processing delay. Check bit errors, determine output link, typically < msec  2) dqueue = queuing delay = time waiting in queue for transmission. Depends on congestion level of router  3) dtrans = transmission delay = L/R. L = packet length (bits). R = link bandwidth (bps)  4) dprop = propagation delay = d/s. d = length of physical link. s = propagation speed in medium (~2 \* 108 m/sec)  End-to-end packet delay = time taken for packet to travel from src to dest. Consists of all 4 types of delay above  Throughput = how many bits transmitted per unit time (for end-to-end communication). Link capacity/bandwidth only for a specific link | | | |
| Internet Protocol Layer | | | Application (treat Internet as black box) -> Transport (process-to-process data transfer) -> Network (routing of datagrams from host to host; Internet) -> Link (data transfer btw neighbouring network elements) -> Physical (bits on the wire/air) | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Network Apps | Application Layer Protocol is used by every Internet App. Network apps run on hosts and contain communicating processes  Network apps architecture: client-server OR peer-to-peer OR hybrid | | | | | | | |
| Architecture | | Client-Server Architecture. Server: waits for incoming requests. Provides requested service to client  Client: Initiates contact w server. Typically requests service from server. For Web, client is usually implemented in browser | | | | | | |
| P2P Architecture: No always-on server. Arbitrary end systems directly communicate  Peers request service from other peers (hosts), provide service in return to other peers. Highly scalable but diff to manage. | | | | | | |
| Hybrid: e.g. instant messaging. Chatting btw 2 users is P2P  Presence detection/location is centralized: user registers its IP address w central server when it comes online. User contacts central server to find IP addresses of friends | | | | | | |
| Transport service an app needs | 1) Data integrity: how much data loss is acceptable. 2) Throughput: some apps require min amount of bandwidth to be "effective"  3) Timing: low latency required?. 4) Security: encryption, authentication   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | App | File transfer | Email | Real-time audio/video | Stored audio/video | Interactive games | Text messaging | | Data Loss | No loss | | Loss-tolerant | | | No loss | | Throughput | Elastic | | Audio: 5kbps-1Mbps. Video: 10kbps-5Mbps | | Few kbps-10kbps | Elastic | | Time-sensitive | No | | Yes: 100s of msec | Yes: few sec | Yes: 100s of msec | Yes and no | | | | | | | | |
| App-layer Protocols | 1) Types of messages exchanged: request, response  2) Message syntax: what fields & how fields are delineated  3) Message semantics: meaning of info in fields | | | | | 4) Rules: when and how apps send & response to messages  5) Open protocols: defined in RFCs; allow for interoperability; HTTP, SMTP  6) Proprietary protocols: Skype | | |
| Overview | A diagram of a process  Description automatically generated1. IP Address (Globally unique address): Identifies host  - IPv4: 32-bit num: dotted decimal notation, e.g. 192.168.0.1  - IPv6: 128-bit: hexadecimal, e.g. 2001:0db8:85a3:0000:0000:8a2e:0370:7334  2. Port Number (Locally unique name): Identifies process  - 16-bit num (1 to 65535). 1-1023 are reserved. IANA assigns port num  Internet Transport Protocols: TCP & UDP TCP: connection oriented; flow controlled; congestion controlled; reliable  - does not provide timing, min throughput guarantee, security  UDP: unreliable data transfer  - does not provide flow & congestion control, timing, throughput guarantee or security | | | | | | | |
| Web and HTTP | A web page is an base HTML (Hyper-text Mark-up language) file w several other objects (other HTML file, JPEG image, Java applet,...)  Each object are addressable by a URL (uniform resource locator)  Web resources are requested and received using HTTP (HyperText Transfer Protocol)  HTTP = Web's application layer protocol using a client/server model. HTTP uses TCP as transport service. TCP in turn uses IP | | | | | | | |
| HTTP | 1a) Client initiates TCP connection to server  1b) Server accepts TCP connection request from client  2) HTTP messages are exchanged btw browser (HTTP client) and Web server (HTTP server) over TCP connection  3) TCP connection closed  RTT (round-trip time): time for packet to travel from client to server and back  HTTP response time: - 1 RTT to establish TCP connection (1a & 1b)  - 1 RTT for HTTP request and first few bytes of HTTP response to return (2)  - Non-persistent HTTP response time = 2 \* RTT + file transmission time | | | | | | HTTP/1.0: TCP connection is non-persistent  In step 2), only can request for 1 object. So if web page need to load multiple objects, steps 1 to 3 have to be repeated multiple times  HTTP/1.1 Sequential: TCP connection is persistent  In step 2) can request for multiple objects (send request, get response, then send next request, ...)  HTTP/1.1 Pipelining: TCP connection is persistent  In step 2) can make new request even before receiving response of old requests | |
| Non-persistent HTTP issues:  - requires 2 RTTs per object  - OS overhead for each TCP connection  - browsers often open parallel TCP connections to fetch reference objs | | | | Persistent HTTP: - server leaves connection open after sending response  - subsequent HTTP messages btw same client/server sent over the same TCP connection  - client sends requests as soon as it encounters a referenced obj (persistent with pipelining)  - as litle as 1 RTT for all the referenced objs | | | |
| With Sequential or Pipelining, server would only send response sequentially  With HTTP/2 Multiplexing, response can come back in any order, even partially (also uses pipelining)  HTTP/3 uses QUIC/UDP instead of TCP. - Remove TCP's head-of-line blocking, - Handle packet loss, - Estimate bandwidth | | | | | | | |
| HTTP Request | GET /~cs2105/demo.html HTTP/1.1 \r\n  Host: www.comp.nus.edu.sg \r\n  User-Agent: Mozilla/5.0 \r\n  Connection: close \r\n  \r\n | | | | <Request Type> <Path> <HTTP type>  # All browsers today call themselves Mozilla  # Whether to close TCP connection after completing request | | | |
| HTTP Response | HTTP/1.1 200 OK  Date: ...  ...  Keep-Alive: timeout=5, max=100  <!DOCTYPE html>  ... | | | Protocol and response code. 200 Ok: Request successful, requested obj follows  301 Moved Permanently. 304 Not Modified: Obj has not changed since specified date/time  403 Forbidden: Authentication error. 404 Not Found: Obj not found.  500 Internet Server Error: Unspecified error  Blank line to mark end of head  Start of data requested | | | | |
| Conditional GET | | | Dont send object if client cache has up-to-date cached version. Specify in <Last-Modified> header  Server response contains no object if cached copy is up-to-date: 304 Not Modified | | | | | |
| Cookies | HTTP was designed to be stateless (server maintains no info about past client requests). To maintain state (e.g. login info): use cookies  1. Server creates cookie and sent to client. 2. Cookie file kept on user's host, managed by user's browser  3. Client specify cookie in header field of HTTP request. 4. Server will receive cookie w http msg, then response with a cookie-specific action | | | | | | | |
| Caching | Can cache Images, Javascripts, CSS. Server dont send obj if client cache has up-to-date version  Specify cache in HTTP request header: *If-modified-since: <date>*  Server response contains no obj if cached copy is up to date: *HTTP/1.0 304 Not Modified* | | | | | | | |
| Domain Name Service (DNS) | DNS translates btw host name and IP address. E.g. www.comp.nus.edu.sg and 137.132.80.57. CMD: nslookup, dig  Client must carry out a DNS query to determine IP address corresponding to the server name prior to the connection  DNS Resource Record (RR): Mapping btw host names and IP addresses. RR Format: <name, value, type, TTL>   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Type | A (adress) | NS (name server) | CNAME (canonical name) | MX (mail exchange) | | Name | Hostname | Domain, e.g. nus.edu.sg | Alias for real name, e.g. www.comp.nus.edu.sg | Domain of email addr | | Value | IP Address | Hostname of authoritative name server for domain | Real name, e.g. www0.comp.nus.edu.sg | Name of mail server managing the domain | | | | | | | | |
| DNS Servers | DNS stores RR in distributed databases implemented in a hierarchy of many name servers. - 13 root name servers worldwide   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Root | | | | | | .com | | .edu | .sg | | | google | facebook | mit | ns | gov |   - Top-level domain (TLD) servers: for .com, .org, .net, .edu, ... and all top-level country domains, e.g. .uk, .sg, .jp  - Authoritative servers: Organization's own DNS server. Provide authoritative hostname to IP mappings for organization's named host  - Local DNS server: does not strictly belong to hierarchy. Each ISP (residential, company, uni) has 1 local DNS server aka default name server. When host makes DNS query, query is sent to its local DNS server -> retrieve name-to-address translation from local cache -> Local DNS server acts as proxy and forwards query into hierarchy if answer not found locally  Once a name server learns a mapping, it caches it, which expire after some TTL. DNS runs over UDP | | | | | | | |
| DNS query | Recursive: host local DNS server (ns1.nus.edu.sg) root DNS server TLD DNS server authoritative DNS server (a.ns.facebook.com) TLD root local host | | | | | | | Iterative: host local DNS server root local TLD local authoritative local host |

|  |  |  |
| --- | --- | --- |
| Processes | A host can run multiple processes. Apps runs in hosts as processes.  Within the same host, 2 processes communicate using inter-process communication (defined by OS) Processes in diff host communicate by exchanging messages (according to protocols)  IP address is used to identify host device (32-bit). Port num identifies process on host (HTTP: 80, POP: 25, WoW: 3724)  Network deliver packet to the right host. Transport dispatch packet to right process in host | |
| Sockets | Socket = software interface btw processes and transport layers. Process sends/receives messages to/from its socket. Programming-wise = a set of API calls  Conceptually, Socket = IP addr + Port num | 1) Stream socket (TCP socket) uses TCP (Transmission Control Protocol) as its transport layer  - Stream abstraction. - Connection-oriented. - Reliable  2) Datagram socket (UDP socket) uses UDP (User Datagram Protocol).  - Datagram abstraction. - Connection-less. - Unreliable (transmitted data may be lost, corrupted or received out-of-order) |
| UDP Socket | In UDP, data is sent as datagrams (packets). Only 1 socket is needed. Communication w anyone is through same socket  App creates each packet: - Specifies recipient (dest IP addr + port). - OS will attach return info (source IP + port)  Receiver identifies sender: - Extract source IP + port from received packet  For code on LHS:  Left is server. Right is client: | |
| TCP Socket | A diagram of a graph  Description automatically generatedTCP sockets: process establishes a connection to another process. While connection is in place, data flows btw processes in cts streams  TCP Server: - creates a welcome or listening socket. - When contacted by client, forks a new socket for server process to communicate w client  TCP Client: - creates a socket to establish a connection w server. - multiple sockets for multiple connections  A screenshot of a computer code  Description automatically generatedEvery connection has its own socket instance | |
| Layering | Application: message (msg) Transport: segment (Ht + msg. Ht contains src and dest port) Network: datagram (Hn + Ht + msg. Hn contains src and dest IP addr) Link Physical | |
| TCP vs UDP socket | How does TCP make segments when it communicate in streams:  TCP: 2 processes communicate as if there is a pipe btw them. Pipe remains in place until 1 of the 2 processes closes it.  - When 1 of the processes wants to send more bytes to the other process, simple write data to that pipe  - Sending process doesn't need to attach a dest IP addr and port num to the bytes in each sending attempt (which is also reliable)  UDP: programmers need to form UDP datagram packets explicitly and attach dest IP addr/port num to every packet | |

|  |  |  |
| --- | --- | --- |
| Transport Layer Services | Internet transport layer protocols: 1) TCP: connection-oriented & reliable. 2) UDP: connection-less and unreliable  Transport layer protocols run in hosts. - Sender side: breaks app message into segments (as needed), passes them to network layer (IP layer)  - Receiver: reassembles segments into msg, pass to app layer. - Packet switches (routers) in btw only check dest IP addr to decide routing | |
| Transport vs Network Layer | A diagram of a network system  Description automatically generatedNetwork layer provides host-to-host, best-effort and unreliable communication  Underlying network may: - corrupt/drop/re-order packets. - deliver packets after an arbitrarily long dely  End-to-end reliable transport service should: - guarantee packets delivery & correctness, - deliver packets to app in same order they are sent   |  |  |  | | --- | --- | --- | | rdt | Scenario | Features | | 1.0 | no error/perfectly reliable | nothing | | 2.0 | data corruption | Checksum, ACK/NAK | | 2.1 | 2.0 + ACK/NAK corruption | 2.0, seq num | | 2.2 | Same as 2.1 | NAK free | | 3.0 | 2.1 + packet loss | 2.1, timeout/re-transmit |   Complexity of rdt protocol depends on unreliability of channel  CS2105 only consider 1 dirn of data transfer but control info may flow both ways  Alternating-bit protocol  Reliable delivery service = no corrupt or loss. Might out of order | |
| Reliable Data Transfer (rdt) |
| Finite State Machine (FSM) | Used to describe sender and receiver of a protocol  rdt 1.0: LHS = sender. RHS = receiverA diagram of a call  Description automatically generatedA diagram of a diagram  Description automatically generated | |
| rdt 2.0 | A diagram of a blue circle with arrows and a blue circle with white text  Description automatically generatedAssume channel may flip bits in packets  To detect bit errors: receiver can use checksum  To recover from bit errors: 1) Acknowledgements (ACKs) - receiver tells sender packet received is OK  2) Negative ACKs (NAKs) - receiver tells sender packet has errors  3) Sender retransmits packets on receipt of NAK  Stop-and-wait protocol: Sender sends 1 packet at a time, then waits for received response  Problem: ACK/NAK may be corrupted/garbled also  Sol: Sender just retransmits when it receives corrupted feedback -> may cause retransmission of correctly received packet | |
| rdt 2.1 | A diagram of a call center  Description automatically generatedTo handle duplicates: - sender retransmits curr pkt if ACK/NAK is garbled. Use Alternating Bit Protocol: seq num only 0 or 1  - Sender adds seq num to each pkt  A diagram of a process  Description automatically generated- Receiver discards duplicate pkt | |
| rdt 2.2 | A diagram of a diagram  Description automatically generatedReplace NAK w ACK of last correctly received packet  A diagram of a receiver  Description automatically generatedDuplicate ACKs at sender result in same action as NAK: retransmit curr pkt | |
| rdt 3.0 | A diagram of a process  Description automatically generatedAssume may have corruption, lose pkts, incur long packet delay BUT will not reorder pkts  To handle pkt loss: - Sender waits "reasonable" amt of time for ACK  - Sender retransmits if no ACK received till timeout  If packet or ACK is just delayed but not lost:  - timeout will trigger retransmission  - generate duplicates, but received may use seq num to detect it  A diagram of a network  Description automatically generated- received must specify seq num of pkt being ACK  Receiver does nothing if it receives a corrupted data packet | |
| Pipelining | Utilization = time spent sending/total time (fraction of time link is actually being used). Throughput (bps) = L (bits) / total time  During dtrans = L/R, link is being used. But during RTT, link not being used as sender just waiting for ACK  Problem with stop-and-wait: Usender = dtrans / (dtrans + RTT), where RTT = time btw last bit get transmitted and sender getting ACK  Pipelining: sender allows multiple, "in-flight", yet-to-be ACK pkts - range of seq num must be incr. - buffering at sender and/or receiver  Assumption same as rdt 3.0  Send 3 pkt: Usender = 3 \* dtrans / (dtrans + RTT) compared to stop-and-wait which will take 3 \* dtrans / [3 \* (dtrans + RTT)] = dtrans / (dtrans + RTT)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Generic forms of pipelined protocols | # unACK packets | ACK style | out-of-order | timer | retransmit | | Go-Back-N | N pkts in pipeline | cumulative | discarded | oldest unACK | all unACK | | Selective Repeat | selective | buffered | each unACK | 1 unACK | | |
| Go-Back-N | GBN sender: - use a sliding window to keep track of unACKed pkts. - can have up to N unACKed pkts in pipeline.  - insert k-bits seq num in pkt header - keep timer for oldest unACKed pkt. - timeout(n): retransmit pkt n & all subsequent pkts in the window  - receive ACK n: move sliding window starting from n + 1 -> In this sliding window, send pkt which were not sent before, i.e. not in-flight  GBN receiver: - only ACK pkts that arrive in order (simple received: only rmb expectedSeqNum)  - discard out-of-order pkts and ACK the last in-order seq num (cumulative ACK: "ACK m" = all packets up to m are received) | |
| Selective Repeat | Receiver individually ACKs all correctly received pkts: Buffers out-of-order pkts, as needed, for eventual in-order delivery to upper layer  Sender maintains timer for each unACKed pkt: When timer expires, retransmit only that unACKed pkt | |
| Sender. - timeout(n): resend pkt n, restart timer  - Data from above: if next available seq num in window, send pkt  - ACK(n) in [send\_base, send\_base+N]: mark pkt n as received;  if n is smallest unACKed pkt, advance window base to next unACKed seq num | Receiver. - pkt n in [rcv\_base, rcv\_base+N-1]: send ACK(n);  out-of-order to be buffered;  in-order to be delivered (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt  - pkt n in [rcv\_base-N, rcv\_base-1]: ACK(n)  - otherwise: ignore |

|  |  |  |
| --- | --- | --- |
| UDP (User Datagram Protocol) [RFC 768] | UDP adds very little service on top of IP: - Connectionless multiplexing/de-multiplexing. - Checksum  Transmission is unreliable. To achieve reliable transmission over UDP: App layer need to implement error detection and recovery mechanism  Transport-layer multiplexing: Data from multiple sources/sockets sent using only 1 transmission channel | |
| For Connectionless De-multiplexing:  UDP sender: -Creates socket w local port num. -When creating datagram to send to UDP socket, sender must specify dest IP addr & port num  When transport layer receives a UDP segment: - Check dest port num in segment. - Direct UDP segment to the socket w port num  - IP datagrams (from diff sources) w same dest port num will be direted to the same UDP socket at dest | |
| A diagram of a computer  Description automatically generated with medium confidenceUDP segment structure  - source port/dest port/length/checksum all have 16 bits = 2 bytes  - if no app data. UDP header has length of 8 bytes, so length = 0000 0000 0000 1000  Checksum algos: - Cyclic Redundancy Check (CRC) - Message Digest v5 (MD5),  - Secure Hash Algo 1 (SHA-1), - Secure Hash Algo 2 (SHA-2)  - UDP/TCP Checksum (RFC 768)  UDP Checksum: to detect errors (single bit flips) in transmitted segment  Sender compute checksum value cs. Include checksum value in UDP checksum field  To compute checksum: 1) Split segment into 16-bit ints (checksum initially 0)  2) Add integers w wrap around carry. 3) Compute 1's complement  Receiver compute checksum value cr. If cr != 1111 1111 1111 1111, then confirm got error. If all 1s, then ONLY high chance that no error detected  UDP checksum is optional. TCP is not | |
| Pros: - No connection set-up = reduce delay. - No connection state at sender or receiver = need less resources  - Small header size = less overhead. - No congestion control = can blast as fast as possible | |
| TCP (Trans-mission Control Protocol) [RFC 793, RFC 1122, RFC 1323, ...] | Connection-oriented = handshaking (exchange of control messages) before sending app data  Reliable, in-order byte stream: - App passes data to TCP and TCP forms segments in view of MSS. - UDP, app forms packets: DatagramPacket  Flow control and congestion control: Prevent unncessary losses | |
| Connection-oriented De-mux: TCP connection/socket is identified by 4-tuple, (srcIP, srcPort, destIP, destPort)  - Receiver uses all 4 values to direct a segment to appropriate socket  - Note will always have 1 welcome socket only for handshake. No other data will be transferred in this socket | |
| A diagram of a computer program  Description automatically generated with medium confidenceTCP segment structure  - Similarly, src port, dest port, checksum are 16 bits  - Offset records size of headers in 32-bit words. Minimal header is 5 rows of 32 bits.  - Options is optional. So if have more header fields, offset have to be set.  E.g. offset = 6 -> 1 more extra 32 bits word in header  - ACK Indicate if ACK# is valid  A screenshot of a computer  Description automatically generated- Sequence number: byte number of the first byte of data in the segment  E.g. 500 000 byte file w 1000 byte segments  - Acknowledgement number: Seq. # of next byte of data expected  - TCP only ACKs up to the missing byte (cumulative ACK) | |
| A close-up of a mathematical equation  Description automatically generatedTCP Echo Server: TCP is full duplex, i.e. bi-directional data flow  ACKs are "piggybacked" on data segment  Seq num = 42 + len('Hello') = 47 = Ack num  Size of segment determined by Maximum Segment Size (MSS) which is typically derived from link layer's MTU. Segment here = data | |
| TCP Sender  *loop(forever)*  *switch(event)*  *event: data received from app*  *create TCP segment w nextSeqNum*  *if (timer not currently running)* ## Sender only keep 1 timer  *start timer*  *pass segment to IP*  *nextSeqNum += len(data)* | *event: timer timeout*  ## Retransmit only oldest unACK segment  *retransmit unacknowledge segment w smallest seq num*  *start timer*  *event: ACK received, with ACK num y* ## Cumulative ACK  *if (y > sendBase)*  *sendBase = y*  *if (there are still unacknowledged segment)*  *start timer* |
| TCP Receiver. Cumulative ACK prevents retransmission   |  |  |  | | --- | --- | --- | | Segment Received | In-order segment | All data already ACKed: - Wait up to 500ms for next segment. (Delayed ACK) - Send ACK if no segment arrives | | Outstanding segment not ACKed: - Send cumulative ACK immediately | | Out-of-order segment w higher seq# than expect. (Gap is created): - Immediately send Duplicate ACK of expected byte | | | Segment partially or completely fills gap: - Send cumulative ACK immediately | |   A screenshot of a computer  Description automatically generated | |
| TCP Timeout value: Too short = premature timeout and unnecessary retransmissions. Too long = slow reaction to loss. Timeout must > RTT  Estimating RTT: 1) Take Sample RTT = RTTs once every RTT. 2) Compute Estimated RTT = RTTe = (1 - ) \* RTTe + \* RTTs (Exponential Weighted Moving Average; EWMA. Typical = 1/8)  Setting Retransmission Time Out (RTO): 1) Compute Deviation of RTT = RTTdev = (1 -ß) \* RTTdev + ß \* |RTTs - RTTe|. Typical ß = 1/4  2) RTO Interval is set to RTTe + 4 \* RTTdev. (4 \* RTTdev is for safety margin) | |
| TCP Fast Retransmission [RFC 2001]. Timeout is often relatively long. Soln: If 3 Duplicate ACKs are received, resend segment immediately | |
| A close-up of a mathematical equation  Description automatically generatedTCP Connection Establishment using 3-way handshake: Agree on connection and exchange parameters. Also to establish and synchronize eq nums for data transmission Header: ACK and SYN  Half-open connections: 1) SYN Flooding: DoS style attack by sending SYN  A diagram of a function  Description automatically generated with medium confidence2) SYN/ACK Flooding: DoS style to overwhelm network  TCP Closing Connection: Each side closes their own side of the connection. Send segments with FIN bit set  - No more sending of data after FIN, but can still receive | |
| TCP Flow Control. Header = receive window. Data is delivered from TCP to App layer. Receiver buffers data to app layer.  In Receiver buffer, could have data awaiting deliver and some extra space. Extra space = rwnd = receive window = tell sender how much data it can send. If rwnd is 0, i.e. full. For sender to know when it empties, it periodically send a 0-data segment aka 0-window probe | |
| URG, PSH, RST, urgent pointer are legacy header fields. Other box btw offset and URG are for experimental use | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IP addr | | | To identify host or router. - Globally unique. - 32-bit identifier. - Associated w each host or router interface (connection btw host/router and physical link. Routers typically have multiple interfaces. Host typically has 1 or 2 interfaces [wired Ethernet, wireless 802.11])  Binary: 11011111 00100010 00000011 10000001 = Decimal 223 34 3 129. So dotted-decimal IP addr notation: 223.34.3.129  Routing: determines source-dest route taken by packets (use routing algos)  Forwarding: move packets from router's input to appropriate router output | |
| Subnets | | | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Dest | A.B.1.2 | A.B.3.2 | P.Q.1.10 | P.Q.4.181 | X.Y.3.1 | X.Y.3.2 | X.Y.3.3 | | Output link |  |  |  |  |  |  |  |   Reduce Forwarding table from to   |  |  |  |  | | --- | --- | --- | --- | | Dest | A.B.\*.\* | P.Q.\*.\* | X.Y.\*.\* | | Output Link |  |  |  |   Reduce size of table by a systematic IP addr allocation, resulting in Address/Route Aggregation  Subnet = network formed by a group of "directly" interconnected hosts  - Host in same subnet can physically reach each other w/o intervening router. - Connect to outside world through a router  - To determine the subnets, detach each interface from its host or router, creating islands of isolated networks | |
| IP addr & Subnet | | | IP address comprises 2 parts: network (subnet) prefix [n bits] + host ID [32 - n bits]. Total 32 bits  - Host in same subnet have same network prefix of IP addr. - Subnet prefix of IP addr can be of arbitrary length  - Addr format: a.b.c.d/x, where x = num of bits in subnet prefix of IP addr. - 200.23.16.42/23. Subnet contains 29 IP addresses  Subnet Mask to determine which subnet an IP addr belongs to. - Set all subnet prefix bits to 1, and host ID bits to 0  E.g. IP addr: 200.23.16.42/23: 11001000 00010111 00010000 00101010. Subnet mask = 11111111 11111111 11111110 00000000  Use bitwise AND to add IP addr and Subnet mask to get Network addr = 11001000 00010111 00010000 00000000 | |
| IP addr Allocation | | | Organization obtain a block of IP addr by buying from registry or rent from ISP's addr space  ISP charge based on block size, ie. num of IP available in the subnet they allocate to you  Hierarchical addressing allows efficient advertisement of routing info. If forwarding table have multiple matches, use longest prefix match  ISP get a block of address from ICANN (Internet Corporation for Assigned Names and Numbers) who allocates addresses, manages DNS, assigns domain names, resolves disputes. | |
| Special IP | | | 0.0.0.0/8 Non-routable meta-address for special use  127.0.0.0/8 Loopback addr. A datagram sent to an addr within this block loops back inside the host | 10.0.0.0/8 or 172.16.0.0/12 or 192.168.0.0/16 For private addr, can be used w/o any coordination w an Internet registry  255.255.255.255/32 Broadcast addr. All hosts on the same subnet receive a datagram w such a dest addr |
| Private IP addr | | | 10.0.0.0/8 or 172.16.0.0/12 or 192.168.0.0/16. Are not globally unique. Used in organizations, universities, homes  Are not routable on the backbone Internet, but are routable within an organization | |
| NAT (Network Address Translation) | | | A diagram of a plane  Description automatically generatedAll datagrams leaving local network have same source NAT IP addr: 137.132.228.5  Within local network, hosts use private IP addr 172.26.184.\* for communication   |  |  | | --- | --- | | NAT translation table | | | WAN side | LAN side | | 137.132.228.5, 5001 | 172.26.184.3, 3213 | | 137.132.228.5, 5002 | 172.26.184.5, 3213 |   1) Host 172.26.184.3 sends datagram to 128.119.40.186, 80  2) NAT router changes datagram source addr, port, updates table  3) Reply arrives. Dest addr is 137.132.228.5, 5001. 4) NAT router changes datagram dest addr to 172.26.184.3, 3213  NAT routers must: - replace (source IP addr, port num) of every outgoing datagram to (NAT IP addr, new port num)  - Remember (in NAT translation table) the mapping from (source IP addr, port num) to (NAT IP addr, new port num)  - Replace (NAT IP addr, new port num) in destination fields of every incoming datagram w corresponding (source IP addr, port num)  No need to rent a range of public IP addresses from ISP: just 1 public IP for NAT router  All hosts use private IP addr. Can change addr of hosts in local network w/o notifying outside world. Can change ISP w/o changing addr of hosts in local network. Hosts inside local network not explicitly addressable and visible by outside world (security)  NAT support 16-bit port number field. So over 60000 simultaneous connections can be made w a single WAN-side addr | |
| IPv4 Datagram Format | | IP header = 20 bytes. ver = IP protocol version num (4). IP datagram length = length of header + data  A purple box with red text  Description automatically generatedTTL = # of remaining hops (decrement at each router). Upper layer protocol = TCP/UDP/ICMP  Header checksum = same checksum in Transport. - updated at each hop since TTL changes  Diff links have diff MTU (Max Transfer Unit) = max amt of data a link-level frame can carry.  Too large IP datagrams may be fragmented by routers and only reassembled at final dest  2nd row use for fragmentation.  Flag (frag flag) = set to 1 if there is next fragment from same segment. 0 if last fragment  Offset = expressed in units of 8 bytes. Specifies offset of fragment relative to beginning of original unfragmented IP datagram  Original header: length = 1200, ID = x, flag = 0, offset = 0.  Fragmentation: IP1) length = 500, ID = x, flag = 1, offset = 0.  IP2) length = 500, ID = x, flag = 1, offset = 60 (480/8).  IP3) length = 240, ID = x, flag = 0, offset = 120 (960/8) | | |
| DHCP | | | Dynamic Host Configuration Protocol - allows host to dynamically obtain its IP addr from DHCP server when it joins network  - IP address is renewable - allow reuse of address (only hold addr while connected)  DHCP runs over UDP (DHCP is in app layer). DHCP server port num = 67. DHCP client port num = 68  0.0.0.0 is special IP. 255.255.255.255 is broadcast addr. 223.1.2.4 is IP that will be assigned to your host   |  |  |  | | --- | --- | --- | | Arriving Client | Datagram sent/received | DHCP server, 223.1.2.5 | | DHCP discover | src: 0.0.0.0, 68; dest: 255.255.255.255, 67; yiaddr: 0.0.0.0; transaction ID: 654 |  | |  | src: 223.1.2.5, 67; dest: 255.255.255.255, 68; yiaddr: 223.1.2.4; ID: 654; lifetime: 3600s | DHCP offer | | DHCP request | src: 0.0.0.0, 68; dest: 255.255.255.255, 67; yiaddr: 223.1.2.4; ID: 655; lifetime: 3600s |  | |  | src: 223.1.2.5, 67; dest: 255.255.255.255, 68; yiaddr: 223.1.2.4; ID: 655; lifetime: 3600s | DHCP ACK |   DHCP may also provide host: - IP addr of local DNS server, - Network mask, - IP addr of 1st-hop router (default gateway) | |
| Internet | | | A diagram of a network  Description automatically generatedNetwork of networks: hierarchy of autonomous systems (AS), e.g. ISPs, each owns routers and links.  Routing in Internet:  1) Intra-AS routing: finds a good path btw 2 routers within an AS. Commonly used protocols = RIP, OSPF  - Single admin, so no policy decisions needed  - Routing mostly focus on performance  2) Inter-AS routing: handles interfaces btw ASs. Standard protocol = BGP.  - Admin often wants to control how its traffic is routed, who routes through its net, etc. - Policy may dominate over performance | |
| Intra-AS Routing | Can view a network of routers as a graph, where vertices = routers and edges = physical links btw routers  Can associate a cost to each link: cost could always be 1, or inversely related to bandwidth, or related to congestion, or distance, ...  Routing = finding a least cost path btw 2 vertices in a graph. 2 types of routing algos  1) Link state algos: - all routers have complete knowledge of network topology & link cost (routers periodically broadcast link costs to e.o)  - Use Dijkstra algo to compute least cost path locally (using global map)  2) Distance vector algos: - routers know physically connected neighbors and link costs to neighbors.  - routers exchange "local views" w neighbors and update own "local views" asynchronously (based on neighbors' view)  - Iterative computation: 1. Swap local view w direct neighbors. 2. Update own's local view. 3. Repeat till no more change to local view  - Distributed, self-stopping: each node notifies neighbors only when its DV changes | | | |
| A grid with letters and numbers  Description automatically generatedLet c(x, y) = cost of link btw routers x and y (∞ if x and y are not direct neighbors) & dx(y) = cost of least-cost path from x to y (from x's pov). c(a, x, y) = c(a, x) + dx(y)  Bellman-Ford Eqn: dx(y) = minv{c(x, v) + dv(y)} where min is taken over all direct neighbors v of x  - x needs to know the cost from each of its direct neighbor to y.  - Each neighbor v sends its dist vector (y, k) to x, telling x that cost from v to y is k  - So dist vector table should have min cost at the end  E.g. each router x, y, z sends its dist vectors to its directly connected neighbor  - If x finds out y have a path to z that is cheaper than x currently knows, x update its dist vector to z accordingly. In addition, x note down that all packets for z should be sent to y. This info is used to create forwarding table of x | | | |
| RIP (Routing Info Protocol) implements the dist vector algo. Uses hop count as the cost metric (i.e. insensitive to network congestion)  Exchange routing table every 30s over UDP port 520. - "Self-repair": if no update from neighbor router for 3mins, assume neighbor has failed | | | |
| ICMP - Internet Control Message Protocol | | |  |  |  | | --- | --- | --- | | Type | Code | Description | | 8 | 0 | echo request (ping) | | 0 | 0 | echo reply (ping) | | 3 | 1 | dest host unreachable | | 3 | 3 | dest port unreachable | | 11 | 0 | TTL expired | | 12 | 0 | bad IP header |   Used by hosts & routers to communicate network-level info. ICMP msg are carried in IP datagrams (ICMP header start after IP header)  - Error-reporting: unreachable host/network/port/protocol. - Echo request/reply (used by ping)  ICMP header = Type + Code (aka subtype) + Checksum + others  ping = see if remote host will respond to us (do we have a connection?)  traceroute = send a series of small packets across a network, w diff TTL, and attempts to display the route/path that the messages would take to get to a remote host  When TTL = 0, pkt is discarded and an ICMP error msg is sent to datagram's src addr | | |
| Network Layer Services | | IP = network layer. RIP, BGP = network service, but in app layer, and in data plane.  Data plane: local, per-router fn. Determines how datagram arriving on router input port is forwarded to router ouput port.  Control plane: network-wide logic. Determines how datagram is routed among routers along end-end path from src to dest host.  - traditional routing algos: implemented in routers OR software-defined networking (SDN): implemented in (remote) servers  Routing protocol dont route pkts, only creates forwarding table. Pkt forwarding only done in data plane, by routers using a forwarding table  Router longest prefix match: often w ternary content addressable memories (TCAMs). Content addressable: retrieve addr in 1 clock cycle | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Summary | Application: HTTP, DNS, DHCP. | Transport: TCP, UDP. | Network: IP, ICMP |
| Intro | Communication channel = the transmission medium of the data signals. E.g. copper wire, optical fiber, radio, satellite, ...  Node = devices exchanging data. E.g. hosts, routers, .... Link = communication channels that connect adjacent nodes | | |
| 1 possible mtd is to inter-connect the N nodes. But each link needs to be addressed and does not scale (N-1 links needed)  Soln: Inter-Connect the N nodes via a broadcast link. Each link needs to be addressed (framing = add header and trailer to IP datagram), need define protocol (link access control), need handle errors (detection and correction) | | |
| Link Layer | Sends datagram btw adjacent nodes (hosts/routers) over a single link. Adjacent = single hop connects the 2 nodes. Frame = layer-2 packet.  - IP datagrams are encapsulated in link-layer frames for transmission. - Diff link-layer protocols may be used on diff links.  - Data-link layer has responsibility of transferring datagram from 1 node to physically adjacent node over a link | | |
| Link access control = when multiple nodes share a single link, need to coordinate which nodes can send frames at a certain point of time  Error detection = Errors usually caused by signal attenuation or noise. Receiver detects presence of errors (may signal sender for retransmission or simply drops frame)  A diagram of a connection between a couple of adapters  Description automatically generatedError correction = receiver identifies and corrects bit error(s) w/o resorting to retransmission  Reliable delivery = seldom used on low bit-error link (e.g. fiber) but often used on error-prone links (e.g. wireless link)  Link layer is implemented in "adapter" (aka NIC = network interface card) or on a chip. E.g. Ethernet card, Wi-Fi adapter  Adapters are semi-autonomous, implementing both link & physical layers | | |
| Error detection and Correction | A diagram of a computer program  Description automatically generatedEDC = Error Detection and Correction bits. D = data protected by error checking, may includer header fields  Error detection schemes are not 100% reliable. Usually, larger EDC field yields better detection (and even correction)  1) Checksum: TCP/UDP/IP. OR 2) Parity Checking OR 3) CRC: link layer  1) Checksum treat segment contents as seq of 16-bit integers. Then calculate 1's complement of the sum of segment contents | | |
| 2a) Parity Checking: single bit. Suppose binary info to be sent, D, has d bits  In an even parity scheme: - sender simply includes 1 additional bit (aka parity bit) s.t. the total number of 1s in the d+1 bits is even  - can detect odd num of single bit errors. Cannot detect even num of single bit error  A black and white diagram with black text  Description automatically generated- works exceptionally well (mathematically): since prob of multiple bit errors is low (if errors are independent)  - However, errors often clustered tgt in "bursts": prob of undetected errors in a frame can approach 50%  2b) Parity Checking 2-D: - the d bits in D are divided into i rows and j cols.  - A parity value is computed for each row and for each col: resulting i + j + 1 parity bits comprise the link-layer frame's error-detection bits,  - EDC field = di+1,1 … di+1,jd1,j+1 … di,j+1 di+1,j+1  - Can detect and correct single bit errors in data. - Can detect any 2-bit error in data | | |
| 3) Cyclic Redundancy Check (CRC): want to transfer non-binary num D w/o error. R = r digit error detection code  Need to generate R s.t. sender can compute R easily. Receiver can verify integrity of D easily. Soln: use properties of division  Use a special r digit num G = Generator. E.g. let D = 21027845, r = 3 and G = 401  A computer code with numbers and arrows  Description automatically generated with medium confidenceCreate new num X by appending r 9's to D. X = 21027845 999 = D \* 10r + (10r - 1). Find remainder y of X/G, i.e. y = X % G = 281.  Message M being transmitted is M = X - y = 21027845999 - 281 = 21027845718 which is divisible by G  On receiver end, find remainder = y1 = M % G. If y1 = 0: no error detected. Else: discard faulty data  For CRC, D = d data bits as a binary number. G = generator of r + 1 bits, agreed by sender and receiver beforehand. R = r bit CRC  Calculations are done modulo 2. Does not have carries for addition or borrows for subtraction. Start from 1st LHS '1'  Both add and subtract are identical to XOR, x + y = x - y = x y. E.g. 0 + 1 = 0 - 1 = 0 1 = 1. 1011 0101 = 1110  For performing division, append r 0's to D. Due to properties of modulo 2, remainder directly gives us R  D = 101110, r = 3, G = 1001. Sender sends (D, R) = (101110, 011)  Receiver knows G, so divides (D, R) = 101110011 by G. If non-0 remainder: error detected  - Easy to implement on hardware.  - Powerful error-detection coding widely used in practice (Ethernet, Wi-Fi): can detect all odd num of single bit errors.  - CRC of r bits can detect: all burst errors of ≤ r bits & all burst errors of > r bits w prob 1 - 0.5r  CRC aka Polynomial code: - a k-bit frame is reagarded as the coefficient list for a polynomial w k terms, ranging from xk-1  E.g. 110001 = 1x5 + 0x4 + 0x3 + 0x2 + 0x1 + 1x0 = x5 + x4 + 1 | | |
| Multiple Access Links and Protocols | Several computer monitors connected to each other  Description automatically generatedTypes of Network Links: 1) Point-to-point link: sender and receiver connected by a dedicated link  E.g. protocols: Point-to-Point Protocol (PPP), Serial Line Internet Protocol (SLIP). No need for multiple access control  2) Broadcast link (shared medium): multiple nodes connected to shared broadcast channel  - when node transmits a frame, channel broadcasts the frame and every other node receives a copy  E.g. Wi-Fi, Satellite, Ethernet w bus topology (shown on RHS) | | |
| Multiple Access Protocols: in broadcast channel, if 2 or more nodes transmit simultaneously = collision if node receives 2 or more signals at the same time. Multiple Access Protocols in Increasing complexity:  1) Channel partitioning: - divide channel into fixed, smaller "pieces" (e.g. time, slots, freq). - Allocate piece to node for exclusive use  2) Taking turns: each node take turns to transmit  3) Random Access: - channel is not divided, collisions are possible (2 or more transmitting nodes). - "recover" from collisions  Given a broadcast channel of rate R bps, the desired properties are a) Collision free. b) Efficient: when only 1 node wants to transmit, it can send at rate R. c) Fairness: when M nodes want to transmit, each can send at average rate R/M.  d) Fully decentralized: no special node to coordinate transmission  Mandantory: coordination about channel sharing must use channel itself, i.e. no out-of-band channel signaling | | |
| 1a) TDMA (time division multiple access): Access to channel in "rounds".  Each node gets fixed length time slots in ea round: Length of time slot = data frame transmission time  Frame here = collection of N time slots. To disambiguate, use data frame or time frame  On diagram, nodes 1,3,4 have data to send, slots 2,4,6 are idle  a) Collision Free. b) Inefficient as unused slots go idle. Max throughput for a node is R/N. c) Perfectly Fair. d) Decentralized | | |
| 1b) FDMA (freq division multiple access): Channel spectrum divided into freq bands (e.g. 4-8KHz)  Each node is assigned a fixed freq band. Unused transmission time in freq bands go idle  a) Collision Free. b) Inefficient as unused slots go idle. Max throughput for a node is R/n. c) Perfectly Fair. d) Decentralized | | |
| A computer screen with a few monitors  Description automatically generated with medium confidence2a) Polling: Requires 1 of the nodes to be designated as a master node.  Master node polls each of the nodes in a round-robin fashion  - master informs node 1, it can transmit up to some max num of frames  - after node 1 transmits some frames, master node tells node 2 it can transmit up to the max num of frames...  a) Collsion Free. b) Higher efficiency. Overhead of polling. c) Perfectly Fair.  d) Not decentralized. Master node is a single point of failure | | |
| 2b) Token Passing: A special frame, token is passed from 1 node to next, sequentially  When a node receives a token: - hold onto token only if some frames to transmit (can only send up to a max num of frames before it forwards token to next node). - otherwise if no frames to transmit, forward token to next node  a) Collision Free. b) Higher efficiency. Overhead of token passing. c) Perfectly Fair. d) Decentralized  Downsides: - token loss can be disruptive (due to data frame loss or system bugs). - Node failure can break the ring | | |
| 3) Random Access Protocols: When node has data to send, transmit at full channel data rate R. No a priori coordination among nodes | | |
| A diagram of numbers and letters  Description automatically generated3a) Slotted ALOHA: - All frames are of equal size, L bits.  - Time is divided into slots of equal length. - Length = time to transmit 1 frame = L/R  - Nodes start to transmit only at beginning of a slot. - Time is synchronized at each node  When node has fresh frame to send:  - wait until beginning of next slot and transmits entire frame in slot  - if no collision: data transmission is success. - if collision: data trasmission is a failure  - retransmit frame in each subsequent slot w prob p until success  a) Not collision free. b) Efficient since when only 1 node is active, it gets a throughput of R.  Not efficient when there are many active nodes (max efficiency of only 37%). - Slots are wasted due to collision and being empty  - 100Mbps system will give only 37 Mbps. c) Perfectly Fair. d) Decentralized | | |
| 3b) Pure (unslotted) ALOHA: Even simpler than slotted ALOHA. No time slots. No synchronization  When node has fresh frame to send: - transmits entire frame immediately  - if no collision: data transmission is success. - if collision: data trasmission is a failure. - wait for 1 frame transmission time. Then retransmit frame w prob p until success. - Chance of collision increases: frame sent at t0 collides w other frames sent in (t0 - 1, t0 + 1)  a) Not collision free. b) Efficiency when only 1 node is active, it gets throughput of R  Not efficient when there are many active nodes (max efficiency only 18%). Slots wasted due to collision and being empty  - 100 Mbps system only give 18 Mbps. c) Perfectly Fair. d) Decentralized | | |
| - A major flaw in ALOHA is a node's decision to transmit is independent of activity of other nodes attached to broadcast channel.  3c) CSMA (Carrier Sense Multiple Access): If channel sensed idle: transmit entire frame. If channel sensed busy: defer transmission.  However, collisions can still occur: due to propagation delay, 2 nodes may not hear e.o's transmission immediately  - A major flaw in ALOHA and CSMA is a node does not stop transmitting even when collision is detected | | |
| 3d) CSMA/CD (Collision Detection): If channel sensed idle: transmit entire frame. If channel sensed busy: defer transmission.  If collision detected: abort transmission (retransmit after a random delay). So this doesn't waste the whole frame transmission time | | |
| CSMA/CD Backoff Algo. For ALOHA, wait 1 frame transmission time and retransmit frame w prob p until success.  A diagram of a graph  Description automatically generatedProb of collision in subsequent time slots remain the same, and even incr if new node start transmitting  Soln: adapt retransmission attempts to estimated curr load. More collision = longer back-off interval  Binary Exponential backoff:  After 1st collision: - choose K at random from {0, 1}. - wait K time units before retransmission. - p = 1/2  After 2nd collision: - choose K from {0, 1, 2, 22-1}. - wait K time units before retransmission. - p = 1/4  After mth collision: - choose K at random from {0, 1, 2, 3, 4, 5, 6, …, 2m-1}. - p = 1/2m  For Ethernet, 1 time unit = 512 bit transmission time. Ethernet requires min frame size = 64 bytes  If Ethernet has speed 10 Mbps, and use K = k, then wait for k \* 512 / 107 seconds  Minimum Frame Size. If frame size is too small, collision happens but may not be detected by sending nodes, i.e. no retransmission. Need dtrans ≥ dprop to be able to detect  a) Not collision Free. b) Efficient. c) Fair. d) Decentralized. 3d) used in Ethernet | | |
| Summary | 1a) Mobile phone (GSM). 1b) Radio, satellite systems. 2a) Bluetooth. 2b) FDDI and token ring. 3a/b) Wireless packet switched network | | |

|  |  |  |
| --- | --- | --- |
| Link Layer Addressing | Every adapter (NIC) has a MAC address (aka physical or LAN addr). MAC = Media Access Control. - Used to send & receive link layer frames.  MAC addr typically 48 bits, burned in NIC ROM (Read-only Memory) (sometimes software settable).  E.g. 5C-F9-DD-E8-E3-D2 (hexadecimal/base 16 notation) = 0101 1100 1111 1001 1101 1101 1110 1000 1110 0011 1101 0010  MAC addr allocation is administered by IEEE. 1st 3 bytes identifies vendor of an adapter. Broadcast addr = FF-FF-FF-FF-FF-FF | |
| Ethernet | LAN (Local Area Network) = computer network that interconnects computers within a geographical area  LAN technologies: IBM Token Ring = IEEE 802.5 standard. Ethernet = IEEE 802.3 standard. Wi-Fi = IEEE 802.11 standard. ...  Ethernet = "Dominant" wired LAN technology. MTU = Max Transmission Unit | |
| 802.3 Ethernet Standards. Diff speeds: 2, 10, 100 (Mbps), 1, 10, 100 (Gbps)  Diff physical layer media: cable, fiber optics  MAC protocol and frame format remain unchanged  Sending NIC (adapter) encapsulates IP datagram in Ethernet frame  If NIC receives a frame w matching dest addr, or w broadcast addr: passes data in frame to network layer protocol  Otherwise: NIC discards frame  - Full Duplex.  A close-up of a box  Description automatically generated**Data** max size = 1500 bytes. Max size = link MTU  Data min size = 46 bytes to ensure collision will be detected  **CRC**: corrupted frame will be dropped  **Type** = Higher layer protocol. E.g. IP, AppleTalk, ARP, ...  - Type field permits Ethernet to multiplex network layer protocols.  - Analogous to protocol field in network-layer datagram and port-number fields in transport-layer segment  **Preamble**: 7 bytes w pattern 10101010 (AAHex). Followed by 1 byte w pattern 10101011 (ABHex), aka "start of frame"  - Used to synchronize receiver and sender clock rates | |
| Preamble provides a "square wave" pattern telling receiver sender's clock rate  - tells receiver width of a bit  - which is impt if there is a long string of bits of the same value, e.g. 19 or 20 zeros | |
| Ethernet Data Delivery Service is Unreliable: receiving NIC doesn't send ACK or NAK to sending NIC  - data in dropped frames will be recovered only if initial sender uses higher layer rdt (e.g. TCP); otherwise dropped data is lost  Ethernet's multiple access protocol: CSMA/CD w binary (exponential) backoff | |
| Ethernet: Physical Topology | 1) Bus Topology: Original Ethernet LAN used a coaxial bus to interconnect nodes  Is a broadcast LAN: All transmitted frames received by all adapters connected to the bus: all nodes can collide w each other  Cons: Backbone cable = if damaged, entire network fail. Diff to troubleshoot problems. Slow & not ideal for large networks (have collisions) | |
| A computer network connection with a computer  Description automatically generated with medium confidence2) Star Topology  2a) Hub: Nodes are directly connected to a hub. Hub = physical-layer device that acts on individual bits rather than frames. - When bit arrives from 1 interface, hub simply re-creates the bit, boosts its energy strength & transmits the bit onto all other interfaces  Pros: Cheap. Easy Maintenance (modular design of network). Cons: very slow and not ideal for larger networks (collisions)  2b) Switch: Nodes directly connected to a switch. Switch = layer-2 device that acts on frames rather than individual bits.  No collisions. A bona-fide store-and-forward packet switch | |
| Ethernet Switch | A computer network with several computers connected to each other  Description automatically generatedLink-layer device used in LAN. - Examine incoming frame's MAC addr: selectively forward frame to 1 or more outgoing links. - Store and forward Ethernet frames. - Uses CSMA/CD to access link  Transparent: hosts are unaware of presence of switches. Plug-and-play (self-learning): switches don't need to be configured (nodes create own ARP tables w/o intervention from network admin). RHS = switch w 6 interfaces  Multiple simultaneous transmissions. Nodes have dedicated direct connection to switch  - Switches buffer packets. - Ethernet protocol used on each incoming link but no collisions  Switching: A-to-A' and B-to-B' can transmit simultaneously, w/o collisions.  Switches can be connected in hierarchy. Switch Forwarding Table uses Selective Forwarding  Each switch has a switch table. Format of entry: <MAC addr of host, interface to reach host, TTL>. E.g. To reach C, use interface 3  Switch learns which hosts can be reached through which interfaces: - when frame received, switch "learns" location of sender  Frame filtering/forwarding: When frame received at switch:  1) Record incoming link, MAC addr of sending host. 2) Index switch table using MAC dest addr  3) If entry found for destination: a) If dest on segment from which frame arrived: drop frame. b) Else forward frame on interface indicated  4) Else flood: forward on all interfaces except arriving interface | |
| Switches vs Routers | Routers: - Check IP address. - Store and forward. - Compute routes to dest. - Sits on Network layer  Switches: - Check MAC addr. - Store and forward. - Forward frame to outgoing link or broadcast. - Sits on Link layer | |
| ARP (Address Resolution Protocol) | | Use ARP [RFC 826] to know MAC addr of a receiving host, given its IP addr. - ARP provides a query mechanism to learn MAC addr  Each IP node has an ARP table: - stores mapping of IP addr and MAC addr of other nodes in same subnet. - <IP addr; MAC addr; TTL>  TTL typically a few mins. - Switch itself has no mac addr. |
| Sending Frame in Same Subnet | Suppose A send data to B and both on same subnet  Case 1) A knows B's MAC addr from its ARP table  - create frame w B's MAC addr and sent it  - only B will process this frame  - other nodes may receive but will ignore this frame  Case 2) B's MAC addr not in A's ARP table  1. A broadcasts an ARP query packet, containing B's IP addr  - Dest MAC addr set to FF-FF-FF-FF-FFF-FF  - All other nodes in same subnet will receive this ARP query packet, but only B will reply to it  2. B replies A w its MAC addr. - Reply frame sent to A's MAC addr. 3. A caches B's IP-to-MAC addr mapping in its ARP table (until TTL expires) | |
| Sending Frame to Another Subnet | A computer screen shot of a number  Description automatically generatedA diagram of a circular object with a red and black text  Description automatically generatedA creates IP datagram w IP source A, dest B  A screen shot of a computer  Description automatically generatedIf A creates frame above, all adapters in net 1 will ignore this frame due to mismatch of dest MAC addr  Instead, A creates IP datagram w IP source A, dest B (B IP now 130.168.0.1). A creates link-layer frame w R's MAC addr as dest addr, frame contains A-to-B IP datagram (shown on right)  A screen shot of a computer  Description automatically generatedSo frame sent from A to R. Frame received at R, datagram removed, passed up to IP layer in R.  R then forwards datagram w IP source A, dest B.  R creates link-layer frame w B's MAC addr as dest addr and R's MAC addr as src addr, frame contains A-to-B IP datagram (shown on left) | |
| IP addr vs MAC addr | IP addr: - 32 bits in length. - Network-layer address used to move datagrams from src to dest. - Dynamically assigned; hierarchical (to facilitate routing)  MAC addr: - 48 bits in length. - Link-layer addr used to move frames over every single link. - Permanent, to identify the hardware (adapter) | |

|  |  |  |
| --- | --- | --- |
| Network Security | 1) Eavesdrop: intercept messages. 2) Actively insert messages into connection. 3) Impersonation: can fake (spoof) source address in packet  4) Hijacking: "take over" ongoing connection by removing sender or receiver, inserting oneself in place (Man in the middle)  5) Denial of service: prevent service from being used by others (e.g. by overloading resources) | |
| 1) Confidentiality: only sender, intended receiver should "understand" message contents. (Substitution Cipher, Symmetric Key, Public Key)  2) Authentication: sender, receiver want to confirm identity of each other. (Digital signature)  3) Message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) w/o detection. (Cryptographic Hash)  4) Access and availability: services must be accessible and available to users. (Firewall) | |
| Crypto-graphy | Aim: Confidentiality, Authentication, Message integrity  Allows sender to disguise data s.t intruder can gain no info from intercepted data. Allow receiver to recover original data from disgused data  Notation: m = plaintext message (message in original form). KA(.) = encryption algo, with key KA. KA(m) = ciphertext (encrypted message)  Key = string of numbers or characters, provided as input param to the encryption/decryption algo  KB(.) = decryption algo, w key KB. KB(KA(m)) = m | |
| 2 types of encryption: 1) Symmetric Key Cryptography: sender and receiver use the same key, i.e. KA = KB (key same but algo might not be)  2) Asymmetric Key Cryptography (aka Public Key Cryptography): sender and receiver use diff key, i.e. KA ≠ KB | |
| Symmetric Key Crypto-graphy | Sender and receiver need to decide on common key prior to communication via some other secure means, like face to face meeting | |
| Caesar's cipher = form of substitution cipher. Fixed shift of alphabet. E.g. right shift by 3: abcdefghi...vwxyz -> defghi...wxyzabc  Encryption Key: only need shift number, 25 possible values. Weakness: Easy to break w brute force search | |
| Monoalphabetic cipher = form of substitution cipher. Substitute 1 letter for another  Encryption key: mapping from set of 26 letters to set of 26 letters. 26! ~ 1026 Mappings possible  Still can break encryption w statistical analysis. Letters e (13%) and t (9%) are most freq letters.  - Knowing that particular 2 and 3 letter occurrences of letters appear quite often tgt (e.g. "in", "it", "the", "ion", "ing")  - If intruder has some knowledge abt possible contents of msg, then even easier to break code | |
| Polyalphabetic encryption: Use multiple mappings for each letter instead of only 1 mapping as in Monoalphabetic Cipher  E.g. use n substitution ciphers, C1, C2, …, Cn. Define a cycling pattern: e.g. n = 4: C1, C3, C4, C3, C2  For each new plaintext symbol, use subsequent substitution pattern in cyclic pattern: "dog": d from C1, o from C3, g from C4 | |
| Block Ciphers: msg to be encrypted is processed in blocks of K bits. Each block is encrypted independently  To encode a block, cipher uses a one-to-one mapping. Num of keys = 2K!  DES: Data Encryption Standard. - US encryption standard [NIST 1993]. - 56-bit symmetric key, 64-bit block  - 56-bit-key-encrypted phrase decrypted (brute force) in less than a day. To make DES more secure, use 3DES = encrypt 3 times w 3 diff keys  AES: Advanced Encryption Standard. - Symmetric-key NIST standard, replaced DES. - 128 bit blocks; 128, 192 or 256 bit keys  - If machine can brute force decrypte DES in 1s, 128-AES would take 149 trillion years | |
| Breaking an encryption scheme | | 1) Ciphertext only attack: only has ciphertext to analyze. 2) Known-plaintext attack: have plaintext corresponding to ciphertext  3) Chosen-plaintext attack: can get ciphertext for chosen plaintext |
| Public Key Crypto-graphy | Drawback w symmetric key crypto: sender and receiver need to agree on shared secret key  For public key crypto: - sender, receiver don't share secret key. - sender uses a public encryption key known to all  - receiver uses a private decryption key known only to receiver. - radically diff approach [Diffie-Hellman'76, RSA'78]  Let = public key, = private key. Plaintext msg = m -> ciphertext = (m) -> plaintext msg = m =  Requirements: 1) need and s.t. m = . 2) Given public key , should be impossible to compute private key | |
| Use modular arithmetic. x mod n = x % n. 1) [(a mod n) + (b mod n)] mod n = (a + b) mod n. 2) [(a mod n) - (b mod n)] mod n = (a - b) mod n.  3) [(a mod n) \* (b mod n)] mod n = (a \* b) mod n. From 3) can get (a mod n)d mod n = ad mod n | |
| Message = just a bit pattern, can be uniquely represented by an int. So encrypting key same as encrypting a number  RSA: Creating a public/private key pair: 1) Choose 2 large prime numbers p, q  2) Compute n = pq, z = (p-1)(q-1). 3) Choose e (where e < n) that has no common factors with z (i.e. e and z are "relatively prime")  4) Choose d s.t. ed - 1 is exactly divisible by z (i.e. ed mod z = 1). 5) Public key = (n, e) = and Private key = (n, d) =  1) To encrypt message m (Note: m < n), compute c = me mod n. 2) To decrypt received bit pattern, c, compute cd mod n  - Works because cd mod n = (me mod n)d mod n = med mod n = m | |
| E.g. let p = 5, q = 7. Then n = pq = 35. z = (p-1)(q-1) = 24. Choose e = 5 s.t. (e < n & e and z are relatively prime).  Choose d = 29 s.t. ed mod z = 1. Suppose message has bit pattern of 00001100, then m = 12. me = 248832. c = me mod n = 17  Decrypt: c = 17. cd = 481968572106750915091411825223071697. m = cd mod n = 12 | |
| Another property: , i.e. using public key, then private key same as using private key then public key  (me mod n)d mod n = med mod n = mde mod n = (md mod n)e mod n | |
| RSA in practice: exponentiation in RSA computationally intensive. DES ≥ 100 times faster than RSA, but needs prior knowledge of key KS  Combine both approaches. Select key KS. Use RSA to transfer KS. Use KS as symmetric key in DES for encrypting data for this session  Symmetric key KS = session key | |
| Message Integrity | Internet checksum: produces fixed length digest (16-bit sum) of message. - is many-to-one (diff message have same checksum)  - Checksum is designed to detect accidental errors not attacks.  CRC: better than checksum but still poor. Output is biased to input (minor change in input produce minor change in output). Also many-to-1 | |
| Cryptographic Hash Fn: If a fn H(.) that takes an input m and produces fixed size msg digest (fingerprint). - Many-to-1  H(m) will be of fixed length. The hash fn is s.t. it is computationally infeasible to find any 2 diff messages x and y s.t. H(x) = H(y)  Informally, computationally infeasible for an intruder to substitute 1 message for another message  MD5 hash fn widely used (RFC 1321): compute 128-bit message digest. SHA-1 also used: US standard [NIST], 160-bit message digest  But SHA-1 and MD5 are cryptographically broken. NIST formally deprecated use of SHA-1. Replaced by SHA-2, SHA-3 | |
| So naive soln to ensure message integrity: send (m, H(m)). But attacker can just replace (m, H(m)) with (m', H(m')) and receiver won't know | |
| Soln: Message Authentication Code. Sender and receiver share a "Authentication key" s. To ensure msg integrity: send (m, H(m+s)), where H(m+s) = message authentication code. This works as s is a secret key known to receiver and no one else.  Receiver can generate authentication code directly from m and compare w received code | |
| Digital Signatures = signed msg digest | Cryptographic techniques analogous to handwritten signatures. Signature must be 1) Verifiable and 2) Unforgeable  Simple digital signature for msg m: Sender signs m by encrypting w his private key , creating signature , Then sends (m, )  Receiver receives (m, ). Verifies m signed by sender by applying sender public key to to check  If correct, whoever signed m must have used sender's private key.  Receiver thus verifies that sender signed m, no one else signed m, and sender signed m and not m'  Non-repudiation: receiver can take m and signature to court and prove that sender signed m | |
| Optimization: computationally expensive to public-key-encrypt long messages. Aim: fixed-length, easy-to-compute digital "fingerprint"  Hash fn properties: produces fixed-size msg digest (fingerprint) & given msg digest x, computationally infeasible to find m s.t. x = H(m)  Sender: Large message m -> Hash fn H -> H(m) -> digital signature (encrypt w sender's private key, ) -> encrypted msg digest  Final send (m,  Receiver: For encrypted msg digest -> digital signature (decrypt w sender's public key, ) -> = H(m)  And for Large message m -> Hash fn H -> H(m). Then compare both H(m) are equal | |
| Public-key certification: Certification authorities (CA) who maintain a public DB of everyone's public key  Anyone who receives a message and want to decrypt with sender's public key, will access this DB to find the key  CA signs it's message to ensure no one can intercept communication with CA and alters it  But we dk CA's public key. Soln: make this a universal knowledge. Maintain a list of CAs trusted a priori. OS has a list of "Trusted Root Certification Authorities"  CA binds public key to a particular entity, E. E (person, router) registers its public key with CA  - E provides "proof of identity" to CA. - CA creates certificate binding E to its public key. - Certificate containing E's public key digitally signed by CA. Sender sends identifying info and public key to CA.  If someone needs sender's public key, CA encrypt with CA private key, to give certificate for sender's public key, signed by CA w  Receiver gets certificate containing encrypted . Then decrypt with CA public key to get actual sender's public key | |
| Operatio-nal Security: firewalls | Firewall isolates organization's internal net from larger Internet, allowing some packets to pass, blocking others.  - prevent DoS attacks: e.g. SYN flooding: attacker established many bogus TCP connections, no resources left for "real" connections  - prevent illegal modification/access of internal data. - allow only authorized access to inside network  3 types of firewalls: stateless packet filters OR stateful packet filters OR application gateways | |
| Stateless packet filtering: internal network connected to Internet via router firewall  - router filters packet-by-packet, decision to forward/drop packet based on 1) source IP addr, dest IP addr. 2) TCP/UDP src and dest port num  3) ICMP msg type. 4) TCP SYN and ACK bits   |  |  | | --- | --- | | Policy | Firewall Setting | | All incoming, outgoing UDP flows are blocked | Block incoming and outgoing datagrams w IP protocol field = 17 | | Prevents external clients from making TCP connections w internal clients, but allow internal clients to connect to outside | Block inbound TCP segments w ACK = 0 | | 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 | | |
| ACL (access control lists): table of rules, applied top to bottom to incoming packets: (action, condition) pairs   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | action | src addr | dest addr | protocol | src port | dest port | flag bit | | allow | 222.22/16 | outside of 222.22/16 | TCP | > 1023 | 80 | any | | allow | outside of 222.22/16 | 222.22/16 | TCP | 80 | > 1023 | ACK | | deny | all | all | all | all | all | all | | |
| Limitations of firewalls: - IP spoofing: router can't know if data "really" comes from claimed src. - Can become a bottleneck  - tradeoff: degree of communication w outside word, level of security | |
| Secure  e-mail | To provide secrecy, sender authentication, msg integrity, use own private key , receiver public key , newly created symmetric key KS  Sender: m -> H(.) -> = . Concat (m, ) = m -> KS(.) = KS(m ). Also KS -> (.) = (KS)  Sends: (KS(m ), (KS))  Receiver: (KS) -> = KS. Then KS(m ) -> KS(.) = (m + ). Then can get m,  m -> H(m). And -> = H(m). Compare both H(m) for authentication | |

|  |  |
| --- | --- |
| Multimedia Networking | 1) Streaming stored audio, video. Streaming can begin playout before downloading entire file  - Stored at server/CDNs: can transmit faster than audio/video will be rendered (implies storing/buffering at client) 2) Conversational ("two-way live") voice/video over IP: - interactive nature of human-to-human conversation limits delay tolerance (cannot delay more than 400ms)  3) Streaming live ("one-way live") audio,video. Typically done w CDNs |
| Multimedia video | Sea of images displayed at constant rate. Digital image: array of pixels (each pixel represented by bits)  Most salient characteristic of video is its high bit rate.  To reduce data usage, compress video: use redundancy within and btw images to decr num of bits used to encode image  a) Spatial Coding (within image): instead of sending N values of same color, send only 2 values: color and num of repeated values (N)  b) Temporal coding (from 1 image to next): instead of sending complete frame at i+1, send only diff from frame i  CBR (constant bit rate): video encoding rate fixed. - not responsive to complexity of video  - need to set bitrate relatively high to handle more complex segments of video.  - Consistency of CBR makes it well-suited for real-time encoding. E.g. for real-time live streaming  VBR (variable bit rate): video encoding rate changes as amt of spatial, temporal coding changes  - VBR best suited for on-demand video due to longer time to process data  E.g. MPEG 1 (CD-ROM) 1.5Mbps. MPEG 2 (DVD) 3-6 Mbps. MPEG4/H.264 (often used in Internet, < 2Mbps). H.265 (4K video, > 10Mbps) |
| Multimedia audio | Analog audio signal: - sampled at constant rate. - Telephone: 8000 samples/sec. - CD music: 44100 samples/sec  Each sample quantized, i.e. rounded. Each quantized value represented by bits, e.g. 8 bits for 256 values.  E.g. 800 samples/sec w 256 quantized values (8 bits) = 64000 bps  Receiver converts bits back to analog signal (DAC): some quality reduction  CD: 1.411 Mbps. MP3: 96,128,160 kbps  Internet telephony: > 5.3 kbps  ADC = analog-to-digital converter.  DAC = digital-to-analog converter |
| Streaming stored video | A diagram of a video game  Description automatically generatedChallenges: - continuous playout constraint (once client playout begins, playback must match original timing, but network delays are variable (jitter))  - client interactivity: pause, fast-forward, rewing, jump through video  - video packets may be lost, retransmitted  A diagram of a staircase  Description automatically generatedClient-side buffering and playout delay: compensate for network-added delay, delay jitter  Let x(t) = variable fill rate, Q(t) = buffer fill level, CBR r = playout rate, B = client app buffer size  1) Initial fill of buffer until playout begins at tp. 2) Playout begins at tp  3) Buffer fill level varies over time as fill rate x(t) varies and playout rate r is constant  Playout buffering: average fill rate (), playout rate (r): - < r: buffer eventually empties (causing freezing of video playout until buffer fills)  - > r: buffer will not empty provided initial playout delay is large enough to absorb variability in x(t)  - initial playout delay tradeoff: buffer starvation less likely w larger delay but larger delay until user begins watching |
| Streaming multimedia: UDP | Server sends at rate appropriate for client. Often, send rate = encoding rate = constant rate: push-based streaming (server push)  UDP has no congestion control, hence transmission w/o rate control restrictions  RTP = real time protocol. RTSP = real time streaming protocol  - Short playout delay (2-5s) to remove network jitter. Error recovery: app level, time permitting. - Video chucks encapsultated using RTP.  - Control connection maintained separately using RTSP: used for establishing and controlling media sessions btw endpoints. Client issue commands such as play, record and pause  Drawbacks: - need for a separate media control server like RTSP, incr cost and complexity. - UDP may not go through firewalls |
| Streaming multimedia: HTTP | Multimedia file retrieved via HTTP GET, pull-based streaming (client pull). Sends at max possible rate under TCP  Pros: HTTP/TCP passes more easily through firewalls. Network infra (like CDNs and Routers) finetuned for HTTP/TCP  Cons: Fill rate fluctuates due to TCP congestion control, retransmissions (in-order delivery). - arger playout delay: smooth TCP delivery rate |
| Conversa-tional Multimedia: VoIP (voice over IP) | End-end delay requirement: needed to maintain "conversational" aspect. Higher delays noticeable, impair interactivity  < 150ms = good. > 400ms = bad. Includes app-level (packetization, playout), network delays. Data loss > 10% = conversation unintelligible  Challenge: Internet (IP layer) is a best-effot service. No upper bound on delay nor on percentage of packet loss  Speaker's audio: - alternating talk spurts, silent periods. - pkts generated only during talk spurts. - 20ms chunks at 8KB/sec = 160B of data  - App-layer header added to each chunk. - Chunk + header encapsulated into UDP or TCP segment: app sends segment into socket every 20 ms during talk spurt |
| A diagram of steps with red and blue text  Description automatically generatedNetwork loss: IP datagram lost due to network congestion (router buffer overflow, ...)  Delay loss = IP datagram arrives too late for playout at receiver: - delays (processing, queueing in network; end-system (sender, receiver) delays). - VoIP apps typically use UDP to avoid Congestion control  Loss tolerance: depending on voice encoding, loss concealment, pkt loss rates btw 1% and 10% can be tolerated  End-to-end delays of 2 consecutive pkts: diff can be more or less than 20ms (transmission time diff) |
| Fixed playout delay: receiver attempts to playout each chunk exactly q msecs after chunk was generated. Diagram shown above  - If chunk arrives after t + q, where chunk has time stamp t: data arrives too late for playout = data lost  Every chunk will have seq num and timestamp. Tradeoff in choosing q: large q = less packet loss. small q = better interactive experience |
| A diagram of a graph  Description automatically generated with medium confidenceAdaptive playout delay. Aim: low playout delay, low late loss rate. Soln: adaptive playout delay adjustment  - estimate network delay, adjust playout delay at beginning of each talk spurt  - silent periods compressed and elongated: chunks still played out every 20ms during talk spurt  Adaptively estimate packet delay (EWMA = exponentially weighted moving avg):  Delay estimate after ith pkt = di = (1 -)di-1 + (ri - ti), where ri - ti = measured delay of ith pkt = time received - time sent (timestamp)  Estimate of avg deviation of delay after ith pkt = vi = (1 - )vi-1 + |ri - ti - di|  Estimates, di and vi calculated for every received pkt, but used only at start of talk spurt  - For first pkt in talk spurt, playout timei = ti + di + 4vi. - remaining pkts in talk spurt played out periodically |
| Recovery from pkt loss. If use ACK/NAK, too slow as each takes ~ 1 RTT  Soln: use forward error correction (FEC) = send enough bits to allow recovery w/o retransmission (similar to 2D parity)  Simple FEC: for every grp of n chunks: {- create redundant chunk by XOR-ing n original chunks. - send n+1 chunks}  - Can reconstruct original n chunks if at most 1 lost chunk from n+1 chunks, w playout delay  Cons: Increasing bandwidth by factor 1/n. Playout delay incr during pkt loss (receiver waits for n+1 chunks before playout)  Another FEC scheme: "piggyback lower quality stream". Send lower resolution audio stream as redundant info (norminal stream PCM at 64 kbps and redundant stream GSM at 13 kbps). Non-consecutive loss: receiver can conceal loss  A diagram of a network  Description automatically generatedGeneralization: can also append (n-1)st and (n-2)nd low-bit rate chunk  Interleaving to conceal loss:  - Audio chunks divided into smaller units, e.g. four 4ms units per 20ms audio chunk  - pkt contains small units from diff chunks  - if pkt lost, still have most of every original chunk (concealed by pkt repetition or interpolation)  0 no redundancy overhead, but incr playout delay, even w/o error |
| DASH: Dynamic Adaptive Streaming over HTTP | Video-on-Demand (VoD) video streaming increasingly uses HTTP streaming  - Simple HTTP streaming just GETs a (whole) video file from an HTTP server  Cons: wasteful, needs large client buffer. - all client receive the same encoding of video, despite variation in device/network bandwidth  Soln: DASH. For server: - divides video file into multiple chunks. - each chunk stored, encoded at diff rates  - manifest file: provides URLs for diff encodings  For client: - periodically measures server-to-client bandwidth. - consulting manifest, request 1 chunk at a time (chooses max coding rate sustainable given curr bandwidth. can choose diff coding rates at diff points in time depending on available bandwidth)  "Intelligence" at client: client determines - when to request chunk (so that buffer starvation/overflow don't occur)  - what encoding rate to request (higher quality when more bandwidth available)  - where to request chunk (can request from URL server that is "close" to client or has high available bandwidth |
| Data encoded into diff qualities and cut into short segments (streamlets, chunks)  Client first downloads manifest file, which describes the available videos and qualities  Client/player executes an adaptive bitrate algo (ABR) to determine which segment to download next |
| Pros: server is simple, i.e. regular web server (no state, proven to be scalable). - No firewall problems (use port 80 for HTTP)  - standard (image) web caching works  Cons: - DASH is based on media segment transmissions, typically 2-10s in length. - By buffering a few segments at client side, DASH don't provide low latency for interactive, 2-way apps (e.g. video conferencing) |
| CDN: Content Distribu-tion Networks | Using just single large server doesn't scale as single point of failure, point of network congestion, long path to distant clients, multiple copies of video sent over outgoing link  CDN store/serve multiple copies of videos at multiple geographically distributed sites:  a) enter deep: push CDN servers deep into many access networks. - usually at ISP. - close to users. - used by Akamai, 1700+ locations  b) bring home: smaller number (10's) of large clusters in IXPs (internet exchange point) but not within access networks. - Used by Limelight  CDN stores copies of content at CDN nodes. - Client request content, service provider returns manifest  - Using manifest, client retrieves content at highest supportable rate. - May choose diff rate or copy if network path congested |
| Summary | Encoding exploiting: spatial redundancy, temporal reundancy. Client-side Buffering: playout delay, congestion control  VoIP: FEC, Error concealment. Video Streaming: UDP, HTTP, DASH, CDN |