COMP3331-midsem

Computer Networks and Applications (University of New South Wales)

COMP3331

INTRODUCTION TO COMPUTER NETWORKS

1.1. What is the Internet

- Nuts + bolts view – an interconnection of different computer networks
    - Millions of connected computing devices
        - Hosts – end systems + running network apps
    - Communication links
        - Fiber, copper, radio, satellite
        - Bandwidth – transmission rate
    - Packet switches – forward packets (chunks of data)
        - Routers + switches
    - Internet – network of networks
        - Interconnected ISPs (Internet Service Providers)
    - Protocols – controls sending, receiving msgs
    - Internet standards – rules for protocols that would be applied for deployed routers
        - Has to go by rigorous testing, acceptance by community
        - RFC – Request for comments
        - IETF – Internet Engineering Task Force
- Service view – an infrastructure that provide service to networked applications
    - E.g. Web, VoIP, email, games, e-commerce, social nets, etc.
    - Provides programming interface to apps
        - Hooks that allow sending and receiving app programs to 'connect
    - Protocols – defines format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt
        - E.g. TCP, IP, HTTP, Skype, 802.11
        - All communication activity in Internet governed by protocols
        - Some are open or proprietary
            - Skype – proprietary, does not make code available for changes
        - TCP connection request → TCP connection response → get html file → receive (handshaking)

1.2. Network edge – end systems, access network, links

- Network structure
    - Network edge
        - Hosts – client + servers
        - Servers often in data center
    - Access networks, physical – wired, wireless communication links
        - Clients access the service of the internet through access networks
    - Network core – interconnected routers, network of networks
- Access networks + physical media
    - How to connect end systems to edge router
        - Residential access nets
            - Multiple devices working on modem (works both a switch, network)
        - Institutional access networks (school, company) – Ethernet
            - End systems typically connected to Ethernet switch
            - 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps transmission rate
        - Mobile access networks – wireless access networks

            - Shared wireless access network connects end system to router
                - Via base station (access point)
            - Wireless LANs – within building
            - Wide-area wireless access – provided by telco operator e.g. 3G, 4G
    - Access net – Digital Subscriber Line (DSL)
        - Use existing telephone line to central office DSLAM
            - Voice (PTSN), data transmitted at different frequencies over dedicated line to central office
                - Dedicated connection – your responsibility to share among devices at client end
            - Everything independent of each other on same cable
                - Splitter – has 2 cable, for DSL modem + phone line
                - Filters the voice (low-pass) and data (high-pass) into independent streams in the same outgoing cable (PTSN, broadband, upstream, downstream)
            - Data over DSL phone line goes to Internet
            - Voice over DSL phone line goes to telephone net
        - ADSL – Asymmetry digital subscriber line
            - Different data rate for upload and download
            - More downstream bandwidth – most users need priority on downloading
                - Prevents client to running own server at home
            - < 2.5 Mbps upstream transmission rate (typically < 1 Mbps)
            - < 25 Mbps downstream trans. rate (typically < 10 Mbps)
        - Cable length from DSLAM influences the strength of signal's bandwidth
    - Access net – cable network
        - Data, TV transmitted at different frequencies over shared cable distribution network
            - Shared connection – multiple subscribers sharing the same cable headend
        - HFC – hybrid fiber coax & asymmetric (< 30 Mbps down, 2Mbps up)
        - Frequency division multiplexing – different channels transmitted in different frequency bands (video, data, control)
    - Fiber to the home (FTTH)
        - Fully optical fiber path all the way to the home e.g. Verizon FIOS, Google, NBN
        - Active (like switched Ethernet) or passive optical
    - Power cabling
        - Ethernet over power – powerline networking, ethernet is carried on the power cabling in your home
        - Power over Ethernet – device draws power from the Ethernet itself, would not require power, has access point and is drawing power from elsewhere

1.3. Network core – packet switching, circuit switching, network structure

- The network core
    - Mesh of interconnected routers/switches
    - Two forms of switched networks
        - Circuit switching – used in the legacy telephone networks

        - Packet switching – used in the Internet
- Circuit switching – end-end resource allocated to, reserved for 'call' between source + destination
    - Dedicated resources – no sharing + circuit segment idle if not used by call
    - Need to establish path with circuits + links before data can be sent over
    - FDM vs TDM
        - FDM – users is a dedicated frequency range of bandwidth
        - TDM – users are given the maximum bandwidth for fixed interval of time
    - Timing in circuit switching
        - Circuit establishment → data transfer → circuit teardown
        - Delays in forward path of circuit establishment – checks if a path exists from source to destination + checks if it has the capacity to provide the circuit
    - Advantage of circuit switching
        - Reliable for data transfer – guaranteed bandwidth once path is established
    - Disadvantages of circuit switching – not feasible
        - Inefficient
            - Computer communication tends to be very bursty
            - Dedicated circuit cannot be used or shared in periods of silence
            - Cannot adopt to network dynamics
        - Fixed data rate
            - Computers communicate at very diverse rate e.g. videos, web browsing
            - Fixed data rate is not useful
        - Connection state maintenance
            - Considerable overhead – requires per communication state to be maintained
            - Not scalable
- Packet switching
    - Packets – data sent as chunks of formatted bits
    - Packets consist of a 'header' and 'payload'
        - Header – instructions to network for packet handling
            - Internet addresses (source + dest.), age (TTL), checksum to protect header (sanity check – check if it got the correct package or not)
            - Utilized by core routers – to look up where packet is going
            - Utilized by destination – to checksum + source dest.
        - Payload – actual original data being carried
    - Switches 'forward' packets based on their header and its assigned routing table
        - Reference routing table – decide base of routing protocol the outgoing link
    - Timing in packet switching
        - Time to process the packet at switch – assume it's relatively negligible
            - Working in line speed (no more than a few milliseconds)
        - Cut through switching – start transmitting soon as it has processed the header
            - Not used in public internet – checksum calculated on both the header + payload
                - Cannot verify checksum unless payload is also there

            - Check if packet is in good health, to be able to use the header's fields with confidence
        - Store and forward switching – switch forwards a packet after it has received it entirely
- Each packet travel independently
    - No notion of packet belonging to a 'circuit'
    - Each packet of a whole chunk of data may take a different route
- No link resources are reserved in advance
    - Packet switching leverages statistical multiplexing
    - Statistical multiplexing – assumes that not all data flows burst/overload at the same time
        - Transient overload – If it does overload at same time (only capacity to transit one packet at a time)
            - Queue overload into a buffer – temporary absorbs other packet
        - Persistent overload – large load from several data flows at same time
            - Queue reaches capacity and buffer eventually drops packet
- Advantages of packet switching
    - Does not maintain any state – scalable
    - Great for bursty data – resource sharing, simpler, no call setup
    - Delay relies on state of router
    - Allows more users to use network, based on the statistics on how many are using it on the same time
- Disadvantage
    - Excessive congestion possible – packet delay + loss
        - Protocols needed for reliable data transfer, congestion
    - Adding header to the packet becomes the overhead
    - Bandwidth guarantees needed for audio/video apps – still unsolved problem
- Internet structure – network of network
    - End systems connect to Internet via access ISPs e.g. residential, company
        - ISPs interconnected – hosts send packet to each other
        - Network of networks – driven by economics + national policies
    - Connecting millions of access ISPs together
        - Not scalable – connecting each access ISP to each other directly, O(N^2) connections
        - Connect each access ISP to a global transit ISP – customer and provider ISPs have economic agreement
            - Global ISP has competitors which also be interconnected
                - Internet exchange point (IXP) – multiple global ISPs and other access nets can join at this point
                - Peering link – establish private links between ISPs
                - IXP over peering link – saving on infrastructure + reduce hops to other transit links
            - Regional networks – connect access net to ISPs
        - Center of network structure – small no. of well-connected large networks

        - Content provider networks e.g. Google, Microsoft – run private network, to bring services, content close to end users, often bypasses tier-I, regional ISPs
        - "Tier-1" commercial ISPs – national + international

1.4. Delay, loss, throughput in networks

- Key properties of links – pipe an alogy
    - Bandwidth – width of link, no. of bits sent per unit of time (bps)
    - Propagation delay – length of link, propagation time to travel along link (secs)
    - Bandwidth-delay product – volume of link, amount of data in flight (bps*s=bits)
- How do loss and delay occur
    - Packet queue in router buffers
        - Packet arrival rate to link (temporarily) exceeds output link
        - Packets queue, wait for turn (queue is finite)
            - Delay – packet being transmitted/packet queueing
            - Loss – arriving packet dropped if no free buffers
- Four source of packet delay
    - dnodal = dproc + dqueue + dtrans + dprop
    - Nodal processing (dproc)
        - Check bit error (checksum), determine output link, typically < msec
    - Queueing delay (dqueue)
        - Time waiting at output link for transmission
        - Depends on congestion level of router – hard to predict
    - Transmission delay (dtrans)
        - dtrans = L/R [L: packet length (bits), R: link bandwidth (bps)]
        - Delay incurred while pushing data onto wire
    - Propagation delay (dprop)
        - dprop = d/s [d: length of physical link, s: prop. speed (~2*10^8 m/s)]
        - Time spent travelling on the wire
- Queueing delay
    - aL bits arrive to queue [a: packet arrival rate (packets/s), L: packet length (bits)]
    - R bits leave the router [R: link bandwidth (bits/s)]
    - Traffic intensity – aL/R
        - When aL > R – queue will fill up + packet losses
        - Average queueing delay is dependant on pattern of packet arrival bursts
        - ~ 0 – small delays, ~1 – large delays, >1 – infinite average delay
- "Real" internet delays + routes
    - Traceroute program – provide delay measurement from source to router along end-end Internet path towards destination
        - Big increase in delay measurement – trans-oceanic link
        - No response – probe lost, router not replying
    - End-to-end delay = sum of all dnodal along the path
- Packet loss
    - Queue (buffer) preceding link in buffer h as finite capacity
    - Packet loss – packet arriving to full queue dropped
    - Lost packet may be retransmitted
- Throughput – rate at which bits transferred between sender/receiver
    - Instantaneous – rate at given point in time

    - Average – rate over longer period of time
    - Bottleneck link – link on end-end path that constrains end-end throughput

1.5. Protocol layers

- Network design principles
    - Keep core simple – move complexity to edges
    - Do layering – different module/layers for different functionality
- Layering + decomposition – remove complexity from core to the end systems
    - Simplistic decomposition of sending packets across
        - Consider sending packet along single wire
        - Stitch prior functionality together to cross country/globe
    - Tasks in networking – layering bottom up approach between client/server
        - Bits/packets on wire → deliver packets within local network → deliver across global → ensure packet get to dest.→ process data
        - Internet protocol stack
            - Application – supporting network apps e.g. HTTP, Skype
            - Transport – process-process data transfer e.g. TCP, UDP
            - Network – routing of datagrams from source to dest. e.g. IP
            - Link – data transfer between neighbouring network elements e.g. ethernet, Wifi
            - Physical – bits 'on the wire'
        - Layers depend on below, support above, independent of others
        - Multiple versions in layer
            - Interface differ somewhat
            - Components pick which lower level protocol to use
            - Only one IP layer – unifying protocol
        - Allows abstraction + lowest level has most packaging
        - No layering – each new app has to be re-implemented for every network technology + existing apps need to be upgraded for new tech
    - Advantages
        - Abstraction – only deal with what you want to achieve out of system, don't deal with other complexities
        - Allows intermediate layer – provide common abstractions/API for various network technologies
        - Security – each layer only looks at info specific to it
        - Troubleshooting – module specific problems identified quickly
    - Disadvantages
        - Layer N may duplicate lower level functionality e.g. error recovery
        - Information hiding may hurt performance e.g. packet loss due to corruption vs congestion
        - Header start to get really big e.g. TCP+IP+ethernet headers add to 54 bytes
        - Layer violations when the gains too great to resist e.g. TCP-over-wireless
        - Layer violations when network doesn't trust ends e.g. firewalls
- Distributing layers across network
    - Complexity – need to implement layers across multiple machines
        - Hosts – all layers must exit here
            - Bits arrive on wire (physical), must make it up to app (support other layers)

        - Routers – below + excl. transport layer (doesn't support reliable delivery)
            - Bits arrive on wire (physical), packets must be delivered to next-hop (datalink), participate in global delivery (network layer)
        - Switches – only has physical + link layers
    - Logical communication – layers interacts with peer's corresponding layer
    - Physical communication – host (app→phys) → router (phys→net→phys) → host (phys→app)
    - Encapsulation – original message expands due to appending of extra control info to header (and trailers) as its going through the internet protocol stack
        - Message (app) → segment → datagram → frame (datalink)
        - On reception – takes off + checks the corresponding header to that layer

1.6. Networks under attack: security

1.7. History

APPLICATION LAYER

2.1. Principles of network applications

- Goals of application layer
    - Conceptual, implementation aspects of network application protocols
        - Transport-layer service models (handshaking)
        - Client-server paradigm
        - Peer-to-peer paradigm
    - Learn about protocols by examining popular application-level protocols
        - E.g. HTTP, SMTP/POP3/IMAP, DNS
    - Creating network applications e.g. socket API
- Creating a network app
    - Write programs that run on (different) end systems + communicates over network
    - Not for network core-device – don't run user apps
    - Apps on end systems allows for rapid app development, propagation
- Interprocess Communication (IPC) – processes talk to each other through IPC
    - 1 machine → shared memory, >1 machine → abstraction (message passing)
- Sockets – door between app and transport layer
    - Process sends/receives message to/from its socket
    - Treats transport layer + below as blackbox – for delivering message to socket at receiving process
    - Application has a few options, OS handle the details
- Addressing processes
    - To receive message, process must have identifier
        - Identifier – has IP address + port numbers associated with process
            - Host device has unique 32-bit IP address – runs many processes
            - Software port number identifies the process on host
- Client-server architecture
    - Server
        - Always-on host – permanent IP + static port conventions
        - Exports well-defined request/responsive interface
        - Long-lived process that waits for requests + carries out received request
        - May communicate with other servers to respond

- Clients
  - May be intermittently connected – possible dynamic IP addresses
  - Short-lived process that make requests
  - 'User-side' of application + initiates communication
  - Do not communicate directly with each other
- P2P architecture
  - For distributed systems – file sharing (BitTorrent), games, video distribution + chat
  - No always-on server – no permanent rendezvous involved
  - Arbitrary end systems (peers) directly communicate
  - Symmetric responsibility (unlike client/server)
  - Advantages
    - Peers can both request from + provide services to other peers
      - Self scalability – new peers bring new service capacity + demands
    - Speed – parallelism, less contention
    - Reliability – redundancy, fault tolerance
    - Geographic distribution
  - Disadvantages of decentralized control
    - No distribution control
    - State uncertainty – no shared memory or clock
    - Action uncertainty – mutually conflicting decisions
    - Distributed algorithms are complex
- App-layer protocol defines
  - Types of messages exchanged e.g. request, response
  - Message syntax – fields + formatting in messages
  - Message semantics – meaning of info in fields
  - Rules for when + how processes send + respond to messages
  - Classifications
    - Open protocols – defined in RFC + allows for interoperability e.g. HTTP
    - Proprietary protocols e.g. Skype
- Possible requirements for transport services for apps
  - Data integrity – 100% data transfer (file transfer, web transact.)/loss tolerant (audio)
  - Timing – need low delay to be 'effective' (Internet telephony, interactive games)
  - Throughput – makes use of whatever (elastic)/min. amount required (multimedia)
  - Security – encryption, data integrity
- Internet transport protocols services
  - If transport layer does not provide some services, the app layer must handle it
  - TCP – only provides reliability (message sent in same order, without error)
    - Flow control = sender won't overwhelm receiver
    - Congestion control – throttle sender when network overloaded
    - Connection-oriented – setup required between client + server processes
  - UDP – does not provide reliable transport
    - Can be applied to streaming multimedia or internet telephony

2.2. Web and HTTP

- Review
  - Web page – consists of base HTML-file which includes several referenced objects
  - Each object is addressable by a URL (Uniform Resource Locator)

        - protocol://host-name[:port]/directory-path/resource
            - hostname – DNS name, IP address
            - port – defaults to protocol's standard port e.g. http: 80, https: 443
            - directory path – reflects file system
- HTTP overview
    - HTTP – hypertext transfer protocol
        - Web's app layer protocol
        - Client/server model
            - Client – browser that request, receives + displays Web objects
                - Initially requests index.html file + identifies referenced objects
            - Server – sends objects in response to requests
        - Uses TCP
            - Client initiate TCP connection (creates socket) to server, port 80
            - Server accepts connection from client
            - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
            - TCP connection closed
        - HTTP is 'stateless' – can refresh web browser, re-request data
            - Server maintains no info about past client requests
            - Protocols that maintain state are complex – must reconcile state if server/client crashes, in order for it to be consistent
    - HTTP request message
        - Two types of HTTP messages – request, response
        - Request message – request (GET, POST, HEAD commands) + header lines
            - Host – address of the server, not client
            - Carriage return, line feed at start of line indicate end of header lines
            - GET/HEAD command – no body (body for POST)
        - Response message – status (protocol, status code, status phrase) + header lines
            - First Date – date at which request is fulfilled
            - Last modified time – last time that resource was modified
                - If resource not modified and it is cached – can reuse resource, and does not have to re-download
        - HTTP response status codes – appears in 1st line in server/client response
            - 200 OK – request succeeded
            - 301 Moved Permanently – requested object moved, new location specified
            - 400 Bad Request – request msg not understood by server
            - 404 Not Found – requested document not found on this server
        - HTTP is all text
            - Simple protocol – easy to delineate messages (\r\n), human-readable, no issues about encoding/formatting data, variable length data
            - Not most efficient – many protocols use binary fields e.g. sending string "12345678" vs int (size), headers can come in any order, requires string parsing
        - Request Method type ("verbs")

        - HTTP/1.0
            - GET – request page
            - POST – uploads user response to a form (only text)
            - HEAD – ask server to leave requested object out of response
        - HTTP/1.1
            - GET, POST, HEAD
            - PUT – uploads file in entity body to path in specified URL field (came be image)
            - DELETE – deletes file specified in URL field
            - TRACE, OPTIONS, CONNET, PATCH – persistent connections
                - Last 3 require having gone through the authorization process
        - Uploading form input
            - POST method (body of HTTP msg) – web page often includes form input which is uploaded to server through entity body
            - Get (in-URL) method (part of the URL itself) – uses GET method, input is uploaded in URL field of request line
- User-server state – cookies
    - Many web sites use cookies – four components
        - Cookie header line of HTTP response msg (set-cookie)
            - Initial HTTP request – site create unique ID, create entry in backend DB for ID
        - Cookie header line in next HTTP request message
            - Cookie-specific action – readily identify client from backend
        - Cookie file kept on user's host, managed by user's browser
        - Back-end database at Web site
    - Disadvantages – cookies permit site to learn more about you
        - 3rd part cookies e.g. ad networks, follow on many sites
            - Not a cookie from the original site, but is a cookie for one of the requested objects on the HTML page from a different server e.g. banner
- Performance of HTTP
    - Page Load Time (PLT) as metric – from click until user sees page
        - Key measure of web performance
        - Depends on factors e.g. page content/structure, protocols involved, network bandwidth + RTT (round-trip delay time)
    - Performance goals
        - User – fast downloads, high availability
        - Content provider – happy users, cost-effective infrastructure
        - Network (secondary) – avoid overload
        - Achieved through
            - Improving HTTP to achieve faster downloads
            - Caching + replication (relevant to all goals)
            - Exploit economies of scale for infrastructure (webhosting, CDNs, datacenters)
        - Improving PLT
            - Reduce content size for transfer (smaller imgs, compress)

        - Change HTTP to make better use of available bandwidth
            - Persistent connections + pipelining
        - Change HTTP to avoid repeated transfers of same content
            - Caching + web-proxies
        - Move content closer to content e.g. CDNs
    - HTTP Performance
        - Retrieve objects for HTML file's embedded images one at a time
            - New TCP connection per (small) object)
        - Non-persistent HTTP – at most one object sent over TCP connection, which is then closed
            - Downloading multiple objects require multiple connections
- Non-persistent HTTP – response time
    - RTT – time for a small packet to travel from client to server + back
    - Non-persistent HTTP response time = 2 RTT + file transmission time
        - One RTT to initiate TCP connection
        - One RTT for HTTP request + first few bytes of HTTP response to return
        - File transmission time
    - HTTP/1.0 – Non-persistent HTTP
        - Non-persistent – one TCP connection to fetch one web resource
        - Fairly poor PLT
        - 2 scenarios – same sequential mechanism, no parallel requests
            - Multiple TCP connections setups to same server
            - Sequential request/responses even when resources are located on different servers
        - Multiple TCP slow-start phases – loss of momentum from clearing out previous information
    - Concurrent requests + responses
        - Use multiple connection in parallel – beneficial if resources on multiple servers
        - Does not necessarily maintain order of responses
        - Disadvantages – increases the load on the server (if many embedded objects from same server) and/or access link
- HTTP/1.1 – Persistent HTTP
    - Server leaves TCP connection after sending responses
    - Subsequent HTTP messages between same client/server are sent over the same TCP connection
    - Allow TCP to learn more accurate RTT estimate
    - Allow TCP congestion window to increase i.e. leverage previously discovered bandwidth
    - Allows pipelining
        - Without pipelining – still sequential, client issues new request only when previous response has been received
            - One RTT for each referenced object
        - With pipelining (default in HTTP/1.1) – back-to-back requests
            - Client sends request as soon as it encounters a referenced object

            - As little as one RTT for all objects
- Improving HTTP performance of same content
    - Caching – exploits locality of reference + not all objects of webpage is modified frequently
        - Efficiency – works well to a limit, large overlap in content, but many unique requests
    - Web caches – proxy servers
        - Goal – satisfy client request without involving origin server
        - Process
            - User sets browser – web accesses via cache
            - Browser sends all HTTP requests to cache
                - Object in cache – cache returns object
                - Else requests object from origin server, then returns object to client
        - Cache acts as both client (to origin server) + server (for original requesting client)
        - Typically cache is installed by ISP e.g. uni, company, residential ISP
        - Advantages
            - Reduce response time for client request
            - Reduce traffic on an institution's access link
            - Internet dense with caches – enable 'poor' content providers to effectively deliver content
    - Likelihood of cache hit – the more popular a content is, the more views (temporal locality)
- Conditional GET
    - Goal – don't send objects if cache has up-to-date cached version
        - No object transmission delay
        - Lower link utilization
    - Cache – specify date of cached copy in HTTP request (if-modified-since)
    - Server – response contains no object if copy is up-to-date (HTTP/1.0 304 – Not Modified)
        - Etag header – matches with cache check request's if-none-match
    - Uncacheable content – expire header would be before server time
- Replication
    - Replicate popular Web site across many machines
        - Spreads load on servers
        - Places content closer to clients
        - Helps when content isn't cacheable
    - Problem
        - Want to direct client to particular replica
            - Balance load across server replicas
            - Pair clients with nearby servers
        - Expensive
    - Common solution – DNS returns different addresses based on client's geo location, server load, etc.
- Content Distribution Network (CDN) – caching + replication as a service
    - Integrate forward + reverse caching functionality

      - Large-scale distributed storage infrastructure administered by one entity
      - Combination of (pull) caching + (push) replication
        - Pull – direct results of clients' requests
        - Push – expectation of high access rates
  - HTTPS – HTTP over a connection encrypted by Transport Layer Security (TLS)
    - HTTP is insecure – basic authentication has password sent using base64 encoding (readily converted to plaintext)
    - HTTPS provides authentication + bidirectional encryption
  - HTTP/2 – speedier + better content structure
    - Servers can push content and thus reduce overhead of an additional request cycle (client doesn't have to request for the subsequent objects after index)
    - Data compression of HTTP headers – some headers can be long/repetitive
    - Fully multiplexed – requests + responses are sliced in smaller chunks (frames), frames tagged with ID connecting data to request/response
    - Prioritization of order in which objects should be sent

2.3. Electronic mail – SMTP, POP3, IMAP

- Electronic mail
  - 3 major components: user agents, mail servers, simple mail transfer protocol (SMTP)
  - User agent – mail reader e.g. Outlook, Thunderbird, iPhone mail client
    - Composing, editing, reading mail msgs
    - Outgoing, incoming msgs stored on server
  - Mail servers – mailbox containing incoming msgs for user
    - Message queue of outgoing (to be sent) mail msgs
- SMTP protocol – direct connection between mail servers to send email msgs
  - Client – sending mail server, 'Server' – receiving mail server
    - Mail servers required – user agents not always online + cannot send/receive mails when offline
  - Uses TCP to reliably transfer email msg from client to server, port 25
    - Uses persistent connections
  - Direct transfer – sending server to receiving server (no intermediate)
  - 3 phases of transfer – handshaking (greeting), transfer of msgs, closure
    - Not involved at the mail retrieval at mailbox, only sending
  - Command/response interaction (like HTTP, FTP)
    - Commands – ASCII text
    - Response – status code + phrase
  - Msgs must be 7-bit ASCII
    - SMTP server uses CRLF.CRLF to determine end of msg
    - Multimedia is also encoded into 7-bit ASCII
- SMTP vs HTTP
  - HTTP – pull, SMTP – push
  - Both have ASCII command/response interaction, status codes
  - HTTP – each object encapsulated in own msg
  - SMTP – multiple objects sent in multipart msg
- Phishing
  - Detect fake email – examine long headers/raw source
  - Spear phishing – directed at specific individuals/companies (most popular)

        - Attacker may gather personal info about target to increase success rate
    - Clone phishing – legitimate + previously delivered email containing an altered, malicious attachment, sent by spoof email address
- Mail message format
    - RCF 5322 (822, 2822) – standard for text msg format (Internet Msg Format, IMF)
        - Header lines e.g. (to, from, subject)
            - Different from SMTP MAIL FROM, RCPT TO: commands
        - Body: the “message” – ASCII chars only
- Mail access protocols
    - SMTP – delivery/storage to receiver’s server
    - Mail access protocol – retrieval from server
        - POP – Post Office Protocol: authorization, download
        - IMAP – Internet Mail Access Protocol: more features, including manipulation of stored msgs on server
        - HTTP(S): Gmail, Yahoo

2.4. DNS

- Domain Name System (DNS) – service that transforms website names to IP address, and vice versa
    - Distributed database – implemented in hierarchy of many name servers
    - Application-layer protocol – hosts, name servers communicate to resolve names (address/name translation)
        - Core Internet function – implemented as application-layer protocol, complexity at network’s ‘edge’
- History
    - Initially all host address mappings were in a hosts.txt file (maintained by SRI)
        - New versions of hosts.txt periodically, admin could pick names
    - As Internet grew, system broke down
        - SRI couldn’t handle the load, names were not unique, hosts had inaccurate copies of hosts.txt
    - DNS invented to fix this
- Services, structure
    - DNS services – hostname to IP address translation
    - Host aliasing – canonical, alias names
    - Mail server aliasing
    - Load distribution
        - Replicated Web servers – many IP addresses correspond to one name
        - Content Distribution Networks – use IP address of requesting host to find best suitable server
    - Centralized DNS not scalable
        - Single point failure – if intermediate directories are off, cannot PING
        - Traffic volume at single server
        - Distant centralized database – mileage would vary on distance
        - Maintenance of the record – new machines need to be constantly added on, and advertised to the whole internet
- Goals

    - Uniqueness – no naming conflicts
    - Scalable – many names, frequent updates
    - Distributed, autonomous administration
        - Ability to update my own (machines') names
        - Don't have to track everybody's updates
    - Highly available + fast lookups
- Three intertwined hierarchies
    - Hierarchical namespace – as opposed to original flat namespace
        - Top level domains are at top, topmost is root (never referred to)
        - Domains are sub-trees e.g. eecs.berkley.edu
        - Name is leaf-to-root path
        - Depth of tree is arbitrary (limit 128)
        - Name collisions trivially avoided – can reuse names in different domains, each domain is responsible
    - Hierarchically administered – as opposed to centralized
        - Zone – corresponds to an administrative authority that is responsible for that portion of the hierarchy
        - E.g. UCB controls names *.berkeley.edu and *.sims.berkeley.edu
    - (Distributed) hierarchy of servers – as opposed to centralized storage
        - Top of hierarchy – root servers, location hardwired into other servers
            - There are 13 root servers around the world
            - Replicated via any-casting
        - Next level – top-level domain (TLD) servers, professionally managed, can be separated into generic (.com, .edu) + countries (.au, .us)
        - Bottom level – authoritative DNS servers
            - Store the name-to-address mapping
            - Stores 'resource records' for all DNS names in domain which it has authority of
            - Maintained by corresponding administrative authority
        - Each server stores a subset of the total DNS DB
        - Each server needs to know other servers that are responsible for the other portions of the hierarchy
            - All know root, root server knows all top-level domains
- DNS name resolution example (no caching) – iterative process
    - Iterated query – contacted server replies with name of next server to contact
    1. Host make DNS query, query sent to local DNS server
    2. Local NS contacts Root NS, Root NS knows the TLD NS .edu + sends IP address for it back
    3. Local NS contacts TLD NS, TLD NS knows authoritative NS Washington + sends IP address
    4. Local NS contacts authoritative NS, robot is within its zone + sends IP address for it back
    5. IP address for mapping, then caches it
    - Recursive query – puts burden of name resolution on contacted name server
- Top-level domain (TLD) servers
    - Responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains
    - Network Solutions maintains servers for .com TLD, Educause for .edu TLD
- Authoritative DNS servers

- Organization’s own DNS server(s), providing authoritative hostname to IP mappings for organization’s named hosts
    - Can be maintained by organization or service provider
- Local DNS name server
    - Does not strictly belong to hierarchy
    - Each ISP (residential ISP, company, uni) has one – called ‘default name server’
    - Host configured with local DNS server address or learn server via a host configuration protocol
    - Client application
        - Obtain DNS name e.g. from URL
        - Do gethostbyname() to trigger DNS request to its local DNS server
    - When host makes DNS query, query is sent to its local DNS server
        - Has local cache of recent name-to-address translation pairs (may be out of date)
        - Acts as proxy, forwards query into hierarchy
- DNS name resolution
    - Iterated query – contacted server replies with name of next server to contact
    - Recursive query – puts burden of name resolution on contacted name server
- DNS – caching, updating records
    - Once (any) name server learns mapping, it caches mapping
        - Cache entries timeout (disappear) after some time (TTL)
        - TLF servers typically cached in local name servers (root NS not often visited)
    - Subsequent requests need not burden DNS
    - Cached entries may be out-of-date (best effort name-to-address translation)
        - If name host changes IP address, may not be known Internet-wide until all TTLs expire
- Type of DNS records
    - DNS – distributed db storing resource records (RR)
        - RR format: (name, value, type, ttl)
    - Type=A is for IPv4 address (Type=AAAA returns IPv6 address)
        - Name is hostname
        - Value is IP address
    - Type=NS
        - Name is domain e.g. foo.com
        - Value is hostname of authoritative name server for this domain
    - Type=CNAME
        - Name is alias name for some ‘canonical’ name
            - www.ibm.com is really servereast.backup2.ibm.com
        - Value is canonical name
    - Type=MX
        - Value is name of mailserver associated with name
- DNS protocol, messages
    - Query and reply msgs, both with same msg format
    - Msg header
        - Identification – 16 bit no. for query, reply to query uses same no.
        - Flags – query or reply, recursion desired, recursion available, reply is authoritative

    - Question section
    - Answers section – A type records of accessing server through other replicated servers (also shows TTL in seconds)
    - Authority section – NS responsible for managing records under the domain
    - Additional section – IP addresses of all of the NS
- Inserting records into DNS
    - Register name networkutopia.com at DNS registrar e.g. Network Solutions
        - Provide names, IP address of authoritative NS (primary + secondary)
        - Registrar inserts two RR into .com TLD server:
            - (networkutopia.com, dns1.networutopia.com (auth NS), NS)
            - (dns1.networkutopia.com, 212.212.212.1, A)
        - Any sub-domain, you have to insert record into your auth NS
            - Create authoritative server type A record for www.networkutopia.com, type MX record for networkutopia.com
- Reliability
    - DNS servers are replicated (primary/secondary)
        - Name service available if at least one replica is up
        - Queries can be loaded-balanced between replicas
    - Usually, UDP used for queries
        - DNS are very short msgs, but TCP undergoes more delays than UDP
        - Needed for reliability – must implement this on top of UDP
        - Spec support TCP too, but not always implemented
    - Try alternative servers on timeout – exponential backoff when retrying same server
    - Same identifier for all queries – don't care which server responds
- DNS provides Indirection – flexibility within domain
    - Addresses can change underneath – update DNS registrar for the TLD
        - Cached copy of old mapping needs to undergo TTL at all NS
        - After TTL, will undergo usual name resolution query
    - Name could map to multiple IP addresses
        - Enables load-balancing + reducing latency by picking nearby servers
    - Multiple names for the same address
        - E.g. many services (mail, www, ftp) on same machine
        - E.g. aliases like www.cnn.com and cnn.com
- Reverse DNS – IP address → domain name
    - Special PTR record type to store reverse DNS entries
    - Where is reverse DNS used?
        - Troubleshooting tools e.g. traceroute + ping
        - 'Received' trace header field in MTP email
        - SMTP servers for validating IP address of originating servers
        - Internet forums tracking users
        - System logging or monitoring tools
        - Used in load balancing servers/content distribution to determine location o requester
- Trusting your DNS server
    - Censorship – alters the mapping from original content
    - Logging – keep track of IP address, websites, visited, geolocation data
- Attacking DNS

- DDoS attacks (Directed denial of service)
  - Bombard root servers with traffic
    - Not successful to date – few in no., well protected, traffic filtering
    - Local DNS servers cache IPs of TTLD servers, allowing root server to be bypassed
  - Bombard TLD servers – potentially more dangerous
- Redirect attacks
  - Man-in-middle – intercept queries
  - DNS poisoning – send bogus replies to DNS server, which caches
    - Solution – do not allow DNS servers to cache IP address mappings unless they are from authoritative NS
- Exploit DNS for DDoS – send queries with spoofed source address (target IP), requires amplification

2.5. P2P applications

- Pure P2P architecture e.g. file distribution, streaming, VoIP
  - No always-on server
  - Arbitrary end systems directly communicate
  - Peers are intermittently connected and change IP addreses
- File distribution – client-server vs P2P
  - How much time to distribute file (size F) from one server to N peers
    - Peer upload/download capacity is limited resource
  - Client-server – becomes longer with increase in clients
    - Server transmission – must send (upload) N file copies
      - Time to send one copy – F/us
      - Time to send N copies – NF/us
    - Client – each client must download file copy
      - Dmin = min client download rate
      - Client download time – F/dmin

time to distribute F to N clients using client-server approach $D_{c\text{-}s} \geq max\{NF/u_{s,},F/d_{min}\}$

increases linearly in N

  - P2P – as N clients increase, service capacity increases
    - Server transmission – must upload at least one copy
      - Time to send one copy – F/us
    - Client – each client must download file copy
      - Client download time – F/dmin
    - Clients – as aggregate must download NF bits
      - Max upload rate (limiting max download rate is) us + Sum(ui)
      - Client aggregate – NF/max upload rate

time to distribute F to N clients using P2P approach $D_{P2P} \geq max\{F/u_{s,},F/d_{min,},NF/(u_s + \sum_{i=1}^{N} u_i)\}$

increases linearly in *N* …
… but so does this, as each peer brings service capacity

- P2P file distribution – BitTorrent
  - File divided into 256 kB chunks
  - Peers in torrent end/receive file chunks
  - Tracker – tracks peers participating in torrent
  - Torrent – group of peers exchanging chunks of a file
  - Peer joining torrent
    - Obtain list of peers from trackers, connects to subset of peers (neighbours)
    - Has no chunks, but will accumulate them over time from other peers

        - While downloading, peer uploads chunks to other peers
        - Peer may change peers with whom it exchanges chunks
            - Churn – peers may come and go
        - Once peer has entire file, it may leave or remain in torrent
- .torrent files
    - Contains address of trackers for the file
    - Contain a list of file chunks and their cryptographic hashes
        - Ensures chunks are not modified
- BitTorrent – requesting, sending file chunks
    - Requesting chunks
        - Different peers have different subsets of file chunks
        - Periodically, ask each peer for list of chunks they have
        - Requests missing chunks from peers, rarest first
            - Rare peer may go offline, can start uploading rare chunk yourself
    - Sending chunks – tit-for-tat
        - Send chunks to 4 neighbours currently sending her chunks at highest rate
            - Other peers are choked (do not receive chunks)
            - Re-evaluate top 4 every 10 secs
        - Every 30 secs – randomly select another peer, starts sending chunks
            - 'Optimistically unchoke' this peer
            - Newly chosen peer may join top 4
        - Higher upload rate – find better trading partners get file faster
        - Free-riding – have to wait for 'optimistic unchoking' for every chunk they require, if the client themselves refuses to upload data to peers
- Getting rid of the server/tracker – making peers act as trackers as well
    - Harder to shut down torrent, by not making a centralized tracker, but also giving the possibility of having the peers as trackers
    - Distribute the tracker info using Distributed Hash Table (DHT)
    - DHT is a lookup structure
        - Maps keys to arbitrary value, works like a hash table
- Distributed Hash Table (DHT) – distributed P2P DB
    - DB has (key, value) pairs e.g. (file name, BT tracker peer(s))
        - When files are added in, the file name is hashed, and the tracker no. that's closest to the hashed no. is assigned as the tracker for that file
        - Other peers with chunks of this file will be registered to this tracker
    - Distribute the (key, value) pairs over the (millions of peers)
    - A peer queries DHT with key – DHT returns value that match the key
    - Peer can also insert (key, value) pairs
- Assigning keys
    - Convert each key to an integer
    - Assign integer to each peer
    - Put (key, value) pair in the peer that is closest to the key
        - Common convention – closest is the immediate successor to the key e.g. key = 13, successor peer = 14
- Circular DHT
    - Each peer only aware of immediate successor + predecessor
    - Can only query successor, before getting response for tracker

        - Wort case all peers probed, N messages, on average N/2
    - Mesh overlay – each peer tracks all other N-1 peers, only 1 message sent per query
- Circular DHT with shortcuts
    - Each peer keeps track of IP addresses of predecessor, successor, short cuts
    - Possible to deign shortcut so O(log N) neighbours, O(log N) msgs in query
- Peer churn
    - Peers may come + go (churn)
    - Each peer knows address of its two successors
    - Each peer periodically pings its two successors to check aliveness
    - If immediate successor abruptly leaves, choose next successor as new immediate successor
        - If not abruptly, leaving peer transfers its records to its successor

2.6. Video streaming and content distribution networks (CDNs)

- Context
    - Video traffic – major consumer of Internet bandwidth
    - Challenge is scale – how to reach ~1B users
        - Single mega-video server won't work
    - Challenge – heterogeneity
        - Different users have different viewing capabilities
    - Solution – distributed, application-level infrastructure
    - Multimedia – video
        - Video – sequence of images displayed at constant rate
        - Digital image – array of pixels, each pixel represented by pits
        - Coding – use redundancy within + between image to degrease no. of bit used to encode image
            - Spatial (within image) – instead of sending N values of same colour, send only 2 values (colour values + no. of repeated values, N)
            - Temporal (from one image to next) – instead of sending complete frame at i+1, send only differences from frame i
        - Constant bit rate (CBR) – video encoding rate fixed
        - Variable bit rate (VBR) – video encoding rate changes as amount of spatial, temporal coding changes
- Streaming stored video
    - Simple scenario – video server (stored video) → Internet → client
        - Different download capabilities, different geographical locations, differing bandwidth
- Streaming multimedia – DASH protocol
    - DASH – Dynamic, Adaptive Streaming over HTTP
    - Server – divides video file into multiple chunks
        - Each chunk stored, encoded at different rates
        - Manifest file – provides URLs for different chunks
    - Client – periodically measures server-to-client bandwidth
        - Consulting manifest, request one chunk at a time
            - Chooses max coding rate sustainable given current bandwidth

            - Can choose different coding rates at different points in time (depending on available bandwidth at time)
    - 'Intelligence' at client – client determines
        - When to request chunk (so that buffer starvation, or overflow does not 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
- Content distribution networks
    - Service caching and replication – amortise cost of infrastructure
    - Goal – bring content close to user
    - Large-scale distributed storage infrastructure (usually) administer by one entity
    - Combination of (pull) caching + (push) replication
    - Store/serve multiple copies of videos at multiple geographically distributed sites
        - Enter deep – push CDN servers deep into many access networks of ISPs
            - No excess, least network delay involved – content is being directly delivered from within the same ISP
        - Bring home – smaller number (10s) of larger clusters in POPs near (but not within) access networks
            - Deploy CDN node at IXP (Internet Exchange Point)
- Example with Netflix
    - CDN – stores copies of content at CDN nodes e.g. Netflix stores copies of MadMen
        - Manifest file sent from Netflix server for the CDN nodes
    - Subscriber requests content from CDN
        - directly to nearby copy, retrieves content
        - may choose different copy if network path congested
    - "Over the top" – overlay networks out from existing nodes, disregard the nodes that are supporting this existing network
- CDN content access
    - Client request video http://netcinema.com/6Y7B23V
        - Video stored in CDN at http://KingCDN.com/NetC6y&B23V
    1. Bob gets URL for video http://netcinema.com/6Y7B23V from netcinema.com web page
    2. Resolve http://netcinema.com/6Y7B23V via Bob's local DN
    3. Netcinema's DNS turns URL http://KingCDN.com/NetC6y&B23V
    4. Resolve http://KingCDN.com/NetC6y&B23V via KingCDN's authoritative DNS, which returns IP address of KingCDN server with video
    5. Request video from KINGCDN server, streamed via HTTP

2.7. Socket programming with UDP and TCP

TRANSPORT LAYER

3.1. Transport-layer services

- Transport layer context
    - Current perspective – application is boss, network layer is ours to command

    - Network layer – finds paths through network, routing from one end host to another
        - Best effort delivery – no reliable transfer, guarantee paths
        - Consider layer as API with one function sendtohost(data, host)
- Transport services and protocols
    - Provide logical communication between app processes running on different hosts
        - End-to-end point protocol – only involved at the end points
    - Transport protocols run in end system
        - Send side – breaks app msgs into segments, passes to network layer
        - Receive side – reassembles segments into msgs, passes to app layer
        - Exports services to application that network layer does not provide
- Why a transport layer
    - Communication between processes at hosts – makes possible the identification of individual applications that is running on the end systems

3.2. Multiplexing and demultiplexing

- Definitions
    - Multiplexing at ender – handle data from multiple socket, add transport header
        - Attaching identifier to process and which processes it is communicating with
    - Demultiplexing at receiver – use header info to deliver received segments to correct socket
        - Also swaps the source and destination values in header in order to respond back
- Connectionless demultiplexing (UDP)
    - Sending process
        - Creating socket for sending has host-local port no.
        - Datagram for UDP socket – must have dest IP address + dest port no.
    - When host receives UDP segment
        - Checks dest port no. in segment
        - Directs UDP segment to socket with that port no.
        - IP datagrams with same dest port no., but different source IP addresses and/or source port no.s will be directed to same socket at dest
- Connection-oriented demux (TCP)
    - Each connection is identified separately – need to identify a buffer, packet sequence for each client in TCP, maintenance of state on server side
    - TCP socket identified by 4-tuple – IP addresses + port no.s for both source + dest.
        - Source addresses + port no. required since multiple sockets can be created at the same server port no.
    - Demux – receiver uses all 4 values to direct segment to appropriate socket
    - Server host may support many simultaneous TCP sockets
        - Each socket identified by own 4-tuple
    - Web servers have different sockets for each connecting client
        - Non-persistent HTTP will have different socket for each request
- Revisiting TCP sockets
    1. At client process, client socket sends across TCP handshake to the server process' welcoming socket at port X
    2. As soon as TCP handshake is over, server redirects the client to a separate connection socket for the client at the same port X which is dedicated for that specific client

- Scanning ports
    - Servers wait at open ports for client requests
    - Hackers often perform port scans to determine open, closed + unreachable port on candidate victims (several well-known ports)
    - Hackers can exploit known flaws with thee known apps

3.3. Connectionless transport – UDP

- User Datagram Protocol (UDP)
    - Cannot handle with reliable transfer service, only multiplexing/demux
    - Connectionless – no handshaking, each UDP segment handled independently of each other
- UDP segment header (8 bytes)
    - Header is 32 bits wide – source + dest port no. (both 16 bits), length, checksum
- Purpose of UDP
    - No connection establishment – can add delay
    - Simple – no connection state at sender/receiver
    - Small header size
    - No congestion control – UDP can blast as fast as desired
- UDP checksum
    - Goal – detect 'errors' e.g. flipped bits, in transmitted segment
        - Router memory errors during process of queueing + buffer, driver bugs, electromagnetic interference over wireless
    - Sender
        - Treat segment contents, including header fields, as sequence of 16-bit integers
        - Checksum – addition (one's complement sum) of segment contents
        - Sender puts checksum value into UDP checksum field
    - Receiver
        - Add all received together as 16-bit integers
        - Add that to check sum
        - If result is not 1111 1111 1111 1111, error has occurred
- Further on checksum
    - Checksum uses the IP pseudo-header, UDP header, and UDP payload
    - IP pseudo-header itself contains the checksum
        - First put all 0s into checksum, perform checksum, replace checksum
        - Violation of layers – peeking into the IP header, extracting values from it, and calculates
    - UDP receiver would discard the data packet, if they checksum does not pass the sanity checks
- UDP applications
    - Latency sensitive/time critical e.g. voice/video chat, gaming, quick request/response (DNS/DHCP), routing updates (RIP), network management (SNMP)
    - Error correction unnecessary (periodic msgs)

3.4. Principles of reliable data transfer

- Reliable transport

- Packets can be corrupted (bit errors), lost (dropped from queue), delayed (queueing), reordered (congestion on some paths), duplicated (retransmission from receiver)
- Important in application, transport, link layers
  - Require no bits are corrupted, lost or arrive out-of-order at the receiver
- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt) – all reliability has to be encoded at transport layer

- Getting started
  - Incrementally develop sender, receiver sides of rdt protocol
  - Consider only unidirectional data transfer – control info will flow on both directions
  - Presume channel will not re-order packets
- Rdt 1.0: reliable transfer over a reliable channel – underlying channel perfectly reliable → transport layer does nothing
- Rdt2.0: channel with bit errors – underlying channel may flip bits in packet
  - Error detection – checksum to detect bit errors
  - Recovering from error – feedback to sender with control msgs for retransmit
    - Acknowledgements (ACKs) – receiver tells sender that pkt received OK
    - Negative acknowledgements (NAKs) – receiver tells sender that pkt errored
      - Sender retransmits pkt on receipt of NAK
  - Flaw – possibility of ACK/NAK corrupted, sender does not know what happened at receiver
- Rdt 2.1: accounting for ACK/NAK corruption
  - Sequence no. added to pkt – sender has twice many states
    - Must remember whether expected pkt should have sequence no. 0/1
  - Will retransmit for ACK/NAK corruption
  - Stop and wait protocol – sender sends one pkt, then waits for receiver response
    - Duplication detection at receiver
    - Sender retransmits current pkt if ACK/NAK corrupted
    - Sender adds sequence no. (0 or 1) to each pkt
    - Receiver discards (doesn't deliver) duplicate pkt with same sequence no.
- Rdt2.2.: NAK-free protocol
  - Receiver must explicitly includes sequence no. of pkt being ACKed
  - Duplicate ACK at sender results in retransmit current pkt
  - Duplicate detection at both sender + receiver
- Rdt3.0: channels with errors and loss
  - Timeout – sender waits 'reasonable' amount of time for ACK (requires countdown timer)
    - Retransmits if no ACK receiver in time
    - If pkt (or ACK) just delayer (not lost)
      - Retransmission will be duplicate – handled by sequence no.
      - Receiver must specify sequence no. of pkt being ACKed
  - Stop and wait operation – poor performance, limits use of resources
    - Utilization of sender (Us)- fraction of time sender busy sending
    - Us = (L/R) / (RTT + L/R)
- Pipelined protocols (sliding window)
  - Pipelining – sender allows multiple, 'in-flight', yet-to-be-acknowledged pkts
    - Range of sequence no. must be increased

        - Buffering at sender and/or (optionally) receiver
    - Two generic forms – go-Back-N, selective repeat
    - Increased utilization – can transmit back-to-back all at once
- Go-Back-N – sliding window protocol
    - Sender can have up to N unacked packets in pipeline
        - K-bit sequence no. in pkt header (2^k >= N)
        - 'window' of up to N, consec. unacked pkts
        - 'Window' slides to oldest in-flight pkt (send_base)
        - Every time 'window' slides up, nextseqnum is then acked
    - Sender has a single timer for oldest unacked packet, when timer expires, retransmit all unacked packets
        - Timeout(n) – retransmit pkt n + all higher sequence no. pkts in window
    - No buffer available at receiver – out of order packets are discarded
        - For packets 0-9, if pkt 0 is lost, 1-9 will be dropped at receiver
        - Sender timeout – retransmit pkts 0-9 again
    - Receiver only sends cumulative ack, doesn't ack new pkt if there's gap/loss
        - Receiver acks the last correctly received pkt, before loss
        - If acks from 0-8 is dropped, and ack 9 came back, the window for sender slide to pkt 9
- Selective repeat
    - Sender can have up to N unacked pkts in pipeline
    - Sender maintains timer for each unacked pkt, when timer expires, retransmit only that unacked pkt
    - Receiver has buffer – can accept out of order pkts
        - Buffers out of order pkts for eventual in-order delivery
    - Receiver sends individual ack for each pkt
    - Dilemma – if all pkt acks are lost from receiver window to sender, will accept the timed out and retransmitted first pkt (duplication + out of order)
        - Solution – sequence no. should at least be double of window size
            - Sequence no. >= 2 *window size

3.5. Connection-oriented transport – TCP

- Overview
- Point-to-point – one sender, one receiver, both transmitting data
- Reliable, in order byte stream – no 'msg boundaries'
- Pipelined – TCP congestion + flow control et window size
- Send + receive buffers – like SR
- Full duplex data bi-directional data flow in same connection
    - MSS – max segment size
- Connection-oriented – handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- Flow controlled – sender will not overwhelm receiver

3.5.1. Segment structure

- Header – 20 bytes, 32 bits across (UDP was 8)
    - Source port no., dest port no. – 16 bits each
    - Sequence no. + ACK no. – 32 bits each

- URG – urgent data (not generally used)
- ACK – ACK no. valid
- PSH – push data now (not generally used)
- RST, SYN, FIN – connection estab (setup, teardown commands)
- Internet checksum (as in UDP)
- Receive window – no. bytes receiver willing to accept

3.5.2. Reliable data transfer

- Checksum – computed over header + data
  - Checksum for sending data or ACK
- Sequence numbers are byte offsets – each pkt referred to by their starting byte no.
  - Each segment sent when – it is full (MSS) or it is not full, but times out
  - Sequence number = initial sequence number (ISN) + (k bytes sent)
  - ACK sequence no. – next byte of data expected by the receiver
    - Ack sequence no. = next expected byte = seqno + length(data)
  - TCP segment size
    - IP packet – no bigger than max transmission unit (MTU)
      - Max amount of IP layer can hand over to data-link layer e.g. up to 1500 bytes for Ethernet
    - TCP packet – IP packet with a TCP header and data inside
      - TCP header >= 20 byte long
    - TCP segment
      - No more than max segment size (MSS) byte
      - E.g. up to 1460 bytes from stream
      - MSS = MTU – (IP header) – (TCP header)
- Receiver sends cumulative acknowledgements (like GBN)
  - If an ack is lost from receiver to sender, and the next afterwards comes just fine, it will presume that the lost ack was ok as well
  - If data is lost from sender to receiver, even if the data afterwards is sent find, the ack from the receiver will be the last ack number sent
  - Piggybacking – both sides of a connection send some data (both sides act as 'sender' + 'receiver')
- Receiver can buffer out-of-sequence pkts (like SR)
  - Holds pkts that come after lost pkt until lost pkt is retransmitted and received
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout
  - Does not fire the whole window – will only transmit the leftmost segment
  - TCP timeout value – needs to adapt to the network conditions
    - Longer than RTT – but RTT varies
    - Too short – premature timeout, unnecessary retransmissions
    - Too long – slow reaction to segment loss + connection has lower throughput
  - Estimate RTT
    - SampleRTT – measured time from segment transmission until ACK receipt (ignore retransmissions)
    - SampleRTT – vary, want estimated RTT 'smoother'
      - Average several recent measurements, not just current
    - EstimatedRTT = (1 – a) * Estimated RTT + a * SampleRTT

            - Exponential weighted moving average – influence of past sample decrease exponentially
            - Typical value of a = 0.125
            - More weight is assigned to historic RTT value
    - Timeout interval – EstimatedRTT + 'safety margin'
        - Larger variation in EstimatedRTT → larger safety margin
        - Estimate SampleRTT deviation ('safety margin') from Estimated RTT
            - DevRTT = (1 – b) * DevRTT + b * | SampleRTT - EstimatedRTT |
            - Typical value of b = 0.25
        - TimeoutInterval = EstimatedRTT + 4 * DevRTT
    - Excluding retransmissions in RTT computation
        - Skewing of data – SampleRTT becomes the difference between the start of the retransmission and ACK of interpreted original transmission (even though it belongs to the retransmission)

3.5.3. Flow control

3.5.4. Connection management

3.6. Principles of congestion control

3.7. TCP congestion control