The Application Layer

Michael Brodskiy
Professor: E. Bernal Mor

September 28, 2023

- Network Applications
  - Social networking
  - Web
  - Text messaging
  - E-mail
  - Multi-user network games
  - Streaming stored video (YouTube, Hulu, Netflix)
  - P2P File Sharing
  - And many more
- Creating Network Applications
  - Write programs that:
    * Run on (different) end systems
    * Communicate over network
    * For example, web server software communicates with browser software
  - No need to write software for network-core devices (intermediate nodes)
    * Network-core devices do not run user applications
    * Applications on end systems allow for rapid application development and propagation
- Application Architecture
  - Network architecture — a set of layers and protocols
    * It is fixed, and provides the network application developer with specific set of services

  - Application Architecture — define how the application is structured over various end systems
    * Designed by the application developer
    * Predominant architectural paradigms
      · Client-server
      · Peer-to-peer (P2P)

- Client-server Architecture
  - Server
    * Always-on host
    * Permanent IP-address (like ID)
    * Often in data centers, for scaling
  - Clients
    * Contact, communicate with server
    * May be intermittently connected
    * May have dynamic IP addresses
    * Do not communicate directly with each other
  - Examples: HTTP, IMAP, SFTP

- Peer-Peer (P2P) Architecture
  - No always-on server
  - Arbitrary end systems directly communicate
  - Peers request service from other peers, provide service in return to other peers
    * Self scalability — new peers bring new service capacity, as well as new service demands
  - Peers are intermittently connected and change IP addresses
    * Complex management
  - Example: P2P File Sharing

- Process Communication
  - Process — program running within a host
    * Within same host, two processes communicate using inter-process communication, defined by OS (Operating System)
    * Processes in different hosts communicate by exchansing messages
  - Client process — process that initiates communication
  - Server process — process that wants to be contacted

  - Note: applications with P2P architectures have client processes 7 server processes

- Sockets
  - Process send/receives messages to/from its socket
  - Socket analogous to door
    * Sending process shoves message out the door
    * Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process

- Addressing Processes
  - To receive messages, a process must have an identifier
  - Host device has a unique IP address
  - Identifier includes both IP address and port numbers associated with process on host
    * HTTP server: 80
    * Mail server: 25
  - To send HTTP message to gaia.cs.umass.edu web server:
    * IP address: 128.119.245.12
    * Port number: 80

- An Application Layer Protocol Defines:
  - Types of messages exchanged
    * Example: request, response
  - Message syntax
    * What fields in messages & how fields are delineated
  - Message semantics
    * Meaning of information in fields
  - Rules for when and how processes send & respond to messages

- Application Layer Protocols can be
  - Open protocols
    * Defined in RFCs, everyone has access to protocol definition
    * Allows for interoperability
    * Example: HTTP, SMTP
  - Proprietary protocols

    - Example: Skype

- Transport Layer Services for Applications
  - Transport layer is on the other side of the "door"
  - There are multiple Transport-layer protocols that provide different services
  - The application developer must choose a Transport-layer protocol, depending on the services needed by the application
    - Examples: priority mail, express mail, certified mail
  - A Transport-layer protocol can provide a different array of services

- Transport Services
  - Data integrity/reliable transport
    - Some apps (*e.g.* file transfer, web transactions) require 100% reliable data transfer
    - Other apps (*e.g.* audio) can tolerate some loss
  - Timing
    - Some apps (*e.g.* Internet telephony, interactive games) require low delay to be "effective"
  - Throughput
    - Some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
    - Other apps ("elastic apps") make use of whatever throughput they get
  - Security
    - Encryption, data integrity, ...

- Internet Transport Protocol Services
  - TCP Service
    - Reliable transport between sending and receiving processes
    - Flow control — sender will not overwhelm receiver
    - Congestion control — throttle sender when network overloaded
    - Does not provide timing, minimum throughput guarantee, security
    - Connection-oriented service: setup required between client and service processes
  - UDP Service:
    - Unreliable data transfer between sending and receiving process
    - Does not provide reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

    * Connectionless service: no setup required

- Vanilla TCP & UDP sockets
  - No encryption
  - Clear text passwords sent into socket traverse Internet in clear text

- Transport Layer Security (TLS)
  - Provides encrypted TCP connections
  - Data integrity
  - End-point authentication
  - TSL implemented in Application Layer
    * Applications use TLS libraries, that use TCP in turn
  - TLS socket API
    * Clear text sent into socket traverse Internet encrypted
  - Datagram Transport Layer Service (DTLS) protocol
    * Adaptation of TLS to run over connectionless protocols such as UDP

- Designing Network Applications
  - It is a complex process
  - Requires knowledge of programming, software engineering, and networking
  - From a networking point of view, there are two major decisions:
    1. Type of application (aka Application Architecture)
       * Client-server vs. peer-to-peer
    2. Services requested to the Transport Layer
       * *E.g.* reliable vs. unreliable data transfer

- Web and HTTP
  - Web page consists of several objects, each of which can be store on different Web servers
  - Object can be HTML, JPEG, Java applet, audio file, etc.
  - Web page consists of a base HTML-file, which includes several referenced objects
  - Each object is addressable by a URL (Uniform Resource Locator), *e.g.*,

$$\underbrace{\texttt{www.someschool.edu/}}_{\text{host name}}\underbrace{\texttt{someDept/pic.gif}}_{\text{path name}}$$

- HTTP Overview
  - HTTP — Hypertext Transfer Protcol
  - Web's application layer protocol
  - Client/server model
    * Client: Browser that requests, receives (using HTTP protocol) and "displays" Web objects
    * Server: Web server sends (using HTTP protocol) objects in response to requests
  - Versions
    * HTTP/1.0 (RFC1945)
      · Original HTTP version (early 1990s)
    * HTTP/1.1 (RFC7230,...)
      · Used by most of the HTTP transactions
    * HTTP/2 (RFC7540,...)
      · Standardized in 2015 and increasingly used by browsers and servers
    * HTTP/3 (RFC9114)
      · IETF published it as a proposed standard in June 2022
  - HTTP used TCP (except HTTP/3):
    * Client initiates TCP connect (creates socket) to server, port 80
    * Server accepts TCP 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"
    * Server maintains no information about past client requests
  - Protocols that maintain "state" are complex
    * Past history (state) must be maintained
    * If server/client crashes, their view of "state" may be inconsistent, must be reconciled
  - Non-persistent HTTP
    1. TCP connection opened
    2. At most one object sent over TCP connection
    3. TCP connection closed

    * Downloading multiple objects requires multiple connections
  - Persistent HTTP

1. TCP connection opened to a server
2. Multiple objects can be sent over single TCP connection between client, and that server
3. TCP connection closed

- RTT (Round-Trip Time) definition: time for a small packet to travel from client to server and back
  * Includes propagation, queueing, and processing delays
- Non-persistent HTTP response time (per object)
  * One RTT to initiate TCP connection
  * One RTT for HTTP request/response
  * File/object transmission time
  * Non-persistent HTTP time $\approx$ 2RTT + transmission time (per object)
- Non-persistent HTTP issues:
  * Requires 2 RTTs per object
  * OS overhead for each TCP connection
  * Browsers often open parallel TCP connections to fetch referenced objects in parallel
- Persistent HTTP/1.1:
  * Server leaves connection open after sending response
  * Subsequent HTTP messages between same client/server sent over open connection
  * Client sends requests as soon as it encounters a referenced object
  * As little as one RTT for all the referenced objects (cutting response time in half)
- HTTP Request Message
  * Two types of HTTP messages: request, response
  * HTTP request message is in ASCII (human-readable format)

• HTTP Methods
  - GET method
    * To request an object
    * If user data (like form input), the data is included in URL field of HTTP GET request message (following a '?'):

      `www.somesite.com/animalsearch?table&chair`
  - POST method
    * Web page often includes form input

    - User input sent from client to server in entity body of HTTP POST request message
  - HEAD method
    - Requests headers (only) that would be returned if specified URL were requested with an HTTP GET method
  - PUT method
    - Uploads new file (object) to server
    - Completely replaces file that exists at specified URL with content in entity body
  - DELETE method
    - Deletes file specified in the URL field

- HTTP Response Status Codes
  - 200 — OK
    - Request succeeded, requested object later in this message
  - 301 — Moved Permanently
    - Requested object moved, new location specified later in this msg (Location:)
  - 400 — Bad Request
    - Request msg not understood by server
  - 404 — Not Found
    - Requested document not found on this server
  - 505 HTTP — Version Not Supported

- HTTP/2
  - Key goal: decreased delay in multiple-object HTTP requests
  - HTTP/1.1: introduced multiple, pipelined GETs over single TCP connection (persistent HTTP)
    - FCFS (First-Come-First-Served) scheduling: server responds in-order to GET requests
    - Head-Of-Line (HOL) blocking: with FCFS, small object may have to wait for transmission behind large object(s)
    - Loss recovery (retransmitting lost TCP segments) stalls object transmission
  - HTTP/2: increased flexibility at server in sending objects to client
    - Methods, status codes, most header fields unchanged from HTTP 1.1
    - Transmission order of requested objects based on client-specified object priority (not necessarily FCFS)

    * Push unrequested objects to client
    * Divide objects into frames, schedule frames to mitigate HOL blocking

- HTTP/2 to HTTP/3
  - HTTP/2 over single TCP connection means:
    * Recovery from packet loss still stalls all object transmissions
      · As in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling and increase overall application data throughput
    * No security over vanilla TCP connection
  - HTTP/3 operates over QUIC, a transport protocol built on UDP
    * Adds security, per object error, and congestion-control (more pipelining) over UDP

- Maintaining User/Server State: Cookies
  - Recall: HTTP GET/response interaction is stateless
  - Web sites and client browser use cookies to maintain some state between transactions
  - Four components:
    1. Cookie header line of HTTP response message
    2. Cookie header line in next HTTP request message
    3. Cookie file kept on user's host, managed by user's browser
    4. Back-end database at Web site

- HTTP Cookies: Comments
  - What cookies can be used for:
    * Authorization
    * Shopping carts
    * Recommendations
    * User session state (Web e-mail)
  - Challenge: How to keep state
    * Protocol endpoints: maintain state at sender/receiver over multiple transactions
    * Cookies: HTTP messages carry state
  - Cookies and Privacy:
    * Cookies permit sites to learn a lot about you on their site

        * Third party persistent cookies (tracking cookies) allow common identity (cookie value) to he tracked across multiple web sites

- Web Caches (Proxy Server)
  - Goal: satisfy client request without involving origin server
  - User configures browser to point to a web cache
  - Browser sends all HTTP requests to cache
    * If object in cache, cache returns object to client
    * Else cache requests object from origin server, caches received object, then returns object to client
  - Web cache acts as both client and server
    * Server for original requesting client
    * Client to origin server
  - Typically, cache is installed by ISP (university, company, residential ISP)
  - Why Web Caching?
    * Reduce response time for client request
      · Cache is closer to client
    * Reduce traffic on an institution's access link
    * Internet is dense with caches
      · Enables "poor" content providers to more effectively deliver content
- Conditional GET
  - Web cache: specify date of cached copy in HTTP request:

    If-modified-since: <date>

  - Server: response contains no object if cached copy is up-to-date

    HTTP/1.1 304 Not Modified

  - Goal: don't send object if cache has up-to-date cached version
    * No object transmission delay
    * Lower link utilization
- E-Mail
  - Three major components
    1. User agents (UA)
       * A.k.a. "mail reader"

    * Composing, editing, reading mail messages
    * E.g., Outlook, iPhone mail client
    * Outgoing, incoming messages stored on server
  2. Mail servers
    * Mailbox contains incoming messages for user
    * Message queue of outgoing (to be sent) mail messages
  3. Simple Mail Transfer Protocol (SMTP)
    * Between mail servers to send e-mail messages
      · Client: Sending mail server
      · Server: Receiving mail server
    * SMTP can also be used by user agents to send e-mails to mail server

- E-Mail: SMTP (RFC 5321)
  - Uses TCP to reliably transfer e-mail message from client to server (mail server port 25)
  - Direct transfer: sending mail server (acting like client) to receiving mail server
  - After TCP connection is established, three phases of transfer
    * Handshaking (greeting)
    * Transfer of messages
    * Closure
  - Command/response interaction (like HTTP)
    * Commands: ASCII text
    * Response: status code and phrase
  - Messages must be in 7-bit ASCII
  - Comparison with HTTP:
    * HTTP: pull; SMTP: push
    * Both have ASCII command/response interaction and status codes
    * HTTP: each object encapsulated in its own response message; SMTP: multiple objects sent in multipart message
    * SMTP: multiple objects sent in multipart message
    * SMTP uses presistent connections
    * SMTP requires message (header & body) to be in 7-bit ASCII

- E-Mail Access Protocols
  - SMTP: delivery/storage of e-mail messages to receiver's server
  - E-Mail access protocol: Bob's user agent starts the communication for retrieval from server

    * IMAP: Internet Mail Access Protocol (RFC 3501): messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
  - HTTP: Gmail, Hotmail, etc. provides web-based interface to send or to retrieve e-mail messages

- Domain Name System (DNS)
  - People have many identifiers:
    * SSN, name, passport number, etc.
  - Internet hosts, routers
    * IP addresss — used for addressing packets
    * Name, like CS.UMASS.EDU is used by humans
  - Distributed database implemented in hierarchy of many name servers
  - Application-layer protocol: hosts and name servers communicate to resovle names (address/name translation)
    * Note: core Internet function, implemented as application-layer protocol
    * Complexity at network's "edge"

- DNS Services
  - Hostname to IP address translation
  - Host aliasing
    * Canonical, alias names
  - Mail server aliasing
  - Load distribution
    * Replaced Web servers: many IP addresses correspond to one name
  - Why not centralize DNS?
    * Single point of failure
    * Traffic volume
    * Distant centralized database
    * Maintenance
    * It doesn't scale!
  - Client wants IP accress for Amazon: 1^{st} approximation:
    * Query sent to root server to find .com DNS server (root)
    * Query sent to .com DNS server to get amazon.com DNS server (top level domain)
    * Query sent to amazon.com DNS server to get IP address for www.amazon.com (authoritative)

- Local DNS Server
  - Local DNS Server — helps hosts to resolve a query
  - Does not strictly belong to hierarchy
  - Each ISP or institution (residential ISP, company, university) has one
    * Also called "default name server"
  - When host makes DNS query, query is sent to its local DNS server
    * Has local cache of recent name-to-address translation pairs (but may be out of date!)
    * Acts as proxy, forwards query to hierarchy
- DNS: Root Name Servers
  - Official, contact-of-last-resort by name servers that cannot resolve name
  - Root name servers provide the IP addresss of TLD servers
  - Incredibly important Internet function
    * Internet couldn't function without it!
  - ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain
  - 13 logical root name "servers" worldwide, each "server" replicated many times (about 200 servers in the US)
- TLD and Authoritative Servers
  - Top-level domain (TLD) servers:
    * Responsible for .com, .org, .net, .edu, etc.
    * Network solutions maintains servers for .com, .net TLD
    * Educause for .edu TLD
  - Authoritative DNS servers:
    * Organization's own DNS server(s), providing authoritative hostname to IP mappings for organizations named hosts
- DNS Name Resolution: Iterated Query
  - Example: host at ENGINEERING.NYU.EDU wants IP address for GAIA.CS.UMASS.EDU
  - Iterated Query
    * Contacted server replies with name of server to contact
    * "I don't know this name, but ask this server"
- DNS Name Resolution: Recursive Query

- Same situation as above
- Recursive Query
  - Puts burden of name resolution on contacted name server
  - Heavy load at upper levels of hierarchy?

- Caching, Updating DNS Records
  - Once (any) local DNS server learns mapping, it caches mapping
    - Cache entries timeout (disappear) after some time (TTL)
    - TLD servers typically cached in local name servers
      - Thus, root name servers not often visited
    - 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
    - Update/notify mechanisms proposed IETF standard
      - RFC 2136

- DNS Records
  - DNS: distributed database storing Resource Records (RR)

$$\boxed{\text{RR Format: (NAME, VALUE, TYPE, TTL)}}$$

  - TYPE=A
    - 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" (the real) name
      - WWW.IBM.COM is really SERVEREAST.BACKUP2.IBM.COM
    - VALUE is canonical name
  - TYPE=MX
    - NAME is a mail domain
    - VALUE is hostname of mail server associated with NAME

- DNS Protocol Messages
  - DNS query and reply messages, both have same format:

        - Message header:
            - Identification — 16 bit number for query; reply to query uses same number
            - Flags — query or reply; recursion desired/available; if reply is authoritative

- Inserting Records into DNS
    - DNS registrar: commercial entity that verifies the uniqueness of a domain name
        - Enters the domain name into the DNS database
        - Collects a small fee from you for its services
    - There are many registrars that compete for customers
    - The ICAAN accredits the various registrars
    - Example: a new start-up "Network Utopia"
    - Register domain name NETWORKUTOPIA.COM at some DNS registrat (e.g., Network Solutions)
        - Provide names, IP addresses of authoritative name server (primary and secondary)
        - Registrar inserts NS and A type RRs into .com TLD server for each authoritative name server:

          (NETWORKUTOPIA.COM, ADNS1.NETWORKUTOPIA.COM, NS)

          (ADNS1.NETWORKUTOPIA.COM, 212.212.212.1, A)
    - Create authoritative server locally with IP address 212.212.212.1 and enter:
        - Type A record for your web server: (WWW.NETWORKUTOPIA.COM, "IP ADDRESS", A)
        - Type MX record for your mail server:

          (NETWORKUTOPIA.COM, MAIL.NETWORKUTOPIA.COM, MX)

          (MAIL.NETWORKUTOPIA.COM, "IP ADDRESS", A)

- Peer-To-Peer (P2P) Architecture
    - No always-on server
    - Arbitrary end systems directly communicate
    - Peers are intermittently connected and change IP addresses
    - Examples:
        - File distribution (BitTorrent)

    * Streaming (KanKan)
    * VoIP (Skype)

- P2P File Distribution: BitTorrent
  - File divided into 256kb chunks
  - Peers in torrent send/receive file chunks
  - Tracker: tracks peers participating in torrent
  - Torrent: group of peers exchanging chunks of a file
  - Peer joining torrent:
    * Has no chunks, but will accumulate them over time from other peers
    * Registers with tracker to get list of peers, connects to subset of peers ("neighbors")
  - While downloading, peer uploads chunks to other peers
  - Peer may change peers with whom it exchanges chunks
  - Churn: peers may come and gp
  - Requesting Chunks:
    * At any given time, different peers have different subsets of file chunks
    * Periodically, a certain peer asks each peer for list of chunks they have
    * The peer then requests missing chunks from peers, rarest first
  - Sending Chunks: tit-for-tat
    * A peer sends chunks to those four peers currently sending them chunks at highest rate
      · Other peers are choked by the peer (do not receive chunks)
      · Re-evaluate top 4 every 10 seconds
    * Every 30 seconds randomly "unchoke" another peer
    * Newly chosen peer may join top 4

- Video Streaming and CDNs: Context
  - Stream video traffic: major consumer of internet bandwidth
    * Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic
  - Challenge: scale — how to reach about 1B users?
    * Single mega-video server won't work
  - Challenge heterogeneity
    * Different users have different capabilities
  - Solution: distributed, application-level infrastructure

- Multimedia: Video
  - Video: sequence of images displayed at constant rate
    * Like 24 frames per second
  - Digital image: array of pixels
    * Each pixel represented by bits
  - Coding: use redundancy within and between images to decrese # used to encode image
    * Spatial (within image) — Instead of sending N values of same color (all purple) send only two values: color value (purple) and number of repeated values (N)
    * Temporal (from one image to the next) — Instead of sending complete frame at $i + 1$, send only differences from frame i
  - CBR (Constant bit rate): video encoding rate fixed
  - VBR (Variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
  - Examples:
    * MPEG1 (CD-ROM) 1.5 Mbps
    * MPEG2 (DVD) 3-6 Mbps
    * MPEG4 (Used in Internet) 64kbps-12Mbps
  - Main challenges:
    * Server-to-client rate will varyover time, with changing network congestion levels (in house, in access network, in network core, at video server)
    * Packet loss and delay due to congestion will delay playout, or result in poor video quality
- Streaming stored video
  - Continuous playout constraint: once client playout begins, playback must match original timing
    * But network delays are variable (jitter), so we will need client-side buffer to match playout requirements
  - Client-side buffering and playout delay: compensate for network-added delay and jitter
- Streaming Multimedia: DASH
  - 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 transmission rate
    * Consulting manifest, requests one chunk at a time
      · Chooses maximum coding rate sustainable given current transmission rate
      · Can choose different coding rates at different points in time (depending on available transmission rate 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 transmission rate available)
      · Where to request chunk (can request from URL server that is "close" to client or has high available transmission rate)

- Content Distribution Networks (CDNs)
  - Challenge: how to stream content (selected from millions of videos) to hundreds of thousdands of simultaneous users?
  - Option 1: single, large "mega-server"
    * Single point of failure
    * Point of network congestion
    * Long path to distant clients
    * Multiple copies of video sent over outgoing link
    * Doesn't scale!
  - Option 2: store/server multiple copies of videos at multiple geographically distributed sites (CDN)
    * Enter deep: push CDN servers deep into many networks of access ISPs
      · Close to users
      · Akamai, 240000 servers deployed in more than 120 countries
    * Bring home: smaller number (10's) of larger clusters in POPs (Points of Presence) near (but not within) access ISPs
      · Typically, POPs are Internet Exchange Points (IXPs)
      · Used by Limelight
  - CDN: stores copies of content at CDN nodes
  - Subscriber requests content from CDN

    * Directed to nearby copy, retrieves content
    * May choose different copy if network path congested
- Over-the-top (OTT)
    * OTT challenges: coping with a congested Internet
        · From which CDN node to retrieve content?
        · Viewer behavior in presence of congestion?
        · What content to place in which CDN node?