Design and Engineering of Computer Systems

Lecture 24: Application layer protocols

Mythili Vutukuru

IIT Bombay

Application layer

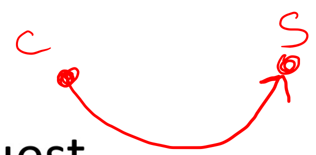
- User applications (world wide web, e-commerce, video streaming, social media, …) exchange information over the Internet
  - Application software is built using socket API or other communication libraries
- Application layer protocols: a set of rules by which different components of an application understand each other
  - Example: which messages should be exchanged and in what order, format of the message headers, …
- Many application layer protocols are in use today
  - Hyper Text Transfer Protocol (HTTP) developed for web, used in many other apps
  - Simple Mail Transfer Protocol (SMTP) for email transfer
- Protocol only specifies how to communicate, not what data is exchanged
  - HTTP can be used to exchange HTML web pages, images, videos, ..

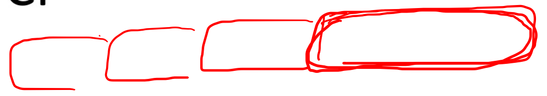
Hyper Text Transfer Protocol (HTTP)

- HTTP developed to help web clients (browsers) and web servers (which host websites) to communicate with each other
  - Widely used in many other applications as well
- What happens when you access a web page?

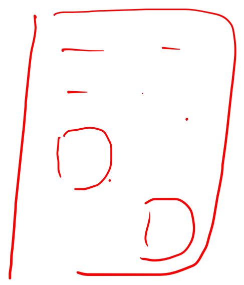
nptel.ac.in/courses/. . . . _ . . . .

  - User types URL (Uniform Resource Locator = domain name+location of content) in the browser's address bar
  - Web server listens on port 80 (or 443 with secure HTTP) on a public IP address
  - Client obtains server IP address via DNS, opens TCP connection, sends HTTP request
  - Server processes received request, sends HTTP response back via TCP
  - Client browser displays (renders) HTTP response received





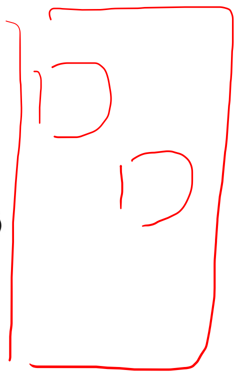
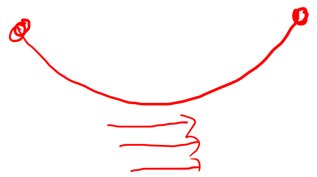
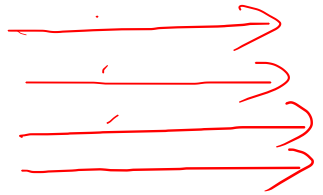
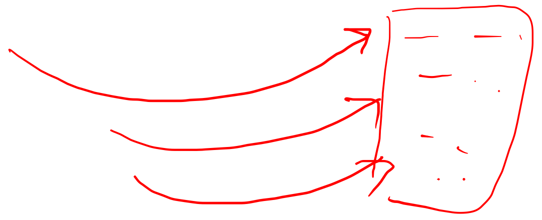
- Data exchanged between clients and servers consists of
  - Web pages with HTML (Hyper Text Markup Language) content
  - Images, multimedia files, and various other embedded objects in a web page



HTTP request

- HTTP request is the first step by client in request-response communication
  - Can be used to fetch information (GET), update information (POST), …
  - Specifies the resource location at server which has to be fetched/updated
  - Other header fields like user agent information, preferred language, ..
  - Body of the request contains actual content (e.g., updated value in POST)
- A web page consists of multiple objects (main HTML page, embedded images, style files, web page scripts to run at client side, …)
  - Once browser gets a response with content of one web page, separate HTTP GET requests made for embedded objects in that page
  - Persistent connections: client can retain/reuse same TCP connection for future HTTP requests, instead of opening a separate TCP connection for each HTTP request
  - Parallel connections: client can open multiple TCP connections in parallel in order to fetch multiple web objects concurrently for faster performance

GET /course/. . . .
headers
body

HTTP response

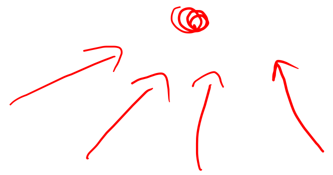
- HTTP response is returned by server in response to HTTP request
  - Contains status code indicating success or failure (200 OK, 404 server not found, …)
  - Various headers, like type of content being sent, timestamps, …
  - Body of HTTP message has actual content being fetched (by GET request)
- Two ways of generating HTTP response at server
  - Static response: already exists at server (e.g., image file on disk), simply transfer to client
  - Dynamic response: response to be computed and sent to client (e.g., user enters key words in search, e-commerce web site has to search its product database and return response)
- HTTP is stateless: server sends response and forgets about client request
  - How do websites track users? Using HTTP cookies (special data returned in HTTP response that is sent in future HTTP requests to help identify the client)
- HTTP responses can be cached in browsers or other caches in network (more later)
  - HTTP response headers specify how long the item is valid in cache

$\boxed{\text{Status code} \\ \text{headers} \\ \text{context}}$

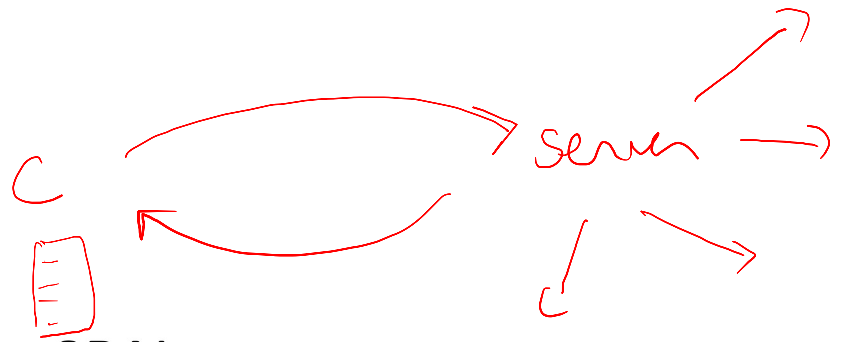
req

resp
+
cookie

cookie



Web application frameworks



- Static content is typically distributed and served by CDNs
- Most web servers/applications today generate and serve dynamic content
    - HTTP request contains user information (e.g., key words in searching e-comm products)
    - Web server processes request, contacts several other application servers/databases, and dynamically generates response (e.g., list of products matching search key words)
    - Client-side scripts dynamically modify received response when displaying, in order to enhance user experience (e.g., change display format of web page)
- Web application frameworks ease development of web applications in complex real-life computer systems
    - In-built support for web servers, databases, and other required components
    - Easy scripting for request parsing, response generation
    - Support for a wide range of programming languages
    - Tools to ease development of user-facing frontend, server processing at backend
    - One thread per connection model vs. event-driven programming API

HTTP optimizations

- HTTP evolved from popular version 1.1 to version 2, version 3, ..
- HTTP/2 improves over HTTP/1.1 in several ways
  - Efficient data transfer using binary compressed format (not text)
  - Server can push objects that it knows client will need, even before client requests it
  - Multiple streams over same TCP connection for multiple objects, with prioritization
- Why multiple streams in same TCP connection?
  - Effectively use TCP bandwidth in steady state, no overhead of multiple TCP connections
  - Problem: if packet lost in one stream, all streams blocked due to in-order delivery of TCP
- HTTP/3 uses new transport protocol QUIC (Quick UDP Internet Connections)
  - Based on UDP, congestion control and reliability handled inside application
- Other optimizations also used for web pages to load faster
  - Redesign web pages so that important content comes first, page starts to render sooner

Data serialization formats

- Application layer data structures/objects need to be serialized into a stream of bits in order to transfer in a message (e.g., as part of HTTP request or response)
  - Also need to deserialize received bits in message into a structure/object
- JSON (JavaScript Object Notation) is popular text-based format to store data structures, with support for serialization/deserialization in many languages
  - Example, received data str = {"name": “banana", “type": “fruit”, “color": “yellow"}
  - Can be parsed into a structure in many languages, obj = JSON.parse(str)
  - Once converted into object, individual fields can be accessed (obj.name, obj.type, obj.color)
- Protocol buffers is popular library used for serialization of data structures
  - User can define data structures (messages) which have various fields of different data types
  - Protocol buffers compiler automatically generates code to set/get various fields of message, convert from message to output stream, convert from input stream to message
  - Application code can use these generated functions to access message objects
- Several such data serialization formats exist, in several libraries and frameworks

Email transfer via SMTP

$$\begin{tikzcd}
 & \text{sender.com} \arrow[r, "\text{smtp}"] & \text{rx.com} \arrow[dr] & \\
A \arrow[ur, bend right, "\text{smtp}"'] & & & B
\end{tikzcd}$$

- Email is stored and retrieved from mail servers: server process listening on well-known port (25), running software to manage email
- Users access mail servers via user agents, e.g., special email clients
  - Webmail: user agents use HTTP to communicate with mail servers
- Simple Mail Transfer Protocol (SMTP) is a push-based protocol to push email messages from user agent to mail server, or between mail servers
  - Unlike HTTP which was designed to be a pull-based protocol
- Example: send email from userA@sender.com to userB@rx.com
  - User agent of userA obtains IP address of mail server in sender.com via DNS
  - User agent of userA opens TCP connection to mail server of sender.com, delivers email via SMTP messages
  - Mail server of sender.com delivers email to mail server of rx.com via SMTP messages
  - User agent of userB retrieves email from mail server of rx.com at a later time via IMAP or HTTP or other pull-based protocols

Remote Procedure Call (RPC)

- Alternate way of thinking about client-server communication: clients access server as if calling functions in local code (RPC software takes care of remote execution)
- Example: user wants to search products matching key words on e-comm website
  - User sends key words in HTTP request, server returns HTTP response containing list of products
  - Or, client invokes remote function at server like "search(key words)" and obtains response
- RPC libraries used to implement RPC in client-server applications
  - Provide interface description language: client and server use common definition of services/functions at server, messages to be exchanged as arguments to the function
  - RPC library automatically generates client stub (serialize message, communicate with server over sockets) and server stub (communicate with client over sockets, serialization of messages, skeleton service implementation)
  - User writes the actual application logic of the service implementation, which is invoked by the RPC library when requests are received from clients

RPC design choices

- Many different RPC frameworks available today (e.g., gRPC), with choices for
  - Serialization/deserialization formats
  - Communication mechanism (TCP, UDP), format of messages exchanged
  - Whether RPC is blocking or event-driven
- RPC libraries need to handle network failures (unlike local function call)
  - Some implementations try once to execute function at server and do nothing if failure, some implementations try multiple times till function executed at server at least once
  - Server code must be idempotent (ok to repeat execution of function) or RPC library must filter duplicate requests for correctness (if not idempotent)
- Pros and cons of RPC vs. other application-layer protocols
  - Client and server tied together more closely with RPC (need to agree on function names, number of arguments etc.), whereas other protocols have less dependency and hence more flexibility
  - RPC can handle many different applications, instead of separate protocols for each
  - RPC communication can be better optimized for specific application (e.g., no unnecessary application protocol headers)

Summary

- In this lecture:
  - Application layer protocols: HTTP, SMTP
  - Data serialization in applications
  - Remote Procedure Calls
- Capture packets passing through your computer while you access a web page. Inspect the packets using a tool like Wireshark. Observe the various HTTP headers in a request and response.