Application Layer

Dr. Keyur Parmar

S. V. National Institute of Technology (NIT), Surat

Outline

1. Network Applications - Principles
2. Web Application and HTTP
3. Electronic Mail Application and SMTP
4. Domain name system (DNS)
5. Peer-to-Peer File Distribution
6. Video Streaming and DASH
7. Socket Programming

Network Applications - Principles

Why do we need networking infrastructure?

List few applications that require networking infrastructure.

Internet Applications – Examples

- Text-based applications
  - Text e-mail, Remote access to computers, File transfers, Newsgroups
- World Wide Web
  - Web surfing, Search, Electronic commerce
- Video conferencing
- Multiplayer online games
- Social networking applications
- Mobile payment apps
- Messaging apps

Goal: To write distributed and interactive applications that run on different hosts.

What do we need to learn to write distributed applications?

- How to select transport layer protocol
- Application Programming Interface (API)

- Application layer: Facilitates communication between applications running on different hosts

Network Applications - Architectures

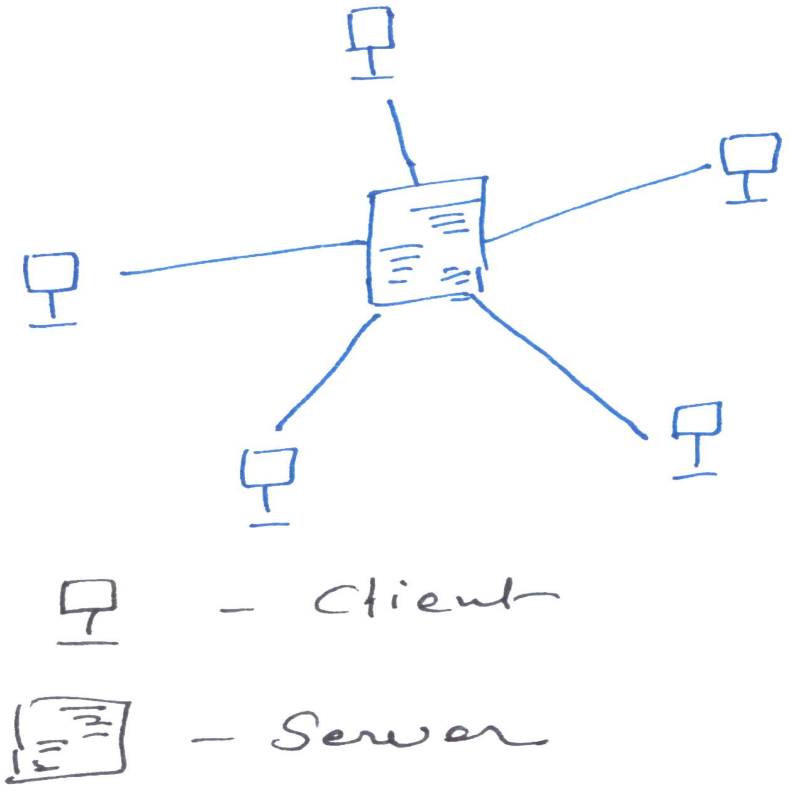
1. Client-server architecture
2. Peer-to-peer (P2P) architecture

Client-server Architecture

Client-server Architecture

- Client-server architecture consists of
  - Client - Host that requests data from other hosts.
  - Server - Host that serves requests of other hosts.
- Clients do not interact with each other.
- Servers are always-on hosts, and they usually have well-known address (i.e., IP address).
- Examples - Web, File Transfer, and Email

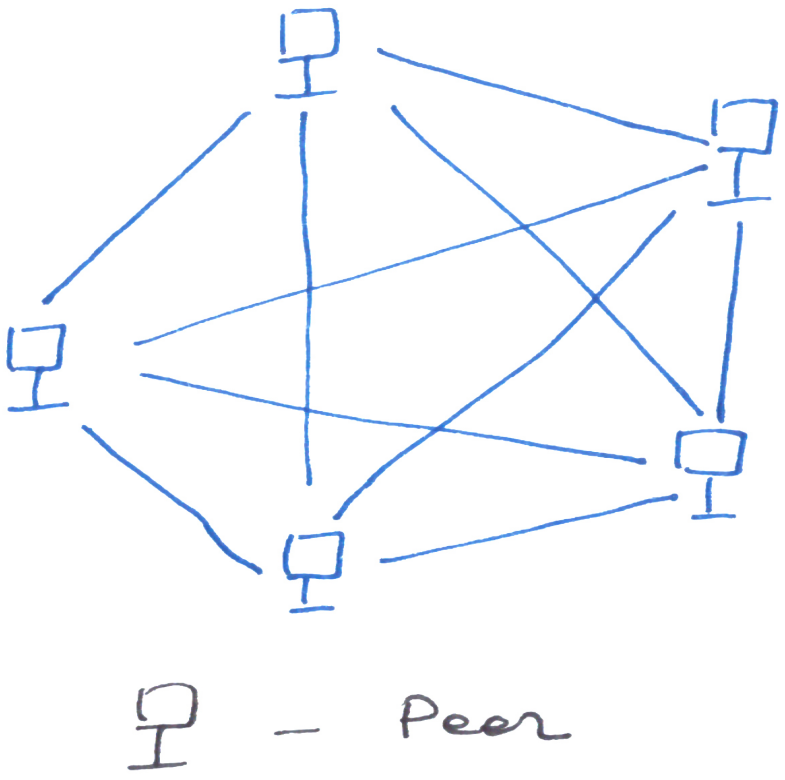
Client-server Architecture



Peer-to-peer (P2P) Architecture

- Applications running on different hosts (i.e., peers) interact with each other directly.
- Does not require dedicated servers.
- Example - BitTorrent and Bitcoin

Peer-to-peer (P2P) Architecture



Network Applications - Architectures - Pros and Cons

- Self scalability
  - Peer-to-peer architecture - Self scalable
  - Client-server architecture - Does not scale automatically.

Network Applications - Architectures - Pros and Cons

- Cost effective
  - Peer-to-peer architecture - Doesn't require server infrastructure and server bandwidth.
  - Client-server architecture - Requires server infrastructure and server bandwidth.

Network Applications - Architectures - Pros and Cons

- Security, Performance, and Reliability
  - Peer-to-peer architecture - Not inherently secure, efficient, and reliable.
  - Client-server architecture - Secure, efficient, and reliable as compared to the peer-to-peer architecture.

Client-Server Processes

- In the client-server architecture and peer-to-peer architecture, we label the hosts as client and server, depending on their role in the communication process.

How do the programs running on different hosts interact with each other?

Sockets

- Socket - A software interface between the application layer and the transport layer.
- Sockets are also known as Application Programming Interface (API) between the application and the computer network.
- A process (i.e., running program) sends/receives data to/from a process running on different host using sockets.

Sockets

- Application developer provides following information.
  - Transport protocol
  - Transport layer parameters such as maximum buffer and maximum segment size.

How do we identify the processe running on a different host?

Process - Address

- To uniquely identify a process, use
  - Address of the host - IP address
  - Process identifier - Port number

Well-known Port Numbers

- FTP (Data) - 20
- SSH (Remote Login Protocol) - 22
- Telnet - 23
- Simple Mail Transfer Protocol (SMTP) - 25
- Domain Name System (DNS) - 53
- HTTP - 80
- HTTPS - 443

Port Numbers - Internet Assigned Numbers Authority (IANA).

How do we decide the transport layer protocol for our web application?

Transport Layer Services

Transport Layer Services

- Reliable Data Transfer
- Throughput
- End-to-end Delay
- Security

Reliable Data Transfer

- Reliable Data Transfer - Data forwarded by the source application will reach the destination application completely and correctly.
- Loss-tolerant Applications - Applications that can withstand the data loss. For example, conversational audio/video applications.

Throughput

- Throughput - The rate at which the destination application receives data forwarded by the source application.
- If the communication link is shared by many hosts, the throughput fluctuates with time.
- Bandwidth-sensitive Applications - Applications with specific throughput requirements. For example, live audio/video communication.
- Elastic Applications - Applications without any specific throughput requirements. For example, file transfer.

End-to-end Delay

- End-to-end Delay - Time required to exchange data from the source application to the destination application.
- Real-time Applications - Applications having time constraints related to end-to-end delays.
- Examples: Teleconferencing and multiplayer games

Security

- Security - Security service provides protection of data from adversaries. Security services include confidentiality, authentication, availability, integrity, etc.
- Examples: Email and digital payment

Transport Layer Protocols

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

Transmission Control Protocol (TCP) - Services

- Connection-oriented service - Establish the full-duplex TCP connection (Handshaking between the client and server) between the sockets of client and server processes.
- Reliable Data Transfer Service - Data forwarded by the source application will reach the destination application completely and correctly.

User Datagram Protocol (UDP) - Services

- Lightweight protocol - Provides minimal services.
- Connectionless Service - No handshaking before exchanging data between the client and server processes.

If UDP provides only minimal services, why do we use UDP for any application?

Transport Layer Protocols

- Transport layer protocols, TCP and UDP, do not provide any of the following services.
  - Throughput
  - End-to-end delay
  - Security

Web Application and HTTP

Web Application - Advantages

- Contents on demand
  - Traditional radio and television - Broadcast contents at designated date/time.
  - Web application - Provides contents on demand.
- Platform for other applications such as Youtube, Web-based email, etc.
- Easy to publish contents.

What is Hypertext Transfer Protocol (HTTP)?

What Is a Protocol?

- A protocol defines
  - Format of messages exchanged between two or more communicating entities
  - Order of messages exchanged between two or more communicating entities
  - Actions taken on the transmission and/or receipt of a message or other event

Hypertext Transfer Protocol (HTTP)

- HTTP - Application layer protocol of the web.
- HTTP specifications define the communication protocol between
  - Client program
  - Server Program
- Hosts exchange HTTP messages to communicate with each other.
- Stateless protocol - HTTP server does not retain any information about the clients' requests.

Terminology

- Objects - Files such as pdf document, jpg image, CCS style sheet, mp4 video clip, etc.
- Objects - Referred by a unique address, i.e., URL.
- A URL consists of
  - Host name where the object is located
  - Path to the object
- Example: `http://keyurparmar.in/Teaching.html`
  - Host name - `keyurparmar.in`
  - Path to the object - `/Teaching.html`

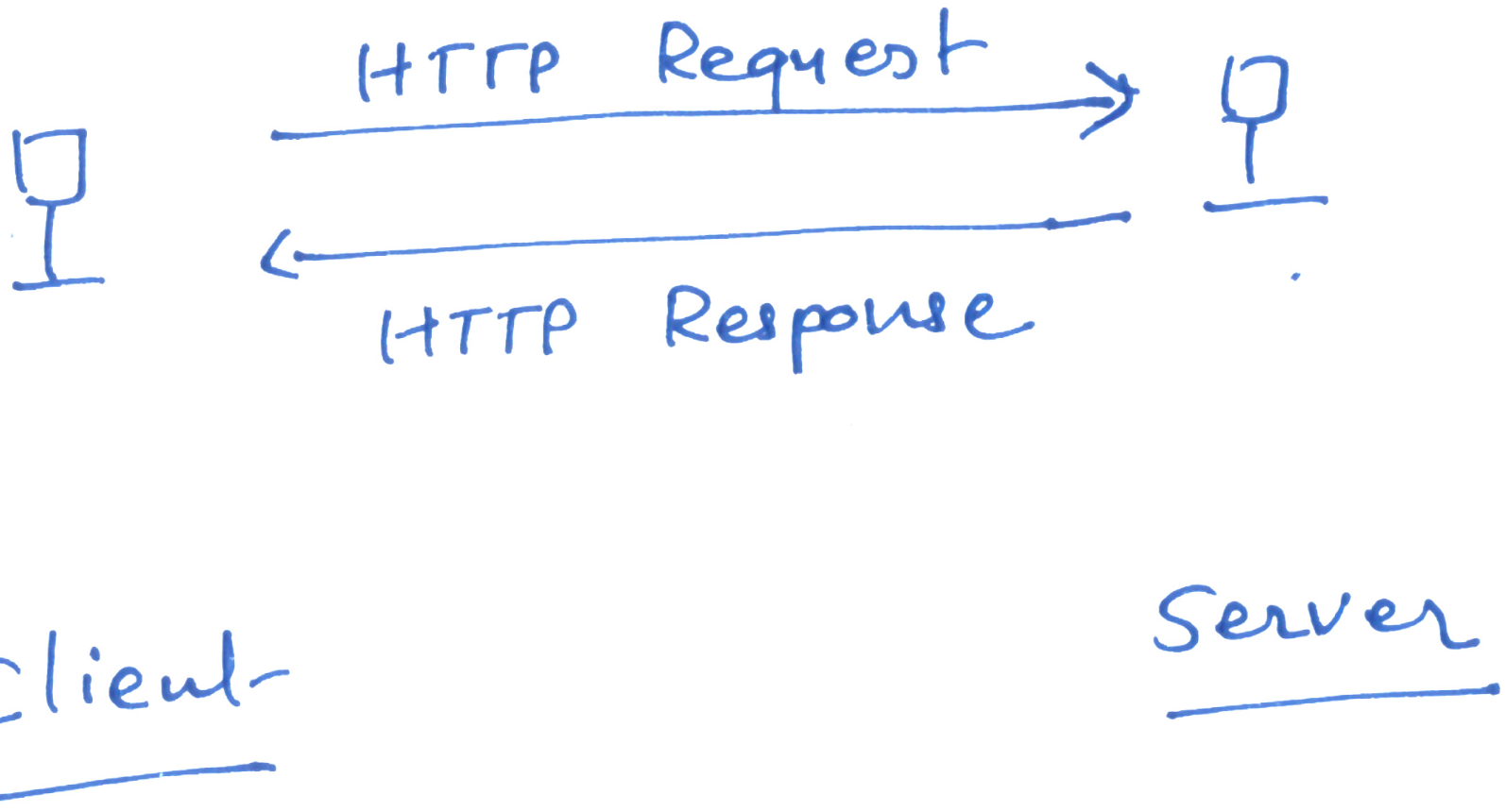
Terminology

- Web page consists of
  - Base HTML file
  - Object(s)
- Example: If a Web page consists of a base file and two images, then it has total 3 objects.

HTTP Request-Response Behavior

- User - Requests a Web page.
- Browser - Sends HTTP request messages to the server.
- Server - Process the HTTP request message, and sends HTTP response messages to the browser.
- The response message consists of the Web page.

HTTP Request Message Format



How many HTTP request messages the client sends to the server?

Should each HTTP request-responce pair require a separate TCP connection?

Persistent and Non-persistent Connections

- Persistent Connections - Application uses a single TCP connection to send and receive multiple requests and responses.
- Non-persistent Connections - Application uses a new TCP connection for each request-response pair.
- By default, HTTP uses persistent connections. However, it can be configured to use non-persistent connections.

How does the HTTP transfer a Web page from the server to the client when the connection is non-persistent?

Transferring Web Page - HTTP with Non-persistent Connections

- Web page URL: http://keyurparmar.in/index.html
- Assume that the URL consists of a base HTML file and 5 images.

Steps: Transferring Web Page - HTTP - Non-persistent Connections

1. Client process initiates a TCP connection to the server.
2. A client process sends the HTTP request message to the server.
3. Server process retrieves the object from storage and encapsulates the object in HTTP response message, and forward the message to the client process.

Steps: Transferring Web Page - HTTP - Non-persistent Connections

4. Server process imitates to close the TCP connection.
5. Once the client process receives the HTTP message from a server, the TCP connection terminates.
6. The client process examines the base HTML file and if it references any objects, it again initiates TCP connections with the server to get the copy of those objects.

Transferring Web Page - HTTP with Non-persistent Connections

- While transferring a Web page, a TCP connection is closed after the server sends an object, i.e., the connection does not persist for other objects of the same Web page.
- To transfer other objects from the server, we need to establish separate TCP connections.
- Non-persistent TCP connection
  - One HTTP request message from the client.
  - One HTTP response message by the server.

TCP - Non-persistent Connections

- Serial TCP connections
- Parallel TCP connections

How much time does the HTTP take to get the object form the server?

Round Trip Time (RTT)

- Round-trip time(RTT) - Time for a packet to travel from client to server and then back to the client.
- Round Trip Time
  - TCP Three-way Handshake
  - HTTP Request-Response

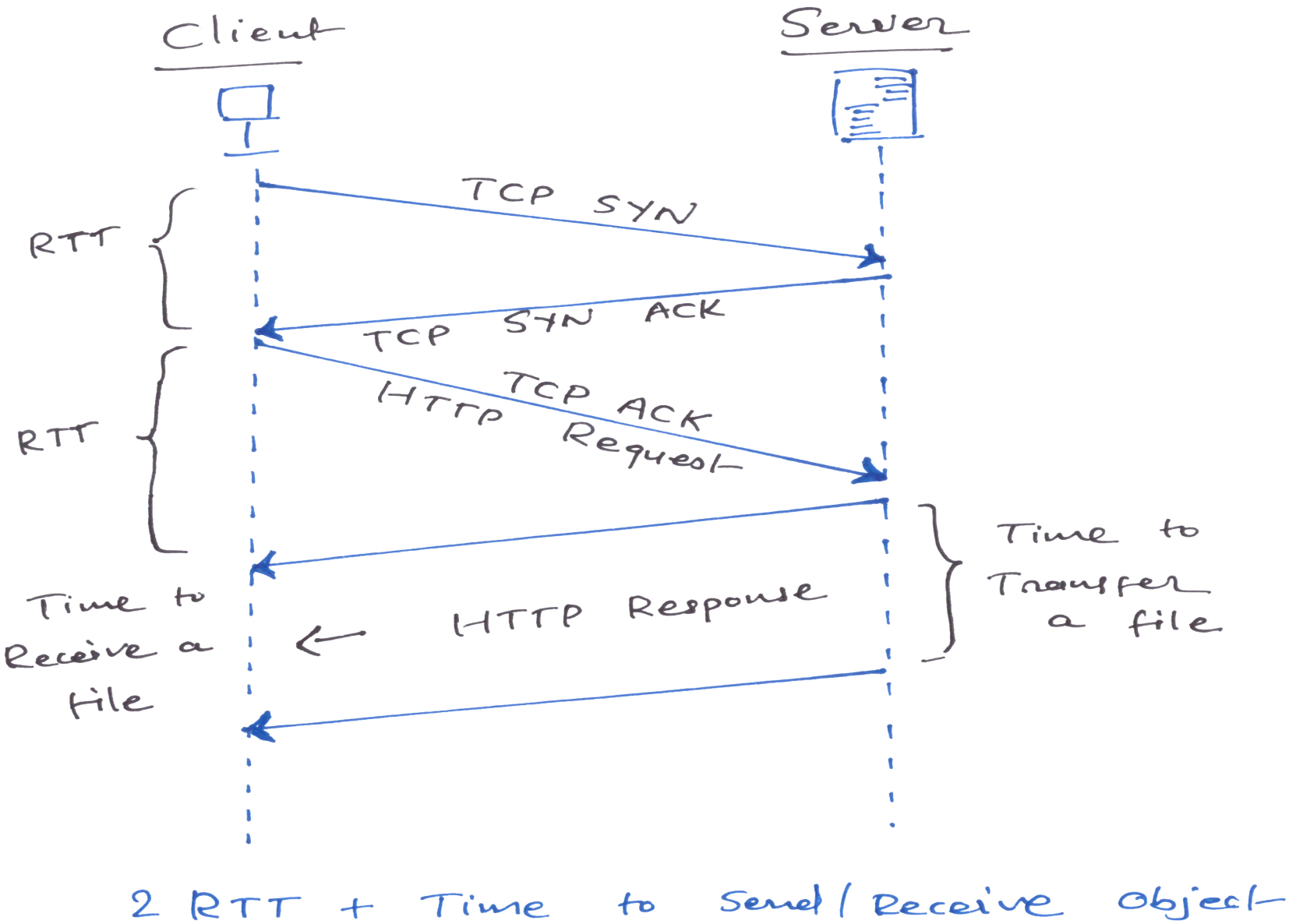
Round Trip Time (RTT)

- TCP Three-way Handshake
  - Request from client to establish TCP Connection
  - Server acknowledgment
  - Client acknowledgment

Round Trip Time (RTT)

- HTTP Request-Response
  - HTTP Request message
  - HTTP Response message
- Client sends the acknowledgment and HTTP request message at the same time.
- Total Response Time = 2 RTTs + Transmission time (server)

HTTP Request Message Format



Non-persistent Connections – Disadvantages

- Memory
  - TCP connection for each requested object.
  - Server – Significant overhead
- Time
  - 2 RTTs + Transmission time

Can we improve the time and memory as required by the non-persistent connections?

Persistent Connections

- Only one TCP connection for all request-response pairs.
- Supports multiple requests and responses at the same time over the same TCP connection.
- For example, a client may send multiple requests without waiting for the server to respond the requests.

HTTP Message Format

HTTP Message Format

- HTTP Messages
  - Request message
  - Response message

HTTP Request Message

- A request message can typically have one or more lines.
- HTTP messages end with carriage return and a line feed.

HTTP Request Message

- Request Line
  - Methods – GET, POST, HEAD, PUT, DELETE.
  - URL
  - HTTP Version

HTTP Request Message - Methods

- HTTP Methods
  - GET - To request an object from the server.
  - HEAD - To request an object header information only (not the actual object).
  - POST - To send data from client to server. Entity body consists of data. For example, search string.
  - PUT - To create a new resource at the server (or upload a file).
  - DELETE - To delete a specific resource from the server.

HTTP Request Message

- Header Line(s)
  - Host – Host name
  - Connection – keep-alive or close – Specifies persistent or non-persistent connections.
  - User-agent: Browser
  - Accept-language: Language (of the object)

HTTP Request/Response Message

- Entity Body
  - HTTP Request - Specific keywords supplied by the user to get the specific Web page.
  - HTTP Response - Message body.

HTTP Request Message - Example

- Request Line

  GET /somedir/page.html HTTP/1.1

- Header Lines

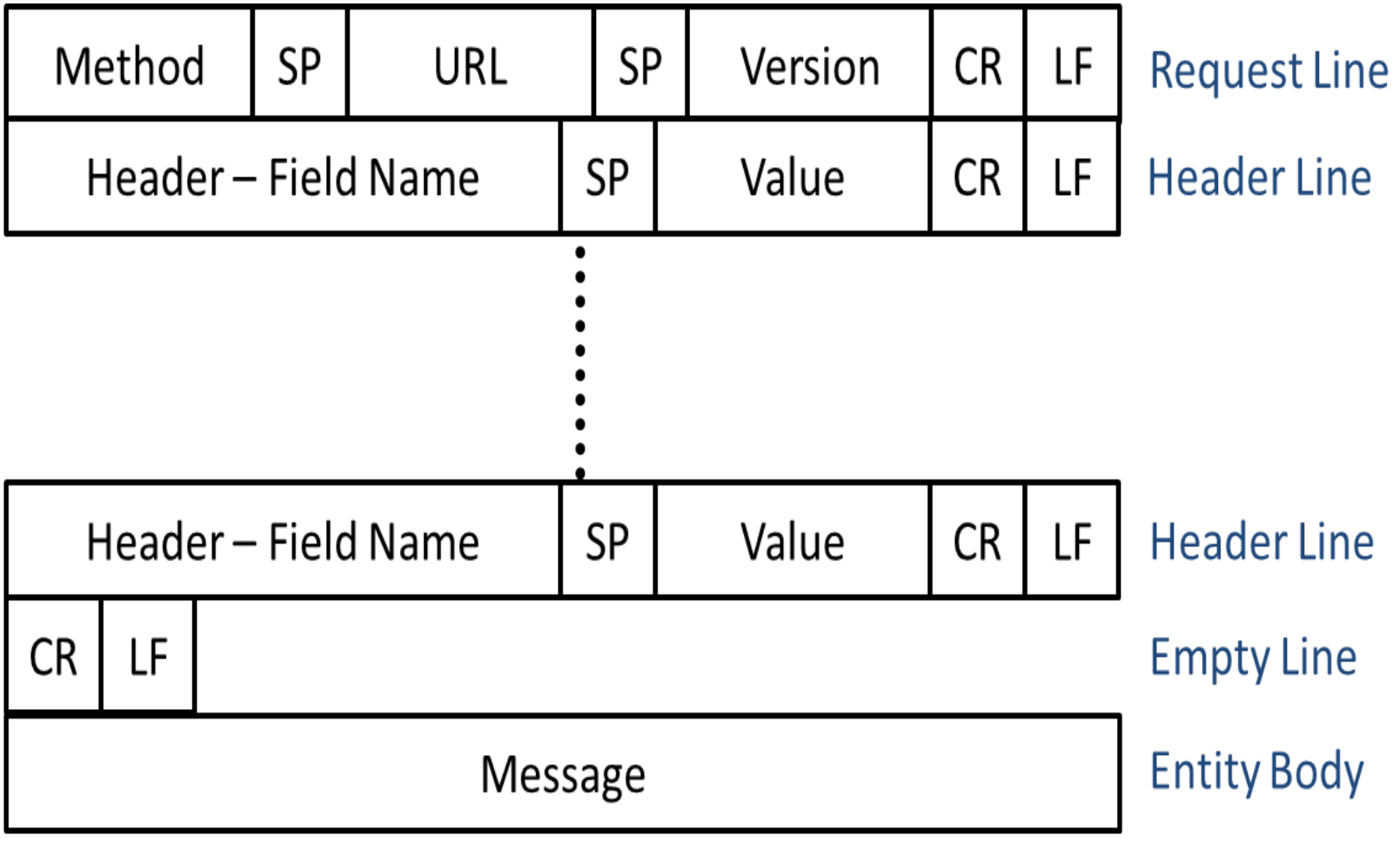
  Host: www.someschool.edu

  Connection: close

  User-agent: Mozilla/5.0

  Accept-language: fr

HTTP Request Message Format



HTTP Response Message

- Status Line
  - Protocol version
  - Status code
  - Status message

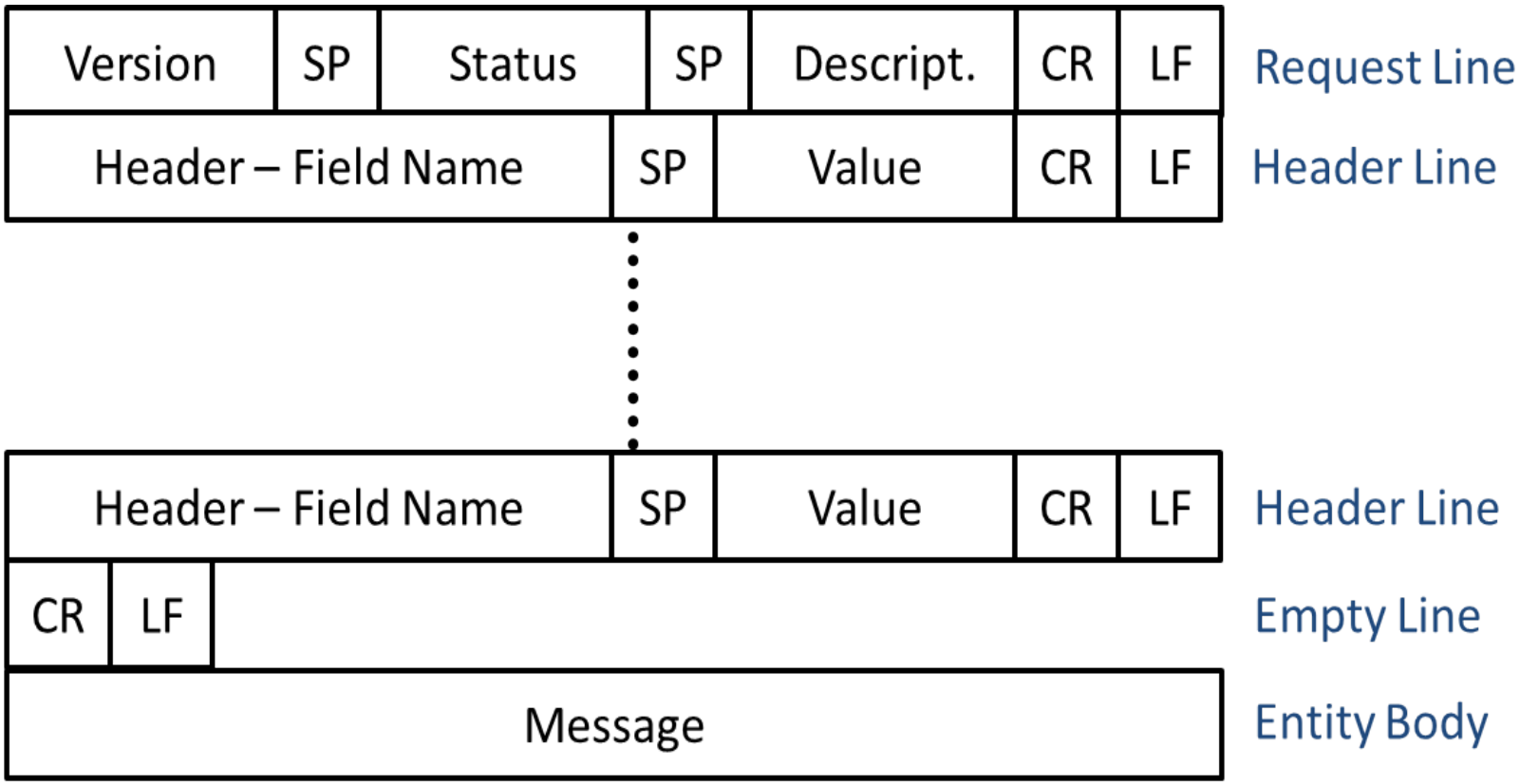
HTTP Response Message

- Common Status Codes
  - 200 - OK
  - 301 - Moved
  - 400 - Bad Request (not understood by the server)
  - 404 - Not Found
  - 505 - HTTP Version Not Supported

HTTP Response Message

- Header Lines
  - Date - Time when the server retrieves the object
  - Last-Modified - Date and time when the object was created or last modified.
  - Number of bytes

HTTP Response Message - Format



HTTP Response Message - Example

- Status Line

  HTTP/1.1 200 OK

- Header Lines

  Connection: close

  Date: Tue, 18 Aug 2015 15:44:04 GMT

  Server: Apache/2.2.3 (CentOS)

  Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT

  Content-Length: 6821

  Content-Type: text/html

- Message Body

Cookies

Cookies

- Goal: To keep track of users (used by Web pages).
- Cookie - A unique number used by the server to identify the client.
- Create a user session layer on top of stateless HTTP.

Cookies

- Header line in the HTTP response message
- Header line in the HTTP request message
- File kept on the user's end system and managed by the user's browser
- Back-end database at the Web site

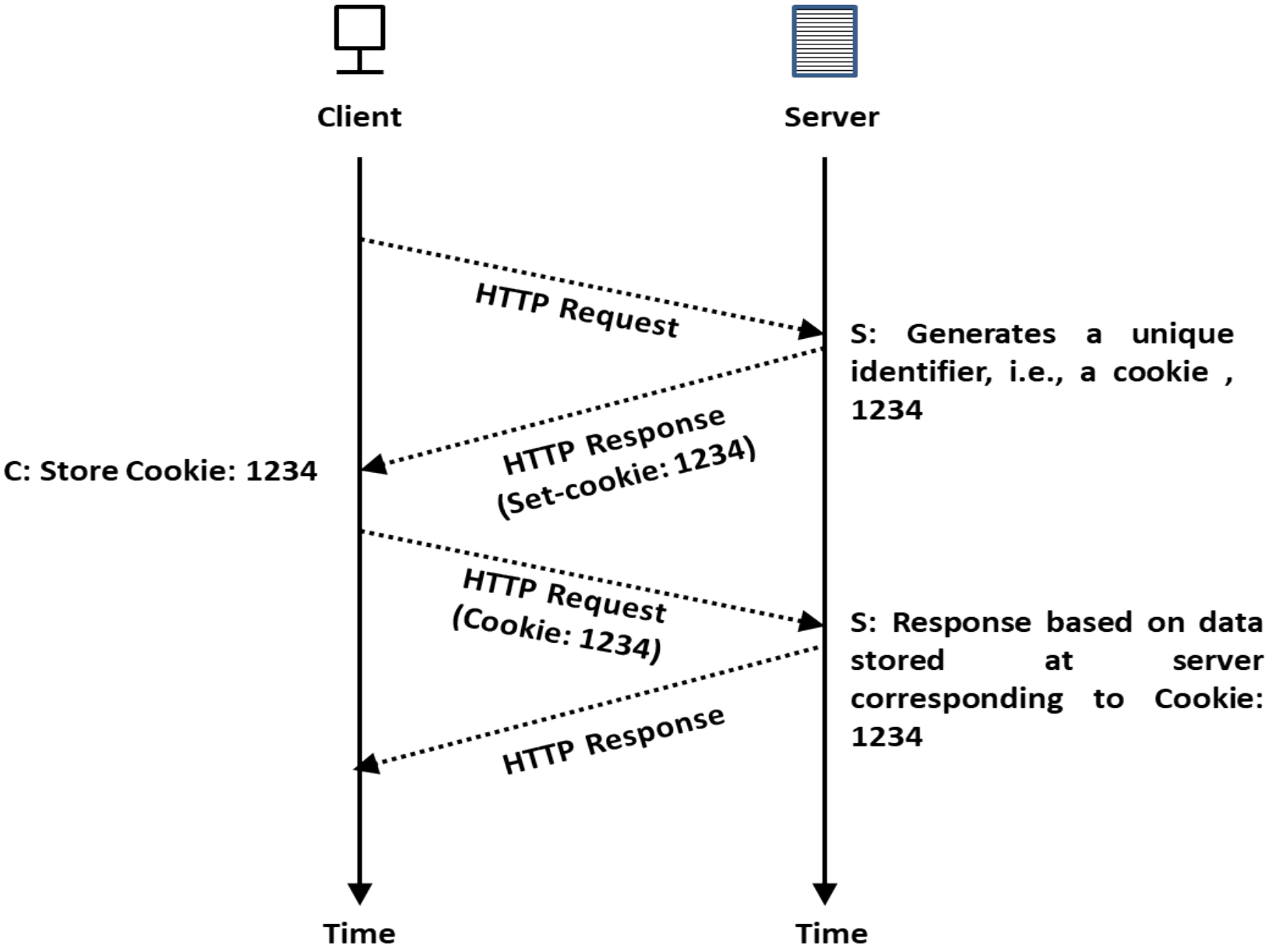
How Cookies Work

- Client sends the request to server.
- Server creates unique identification number and creates an entry in its database.
- Server responds to the user
    - Set-cookie: Identification number

How Cookies Work

- Client appends the line to the cookie file.
    - Set-cookie: Host name and identification number
- When the client visits the same server/Web page again, it includes the identification number in the HTTP request message.
    - Cookie: Identification number

Cookies


Client
Server
HTTP Request
S: Generates a unique identifier, i.e., a cookie , 1234
HTTP Response
(Set-cookie: 1234)
C: Store Cookie: 1234
HTTP Request
(Cookie: 1234)
S: Response based on data stored at server corresponding to Cookie: 1234
HTTP Response
Time
Time

Cookies

- Advantage
    - User specific products recommendation
- Disadvantage
    - Privacy breach

Web Cache (or Proxy Server)

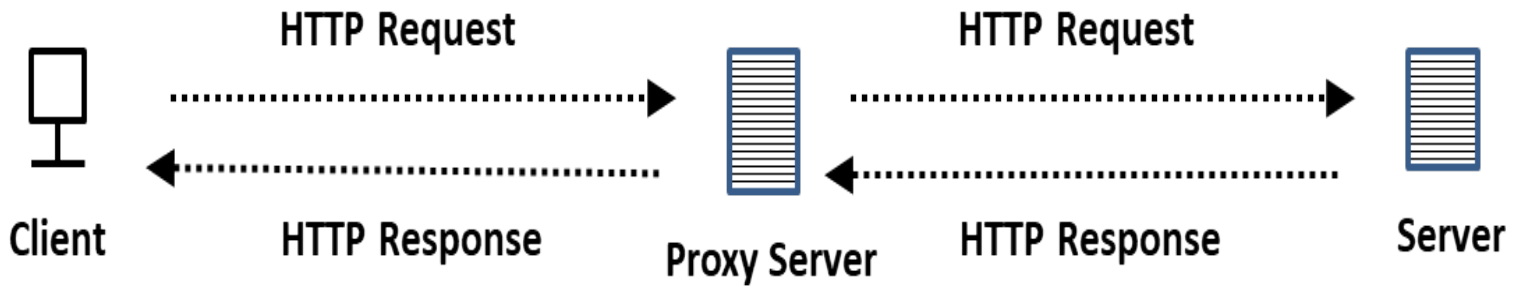
Web Cache (or Proxy Server)

A server that responds to HTTP requests on behalf of the actual server.

How Web Cache Works

- Web cache – Works as the client and server.
- Client requests an object.
- Web cache returns the copy of object, it available.
- If web cache does not have a copy of the object,
    - Web cache initiates a TCP connection with the actual server, and obtains the copy.
    - Web cache stores the copy in its database.
    - Web cache returns the object to the client.

Web Cache


HTTP Request
HTTP Request
Client
HTTP Response
Proxy Server
HTTP Response
Server

Web Cache

- Advantages
  - Reduce response time
  - Reduce traffic and cost
  - To reduce the load on server
- Disadvantage
  - Object may be stale (not the most recent one)

Conditional GET

- Conditional GET – HTTP request message used by the web cache to verify that its objects are up to date.
- HTTP Request Message - Conditional GET
  - Request message: GET method
  - If-Modified-Since: Last-Modified (Date and Time stored by the cache related to the object)
- Server – Forwards the object in the response only if there are any changes in the object. Otherwise it forwards a response without the object.
- The response message has a status code 304 - Not Modified.

Persistent TCP Connections - Head of Line (HoL) Blocking Problem

- Head of Line (HoL) Blocking Problem

  A large object occupies (blocks) the communication channel and it delays the communication of smaller objects.

Persistent TCP Connections - Head of Line (HoL) Blocking Problem

- Example
  - Suppose a Web page consists of a large video clip and few small objects.
  - In a persistent connection, if the server sends the video clip first, it will delay smaller objects.

Persistent TCP Connections - Head of Line (HoL) Blocking Problem

- HTTP/1.1
  - Parallel TCP connections to mitigate HoL blocking problem

HTTP/2

Can we reduce the number of parallel connections and at the same time solve the HoL problem?

Why do we want to reduce the number of parallel connections?

To reduce number of sockets at the server (to improve memory)

Persistent TCP Connections - Head of Line (HoL) Blocking Problem

- HTTP/2 Framing
  - Goal: To reduce the number of parallel TCP connections while returning a single Web page.

HTTP/2 Framing

- Break the response messages into frames.
- Interleave response messages.

HTTP/2 Framing - Example

- Assume a large video clip of 100 frames, and 10 smaller objects of 1 frame each.
- Server forwards 1 frame of a video clip and all 10 objects. Then, it forwards the second frame of a video clip.

HTTP/2 - Response Message Prioritization

- Client prioritizes the request message by assigning a weight between 1 (Priority: Low) and 256 (Priority: High).

HTTP/2 - Server Pushing

- Server can automatically send objects that are referenced by the requested page of the client (without an explicit request form the client).
- Advantage - Reduced delay.

Electronic Mail Application and SMTP

Electronic Mail

- Email - Asynchronous communication medium
- Advantages (over postal mail)
  - Fast
  - Easy to distribute
  - Inexpensive
- Other Advantages
  - Messages with attachments
  - Messages with hyperlinks
  - HTML formatted text
  - Embedded photos

Mail System – Components

- User agents – Allow users to read, compose, reply, forward, and save messages. Example: Gmail
- Mail servers
- Simple Mail Transfer Protocol (SMTP)

Mail System – Components

- Each user has a mailbox in one of the mail servers. Mailbox is used to manage and maintain the messages of a user.

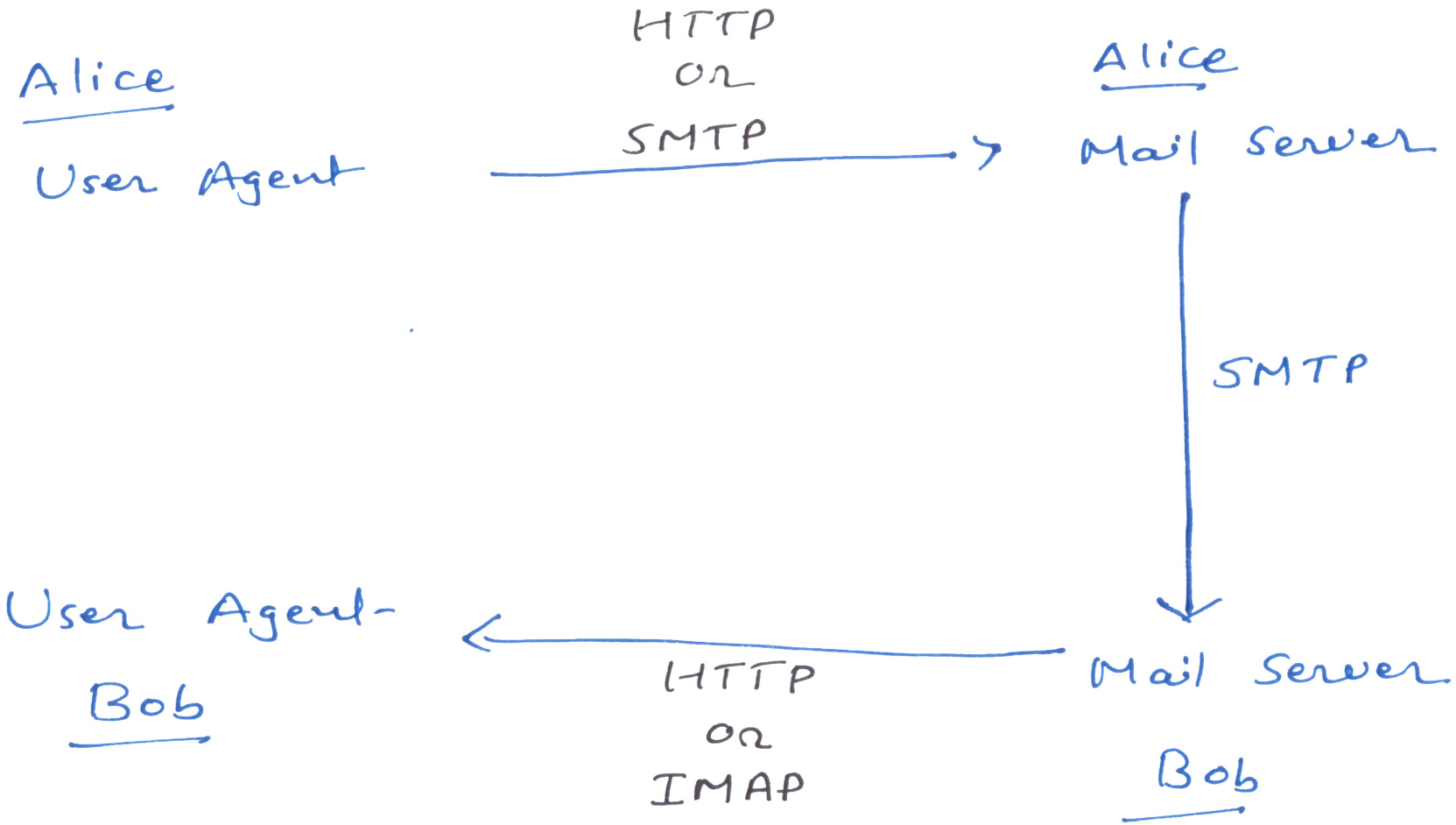
SMTP - Application-layer Protocol for Email

- RFC 5321
- Uses TCP for reliable data transfer.
- Uses port number 25.
- SMTP uses persistent connections (one TCP connection for all messages between client and server in a session).
- Mail Servers – Runs SMTP client and SMTP server
- Limitation - SMTP – Message body - 7-bits only.

How SMTP Works

1. Sender uses User Agent to send the message to the Mail Server of sender.
2. Sender's Mail Server runs SMTP client program which initiates TCP connection with SMTP server program on recipient's Mail Server.
3. After connection establishment, SMTP client program sends the sender's message to the SMTP server program on recipient's Mail Server.
4. SMTP server program places the message in recipient's mailbox.
5. Recipient can read the message using User Agent.

Email - From Alice to Bob



SMTP – Messages

- The SMTP client sends five messages.
  - HELO
  - MAIL FROM: Sender's Email Address
  - RCPT TO: Receiver's Email Address
  - DATA
  - QUIT
- To convey the end of the message, Client sends CR LF . CR LF

Mail Message Formats

- Message Format
    - Keyword: Value
- Header Lines - Mandatory
    - From: Sender's Email Address
    - To: Receiver's Email Address
- Header Line - Optional
    - Subject: Message Header

SMTP Messages

- SMTP commands and header lines both have keywords such as from and to. However, they are different.
- These keywords are a part of SMTP messages, while they are also a part of SMTP handshaking protocol.

Domain name system (DNS)

DNS – The Internet's Directory Service

- Analogy: Identifying Human
  - Name
  - Pancard Number
  - Aadhar Number

How Do We Identify the Hosts

- Hostname – Preferred by users and variable size
    - Example: www.google.com
- IP address – Preferred by devices and fixed size (4 bytes)
    - Example: 142.250.217.100

Domain Name System (DNS)

- Goal:
  - To translate hostnames to IP addresses
  - Host Aliasing – When a complicated hostname has one or more aliases, we can use DNS to find the canonical hostname from the user supplied hostname (Alias).
  - Mail server Aliasing
  - Load distribution – Hostname (Alias) associated with multiple IP addresses (of replicated servers). DNS translates to different IP address each time to balance the load.

Domain Name System (DNS)

- DNS
  - Consists of hierarchy of distributed DNS servers
  - DNS protocol

Domain Name System (DNS) Protocol

- DNS protocol
  - RFC 1034 and 1035
  - Uses User Datagram Protocol (UDP)
  - DNS query and reply messages use Port number 53
  - Application layer protocols such as HTTP and SMTP use DNS to translate hostnames to IP addresses.

How DNS Works

- Host extracts hostname from the URL.
- Host uses the DNS client process to send the request message to DNS server.
- DNS server processes the request and sends the corresponding IP address back to the Host.

What will be the issues if we have only a single DNS server?

DNS Server

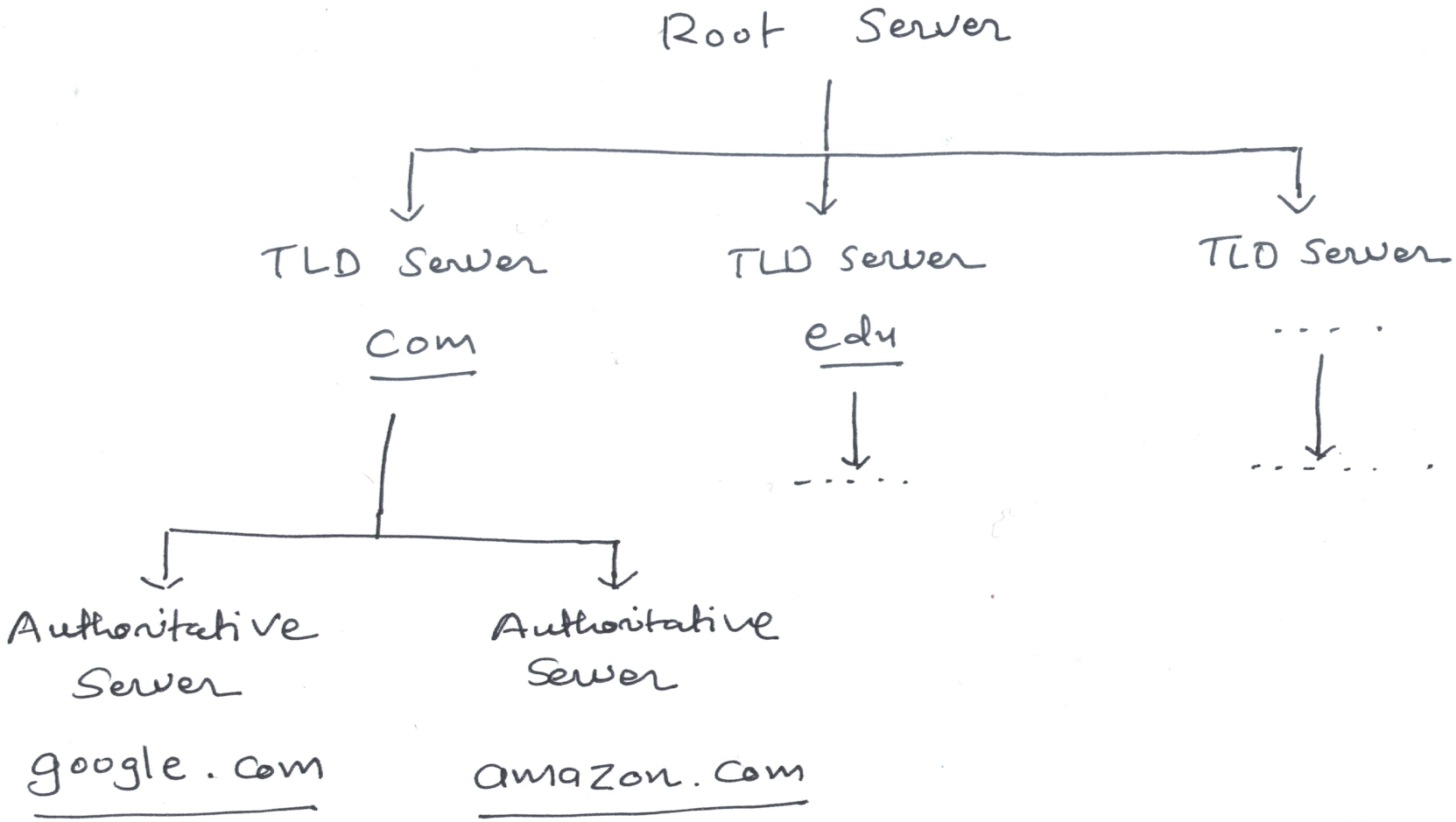
- Single DNS Server – Issues
  - Single point of failure
  - Traffic
  - Distance (end-to-end delay)
  - Maintenance (Large database needs to be Updated frequently)
- Architecture with a single DNS server is not scalable.

How do we resolve the scalability issues of a single DNS?

DNS - Distributed and Hierarchical Database

- DNS
  - Consists of large number of servers
  - Distributed across the globe
  - Organized in hierarchical way

Hierarchy of DNS Servers



How do we find the IP address of the DNS servers?

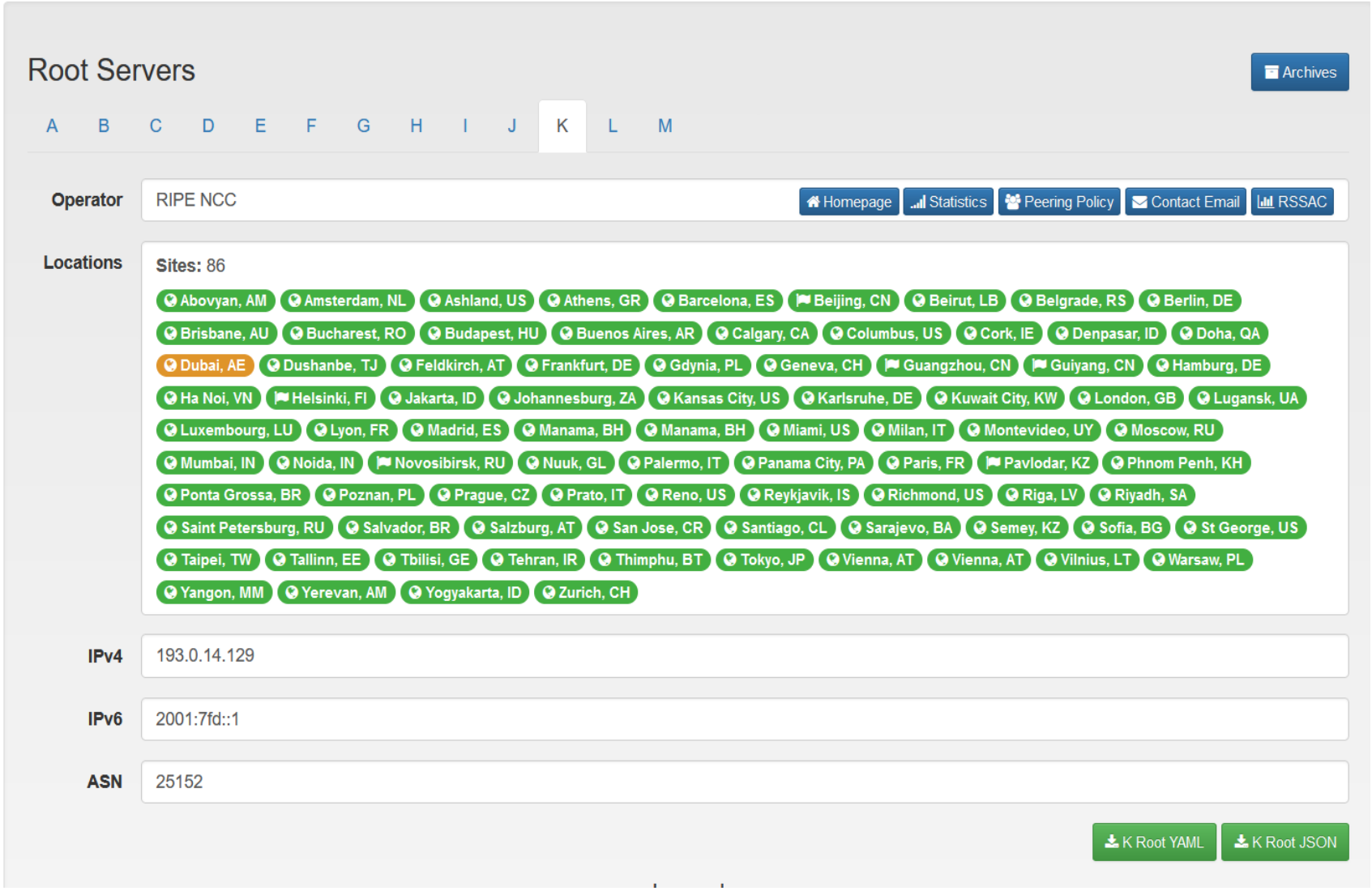
Types of DNS Servers

- Root Servers – Provides information about Top-level Domain servers.
- Top-level Domain (TLD) Servers – Provides information about Authoritative Servers.
- Authoritative Servers – Translate Hostname to IP address.

Types of DNS Servers

- Root Servers
  - Provide IP addresses of TLD servers
  - Total: 13 different servers
  - Root Servers in 1000+ locations
  - Reference: `www.root-servers.org`

DNS - Root Servers



Types of DNS Servers

- Top-level Domain (TLD) Servers
  - Provide IP addresses of authoritative servers.
  - Top-level Domains – `com`, `org`, `net`, `edu`, `gov`, `in`, `fr`, etc.

Types of DNS Servers

- Authoritative Servers
  - Provide IP addresses of Hosts (Web servers, Mail Servers, etc.)

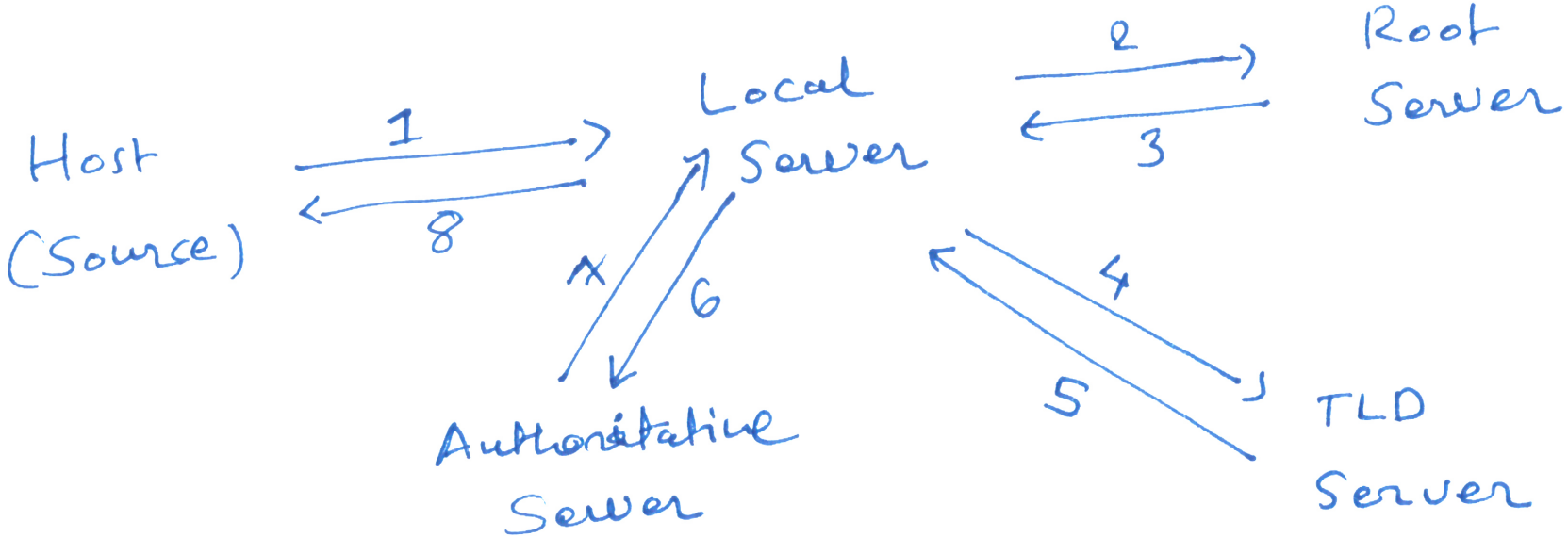
Local DNS Server

- Local DNS Server
  - ISP maintains a local DNS server.
  - Works as a proxy server.

How DNS Query Mechanism Works

- Host sends a DNS query to a local server.
- Local server sends a DNS query (if new) to root server.
- Root server sends IP address of the TLD server.
- Local server sends a DNS query to the TLD server.
- TLD server sends IP address of authoritative server.
- Local server sends a DNS query to authoritative server.
- Authoritative server sends IP address of the destination host.
- Local DNS server forwards IP address to the host (source).

DNS Query


Host
(Source)
1
8
Local
Server
2
3
Root
Server
7
6
4
5
Authoritative
Server
TLD
Server

DNS Caching

- DNS Cache
  - Reduce delay
  - Reduce DNS traffic
- DNS Server – Store the Hostname-IP address information in the memory to serve future DNS queries.
- DNS servers periodically discard the cached information periodically if they are not authoritative server for the hostname.

DNS Resource Records

- DNS Resource Record consists of
  - Name
  - Value
  - Type
  - TTL
- TTL
  - Time after which the record should be removed from the cache.

DNS Resource Records

- Type – A
  - Name – Hostname
  - Value – IP address

DNS Resource Records

- Type – NS
  - Name – Domain name (gmail.com)
  - Value – Hostname of the authoritative server that contains IP addresses for hosts in the domain.

DNS Resource Records

- Type – CNAME
  - Name – Hostname (Alias)
  - Value – Canonical Hostname

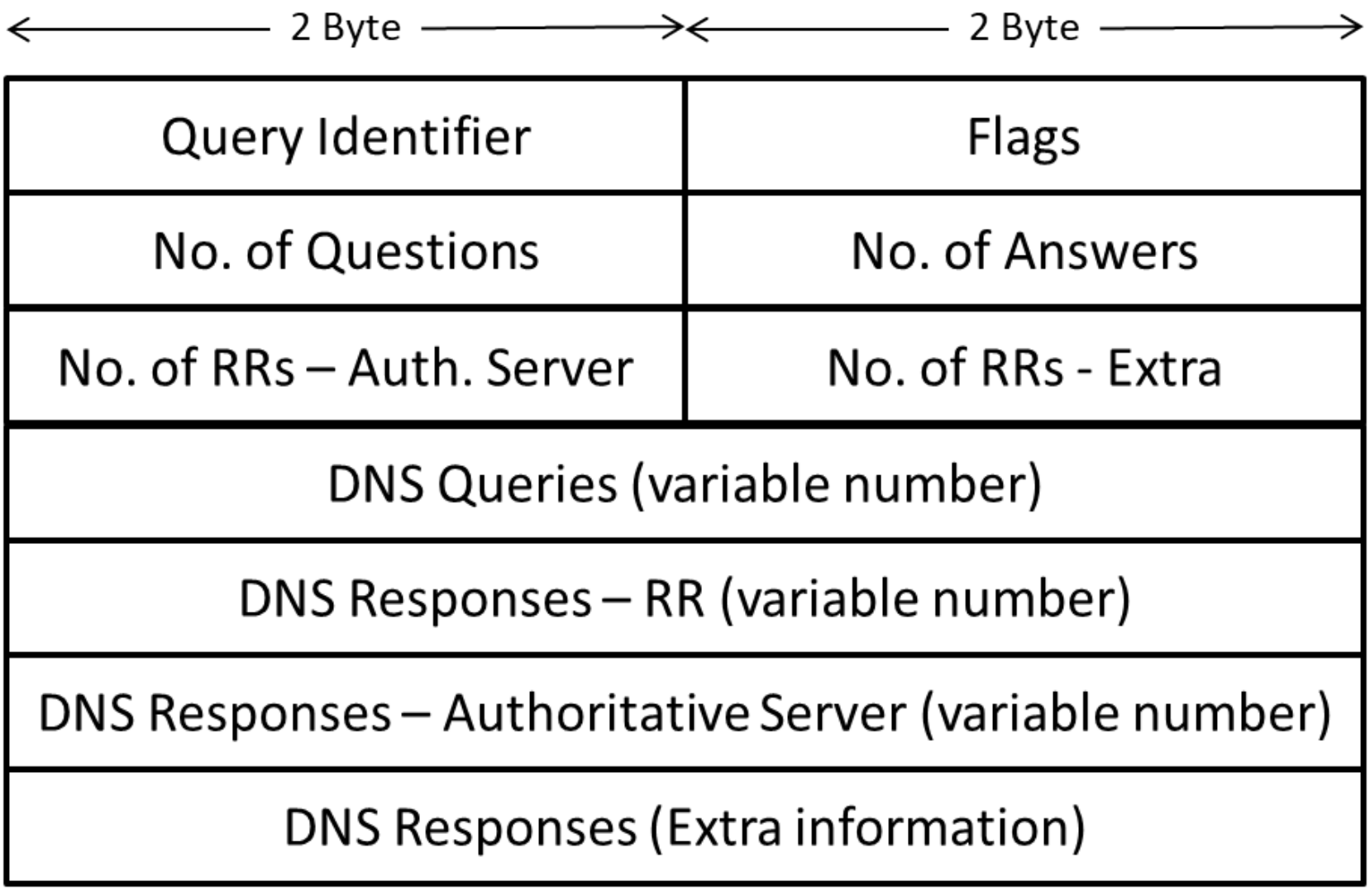
DNS Resource Records

- Type – MX
  - Name – Hostname (Mail server alias)
  - Value – Canonical Hostname (Mail server)

DNS Messages

- Two types of messages
  - Query message
  - Reply message
- Query and reply messages have the same format.

DNS Message Format



DNS Message Format

- 16-bit Number – To identify the DNS query Flags
  - 1-bit – To identify the type of message (Query (0)/Reply (1))
  - 1-bit – To identify authoritative server

DNS Message Format

- Questions – Query
  - Hostname
  - Type – A, NS, CNAME, MX

DNS Message Format

- Answers – Reply
  - DNS Resource Record (Name, Value, Type, TTL)
  - If a hostname is associated with multiple IP addresses, there are multiple resource records in the answer.

DNS Message Format

- Additional Information
  - If the query is to find the canonical hostname corresponding to alias, the additional information section contains the IP address of the corresponding server.

How to Send RR to DNS Server

- How to send resource records to DNS servers?
  - At the time of registering a domain name, the uniqueness of domain name is verified and will be inserted into the DNS database.

Domain Name Registration

- Information about domain name registrars: http://www.internic.net/
- While registering a domain name, we also have to provide the information about IP addresses of primary and secondary authoritative servers.
- The registrar will send the details of authoritative servers to TLD servers.

DNS Vulnerabilities

- Is it possible to attack DNS servers?
- Attacks
  - DDoS Attack on Root or TLD servers
  - Man-in-the-middle attack
  - DNS Poisoning Attack – Sending fake resource records to cache.
- DNS Security Extensions (DNSSEC – RFC 4033) - To protect against such attacks

Peer-to-Peer File Distribution

Peer-to-Peer File Distribution

- Hosts (or Peers)
  - Send data to other peers directly
  - Receive data from other peers directly
  - Not always-on/running
- P2P Architecture – Self Scalability

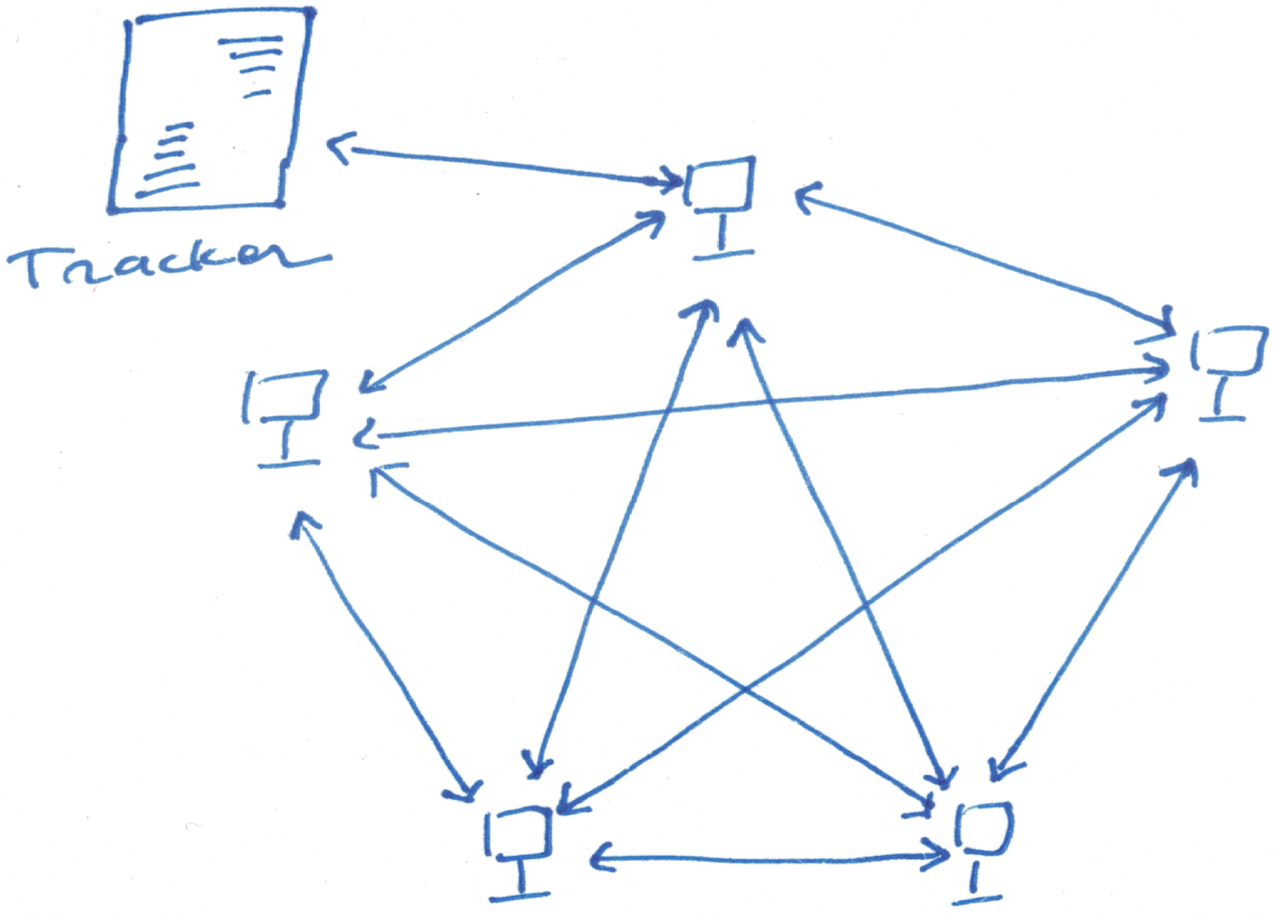
Homework

- Use the quantitative model to analyse the distribution time in client-server architecture and peer-to-peer architecture. [Kurose and Ross Section 2.5]

BitTorrent Protocol

- BitTorrent Protocol – Peer to Peer (P2P) File Distribution Protocol
- Peers
  - Sometimes known as torrents
  - Download/upload chunks of the file from/to other peers

Peer to Peer File Transfer



BitTorrent Protocol - Tracker

- Tracker
  - Keeps track of the peers (IP addresses)
  - Share the IP addresses of other peers with a newly joined peer.

- A peer can establish concurrent TCP connections with other peers.
- The list of peers is constantly changing. When the peers leave, or when new peers send TCP connection requests, the list of peers of a peer changes.

How does a peer decide which chunks to download/upload from/to other peers?

Rarest First Technique

- A peer requests a rarest chunk first. In this way, a rarest chunk will be quickly distributed among peers.

Which peers a peer is going to be connected to?

- Which peers a peer is going to be connected to? Follow tit-for-tat (incentive) mechanism.
  - Peers that provide chunks of a file at the highest rate.
  - Periodically measure the rate and decide list of peers.
  - Randomly select a peer to be connected to. (The randomly selected peer may also send the chunks of a file to this peer at the highest rate if this peer is its one of the highest rate peers.) It also helps a new peer to get chunks of file before it can start sharing the same.

Video Streaming and DASH

What is video streaming?

Playing a partially retrieved video while the remaining video is downloaded.

Video Streaming - Advantages

- Starts early, i.e, instead of waiting for the video to be downloaded fully.
- Saves bandwidth, i.e., if we don't like the video, we can stop the remaining download.

Video Streaming - Challenges

- Scalability, i.e., millions of active users
- Heterogeneity, i.e, different devices and different bandwidths

Video Streaming and Content Distribution Networks

- Pre-recorded videos provided by servers
- Example: Amazon Prime, Netflix, Youtube

What is a video?

- A sequence of images/frames displayed at a constant rate. For example, 24 images/frames per second.

Video Streaming - Performance

- Performance Measures
  - Quality (High/Low)
  - End-to-end throughput
- Video
  - High Quality – Large number of bits
  - Low Quality – Fewer bits
- Service providers create different versions of the video (of different quality).

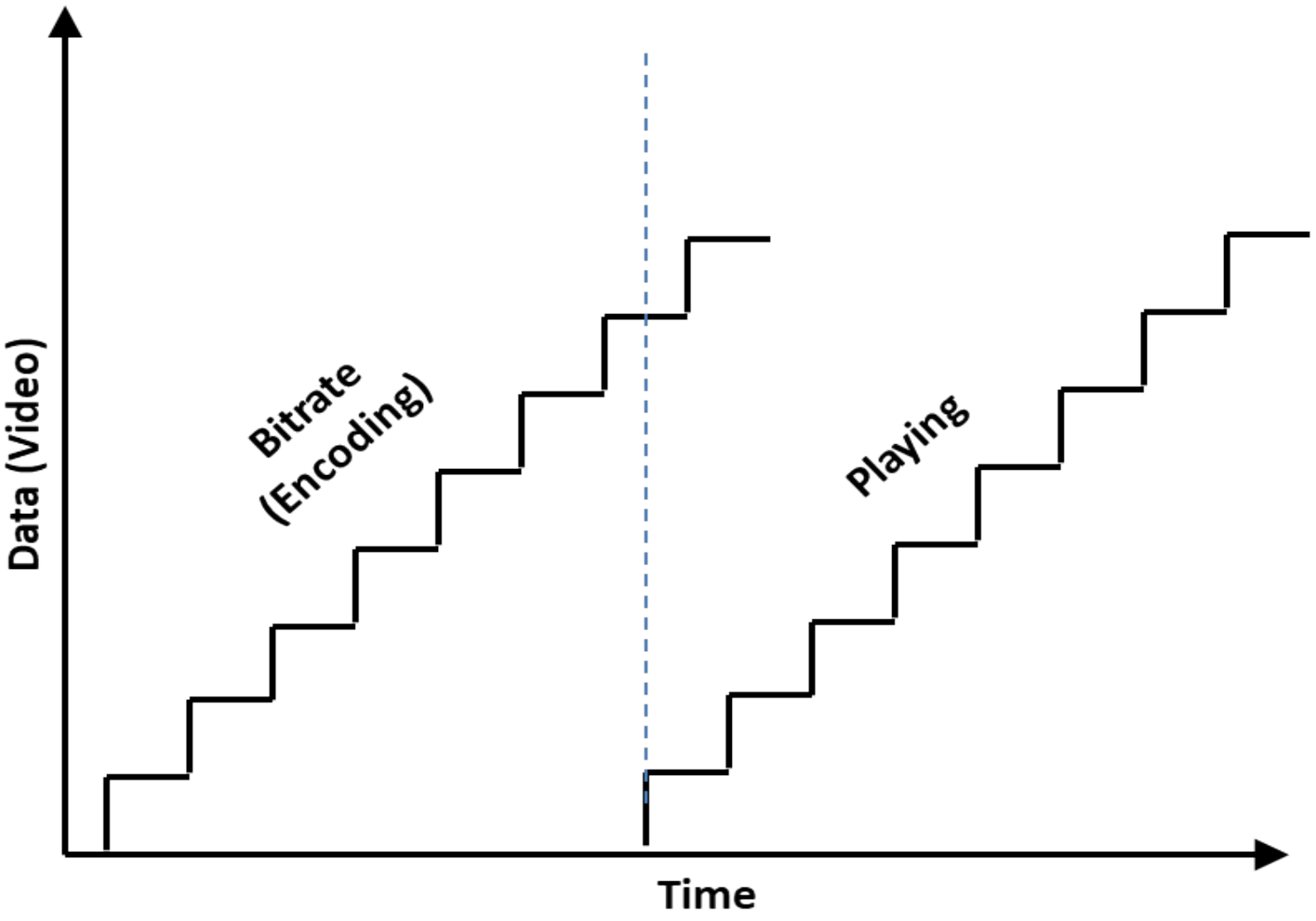
Video Encoding

- Encoding a video - Use redundancy within and between frames to reduce the number of bits required to be communicated.
    - Spatial - Reducing/Compressing the bits by comparing the pixels of same frame.
    - Temporal - Reducing/Compressing the bits by comparing different/successive frames.

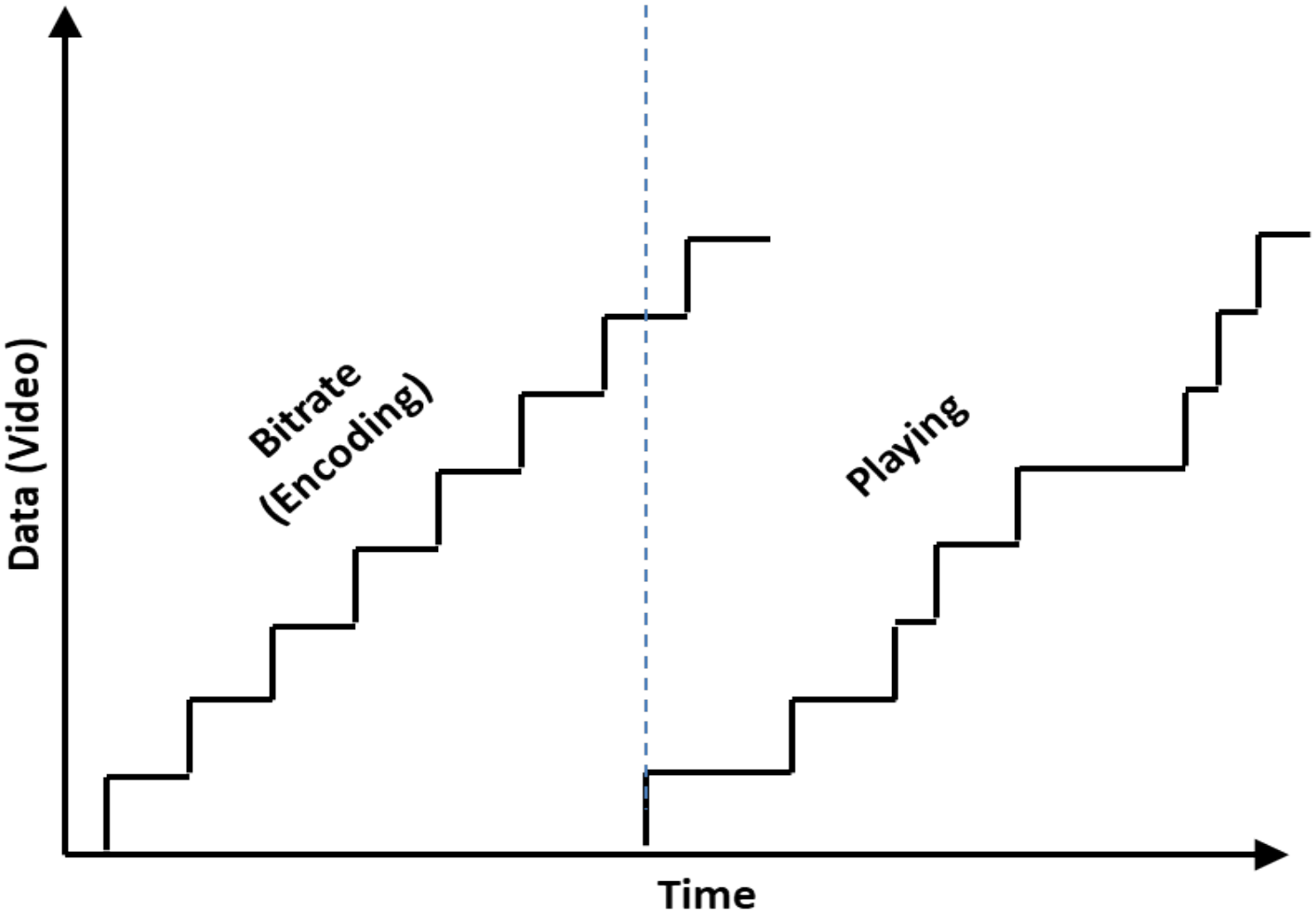
Video Encoding

- Encoding a video
  - Constant bitrate (CBR) - Encoding rate remains constant.
  - Variable bitrate (VBR) - Encoding rate is variable.

Video Streaming - Constant Network Delay



Video Streaming - Variable Network Delay



How can we reduce the impact of variable network delay?

Video Streaming - Variable Network Delay

- Buffering

HTTP Streaming

- HTTP Streaming Example: Youtube
- At server, each video is having a specific URL.

HTTP Streaming

- How to get the video stored at the server?
    - Initiate TCP connection with the server.
    - Use HTTP protocol to get the copy of the video.

How Video Streaming Applications Works

- Video Streaming Application
  - Get the frames from client buffer
  - Decompress the frames
  - Display on user's screen

Video Streaming using HTTP - Disadvantage

- Irrespective of the amount of bandwidth available to clients, HTTP sends the same size videos.

How can we support users having different bandwidths?

Dynamic Adaptive Streaming over HTTP (DASH)

- Goal: To do video streaming considering the available bandwidth to users.
- Example: Netflix

How DASH Works

- Divide the video into chunks.
- Encode chunks at different bitrates and store them with different URLs.
- Server maintains a "manifest file" – contains information about URL and corresponding bit rate
- Client requests manifest file form the server.
- To select a next chunk from the specific URL, a client uses the rate determination algorithm and information about available bandwidth.
- If the number of chunks in the client's buffer are more, client requests chunks having higher bit rate (and higher quality), and vice versa.

How to store the recorded videos for streaming?

How to Store Recorded Videos for Streaming

- Stream videos from a large data center
  - End-to-end throughput (considering the bottleneck communication link)
  - Few communication links may pass the same video many times. It wastes bandwidth and costly (financially).
  - Single point of failure

Content Distribution Networks

- How to store the recorded videos for streaming?
    - Content Distribution Networks (CDNs) - Multiple servers at different geographic locations.

OTT Platforms

- OTT - “Over The Top” (of existing infrastructure)
  - A specific application related service on top of the services provided by the Internet.

Homework

- Read - Case Studies: Netflix and YouTube [Kurose and Ross Section 2.6.4]
- Watch Following Videos.
  - Designing Netflix's Content Delivery System
  - Scaling the Netflix Global CDN

Socket Programming

Socket Programming - Creating Network Applications

- Network Application (client-server architecture)
  - Client program
  - Server program
- Client and server programs reside in different end systems.

- Types of Network Applications
  - Open Application – Protocol is well known such as the one defined in RFC. Application developer must follow the guidelines of the protocol for such applications. Use standard port numbers. (Example: Browser and Server)
  - Proprietary Application - Protocol is not openly published. Use port numbers (other than the standard port numbers).

- Which transport layer protocol is to be used for the application?
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)

Socket Programming with UDP

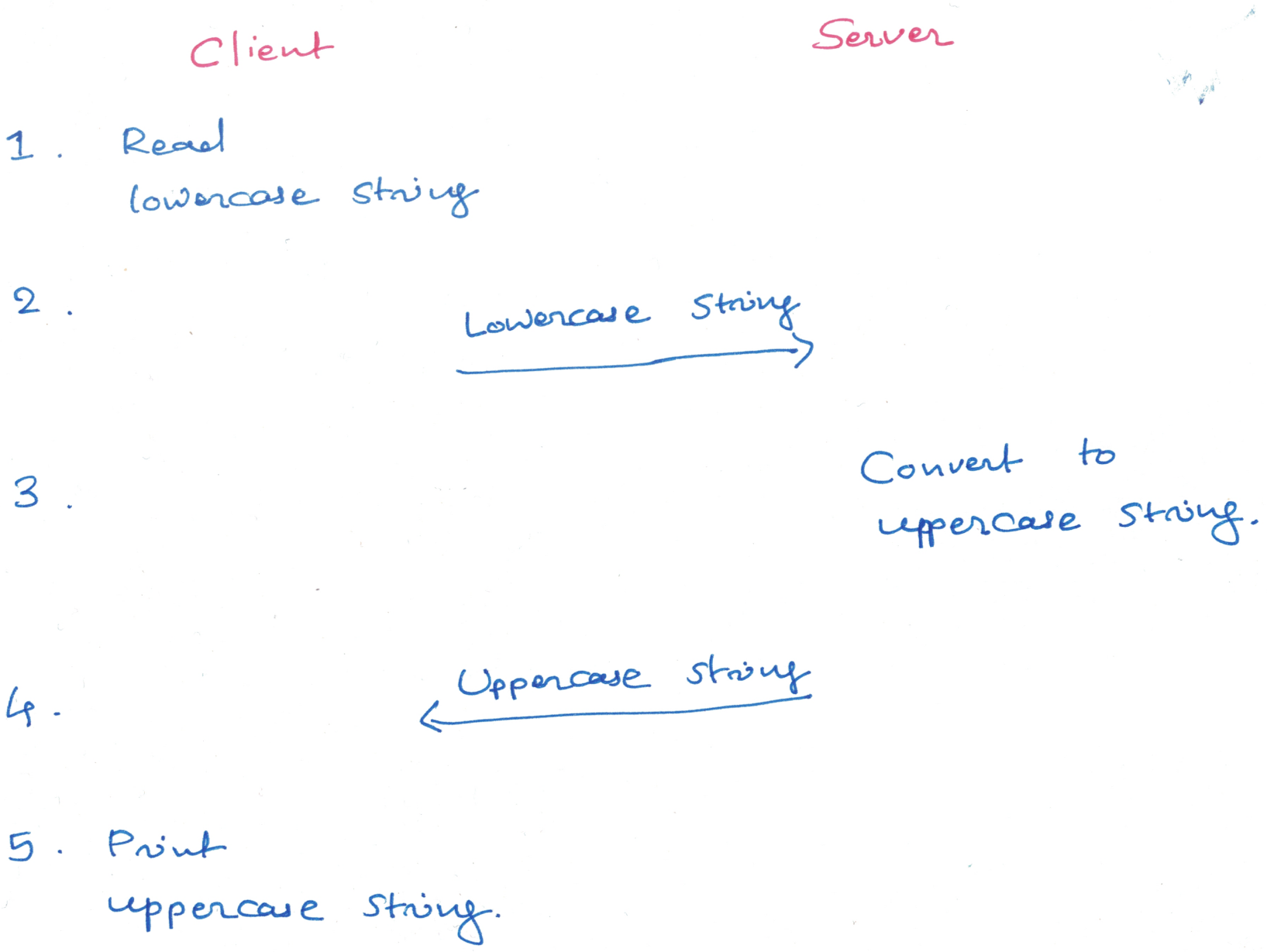
- While using UDP, first attach a destination address to the packet.

  The destination address consists of

  - IP address (Destination) – To identify the host
  - Port Number (Destination) – To identify the socket/process.

- Attach source address (usually operating system attaches source address (IP address and port number).

Example - Client-Server Application



UDP Client (UDPClient.py)

```
1  from socket import *
2  serverName = ‘hostname‘
3  serverPort = 12000 # Randomly chosen
4  clientSocket = socket(AF_INET, SOCK_DGRAM
       )
5  message = input(‘Input lowercase sentence
       :‘)
6  clientSocket.sendto(message.encode(),(
       serverName, serverPort))
7  modifiedMessage, serverAddress =
       clientSocket.recvfrom(2048)
8  print(modifiedMessage.decode())
9  clientSocket.close()
```

UDP Client (`UDPClient.py`)

- `socket` - module for network communications in Python.

  serverName = 'hostname'
  serverPort = 12000

- Above lines initialize variables with the hostname (or IP address) and port number of the server.

  clientSocket = socket(AF_INET, SOCK_DGRAM)

- `socket()` – creates a socket. Argument `AF_INET` is used to represent the underlying network, IPv4. `SOCK_DGRAM` represents a UDP socket.

- Port number of the client socket will be chosen by the operating system.

UDP Client (`UDPClient.py`)

message = input('Input lowercase sentence:')

- `input()` - To accept the input message.

  clientSocket.sendto(message.encode(), (serverName, serverPort))

- `sendto()` – To send the packet from the client to the server.
- `message.encode()` – To covert the message from string to byte.
- Server address – (serverName, serverPort)
- clientPort and clientName is automatically attached to the packet by operating system.

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

- `recvfrom()` – To receive a packet at the specific IP address and port number as mentioned in the packet.
- modifiedMessage – Variable is initialized with the value received from the server.
- serverAddress – Variable is initialized with the address of the server (containing both IP address and port number).
- 2048 – Buffer size

print(modifiedMessage.decode())

- `decode()` – To convert the message from bytes to string.

clientSocket.close()

- `close()` – To close the socket.

UDPServer.py

```
1  from socket import *
2  serverPort = 12000
3  serverSocket = socket(AF_INET,
       SOCK_DGRAM)
4  serverSocket.bind((‘‘, serverPort))
5  print(‘‘The server is ready to receive
       ‘‘)
6  while True:
7      message, clientAddress =
           serverSocket.recvfrom(2048)
8      modifiedMessage = message.decode().
           upper()
9      serverSocket.sendto(modifiedMessage
           .encode(), clientAddress)
```

UDP Server Program

- To initialize a variable serverPort
- serverPort = 12000
- To bind the newly created socket with the designated port number
- serverSocket.bind((", serverPort))
- The above line binds (that is, assigns) the port number 12000 to the server's socket.
- Any packets destined for port number, serverPort, will be received by this socket.
- To execute the server process indefinitely, use the loop.

UDP Server Program

- modifiedMessage = message.decode().upper()
- `decode()` – To decode a message from bytes to string.
- `upper()` – To convert the message to uppercase letters.
- serverSocket.sendto(modifiedMessage.encode(), clientAddress)
- The packet also includes source (server's) address. However, it will be attached automatically.
- Server process continues to run and waits for the next packet.

Socket Programming with TCP

Socket Programming with TCP

- TCP
  - Connection oriented protocol
  - Requires the client and server to establish a connection before they can send and receive packets.
  - Once the TCP connection is established between the client socket and server socket, they can send and receive packets without attaching address information to the packets.

- A server process creates two sockets.
  - First socket – To accept the initial packet from a new client. In other words, it is used to establish a TCP connection between the client and server. It is available for connection indefinitely.
  - Second socket – To send and receive packets once the connection is established. (different for each TCP connection). The socket will be closed after the data exchange ends.

TCPClient.py

```
1  from socket import *
2  serverName = 'servername'
3  serverPort = 12000
4  clientSocket = socket(AF_INET,
       SOCK_STREAM)
5  clientSocket.connect((serverName,
       serverPort))
6  sentence = input('Input lowercase
       sentence')
7  clientSocket.send(sentence.encode())
8  modifiedSentence = clientSocket.recv
       (1024)
9  print('From Server: ', modifiedSentence.
       decode())
```

TCPClient.py

```
10      clientSocket.close()
```

TCP Client Program

- SOCK_STREAM – To represent a TCP socket.
- `connect()` – To establish the TCP connection with a specific socket.
- clientSocket.connect((serverName,serverPort))
- `send()` – To send the data from one socket to the another socket. Here, two sockets are connected by a TCP connection.
- clientSocket.send(sentence.encode())
- modifiedSentence = clientSocket.recv(2048)

TCPServer.py

```
1       from socket import *
2       serverPort = 12000
3       serverSocket = socket(AF_INET,SOCK_STREAM
            )
4       serverSocket.bind((‘‘,serverPort))
5       serverSocket.listen(1)
6       print(‘The server is ready to receive‘)
7       while True:
8           connectionSocket, addr = serverSocket.
                accept()
9           sentence = connectionSocket.recv(1024)
                .decode()
10          capitalizedSentence = sentence.upper()
```

TCPServer.py

```
11          connectionSocket.send(
                capitalizedSentence.encode())
12          connectionSocket.close()
```

TCP Server Program

- `lisen()` – To listen for TCP connection requests form the client
- serverSocket.listen(1)
- 1 - Maximum number of queued connections (at least 1)
- connectionSocket, addr = serverSocket.accept()
- `accept()` – To create a new socket, connectionSocket, for a specific TCP connection.

Conclusions

Conclusions

- Applications - the only reason for which computer networks exist.
- Two architectures - Client-Server Architecture and Peer-to-Peer Architecture.
- App (Protocol) - Web (HTTP), Email (SMTP), DNS (DNS), File Transfer (BitTorrent), Video Streaming (DASH).
- How to send/receive data to/from applications running on different hosts? Sockets...!

Homework

Homework

- Use `Wireshark` to capture HTTP packets related to `GET`, `PUT`, `HEAD`, and `POST` methods of HTTP. Explain each of the methods with respect to `Wireshark` capture.
- Use `Telnet` to send HTTP request, HTTP response, and SMTP message.
- Check `nslookup` command.

References

References

- James Kurose and Keith Ross, Computer Networking: A Top-Down Approach - Textbook
- James Kurose, Computer Networking: A Top-Down Approach - Video Lectures
- Textbook Resources
- Internet Engineering Task Force

Thank You.