|  |
| --- |
| Y Media Labs |
| Training Assignment |
| - |

|  |
| --- |
| Ananthu P Kanive  8-9-2018 |

Table of Contents

Contents

[Client Server Model 2](#_Toc521544253)

[Servers, Services and Clients 2](#_Toc521544254)

[Types of Servers 2](#_Toc521544255)

[Architecture 2](#_Toc521544256)

[Example use cases 3](#_Toc521544257)

[Hyper Text Transfer Protocol 4](#_Toc521544258)

[Request – Response Model 4](#_Toc521544259)

[HTTP messages 4](#_Toc521544260)

[HTTP requests 4](#_Toc521544261)

[HTTP Response 5](#_Toc521544262)

[HTTP Methods 5](#_Toc521544263)

[Making HTTP requests 5](#_Toc521544264)

[Open APIs 5](#_Toc521544265)

[Sending requests through Postman 6](#_Toc521544266)

[Sending requests through curl 6](#_Toc521544267)

[TCP and UDP 7](#_Toc521544268)

[Transmission Control Protocol 7](#_Toc521544269)

[Applications of TCP: 7](#_Toc521544270)

[User Datagram Protocol 7](#_Toc521544271)

[Applications of UDP: 7](#_Toc521544272)

[Video Streaming – UDP or TCP? 8](#_Toc521544273)

[Hash Tables 9](#_Toc521544274)

[Hashing 9](#_Toc521544275)

[Operations involved in creating a hash table 9](#_Toc521544276)

[Operations visible to the user 9](#_Toc521544277)

# Client Server Model

This type of architecture is used for distributing the overall computation across multiple devices.

## Servers, Services and Clients

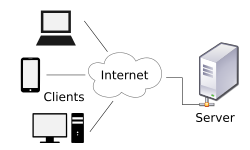
* A ***server*** is a program or a device that provides some functionality.
* The functionality is often called a ***service***.
* A ***client*** is another computer program that uses the functionality of a service.
* Technically, a server is a computer program running on some hardware on the network.
* This hardware is also referred to as servers as their sole purpose is to run these programs.
* A service is one of the functionality provided by the server program.
* A server usually provides multiple services to multiple clients, thus are usually computationally advanced compared to standard computers.
* Most servers run UNIX like operating systems, with windows coming second. Other OSes like the macOS are used in tiny numbers.

### Types of Servers

|  |  |
| --- | --- |
| **Server type** | **Purpose** |
| Application server | Hosts web apps accessible in a network. |
| Catalogue server | Maintains an index or a directory of various data. |
| Computing server | Provides computing resources like CPU and RAM, over a network. |
| Database server | Provides access to a database over a network. |
| File server | Serves files and folders across a network. |
| Game server | Provides multiple devices to play games over a network. |
| Mail server | Acts a receiver or sender of emails. |
| Media server | Shares video or audio over a network. |
| Print server | Shares one or more printers over a network, thus eliminating the hassle of physical access |
| Proxy server | Acts as in intermediate between a client and a server. |
| Web server | Hosts and serves web pages. |

## Architecture

* The client server model is a network architecture based on the producer-consumer principles.
* Clients request some information from a server available in the network.
* The Server produces this information which the Client consumes.
* This distributes the computational work over a network of servers.
* Works on a ***request-response*** mechanism where the client requests and the server responds.



## Example use cases

* Consider ***Wikipedia***, a digital encyclopaedia.
* The client will only need information about a tiny subset of topics.
* Storing all the data in all the clients is not only ineffective, but impractical.
* This can easily be solved by a network server. The server maintains the large database, and a service to query a page from the database.
* When the client requests for information, only the relevant info is sent. The client need not know where or how the data came to be.

# Hyper Text Transfer Protocol

* HTTP is a protocol that run on the application layer of the OSI layer model.
* It is the standard protocol for communication over the web, built primarily with TCP/IP.
* A ***Hypertext*** is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP deals with exchange of these elements.

## Request – Response Model

* HTTP runs on in conjunction with the *client – server model*.
* The client requests and server responses happen using HTTP request – response messages.
* Example – A web browser does a HTTP request to google, and in turn google servers send HTTP responses with some HTML data attached to it

## HTTP messages

* HTTP messages are textual information encoded in ASCII.
* HTTP requests, and responses, share similar structures.

1. **Start-line** describing the requests, or its status.
2. **HTTP headers** (optional)
3. **Blank line** indicating all meta-information for the request have been sent.
4. **body** (optional)

### HTTP requests

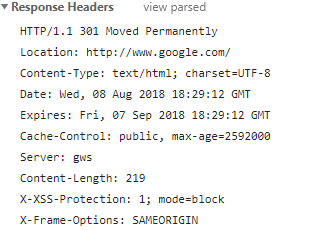
* HTTP requests are messages sent by the client to initiate an action on the server.
* The structure is as follows –
  1. **Start Line**
     + **HTTP method** (like GET, PUT or POST)
     + **Request target**, usually a URL
     + **HTTP version**
  2. **Headers**
     + **General headers**
     + **Request headers**, like User-Agent, Accept-Type
     + **Entity headers,** like Content-Length
  3. **Body**

Example Request (google.com)

### HTTP Response

* HTTP responses are messages sent by the server to the client.
* The structure is as follows –
  1. **Start Line**
     + **The protocol version** usually HTTP/1.1.
     + **A status code** (like 200, 404, or 302)
     + **A status text** (like Not Found)
  2. **Headers**
     + **General headers**
     + **Response headers**, like Vary, Accept-Range
     + **Entity headers,** like Content-Length
  3. **Body**

Example response (google.com)



### HTTP Methods

|  |  |
| --- | --- |
| GET | The GET method retrieve data only. |
| HEAD | The HEAD method asks for a response identical to that of a GET request, but without the response body. |
| POST | The POST method requests that the server accept the entity enclosed in the request. |
| PUT | The PUT method requests insert operation. |
| DELETE | The DELETE method requests delete operation. |
| OPTIONS | The OPTIONS method returns the HTTP methods that the server supports |

## Making HTTP requests

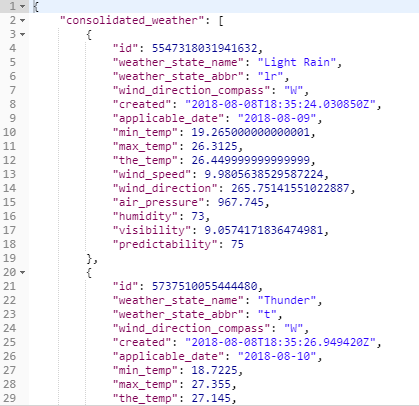
* HTTP requests can be tested wither using the GUI application **Postman** or using the command line tool called **curl**.

### Open APIs

* Open APIs can be used by anyone on the internet to access some functions provided by the API.
* Eg. – Goodreads.com, metaweather.com, jsonplaceholder.com, etc.

### Sending requests through Postman

* API service: metaweather.com
* URL: <https://www.metaweather.com/api/location/2295420/>
* Method: GET
* Result: JSON



### Sending requests through curl

* API service: jsonplaceholder
* URL: <https://jsonplaceholder.typicode.com/posts>
* Body: {"title":"foo","body":"bar","userId":1}
* Method: POST
* Result: JSON

**$ curl -d "{"title":"foo","body":"bar","userId":1}" -X POST https://jsonplaceholder.typicode.com/posts**

{"{title:foo,body:bar,userId:1}": "",

"id": 103

}

* API service: jsonplaceholder
* URL: <https://jsonplaceholder.typicode.com/posts/1/>
* Method: DELETE
* Result: none

**curl -X DELET https://jsonplaceholder.typicode.com/posts/1**

# TCP and UDP

* Both TCP and UDP are transport Layer protocols responsible for data transfer over a network.
* They are built on top the Internet Protocol.

## Transmission Control Protocol

* It is the most commonly used protocol on the Internet.
* It is a connection oriented, highly reliable protocol.
* Connection is established using a three way hand shake.
* Each packet is acknowledged to achieve absolute reliability.

### Applications of TCP:

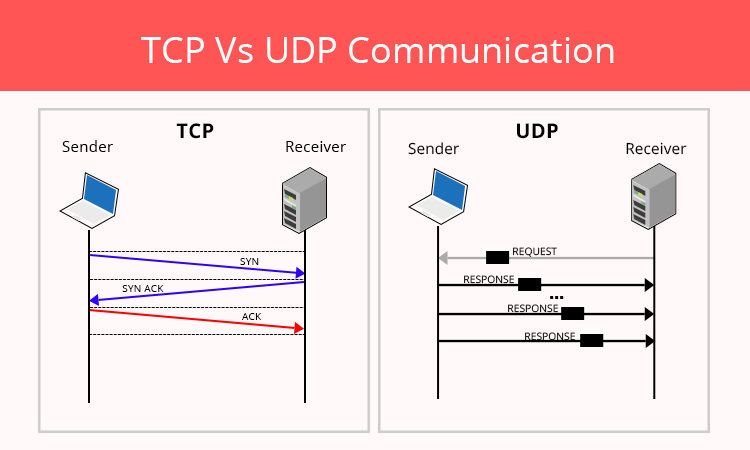
* File Transfer Protocol (FTP), which is used in sending large files.
* Simple Mail Transfer Protocol (SMTP)

## User Datagram Protocol

* It is an unreliable connectionless protocol.
* There is no need to establish any kind of prior connection.
* Packets are not acknowledged by the receiver.
* These properties make UDP more efficient in terms of bandwidth and latency.

### Applications of UDP:

* computer gaming
* voice or video communication
* live conferences



## Video Streaming – UDP or TCP?

* In most cases UDP is used for video streaming.
* As UDP uses less bandwidth and has low latency, it can deliver large videos across a network.
* A few dropped packets lead to small glitches which are often not noticeable.
* **When can you use TCP for streaming** –
  + To get very high quality video on demand.
  + Encrypted video stream.
* **Drawbacks of using TCP for streaming** –
  + TCP will not send packets if a string old previous packets are unacknowledged.
  + This leads to higher buffer times.
  + TCP packets are connection oriented and have larger packets. This increases bandwidth requirement and speed of the stream.
  + TCP does not support multicast, which most video streams use.
  + The higher latency of TCP makes it less desirable for LIVE video.

# Hash Tables

* Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects
* The idea of hashing is to distribute entries (key/value pairs) uniformly across an array.
* The values are then stored in a data structure called **hash table**.
* Each element is assigned a key (converted key).

## Hashing

* **Hashing** is implemented in two steps:
  + An element is converted into an integer by using a hash function.
  + The element is stored in the hash table where it can be quickly retrieved using hashed key.
* A **hash function** is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table.
* A good hash function has these properties:
  + Easy to compute
  + Uniform distribution of data
  + Less collisions
* Hash collisions are when two values correspond to the same hash value.
* Various algorithms are used to uniformly distribute the items with colliding hash values.

## Time Complexity

* **Best case**: Search, insert and delete – O(1)
* **Worst case**: Search, insert and delete – O(n)

## Operations involved in creating a hash table

* Select a suitable hash function
* Select appropriate data structure for buckets
* Implement Collison resolution mechanisms
* Implement basic operations like insert, delete and search.

### Operations visible to the user

* Only the basic operations like insert, delete and search can be used by the user.
* Other operations like the hash generation, collision resolution, allocation of buckets are not accessible by the user.
* Modifying the hash generation methods can lead to highly inefficient hash generation, therefore invalidating the hash map.
* Collison mechanisms that are implemented are reliant on the hash functions and the underlying data structures used.
* Therefore Hashing, Collision, Allocation are abstracted away from the user of a hash table.

## Sample hashing function

* Let’s consider an ACSII key system.
* The hash must generate a hash key from the given ASCII word.
* hashFunction():
  + Sum = sum of ASCII of each letter in word
  + return sum modulo 10
* Example word:
* Hash for **abc** = (97 + 98 + 99) % 10 => 4
* Hash for **efg** = (101 + 102 + 103) % 10 => 6