Y Media Labs

Training Assignment

Ananthu P Kanive
8-9-2018

Table of Contents

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.

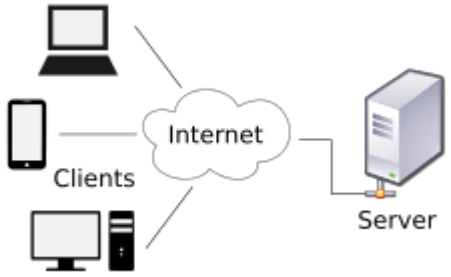
- A server usually provides multiple services to multiple clients, thus are usually computationally advanced compared to standard computers.
- Most servers run UNIX like operating systems, with windows coming second. Other OSes like the macOS are used in tiny numbers.

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.

Hypertext Transfer Protocol

- HTTP is a protocol that run on the application layer of the OSI 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

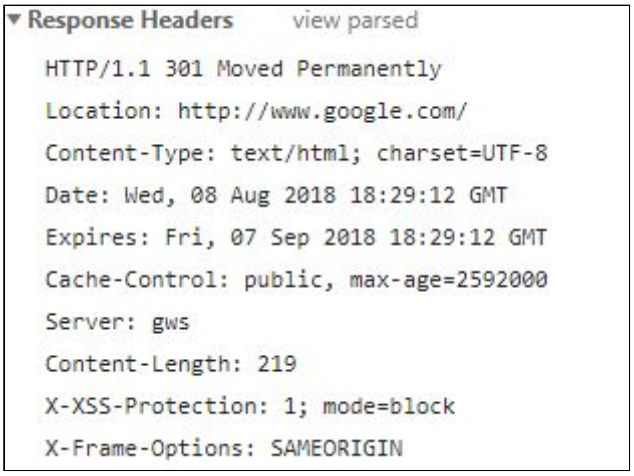
- HTTP requests are messages sent by the client to initiate an action on the server.
- The structure is as follows –
  1. **Start Line**
     - **HTTP method** (like GET, PUT or POST)
     - **Request target**, usually a URL
     - **HTTP version**
  2. **Headers**
     - **General headers**
     - **Request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

Example Request (google.com, chrome)

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, chrome)



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

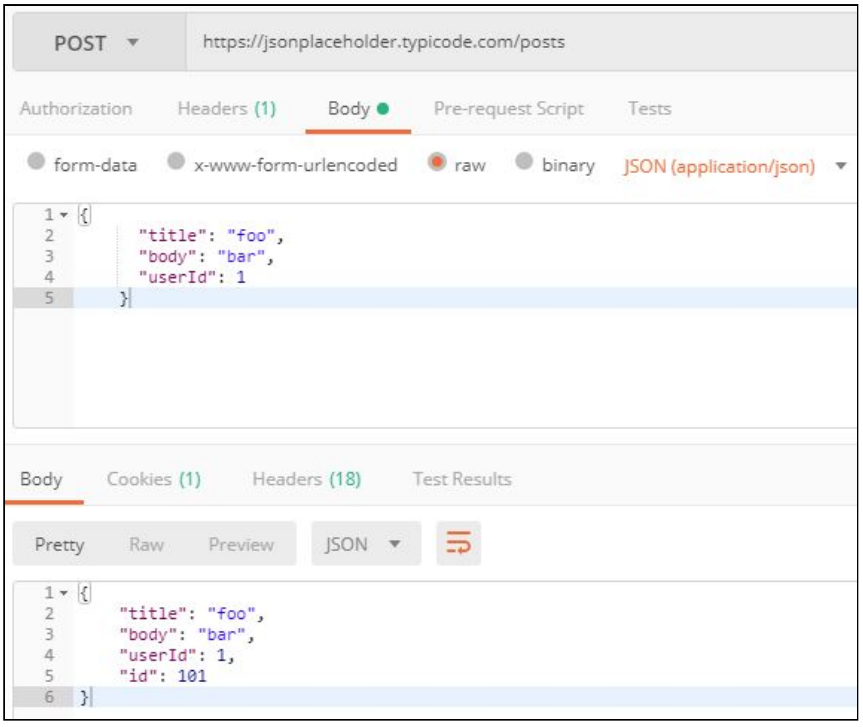
- Open APIs can be used by anyone on the internet to access some functions provided by the API.
- Eg. – Goodreads.com, metaweather.com, jsonplaceholder.com, etc.

Sending requests through Postman

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


```
{
    "consolidated_weather": [
        {
            "id": 5547318031941632,
            "weather_state_name": "Light Rain",
            "weather_state_abbr": "lr",
            "wind_direction_compass": "W",
            "created": "2018-08-08T18:35:24.030850Z",
            "applicable_date": "2018-08-09",
            "min_temp": 19.265000000000001,
            "max_temp": 26.3125,
            "the_temp": 26.449999999999999,
            "wind_speed": 9.9805638529587224,
            "wind_direction": 265.75141551022887,
            "air_pressure": 967.745,
            "humidity": 73,
            "visibility": 9.0574171836474981,
            "predictability": 75
        },
```


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



Sending requests through curl

Usage for HTTP requests

```
curl -d [data] -X [http_method] [URL]
```

Sample requests

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

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

```
{}
```

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

```
curl -X DELETE https://jsonplaceholder.typicode.com/posts/1
```

3. Method: **GET**
    - API service: jsonplaceholder
    - URL: https://jsonplaceholder.typicode.com/posts
    - Result: JSON

```
curl -X GET https://jsonplaceholder.typicode.com/posts?userId=1&id=1
```

```
[{  "userId": 1,

    "id": 1,

    "title": "sunt aut facere repellat provident occaecati excepturi optio
reprehenderit",

    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et
cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem
eveniet architecto"

  }}
```

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 handshake.
- Each packet is acknowledged to achieve absolute reliability.

Characteristics of TCP

1. **Unicast protocol** - supports data exchange between precisely two parties.
2. **Connection state** *:* TCP uses synchronized state between the two endpoints.
3. **Reliable**
4. **Full duplex**
5. **Streaming** - Although TCP uses a packet structure for network transmission, TCP is a true streaming protocol.
6. **Rate adaptation** - The rate of data transfer is intended to adapt to the prevailing load conditions within the network.

Applications of TCP:

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

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.

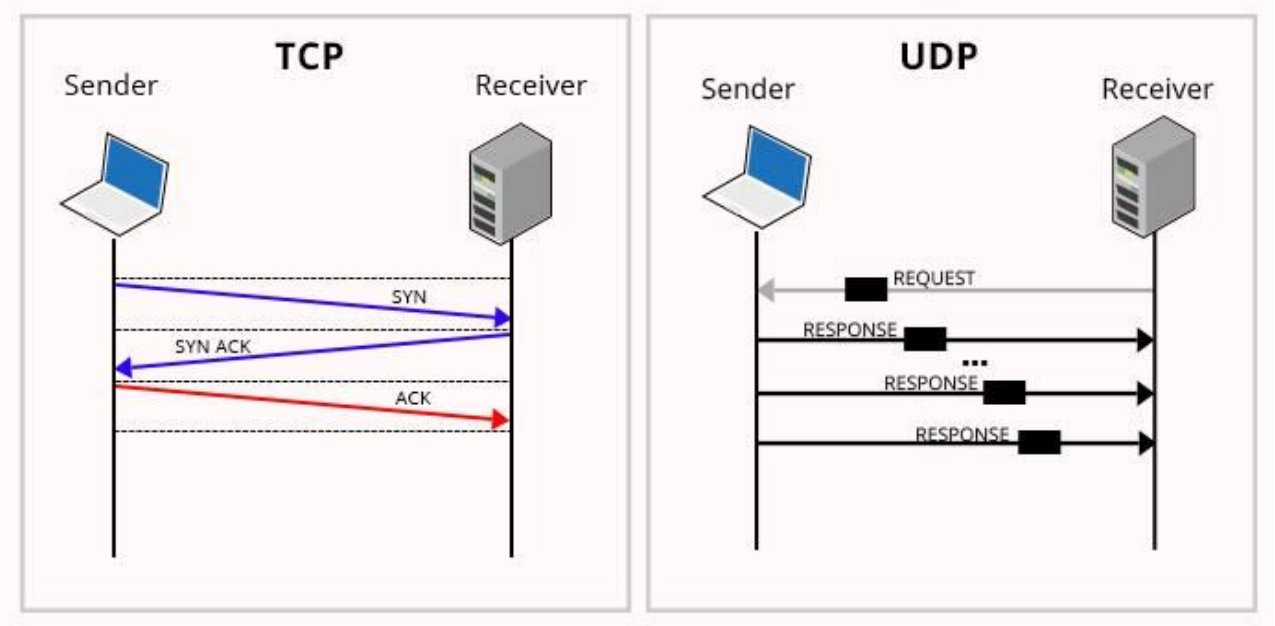
The characteristics of UDP

- End-to-end.
- Unreliable, connectionless delivery
- Best effort semantics
- No ack, no sequence, no flow control
- Fast, low overhead

Applications of UDP:

- computer gaming
- voice or video communication
- live conferences

(Comparison between a TCP and UDP transmission)



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 of 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 requirements 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.

Dictionary (Associative Array)

- Data dictionaries are a collection of data which must be accessible through a unique key.
- It is not purely a data structure, but is implemented using a variety of data structures.
- A popular approach is to use **hash tables**.

Hashing

- A **hash function** is any function that can be used to store a data set of any size to a data set of a fixed size. (eg. 300 elements stored in a hash table of size 10)
- Various algorithms are used to uniformly distribute the items.

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 collision 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.\

Implementation

A simple hashing function

```
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

Data Structure for the hash table

- The hashFunction() will return values between 0 to 9. Thus an array of size 10.
- It can be used to store values corresponding to the hash value to its index.
- Each of the array element will contain a link to a linked list to store the actual data.

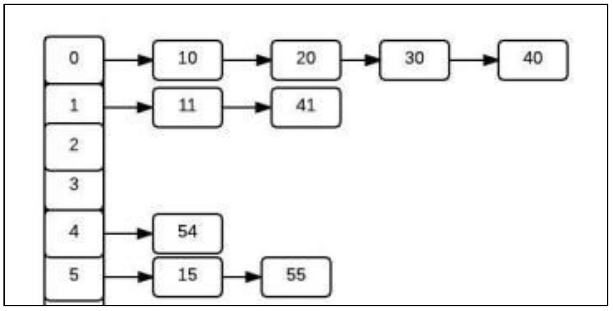
Insert, Delete and Search operations

- In hash tables, basic operations are as trivial as finding the hash value using the hashFunction()
- Insert, delete and search is done using linked-list operations.

```
findElement(id):
        hash = hashFunction(id)
        list = hashTable[hash]
        element = list.find(id)
        return element
```

Simple Collision Resolution mechanism

- As a linked list is used for storing values, values with the same hash can be appended to the list.
- During a lookup, the program must traverse through the list to find a suitable match.
- This type of collision resolution is called **separate chaining**.



insertElement(id, element):

```
        hash = hashFunction(id)
        list = hashTable[hash]
        if list is Null:
                create new list
                add element
                hashTable[hash] = list
        else
                list.append(element)
```

Other structures that can be used

- **Binary Search trees** can be used when query searching is a major activity.
- **Arrays** are sometimes used for **small data sets** as the performance gain by other structures may not be significant.
- **Suffix Trees** are used when the keys are purely ASCII.