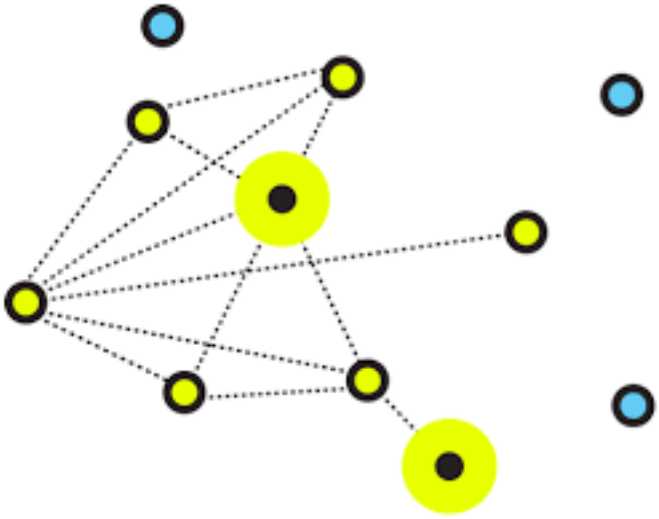
Chapter IV

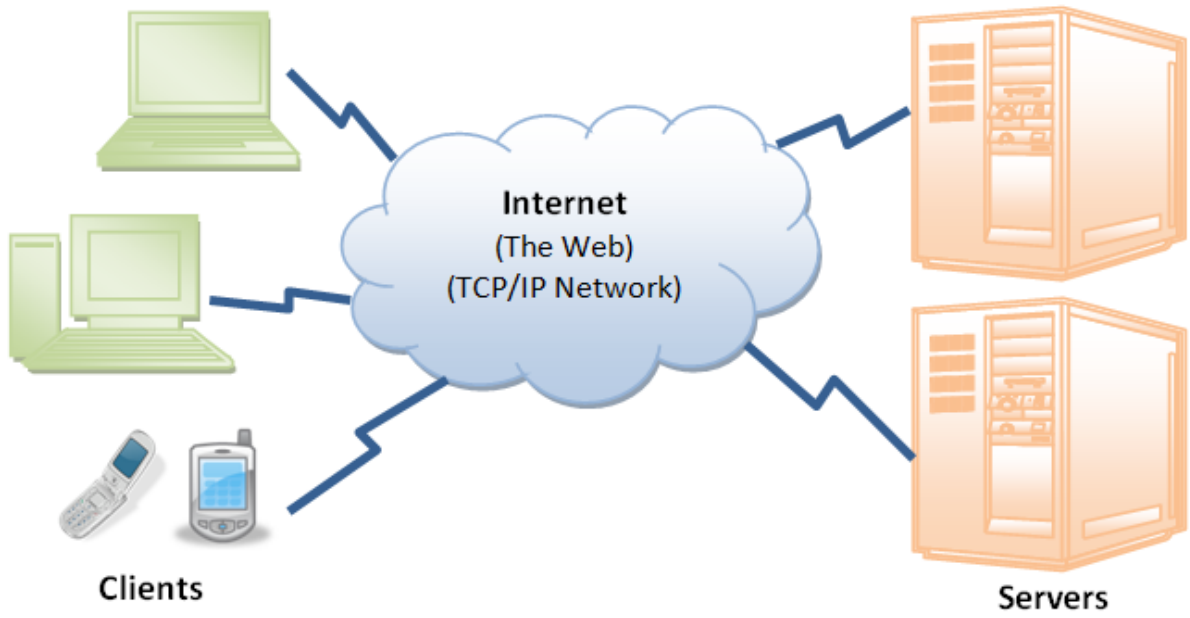
Networking



1.Introduction

The WEB

Internet (or The Web) is a massive distributed client/server information system as depicted in the following diagram.

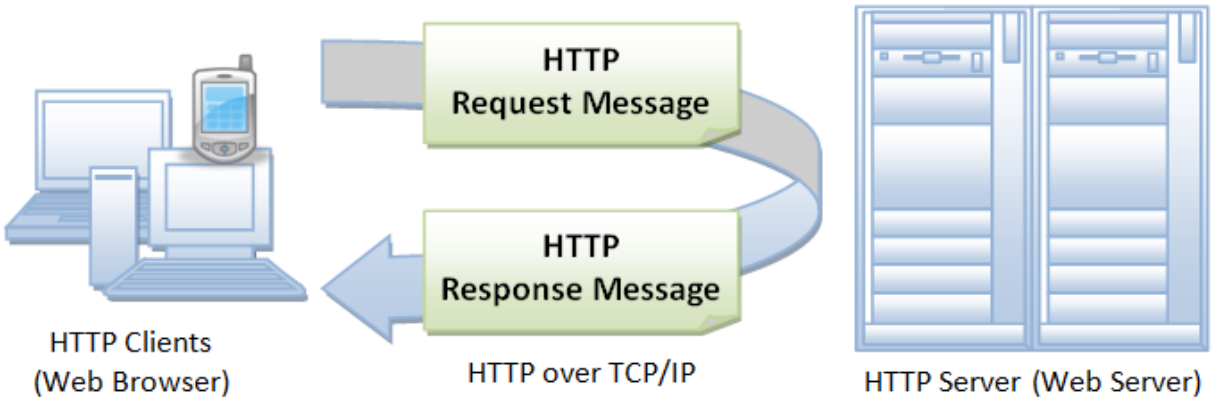


Many applications are running concurrently over the Web, such as web browsing/surfing, e-mail, file transfer, audio & video streaming, and so on. In order for proper communication to take place between the client and the server, these applications must agree on a specific application-level protocol such as HTTP, FTP, SMTP, POP, and etc.

HyperText Transfer Protocol (HTTP)

HTTP (Hypertext Transfer Protocol) is perhaps the most popular application protocol used in the Internet (or The WEB).

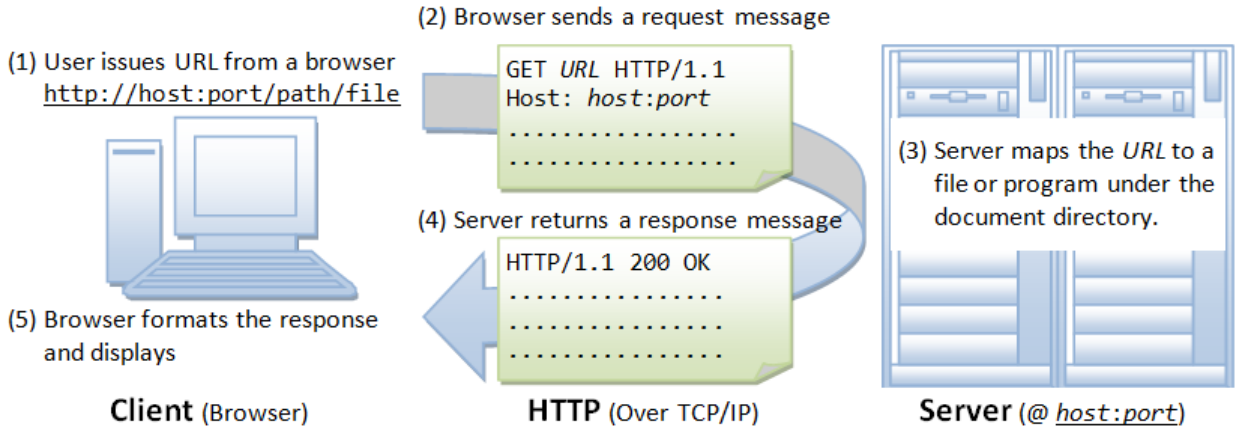
- HTTP is an asymmetric request-response client-server protocol as illustrated. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a pull protocol, the client pulls information from the server (instead of server pushes information down to the client).



- HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests.
- HTTP permits negotiating of data type and representation, so as to allow systems to be built independently of the data being transferred.
- Quoting from the RFC2616: "The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request **methods**, **error** codes and **headers**."

Browser

Whenever you issue a URL from your browser to get a web resource using HTTP, e.g. http://www.nowhere123.com/index.html, the browser turns the URL into a request message and sends it to the HTTP server. The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message. This process is illustrated below:



Uniform Resource Locator (URL)

A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following syntax:

protocol://hostname:port/path-and-file-name

There are 4 parts in a URL:

1. **Protocol**: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
2. **Hostname**: The DNS domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.
3. **Port**: The TCP port number that the server is listening for incoming requests from the clients.
4. **Path-and-file-name**: The name and location of the requested resource, under the server document base directory.

eg,

in the URL http://www.nowhere123.com/docs/index.html, the communication protocol is HTTP; the hostname is www.nowhere123.com. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is "/docs/index.html".

HTTP Protocol Translation

whenever you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server.

For example, the browser translated the URL

Https://www.dribbble.com/index.html into the following request message:

```
GET /index.html
Host: www.dribbble.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

When this request message reaches the server, the server can take either one of these actions:

1. The server interprets the request received, maps the request into a file under the server's public / document directory, and returns the file requested to the client.
2. The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.
3. The request cannot be satisfied, the server returns an error message.

An example of the HTTP response message is as shown:

Accept-Ranges →bytes

Cache-Control →max-age=0, private, must-revalidate
Connection →keep-alive
Content-Encoding →gzip
Content-Type →text/html; charset=utf-8

```
<html><body><h1>.  .the file content. . </h1></body></html>
```

The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf".

For More Information :
https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

2.Network Request Components

- **Header (Request and response header)**
- **Methods (HTTP request methods)**
- **PARAMETERS**
- **BODY**

Headers

1. **Request Headers**

The request headers are in the form of `name:value` pairs. Multiple values, separated by commas, can be specified.

"accept":" "application/json"

"authorization":"Bearer HWXJS2122!….."

"content-type":"application/json".

2. **Response Headers**

response header returns as following example:

```
Content-Type: text/html
Content-Length: 35
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
```

The response message **body** contains the resource data requested.

Methods

HTTP protocol defines a set of request methods. A client can use one of these request methods to send a request message to an HTTP server. The methods are:

- **GET**: A client can use the GET request to get a web resource from the server.
- **POST**: Used to post data up to the web server.
- **PUT**: Ask the server to store the data.
- **DELETE**: Ask the server to delete the data.
- **HEAD**: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the

last-modified date of the data, this can be used to check against the local cache copy.

- **TRACE**: Ask the server to return a diagnostic trace of the actions it takes.
- **OPTIONS**: Ask the server to return the list of request methods it supports.
- CONNECT: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.
- Other extension methods

(method name is case sensitive and must be in uppercase)

3.Making Network Request In Flutter

Library: http.dart

This library comes with the Flutter framework and lives inside the http package. This dart file has all the methods to make a HTTP request. You can read about this library in great detail through the Flutter website.
Available Methods: get , post, put , delete.

1. **GET** Request (Eg.1)

Eg.1 var result = get('https://jsonplaceholder.typicode.com/ photos');

The make network call (Asynchronous) to synchronous , use await and async the final result will be alike:

```
void fetchData() async {
   var result = await get('https://jsonplaceholder.typicode.com/photos');
   print(result.body);

}
//Alternatively

void fetchData()  {
  get('https://jsonplaceholder.typicode.com/photos').then((result){
    print(result.body);
 });
```

The variable result will hold the NetworkResponse. It will have many things in it like headers, statusCode etc which includes the body as well. result.body will be holding the JSON body.

Eg.2

```
var APIBASEURL = "jsonplaceholder.typicode.com":
var _path = "/photos":
var _params ;  // apply as  Map{String,String} when needed
 Uri uri = new Uri.http(APIBASEURL, _path, _params);
 print(uri);

 final response = await get( uri ,
        headers: {
                HttpHeaders.acceptHeader: 'application/json',
                HttpHeaders.authorizationHeader: auth,
                // HttpHeaders.contentTypeHeader: 'application/json',

     }
);

if (response.statusCode == 200 ) {

        var json = jsonDecode( utf8.decode(response.bodyBytes) );
        //Process json
} else {
        //Show Error as per server response.status error code
}
```

Eg.3

```
try {

        var APIBASEURL = “jsonplaceholder.typicode.com":

        var _path = “/photos”:

        var _params ;  // apply as  Map{String,String} when needed

        Uri uri = new Uri.http(APIBASEURL, _path, _params);
        print(uri);

        final response = await get( uri ,
                headers: {
                        HttpHeaders.acceptHeader: 'application/json',
                        HttpHeaders.authorizationHeader: auth,
                        // HttpHeaders.contentTypeHeader: 'application/json',

                }
        );

        if (response.statusCode == 200 ) {
                var json = jsonDecode( utf8.decode(response.bodyBytes) );
                //Process json
        } else {
                //Show Error as per server response.status error code
        }
} catch ( e ) {

        print(e )

}
```

2. **POST** Request

Eg.

```
try {

        Map<String, String> _param = param;

        _params["name"] =  username;


       _params["phone"] =  phone_number;

      _params["email"] =  email ?? "";

      _params["profile"] =  profile ?? "";


        String _path = '/api/v1/rent';

     var user_token = Token.current;
     String auth =  token_type + " " +  tokenvalue;

     print(auth);

     Uri uri = new Uri.http(APIBASEURL, _path, _param);
     print(uri);

     final response = await post(uri, headers: {
       HttpHeaders.acceptHeader: 'application/json',
       HttpHeaders.contentTypeHeader: 'application/json',
       HttpHeaders.authorizationHeader: auth
       //"callMethod": "DOCTOR_AVAILABILITY"
     });
     if (response.statusCode == 201) {


       }

} catch (e ) {



}
```

3. **PUT** Request

```
try {

        Map<String, String> _param = param;

       _params["name"] =  username;


       _params[“phone"] =  phone_number;

      _params["email"] =  email ?? "";

      _params["profile"] =  profile ?? "";


        String _path = '/api/v1/rent';

     var user_token = Token.current;
     String auth =  token_type + " " +  tokenvalue;

     print(auth);

     Uri uri = new Uri.http(APIBASEURL, _path, _param);
     print(uri);

     final response = await post(uri, headers: {
       HttpHeaders.acceptHeader: 'application/json',
       HttpHeaders.contentTypeHeader: 'application/json',
       HttpHeaders.authorizationHeader: auth
       //"callMethod": "DOCTOR_AVAILABILITY"
     });
     if (response.statusCode == 201) {


        }

} catch (e ) {



}
```