**HTTP**

Table of Contents

[URLs 3](#_Toc100513467)

[Connection Establishment 4](#_Toc100513468)

[HTTP Requests 4](#_Toc100513469)

[HTTP Response 6](#_Toc100513470)

[HTTPS and S-HTTP 7](#_Toc100513471)

[Statelessness of HTTP 7](#_Toc100513472)

[Session Identifier 8](#_Toc100513473)

**Hyper Text Transfer Protocol** (HTTP) is a protocol which allows the fetching of resources of any type. It is the foundation of data exchange on the web. It is a **client-server protocol**, meaning the client, i.e. the Web Browser, initiates requests. A complete document is reconstructed from the different sub-documents fetched, such as text, layout descriptions, images, video, etc.

HTTP specifies:

* How a client and server establish a connection
* How the client requests data from the server
* How the server responds to that request
* How the connection is closed

Before establishing an HTTP connection, the client and server must establish a **TCP connection**. The HTTP connection itself is always initiated by the client, creating a master-slave relationship.

HTTP is a **stateless protocol**, meaning each request is independent, but it is also **sessionful**, meaning consecutive requests from the same client will be recognized to be coming from the same client. The details of this will be discussed later.

## URLs

A **Uniform Resource Locator** (URL) is used to locate a specific resource on the internet. It is used when a web client makes a request to a server for the resource.



scheme://host:post/path?query1=value1&query2=value2

A URL has several parts:

* **Scheme** – This identifies the protocol to be used to access the resource, HTTP, HTTPS, FTP, etc.
* **Host** – This is a unique name which can be used instead of the IP address to identify the server.
* **Port** – Host names can sometimes be followed by a port number. HTTP for example, works on port number 80 by default. If the default port number for the protocol is being used, it can be omitted. More information about port numbers can be found [here](https://www.ibm.com/docs/en/cics-ts/5.1?topic=concepts-port-numbers#dfhtl2d).
* **Path** – This is the local path to the resource on the server.
* **Query String** – This section provides several key-value pairs, separated by ampersands. It begins with a question mark. The key-value pairs are used to send some extra information to the server from the client.

More information about the components of a URL can be found [here](https://www.ibm.com/docs/en/cics-ts/5.1?topic=concepts-components-url).

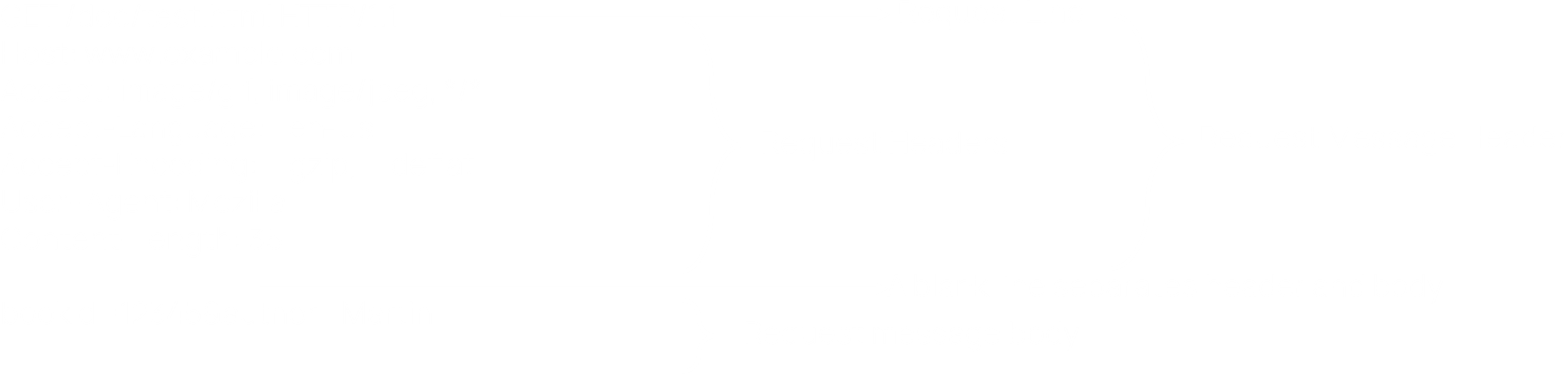
## Connection Establishment

The steps of connection establishment in HTTP are simple:

1. The user issues a URL from the browser.
2. The browser sends a request message.
3. The server maps the URL to a file.
4. The server returns a response message.

The request and response messages are of particular interest.

## HTTP Requests



An **HTTP Request** consists of:

* A request method
* A request URL
* Header fields, which contain metadata
* Body, which contains data, such as user-entered data or uploaded files that are to be sent to the server

In the diagram above, the **request line** contains the request method, the relative path to the resource on the server and the HTTP version being used. The request line along with the **request headers** form the **request message header**. Then there is a blank line, followed by the **request message body**.

The **request method** is also very important. It can be of several types:

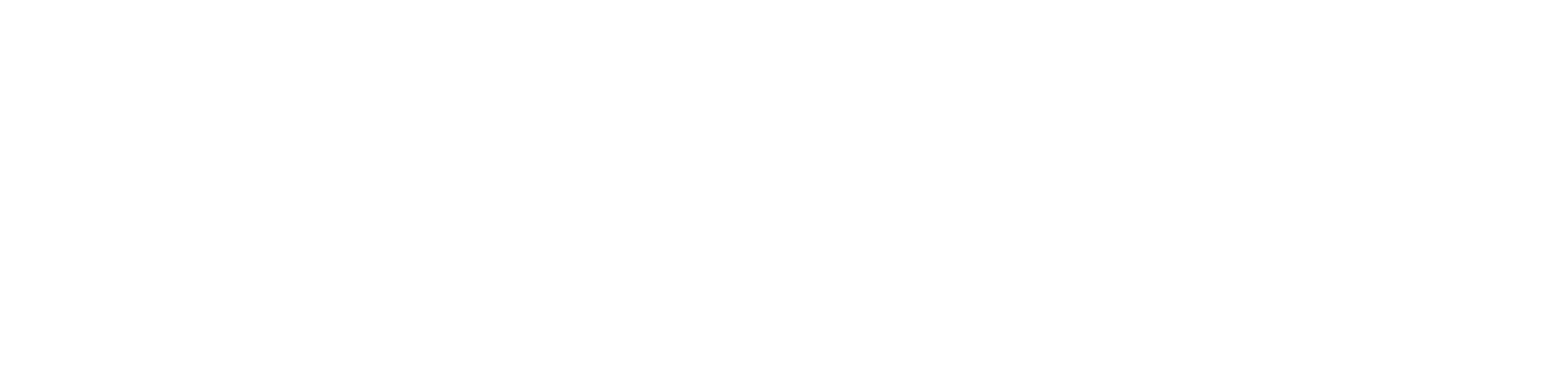
* **GET** – This is used to retrieve data from the server, perhaps using specified query parameters.
* **HEAD** – This returns just the headers for the requested URL.
* **POST** – This is used to send data of unlimited length to the server.
* **PUT** – This is used to store a resource on the server.
* **DELETE** – This is used to remove a resource from the server.
* **OPTIONS** – This returns the HTTP methods supported by the server.
* **TRACE** – This returns the header fields sent with the request.

The above methods are supported in **HTTP 1.1**. HTTP 1.0 only supports the GET, HEAD and POST methods.

Most user interactions fall under **CRUD**, which stands for **Create, Read, Update, Delete**. The POST method is used to create, the GET method is used to read, the PUT method is used to both create and update while the DELETE method is used to delete.

The GET and the POST methods in particular are quite similar. The GET method can also send some data to the server in the form of queries in the URL. The difference is that the GET method uses the URL to send the information, which lacks privacy and is also limited to 256 characters, while the POST method uses the request body, which provides privacy and also has no limit.

## HTTP Response



An **HTTP Response** contains:

* A result code
* Header fields
* A body, which contains the requested resource, or explanatory text in case of an error

Both the result code and the header fields must come before the body.

In the diagram above, the **status line** contains the HTTP version, the response status code and the status code description.

Commonly used result codes include:

* 100 – Continue
* 200 – OK
* 404 – Resource not found
* 401 – Request requires HTTP authentication
* 500 – Error inside HTTP server
* 503 – Server overload

## HTTPS and S-HTTP

**HTTPS** is exactly the same as HTTP, except that the **default port** is 443 and there is an additional encryption layer between the HTTP and the TCP layers. This encryption is provided using **Transport Layer Security** (TLS), formerly called the **Secure Sockets Layer** (SSL).

**S-HTTP** (Secure HTTP) is exactly the same as HTTPS. The only reason we ended up using HTTPS everywhere is because when these two protocols were defined, Microsoft and Netscape, the creator of HTTPS, supported HTTPS instead.

## Statelessness of HTTP

As discussed before, HTTP is a **stateless** protocol. This means when a client makes a request to a server, the server is not able to identify exactly which client made the request. Because of this, if multiple separate requests are made from the same client, all the requests are **independent**.

The benefit to this is that no information regarding the state needs to be stored. For example, if HTTP was stateful, it would need to send all the information about the current state when making a request. This is avoided. However, this also poses a problem. Since no state information is stored, we are also unable to store any information about the **user session**.

Consider that we are on a Facebook page and we like one post and then try to like another one. If HTTP is truly stateless, we should be forced to login again. This is because which specific user is logged in to the current device is part of the user session information. However, this does not happen in reality. This suggests that the user session information is being stored somehow.

### Session Identifier

Whenever the client makes a request to the server, the server first checks the request headers for a **session identifier**. If the request is the **initial request**, there is no session ID in the header. This allows the server to figure out that this is the start of a new session. This will happen if the user logs in.

When the initial request without the session ID is made, the sever generates a **unique session ID** and sends this ID with the response. For all **future requests** (until the user logs out that is), the client puts this session ID in the request header.

The user’s **session information** is stored in the **server’s database**. The server checks that the provided session ID is **valid**, and uses it to retrieve the user’s session information. Based on this, it sends back a response containing an HTTP page.

If the session ID is **invalid**, the user is **forcefully logged out** on the client end.

On the client side, the session ID is stored in **cookies**. Note that only the ID is stored, not the user’s session information itself.