Task 1

Date: 9 JAN 2021

1. **Difference between http1.1 and http2.**

HTTP:

HTTP stands for hypertext transfer protocol, and it is the basis for almost all web applications. HTTP is the method computers and servers use to request and send information.

For instance, when someone navigates to abcdef.com on their laptop, their web browser sends an HTTP request to the abcdef servers for the content that appears on the page. Then, abcdef servers send HTTP responses with the text, images, and formatting that the browser displays to the user.

Differences:

* HTTP/2 solves several problems that the creators of HTTP/1.1 did not anticipate.
* HTTP/2 is much faster and more efficient than HTTP/1.1.
* One of the ways in which HTTP/2 is faster is in how it prioritizes content during the loading process.

Prioritization:

* Prioritization refers to the order in which pieces of content are loaded.
* However, prioritization affects a webpage's load time in HTTP2.

Prioritization in HTTP/2 and the impact in the performance:

* In HTTP/2, we have hands-on, detailed control over prioritization. This allows us to maximize perceived and actual page load speed to a degree that was not possible in HTTP/1.1.
* HTTP/2 offers a feature called weighted prioritization.
* This allows us to decide which page resources will load first, every time.
* In HTTP/2, when a client makes a request for a webpage, the server sends several streams of data to the client at once, instead of sending one thing after another. This method of data delivery is known as multiplexing.
* We can assign each of these data streams a different weighted value, and the value tells the client which data stream to render first.

Other differences between HTTP/2 and HTTP/1.1 that impact performance?

Multiplexing:

* HTTP/1.1 loads resources one after the other, so if one resource cannot be loaded, it blocks all the other resources behind it.
* In contrast, HTTP/2 is able to use a single TCP connection to send multiple streams of data at once so that no one resource blocks any other resource.
* HTTP/2 does this by splitting data into binary-code messages and numbering these messages so that the client knows which stream each binary message belongs to.

Server push:

* A server only serves content to a client device if the client asks for it. However, this approach is not always practical for modern webpages, which often involve several dozen separate resources that the client must request.
* HTTP/2 solves this problem by allowing a server to "push" content to a client before the client asks for it.

Header compression:

* Small files load more quickly than large ones.
* To speed up web performance, both HTTP/1.1 and HTTP/2 compress HTTP messages to make them smaller.
* However, HTTP/2 uses a more advanced compression method called HPACK that eliminates redundant information in HTTP header packets.
* This eliminates a few bytes from every HTTP packet. Given the volume of HTTP packets involved in loading even a single webpage, those bytes add up quickly, resulting in faster loading.

1. **Evolution of HTTP / HTTP version History:**

HTTP has four versions — HTTP/0.9, HTTP/1.0, HTTP/1.1, and HTTP/2.0. Today the version in common use is HTTP/1.1 and the future will be HTTP/2.0.

**HTTP/0.9 — The One-line Protocol**

1. Initial version of HTTP — a simple client-server, request-response, telenet-friendly protocol
2. Request nature: single-line (method + path for requested document)
3. Methods supported: GET only
4. Response type: hypertext only
5. Connection nature: terminated immediately after the response
6. No HTTP headers (cannot transfer other content type files), No status/error codes, No URLs, No versioning.

**HTTP/1.0 — Building extensibility**

1. Browser-friendly protocol
2. Provided header fields including rich metadata about both request and response (HTTP version number, status code, content type)
3. Response: not limited to hypertext (Content-Type header provided ability to transmit files other than plain HTML files — e.g. scripts, stylesheets, media)
4. Methods supported: GET , HEAD , POST
5. Connection nature: terminated immediately after the response

**HTTP/1.1 — the standardized protocol**

1. This is the HTTP version currently in common use.
2. Introduced critical performance optimizations and feature enhancements — persistent and pipelined connections, chunked transfers, compression/decompression, content negotiations, virtual hosting (a server with a single IP Address hosting multiple domains), faster response and great bandwidth savings by adding cache support.
3. Methods supported: GET , HEAD , POST , PUT , DELETE , TRACE , OPTIONS
4. Connection nature: long-lived

**HTTP/2.0 and the future**

1. All above features are being used by major web servers and browsers today.
2. But modern enhancements like HTTP/2.0, Server Side Events (SSE), and Websockets have changed the way that the traditional HTTP works.
3. **Difference between Browser JS(console) vs Nodejs**
4. In browser “window” is a predefined global object which has functions and attributes, whereas Nodejs doesn’t have it.
5. In browser “location” is another predefined object, whereas Nodejs doesn’t have it.
6. In browser “require” is not predefined object, whereas Nodejs has it.
7. In browser module is not required, where as in Nodejs you have to keep your code inside the module.
8. In browser “document” is a predefined object, whereas Nodejs doesn’t have it.
9. **What happens when you type an URL in the browser and press enter?**

* When you a URL into the address bar of your browser, the browser checks the cache for a DNS record to find the corresponding IP address of the URL entered.
* DNS (Domain Name System) is a database that maintains the name of the website (URL) and the particular IP address it links to.
* Every single URL on the internet has a unique IP address assigned to it.
* The IP address belongs to the computer which hosts the server of the website we are requesting to access.

To find the DNS record, the browser checks four caches.

● First, it checks the **browser cache**. The browser maintains a repository of DNS records for a fixed duration for websites you have previously visited. So, it is the first place to run a DNS query.

● Second, the browser checks the **OS cache**. If it is not in the browser cache, the browser will make a system call (i.e., gethostname on Windows) to your underlying computer OS to fetch the record since the OS also maintains a cache of DNS records.

● Third, it checks the **router cache**. If it’s not on your computer, the browser will communicate with the router that maintains its’ own cache of DNS records.

● Fourth, it checks the **ISP cache**. If all steps fail, the browser will move on to the ISP. Your ISP maintains its’ own DNS server, which includes a cache of DNS records, which the browser would check with the last hope of finding your requested URL.

* If the requested URL is not in the cache, ISP’s DNS server initiates a DNS query to find the IP address of the server that hosts the entered URL
* For a computer to connect with the server that hosts the entered URL, a user need the IP address of maps.google.com. The purpose of a DNS query is to search multiple DNS servers on the internet until it finds the correct IP address for the website. This type of search is called a recursive search since the search will repeatedly continue from a DNS server to a DNS server until it either finds the IP address we need or returns an error response saying it was unable to find it.
* In this situation, we would call the ISP’s DNS server a DNS recursor whose responsibility is to find the proper IP address of the intended domain name by asking other DNS servers on the internet for an answer. The other DNS servers are called name servers since they perform a DNS search based on the domain architecture of the website domain name.
* Further, mostly all the URL contain a third-level domain, a second-level domain, and a top-level domain. Each of these levels contains their own name server, which is queried during the DNS lookup process.
* For example, futures.abc.com, first, the DNS recursor will contact the root name server. The root name server will redirect it to the .com domain name server. .com name server will redirect it to the abc.com name server. The abc.com name server will find the matching IP address for futures.abc.com in its’ DNS records and return it to your DNS recursor, which will send it back to your browser.
* These requests are sent using small data packets that contain information such as the content of the request and the IP address it is destined for (IP address of the DNS recursor). These packets travel through multiple networking equipment between the client and the server before it reaches the correct DNS server. This equipment use routing tables to figure out which way is the fastest possible way for the packet to reach its’ destination. If these packets get lost, you’ll get a request failed error. Otherwise, they will reach the correct DNS server, grab the correct IP address, and come back to your browser.
* The browser initiates a TCP connection with the server.

Once the browser receives the correct IP address, it will build a connection with the server that matches the IP address to transfer information. Browsers use internet protocols to build such connections. There are several different internet protocols that can be used, but TCP is the most common protocol used for many types of HTTP requests.

* The browser sends an HTTP request to the webserver.

Once the TCP connection is established, it is time to start transferring data! The browser will send a GET request asking for futures.abc.com web page. If an user is entering credentials or submitting a form, this could be a POST request. This request will also contain additional information such as browser identification (User-Agent header), types of requests that it will accept (Accept header), and connection headers asking it to keep the TCP connection alive for additional requests. It will also pass information taken from cookies the browser has in store for this domain.

* The server handles the request and sends back a response.

The server contains a webserver (i.e., Apache, IIS) that receives the request from the browser and passes it to a request handler to read and generate a response. The request handler is a program (written in ASP.NET, PHP, Ruby, etc.) that reads the request, its’ headers, and cookies to check what is being requested and also update the information on the server if needed. Then it will assemble a response in a particular format (JSON, XML, HTML).

* The server sends out an HTTP response.

The server response contains the web page you requested as well as the status code, compression type (Content-Encoding), how to cache the page (Cache-Control), any cookies to set, privacy information, etc.

* The browser displays the HTML content (for HTML responses, which is the most common).