**Task 01**

1. **Difference between HTTP1.1 vs HTTP2**

**HTTP/1.1**

* In HTTP/1.1, flow control relies on the underlying TCP connection. When this connection initiates, both client and server establish their buffer sizes using their system default settings. If the receiver’s buffer is partially filled with data, it will tell the sender its *receive window*, i.e., the amount of available space that remains in its buffer. This receive window is advertised in a signal known as an *ACK packet*, which is the data packet that the receiver sends to acknowledge that it received the opening signal. If this advertised receive window size is zero, the sender will send no more data until the client clears its internal buffer and then requests to resume data transmission. It is important to note here that using receive windows based on the underlying TCP connection can only implement flow control on either end of the connection.
* Because HTTP/1.1 relies on the transport layer to avoid buffer overflow, each new TCP connection requires a separate flow control mechanism. HTTP/2, however, multiplexes streams within a single TCP connection, and will have to implement flow control in a different manner.

**HTTP/2**

* HTTP/2 multiplexes streams of data within a single TCP connection. As a result, receive windows on the level of the TCP connection are not sufficient to regulate the delivery of individual streams. HTTP/2 solves this problem by allowing the client and server to implement their own flow controls, rather than relying on the transport layer. The application layer communicates the available buffer space, allowing the client and server to set the receive window on the level of the multiplexed streams. This fine-scale flow control can be modified or maintained after the initial connection via a WINDOW\_UPDATE frame.
* Since this method controls data flow on the level of the application layer, the flow control mechanism does not have to wait for a signal to reach its ultimate destination before adjusting the receive window. Intermediary nodes can use the flow control settings information to determine their own resource allocations and modify accordingly. In this way, each intermediary server can implement its own custom resource strategy, allowing for greater connection efficiency.
* This flexibility in flow control can be advantageous when creating appropriate resource strategies. For example, the client may fetch the first scan of an image, display it to the user, and allow the user to preview it while fetching more critical resources. Once the client fetches these critical resources, the browser will resume the retrieval of the remaining part of the image. Deferring the implementation of flow control to the client and server can thus improve the perceived performance of web applications.
* In terms of flow control and the stream prioritization mentioned in an earlier section, HTTP/2 provides a more detailed level of control that opens up the possibility of greater optimization. The next section will explain another method unique to the protocol that can enhance a connection in a similar way: predicting resource requests with *server push*.

1. **Http version history**

* The **Hypertext Transfer Protocol** (**HTTP**) is an [application layer](https://en.wikipedia.org/wiki/Application_layer) protocol for distributed, collaborative, [hypermedia](https://en.wikipedia.org/wiki/Hypermedia) information systems. HTTP is the foundation of data communication for the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), where [hypertext](https://en.wikipedia.org/wiki/Hypertext) documents include [hyperlinks](https://en.wikipedia.org/wiki/Hyperlink) to other resources that the user can easily access, for example by a [mouse](https://en.wikipedia.org/wiki/Computer_mouse) click or by tapping the screen in a web browser.
* Development of HTTP was initiated by [Tim Berners-Lee](https://en.wikipedia.org/wiki/Tim_Berners-Lee) at [CERN](https://en.wikipedia.org/wiki/CERN) in 1989. Development of early HTTP [Requests for Comments](https://en.wikipedia.org/wiki/Requests_for_Comments) (RFCs) was a coordinated effort by the [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) and the [World Wide Web Consortium](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C), with work later moving to the IETF.
* **HTTP/1.1** was first documented in [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [2068](https://tools.ietf.org/html/rfc2068) in 1997, and as of 2021, it (plus older versions) is less popular (used by less than a third of [websites](https://en.wikipedia.org/wiki/Website); it's always a backup protocol) for web serving than its successors. That specification was obsoleted by [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [2616](https://tools.ietf.org/html/rfc2616) in 1999, which was likewise replaced by the [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7230](https://tools.ietf.org/html/rfc7230) family of RFCs in 2014.
* [**HTTP/2**](https://en.wikipedia.org/wiki/HTTP/2)is a more efficient expression of HTTP's semantics "on the wire", and was published in 2015, and is used by over 50% of websites; it is now supported by virtually all web browsers and major web servers over [Transport Layer Security](https://en.wikipedia.org/wiki/Transport_Layer_Security) (TLS) using an [Application-Layer Protocol Negotiation](https://en.wikipedia.org/wiki/Application-Layer_Protocol_Negotiation) (ALPN) extension where [TLS 1.2](https://en.wikipedia.org/wiki/TLS_1.2) or newer is required.
* [**HTTP/3**](https://en.wikipedia.org/wiki/HTTP/3) is the proposed successor to HTTP/2, and 2/3rd of web browser users (both on desktop and mobile) can already use HTTP/3, on the 18% of websites that already support it; uses [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol) instead of [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) for the underlying transport protocol. Like HTTP/2, it does not obsolete previous major versions of the protocol. Support for HTTP/3 was added to [Cloudflare](https://en.wikipedia.org/wiki/Cloudflare) and [Google Chrome](https://en.wikipedia.org/wiki/Google_Chrome) in September 2019 (since enabled by default), and can be enabled in the stable versions of Firefox[[10]](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#cite_note-10) and Safari.

1. **List 5 difference between Browser JS(console) vs Nodejs**

**Browser JS(console)**

|  |
| --- |
| * Javascript is a programming language that is used for writing scripts on the website. * Javascript can only be run in the browsers. * It is basically used on the client-side. * Javascript is capable enough to add HTML and play with the DOM. * Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox. * Javascript is used in frontend development. * Some of the javascript frameworks are RamdaJS, TypedJS, etc. * It is the upgraded version of ECMA script that uses Chrome’s V8 engine written in C++.   **NodeJS**   * NodeJS is a Javascript runtime environment. * Nodejs does not have capability to add HTML tags. * Nodejs can only run in V8 engine of google chrome. * Nodejs is used in server-side development. * Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm. * Nodejs is written in C, C++ and Javascript. |
| **4.what happens when you type a URL in the address bar in the browser?**   1. You enter a URL into a web browser 2. The browser looks up the IP address for the domain name via DNS 3. The browser sends a HTTP *request* to the server 4. The server sends back a HTTP *response* 5. The browser begins rendering the HTML 6. The browser sends requests for additional objects embedded in HTML (images, css, JavaScript) and repeats steps 3-5. 7. Once the page is loaded, the browser sends further async requests as needed. |
|  |
|  |
|  |
|  |