Jahong Liu

System Design Basics 7: Web Socket

Web Socket

Web Socket

In this lesson, let's dig deeper into the application level protocols. There are multiple application level protocols such as HTTP, WebSocket, FTP, SMTP, SSH, and WebRTC. Every single one of these protocols makes use of TCP, except WebRTC, which uses UDP.

The problem

Let's consider the scenario of building a chat application. In this context, HTTP works best when dealing with stateless data. It follows a request-response protocol, where the client requests a page and the server responds with the requested page. However, this approach is unidirectional lacks real-time communication capabilities. To stay updated, the client must continually make requests to the server, resulting in significant overhead. This situation is far from ideal, especially in a group char setting where receiving messages in ral time is crucial. Imagine a situation where the client fetches messages from the server every minute. This would be highly suboptimal.

Therefore, while we can leverage HTTP for certain aspects, we encounter a challenge in implementing polling. Polling involves checking the server's status at regular intervals, potentially every minute, to retrieve new messages. Determining the appropriate intervals for checking these messages poses a dilemma. A one-minute gap might be too long, leading to delayed message delivery. On the other hand, checking every second would generate an excessive number of requests, placing a heavy load on the server and consuming unnecessary resources.

In summary, while HTTP servers its purpose, it presents limitations when it comes to real-time chat applications. The reliance on polling raises questions about choosing the appropriate intervals for message checks, balancing efficiency and responsiveness without overwhelming the system.

WebSocket as the solution

WebSocket on the other hand, is a distinct protocol that facilitates two-way communication between a client and a server, enabling seamless back-and-forth data

transmission. This capability proves highly valuable in scenarios that demand real-time updates, such as chat applications, live streaming apps, or real-time gaming apps.

Establishing a WebSocket connection

Indeed, WebSocket can be seen as an “upgrade” from HTTP in the sense that it starts with a standard HTTP request and then transitions to a WebSocket connection if both the client and server support the protocol. This allows for continuous, bidirectional communication between the client and server.

Major web browsers like Google Chrome, Firebox, and Edge have built-in support for the WebSocket protocol. Additionally, popular server-side web application frameworks such as Node.js, Djando, and ASP.NET also offer support for WebSocket. This widespread support enables developers to leverage the benefits of real-time, two-way communication in their web applications.

Below is an overview for establishing a WebSocket connection.

1. Client Sends a WebSocket Handshake Request: The client will establish a WebSocket connected by initiating a WebSocket handshake request. This is an HTTP Upgrade request with a few special headers.
2. Server Response (Handshake Response): If the server supports the WebSocket protocol and is willing to accept the connection under the conditions the client has requested, it will return with a status code 101, which indicates the server understand the upgrade header field request and indicates which protocol it is switching to.
3. Data Transfer: After the handshake, the client and server send data between each other in real time. This is more efficient than constantly opening and closing new HTTP connections. WebSocket are truly bi-directional, and this way the client will not have to keep checking the server.

By default:

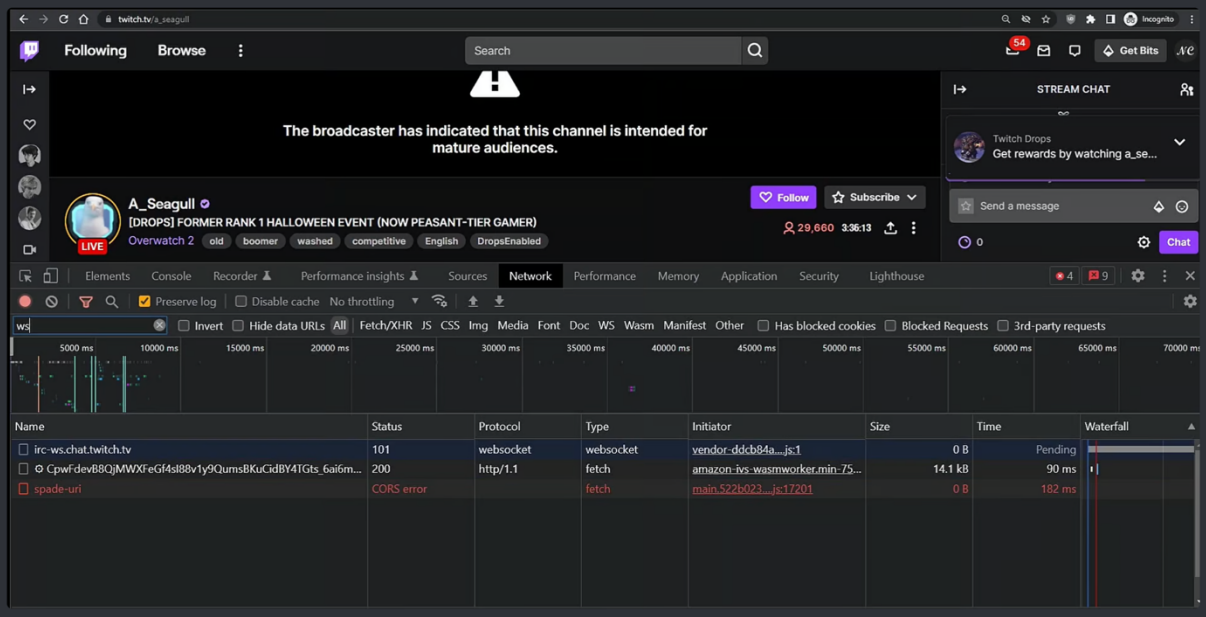
- WebSocket (WS) connections use port 80, similar to HTTP.
- WebSocket Secure (WSS) connections use port 443, similar to HTTPS.

> It is important to note that while devices and browsers may support WebSocket, the network they're connected to must also allow WebSocket connections. Some restrictive firewalls might block WebSocket connections but because it is so ubiquitous and compatible with existing web infrastructure, it is generally well-supported.

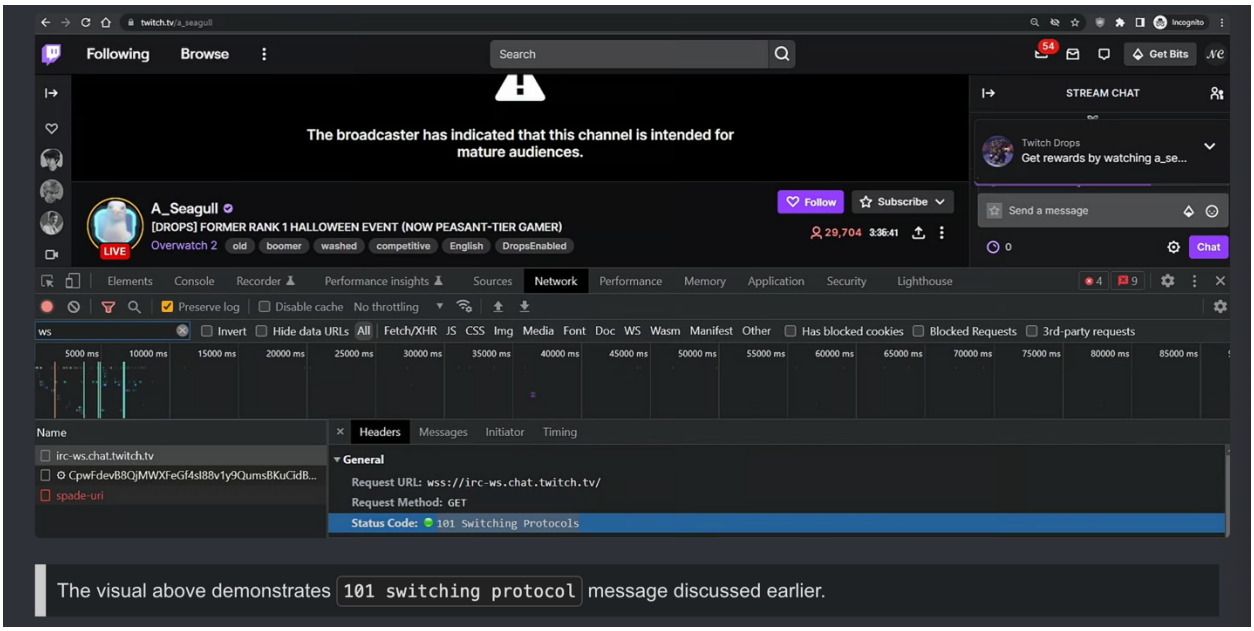
> Recall that there are several versions of HTTP, including HTTP/2. The introduction of HTTP/2 allows for multiplexing, which means that multiple requests in parallel can be initiated over a single TCP connection. However, this is not a perfect replacement to WebSocket as they are still very prevalent to this day.

Example from Twitch.com

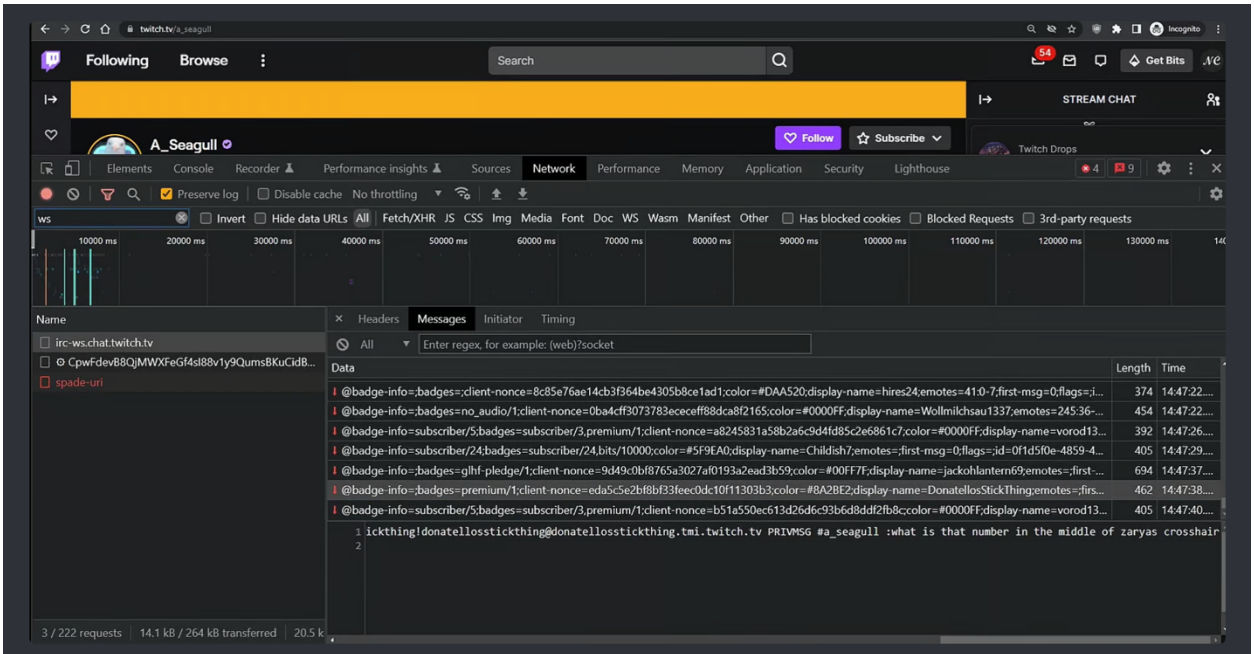
If we look at a live streaming website where chat is being updated in real-time, the protocol that it is highly likely using is WebSocket, and not HTTP. We can dive a little deeper and prove this. We can use the network tab in developer tools for this. In the network tab, filtering for `ws` reveals that the protocol being used is `websocket`, as WebSocket serve this exact purpose.



> The visual above demonstrates that websocket is indeed the protocol being used in real-time chat.



The visual above demonstrates `101 switching protocol` message discussed earlier.



The visual above denotes the messages sent in the chat.