Web Socket

Web Socket

A Web Socket is a

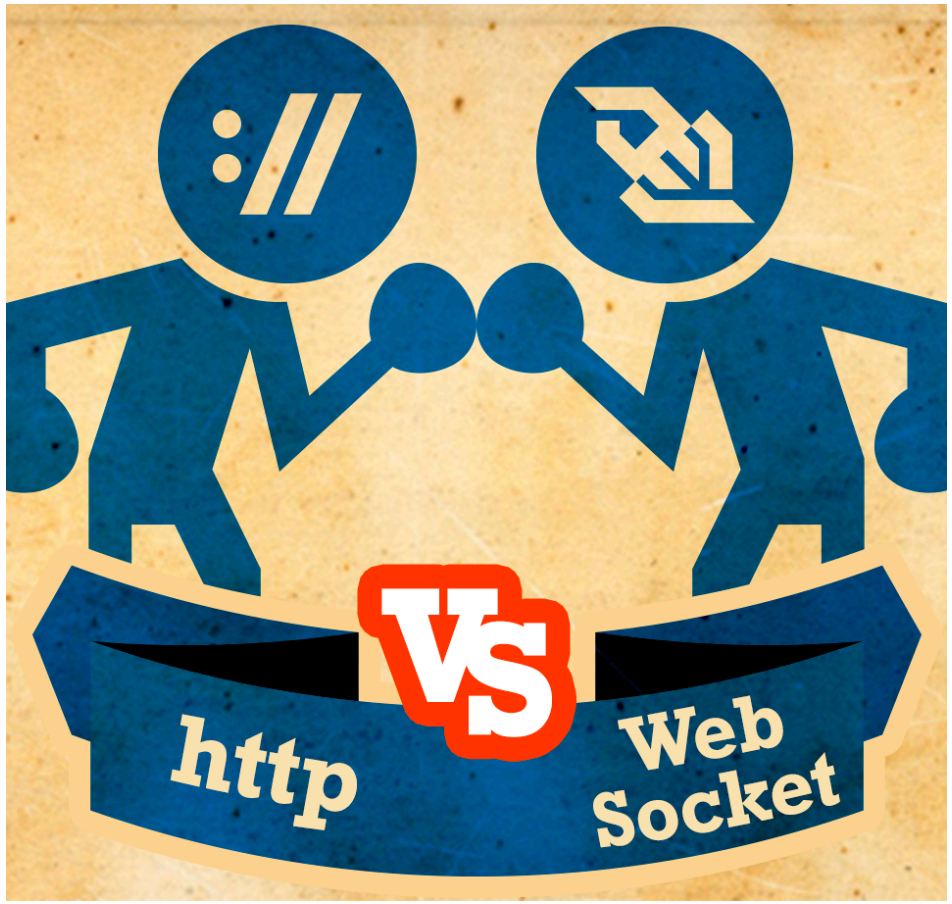
low-latency (real-time),

full-duplex (bidirectional),

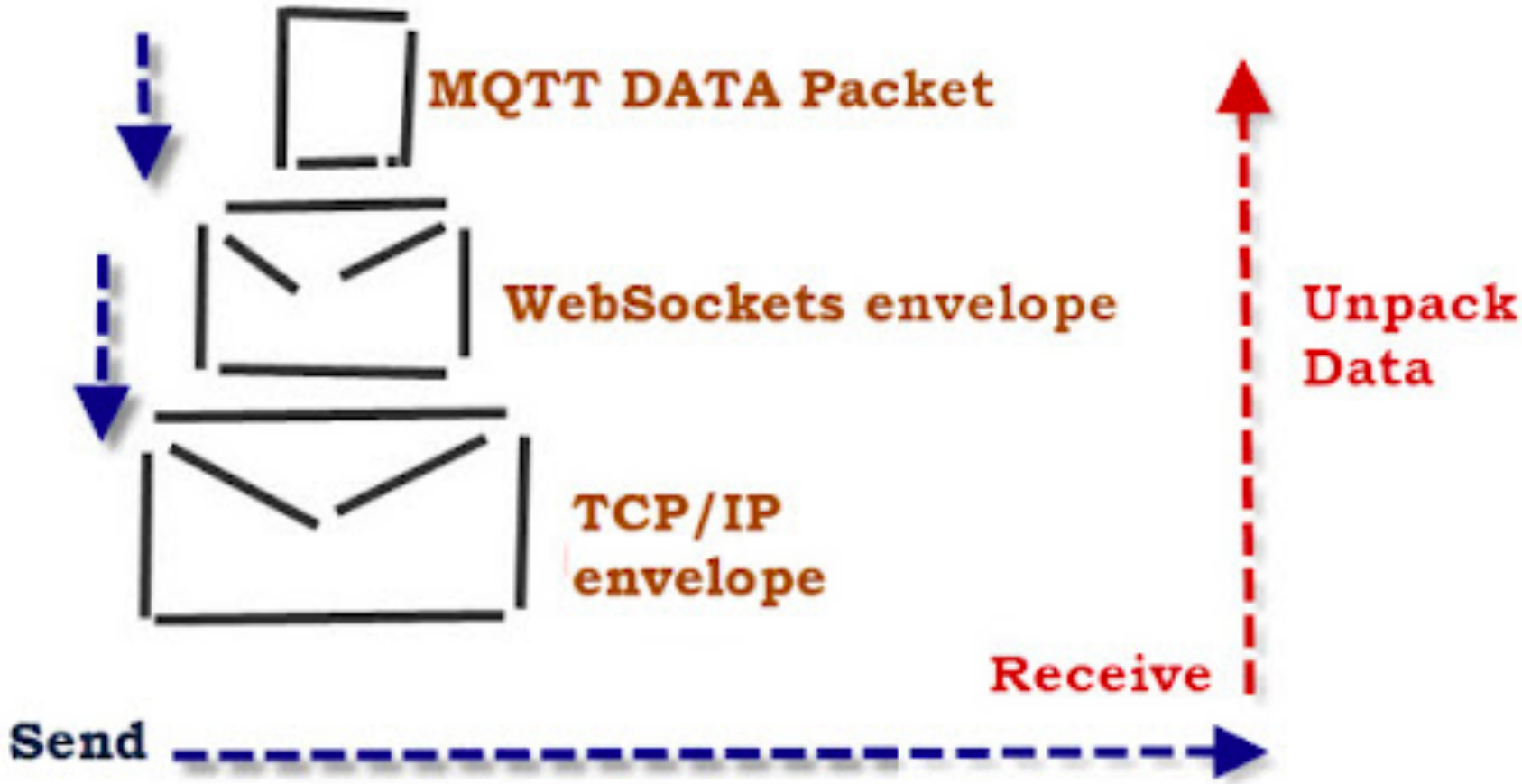
long-running (persistent),

single connection (TCP) between a client and server.

HTTP	WebSocket
Duplex	
Half	Full
Messaging Pattern	
Request-reponse	Bi-directional
Service Push	
Not natively supported. Client polling or streaming download techniques used.	Core feature
Overhead	
Moderate overhead per request/connection.	Moderate overhead to establish & maintain the connection, then minimal overhead per message.
Intermediary/Edge Caching	
Core feature	Not possible
Supported Clients	
Broad support	Modern languages & clients



MQTT Over Websockets Illustration



Web Socket

The full-duplex aspect of Web Sockets allows a server to initiate communication with a client whenever it would like, which is contrary to other protocols such as HTTP and AJAX in which the client must initiate communication.

- WebSocket is a protocol providing full-duplex communication channels over a single TCP connection. Where as, HTTP providing half-duplex communication.
- Information exchange mode of Web Socket is bidirectional. Means, server can push information to the client (which does not allow direct HTTP).

Which one you choose for IoT Application?
Interrupt based application or a polled application

What are the drawbacks of using HTTP for IoT Applications

A Normal web page

- After loading a webpage, the connection between client and server is closed.
- To update a webpage, it needs to reload the entire webpage.

Disadvantage:

- When we want to update a part of a webpage, we have to update entire webpage, producing unnecessary traffic load in the network.
- The server cannot send data to the client if the client does not reload page.
- This is not suitable for real-time monitoring applications.

Webpage containing Ajax

When the client wants to update a part of a webpage, it makes an Ajax request to get data from the server and update the part of the webpage. The connection between client and server is closed after each Ajax request.

Advantage

- Update part of web page without reloading the entire page (reduce traffic load)
  Send data to a server - in the background

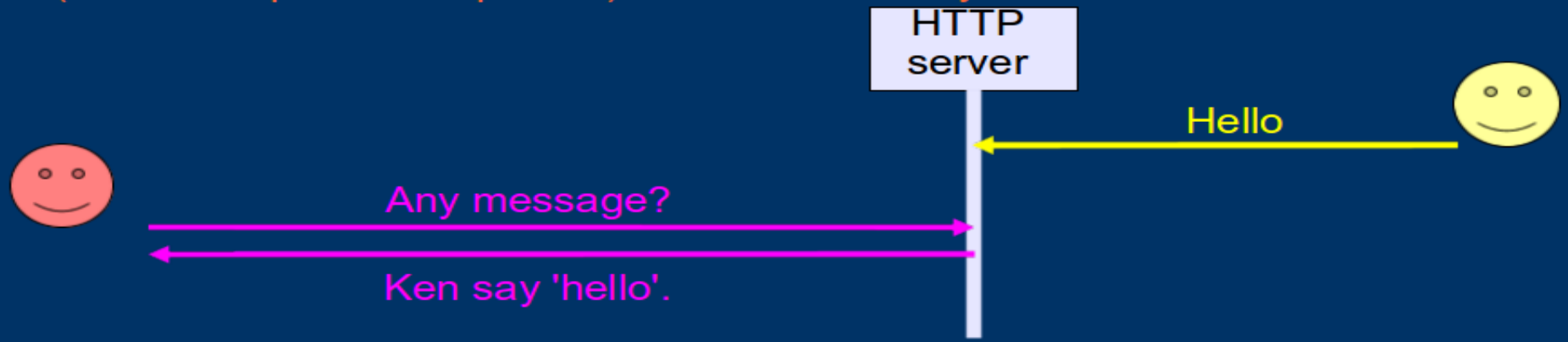
Disadvantage

- Since the connection is closed after each request, the server cannot send data to client if the client does not request data. This is not suitable for real-time monitoring applications.

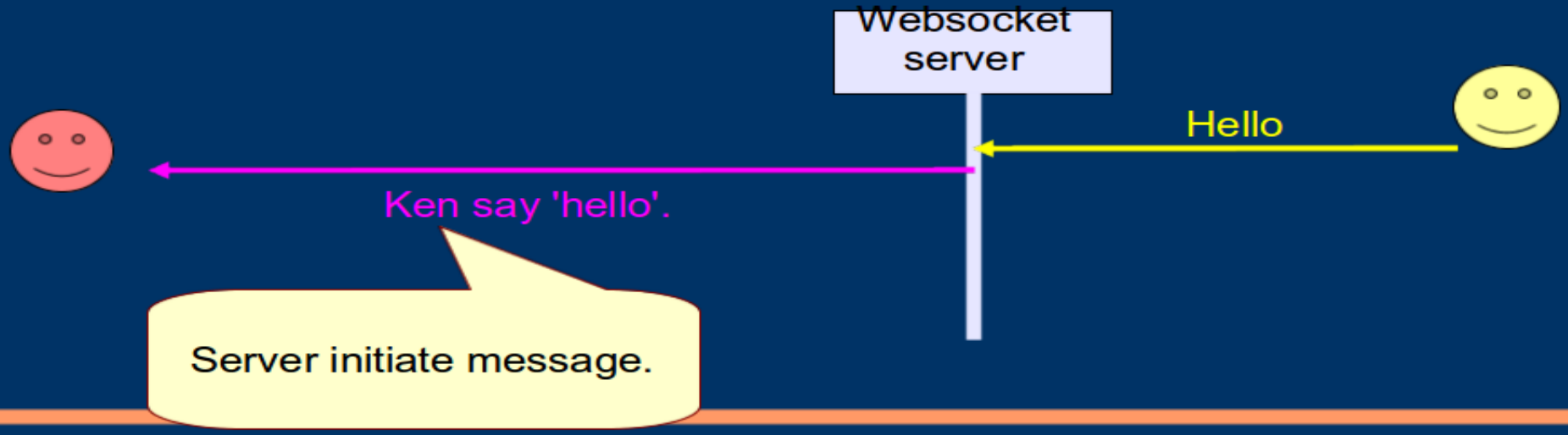
Activity : Explore XHR

Before & After websocket

Before (HTTP : request & response) too traditional way...



After (Websocket : bidirectional)



Before and After Web Sockets

- HTTP based paradigm and trappings where a client must periodically initiate a connection to the server and then place a request to “pull” new data or state information.
- Furthermore, in HTTP, the server cannot “push” new data to the client as states change, but must first wait for a request from the client.

Long Polling

- Technologies that enable the server to send data to the client in the very moment when it knows that new data is available have been around for quite some time. They go by names such as "**Push" or "Comet**".
- One of the most common hacks to create the illusion of a server initiated connection is called **long polling**. With long polling, the client opens an HTTP connection to the server which keeps it open until sending response.
- Whenever the server actually has new data it sends the response
- Long polling work quite well. You use them every day in applications such as Gmail chat.
- Facebook uses Long Polling

How these issues are addresses by Web Sockets ?

How these issues are addresses by Web Sockets ?

- WebSockets takes a different approach by establishing a **persistent, bidirectional** **con**nection that allows the server to update the client application without an initiating request from the client.
- This not only allows for low-latency communication between a server and client, but it also **reduces network traffic** by eliminating the need for a client to send a packet just to request data from the server
- The server can just send data as soon as it’s available or as states have changed.

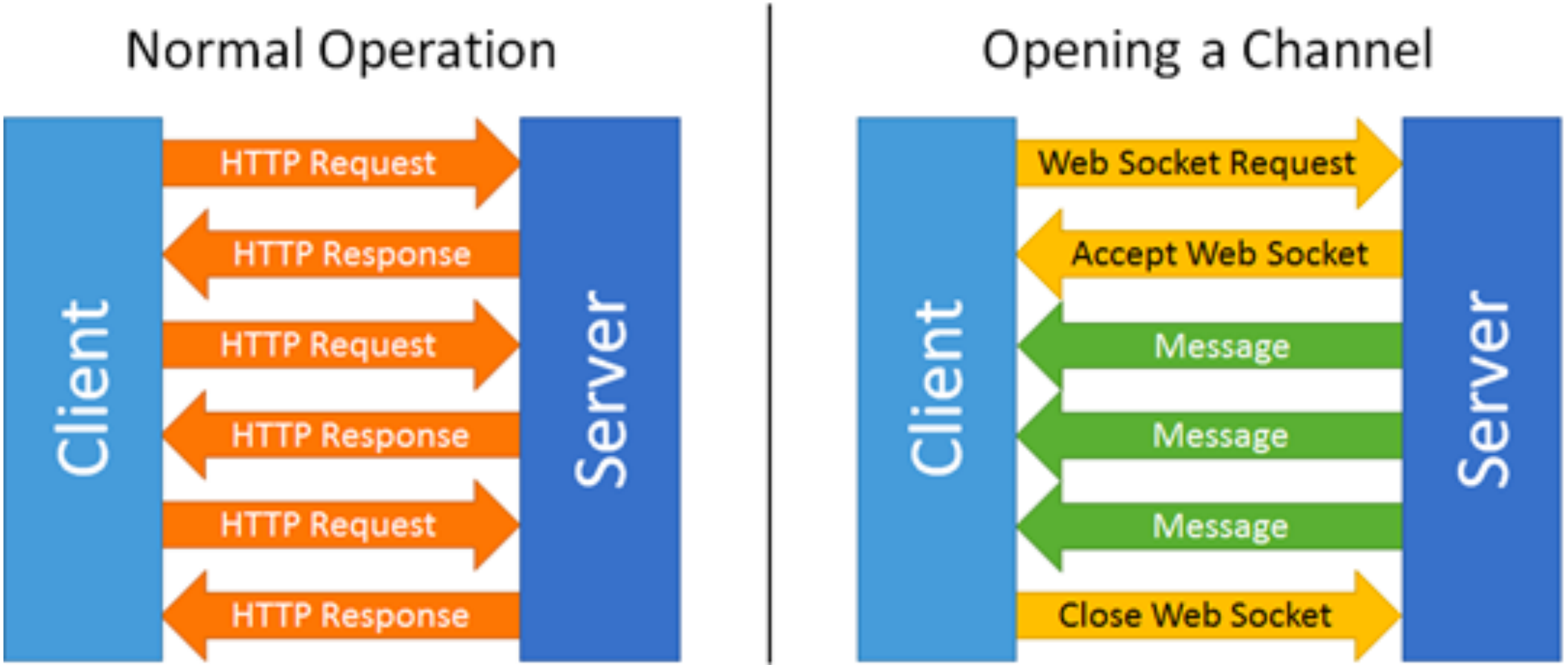
No request necessary.

Interoperability between HTTP and Web Sockets !!!

Interoperability between HTTP and Web Sockets !!!

- The Web Socket protocol was designed to work well with the legacy web infrastructure.
- It uses an HTTP-compatible handshake to establish a connection between the client and server.
- The process to start a Web Socket connection begins with the client sending an HTTP GET with the Web Socket "Upgrade" field.
- After a connection is initialized communication switches to a bidirectional binary protocol unlike the HTTP protocol.

Normal vs. WS communication



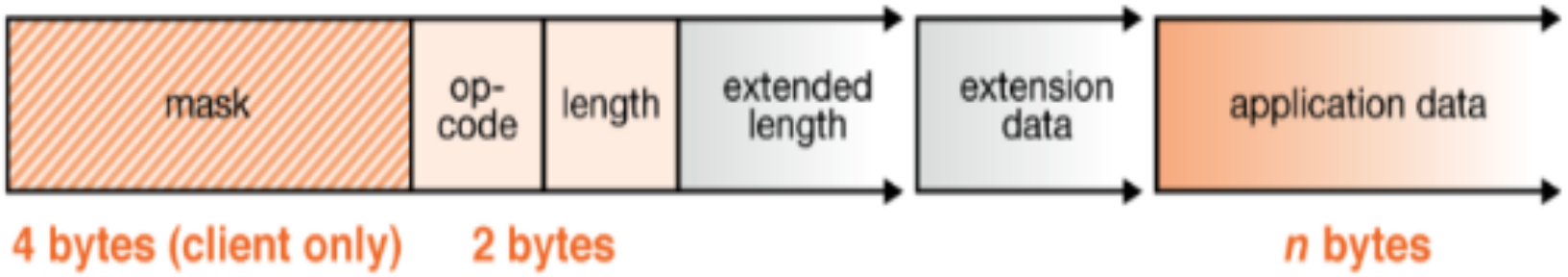
```
GET /INDEX HTTP/1.1
Host: 192.168.1.220
Connection: Upgrade
Upgrade: websocket
Origin: http://192.168.1.220
```

If the server supports WebSockets, it responds with a HTTP UPGRADE response.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
```

WS Packet Structure

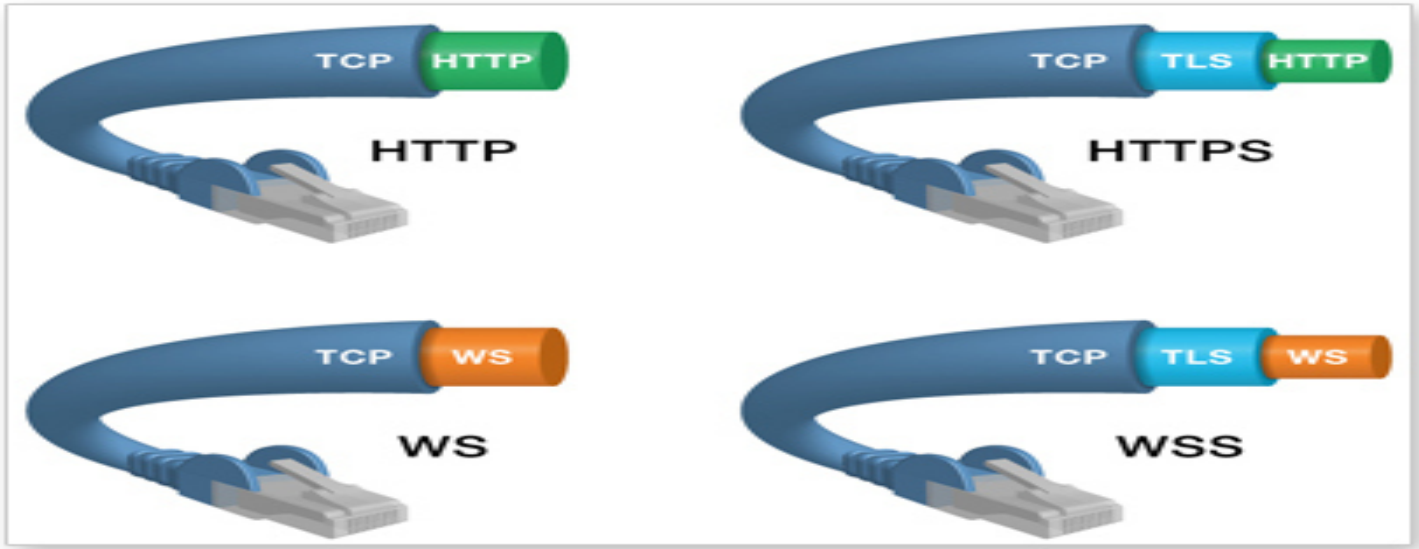
- WebSockets use the same ports as HTTP – 80 for HTTP, and 443 for HTTPS.
- A WebSocket header is only 2 to 6 bytes long.



WS Packet Structure

- HTTP does not define any limit. However most web servers do limit size of headers they accept. For example in Apache default limit is **8KB**, in IIS it's **16K**. Server will return 413 Entity Too Large error if headers size exceeds that limit.
- As a comparison, Google's SPDY research reports HTTP request headers can vary in size from ~200 bytes to over 2KB. Think of all the network traffic and communication latency you can reduce by switching from an HTTP-based implementation to a WebSocket implementation.

WSS

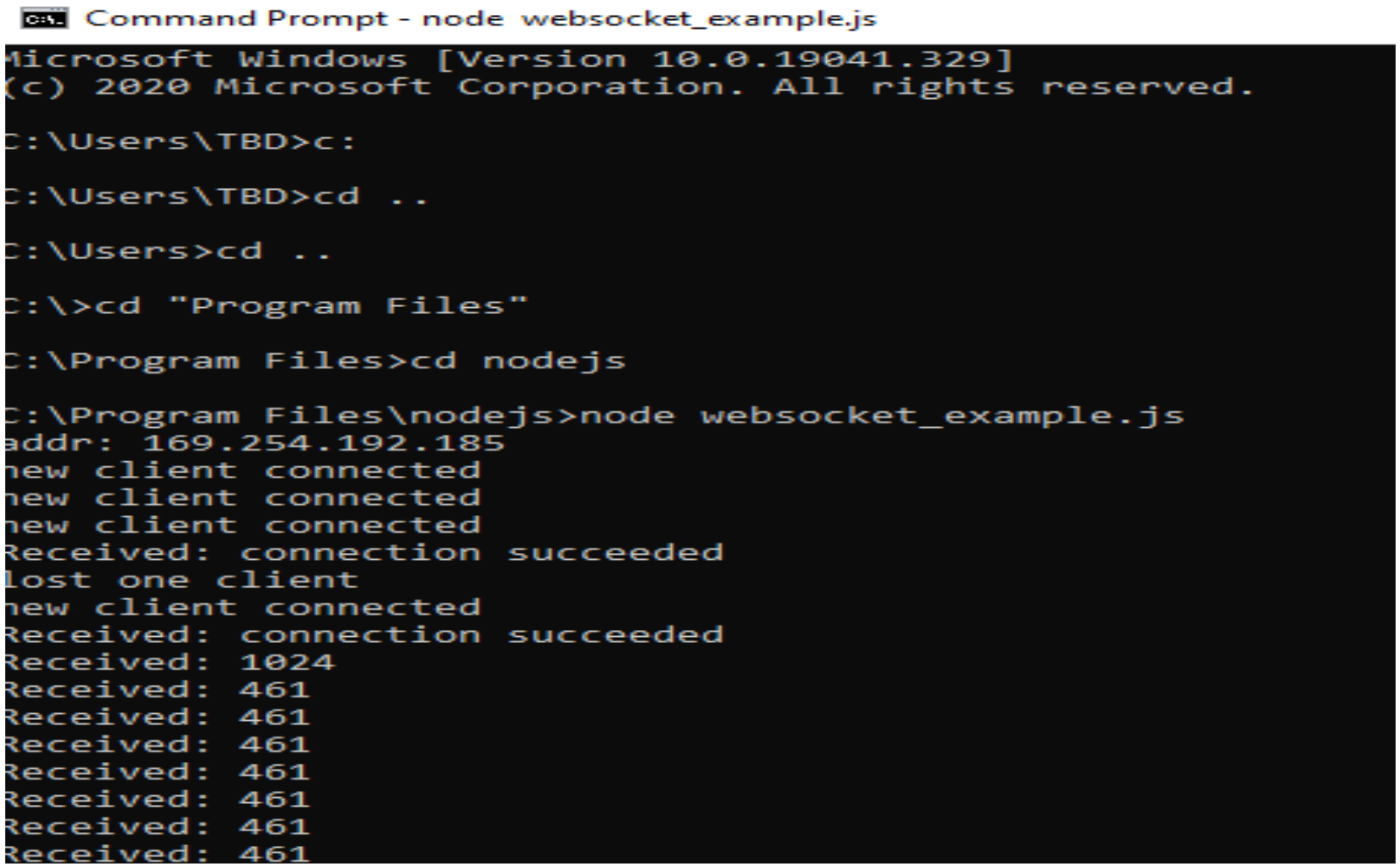


If an encrypted WebSocket connection is used, then the use of Transport Layer Security (TLS) in the WebSocket Secure connection ensures that an HTTP CONNECT command is issued when the browser is configured to use an explicit proxy server. This sets up a tunnel, which provides low-level end-to-end TCP communication through the HTTP proxy, between the Web Socket Secure client and the Web Socket Server.

Demo

Web Socket Server:
https://socket.io/demos/chat/

Demo



Web Socket Server:
192.168.43.67
Port : 3000
(Implemented through node.js)

***Simple* Web Socket Client** – An extension to Chrome is the Web Socket Client

Node MCU with IP Address 192.168.43.123 is the actual end device with Sensor to push data to Web Socket server.

Reference: [Real-time Websocket Connection between Sensor Reading ESP8266 and Node.JS Server] From ESP8266 Shop