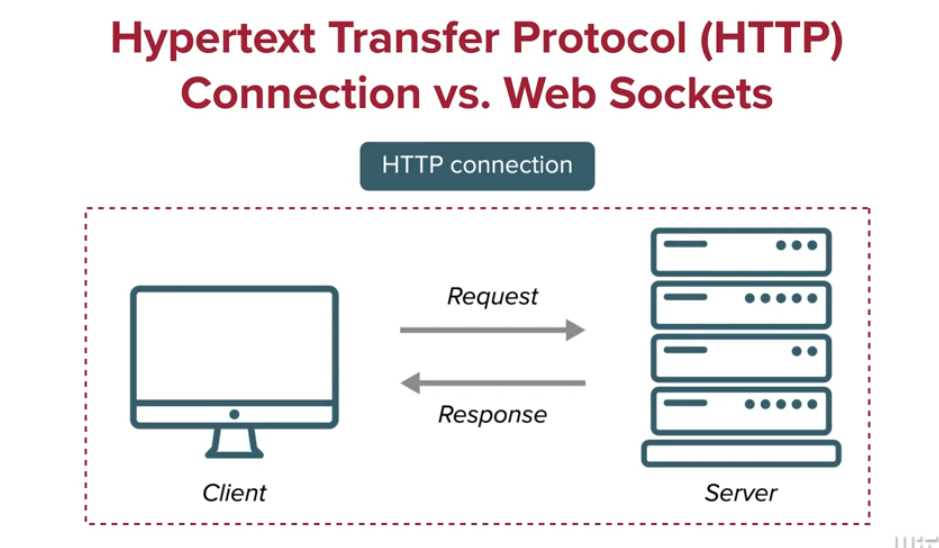
**MIT xPRO Data Engineering Certificate**

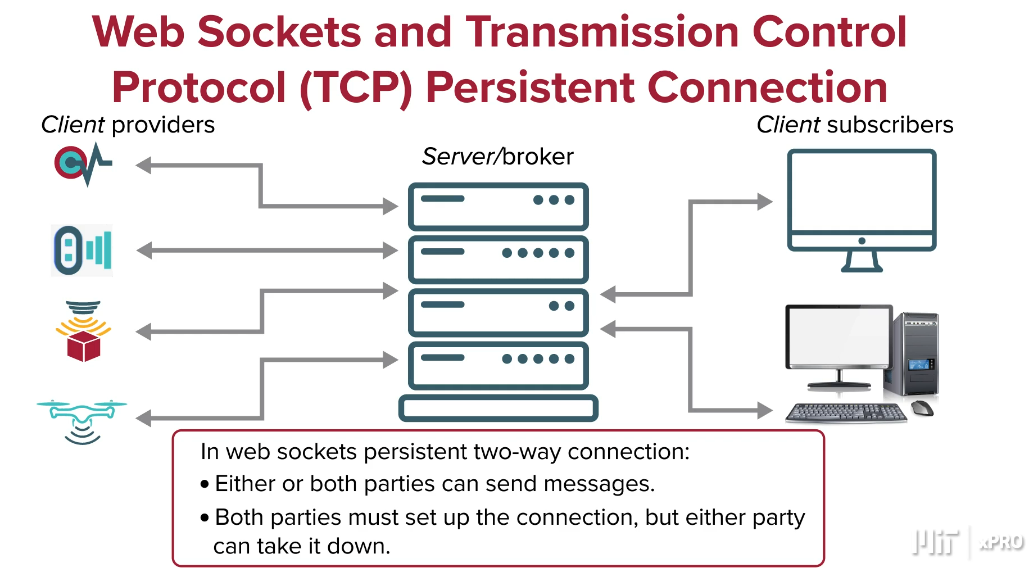
**WebSockets**

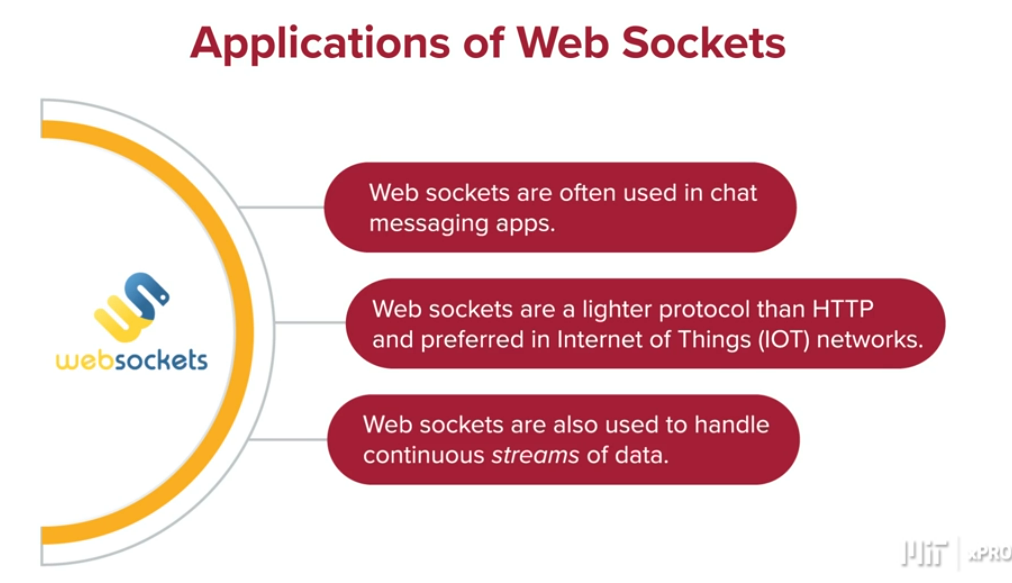
Web sockets are a two-way computer communication protocol over TCP networks. Web sockets can be used to handle large *streams* of data. Both HTTP and web sockets run over TCP networks, and web sockets offer a persistent connection, whereas HTTP is a request - response protocol. Web sockets are often used with streaming data such as with IoT sensors. (Another communication protocol for streaming is MQQT).

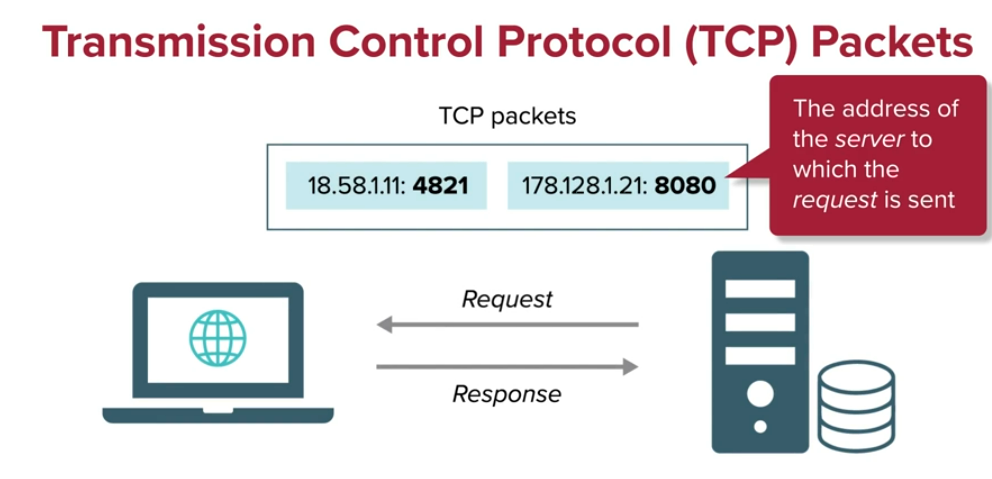


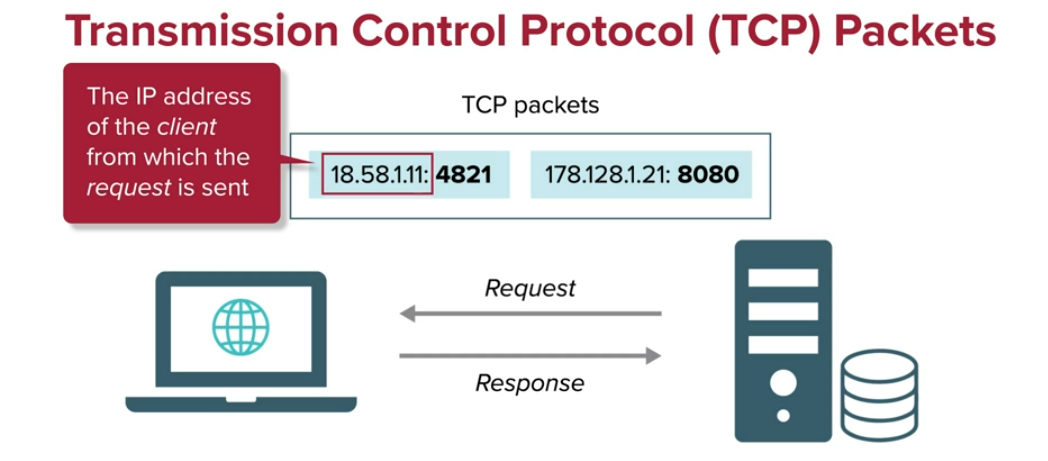


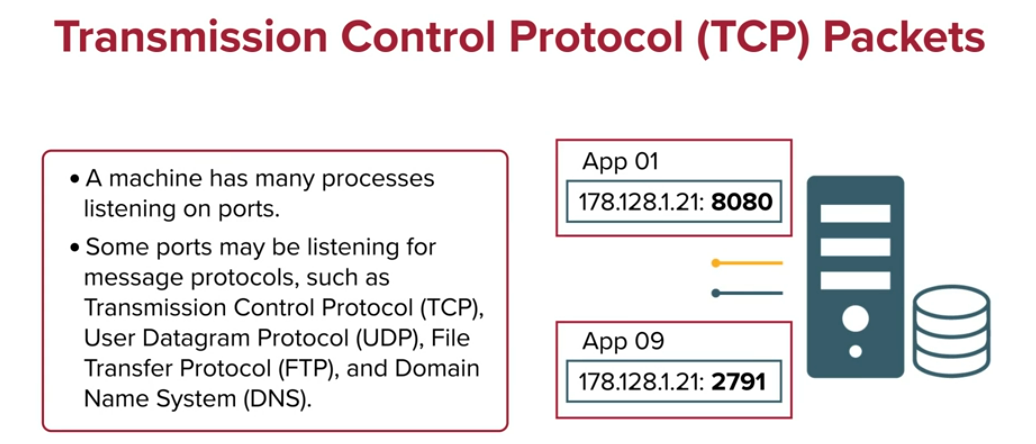


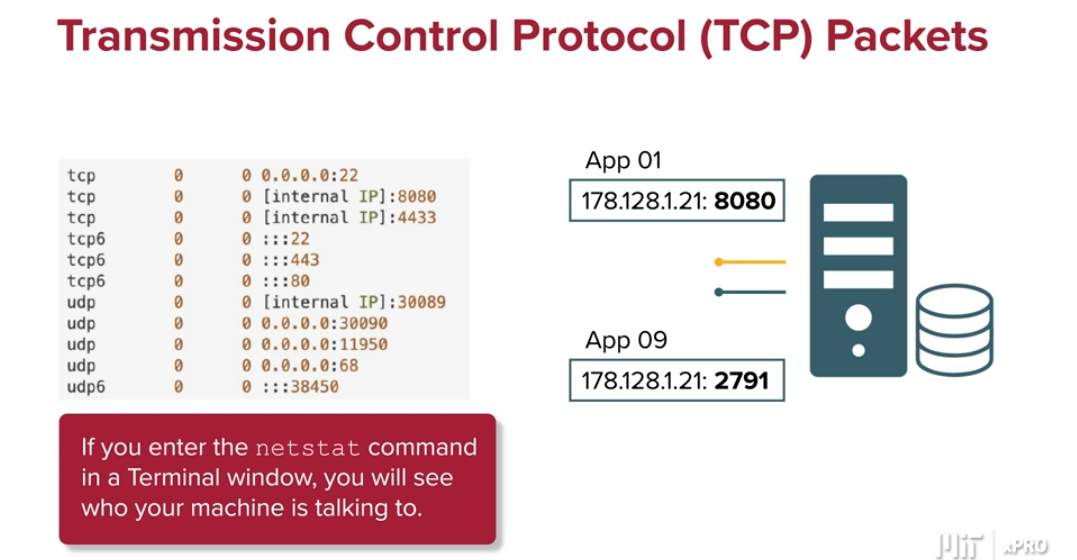


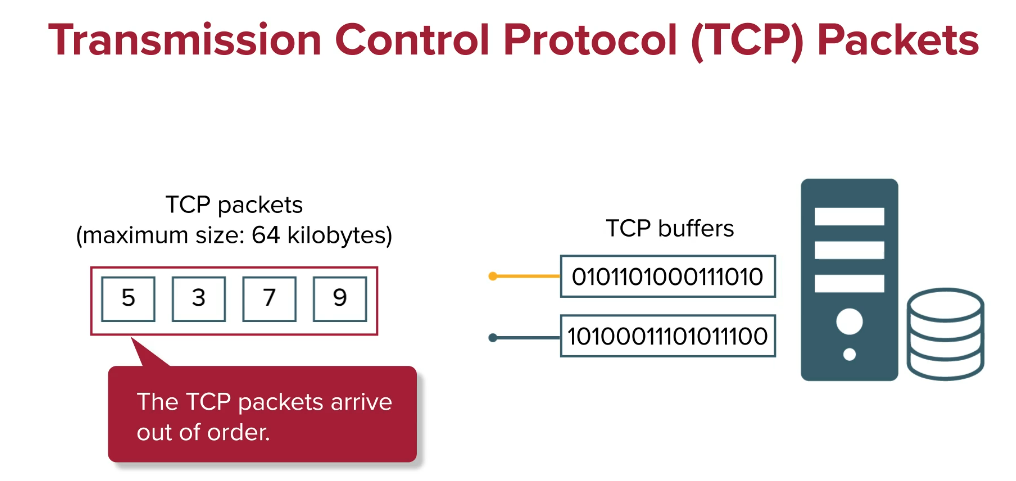


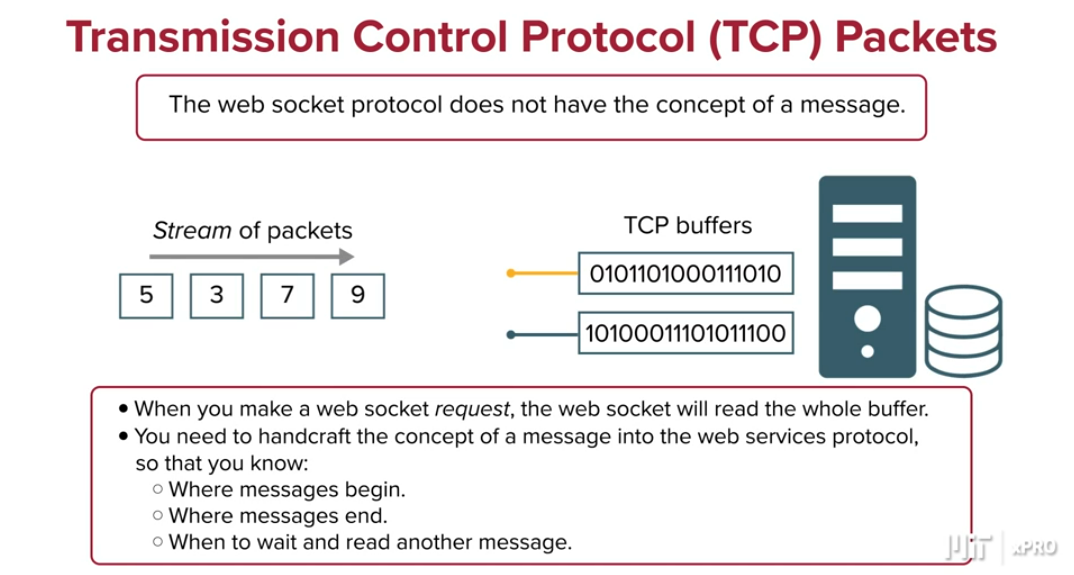


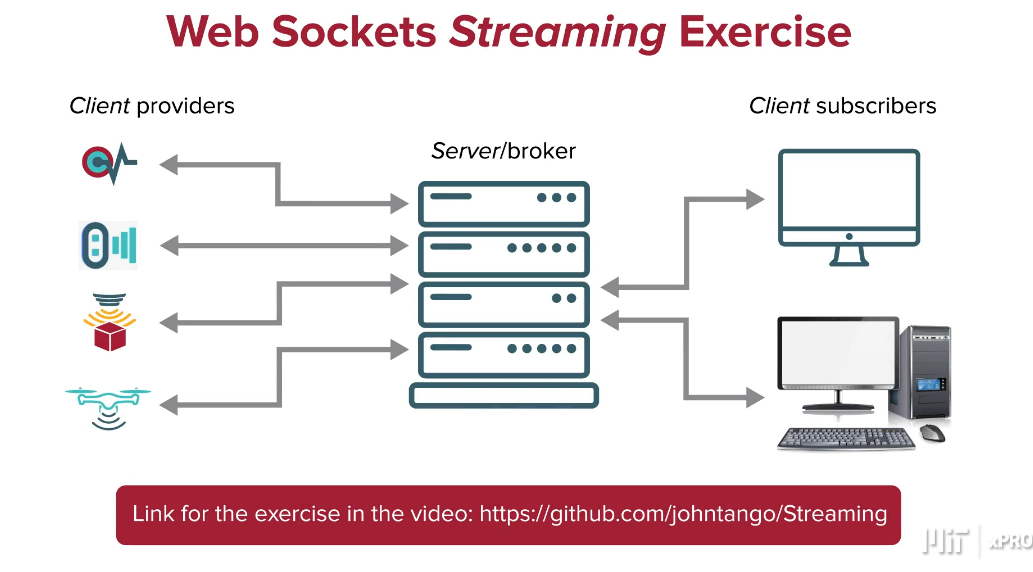


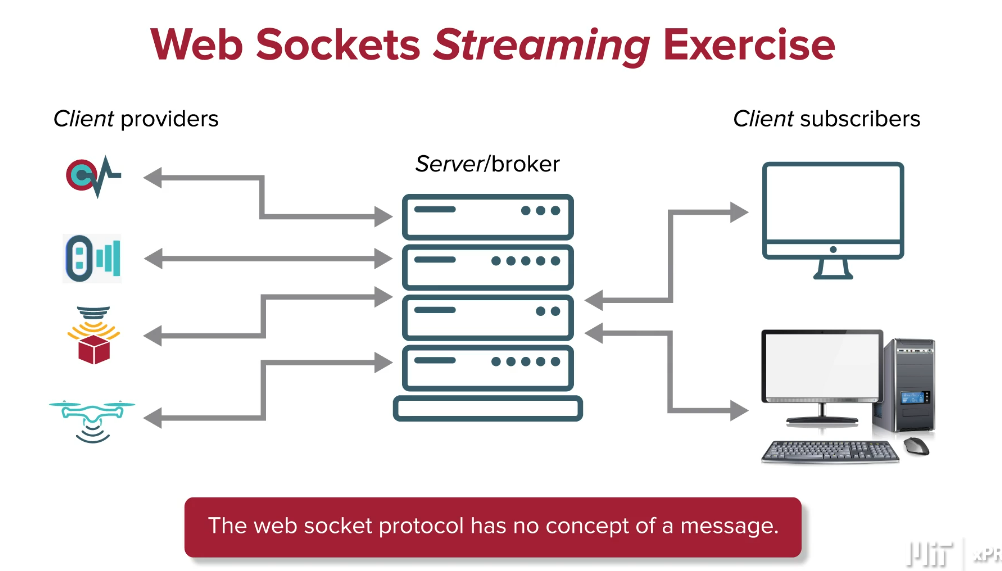


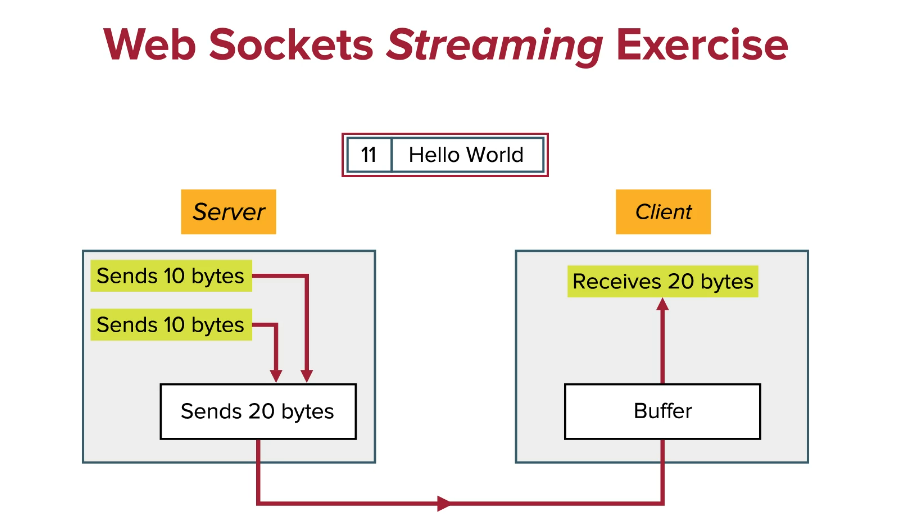




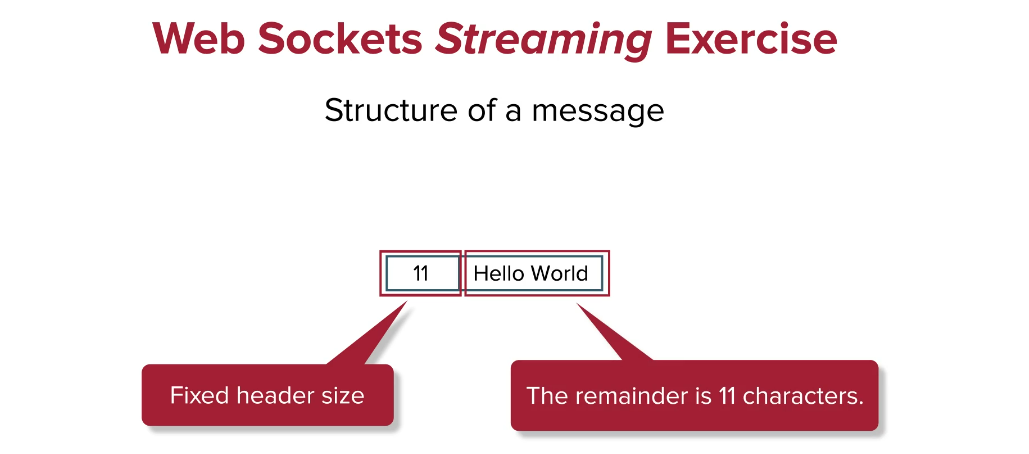


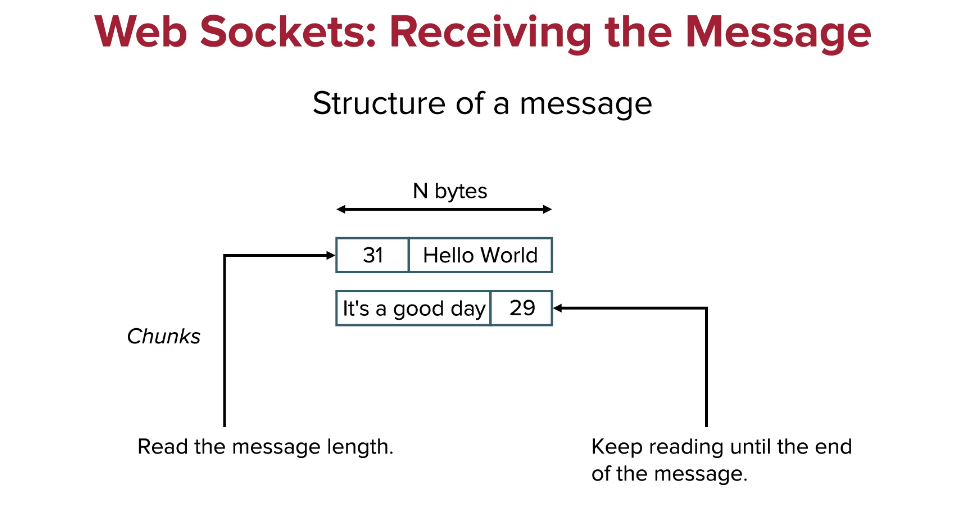


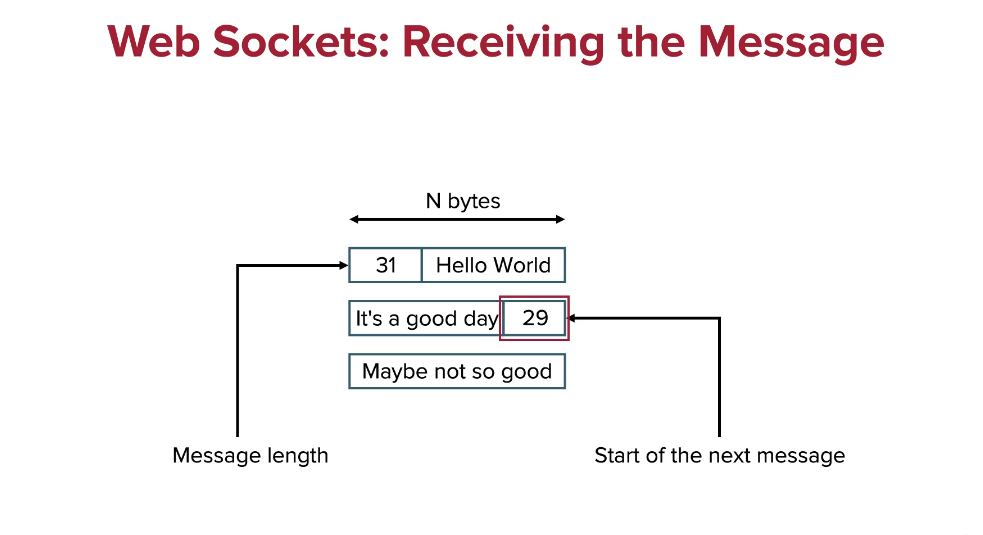




In the example above we need instructions about how to treat the bytes and what is the intended message. The problem is the end of a message may not be at the end of a chunk.

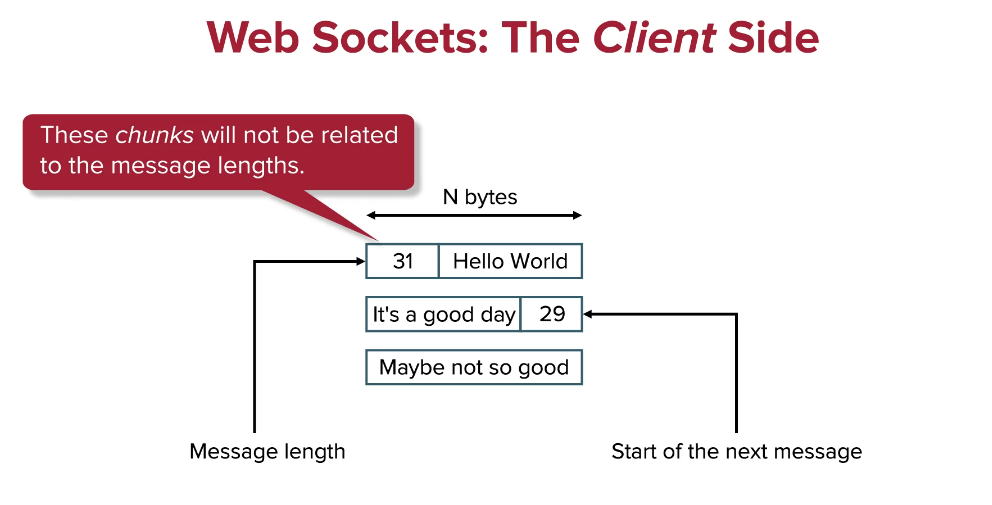


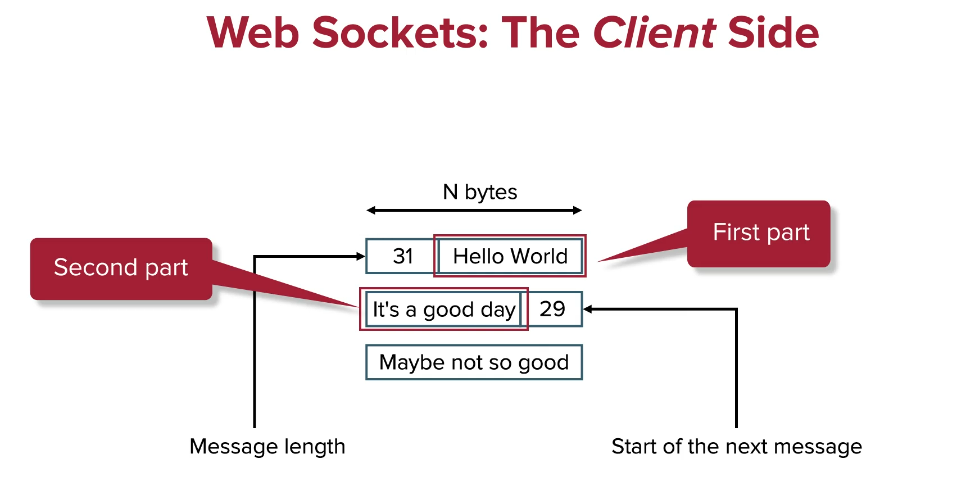


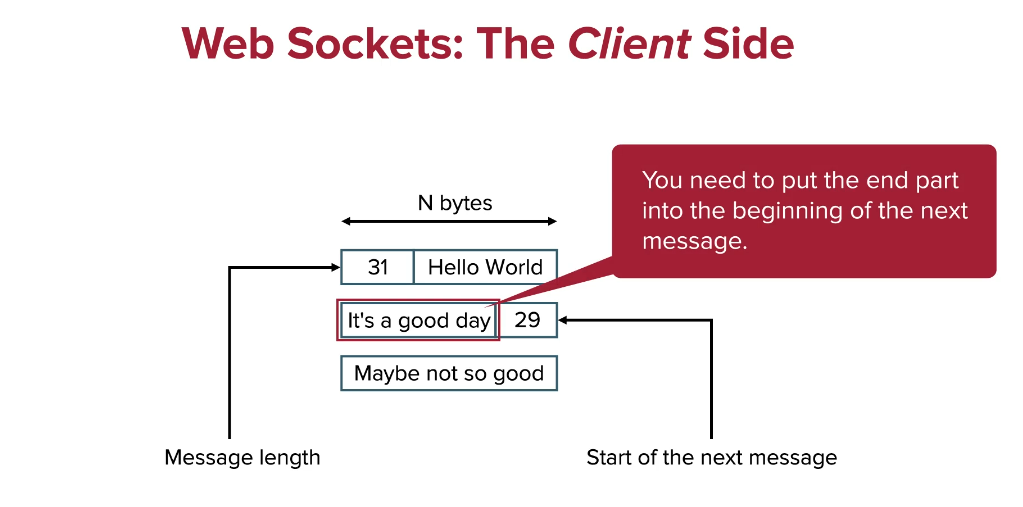


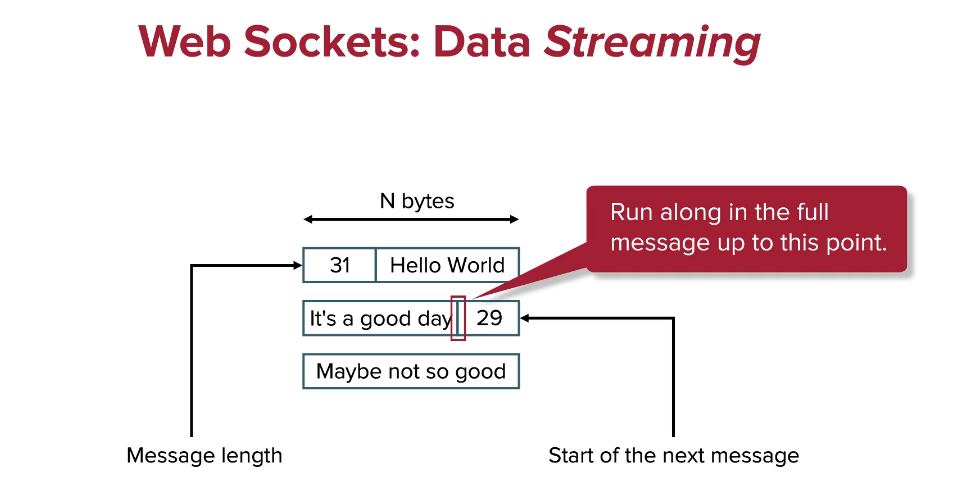
<https://github.com/johntango/Streaming/blob/master/streamServer.py>







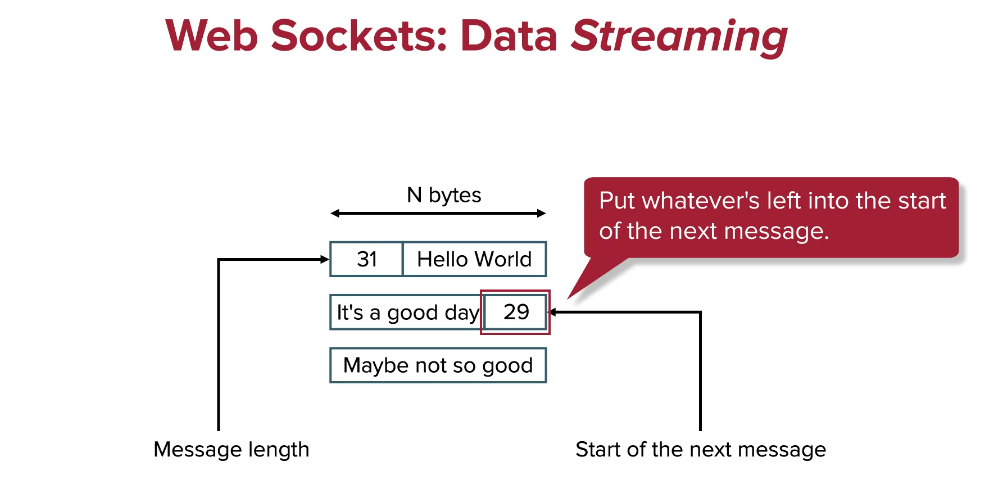


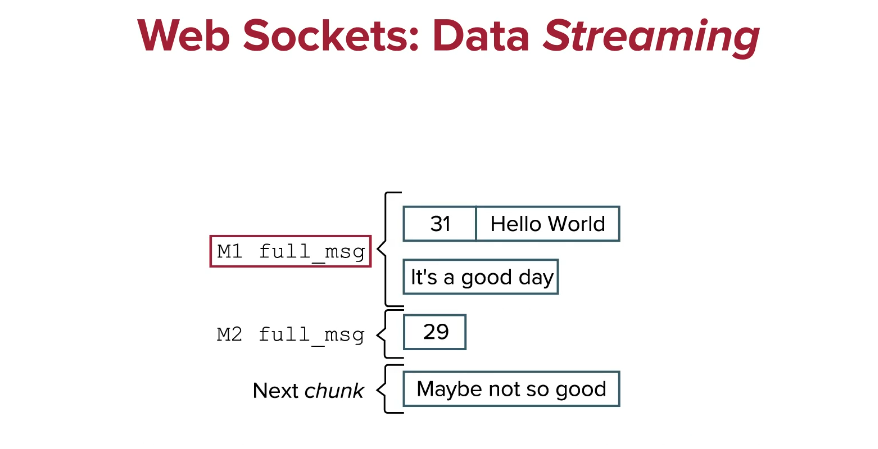


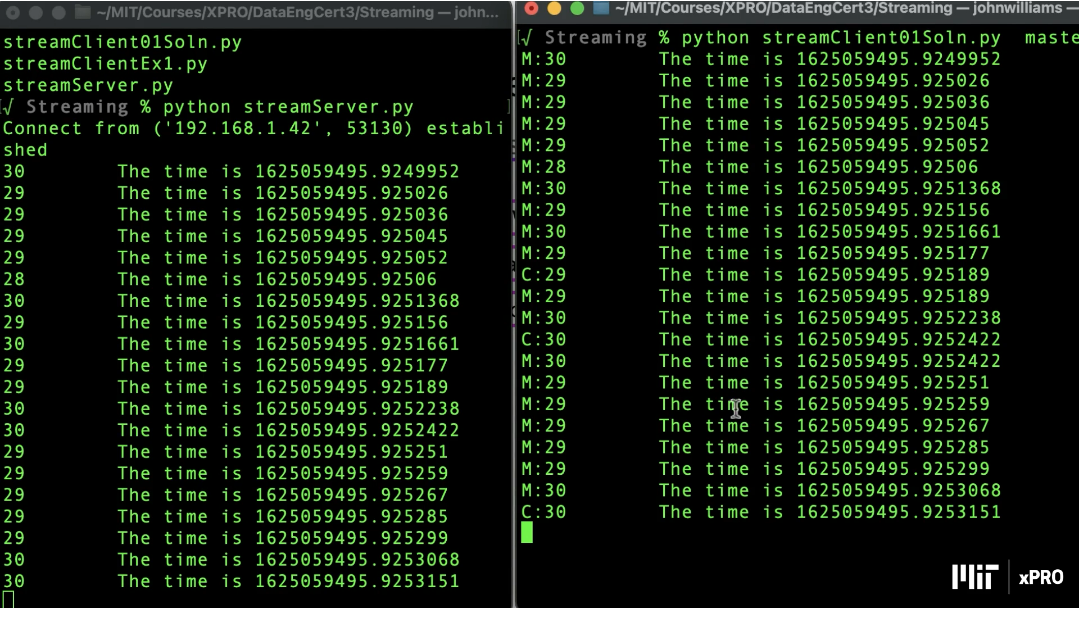
<https://github.com/johntango/Streaming/blob/master/streamClientEx1.py>

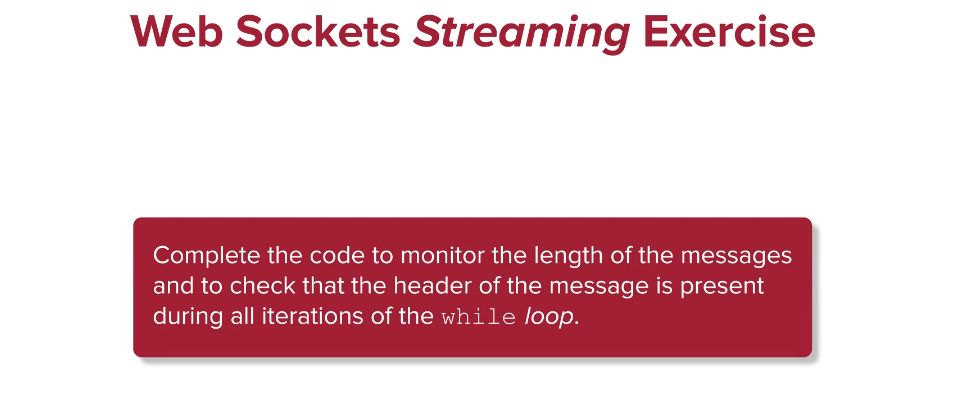


To avoid port problems, break the program first in the client, then in the server.



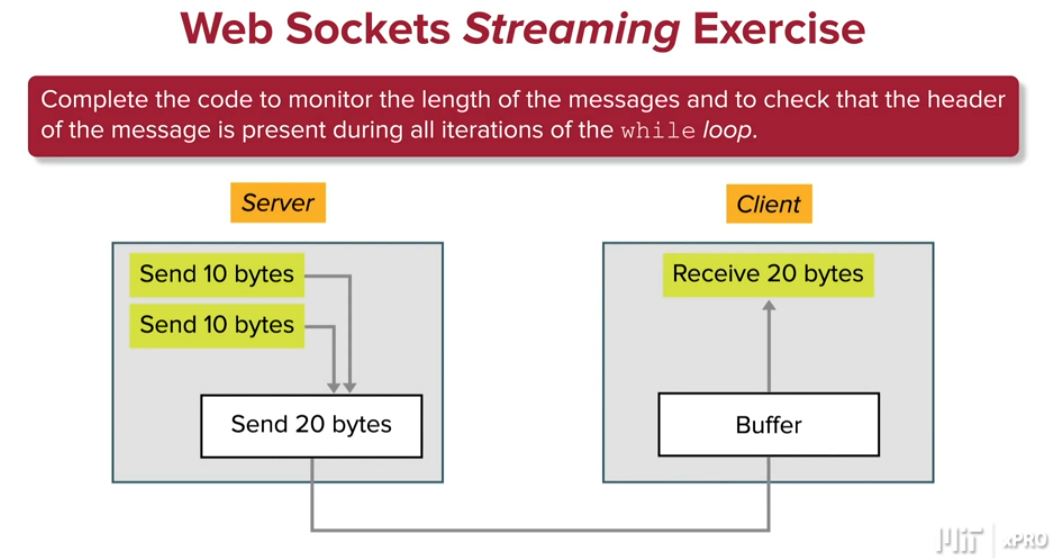






<https://github.com/johntango/Streaming/blob/master/streamClient01Soln.py>





A web socket is a steady connection between a *client* and a *server*. Web sockets provide a bidirectional communications channel that operates over HTTP networks through a single transmission control protocol (TCP) or IP socket connection (Sookocheff 2019). In this mini-lesson, you will learn how web sockets work.

**Why Are Web Sockets Needed?**

Web sockets were created in response to the limitations of HTTP-based technology. “With HTTP, a *client* *requests* a resource, and the *server* responds with the requested data. HTTP is a strictly unidirectional protocol — any data sent from the *server* to the *client* must first be requested by the *client*” (Sookocheff 2019). The unilateral communication of HTTP limits the functionality of this method.

However, a workaround for this limitation does exist. Long polling allows a *client* to make an HTTP *request* with a long timeout period so that the *client* connection can be held to allow the *server* more time to *push* data to the *client* (Sookocheff 2019). The data is *pushed* to the *client* as soon as it becomes available, enabling real-time communications. Long polling is effective, but works most efficiently when communications from the *server* are infrequent. A downside of long polling is that “resources on the *server* are tied up throughout the length of the long poll, even when no data is available to send” (Sookocheff 2019).

**How Do Web Sockets Work?**

Web sockets allow developers to build real-time applications by avoiding these long polling issues that exist in the HTTP *request*-response handshake. Web sockets do use HTTP as a communication method. However, instead of being a one-sided, one-time connection, web sockets keep the TCP connection open after the initial HTTP response is initiated by the *client* and received by the *server* Sookocheff 2019). This allows the web socket connection between *client* and *server* to remain open as a communication portal. Data and communications continue through the web socket until both parties decide that the connection should be closed (Sookocheff 2019). Within the transmission control protocol (TCP), data is exchanged in *packets* which are small *chunks* of data delivered between the *server* and the *client*.

There are some important points to understand about web socket connections. Because web sockets do not follow HTTP protocols, they follow the ws:// or wss:// extension instead of the http:// or https:// extension (Sookocheff 2019). The rest of the web socket URL is structured the same as an HTTP URL and contains “a host, port, path, and any *query* parameters” (Sookocheff 2019). For example, a web socket URL may look like the following examples:

| "ws:" "//" host [ ":" port ] path [ "?" query ]  "wss:" "//" host [ ":" port ] path [ "?" query ] |
| --- |

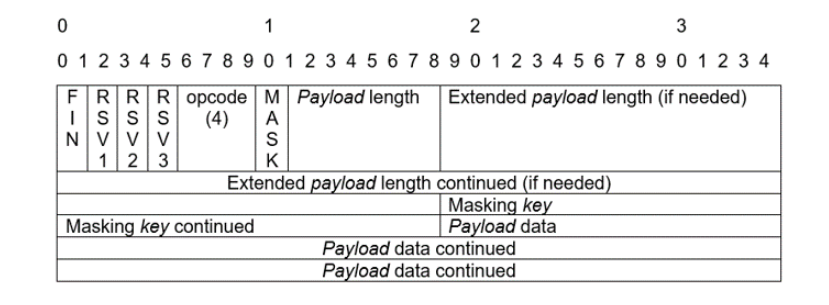
This extension is the standard web socket communication design to establish steady connection between *client* and *server*. Additionally, web sockets allow for connections between a *server* and a *client* to be made between URLs that have the same extension.

**Web Socket Protocol**

Web socket protocol communicates using frames, which include any information that is needed as well as the message that you intend to send. If a *dataframe* is too large, it is divided into smaller *chunks* to allow it to be communicated over a web socket connection. The data communication always includes a frame type, a *payload* length, and a data portion (Sookocheff 2019).

Frames include different pieces of information and “in its most basic form the web socket protocol has three non-control frames and three control frames” (Jameson 2017). Non-control frames explain whether the information being communicated includes raw bytes, encoded bytes, or is a continuation of a message sent previously (Jameson 2017). Control frames can indicate whether a *client* or *server* wants to close the connection or is responding to a close *request* from the other party. Control frames can also consist of ping and pongs, which are operation codes that you will learn about below.

The image below shows an example of the web socket framing protocol:



Every frame has an operation code that determines how to interpret and use the requested frame’s *payload* data. Listed below are some opcode values and descriptions which enable you to interpret what operations are being sent between the *client* and the *server* through the web socket connection (Fette & Melnikov 2011):

**Opcode Values**

| **Operation Code (Opcode) Value** | **Description** |
| --- | --- |
| 0x00 | This opcode frame “continues the *payload* from the previous frame” (Sookocheff 2019). |
| 0x01 | This opcode describes a text frame, which is decoded by the *server*. |
| 0x02 | This opcode describes a binary frame, which is delivered to the *client* without being edited by the *server*. |
| 0x03-0x07 | This opcode saves the frame to be used at a later time. |
| 0x08 | This opcode is a close frame that indicates that the *client* or the *server* wants to close the connection. |
| 0x09 | This opcode is a ping frame that “serves as a heartbeat mechanism ensuring the connection is still alive” between the *client* and the *server* (Sookocheff 2019). The receiver of a ping frame must respond with a pong frame to confirm the connection still exists. |
| 0x0a | This opcode is a pong frame that “serves as a heartbeat mechanism ensuring the connection is still alive” between the *client* and the *server* (Sookocheff 2019). The receiver of a pong frame must respond with a ping frame to confirm the connection still exists. |
| 0x0b-0x0f | This opcode saves the frame to be used at a later time. |

**Closing a Web Socket**

Closing a web socket connection always initiates with a message from the *client* or the *server* in which the frame contains a close operation code, opcode 0x08, and a description in the body of the message that explains the reason for closing the connection. Once the close communication is established and acknowledged between both parties, then the web socket connection can be stopped and closed.

Now you have an understanding of web sockets and how they differ from HTTP protocol.