LAU
الجامعة اللبنانية الأميركية
Lebanese American University

# CSC443: Web Programming

Lecture 20: Web Sockets

Haidar M. Harmanani

## HTML5 WebSocket

- Standardized by IETF in 2011.
- Supported by most major browsers including Google Chrome,
- Internet Explorer, Firefox, Safari and Opera
- Basically, it allows the browser to create a **socket client (inside a web page)** that connects and talks to a WebSocket server.

# Why WebSockets?

- HTTP is half-duplex
- HTTP has too much overhead
- Ajax doesn't help



# HTML5 WebSocket protocol

- WebSockets is a big step forward for HTML
  - Two-way communication without expensive/annoying client-side polling
  - Ideal for low-latency persistent connections
    - e.g. for real-time Web applications
    - Only requires 2 bytes of overhead per message
  - Requires server-side support
    - e.g. via JavaScript or Python on the Web server

# Why WebSocket?

- Low latency client-sever and server-client connections
- Legacy HTTP
  - Connect, send a request, get a response, disconnect. When sending the next request need to establish the connection again.
  - All communications are initiated by the client, the server cannot proactively send something to the client.
- WebSocket
  - full-duplex communication channel over a single always-on TCP connection



# Why WebSocket?

# WebSocket Uses

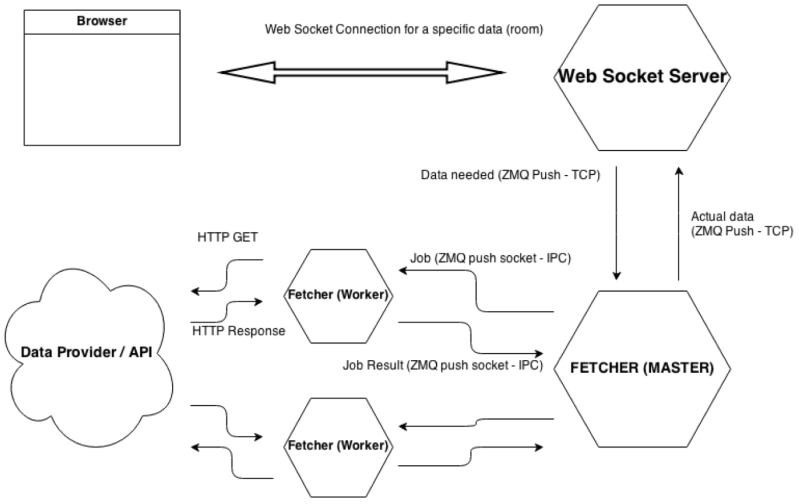
- Real-time updates (sports, finance)
- Games
- Collaboration & Education
- Feeds & rich communication
- Location-based services
- Services based on real-time APIs
- User Monitoring & Logging
- Stock monitors



# How Does it Work?

- Client and server communicate by sending messages to each other.
- Server can easily broadcast message to all its clients.

# How Does it Work?






# How Does it Work?

- It starts off as a HTTP request, which indicates that it wants to "upgrade" to the WebSocket protocol
- If the server can understand it, then the http connection is switched into a WebSocket connection
- Once connected, data is transmitted (bidirectionally) via *frames*

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

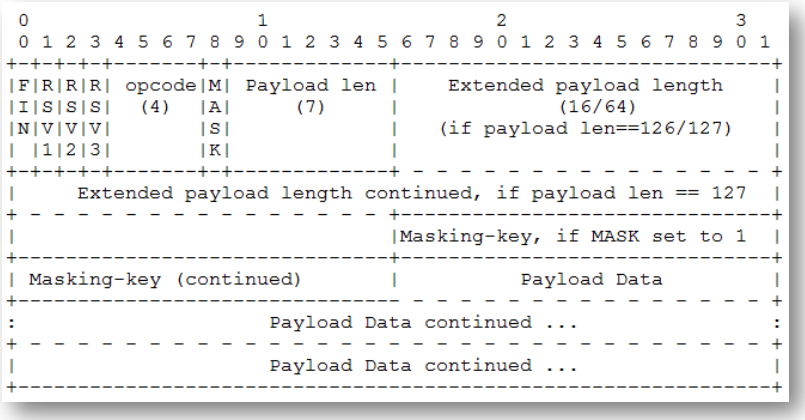
***request***

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

***response***

# WebSocket framing

- Data is transmitted via *frames*, which may be sent by client or server at any time






# WebSocket masking

- When the **`MASK`** bit is set, a 32-bit mask is used to transform each payload byte
  - Masking and unmasking algorithms are identical because of **`XOR`** operation:

```
foreach index j in payload:
  data[j] = data[j] XOR mask[j MOD 4]
```

  - Security feature or too much overhead?

# Using WebSockets

- On the server side, we need a server that supports WebSockets
  - **`mod_pywebsocket`**: a Python-based module for Apache
  - **`Netty`**: a Java network framework that includes WebSocket support
  - **`node.js`**: a server-side JavaScript framework that supports WebSocket server implementation



# Using WebSockets

- On the server side, we need a server that supports WebSockets
  - Node.js
    - Socket.IO
    - Engine.IO
      - Lower level
  - C#/.NET (IIS 8 ASP.NET 4.5)
    - XSockets.NET
    - Fleck
  - Java
    - Atmosphere
  - Ruby
    - EM-WebSocket

# Using WebSocket

- Server side
  - Recommend to use ws, the WebSocket server implementation for Node.js
    - https://github.com/websockets/ws



# Using WebSocket: Server Side

```
1
2  var WebSocketServer = require('ws').Server
3     ,wss = new WebSocketServer({port: 10000});
4
5  wss.on('close', function() {
6      console.log('disconnected');
7  });
8
9  wss.on('connection', function(ws) {
10     ws.on('message', function(message) {
11         console.log(message);
12         ws.send(message);
13     });
14 });
15
```

ws is new socket for the new connection

# Using WebSocket: Client Side

```
socket = new WebSocket("ws://localhost:8000");
socket.onopen = function (event) {}

socket.onclose = function (event) {
    event.code
    event.reason
    event.wasClean
}
socket.onmessage = function (event) {
    event.data
}
```

Event Driven!



# Using WebSocket

STATUS

readyState
- CONNECTING ( = 0 )
- OPEN ( = 1 )
- CLOSE ( = 2 )
- CLOSING

onopen

onmessage

onclose

onerror

METHODS

close(code, reason)

send(data)
- String
- ArrayBuffer
- Blob

# WebSockets client code

- Open a WebSockets connection to a specified WebSockets server:

```
// JavaScript client-side code example

// Create new WebSocket object
var url = "ws://servername.edu:8787/path";
var socket = new WebSocket( url );

// socket.readyState property indicates
// the current status of the connection
// (see next slide)
```

Use "wss" for a secure connection



# WebSockets readyState

- The readyState property is a number:
  - **0** (**`socket.CONNECTING`**): client is establishing the connection to the server
  - **1** (**`socket.OPEN`**): client is connected; use **`send()`** and an **`onmessage`** event handler to communicate
  - **2** (**`socket.CLOSING`**): the connection is in the process of being closed
  - **3** (**`socket.CLOSED`**): connection is closed

# WebSockets client code

- WebSockets API is event-driven (**onopen**, **onmessage**, **onerror**, **onclose**):

```
// JavaScript client-side code example (continued)

// Opening message (sent once)
var msg = "ME IS goodatenglish";
socket.onopen = function() { socket.send( msg ); }

// Listen for incoming message from server
socket.onmessage = function( msg ) {
  alert( "Server says: " + msg );
}
```



# To Probe Further …

- https://www.websocket.org/
- https://github.com/websockets/ws
- https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API