WebSockets
Yeah, about that whole “stateless” thing…

WebSockets
Sometimes you just need a constant connection

- Recall that the HTTP protocol is stateless.
- Each HTTP request is separate and isolated from any other ones.
- We’ve repeated this more than a few times this semester 😉
- What are some of the use cases where a stateless network model starts to fail?

Chat
How does our Chat App get new chat messages?
On Page Load:
Request New Posts Messages
api.csc346.arizona.edu

API Responds with new post JSON data
• Now What?
• If new messages are posted by someone else, how does this browser get them?
• Currently you have to reload the page

Polling

• A common approach is known as polling
• The browser checks with the API on a timer and asks for new chat messages

```javascript
setTimeout(() => {
  loadPosts(newestPostTimestamp, null, 10000)
}, 10000)
```
"Got any new posts?"

"Nope"

10 Seconds Later
"Got any new posts?"

Nope
“Got any new posts?”

“Hey I do! Here you go”
Polling

• This works OK for small numbers of infrequent polling
• What happens when there are many clients?

Polling

Has its downsides

• Polling requires each client to constantly ask the API for new data
• Short polling intervals can overwhelm the API host with incoming requests for updates
• Long polling intervals can result in significant delay getting new data out to clients
• The Host may know there’s a new message, but it has to wait for a client to ask for it
WebSockets

All that is old is new again

- What if we could establish a long-lived network connection between the client and the host?
- This is what WebSockets does

WebSockets

- So are WebSockets just regular TCP Sockets?
- Spoiler, No
- Conceptually, WebSockets and TCP Sockets have similar goals
  - Support Long-Lived Connections
  - Two-Way Communication
  - Not Request Based
- However they are not related technologically
  - WebSockets are an extension to the HTTP Protocol that runs on top of a TCP Socket

WebSockets

Challenges

- Low-level socket programming is hard
- Many network situations only permit “web” traffic over ports 80 or 443
- Session and state information about web application logins are already using Cookies, we don’t want a new way of handling state
- Security and encryption are already established for HTTPS communications, developing an additional model would be annoying
WebSockets
Solutions
• Implement a new type of HTTP request
• New request creates a “socket” inside an HTTP request
• Can stay open forever
• Bi-directional comm (not request/response)
• Relatively inexpensive (server memory, network)
• Uses standard HTTP mechanisms for encryption, cookies, etc.
• Uses standard HTTP/HTTPS ports

A regular HTTP request initiates the WebSocket handshake
• Additional headers are sent, telling the host that the client would like to upgrade this connection to a WebSocket
• Passes along a client key
• This is just an identifier, not a cryptographic key

If the server supports WebSockets, it responds with the correct headers
• The Sec-WebSocket-Accept response header is calculated in a seemingly overcomplicated way, but exists so that it’s obvious to the client whether the server supports WebSockets
The `Sec-WebSocket-Accept` header is important in that the server must derive it from the `Sec-WebSocket-Key` that the client sent to it.

To get it, concatenate the client’s `Sec-WebSocket-Key` and the string "258EAFA5-E914-470A-85CA-C5AB0DC85B11" together, take the SHA-1 hash of the result, and return the base64 encoding of that hash.

You likely will never have to do this unless you want to implement a WebSockets-compliant HTTP server. Still useful to know that it’s part of the handshake.

From that point on, there is a persistent connection between the client and host.

Connection remains open until one side or the other explicitly closes it.

Data can be sent and initiation in either direction by either the client or the host at any time.

Data transfer is now a binary format.
WebSockets
Using with JavaScript

- Handshake details are handled by the browser
- Presents a JavaScript interface to us: `new WebSocket(..)`

```javascript
const apiHost = "wss://chat-api.csc346.arizona.edu/chats"
const exampleSocket = new WebSocket(apiHost)
```

WebSockets
Using with JavaScript

  - `ws://` kicks off a handshake over `http://`
  - `wss://` kicks off the handshake over `https://`

```javascript
const apiHost = "wss://chat-api.csc346.arizona.edu/chats"
const exampleSocket = new WebSocket(apiHost)
```

WebSockets
Sending messages to the server

```javascript
const apiHost = "wss://chat-api.csc346.arizona.edu/chats"
const exampleSocket = new WebSocket(apiHost)
exampleSocket.send("Message to the server")

data = {
    "type": "newchat",
    "message": "Here's a new chat message",
    "user": "fischerm"
}
exampleSocket.send(data)
```
const apiHost = "wss://chat-api.csc346.arizona.edu/chats"
const exampleSocket = new WebSocket(apiHost)
exampleSocket.addEventListener('message', (event) => {
  console.log('Message from server ', event.data);
});
### WebSockets

**Server Responsibilities**

- The server side has a few duties
  - Accept HTTP Connections and look for the `Upgrade: websocket` and `Connection: Upgrade` headers
  - Calculate the correct `Sec-WebSocket-Accept` response value
  - Keep the WebSocket open
  - Keep track of all open WebSockets, and allow an API to send messages to specific clients

---

### WebSockets

**AWS API Gateway**

- Most Cloud Providers have a managed service for WebSockets
- AWS API Gateway supports multiple API specifications
  - REST
  - Basic HTTP
  - WebSockets

---

### WebSockets

**AWS API Gateway**

- API Gateway takes care of all the protocol level work associated with WebSockets
  - Accepts and Upgrades WebSocket connections
  - Calculates `Sec-WebSocket-Accept` responses
  - Keeps Socket connections open
  - Assigns Connection IDs to each open WebSocket and tracks activity
  - Sends activity to a backend processor, ie Lambda
WebSockets
AWS API Gateway

Server Code Demo