WebSockets

References for study:

A. Lombardi, Websocket, O'Really, 2015
RFC 6455, The Websocket Protocol, 2011

Why WebSockets?

- Modern applications are becoming more interactive
- Request-response interactions have been proven good for many applications but not for all
  - cooperative work (editing etc.)
  - highly interactive user interfaces
  - gaming, video-conferencing
  - asynchronous notifications from servers to clients (push)

Implementing Push Notifications with Request-Response

- Various possible solutions exist:
  - polling => inefficient
  - long polling => inefficient for frequent notifications
  - HTTP streaming
  - subscription => client must be able to play the server role

Not the natural use of request-response

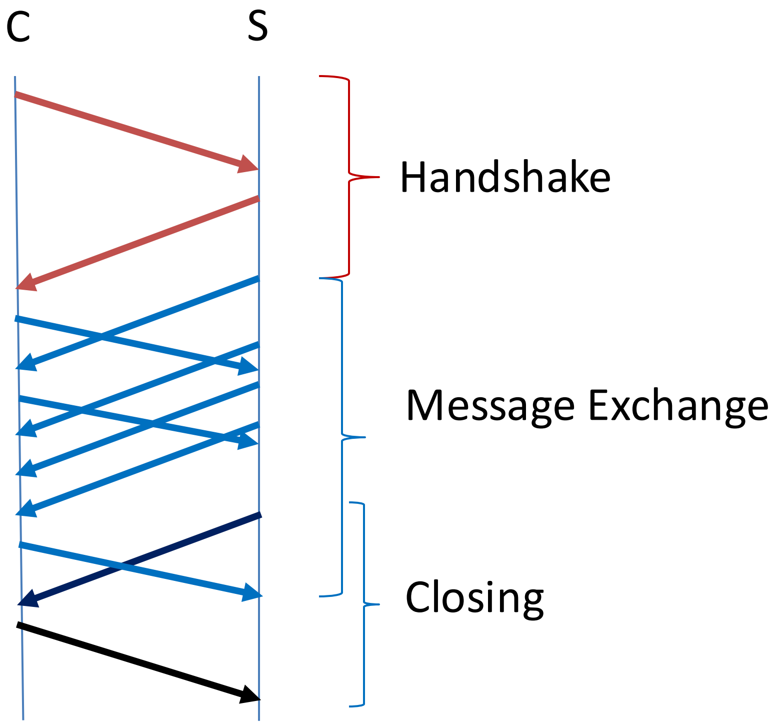
COMET

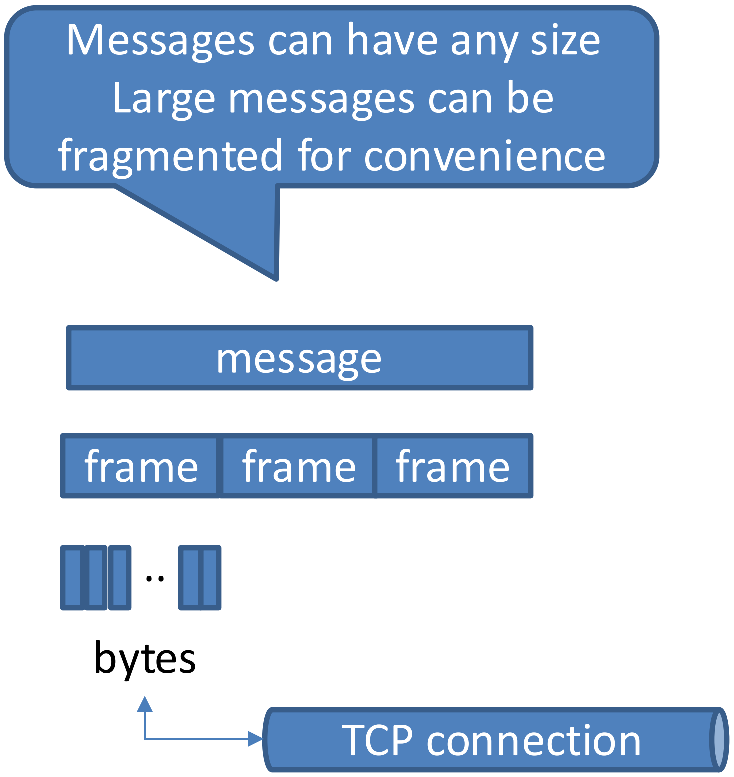
=> Not fully satisfactory for bidirectional low-latency communication

WebSockets

- Application-level protocol that provides reliable low-latency general-purpose bidirectional channels
  - TCP-like communication service, with some differences
- Reuse of the Web infrastructure
  - same HTTP ports (80, 443)
  - compatible with proxies and other web intermediaries
  - can work side-by-side with regular HTTP-based communications (WebSocket server can even share ports with HTTP server)
  - SOP-based security model
- Message-based communication over a binary framing structure layered upon TCP

WebSocket Protocol (RFC 6455)




Messages can have any size
Large messages can be fragmented for convenience

WebSockets vs TCP

- WebSockets offer TCP connections with some additional features:
  - web origin-based security model for browsers
  - addressing and protocol naming mechanism
    - multiple endpoints on one port
    - multiple host names on one IP address
  - framing and messaging mechanism with no message length limit
  - in-band additional closing mechanism that works with proxies and intermediaries

WebSocket URIs

- Websocket endpoints are uniquely identified by URIs
- Like HTTP URIs, but with ws or wss scheme

ws: // host :port path	ws on TCP
wss: // host :port path	ws on TLS on TCP

Handshake

- Client (upgrade) request:

```
GET ws://foo.com:80/webso HTTP/1.1
Host: foo.com
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Key:dGhlIHNhbXBsZSBub25jZQ==
Origin: http://foo.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

WS endpoint

- Server response:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

subprotocol negotiation
see
https://www.iana.org/assignments/websocket/websocket.xhtml

Frame Layout

```
0                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-------+-+-------------+-------------------------------+
|F|R|R|R|opcode |M| Payload len | Extended payload length       |
|I|S|S|S| (4)   |A| (7)         |          (16/64)              |
|N|V|V|V|       |S|             | (if payload len==126 or 127)  |
| |1|2|3|       |K|             |                               |
+-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +
| Extended payload length continued, if payload len == 127      |
+ - - - - - - - - - - - - - - - +-------------------------------+
|                               |Masking-key, if MASK set to 1  |
+-------------------------------+-------------------------------+
| Masking-key (continued)       | Payload Data                  |
+-------------------------------- - - - - - - - - - - - - - - - +
: Payload Data continued ...                                    :
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
| Payload Data continued ...                                    |
+---------------------------------------------------------------+
```

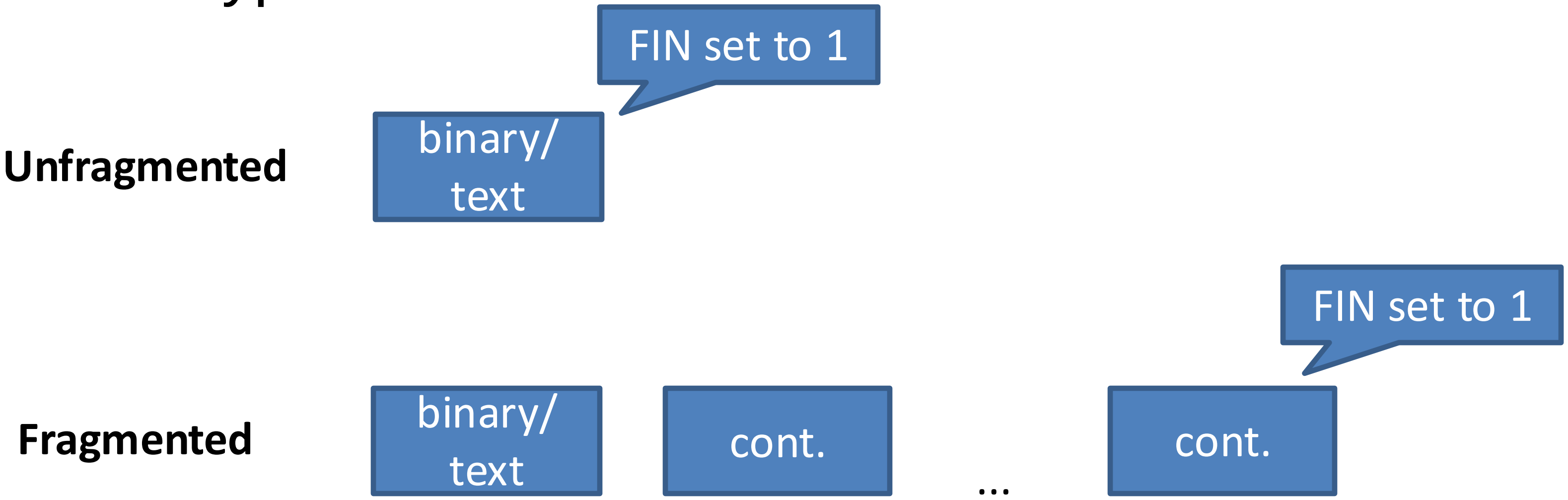
Frame Types

- Control
  - Connection Close
  - Ping/Pong
- Data
  - Binary
  - Text
  - Continuation

Messages

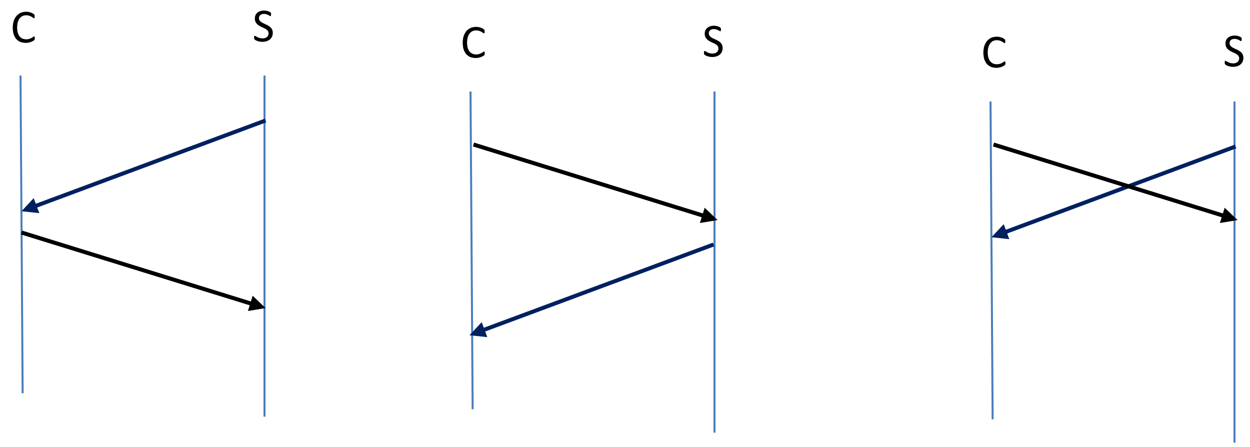
- 2 types of messages:
  - Binary
  - Text (UTF-8 encoded)

Messages are encoded as sequences of data frames, all of the same type:



Closing

- Can be initiated by either endpoint
- After a close frame, no more data frames are sent
- An endpoint responds to a close frame with a close frame, if one was not already sent

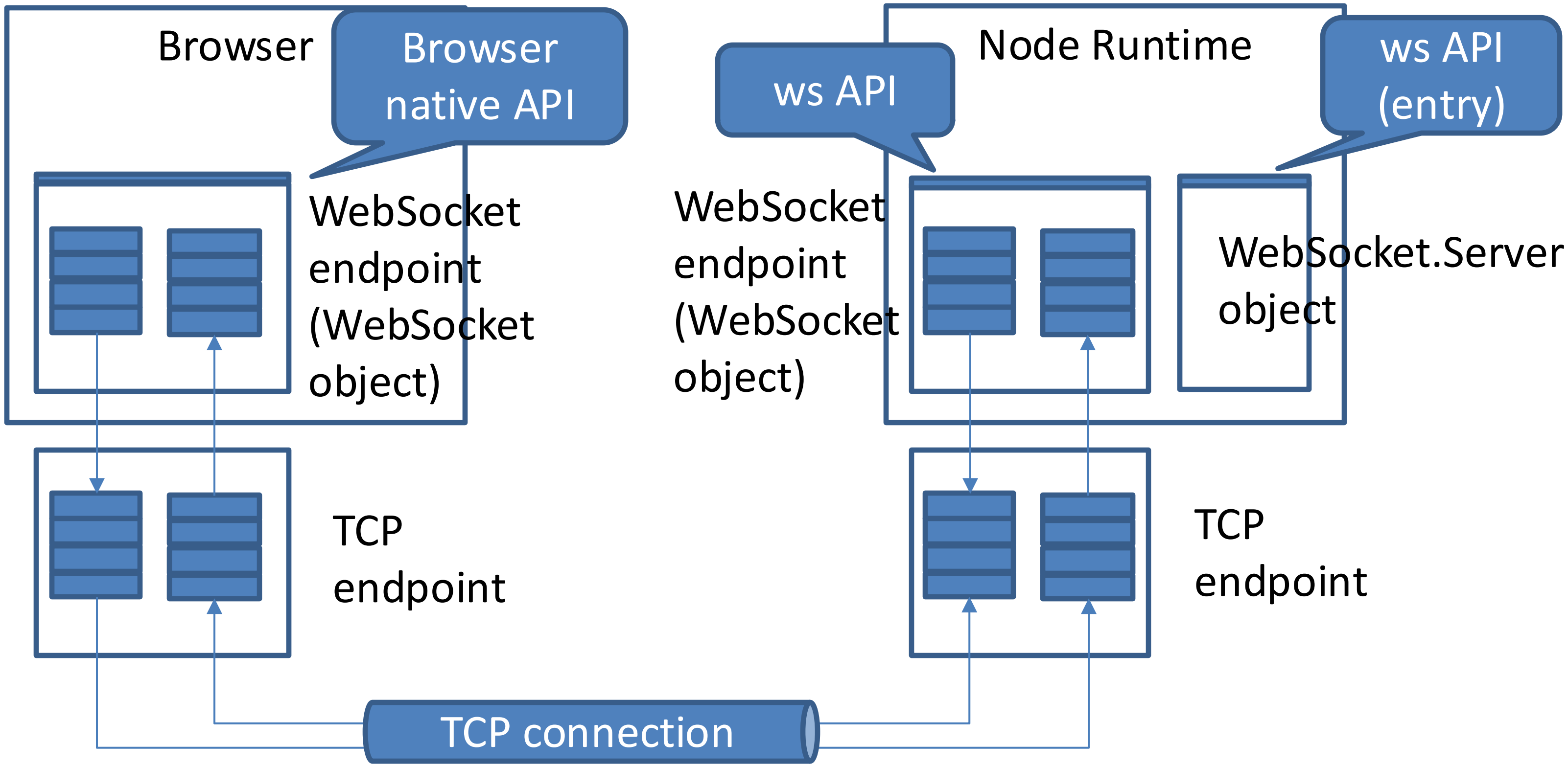


- After the closing handshake, the TCP connection is closed

Using WebSockets

- WebSockets provide minimum data types and control procedures
- Applications using WebSockets implement higher-level protocols
  - message types, metadata, procedures
  - subprotocols are named protocols built upon WebSockets

WebSocket Programming with Javascript



WebSocket Programming in the Browser (Client-side)

- **Native Javascript API**

https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

  - Constructor: WebSocket(url [, protocols])
  - Constants:
    - CONNECTING
    - OPEN
    - CLOSING
    - CLOSED
  - Methods:
    - send(data)
    - close([statuscode [,reason]])

WebSocket Programming in the Browser (Client-side)

https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

– Events:

open	a connection has been opened
close	a connection has been closed
message	a message has been received
error	an error has occurred

– Properties

- onopen, onclose, onmessage, onerror

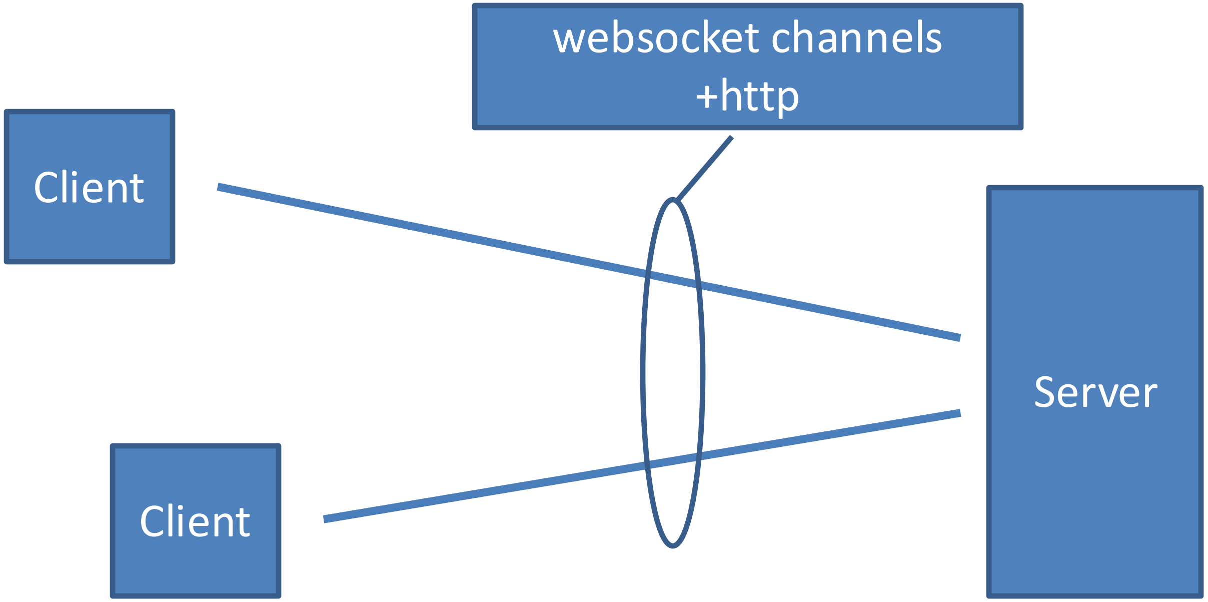
url	absolute URL of the server endpoint
protocol	the subprotocol selected by the server
readyState	current state of connection
binarytype	("blob" | "arraybuffer")
bufferedamount	outstanding data to be sent

Programming WebSocket Servers in Node with ws Library

- https://www.npmjs.com/package/ws
- WebSocket.Server class: a WebSocket server
  - Constructor: WebSocket.Server(options [, callback])
  - Main properties and methods:
    - clients a set including all connected clients
    - close([callback]) close the server
  - Main Events:
    - connection WebSocket, http.IncomingMessage
    - error Error

same properties as in Client API

Example: A simple Chat


websocket channels
+http
Client
Client
Server