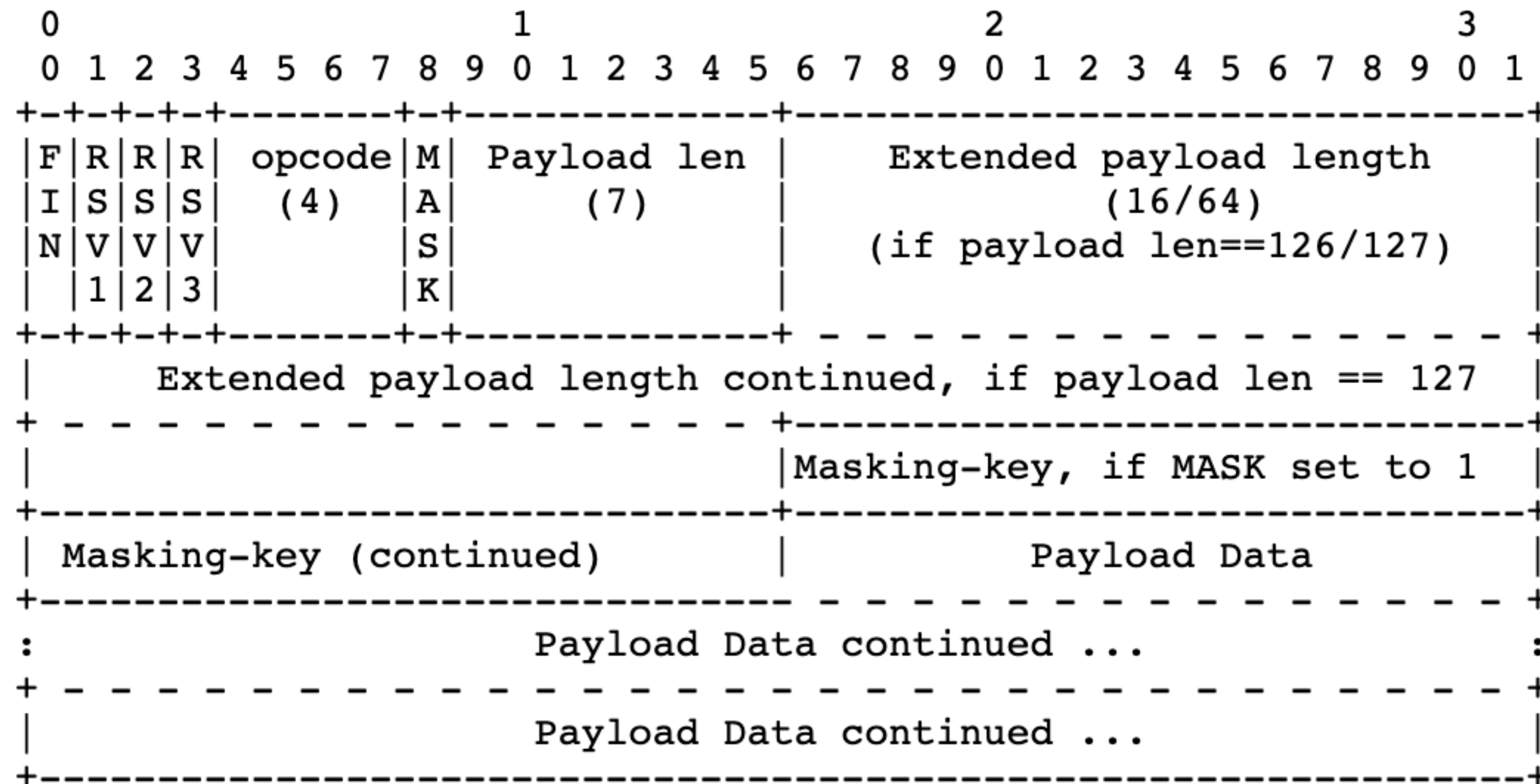# WebRTC

WebSocket Review

- Client sends an HTTP GET request to the WebSocket path

- Client sets headers

  - Connection: Upgrade

  - Upgrade: websocket

  - Sec-WebSocket-Key: <random_key>

- Server responds with 101 Switching Protocols with headers

  - Connection: Upgrade

  - Upgrade: websocket

  - Sec-WebSocket-Accept: <accept_response>

WebSocket Review

- The client generates a random "Sec-WebSocket-Key" for each new WebSocket connection
- The server appends a specific GUID to this key
  - "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"
- Computes the SHA-1 hash
- "Sec-WebSocket-Accept" is the base64 encoding of the hash
- Why?
  - Ensure client and server both implement the protocol
    - Highly unlikely this value would be returned by accident
  - Avoid caching

WebSocket Review

```
 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==126/127)   |
| |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 ...                |
+---------------------------------------------------------------+
```

https://tools.ietf.org/html/rfc6455#section-5.2

WebSocket Review

- Payload Length

  - Represented as either 7, 16, or 64 bits

- Masking

  - Read the 4 mask byes

  - Unmask the payload by XORing each byte of the payload with the matching mask byte

- Do not mask frames send by your server

WebRTC Demo

WebRTC - Overview

- Web Real Time Communication

- Establishes a **live streaming peer-to-peer** connection

  - We'll stream video and audio to make a video chat app

- Stable release in 2018

- Widely adopted by major browsers

  - Most of the WebRTC logic/code is built into your browser

WebRTC - Overview

- WebRTC establishes a live streaming **peer-to-peer** connection

- **Peer-to-peer**

  - Two clients will communicate without the use of a server

  - Your server will only help the clients establish the connection

  - The server does not handle the steaming data

- Excellent for anyone concerned with privacy

  - Though your ISP can still see your data..

WebRTC - Overview

- WebRTC establishes a **live streaming** peer-to-peer connection

- **Live streaming**

  - The protocol is meant for live (real-time) streaming

  - End-to-end delay is critical!

  - Even a small delay will result in clients talking over each other on a voice/video call

WebRTC - Overview

- WebRTC establishes a **live streaming** peer-to-peer connection

- **Live streaming**

- TCP can be slow!

  - Meant for reliability

  - If a packet is dropped, request a resend and wait

  - Only deliver bytes after all packets arrive and are reassembled

  - **Not suitable for live [real-time] streaming***

Other protocols are used when delays are tolerable (ie. not real-time) like YouTube Live or Twitch

WebRTC - Overview

- WebRTC establishes a **live streaming** peer-to-peer connection

- **Live streaming**

- WebRTC uses UDP instead of TCP

- UDP (User Datagram Protocol)

  - Meant for speed

  - If a packet is dropped, it's lost forever. Move on with your life

  - Bytes are delivered as soon as they are received

  - Very close to using raw IP packets

WebRTC - Overview

- Servers are still involved

  - We'll discuss 3 types of servers that assist in WebRTC connections

- Signalling Server

  - This is the one you'll implement

  - Passes messages between clients to help them establish a peer-to-peer connection

  - Once the connection is established, the server's job is done (Unless you want to pass a disconnect message)

WebRTC - Overview

- Servers are still involved

  - We'll discuss 3 types of servers that assist in WebRTC connections

- STUN (Session Traversal Utilities for NAT) Server

  - A server that tells clients their public IP/port

  - Clients behind a NAT or firewall may not know their public IP/port

  - Ask the STUN server then send this info to the signaling server

WebRTC - Overview

- Servers are still involved

  - We'll discuss 3 types of servers that assist in WebRTC connections

- TURN (Traversal Using Relays around NAT) Server

  - **Optional for WebRTC connections**

  - All WebRTC packets for a connection are routed through the TURN server if one is used

  - Helpful when the NATs/Firewalls are too restrictive to allow a true peer-to-peer connections (eg. Symmetric NATs require a TURN)

  - Kind of defeats the purpose of WebRTC if you ask me..

WebRTC

- Reminder

  - Only need to implement the signaling server for the HW

  - Details on what to code in the next lecture