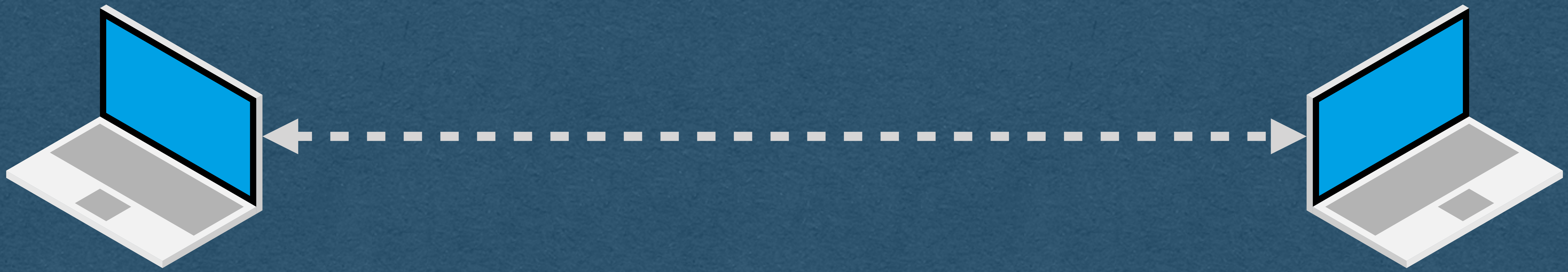
WebRTC

WebRTC

- Need to establish a real-time streaming peer-to-peer connection
- But how?
  - Need the IP address of your peer
  - Need to agree on the details of the connection



WebRTC - Connecting

- One peer needs to get an offer to the other peer
- This is an offer to establish a connection that contains:
  - audio/visual codec, bitrate, etc details (How to interpret the bytes once the streaming starts)
  - A username fragment ("ice-ufrag") as a unique identifier

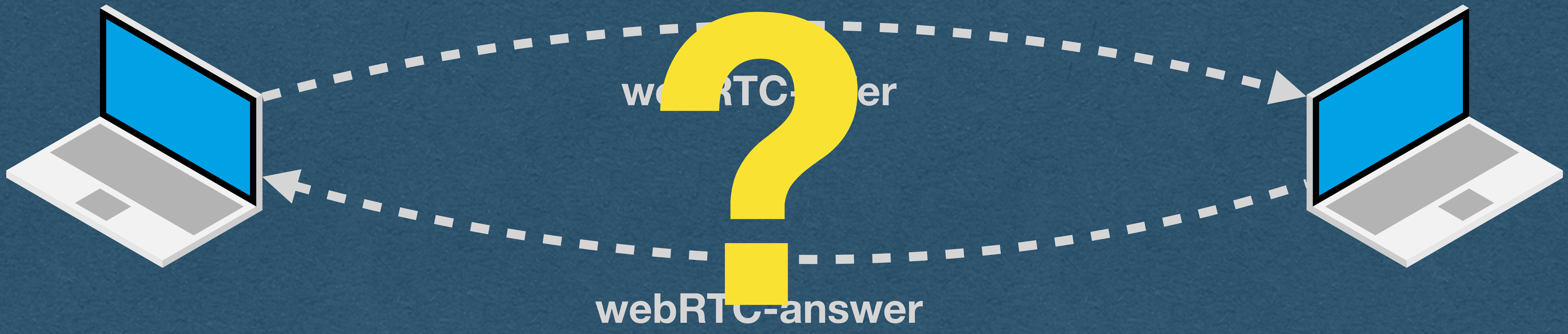
WebRTC - Connecting

- The peer responds with an answer
- The answer contains their audio/visual data
- Contains their own ufrag so the connection can be identified

- Once the answer is received, the peers agree to connect

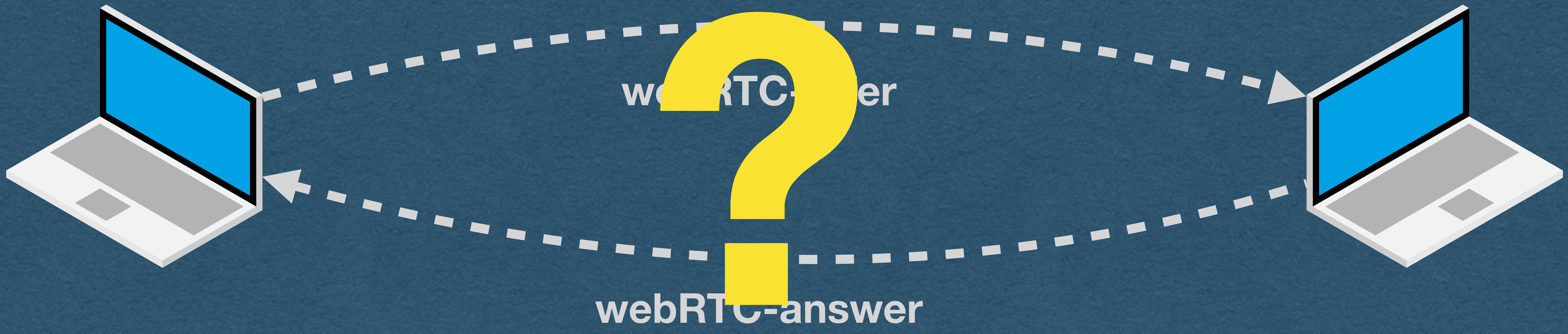
WebRTC - Connecting

- But there's a problem
- How do we send these messages between two peers?



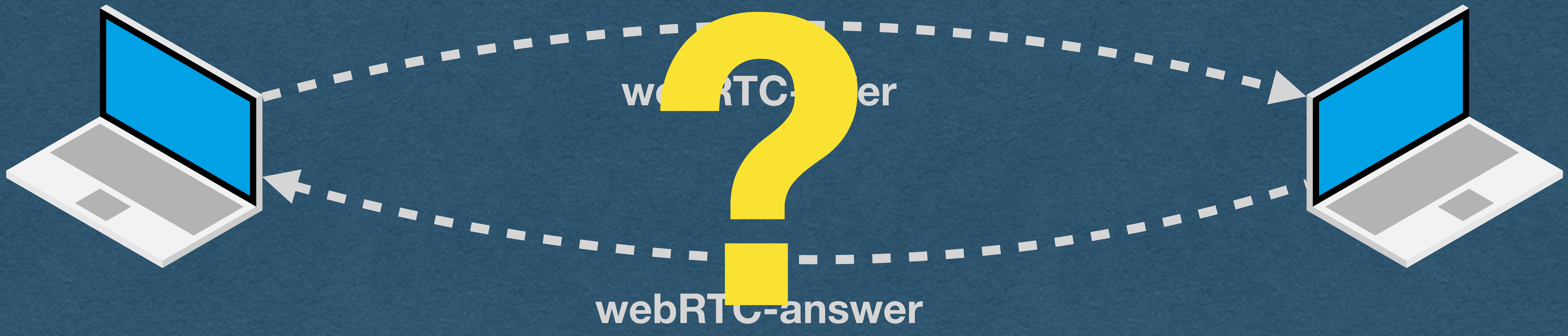
WebRTC - Connecting

- For usual web traffic with a server:
  - Type in a domain name or click a link containing a domain name
  - Use DNS to lookup the [static] IP address of the server
  - Send a request to the IP address on port 80 or 443



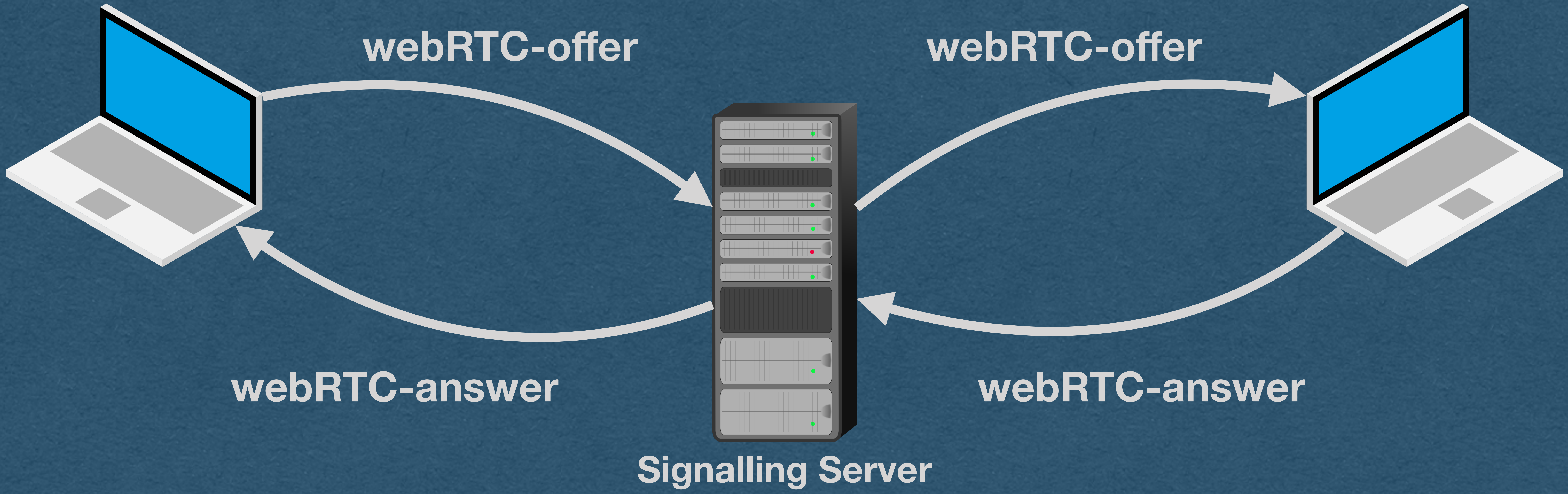
WebRTC - Connecting

- For peer-to-peer traffic:
  - We need to discover the IP and port of the peer without DNS
  - Peer IP/port can change (dynamic IP)



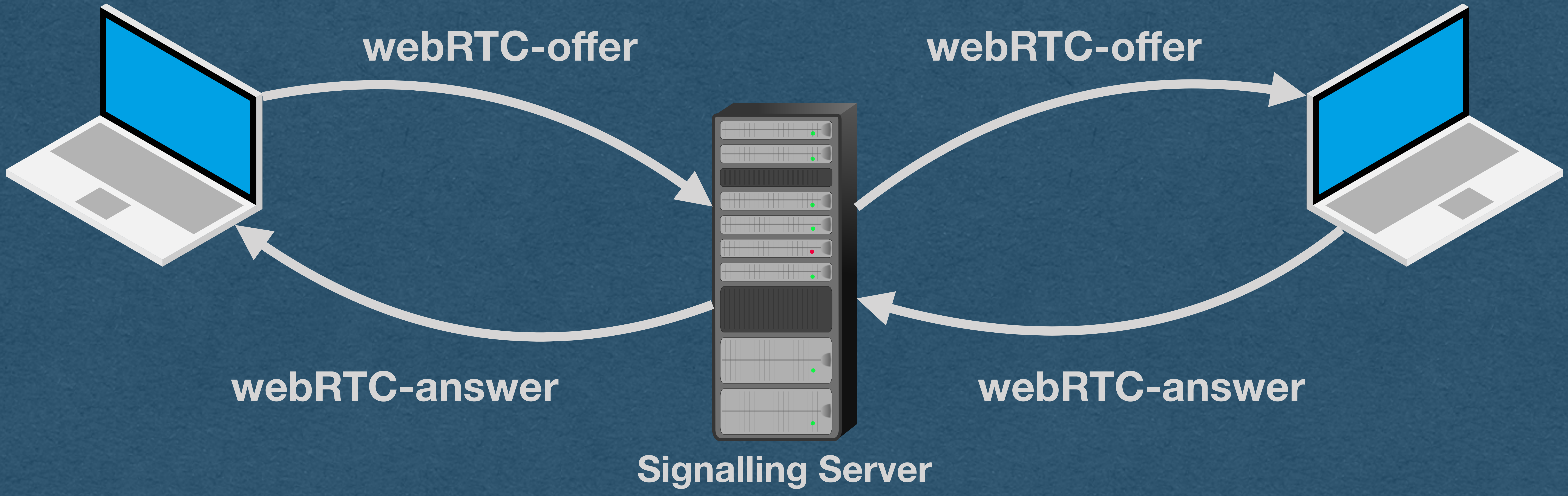
WebRTC - Signalling Server

- Offer and answer are sent through a signaling server
  - On the HW - **You are the signaling server!!**
- Both peers connect to your server
- Send offer/answer to the server and the server forwards the messages to the other peer



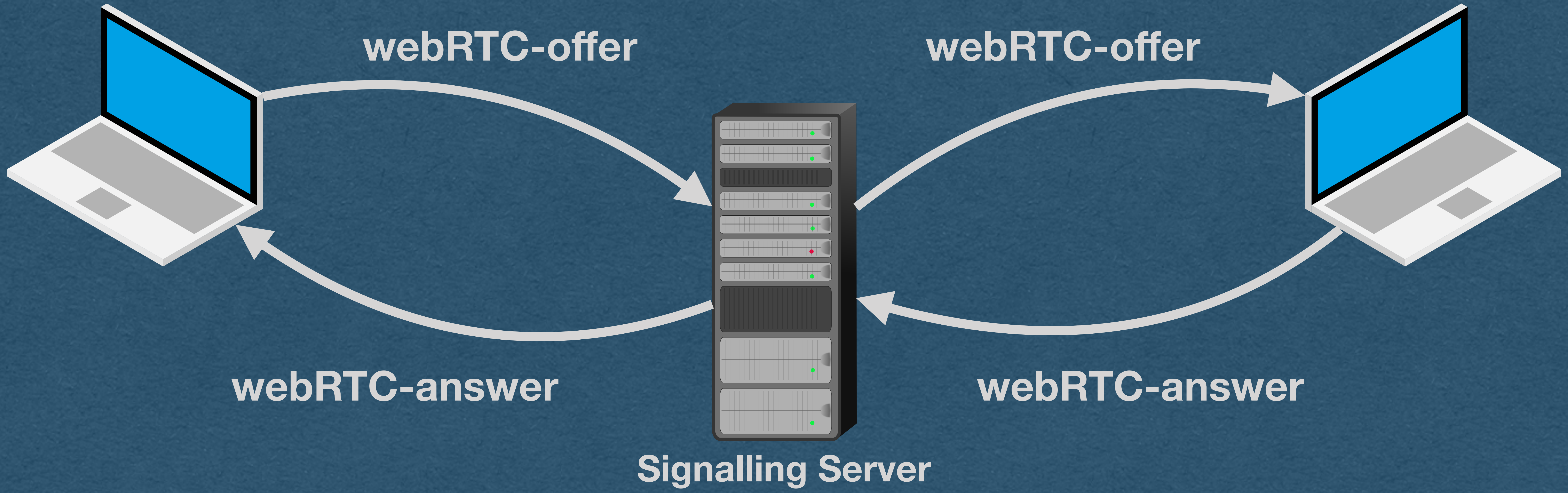
WebRTC - Signalling Server

- We need a method to send these messages
- We'll need a way for the server to be able to send messages to each client in real-time



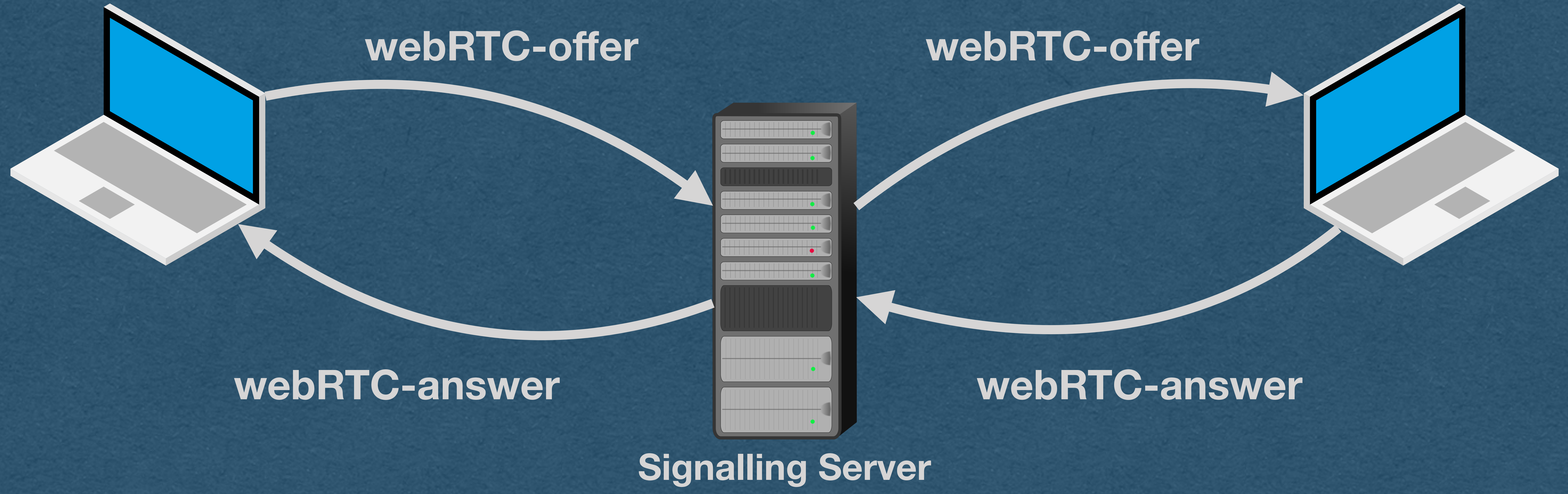
WebRTC - Signalling Server

- We need a method to send these messages
- We'll need a way for the server to be able to send messages to each client in real-time
- Wow! It's really convenient that we have WebSockets!



WebRTC - Signalling Server

- When your server receives a WebSocket frame containing an offer or answer
    - Send the payload to the other peer over their WebSocket



WebRTC - NATs

- We have another issue..

- How do we know the IP address and port for each peer?

WebRTC - NATs

- Devices are commonly "hidden" behind NAT routers
  - Network Address Translation
- With a NAT:
  - You have a local IP address for communication on your local network
  - When communicating outside your network, the NAT router sends your message using a public IP address and port number

WebRTC - NATs

- With a NAT:
  - Many devices on a local network can share a single public IP address
  - Each device does not know it's public IP/port used when it communicates to the outside world
  - [Also the problem that port-forwarding solves]

WebRTC - STUN Server

- Solution: Use a STUN (Session Traversal Utilities for NAT) Server
- Each peer connects to a STUN server and asks for their public IP/port
- STUN server checks the origin IP/port and informs the client
- We'll use Google's free STUN server (stun2.1.google.com:19302)

WebRTC - ICE Candidate

- Each peer sends their public IP/port, and connection information, in an ICE (Interactive Connectivity Establishment) candidate message
- Whenever your signaling server receives an ICE candidate, forward it to the other peer

WebRTC - ICE Candidate

- Candidate contains:
  - Connection type - UDP, not TCP
  - IP/port - Local in this example
  - Username fragment - Uniquely identifies the connection

{"candidate":"candidate:2382557538 1 udp 2122260223 192.168.1.19 54090 typ host generation 0 ufrag FGP/ network-id 1 network-cost 10","sdpMid":"0","sdpMLineIndex":0}

WebRTC - Connection

- And now we can establish a peer-to-peer connection!

WebRTC - Summary

- One peer sends an offer to the other
- Other peer responds with an answer
- Both peers get their public IP/port from their STUN Servers
- Both peers send their ICE candidates to the other

WebRTC - Summary

- Once the connection is established
- The servers step aside and the clients stream directly to each other
  - True peer-to-peer!

WebRTC - Summary

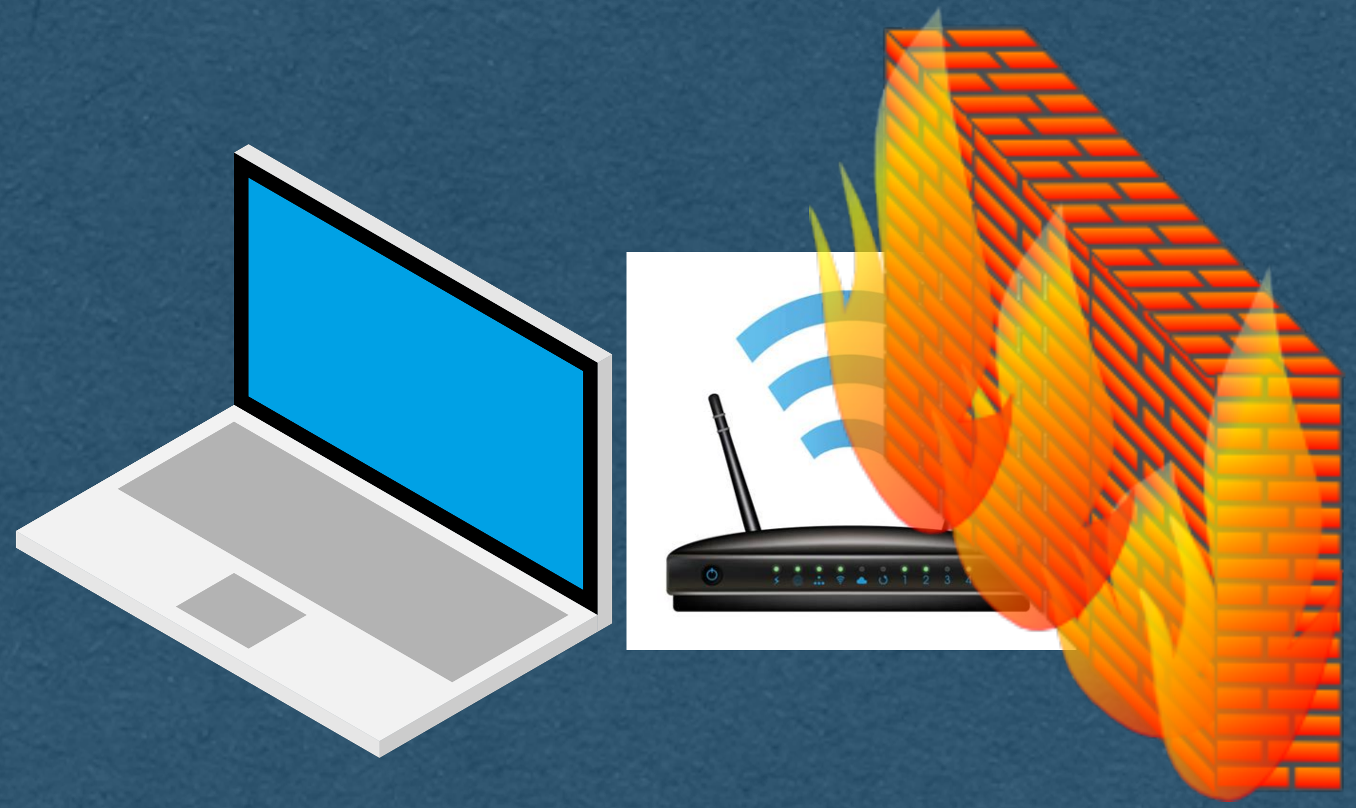
- Your role in all of this?
  - Route the offer/answer/candidate messages between peers
- No need to read/parse/interpret the RTC portion of these messages
  - Extract the payload from the WS frame, send it to the appropriate peer as a new WS frame

WebRTC - Restrictions

- The browser will not allow WebRTC connections when connected to a site using **HTTP** (as opposed to HTTPS)
- **Must have an encrypted connection to use WebRTC**
  - *Unless connecting over localhost (Let's us test locally)

WebRTC - Restrictions

- Sometimes a peer-to-peer connection cannot even be established
  - Can have restrictive firewalls
  - Dynamic NATs might change your port unexpectedly
  - Organizations might block certain traffic on their network



WebRTC - TURN Server

- In cases where peer-to-peer is blocked:
  - Use a TURN (Traversals Using Relays around NAT) Server
  - After the connection is established using a signaling server, each peer routes their streaming data through a TURN server

WebRTC - TURN Server

- If you ask me... using a TURN server defeats the purpose of using a peer-to-peer technology
- ... Unless you run your own TURN server!

WebRTC - On the HW

- You implement the signaling server
- For AO2, you may assume that there are exactly 2 WS connections
    - When you receive a WebRTC message from one connection, send it to the other connection

WebRTC - On the HW

- For AO3, you can have any number of peers (max of 4 connections when grading)
  - Must modify the front end to support multiple WebRTC connections (You are expected to study the front end and understand how it works)
  - Each peer maintains a connection to each peer
  - Server must route WebRTC messages to the appropriate peer