WebRTC – from zero to hero!

v0.1.0

About us

Jakub Pisarek
@sgfn
@jpisarek

Michał Śledź
@mickel8
@mickel8v2

Workshop agenda

- What's WebRTC and when to use it?
- Theoretical introduction to the WebRTC protocol stack
- Setting up a peer-to-peer audio/video connection between two web browser tabs
- Introduction to WebRTC monitoring and debugging
- WHIP and WHEP – stream from OBS using WebRTC
- Mastering Transceivers
- Why do we need media servers?
- Building a simple video-conferencing application with Fishjam Cloud

Workshop schedule

- 09:00 – 11:00: Session 1
- 11:00 – 11:30: Coffee break
- 11:30 – 13:00: Session 2
- 13:00 – 14:00: Lunch break
- 14:00 – 15:30: Session 3
- 15:30 – 16:00: Coffee break
- 16:00 – 17:00: Session 4

What’s WebRTC?

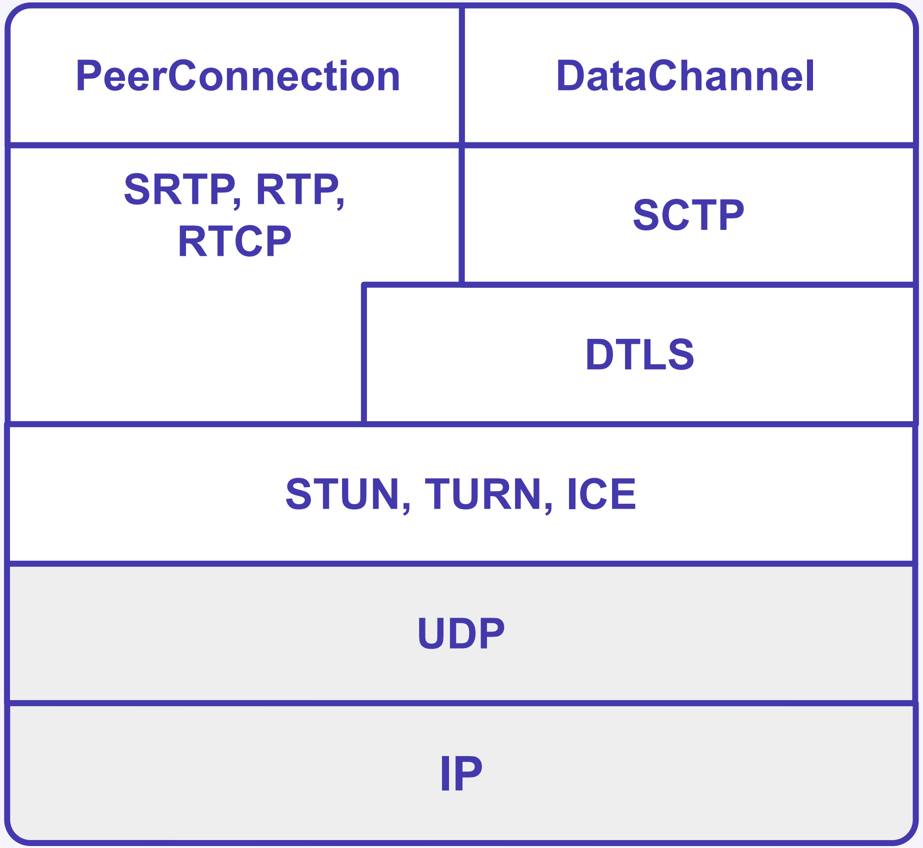
What’s WebRTC?

- **Web R**eal-**T**ime **C**ommunication
- A set of protocols that allows for secure, P2P, real-time audio&video exchange between browsers.

A set of protocols

- est. connection -> protocol A
- sending media -> protocol B
- encrypting media -> protocol C
- sending data -> protocol D
- encrypting data -> protocol E
- negotiating parameters -> protocol F

PeerConnection API


PeerConnection
DataChannel
SRTP, RTP, RTCP
SCTP
DTLS
STUN, TURN, ICE
UDP
IP

v0.1.0

Real-time

- latency below 200ms
- data prioritization – audio is the most important, then video and its quality
- how to deal with poor networks – retransmissions, forward error correction, adaptive streaming, bandwidth estimation
- we have to be flexible and adapt to the changing environment

P2P

- we can directly connect two people that are in their private networks without forwarding traffic through a server
- one of the most important features of WebRTC

Secure

- data is always encrypted
- you cannot obtain access to audio and video devices from non-https websites (excluding localhost)
- video players are muted by default unless there is an interaction with the website

Implemented in web browsers

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
`RTCPeerConnection`	✓ 56 ...	✓ 15 ...	✓ 44 ...	✓ 43 ...	✓ 11	✓ 56 ...	✓ 44 ...	✓ 43 ...	✓ 11	✓ 6.0 ...	✓ 56 ...

WebRTC applications

- Google Meet
- Discord
- Microsoft Teams
- Slack

Non-WebRTC applications

- YouTube
- Twitch

Ex. Create a PeerConnection object in browser console

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

A set of interfaces

- RTCPeerConnection – send/receive audio and video
- **getUserMedia – obtain access to microphone and camera**
- RTCDataChannel – send/receive arbitrary data

Ex. Obtain access to audio and video devices

- https://github.com/elixir-webrtc/workshop/
- ex1

When to use WebRTC?

- interactive communication
- video conferencing
- real-time audio/video AI processing (Speech-To-Text, Image recognition, conversations with bots)
- real-time broadcasting (Broadcaster, broadcast-box)
- telemedicine

WebRTC is standardized by W3C and IETF

W3C – responsible for the web browser API. It’s the same organization that’s in charge of e.g. CSS

- https://www.w3.org/TR/webrtc/
- https://www.w3.org/TR/css-flexbox-1/

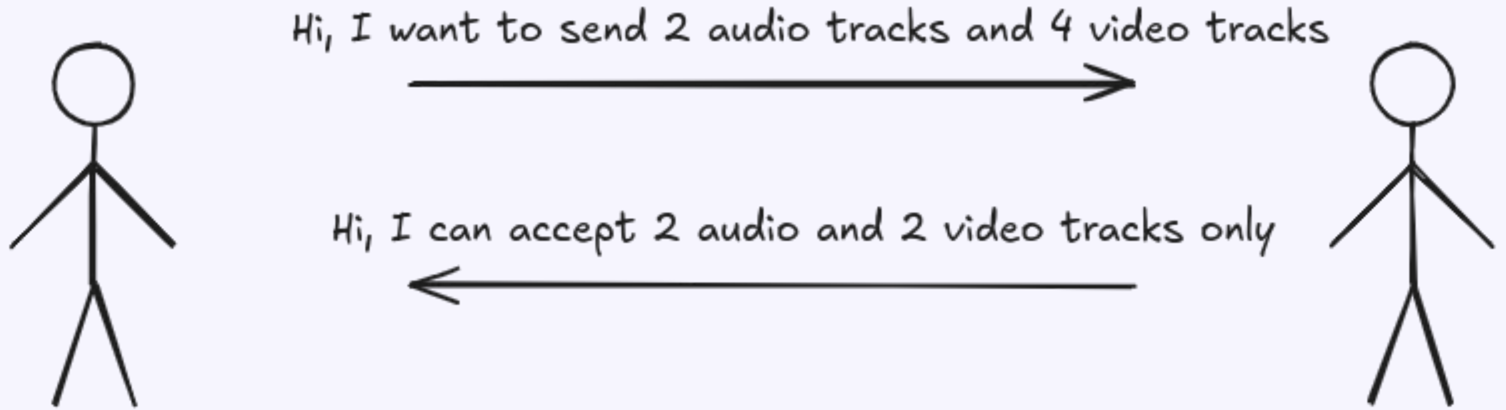
IETF – responsible for specific protocols (ICE, RTP, SDP, etc.) described in RFC documents

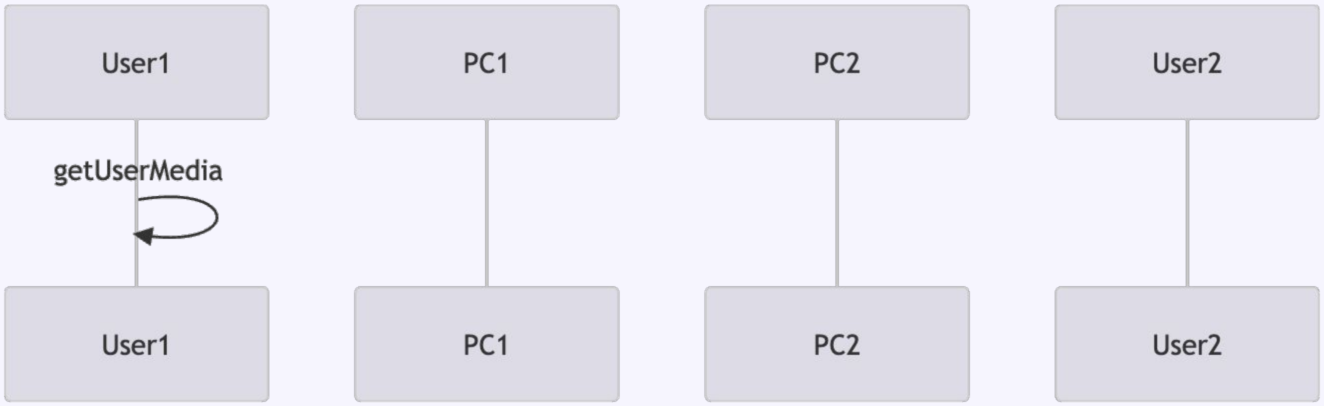
- https://datatracker.ietf.org/doc/html/rfc8829

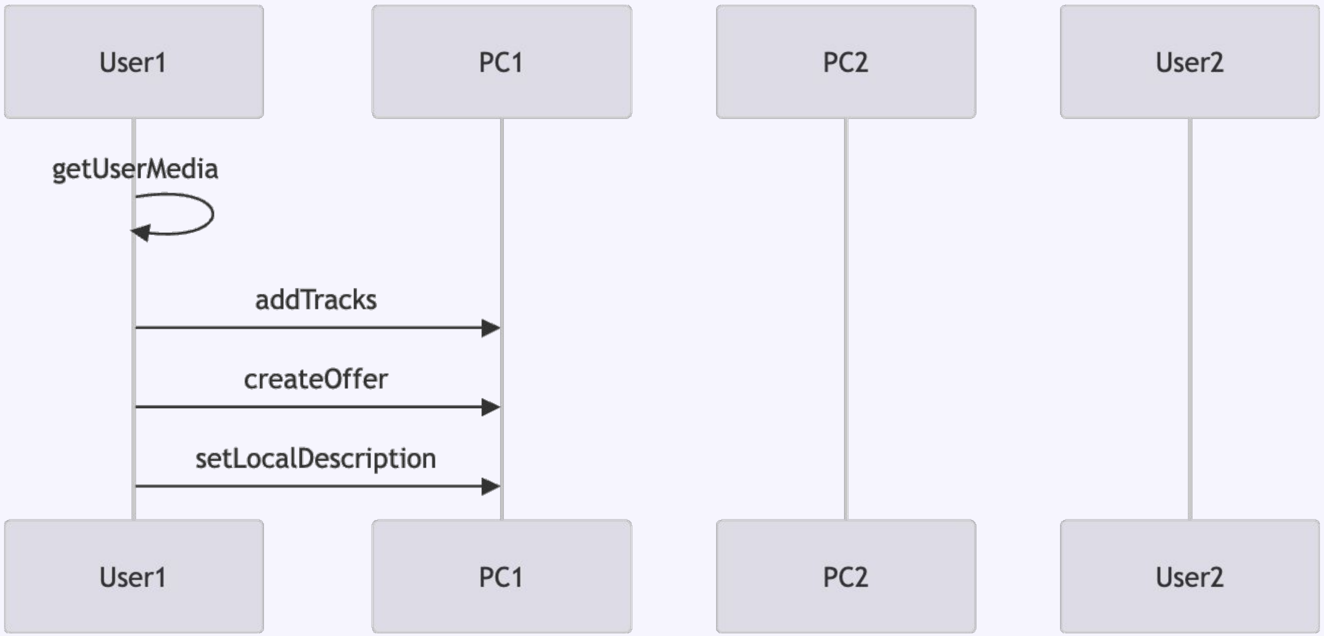
Documentation:

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

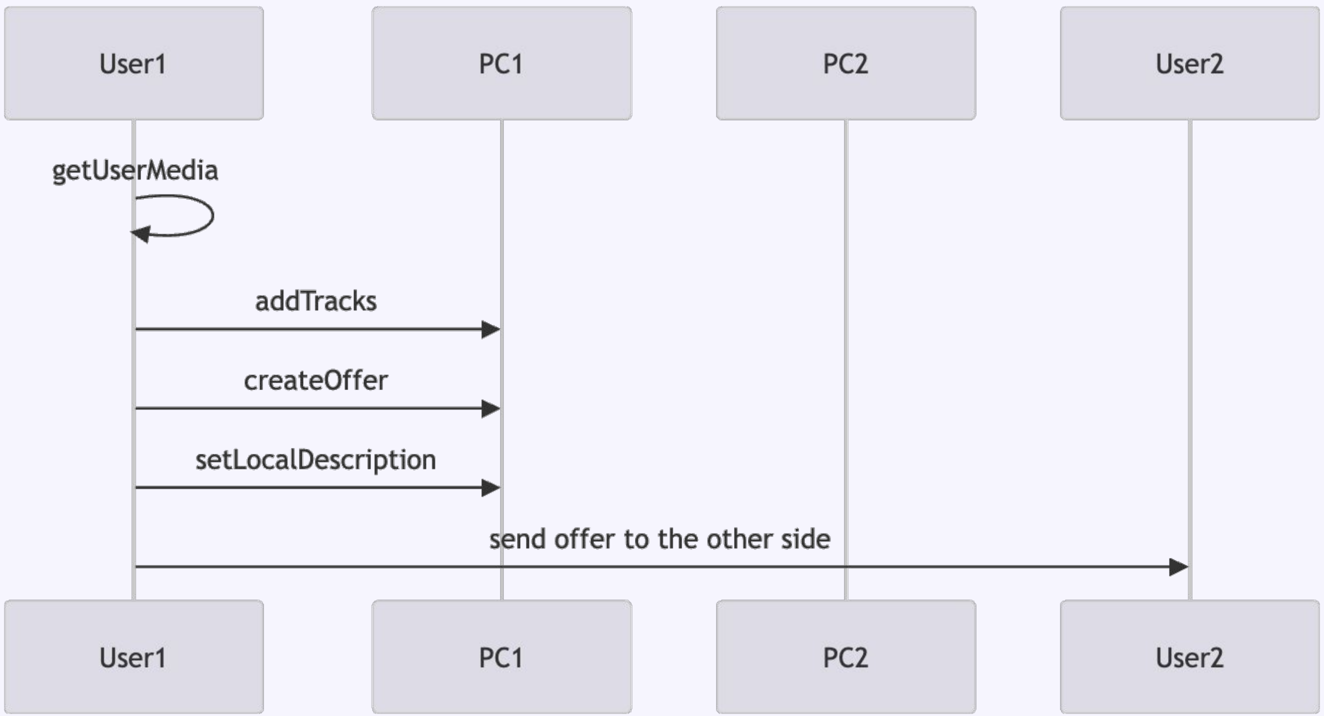
Negotiating session parameters



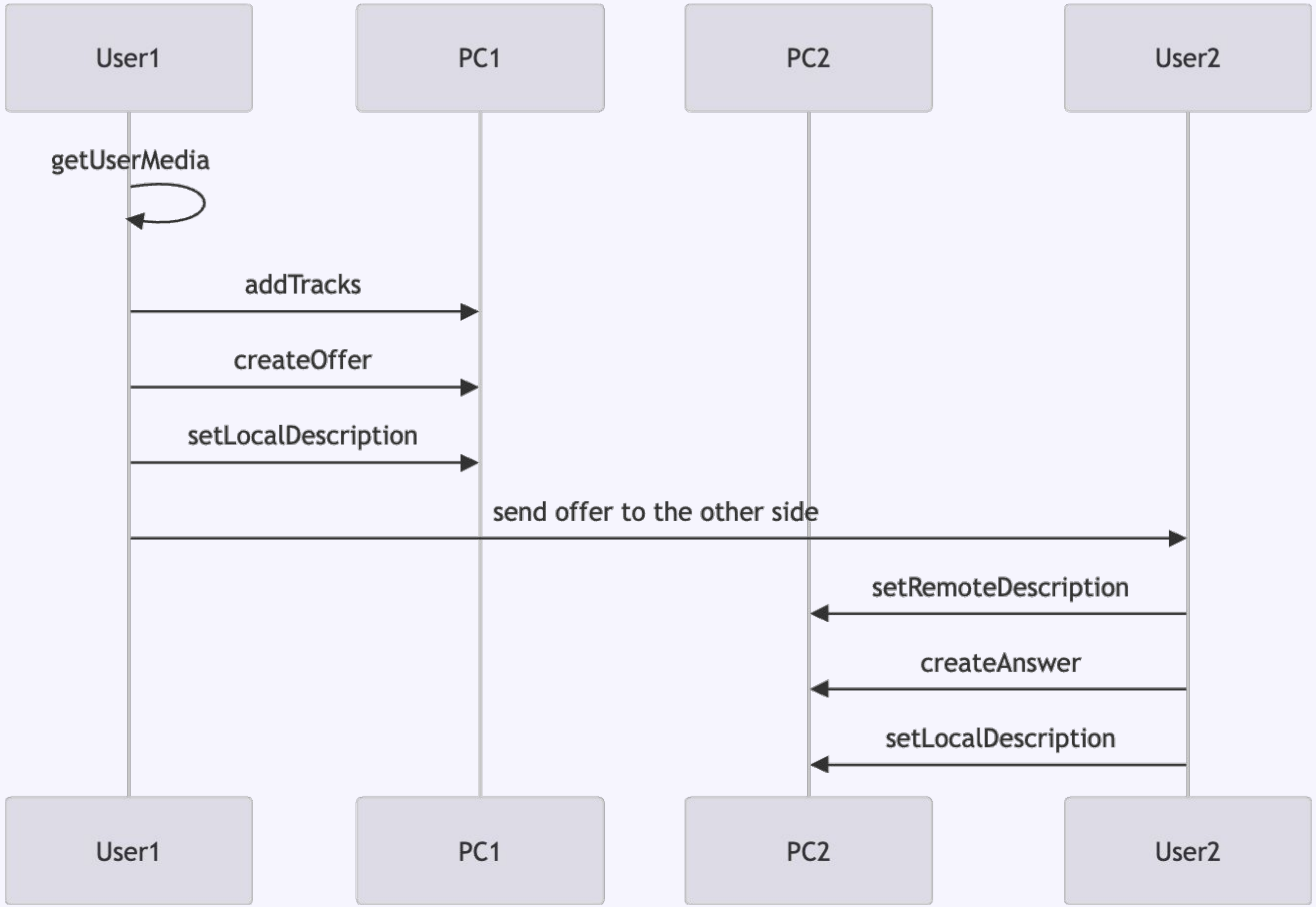

User1
PC1
PC2
User2
getUserMedia
User1
PC1
PC2
User2


User1
PC1
PC2
User2
getUserMedia
addTracks
createOffer
setLocalDescription
User1
PC1
PC2
User2

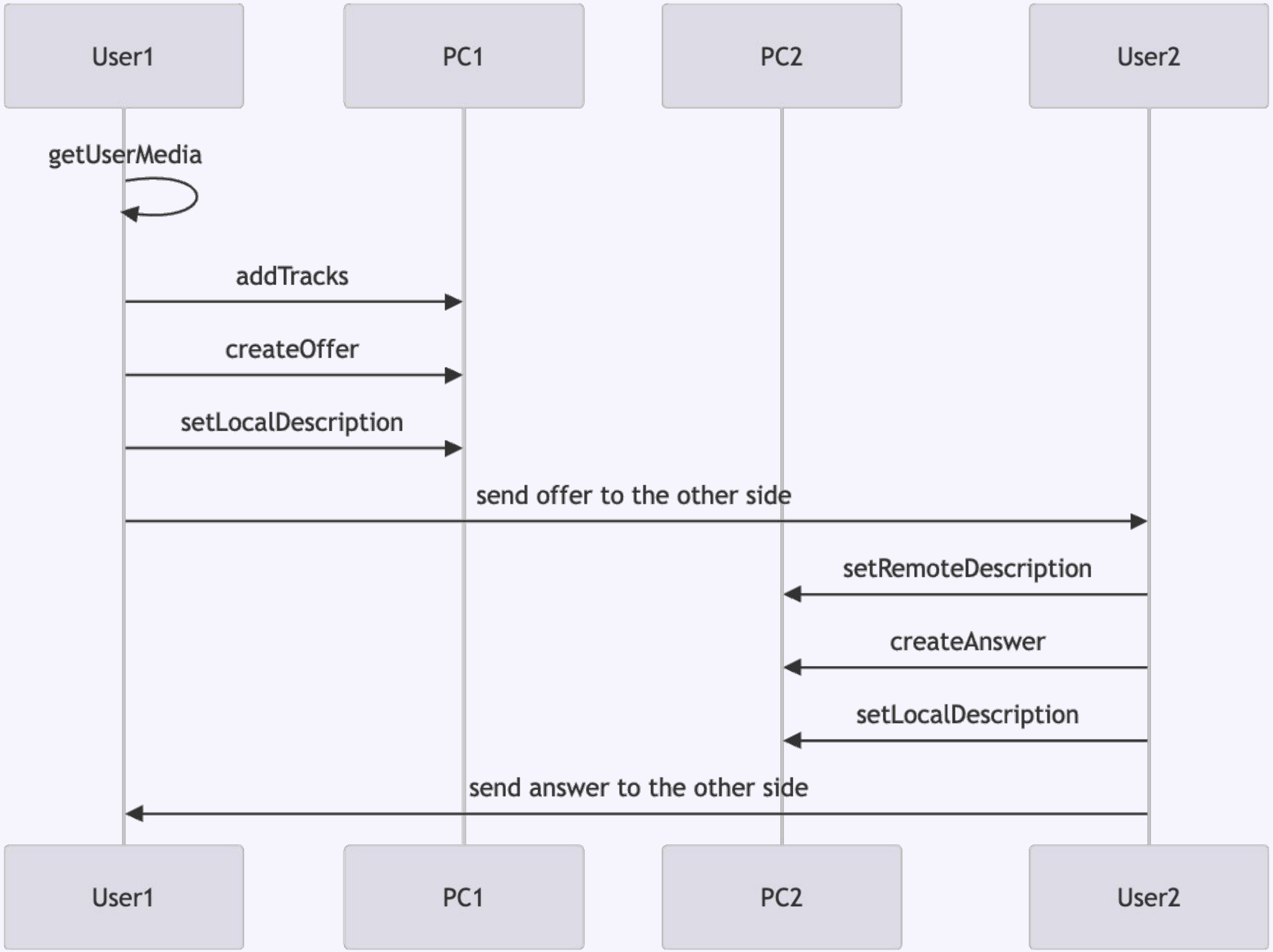
v0.1.0


User1
PC1
PC2
User2
getUserMedia
addTracks
createOffer
setLocalDescription
send offer to the other side
User1
PC1
PC2
User2

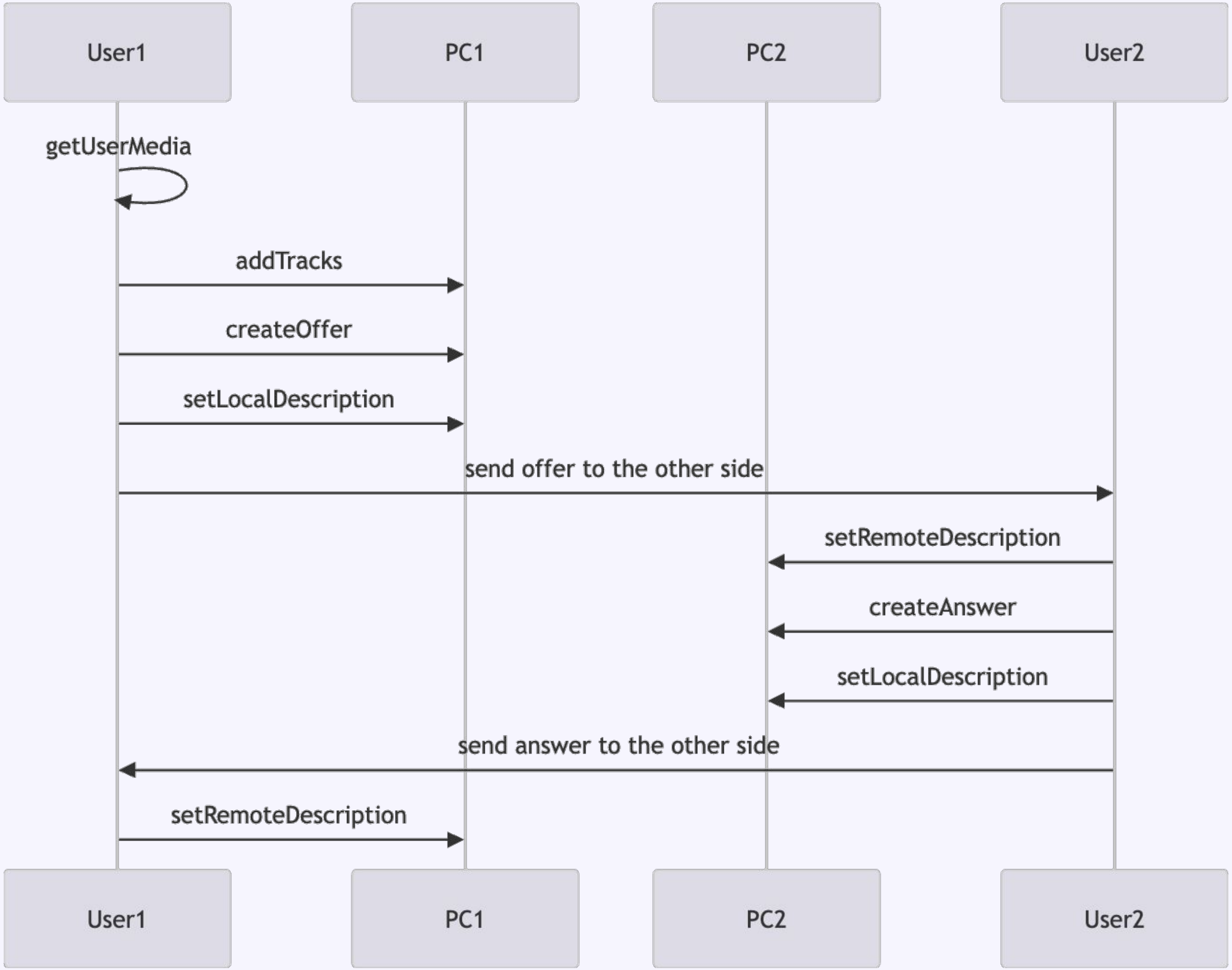
v0.1.0


User1
PC1
PC2
User2
getUserMedia
addTracks
createOffer
setLocalDescription
send offer to the other side
setRemoteDescription
createAnswer
setLocalDescription
User1
PC1
PC2
User2

v0.1.0


User1
PC1
PC2
User2
getUserMedia
addTracks
createOffer
setLocalDescription
send offer to the other side
setRemoteDescription
createAnswer
setLocalDescription
send answer to the other side
User1
PC1
PC2
User2

v0.1.0



User1
PC1
PC2
User2
getUserMedia
addTracks
createOffer
setLocalDescription
send offer to the other side
setRemoteDescription
createAnswer
setLocalDescription
send answer to the other side
setRemoteDescription
User1
PC1
PC2
User2

v0.1.0

Ex. Negotiate the session parameters

- https://github.com/elixir-webrtc/workshop/
- ex2

Ex. Implement the `ontrack` callback

- https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/track_event
- Pin the received MediaStream to the video player
- Hint: use **event.streams[0];**

WebRTC monitoring and debugging

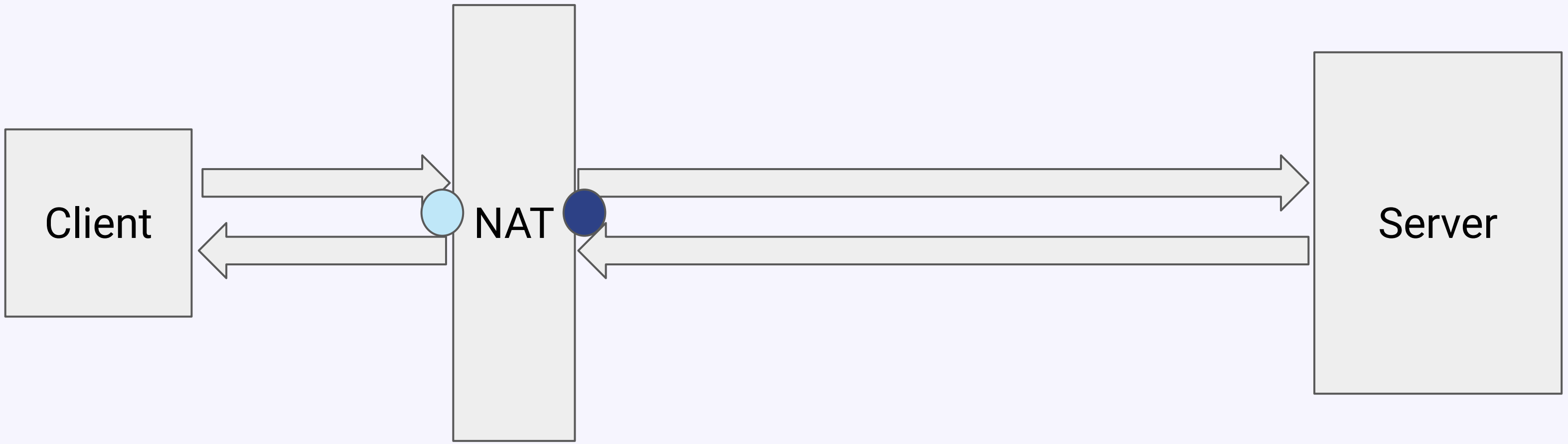
v0.1.0

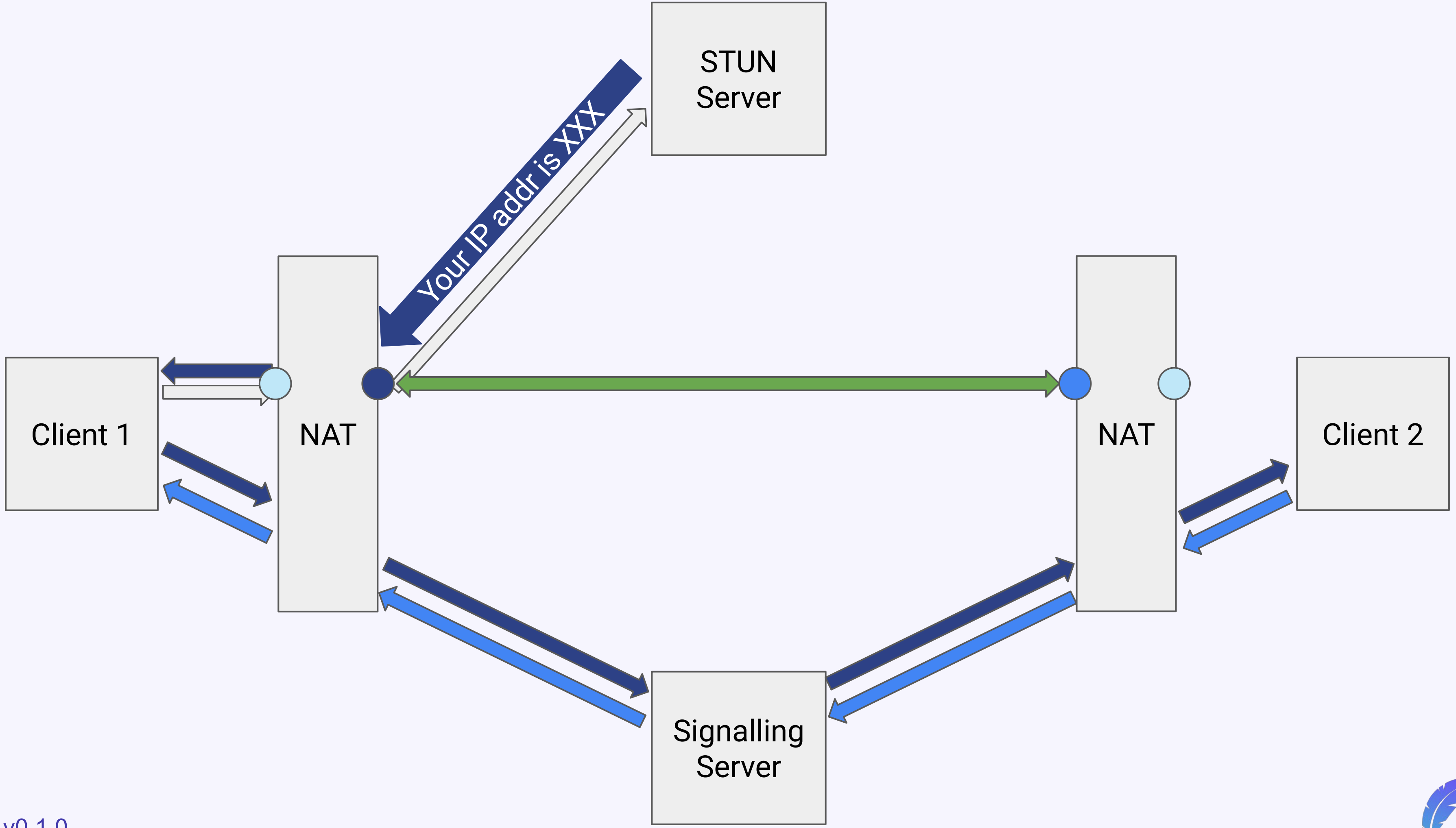
Ex. Use chrome://webrtc-internals to find an answer to the following questions:

- What is the state of (Peer)Connection?
- What is the state of ICEConnection?

What is ICE?

How does the Internet work?




STUN Server
Your IP addr is XXX
Client 1
NAT
NAT
Client 2
Signalling Server

v0.1.0

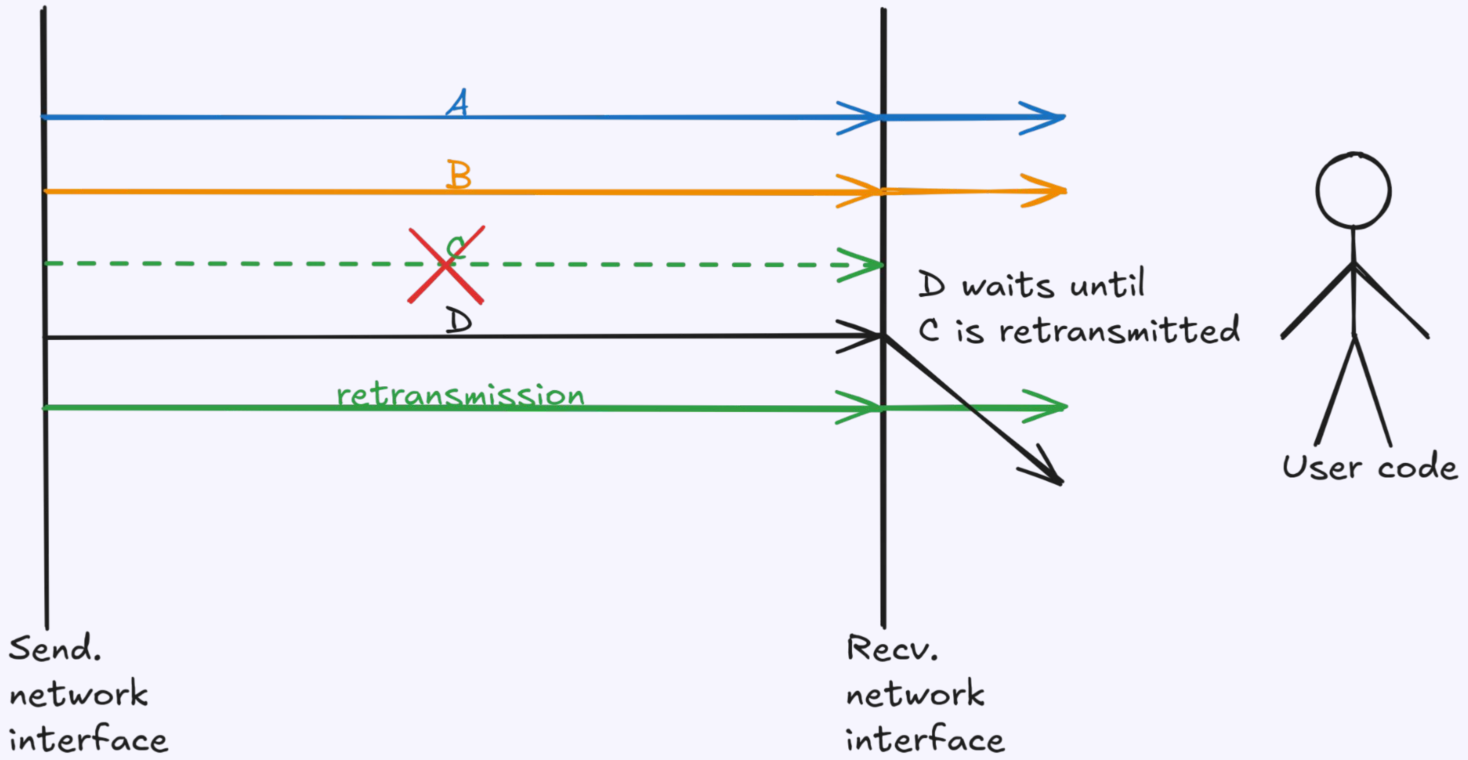
ICE

- **I**nteractive **C**onnectivity **E**stablishment
- A technique used in computer networking to find ways for two computers to talk to each other **as directly as possible** in P2P networking
- Generally uses UDP under the hood

Why do we avoid TCP for real-time communication?

- head-of-line blocking problem
- assuming packets are sent in the following order: C -> B -> A->
- if we lose packet A, we cannot process packets B and C until A is retransmitted
- so we have to wait -> latency
- codecs can deal with lost data to some degree

Head of line blocking problem



Ex. Exchange ICE candidates between Peer Connections

- When there is a new candidate on pc1, add it to pc2.
- Hint: use https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/icecandidate_event
- Hint: use https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/addIceCandidate
- Hint: use **event.candidate**

Ex. Find in chrome://webrtc-internals

- codecs
- packets sent per second
- bits sent per second
- qualityLimitationDurations

Hint: Look for the **outbound-rtp** tab.

What is RTP?

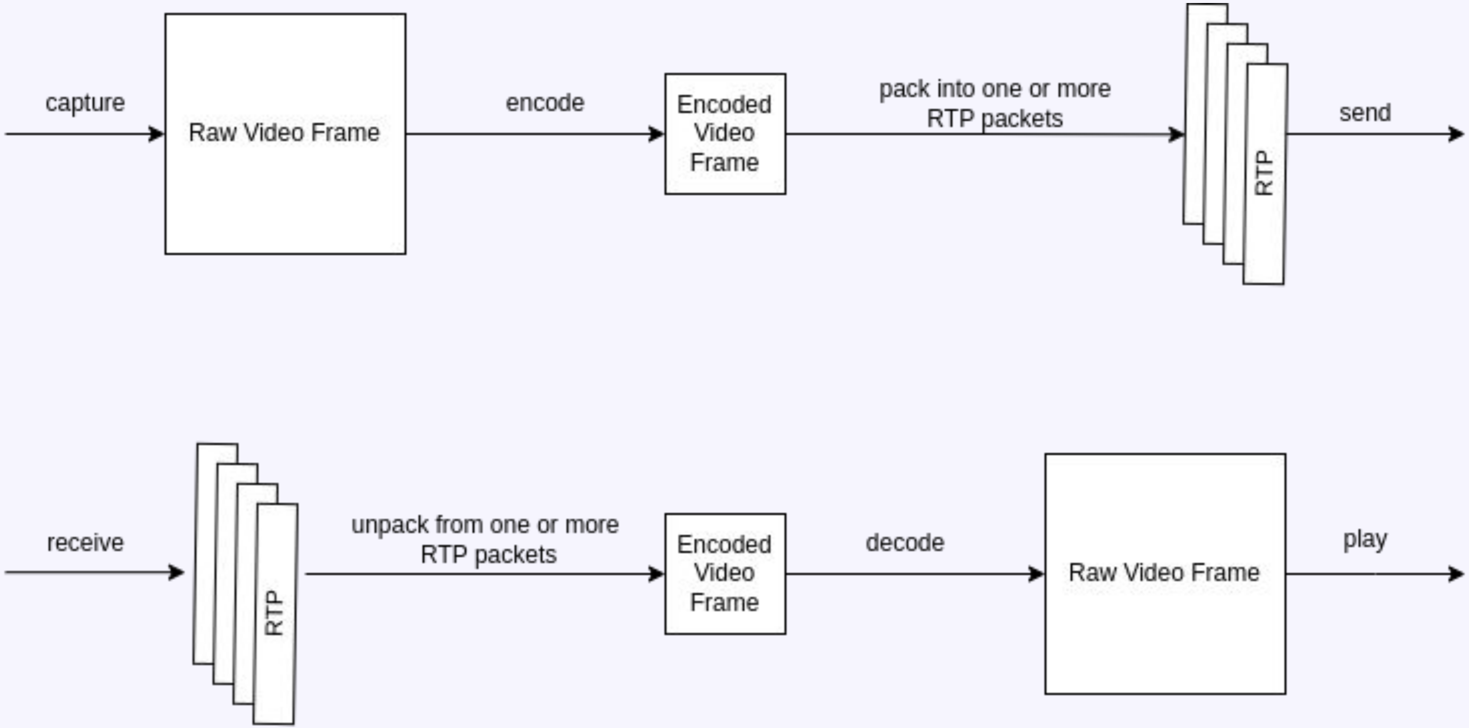
What is RTP?

- **R**eal-time **T**ransport **P**rotocol
- UDP is a very simple protocol – it doesn't contain sequence numbers or timestamps
- We need:
    - means for detecting packet loss and reorders
    - means for synchronization and playback time
    - information about codecs
    - identifiers to map packets to tracks/SDP m-lines

RTP packet

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RTP flow


capture
Raw Video Frame
encode
Encoded Video Frame
pack into one or more RTP packets
RTP
send
receive
RTP
unpack from one or more RTP packets
Encoded Video Frame
decode
Raw Video Frame
play

RTCP – RTP Control Protocol

- Used for:
  - synchronization
  - bandwidth estimation
  - jitter calculation

Network jitter

Ideal conditions:

With jitter:

Ex. Run Chrome with logs and find information about the first RTP packet

chrome --enable-logging='stderr' --vmodule='*/webrtc/*=2'

Important: Close all Chrome instances before running this command.

On MacOS: Substitute `chrome` with

`/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome`

On Windows: ???

Ex. Dump RTP packets sent/received by the browser

- https://github.com/elixir-webrtc/workshop
- ex3

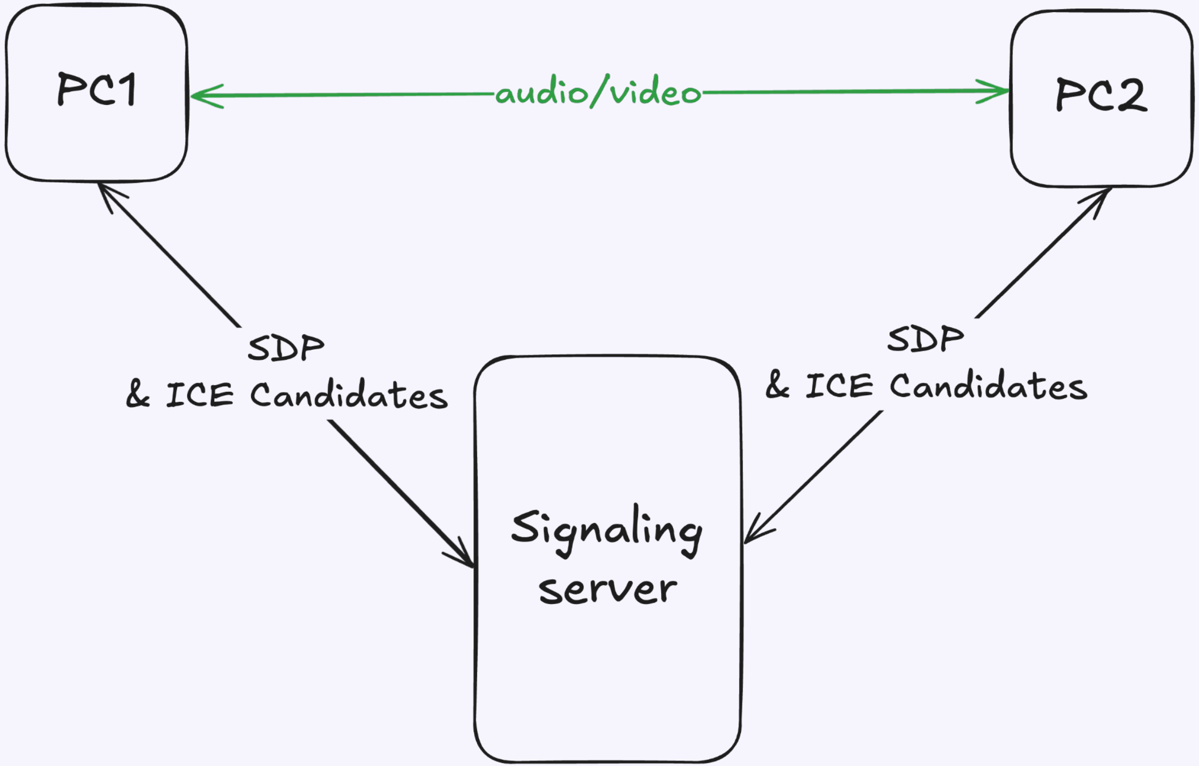
RTP tips&tricks

- The RTP header is not encrypted
- RTP header extensions carry additional information, such as MID – the identifier that allows to bind RTP packet with m-line section in the SDP
- Payload type is a number identifying the codec being used. 96–127 is a dynamic range, the exact meaning is conveyed in the SDP

Signaling

v0.1.0

Signaling


PC1
audio/video
PC2
SDP
& ICE Candidates
SDP
& ICE Candidates
Signaling
server

Signaling

- WebSockets
- SIP
- BroadcastChannel
- this can basically be anything, from an email to a pigeon :)

Ex. Modify the previous example (ex2) to run between two tabs

- https://github.com/elixir-webrtc/workshop/
- ex4

Ex. Modify ex4 to establish bidirectional connection in a single negotiation

- https://github.com/elixir-webrtc/workshop
- ex5

Why does it work?

Ex. Inspect the offer of pc1

v=0
m=audio 9 UDP/TLS/RTP/SAVPF 111 0
a=sendrecv
a=rtpmap:111 opus/48000/2
a=rtpmap:0 PCMU/8000
m=video 9 UDP/TLS/RTP/SAVPF 96 98
a=sendrecv
a=rtpmap:96 VP8/90000
a=rtpmap:98 VP9/90000

Transmitter + Receiver = Transceiver

Transceiver

- can send, receive, or send and receive tracks
- one transceiver can handle only one type of track – either audio or video
- transceiver has a direction – sendonly, recvonly, sendrecv, inactive
- https://developer.mozilla.org/en-US/docs/Web/API/RTCRtpTransceiver

SDP rules

- every transceiver maps to a single mline
- the number of mlines in offer and answer has to be the same
- the number of mlines cannot decrease
- some changes in the connection state require sending a new SDP offer/answer – this is known as renegotiation

Ex. Inspect transceivers in pc1

- Use https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/getTransceivers
- Hint: you can pin **pc1** to the **window** to have an access to **pc1** in the web browser's console: **window.pc1 = pc1**
- How many transceivers are there?
- What are their directions?
- What's the difference between direction and currentDirection?

Perfect Negotiation

- What if both sides want to modify the connection at the same time?
- One of the sides has to be the *polite* one and revert its changes.
- Use **setLocalDescription({type: “rollback”})**
- https://developer.mozilla.org/en-US/docs/Web/API/RTCSessionDescription/type
- https://blog.mozilla.org/webrtc/perfect-negotiation-in-webrtc/

WHIP/WHEP

v0.1.0

WHIP/WHEP

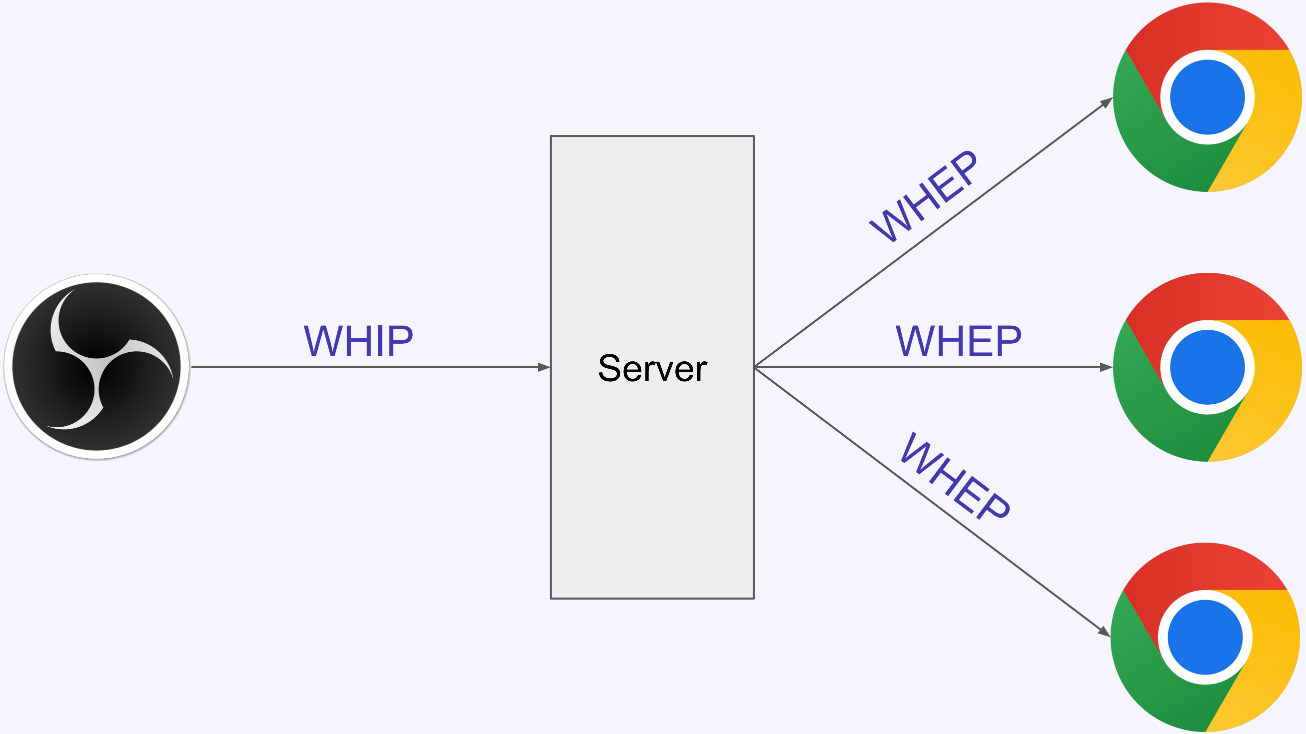
- WebRTC doesn’t standardize the signaling mechanism
- there are a lot of simple scenarios that don’t use renegotiation (e.g. streaming)

WHIP

- **W**ebRTC-**H**TTP **I**ngestion **P**rotocol
- A simple protocol based on HTTP for supporting WebRTC as a media ingestion method
- Describes a very specific usage of WebRTC
- Uses HTTP for exchanging SDP offer/answer and ICE candidates
- Can only send up to one audio and one video
- No renegotiation possible
- Example: OBS can use WHIP to stream media to a server

WHEP

- **W**ebRTC-**H**TTP **E**gress **P**rotocol
- The same as WHEP but for egress
- Web clients can use it to receive media from a server



v0.1.0

Ex. Stream from OBS to the Broadcaster

- Open OBS and go to **Settings > Stream.**
- Change **Service** to **WHIP.**
- Pass https://bigfish.jellyfish.ovh/api/whip as the **Server** value and *webrtcworkshop* as the **Bearer Token**. Press **Apply.**
- Go to **Settings > Output.**
- Set **bitrate** to 500kbps. Press **Apply.**
- Choose a source of your liking (e.g. a webcam feed) and press **Start Streaming.**
- Access https://bigfish.jellyfish.ovh.

Mastering Transceivers

v0.1.0

Ex. Negotiate a unidirectional session without MediaStreamTracks

Follow: https://hexdocs.pm/ex_webrtc/mastering_transceivers.html#warmup

Ex. Offer to receive data

Follow:
https://hexdocs.pm/ex_webrtc/mastering_transceivers.html#offer-to-receive-data

Ex. Negotiate bidirectional session without MediaStreamTracks

Follow:
https://hexdocs.pm/ex_webrtc/mastering_transceivers.html#bidirectional-connection-using-a-single-negotiation

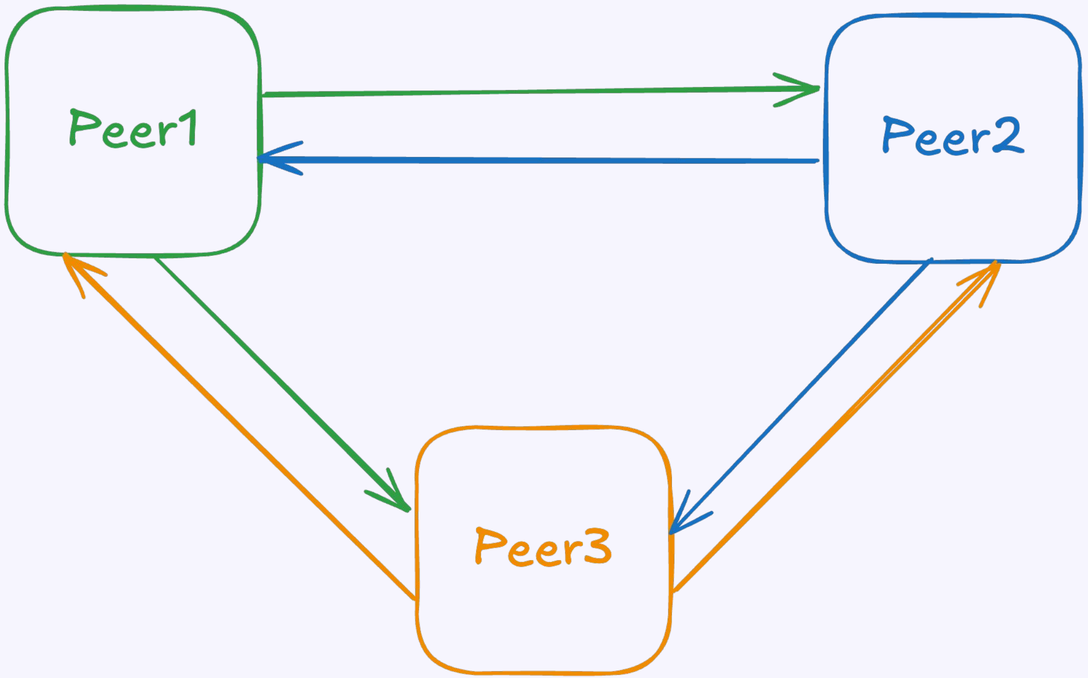
Ex. Reject the incoming track

Follow:
https://hexdocs.pm/ex_webrtc/mastering_transceivers.html#rejecting-incoming-track

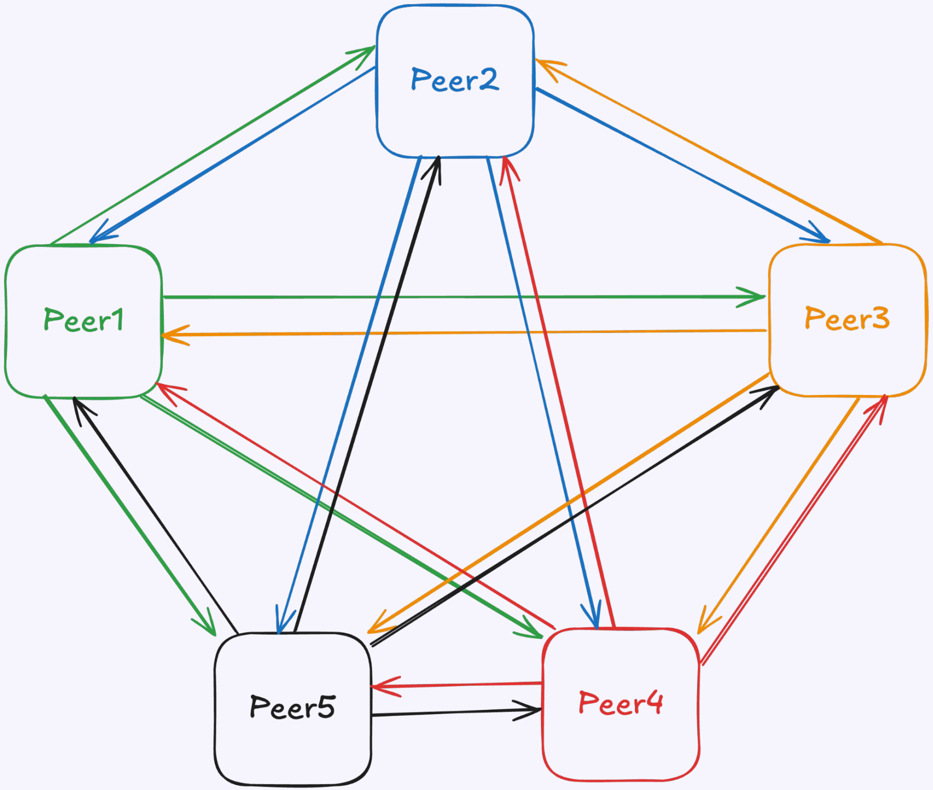
Media Servers

v0.1.0

Media Servers

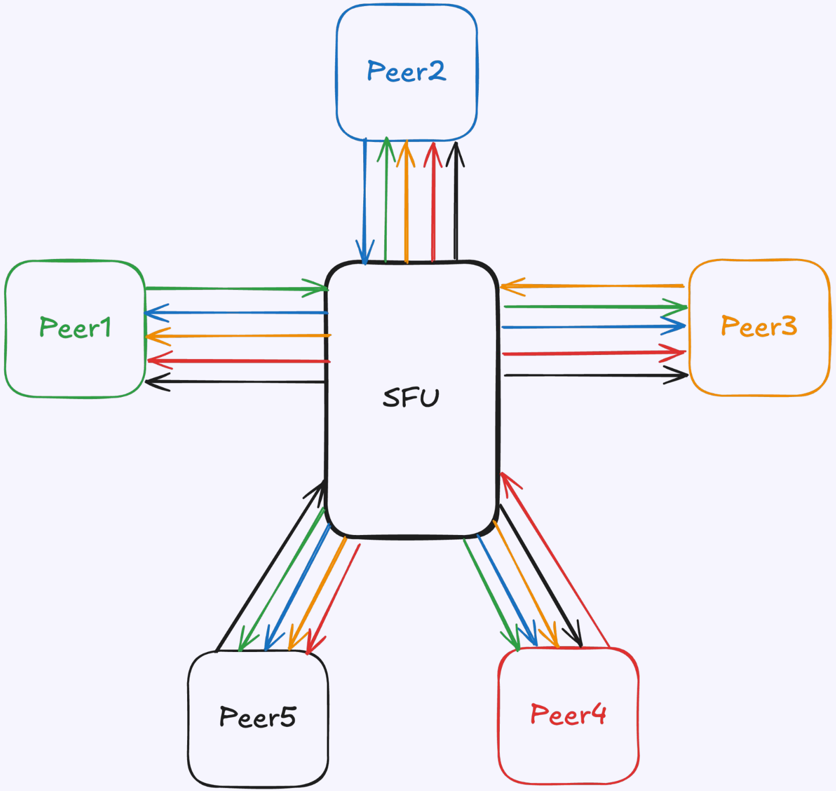


Media Servers

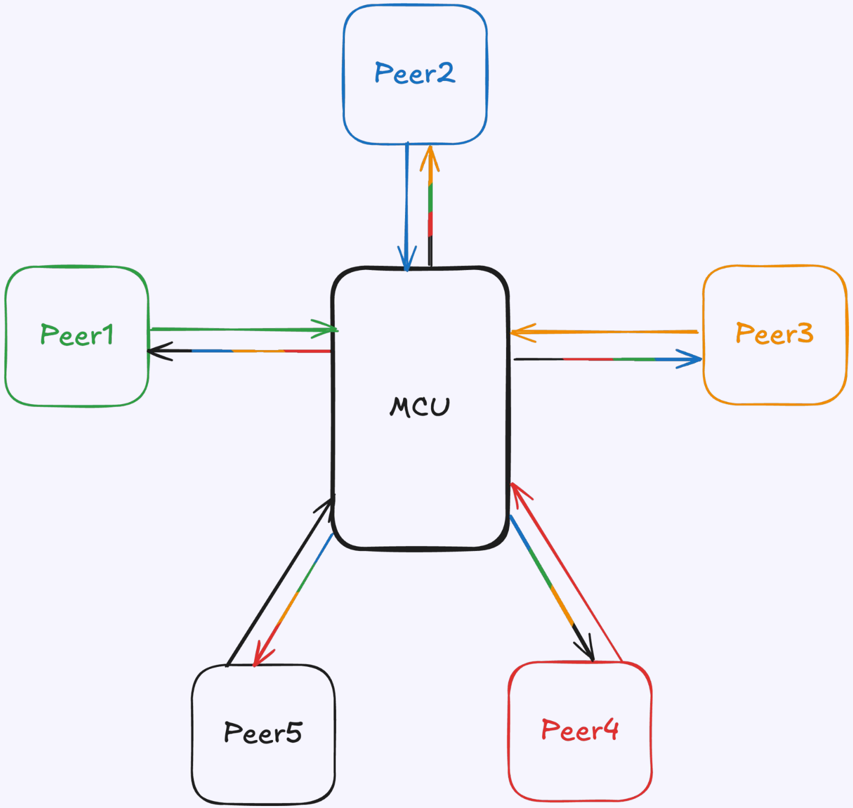


v0.1.0

Media Servers



Media Servers



Media Servers

- hide the negotiation process – SDKs
- operate on higher abstraction level
- perform a lot of end to end optimizations
    - display only N video tiles
    - adaptive streaming – send different qualities (resolutions, FPS) depending on the needs (bandwidth, grid layout, user preferences etc.)
    - don’t send video when the user switches tabs
- can provide additional features – recordings, transcriptions

Ex. Videoconferencing app using Fishjam Cloud

- https://tinyurl.com/webrtcworkshop
- https://github.com/elixir-webrtc/workshop/
- ex6

Thank you! :)

https://github.com/elixir-webrtc/workshop

Michał Śledź
@mickel8
@mickel8v2

Jakub Pisarek
@sgfn
@jpisarek