WebRTC on Urbit

Edward Amsden

Black River Software

edward@blackriversoft.com

April 29, 2021

What is WebRTC?

What is WebRTC?

Video calls in the browser!

What is WebRTC?

Video calls in the browser!

Yes. But more generally:

What is WebRTC?

Video calls in the browser!

Yes. But more generally:

- Protocol specification

What is WebRTC?

Video calls in the browser!

Yes. But more generally:

- Protocol specification
- Browser API specification

What is WebRTC?

Video calls in the browser!

Yes. But more generally:

- Protocol specification
- Browser API specification
  - Media device acquisition
  - Peer-to-peer channel setup and interaction

I want to build a WebRTC app!

I want to build a WebRTC app!

You will need:

I want to build a WebRTC app!

You will need:

- ICE servers

I want to build a WebRTC app!

You will need:

- ICE servers
- A side channel for SDP messages

I want to build a WebRTC app!

You will need:

- ICE servers
- A side channel for SDP messages
- Media or data to send to your peer

I want to build a WebRTC app!

You will need:

- ICE servers
- A side channel for SDP messages
- ~~Media or data to send to your peer~~

I want to build a WebRTC app!

You will need:

- ICE servers
- A side channel for SDP messages
- ~~Media or data to send to your peer~~

Given these, `RTCPeerConnection` provides:

I want to build a WebRTC app!

You will need:

- ICE servers
- A side channel for SDP messages
- ~~Media or data to send to your peer~~

Given these, `RTCPeerConnection` provides:

- Data channels

I want to build a WebRTC app!

You will need:

- ICE servers
- A side channel for SDP messages
- ~~Media or data to send to your peer~~

Given these, `RTCPeerConnection` provides:

- Data channels
- Media stream transit

I want to build a WebRTC app!

You will need:

- ICE servers
- A side channel for SDP messages
- ~~Media or data to send to your peer~~

I want to build a WebRTC app!

You will need:

- ICE servers
  `icepond`
- A side channel for SDP messages
- ~~Media or data to send to your peer~~

I want to build a WebRTC app!

You will need:

- ICE servers
  `icepond`
- A side channel for SDP messages
  `switchboard`
- ~~Media or data to send to your peer~~

ICE Servers: STUN and TURN

Where do we send packets?

[1] https://tailscale.com/blog/how-nat-traversal-works/

ICE Servers: STUN and TURN

Where do we send packets? Where our peer tells us to.

[1] https://tailscale.com/blog/how-nat-traversal-works/

ICE Servers: STUN and TURN

Where do we send packets? Where our peer tells us to.

Where do we tell our peer to send packets?[1]

[1] https://tailscale.com/blog/how-nat-traversal-works/

ICE Servers: STUN and TURN

Where do we send packets? Where our peer tells us to.

Where do we tell our peer to send packets?[1]

- What is our public internet address?

[1] https://tailscale.com/blog/how-nat-traversal-works/

ICE Servers: STUN and TURN

Where do we send packets? Where our peer tells us to.

Where do we tell our peer to send packets?[1]

- What is our public internet address?
- What is our port on our public internet address?

[1] https://tailscale.com/blog/how-nat-traversal-works/

ICE Servers: STUN and TURN

Where do we send packets? Where our peer tells us to.

Where do we tell our peer to send packets?[1]

- What is our public internet address?
- What is our port on our public internet address?

Solution:

[1] https://tailscale.com/blog/how-nat-traversal-works/

ICE Servers: STUN and TURN

Where do we send packets? Where our peer tells us to.

Where do we tell our peer to send packets?[1]

- What is our public internet address?
- What is our port on our public internet address?

Solution:

- STUN: "You are talking to me from this IP and port"

[1] `https://tailscale.com/blog/how-nat-traversal-works/`

ICE Servers: STUN and TURN

Where do we send packets? Where our peer tells us to.

Where do we tell our peer to send packets?[1]

- What is our public internet address?
- What is our port on our public internet address?

Solution:

- STUN: "You are talking to me from this IP and port"
- TURN: "Your peer can talk to me on this IP and port and I’ll relay to you."

[1] https://tailscale.com/blog/how-nat-traversal-works/

ICE Servers: STUN and TURN

Where do we send packets? Where our peer tells us to.

Where do we tell our peer to send packets?[1]

- What is our public internet address?
- What is our port on our public internet address?

Solution:

- STUN: "You are talking to me from this IP and port"
- TURN: "Your peer can talk to me on this IP and port and I’ll relay to you."

An ICE Server is either a STUN server or a TURN server.

[1] https://tailscale.com/blog/how-nat-traversal-works/

`icepond`: ICE server acquisition

`icepond`: ICE server acquisition

How do we find ICE servers?

`icepond`: ICE server acquisition

How do we find ICE servers?

- From our central server

`icepond`: ICE server acquisition

How do we find ICE servers?

- ~~From our central server~~

`icepond`: ICE server acquisition

How do we find ICE servers?

- ~~From our central server~~
- Hardcoded into our application

`icepond`: ICE server acquisition

How do we find ICE servers?

- ~~From our central server~~
- ~~Hardcoded into our application~~

`icepond`: ICE server acquisition

How do we find ICE servers?

- ~~From our central server~~
- ~~Hardcoded into our application~~
- User configuration

`icepond`: ICE server acquisition

How do we find ICE servers?

- ~~From our central server~~
- ~~Hardcoded into our application~~
- ~~User configuration~~

`icepond`: ICE server acquisition

How do we find ICE servers?

- ~~From our central server~~
- ~~Hardcoded into our application~~
- ~~User configuration~~
- Hosted servers and user-configurable discovery.

`icepond`: ICE server acquisition

How do we find ICE servers?

- ~~From our central server~~
- ~~Hardcoded into our application~~
- ~~User configuration~~
- **Hosted servers and user-configurable discovery.**

`icepond`: ICE server acquisition

How do we find ICE servers?

- ~~From our central server~~
- ~~Hardcoded into our application~~
- ~~User configuration~~
- **Hosted servers and user-configurable discovery.**

For instance, stars may well include ICE servers in the infrastructure they provide to sponsored planets.

Signaling

Signaling

How do we tell receivers of incoming calls?

Signaling

How do we tell receivers of incoming calls?
How do we tell our peer where to talk to us?

Signaling

How do we tell receivers of incoming calls?
How do we tell our peer where to talk to us?

- Centralized database of calls

Signaling

How do we tell receivers of incoming calls?
How do we tell our peer where to talk to us?

- ~~Centralized database of calls~~

Signaling

How do we tell receivers of incoming calls?
How do we tell our peer where to talk to us?

- ~~Centralized database of calls~~
- Peer-to-peer exchange

Signaling

How do we tell receivers of incoming calls?
How do we tell our peer where to talk to us?

- ~~Centralized database of calls~~
- **Peer-to-peer exchange**

Signaling

How do we tell receivers of incoming calls?
How do we tell our peer where to talk to us?

- ~~Centralized database of calls~~
- **Peer-to-peer exchange**

My app ⟷ my `switchboard` ⟷ your `switchboard` ⟷ Your app

`switchboard`: Call setup and SDP relay

`switchboard`: Call setup and SDP relay

- Takes pokes initiate calls.

`switchboard`: Call setup and SDP relay

- Takes pokes initiate calls.
- Notifies applications of incoming calls.

`switchboard`: Call setup and SDP relay

- Takes pokes initiate calls.
- Notifies applications of incoming calls.
- Receives pokes with SDP messages.

`switchboard`: Call setup and SDP relay

- Takes pokes initiate calls.
- Notifies applications of incoming calls.
- Receives pokes with SDP messages.
- Notifies applications of SDP messages.

Interface

Interface

- icepond

Interface

- icepond
  - `icepond` Nodejs package in `icepond-js` directory

Interface

- icepond
  - `icepond` Nodejs package in `icepond-js` directory
  - `Icepond` class: get ICE servers as they are discovered

Interface

- icepond
  - `icepond` Nodejs package in `icepond-js` directory
  - `Icepond` class: get ICE servers as they are discovered
- switchboard

Interface

- icepond
  - `icepond` Nodejs package in `icepond-js` directory
  - `Icepond` class: get ICE servers as they are discovered
- switchboard
  - `switchboard` Nodejs package in `switchboard-js` directory

Interface

- icepond
  - `icepond` Nodejs package in `icepond-js` directory
  - `Icepond` class: get ICE servers as they are discovered
- switchboard
  - `switchboard` Nodejs package in `switchboard-js` directory
  - `UrbitRTCApp` class: make and receive WebRTC calls

Interface

- icepond
  - `icepond` Nodejs package in `icepond-js` directory
  - `Icepond` class: get ICE servers as they are discovered
- switchboard
  - `switchboard` Nodejs package in `switchboard-js` directory
  - `UrbitRTCApp` class: make and receive WebRTC calls
  - `UrbitRTCPeerConnection` class: `RTCPeerConnection` with SDP signalling implemented over Urbit

Demo

Writing an Urbit WebRTC App - Setting up

Writing an Urbit WebRTC App - Setting up

- Initialize an `Urbit` airlock (see the `@urbit/http-api` library).

Writing an Urbit WebRTC App - Setting up

- Initialize an `Urbit` airlock (see the `@urbit/http-api` library).
- Create an `UrbitRTCApp` instance

Writing an Urbit WebRTC App - Setting up

- Initialize an `Urbit` airlock (see the `@urbit/http-api` library).
- Create an `UrbitRTCApp` instance
- Register a handler for the `’incomingcall’` event.

Writing an Urbit WebRTC App - Setting up

- Initialize an `Urbit` airlock (see the `@urbit/http-api` library).
- Create an `UrbitRTCApp` instance
- Register a handler for the `’incomingcall’` event.
  - Call `reject()` on the event to reject an incoming call.

Writing an Urbit WebRTC App - Setting up

- Initialize an `Urbit` airlock (see the `@urbit/http-api` library).
- Create an `UrbitRTCApp` instance
- Register a handler for the `’incomingcall’` event.
  - Call `reject()` on the event to reject an incoming call.
  - Call `answer()` on the event to answer. This returns a peer connection.

Writing an Urbit WebRTC App - Setting up

- Initialize an `Urbit` airlock (see the `@urbit/http-api` library).
- Create an `UrbitRTCApp` instance
- Register a handler for the `’incomingcall’` event.
  - Call `reject()` on the event to reject an incoming call.
  - Call `answer()` on the event to answer. This returns a peer connection.
- Call `initialize()`

Writing an Urbit WebRTC App - Calls

Writing an Urbit WebRTC App - Calls

- Place calls using the `call()` method. This returns a peer connection. Or...

Writing an Urbit WebRTC App - Calls

- Place calls using the `call()` method. This returns a peer connection. Or...
- Start with a peer connection obtained from calling `answer()` on an event.

Writing an Urbit WebRTC App - Calls

- Place calls using the `call()` method. This returns a peer connection. Or...
- Start with a peer connection obtained from calling `answer()` on an event.
- Create a `Icepond` instance.

Writing an Urbit WebRTC App - Calls

- Place calls using the `call()` method. This returns a peer connection. Or...
- Start with a peer connection obtained from calling `answer()` on an event.
- Create a `Icepond` instance.
- Register a handler for the `’iceserver’` event to add ICE servers to the peer connection.

Writing an Urbit WebRTC App - Calls

- Place calls using the `call()` method. This returns a peer connection. Or...
- Start with a peer connection obtained from calling `answer()` on an event.
- Create a `Icepond` instance.
- Register a handler for the `’iceserver’` event to add ICE servers to the peer connection.
- Add media or data streams.

Writing an Urbit WebRTC App - Calls

- Place calls using the `call()` method. This returns a peer connection. Or...
- Start with a peer connection obtained from calling `answer()` on an event.
- Create a `Icepond` instance.
- Register a handler for the `’iceserver’` event to add ICE servers to the peer connection.
- Add media or data streams.
- Call `initialize()` on both the `Icepond` instance and the peer connection.

Result

Result

This is a WebRTC library which

Result

This is a WebRTC library which commoditizes external infrastructure dependencies,

Result

This is a WebRTC library which commoditizes external infrastructure dependencies, unifies personal identity and addressing, and

Result

This is a WebRTC library which commoditizes external infrastructure dependencies, unifies personal identity and addressing, and removes centralized coordination for call negotiation.

Result

This is a WebRTC library which commoditizes external infrastructure dependencies, **unifies personal identity and addressing**, and removes centralized coordination for call negotiation.

Questions

Links

- Repository: `https://github.com/black-river-software/urbit-webrtc`
- Clay desk: `~dirpub-ritpub-sipsyl %webrtc`