WebRTC and PeerJS

Tecnologie Internet

a.a. 2022/2023

WebRTC

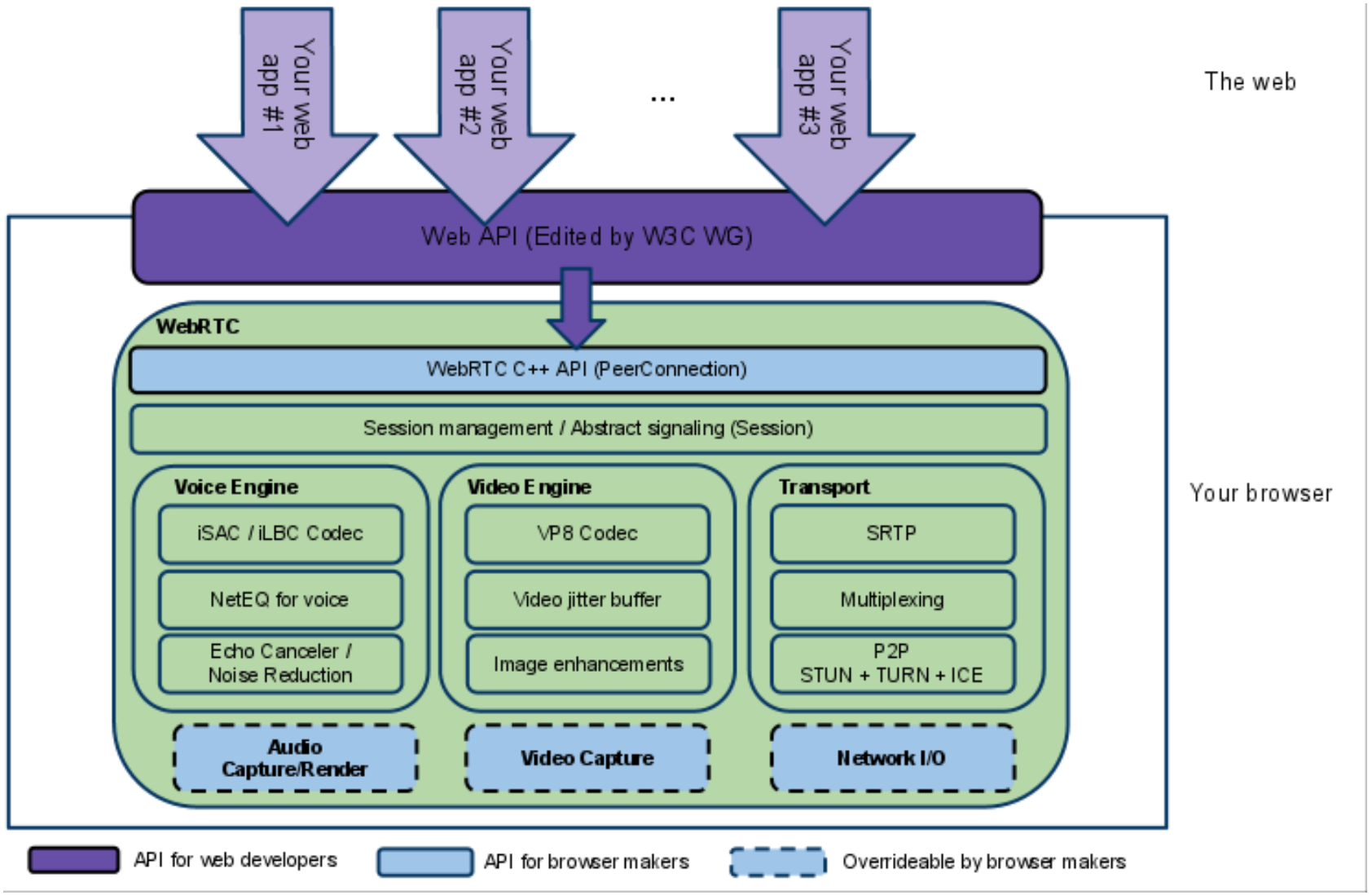
WebRTC is a free, open project that provides browsers and mobile applications with **Real-Time Communications (RTC)** capabilities via simple APIs. The WebRTC components have been optimized to best serve this purpose.

WebRTC offers web application developers the ability to write rich, real-time multimedia applications (e.g., video chat) on the web, without requiring plugins, downloads or installs.

Its purpose is to help build a strong RTC platform that works across multiple web browsers, across multiple platforms.

Architecture

https://webrtc.github.io/webrtc-org/architecture/



No plugins

- Many web applications already use RTC, but need downloads, native apps or plugins.
- Downloading, installing and updating plugins can be complex, error prone and annoying.
- Plugins can be difficult to deploy, debug, troubleshoot, test and maintain—and may require licensing and integration with complex, expensive technology. It's often difficult to persuade people to install plugins in the first place!

WebRTC applications

WebRTC applications need to do several things:

- Get/provide **streaming** audio, video or other data.
- Get **network information** such as IP addresses and ports, and exchange this with other WebRTC clients (known as *peers*) to enable connection, even through NATs and firewalls.
- Coordinate **signaling communication** to report errors and initiate or close sessions.
- Exchange information about media and client **capability**, such as resolution and codecs.

WebRTC APIs

WebRTC has several **JavaScript APIs**:

MediaStream aka getUserMedia(): capture audio and video

RTCPeerConnection: stream audio and video between users

RTCDataChannel: stream data between users

http://w3c.github.io/webrtc-pc/

STUN and TURN

WebRTC is designed to work peer-to-peer, so users can connect by the most direct route possible.

However, WebRTC is built to cope with real-world networking: client applications need to **traverse NAT gateways and firewalls**, and peer-to-peer networking needs fallbacks in case direct connection fails.

As part of this process, the WebRTC APIs use **STUN servers to get the IP address of your computer**, and **TURN servers to function as relay servers** in case peer-to-peer communication fails.

STUN: RFC 3489
TURN: RFC 5766

Security

Encryption is mandatory for all WebRTC components, and its JavaScript APIs can only be used from **secure origins** (HTTPS or localhost).

https://tools.ietf.org/html/draft-ietf-rtcweb-security-arch-16

> The basic assumption of this architecture is that network resources exist in a hierarchy of trust, rooted in the browser, which serves as the user's Trusted Computing Base (TCB). Any security property which the user wishes to have enforced must be ultimately guaranteed by the browser (or transitively by some property the browser verifies). Conversely, if the browser is compromised, then no security guarantees are possible.

http://w3c.github.io/webrtc-pc/#privacy-and-security-considerations

The WebRTC specification assumes that once the Web page has been allowed to access media, it is free to share that media with other entities as it chooses.

PeerJS

PeerJS wraps the browser's WebRTC implementation to provide a complete, configurable, and easy-to-use peer-to-peer connection API.

Equipped with nothing but an ID, a peer can create a P2P data or media stream connection to a remote peer.

https://peerjs.com/

Setup

Include the library:

```
<script src="https://unpkg.com/peerjs@1.4.7/dist/peerjs.min.js"></script>
```

Create a peer:

```
var peer = new Peer();
```

```
var peer = new Peer('TI20182019-Peer1');
```

Data connections

Connect:

```
var conn = peer.connect('another-peers-id');
// on open will be launch when you successfully connect to PeerServer
conn.on('open', function(){
  // here you have conn.id
  conn.send('hi!');
});
```

Receive:

```
peer.on('connection', function(conn) {
  conn.on('data', function(data){
    // Will print 'hi!'
    console.log(data);
  });
});
```

Media calls

Call:

```
getUserMedia({video: true, audio: true}, function(stream) {
  var call = peer.call('another-peers-id', stream);
  call.on('stream', function(remoteStream) {
    // Show stream in some video/canvas element.
  });
}, function(err) {
  console.log('Failed to get local stream' ,err);
});
```

Media calls

Answer:

```
peer.on('call', function(call) {
  getUserMedia({video: true, audio: true}, function(stream) {
    call.answer(stream); // Answer the call with an A/V stream.
    call.on('stream', function(remoteStream) {
      // Show stream in some video/canvas element.
    });
  }, function(err) {
    console.log('Failed to get local stream' ,err);
  });
});
```

PeerServer

To broker connections, PeerJS connects to a PeerServer.

Note that no peer-to-peer data goes through the server; the server acts only as a connection broker (i.e., as a bootstrap server).

1) PeerJS offers a free cloud-hosted version of PeerServer.
   https://peerjs.com/peerserver.html

2) The user can run his/her own PeerServer (it is written in Node.js)
   https://github.com/peers/peerjs-server