Proposed Method with Architecture

**WebRTC** (Web Real-Time Communication) is a technology which enables Web applications and sites to capture and optionally stream audio and/or video media, as well as to exchange arbitrary data between browsers without requiring an intermediary. The set of standards that comprise WebRTC makes it possible to share data and perform teleconferencing peer-to-peer, without requiring that the user installs plug-ins or any other third-party software.

WebRTC consists of several interrelated APIs and protocols which work together to achieve this. The documentation you'll find here will help you understand the fundamentals of WebRTC, how to set up and use both data and media connections, and more.

**A very short history of WebRTC**

One of the last major challenges for the web is to enable human communication via voice and video: Real Time Communication, RTC for short. RTC should be as natural in a web application as entering text in a text input. Without it, we're limited in our ability to innovate and develop new ways for people to interact.

Historically, RTC has been corporate and complex, requiring expensive audio and video technologies to be licensed or developed in house. Integrating RTC technology with existing content, data and services has been difficult and time consuming, particularly on the web.

Gmail video chat became popular in 2008, and in 2011 Google introduced Hangouts, which use the Google Talk service (as did Gmail). Google bought GIPS, a company which had developed many components required for RTC, such as codecs and echo cancellation techniques. Google open sourced the technologies developed by GIPS and engaged with relevant standards bodies at the IETF and W3C to ensure industry consensus. In May 2011, Ericsson built [the first implementation of WebRTC](https://labs.ericsson.com/developer-community/blog/beyond-html5-peer-peer-conversational-video).

WebRTC implemented open standards for real-time, plugin-free video, audio and data communication. The need was real:

* Many web services used RTC, but needed downloads, native apps or plugins. Those included Skype, Facebook and Google Hangouts.
* Downloading, installing and updating plugins is complex, error prone and annoying.
* Plugins are 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!

The guiding principles of the WebRTC project are that its APIs should be open source, free, standardized, built into web browsers and more efficient than existing technologies.

App development using WebRTC

Creating a new application based on the WebRTC technologies can be overwhelming if you're unfamiliar with the APIs. In this section we will show how to get started with the various APIs in the WebRTC standard, by explaining a number of common use cases and code snippets for solving those.

WebRTC APIs

The WebRTC standard covers, on a high level, two different technologies: media capture devices and peer-to-peer connectivity.

Media capture devices includes video cameras and microphones, but also screen capturing "devices". For cameras and microphones, we use navigator.mediaDevices.getUserMedia() to capture MediaStreams. For screen recording, we use navigator.mediaDevices.getDisplayMedia() instead.

The peer-to-peer connectivity is handled by the RTCPeerConnection interface. This is the central point for establishing and controlling the connection between two peers in WebRTC.

WebRTC is used in various apps like WhatsApp, Facebook Messenger, appear.in and platforms such as TokBox. WebRTC has also been integrated with [WebKitGTK+](https://labs.ericsson.com/developer-community/blog/beyond-html5-conversational-voice-and-video-implemented-webkit-gtk) and [Qt](https://www.youtube.com/watch?v=Vm5ebKWKNE8) native apps.

WebRTC implements three APIs:

* [MediaStream](https://www.html5rocks.com/en/tutorials/webrtc/basics/#toc-mediastream) (aka getUserMedia)
* [RTCPeerConnection](https://www.html5rocks.com/en/tutorials/webrtc/basics/#toc-rtcpeerconnection)
* [RTCDataChannel](https://www.html5rocks.com/en/tutorials/webrtc/basics/#toc-rtcdatachannel)

The APIs are defined in two specs:

* [WebRTC](https://w3c.github.io/webrtc-pc/)
* [getUserMedia](https://www.w3.org/TR/mediacapture-streams)

All three APIs are supported on mobile and desktop by Chrome, Safari, Firefox, Edge and Opera.

Our application shows how to use all three APIs to build a simple application for video chat and file sharing.

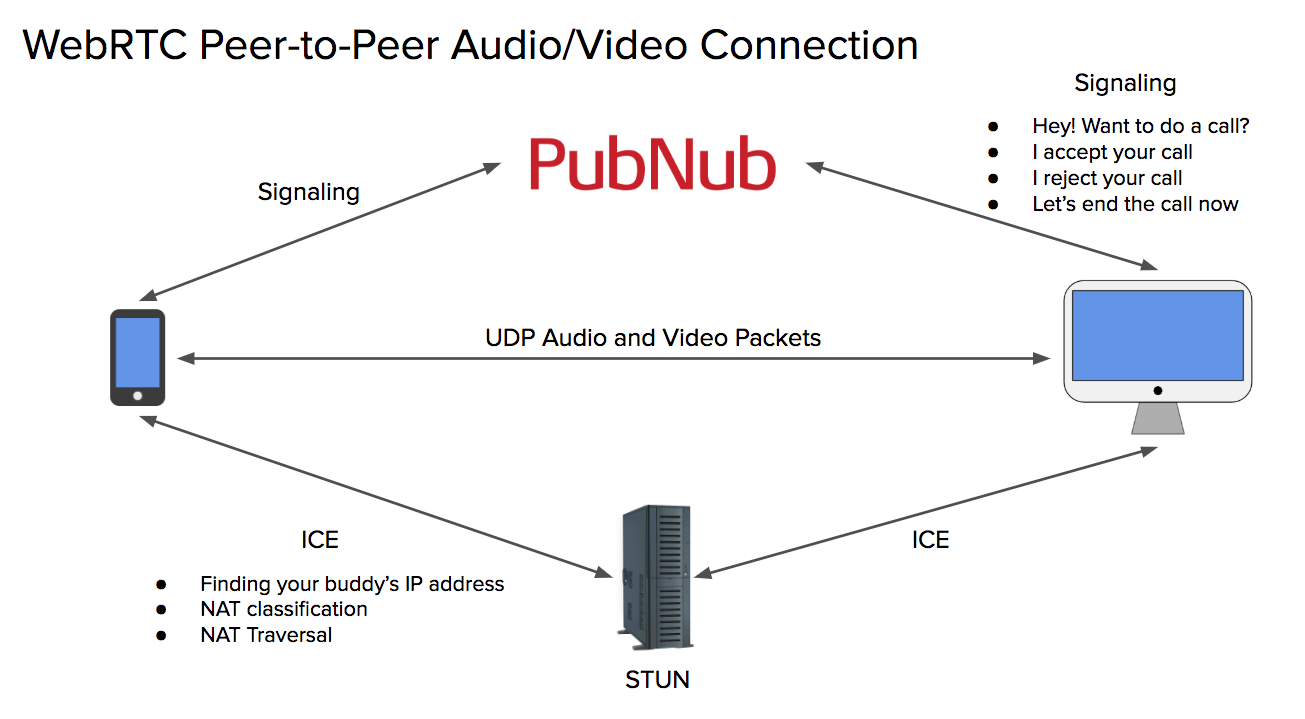
WebRTC Streaming Architecture

Video chat is established on two or more client devices using the WebRTC protocol. The connection can be made using one of two modes. The first mode is peer-to-peer, meaning audio and video packets are streamed directly from client to client with [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol). This works as long as both machines have an IP address that is accessible by the public internet.

Relying on peer-to-peer connections for browser video chat is not wise in production apps. It is common for the Interactive Connectivity Establishment or ICE framework to fail to establish a connection between two users when one or both are behind advanced LAN security.

To mitigate this, you can set your [RTC Configuration](https://developer.mozilla.org/en-US/docs/Web/API/RTCConfiguration) to first attempt peer-to-peer, and then fall back to relayed connection if peer-to-peer fails.

If publicly accessible IP addresses are not an option, like on enterprise WiFi networks, a WebRTC connection must be established over TCP using a TURN server. The ICE framework will decide if this is necessary as users are trying to connect. A TURN server acts as a relay for video and audio data. TURN instances require bandwidth and machine time – so it’s not free like peer-to-peer streaming.



TWILIO