§3.WebRTC

Коммуникации через веб-страницу

Что такое WebRTC?

- WebRTC (real-time communications) – это сетевой протокол с открытым исходным кодом, предназначенный для организации голосовой и видеосвязи через Интернет в режиме реального времени.

Для чего нужен WebRTC

- Множество Web сервисов используют коммуникацию в реальном времени, однако требуют скачивания (Skype, Viber, ~~Google Talk plugin,~~ др.)
- Эти приложения, плагины и сервисы нужно обновлять и настраивать отдельно
- Зачастую людей тяжело заставить установить и обновлять какой-то плагин или приложение.

2010

- WebRTC основывается на продукте от компании Global IP Solution (GIPS), которая была куплена компанией Google в мае 2010-го. Технология использует свои аудиокодеки и открытый видеоформат VP8 (WebM).

Год 2011, 2012

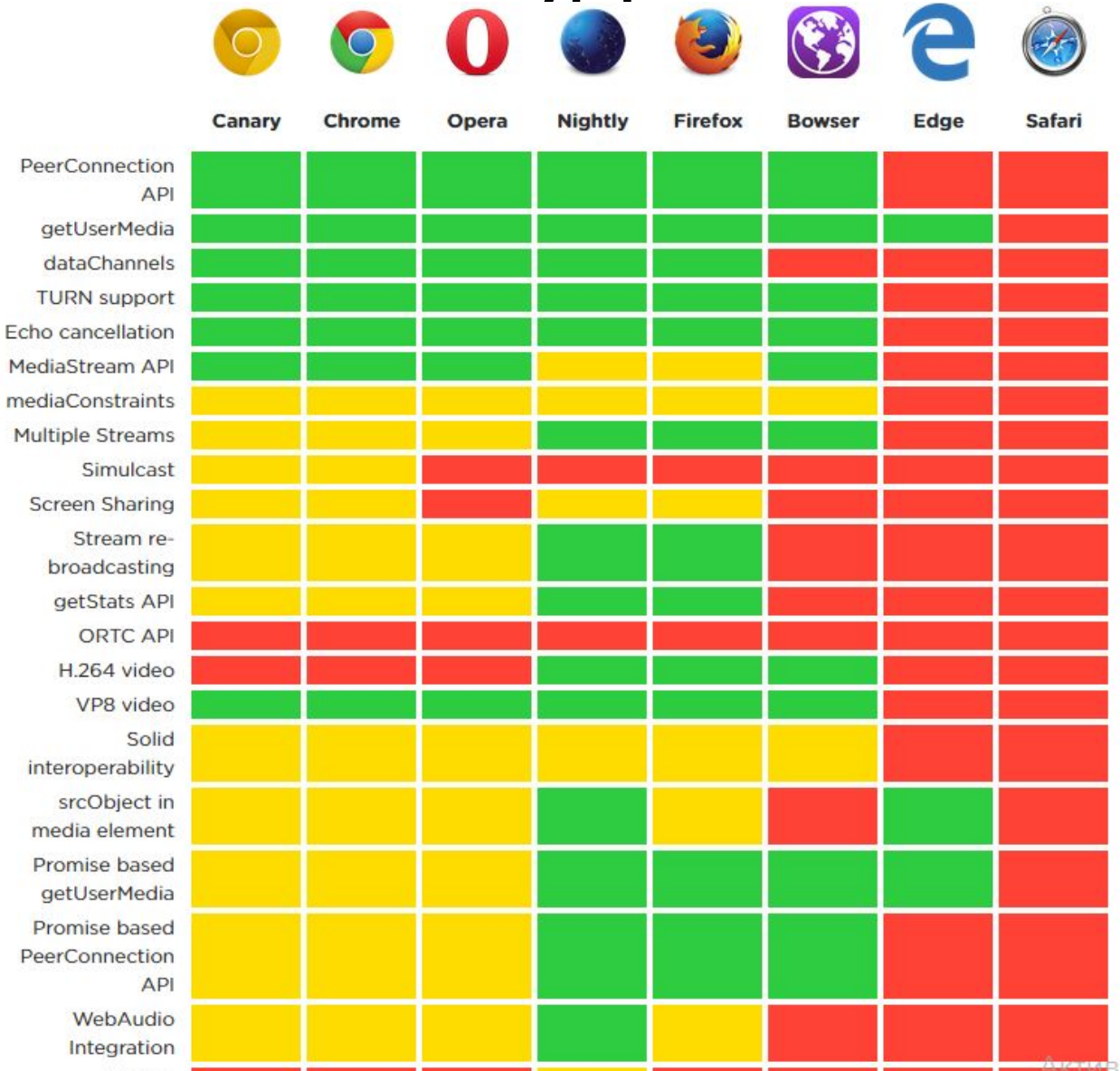
- В браузер Google Chrome технология WebRTC была добавлена в январе 2012 года
- В апреле 2012 года на парижском саммите IETF 83 команда разработчиков Mozilla показала экспериментальную сборку браузера Firefox со встроенной поддержкой WebRTC (был продемонстрирован видеочат между двумя интернет-обозревателями на основе этой технологии).
- Первые сборки Opera с поддержкой WebRTC появились (в рамках Opera Labs) еще раньше – в октябре 2011-го.

Hello Chrome, it's Firefox calling!

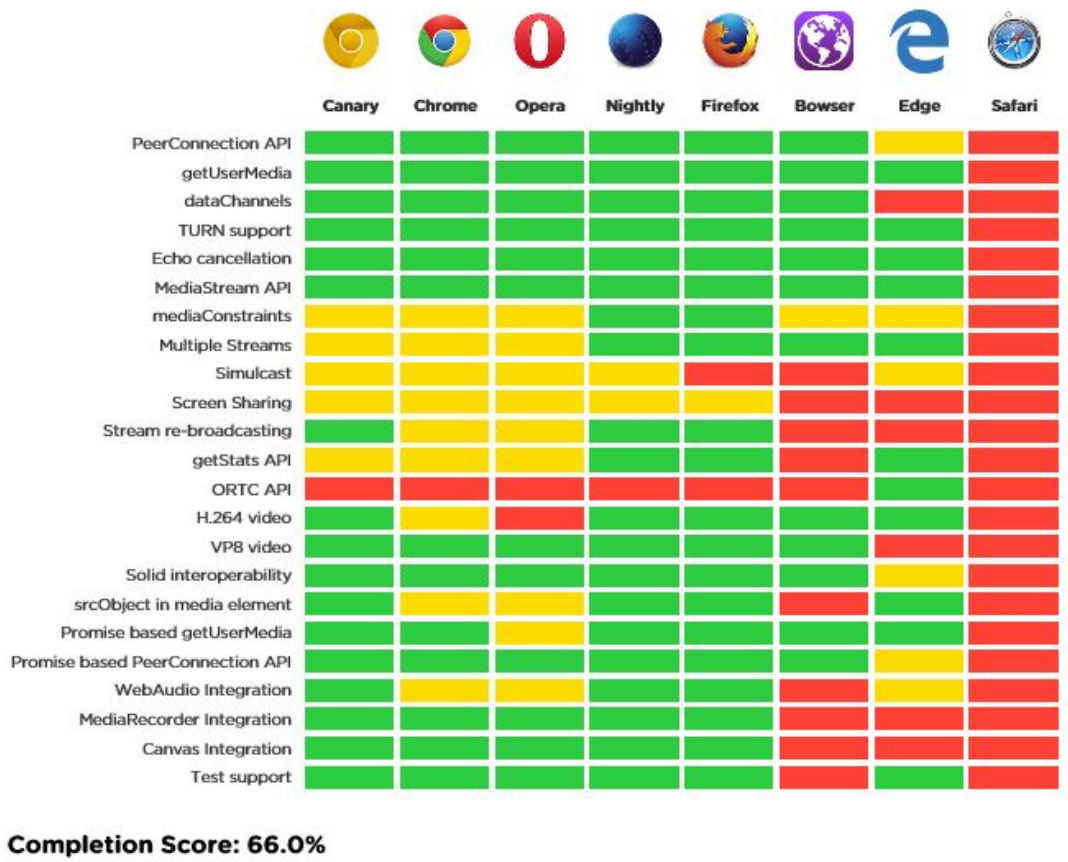
- Такое сообщение появилось в официальном блоге Mozilla 4 февраля 2013 года.
- Как можно понять, событие связано с первым в истории сеансом видеосвязи между браузерами Firefox и Chrome.


chrome

Когда WebRTC будет готово?



Когда WebRTC будет готово?



Microsoft + ORTC ???

- **Bringing Interoperable Real-Time Communications to the Web**
- Monday, October 27, 2014 9:35 AM
- Together with the industry-leading expertise of Skype, we're excited to announce development has begun on the ORTC API for WebRTC, a key technology to make Real-Time Communications (RTC) on the web a reality.

WebRTC for IE

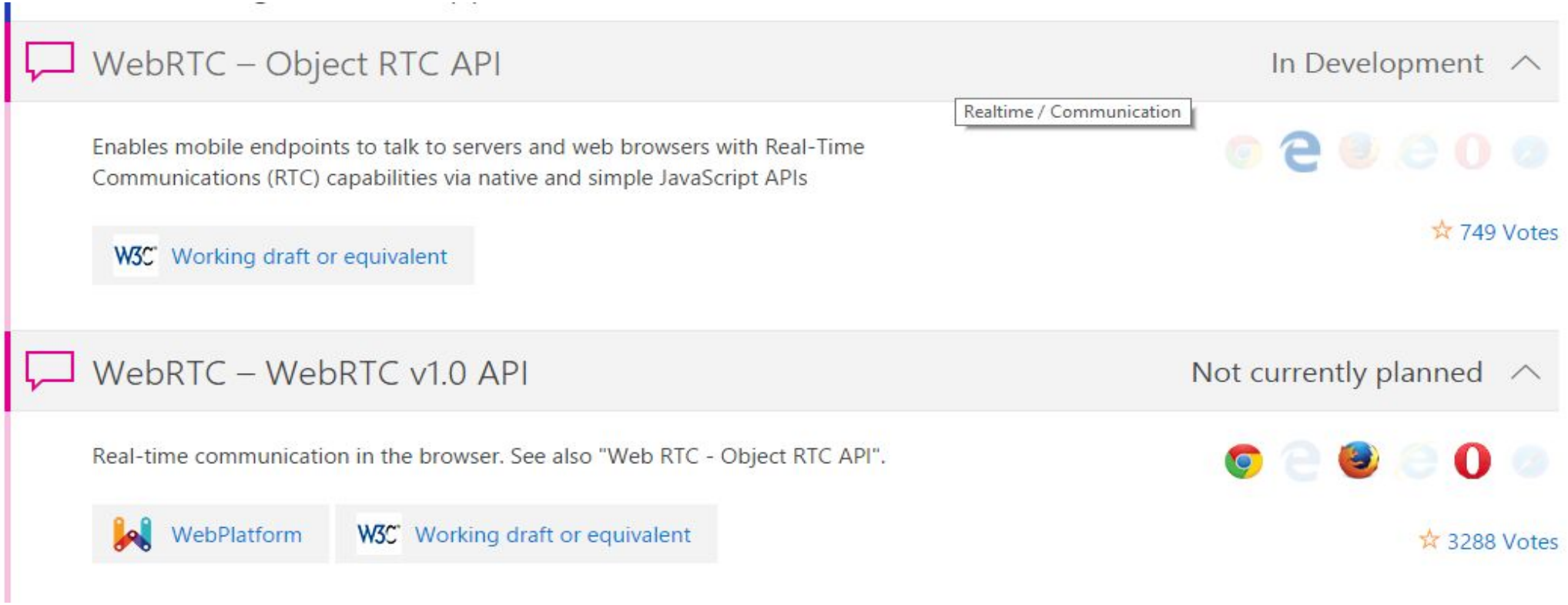
- Microsoft is sun-setting Internet Explorer with the introduction of Windows 10, replacing it with Edge, written from scratch. Edge already supports WebRTC's getUserMedia API, which is where every browser started with WebRTC. By year's end, I expect Edge to have sufficient support of WebRTC to make it interesting -- though Microsoft will most probably **stick with the H.264 codec** for now.
- In 2015, Microsoft won't be adding any WebRTC support to Internet Explorer. That may come later, or not at all.

Microsoft Edge Июль 2015

- Microsoft has been an outlier, but the release of Windows 10 on July 29 moves the company firmly into the WebRTC camp, with integrated support for WebRTC in its new browser.
- When Google released WebRTC in 2011, the project initially supported the Opus audio and VP8 video codecs. Microsoft and others wanted support for more codecs, with the H.264 video codec being the major point of contention. H.264 is an established standard built into video software and hardware solutions, but it is also licensed intellectual property that requires royalty payments.
- After much discussion within the Internet Engineering Task Force, H.264 was added as a requirement to WebRTC in March 2015. Cisco Systems has released both H.264 binaries and source code in a software library called OpenH264, opening a path for support of H.264 in WebRTC and other third-party applications. Mozilla has used the OpenH264 code to add H.264 support to WebRTC in Firefox.

WebRTC - ORTC Platform Status

- https://www.w3.org/community/ortc
- https://dev.modern.ie/platform/status/webrtcwebrtcv10api/



Safari?

- It is still unknown when this (GetUserMedia only) will find its way into Safari, and more specifically in Safari on iOS. Hopefully before the end of the year. (high, but probably unrealistic, hopes for a Sept. 9 announcement).

Кодеки (1 из 4)

- **Аудиокодеки**
- Для сжатия аудио-трафика в WebRTC используются кодеки Opus и G.711.
- **G.711** — самый старый голосовой кодек с высоким битрейтом (64 kbps), который чаще всего применяется в системах традиционной телефонии. Основным достоинством является минимальная вычислительная нагрузка из-за использования легких алгоритмов сжатия. Кодек отличается низким уровнем компрессии голосовых сигналов и не вносит дополнительной задержки звука во время общения между пользователями.

Кодеки (2 из 4)

- **Opus** — это кодек с низкой задержкой кодирования (от 2.5 мс до 60 мс), поддержкой переменного битрейта и высоким уровнем сжатия, что идеально подходит для передачи потокового аудиосигнала в сетях с переменной пропускной способностью. Opus — гибридное решение, сочетающее в себе лучшие характеристики кодеков SILK (компрессия голоса, устранение искажений человеческой речи) и CELT (кодирование аудиоданных). Кодек находится в свободном доступе, разработчикам, которые его используют, не нужно платить отчисления правообладателям. По сравнению с другими аудиокодеками, Opus, несомненно, выигрывает по множеству показателей. Он затмил довольно популярные кодеки с низким битрейтом, такие, как MP3, Vorbis, AAC LC. Opus восстанавливает наиболее приближенную к оригиналу “картину” звука, чем AMR-WB и Speex. За этим кодеком — будущее, именно поэтому создатели технологии WebRTC включили его в обязательный ряд поддерживаемых аудиостандартов.

Кодеки (3 из 4)

- **Видеокодеки**
- Вопросы выбора видеокодека для WebRTC заняли у разработчиков несколько лет, в итоге решили использовать H.264 и VP8. Практически все современные браузеры поддерживают оба кодека. Серверам видеоконференций для работы с WebRTC достаточно поддержать только один.

Кодеки (4 из 4)

- **VP8** — свободный видеокодек с открытой лицензией, отличается высокой скоростью декодирования видеопотока и повышенной устойчивостью к потере кадров. Кодек универсален, его легко внедрить в аппаратные платформы, поэтому очень часто разработчики систем видеоконференцсвязи используют его в своих продуктах.
- Платный видеокодек H.264 стал известен намного раньше своего собрата. Это кодек с высокой степенью сжатия видеопотока при сохранении высокого качества видео. Высокая распространенность этого кодека среди аппаратных систем видеоконференцсвязи предполагает его использование в стандарте WebRTC.
- Компания Google активно продвигает кодек VP8, а Firefox и Cisco — H.264, чтобы обеспечить совместимость с обычными системами видеоконференцсвязи.

Что есть сейчас?

Поддержка следующих API:

- MediaStream (aka getUserMedia) – позволяет получить доступ к потокам данных с камеры и микрофона (возможны и другие источники).
- RTCPeerConnection – передача аудио и видео с шифрованием и управлением пропускной способностью.
- RTCDataChannel – P2P обмен произвольными данными.

Моё первое WebRTC приложение

Приложение должно выполнить следующие действия:

- Получить потоковое видео, аудио или другие данные.
- Получить сетевую информацию (такую как IP адреса и порты) и обменяться этой информацией с другими WebRTC клиентами (peers)
- Обеспечить соединение даже при наличии NAT или сетевого экрана.
- Выполнить отправку сигналов, для уведомления об ошибках и создания/закрытия сессий.
- Выполнить обмен информацией о возможностях клиента (разрешение и поддерживаемые кодеки)
- Начать передавать потоковое видео, аудио или данные.

MediaStream (1 из 4)

- MediaStream API представляет доступ к синхронизированным между собой аудио и видео потокам.
- У каждого MediaStream есть вход сгенерированный с помощью navigator.getUserMedia()
- И выход который может быть передан в video элемент или в RTCPeerConnection.

MediaStream (2 из 4)

- Метод getUserMedia() получает 3 параметра:

navigator.getUserMedia(constraints, successCallback, errorCallback);

- Объект с ограничениями
- Функцию обратного вызова, которая получает MediaStream (на случай успеха)
- Функцию обратного вызова, которая получает информацию об ошибке (на случай неудачи)

MediaStream (3 из 4)

- У каждого MediaSteam есть метка (например 'Xk7EuLhsuHKbnjLWkW4yYGNJJ8ONsgwHBvLQ')
- Массив MediaStreamTracks который возвращается с помощью методов getAudioTracks() и getVideoTracks().
- Каждый MediaStreamTrack имеет тип (‘video’ или ‘audio’) и метку (что-то вроде ‘FaceTime HD Camera (Build-in)’), и представляет один или более каналов аудио или видео.
- Чаще всего будет только одна дорожка аудио и одна дорожка видео. Но легко представить случаи, когда их будет больше.

MediaStream (4 из 4)

- Кроме video элемента и RTCPeerConnection, getUserMedia() может служить входом для Web Audio API.

```
<script>
    function gotStream(stream) {
        window.AudioContext = window.AudioContext || window.webkitAudioContext;
        var audioContext = new AudioContext();

        // Create an AudioNode from the stream
        var mediaStreamSource = audioContext.createMediaStreamSource(stream);

        // Connect it to destination to hear yourself
        // or any other node for processing!
        mediaStreamSource.connect(audioContext.destination);
    }

    navigator.getUserMedia({ audio: true }, gotStream);

</script>
```

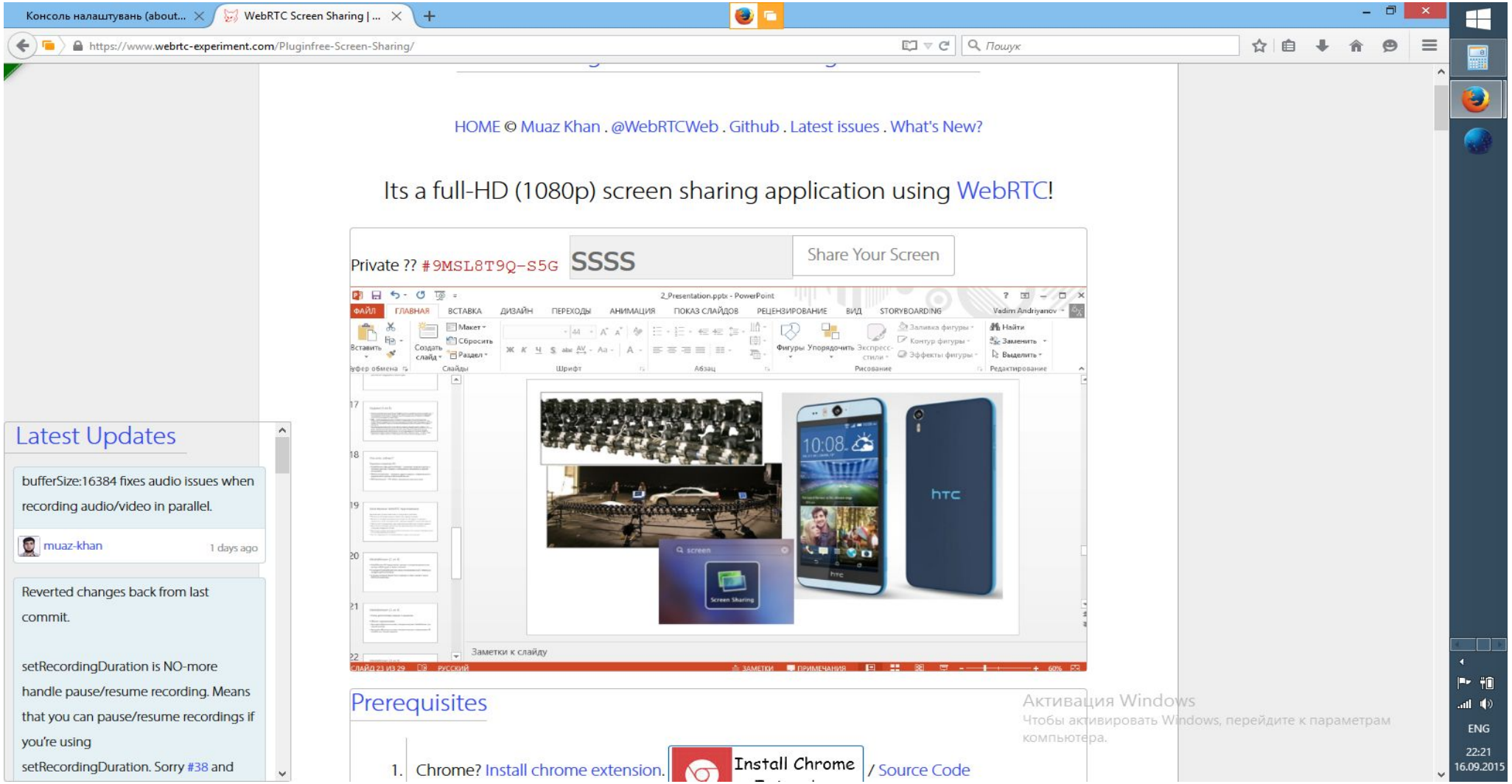
Ограничения

- Желаемая частота кадров 60 ширина 640 высота 480
- Требуемое соотношение ширины к высоте 4:3

```
getUserMedia({
    video: {
        mandatory: { minAspectRatio: 1.333, maxAspectRatio: 1.334 },
        optional [
          { minFrameRate: 60 },
          { maxWidth: 640 },
          { maxHeigth: 480 }
        ]
    }
}, successCallback, errorCallback);
```

FaceKat игра getUserMedia + headtrackr.js

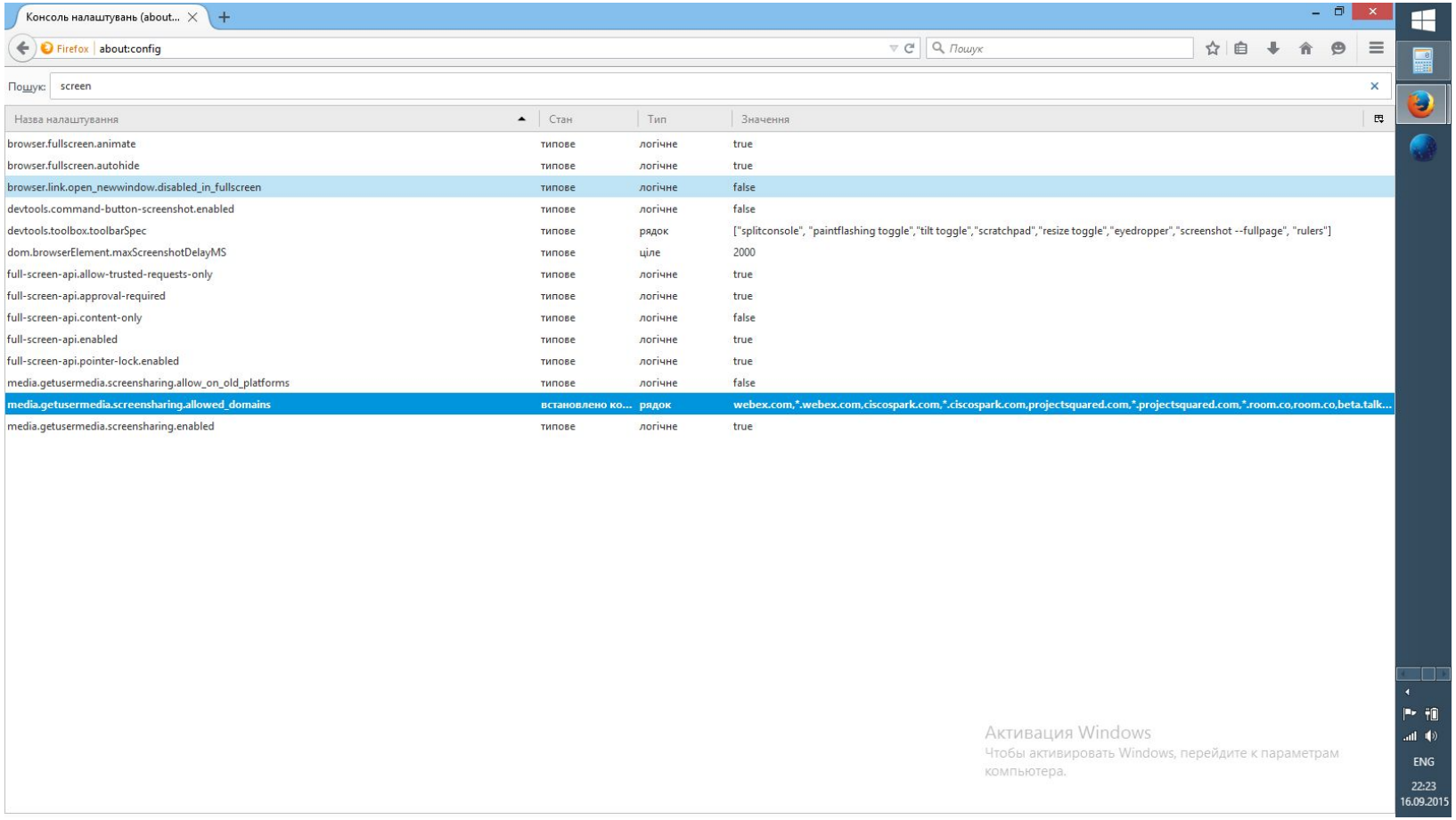
Screensharing (Firefox nightly)



Screensharing (Firefox nightly)

```
if(isChrome) {
  screen_constraints = {
    video: {
      mandatory: {
        chromeMediaSource: 'screen',
        maxWidth: screen.width,
        maxHeight: screen.height,
        minFrameRate: 1,
        maxFrameRate: 5
      },
      optional: []
  }};
} else {
  selector.disabled = false;
  screen_constraints = {
    video: {
      mediaSource: "screen"
    }
  };
}
```

Screensharing (Firefox nightly)



Сигналы (1 из 2)

- **Сигналы: управление сессиями, медиа и сетевая информация**
- WebRTC использует RTCPeerConnection чтобы передавать потоковые данные между браузерами.
- Кроме этого требуется механизм для передачи управляющих сообщений – сигналы.
- Методы и протоколы с помощью которых передаются сигналы не являются частью WebRTC и RTCConnection API.
- Например, для передачи сигналов, можно использовать Socket.io и Node server.

Сигналы (2 из 2)

- Сигналы используются для обмена тремя типами информации:
- Управляющие сообщения: для инициализации или закрытия сессии и уведомления об ошибках.
- Конфигурация: IP адрес и порт.
- Возможности: какие кодеки и разрешения поддерживаются моим браузером и браузером с которым устанавливается связь.

Соединение (1 из 3)

```
var signalingChannel = createSignalingChannel();
var pc;
var configuration = ...;

// run start(true) to initiate a call
function start(isCaller) {
    pc = new RTCPeerConnection(configuration);

    // send any ice candidates to the other peer
    pc.onicecandidate = function (evt) {
        signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));
    };

    // once remote stream arrives, show it in the remote video element
    pc.onaddstream = function (evt) {
        remoteView.src = URL.createObjectURL(evt.stream);
    };

    // get the local stream, show it in the local video element and send it
    navigator.getUserMedia({ "audio": true, "video": true }, function (stream)
        selfView.src = URL.createObjectURL(stream);
        pc.addStream(stream);

        if (isCaller)
            pc.createOffer(gotDescription);
        else
            pc.createAnswer(pc.remoteDescription, gotDescription);

        function gotDescription(desc) {
            pc.setLocalDescription(desc);
            signalingChannel.send(JSON.stringify({ "sdp": desc }));
        }
    });
}
```

```
signalingChannel.onmessage = function (evt) {
    if (!pc)
        start(false);

    var signal = JSON.parse(evt.data);
    if (signal.sdp)
        pc.setRemoteDescription(new RTCSessionDescription(signal.sdp));
    else
        pc.addIceCandidate(new RTCIceCandidate(signal.candidate));
};
```

1. Приложение №1 создает объект RTCPeerConnection
2. Когда найден доступный ‘сетевой кандидат’ вызывается обработчик onicecandidate
3. Отправка кандидата приложению №2
4. Когда приложение №2 получит кандидата будет вызван метод addIceCandidate

Пример ‘сетевого кандидата’

a=candidate:1853887674 1 udp 1845501695 46.2.2.2 36768 typ srflx raddr 192.168.0.197 rport 36768 generation 0

Соединение (2 из 3)

- Кроме того приложение №1 и приложение №2 должны обменяться информацией о конфигурации сессии
- 1. Приложение №1 вызывает метод createOffer(). В функцию обратного вызова передается объект RTCSessionDescription, который описывает локальную сессию приложения.
- 2. В функции обратного вызова, приложение №1 вызывает метод setLocalDescription(). После этого, описание сессии передается приложению №2. *RTCPeerConnection не начнет искать ‘кандидатов’ до вызова setLocalDescription()*
- *3. Приложение №2 получает RTCSessionDescription от приложения №1 и вызывает метод setRemoteDescription()*

Соединение (3 из 3)

- 4. Приложение №2 вызывает метод createAnswer() и передает туда описание сессии полученное от приложения №1. Так, приложение №2 создает сессию совместимую с приложением №1.
- 5. Описание сессии отправляется обратно приложению №1
- 6. Приложение №1 получает описание и вызывает метод setRemoteDescription()
- 7. Связь установлена.

Session Description Protocol

- Session description
- v= (protocol version number, currently only 0)
- o= (originator and session identifier : username, id, version number, network address)
- s= (session name : mandatory with at least one UTF-8-encoded character)
- i=* (session title or short information)
- u=* (URI of description)
- e=* (zero or more email address with optional name of contacts)
- p=* (zero or more phone number with optional name of contacts)
- c=* (connection information—not required if included in all media)
- b=* (zero or more bandwidth information lines)
- One or more Time descriptions ("t=" and "r=" lines; see below)
- z=* (time zone adjustments)
- k=* (encryption key)
- a=* (zero or more session attribute lines)
- Zero or more Media descriptions (each one starting by an "m=" line; see below)

Session Description Protocol

- Time description (mandatory)
- t= (time the session is active)
- r=* (zero or more repeat times)
- Media description (if present)
- m= (media name and transport address)
- i=* (media title or information field)
- c=* (connection information — optional if included at session level)
- b=* (zero or more bandwidth information lines)
- k=* (encryption key)
- a=* (zero or more media attribute lines — overriding the Session attribute lines)

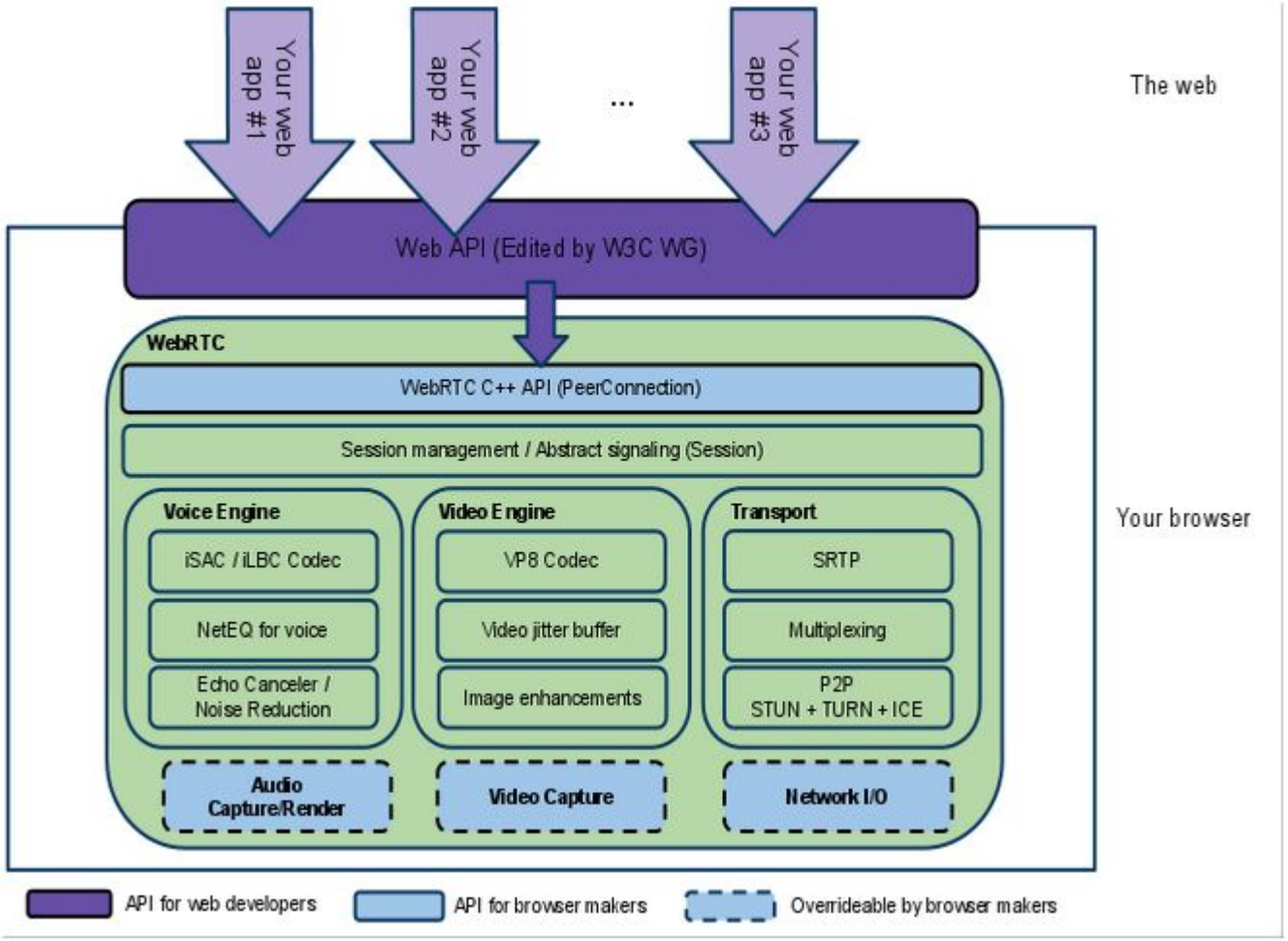
```
v=0
 o=alice 2890844526 2890844526 IN IP4 host.anywhere.com
 s=
 c=IN IP4 host.anywhere.com
 t=0 0
 m=audio 49170 RTP/AVP 0
 a=rtpmap:0 PCMU/8000
 m=video 51372 RTP/AVP 31
 a=rtpmap:31 H261/90000
 m=video 53000 RTP/AVP 32
 a=rtpmap:32 MPV/90000
```

Session Description Protocol

```
(... snip ...)
m=audio 1 RTP/SAVPF 111 ... 1
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=candidate:1862263974 1 udp 2113937151 192.168.1.73 60834 typ host ... 2
a=mid:audio
a=rtpmap:111 opus/48000/2 3
a=fmtp:111 minptime=10
(... snip ...)
```

1. Secure audio profile with feedback
2. Candidate IP, port, and protocol for the media stream
3. Opus codec and basic configuration

RTCPeerConnection



RTCPeerConnection

- Кодеки и протоколы используемые WebRTC делают большой объем работы для того, чтобы сделать коммуникацию в реальном времени возможной даже в ненадежных сетях:
- Сокрытие потери пакетов
- Подавление эха
- Адаптация под пропускную способность
- dynamic jitter buffering
- автоматическая регулировка усиления аудио
- Устранение шума
- Очистка картинки

RTCPeerConnection без сервера (1 из 3)

- Создаем RTCPeerConnection
- Получем поток и добавляем его в RTCPeerConnection

```
// servers is an optional config file (see TURN and STUN discussion below)
pc1 = new webkitRTCPeerConnection(servers);
// ...
pc1.addStream(localstream);
```

RTCPeerConnection без сервера (2 из 3)

- Создаем описание сессии.
- Вызываем setLocalDescription, setRemoteDescription
- Создаем описание совместимой сессии с помощью метода createAnswer()

```
pc1.createOffer(gotDescription1);
//...
function gotDescription1(desc){
    pc1.setLocalDescription(desc);
    trace("Offer from pc1 \n" + desc.sdp);
    pc2.setRemoteDescription(desc);
    pc2.createAnswer(gotDescription2);
}
```

RTCPeerConnection без сервера (3 из 3)

- Создаем RTCPeerConnection принимающей стороны
- Показываем «удаленный» поток когда он будет получен.

```
pc2 = new webkitRTCPeerConnection(servers);
pc2.onaddstream = gotRemoteStream;
//...
function gotRemoteStream(e){
    vid2.src = URL.createObjectURL(e.stream);
}
```

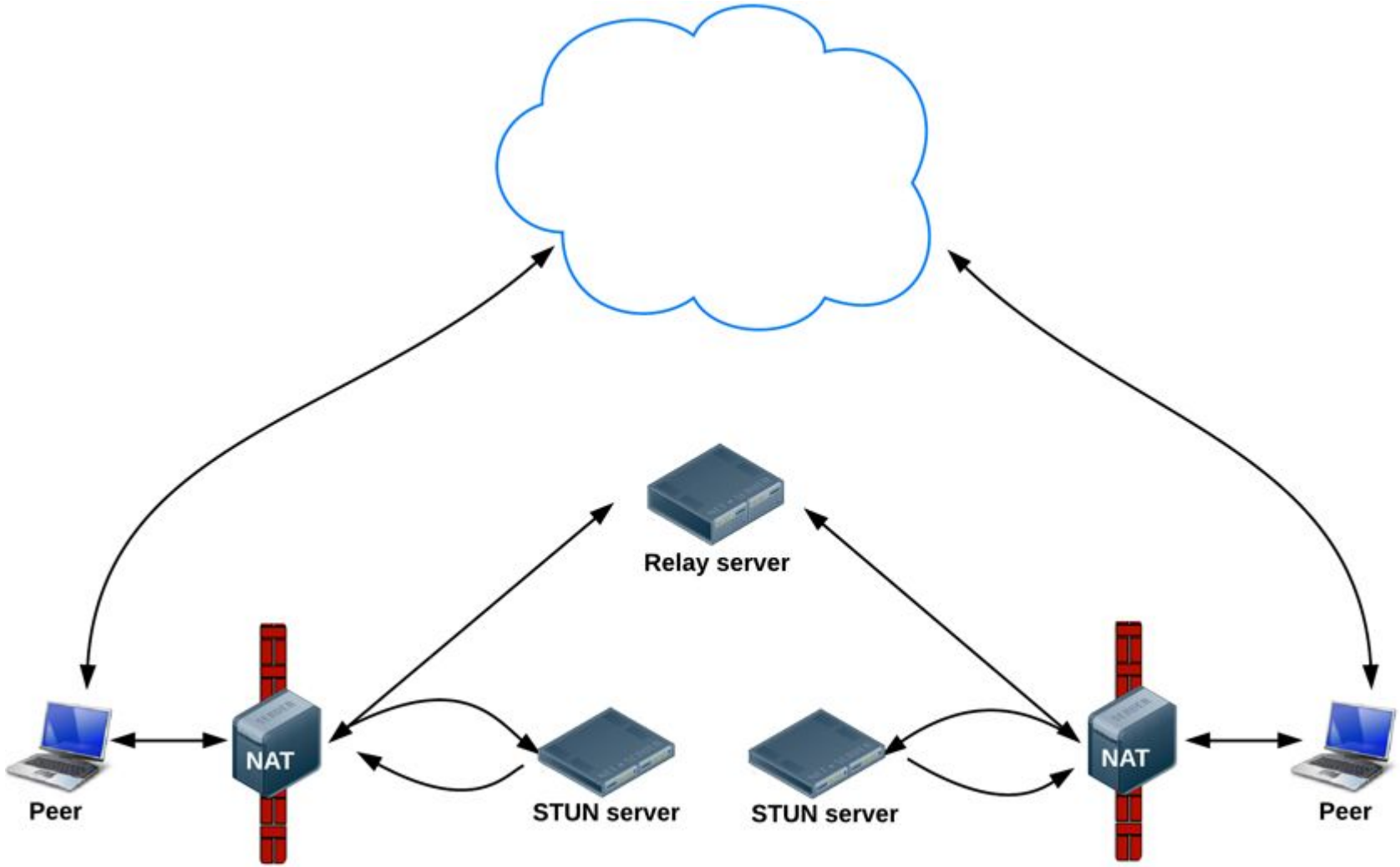
RTCPeerConnection + сервер

В реальном мире для нужен сервер:

- Пользователи находят друг друга и обмениваются информацией о себе (например именами)
- Приложения обмениваются сетевой информацией.
- Приложения обмениваются описанием сессий
- Приложения обходят NAT и сетевые экраны.

Обход NAT (1 из 2)

- Для того чтобы RTCPeerConnection мог обходить NAT, ICE Framework использует протоколы STUN и TURN.
- **STUN** (сокр. от англ *Session Traversal Utilities for NAT*, Утилиты прохождения сессий для NAT, ранее англ. *Simple Traversal of UDP through NATs*, Простое прохождение UDP через серверы NAT) — это сетевой протокол, который позволяет клиенту, находящемуся за сервером трансляции адресов (или за несколькими такими серверами), определить свой внешний IP-адрес способ трансляции адреса и порта во внешней сети, связанный с определённым внутренним номером порта

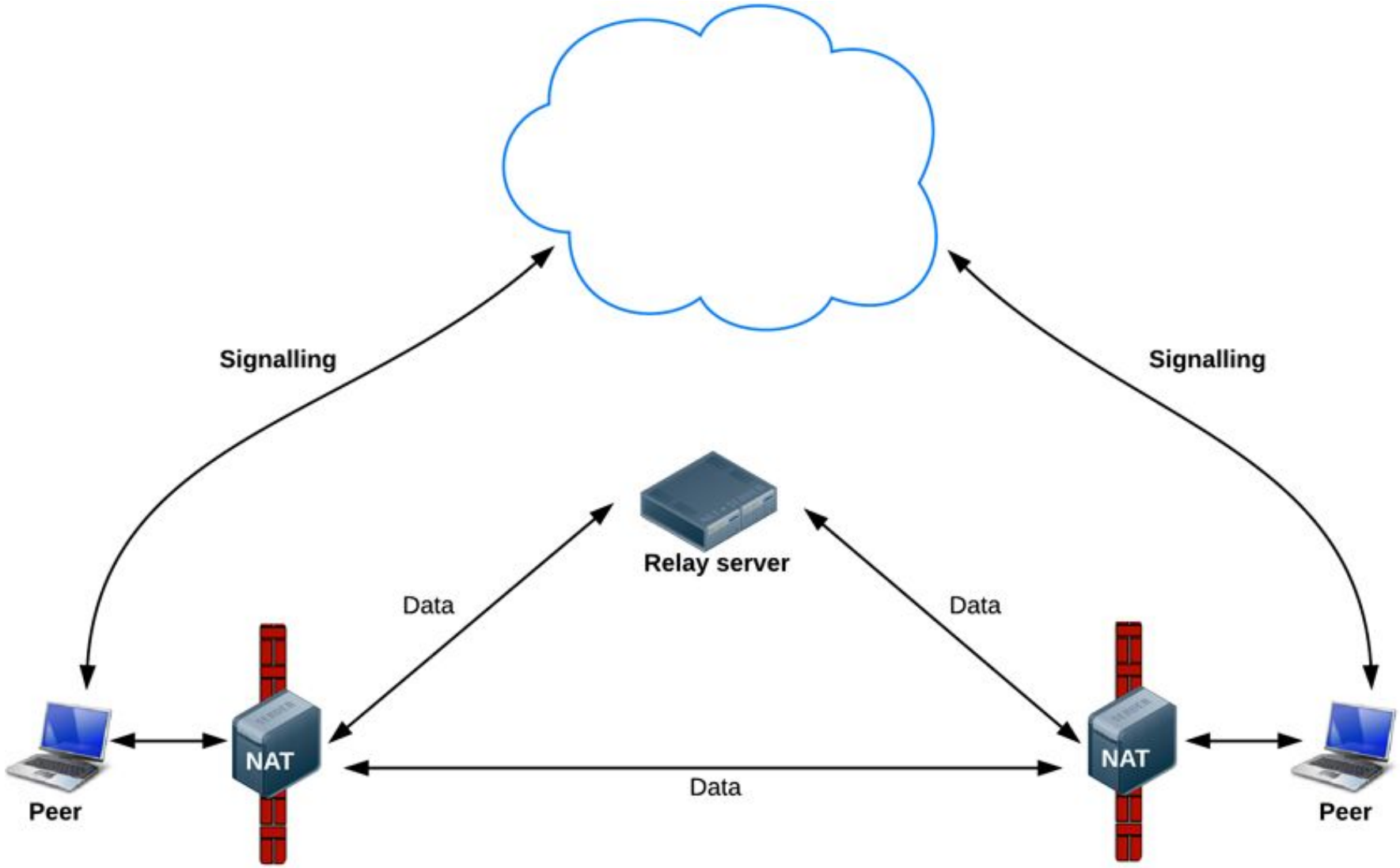

Relay server
NAT
NAT
Peer
STUN server
STUN server
Peer

Обход NAT (2 из 2)

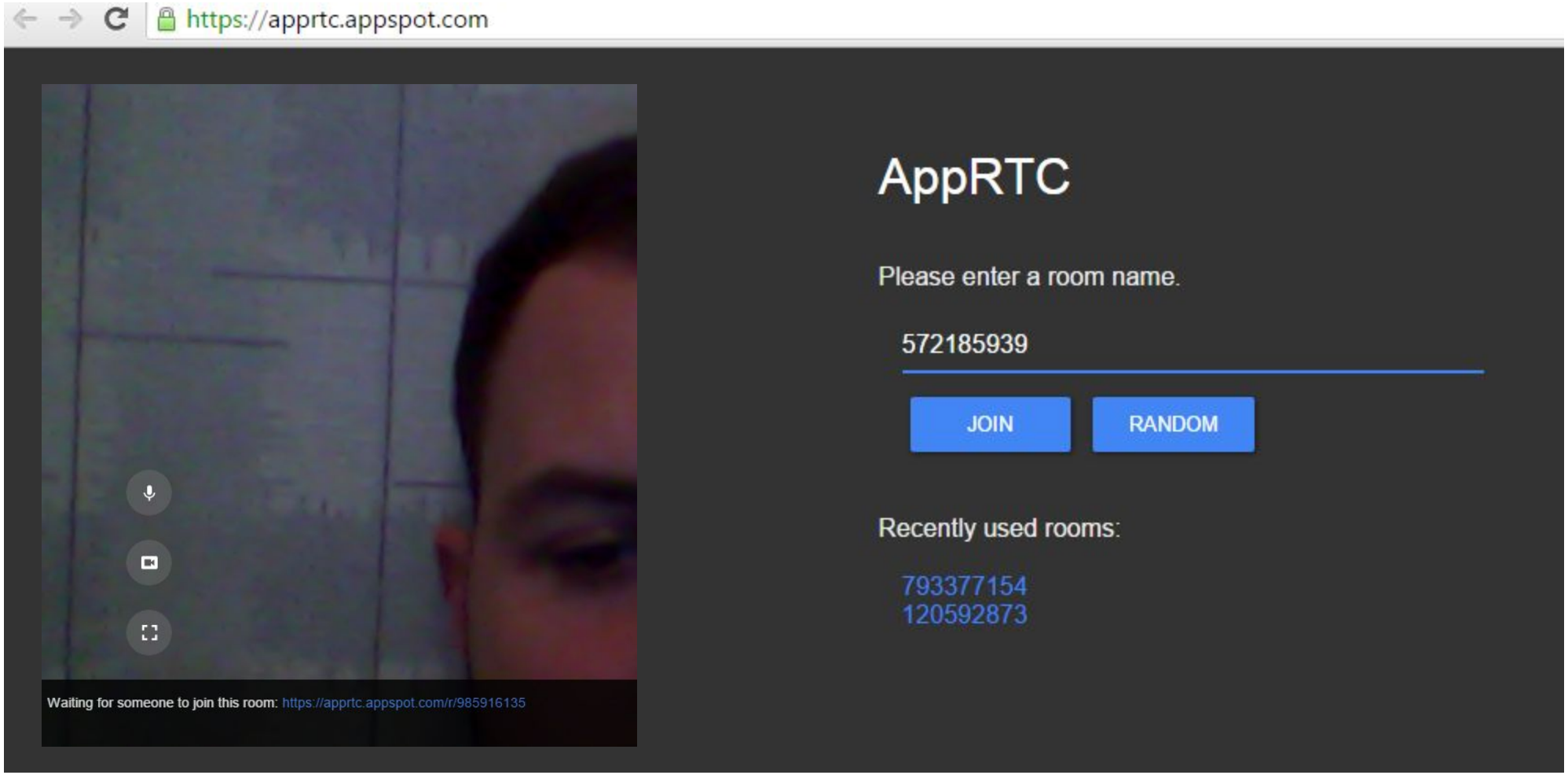
- Session Traversal Utilities for NAT (STUN) предусматривает одно средство для прохождения NAT. STUN позволяет клиенту получить транспортный адрес (IP адрес и порт), который может быть полезен для приема пакетов от peer-ов. Однако адреса, полученные через STUN, не могут быть доступны всем peer-ам. Эти адреса работают в зависимости от топологии сети. Таким образом, STUN сам по себе не может обеспечить комплексное решение для обхода NAT.
- **Симметричный NAT** (Symmetric NAT) — Трансляция, при которой каждое соединение, инициируемое парой «внутренний адрес: внутренний порт» преобразуется в свободную уникальную случайно выбранную пару «публичный адрес: публичный порт». **При этом инициация соединения из публичной сети невозможна.**
- Законченное решение требует средств, с помощью которых клиент мог бы получить транспортный адрес, на который он мог бы получать поток данных от любого peer-а который может передавать пакеты данных в публичный интернет. Это может быть достигнуто лишь путем ретрансляции данных через сервер, который находится в общедоступном Интернете. Эта спецификация описывает Traversal Using Relay NAT (TURN), протокол, который позволяет клиенту получить IP-адреса и порты от таких peer-ов.

Протокол ICE

- IP адрес с самым высоким приоритетом предпочтения будет использоваться для ведения общения между устройствами. В списке маршрутов наивысший приоритет получают маршруты без задействования STUN, более низкий – маршруты с задействованием STUN, и наиболее низкий – маршруты с проксированием медиатрафика через TURN-сервер.
- ICE выполняет всю грязную работу по преодолению различных NAT устройств. Теперь нет необходимости дополнительно настраивать ваши роутеры и маршрутизаторы, для работы с VoIP телефонией.


Signalling
Signalling
Relay server
Data
Data
NAT
NAT
Data
Peer
Peer

apprtc.appspot.com demo



apprtc.appspot.com demo

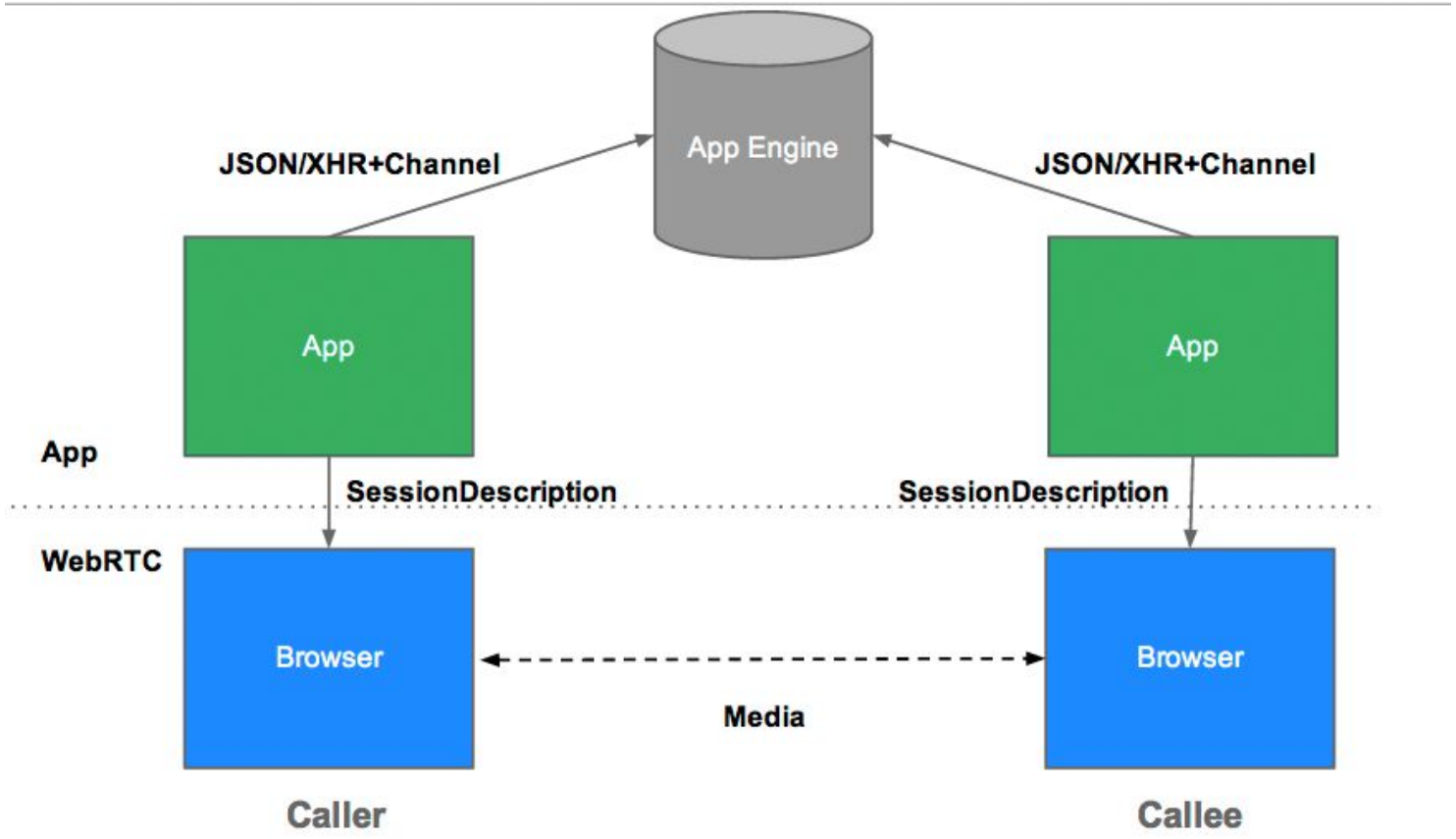
Инициализация приложения

```
function initialize() {
    console.log("Initializing; room=99688636.");
    card = document.getElementById("card");
    localVideo = document.getElementById("localVideo");
    miniVideo = document.getElementById("miniVideo");
    remoteVideo = document.getElementById("remoteVideo");
    resetStatus();
    openChannel('AHRlWrqvgCpvbd9B-Gl5vZ2F1BlpwFv0xBUwRgLF/* ...*/');
    doGetUserMedia();
}
```

Для отправки сигналов используется Google App Engine

```
function openChannel(channelToken) {
    console.log("Opening channel.");
    var channel = new goog.appengine.Channel(channelToken);
    var handler = {
        'onopen': onChannelOpened,
        'onmessage': onChannelMessage,
        'onerror': onChannelError,
        'onclose': onChannelClosed
    };
    socket = channel.open(handler);
}
```

apprtc.appspot.com demo



apprtc.appspot.com demo

- После открытия сигнального канала вызывается getUserMedia()
- Если этот вызов успешен, то вызывается функция обратного вызова onUserMediaSuccess()
- Поток привязывается к элементу localVideo
- Переменная initiator равна 1, поэтому происходит вызов функции maybeStart()

```
function onUserMediaSuccess(stream) {
    console.log("User has granted access to local media.");
    // Call the polyfill wrapper to attach the media stream to this element.
    attachMediaStream(localVideo, stream);
    localVideo.style.opacity = 1;
    localStream = stream;
    // Caller creates PeerConnection.
    if (initiator) maybeStart();
}
```

apprtc.appspot.com demo

- Эта функция вызывается в нескольких асинхронных функциях обратного вызова. Код этой функции выполнится только тогда, когда переменная localStream будет ссылаться на локальный медиа поток, а переменная channelReady будет равна true. Таким образом соединение будет установлено не более одного раза

```
function maybeStart() {
    if (!started && localStream && channelReady) {
        // ...
        createPeerConnection();
        // ...
        pc.addStream(localStream);
        started = true;
        // Caller initiates offer to peer.
        if (initiator)
            doCall();
    }
}
```

apprtc.appspot.com demo

- Основное назначение этой функции в установлении соединения с использованием STUN сервера.
- Назначение события onicecandidate описано выше на слайдах «Соединение»
- К RTCPeerConnection подключаются обработчики событий. Все они, кроме onRemoteStreamAdded, просто выполняют логирование

```
function createPeerConnection() {
    var pc_config = {"iceServers": [{"url": "stun:stun.l.google.com:19302"}]};
    try {
        // Create an RTCPeerConnection via the polyfill (adapter.js).
        pc = new RTCPeerConnection(pc_config);
        pc.onicecandidate = onIceCandidate;
        console.log("Created RTCPeerConnnection with config:\n" + "  \"" +
          JSON.stringify(pc_config) + "\".");
    } catch (e) {
        console.log("Failed to create PeerConnection, exception: " + e.message);
        alert("Cannot create RTCPeerConnection object; WebRTC is not supported by this browser.");
        return;
    }

    pc.onconnecting = onSessionConnecting;
    pc.onopen = onSessionOpened;
    pc.onaddstream = onRemoteStreamAdded;
    pc.onremovestream = onRemoteStreamRemoved;
}
```

apprtc.appspot.com demo

- Этот метод устанавливает удаленный поток с элементом removeVideo

```
function onRemoteStreamAdded(event) {
    // ...
    miniVideo.src = localVideo.src;
    attachMediaStream(remoteVideo, event.stream);
    remoteStream = event.stream;
    waitForRemoteVideo();
}
```

apprtc.appspot.com demo

- После создания соединения, создается описание сессии
- Описание сессии отправляется по сигнальному каналу вызываемому приложению

```
function doCall() {
    console.log("Sending offer to peer.");
    pc.createOffer(setLocalAndSendMessage, null, mediaConstraints);
}

function setLocalAndSendMessage(sessionDescription) {
    // Set Opus as the preferred codec in SDP if Opus is present.
    sessionDescription.sdp = preferOpus(sessionDescription.sdp);
    pc.setLocalDescription(sessionDescription);
    sendMessage(sessionDescription);
}
```

apprtc.appspot.com demo

- Получение и отправка ‘кандидата’

```
function onIceCandidate(event) {
    if (event.candidate) {
        sendMessage({type: 'candidate',
            label: event.candidate.sdpMLineIndex,
            id: event.candidate.sdpMid,
            candidate: event.candidate.candidate});
    } else {
        console.log("End of candidates.");
    }
}

function sendMessage(message) {
    var msgString = JSON.stringify(message);
    console.log('C->S: ' + msgString);
    path = '/message?r=99688636' + '&u=92246248';
    var xhr = new XMLHttpRequest();
    xhr.open('POST', path, true);
    xhr.send(msgString);
}
```

apprtc.appspot.com demo

- Обработчик сигнальных сообщений

```
function processSignalingMessage(message) {
    var msg = JSON.parse(message);

    if (msg.type === 'offer') {
        // Callee creates PeerConnection
        if (!initiator && !started)
            maybeStart();

        pc.setRemoteDescription(new RTCSessionDescription(msg));
        doAnswer();
    } else if (msg.type === 'answer' && started) {
        pc.setRemoteDescription(new RTCSessionDescription(msg));
    } else if (msg.type === 'candidate' && started) {
        var candidate = new RTCIceCandidate({sdpMLineIndex:msg.label,
            candidate:msg.candidate});
        pc.addIceCandidate(candidate);
    } else if (msg.type === 'bye' && started) {
        onRemoteHangup();
    }
}
```

RTCDataChannel (1 из 2)

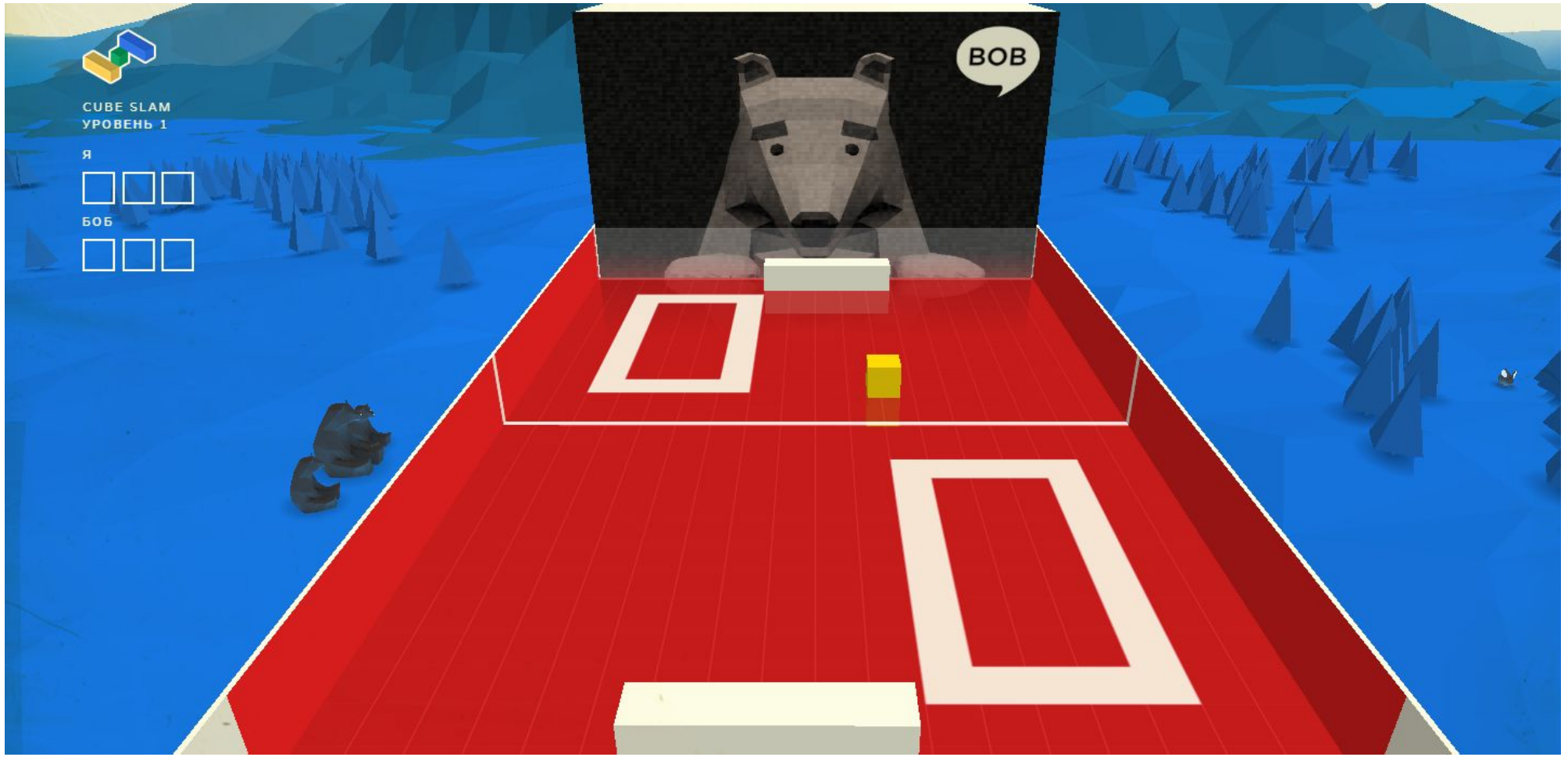
- Кроме аудио и видео WebRTC поддерживает коммуникацию для других типов данных.
- Существует много способов использования данного API:
- Игры
- Управление удаленным рабочим столом
- Текстовый чат
- Передача файлов
- Распределенные сети

RTCDataChannel (2 из 2)

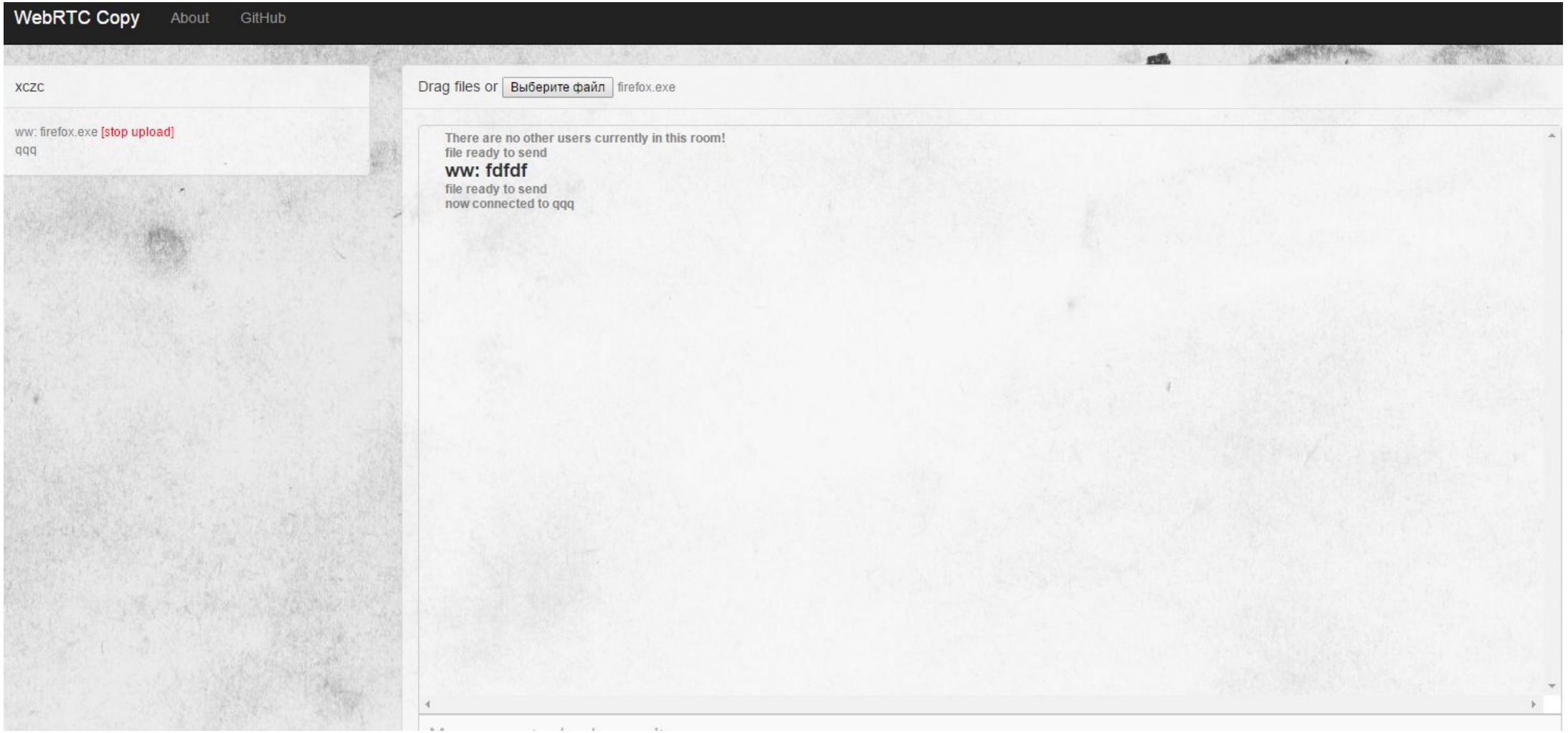
- Связь осуществляется напрямую между браузерами, поэтому RTCDataChannel может работать намного быстрее чем веб-сокеты (даже при наличии STUN серверов)

```
var pc = new webkitRTCPeerConnection(servers,
{optional: [{RtpDataChannels: true}]});

pc.ondatachannel = function(event) {
    receiveChannel = event.channel;
    receiveChannel.onmessage = function(event){
        document.querySelector("div#receive").innerHTML = event.data;
    };
};

sendChannel = pc.createDataChannel("sendDataChannel", {reliable: false});

document.querySelector("button#send").onclick = function (){
    var data = document.querySelector("textarea#send").value;
    sendChannel.send(data);
};
```

CubeSlam



Передача файлов https://rtccopy.com



Диагностика chrome://webrtc-internals

Список литературы

- http://www.webrtc.org/
- http://www.html5rocks.com/en/tutorials/webrtc/basics/
- http://blog.trueconf.ru/reviews/webrtc.html
- http://voipnotes.ru/nat-potocol-turn-rsip-ice/
- https://ru.wikipedia.org/wiki/Traversal_Using_Relay_NAT
- https://www.webrtc-experiment.com/docs/STUN-or-TURN.html
- Сухов К. HTML5 – путеводитель по технологии. – М.: ДМК Пресс, 2013. – 352 с.