УДК 004.93 **Информационные технологии**

Никишанин Роман Олегович, *магистр, Национальный исследовательский Мордовский государственный университет им. Н.П. Огарева*

Ямашкин Станислав Анатольевич, *кандидат технических наук, доцент кафедры автоматизированных систем обработки информации и управления, Национальный исследовательский Мордовский государственный университет им. Н. П. Огарева*

ПЕРЕДАЧА ПОТОКОВЫХ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ WEBRTC

Аннотация: В статье описан способ передачи видео и аудио медиа-потоков в сети Интернет с помощью технологии WebRTC.

Ключевые слова: медиа-поток, WebRTC, JavaScript.

Abstract: The article describes a method for transmitting video and audio media streams on the Internet using WebRTC technology.

Keywords: media-stream, WebRTC, JavaScript.

Введение

В настоящее время нередко возникает ситуация, когда в качестве рабочей задачи требуется создать функционал для веб-приложения, который позволял бы пользователям данного сервиса непосредственного взаимодействовать между собой. Обычно для этих целей создаются видеочаты на примере Skype, Zoom и других популярных платформ. В данном случае требуется решать сразу несколько технических задач. Во-первых, необходимо, чтобы устанавливаемое между пользователями соединение и передача медиа-потока были быстрыми и своевременными, иначе говоря, не было проблем рассинхронизации, потери данных и т.д. Кроме того, требуется обеспечить минимальную нагрузку на сервер, где развернуто веб-приложение, при установлении нового соединения,

ведь таких соединений может одновременно существовать достаточно много в реальных случаях.

На сегодняшний день один из наиболее удачных выборов для реализации передачи медиа-потока между несколькими браузерами является технология WebRTC (Web Real-Time Communications). Это решение является стандартом W3C. Кроме того, оно поддерживается в таких браузерах, как Google Chrome (и других, использующих тот же движок), Mozilla и Opera. Данная технология основана на открытом стандарте, что позволяет отказаться от загрузки дополнительных программ, надстроек и расширений в браузер, нужно просто написать код на JavaScript с использованием специального веб API.

В данной статье будет рассмотрен процесс создания простой передачи медиа-потоков с помощью WebRTC в реальном приложении.

1. Краткое описание принципов работы WebRTC

Чтобы установить соединение между пользователями в сети Интернет WebRTC используется технология Peer-to-Peer. С ее помощью между браузерами устанавливается прямое соединение без использования веб-сервера. В результате происходит прямая передача видеопотоков с одного устройства на другое [2].

Рассмотрим работу технологии.

Сначала Браузер запрашивает разрешение на доступ к веб-камере и микрофону. В браузере, которые инициирует соединение, формируется SDP-пакет. Это текстовый файл, содержащий всю необходимую информацию о параметрах соединения (какой медиа-контент будет передаваться (звук, видео, данные), какие кодеки будут использоваться и т. д.) [3].

Далее, в зависимости от реализации технологии на клиенте, инициатор соединения передаёт этот пакет другим участникам. Обычно для этого используется веб-сервер и WebSocket протокол [3].

После этого на принимающей сторона принимает SDP-пакет, а затем генерирует подобный полученному, которые затем отправляется обратно [3].

В зависимости от реализации, параллельно с предыдущими шагами

выполняется проверка подключения к сети. Передаётся адрес STUN-сервера, который используется, чтобы узнать внешний IP-адрес устройства. Он сравнивается с внутренним IP-адресом для того, чтобы определить, используется ли NAT в данном подключении и, если да, то как маршрутизируются UDP-пакеты [3].

Если все шаги пройдены успешно, то соединение устанавливается. После этого периодически вызывается специальное событие onicecandidate, которое передаёт информацию об IP-адресах, настройках NAT и другую информацию[3].

Далее с помощью специальных движков в браузере происходит сбор мультимедийных данных с веб-камеры и/или микрофона, после чего эта информация передается в зашифрованном виде с помощью протокола SRTP [4].

2. Использование WebRTC в реальном приложении

Сначала необходимо получить доступ к мультимедийным устройствам, которые будут использоваться в качестве источника передаваемых данных (веб-камера и микрофон). К ним можно обратиться с помощью JavaScript через объект navigator.mediaDevices. Из этого объекта мы можем получить все подключенные устройства, фиксировать их изменения (когда устройство подключено или отключено) и открыть устройство для получения медиапотока [1].

Далее воспользуемся функцией getUserMedia(), которая возвращает MediaStream для соответствующих мультимедийных устройств. Эта функция принимает один объект MediaStreamConstraints, который описывает наши требования (использование видео или аудио или обоих источников одновременно). Пример получения доступа к медиа устройствам показан ниже:

```
const constraints = { audio: true, video: true }
this.client.stream = await navigator.mediaDevices.getUserMedia(constraints)
this.localMediumTarget.srcObject = this.client.stream
```

Вызов getUserMedia() инициирует запрос разрешений. Если пользователь принимает разрешение, обещание разрешается с помощью MediaStream, содержащего одну видео и одну аудиодорожку. Если разрешение отклонено,

будет возвращена ошибка PermissionDeniedError. Если нет подключенных подходящих устройств, будет возвращена ошибка NotFoundError.

Далее необходимо обеспечить установку Peer-To-Peer соединения между двумя браузерами. Для этого в WebRTC существует интерфейс RTCPeerConnection. Он определяет, как устанавливается одноранговое соединение, и должен содержать информацию об используемых серверах ICE (Internet Connectivity Establishment). Затем вызывается createOffer() для создания объекта RTCSessionDescription. Это описание сеанса устанавливается как локальное описание с помощью setLocalDescription() и затем отправляется на принимающую сторону. Как это выглядит в реальном приложении представлено ниже:

```
const peerConnectionConfig = {
  iceServers: [
    {
      urls: [
        'stun:stun.l.google.com:19302',
        'stun:global.stun.twilio.com:3478'
      ]
    }
  ],
  sdpSemantics: 'unified-plan'
}

async createOffer () {
    if (!this.readyToMakeOffer) return

    try {
      this.makingOffer = true
      await this.setLocalDescription(await this.peerConnection.createOffer())
    } catch (error) {
      console.error(error)
    } finally {
      this.makingOffer = false
    }
  }
```

```
async setLocalDescription (description) {
    await this.peerConnection.setLocalDescription(description)
}

setupPeerConnection () {
    this.peerConnection = new RTCPeerConnection(peerConnectionConfig)
}
```

На принимающей стороне происходит ожидание входящего предложения, прежде чем создавать свой экземпляр RTCPeerConnection. Как только это будет сделано, устанавливается описание удаленной сессии, используя setRemoteDescription(). Далее вызывается createAnswer() для создания ответа. Этот ответ устанавливается как локальное описание с помощью setLocalDescription() и затем отправляется вызывающей стороне. Реализация имеет следующий вид:

```
async setDescription (description) {
    try {
      if (this.ignore(description)) return

      await this.setRemoteDescription(description)

      if (description.type === 'offer') {
        await this.setLocalDescription(await this.peerConnection.createAnswer())
      }
    } catch (error) {
      if (this.retryCount < RETRY_LIMIT) {
        this.initiateManualRollback()
        this.retryCount++
      } else {
        this.stop()
        console.error(`Negotiation failed after ${this.retryCount} retries`)
      }
    }
  }

async setRemoteDescription (description) {
```

```
    this.isSettingRemoteAnswerPending = description.type === 'answer'
    await this.peerConnection.setRemoteDescription(description) // SRD rolls back as needed
    this.addCandidates()
    this.isSettingRemoteAnswerPending = false
  }
```

Завершающим этапом для установления соединения является сбор кандидатов для установления соединения и отправка медиа-потоков другой стороне. Получить потоки можно с помощью метода getTracks() у объекта MediaStream, затем их необходимо передавать в метод addTrack() объекта RTCPeerConnection. Для этого к объекту RTCPeerConnection добавляются прослушиватели специальных событий icegatheringstatechange и icecandidate, negotiationneeded и track:

```
broadcast (name, data) {
    (this.callbacks[name] || []).forEach(
      callback => callback.call(null, { type: name, detail: data })
    )
}

otherClient.on('track', ({ detail }) => {
      this.startStreamingFrom(otherClient.id, detail)
})

startStreamingTo (otherClient) {
    this.client.streamTo(otherClient)
}

startStreamingTo (otherClient) {
    this.client.streamTo(otherClient)
}

streamTo (otherClient) {
    if (!otherClient.streaming) {
```

```
      this.stream.getTracks().forEach(track => {
        otherClient.peerConnection.addTrack(track, this.stream)
      })
      otherClient.streaming = true
    }
}

this._onnegotiationneeded = () => this.createOffer()

    this._onicecandidate = ({ candidate }) => {
      this.signaller.signal({
        type: candidate ? 'candidate' : undefined,
        to: this.otherClient.id,
        from: this.client.id,
        candidate
      })
    }

    this._oniceconnectionstatechange = (event) => {
      this.client.broadcast(`iceConnection:${this.peerConnection.iceConnectionState}`,
{ otherClient: this.otherClient })

      switch (this.peerConnection.iceConnectionState) {
      case 'completed':
        this.retryCount = 0
        break;
      case 'failed':
        if ('restartIce' in this.peerConnection) this.peerConnection.restartIce()
        break;
      default:
        return
      }
    }

    this._ontrack = (event) => this.otherClient.broadcast('track', event)

    this.peerConnection.addEventListener('negotiationneeded',
this._onnegotiationneeded)
    this.peerConnection.addEventListener('icecandidate', this._onicecandidate)
    this.peerConnection.addEventListener('iceconnectionstatechange',
this._oniceconnectionstatechange)
```

```
    this.peerConnection.addEventListener('track', this._ontrack)
```

После выполнения все перечисленных действий соединение между двумя браузерами будет успешно установлено.

Выводы

Таким образом, в данной статье был рассмотрен видео и аудио медиа-потоков в сети Интернет с помощью технологии WebRTC, а также описано ее использование на примере реального приложения. WebRTC является перспективным и эффективным инструментом для передачи медиа-контента между браузерами посредством специального JavaScript API, которое поддерживается большинством современных браузеров, который не нагружает сервер веб-приложения. Однако у него есть и свои недостатки. WebRTC требует собственной реализации на стороне клиента, что не позволяет устанавливать соединение между двумя разными приложения, которые используют WebRTC. Тем не менее, такие ситуации встречаются не так часто на практике, в остальных же случаях WebRTC является достойной альтернативой конкурентным технологиям в данной области.

Библиографический список:

1. Документация WebRTC [Электронный ресурс]: webrtc.org – Режим доступа: https://webrtc.org/ (дата обращения: 28.05.2023).

2. Немного о WebRTC: что где использовать и случай из практики [Электронный ресурс]: habr.com – Режим доступа: https://habr.com/ru/articles/501416/ (дата обращения: 28.05.2023).

3. Всё о WebRTC [Электронный ресурс]: blog.trueconf.ru – Режим доступа: http://blog.trueconf.ru/reviews/webrtc.html (дата обращения: 28.05.2023).

4. Технология веб-коммуникации в режиме реального времени WebRTC [Электронный ресурс]: cyberleninka.ru – Режим доступа: https://cyberleninka.ru/article/n/tehnologiya-veb-kommunikatsii-v-rezhime-realnogo-vremeni-webrtc (дата обращения: 28.05.2023).