Организация одноранговых сетей с помощью WebRTC

Кулаков Кирилл Александрович

Проблема коммуникации

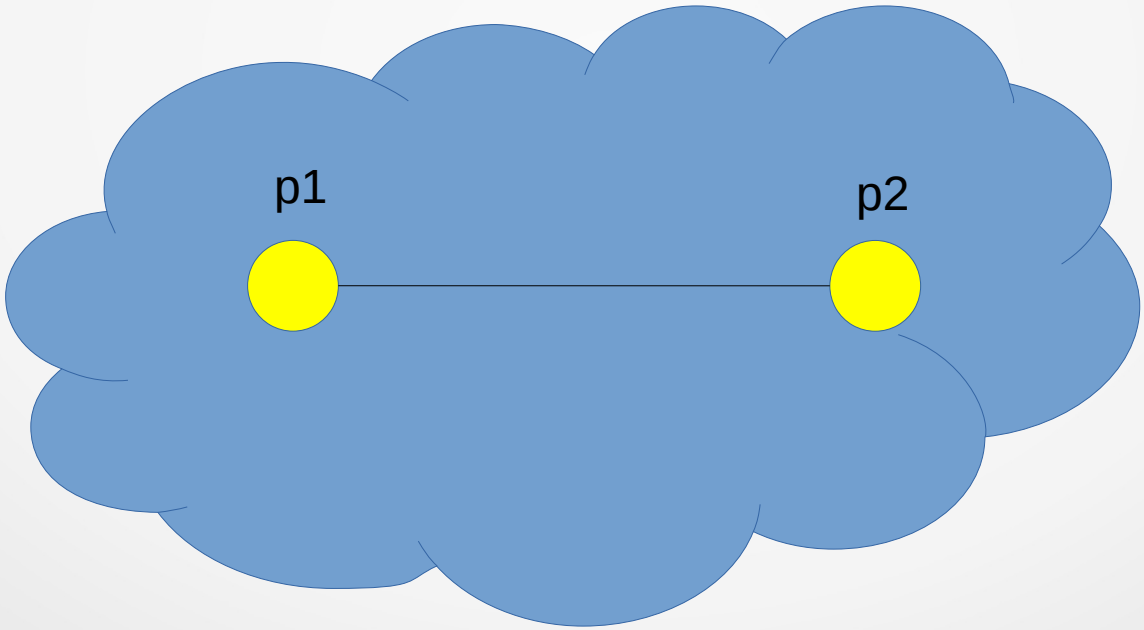
- Клиент-серверная технология: все передается через сервер
- Ниша веб приложений: взаимодействие пользователей
  - Чаты
  - Конференции
  - Трансляции
- Задачи сервера:
  - Организация коммуникации
  - Передача данных

WebRTC

- Технология связи в режиме реального времени в Интернете
- открытый веб-стандарт передачи аудио, видео, текстовых данных и файлов
- доступность в виде обычных API-интерфейсов JavaScript во всех основных браузерах
  - проведение конференции в браузере значительно упрощает процесс проведения конференции — пользователю не нужно устанавливать для этого отдельные приложения;
  - используемые кодеки обеспечивают хорошее качество связи;
  - возможность реализации любых элементов интерфейса средствами HTML5 и JavaScript;
  - открытый исходный код даёт больше возможностей для использования.

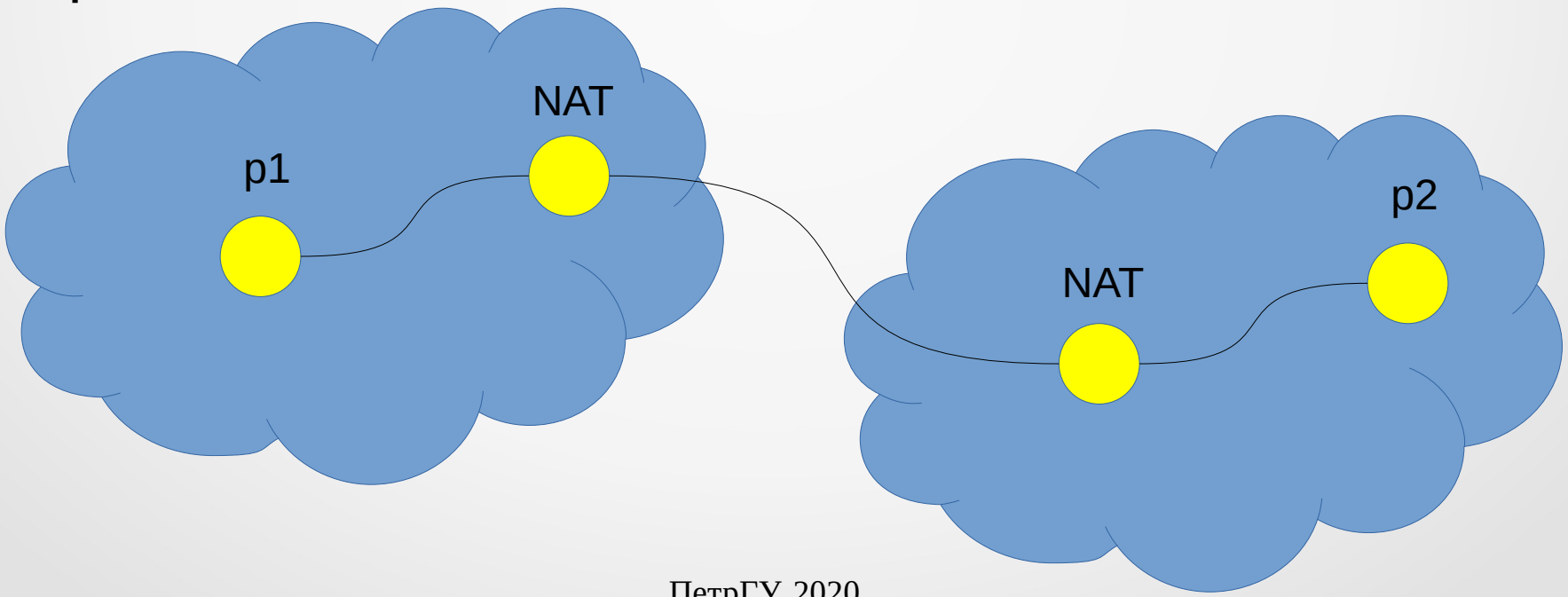
Peer-to-peer

- Технология соединения клиентов напрямую без использования сервера
- Прямое соединение: клиенты из одной сети



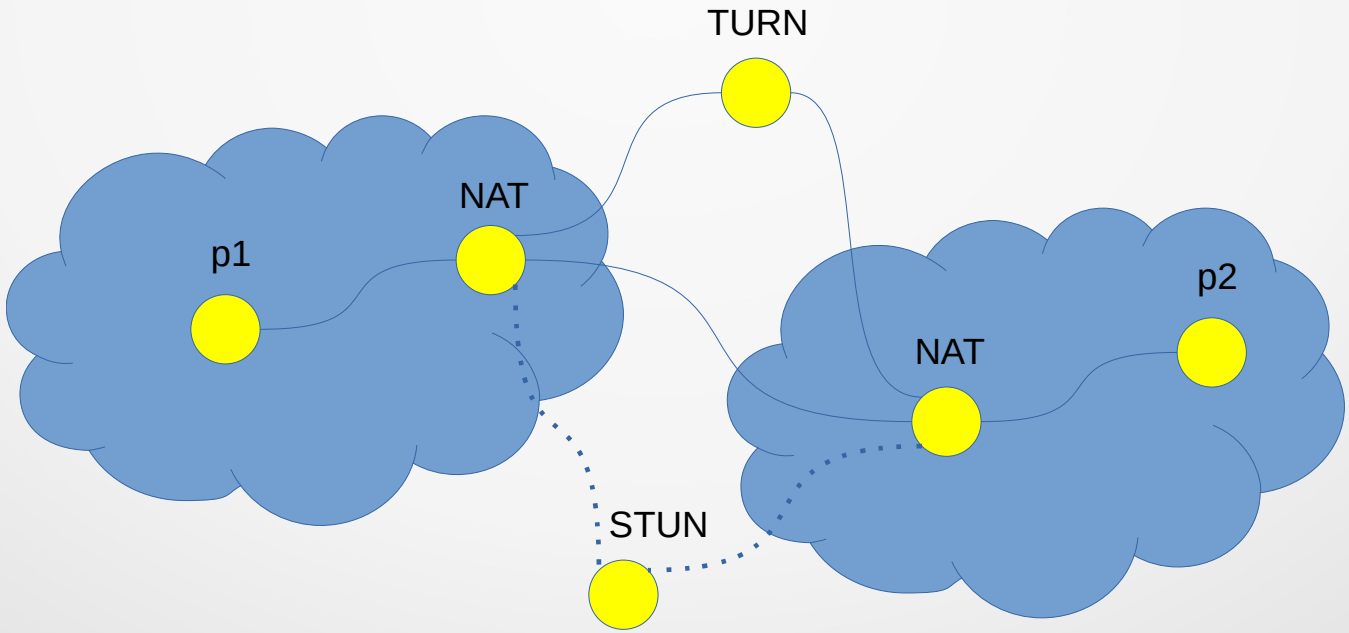
Peer-to-peer

- Более сложный вариант: клиенты находятся в разных сетях
- Использование технологии NAT — сервер перенаправления пакетов во внутреннюю сеть и обратно



Peer-to-peer

- Решение: использование STUN/TURN серверов
  - STUN сервер – это просто сервер в интернете, который возвращает обратный адрес, то есть адрес узла отправителя.
  - TURN сервер – это улучшенный STUN сервер. Если STUN не работает, то переключается в режим ретранслятора



Подключение

- Для работы протокола необходимо установление подключения (инициализация)
- Использование любого протокола обмена данными
- Дескриптор сессии (SDP) — организация логической коммуникации
  - Дескриптор сессии используется для логического соединения двух узлов сети
  - Дескриптор сессии хранит информацию о доступных способах кодирования видео и аудио данных
- Кандидаты (Ice candidate) — организация физической коммуникации
  - Адрес узла может быть своим, а может быть адресом роутера или TURN сервера
  - Кандидат состоит из IP адреса, порта и типа транспорта (TCP или UDP)

Передача данных

- Медиа поток
  - Видео и аудио данные упаковываются в медиа потоки
  - Медиа потоки синхронизируют медиа дорожки, из которых состоят
  - Различные медиа потоки не синхронизированы между собой
  - Медиа потоки могут быть локальными и удаленными, к локальному обычно подключена камера и микрофон, удаленные получают данные из сети в кодированном виде
  - Медиа дорожки бывают двух типов – для видео и для аудио
  - Медиа дорожки имеют возможность включения/выключения
  - Медиа дорожки состоят из медиа каналов
  - Медиа дорожки синхронизируют медиа каналы, из которых состоят
  - Медиа потоки и медиа дорожки имеют метки, по которым их можно различать

API: GetUserMedia()

- GetUserMedia(): получение доступа к локальным медиа ресурсам (аудио, видео)
- Синтаксис: *navigator.mediaDevices.getUserMedia(constraints, successCallback, errorCallback);*
- Пример: *const stream = await navigator.mediaDevices.getUserMedia({audio: true, video: true});*
- Документация: **https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia**

API: RTCPeerConnection()

- точка входа в WebRTC API, которая позволяет установить и инициализировать процесс соединения
- после его создания, добавляется информация о медиа-потоках
- предоставляет методы для соединения с удалённым участником соединения, обслуживания, мониторинга и закрытия соединения
- Документация: **https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection**

API: RTCDataChannel()

- Интерфейс для передачи данных между участниками
- Каждый канал данных связан с RTCPeerConnection
- Создание канала выполняется через метод соединения createDataChannel()
- Документация: **https://developer.mozilla.org/en-US/docs/Web/API/RTCDataChannel**

Пример

- Обмен данными через сервис ScaleDrone https://www.scaledrone.com/

```
const drone = new ScaleDrone('yiS12Ts5RdNhebyM');

// создаем комнату с префиксом 'observable-'

const roomName = 'observable-' + chatHash;

// обработка подключения к сигнальному сервису

drone.on('open', error => {
  if (error) {
    return console.error(error);
  }
  room = drone.subscribe(roomName);
  room.on('open', error => {
    if (error) {
      return console.error(error);
    }
    console.log('Connected to signaling server');
  });
  // получаем список участников чата
  room.on('members', members => {
    if (members.length >= 3) {
      return alert('The room is full');
    }
    // запуск чата если подключилось 2 пользователя
    const isOfferer = members.length === 2;
    startWebRTC(isOfferer);
  });
});
```

Пример

- Запуск протокола WebRTC

```
const configuration = {
  iceServers: [{
    url: 'stun:stun.l.google.com:19302'
  }]
};
function startWebRTC(isOfferer) {
  console.log('Starting WebRTC in as', isOfferer ? 'offerer' : 'waiter');
  pc = new RTCPeerConnection(configuration);

  // передача параметров подключения через сигнальный сервер
  pc.onicecandidate = event => {
    if (event.candidate) {
      sendSignalingMessage({'candidate': event.candidate});
    }
  };
  if (isOfferer) {
    // инициатор чата создает чат
    pc.onnegotiationneeded = () => {
      pc.createOffer(localDescCreated, error => console.error(error));
    }
    dataChannel = pc.createDataChannel('chat');
    setupDataChannel();
  } else {
    // второй пользователь подключается к чату
    pc.ondatachannel = event => {
      dataChannel = event.channel;
      setupDataChannel();
    }
  }
  startListentingToSignals();
}
// Отправка сообщения через сигнальный сервер Scaledrone
function sendSignalingMessage(message) {
  drone.publish({
    room: roomName,
    message
  });
}
```

Пример

- Подключение собеседника к чату

```
function startListentingToSignals() {
  room.on('data', (message, client) => {
    if (client.id === drone.clientId) { // игнорируем наши сообщения
      return;
    }
    if (message.sdp) {
      // если мы получили настройки, то добавляем собеседника
      pc.setRemoteDescription(new
RTCSessionDescription(message.sdp), () => {
        console.log('pc.remoteDescription.type', pc.remoteDescription.type);
        // отвечаем инициатору сообщения
        if (pc.remoteDescription.type === 'offer') {
          console.log('Answering offer');
          pc.createAnswer(localDescCreated, error => console.error(error));
        }
      }, error => console.error(error));
    } else if (message.candidate) {
      // Добавляем собеседника к списку подключений
      pc.addIceCandidate(new RTCIceCandidate(message.candidate));
    }
  });
}
```

- Отправка сообщения

```
form.addEventListener('submit', () => {
  const input =
document.querySelector('input[type="text"]');
  const value = input.value;
  input.value = '';

  const data = {
    name,
    content: value,
  };

  dataChannel.send(JSON.stringify(data));

  insertMessageToDOM(data);
});
```

Пример

- Получение сообщения

```
function setupDataChannel() {
  dataChannel.onmessage = event =>
    insertMessageToDOM(JSON.parse(event.data))
}
function insertMessageToDOM(options) {
  const template = document.querySelector('template[data-template="message"]');
  const nameEl = template.content.querySelector('.message__name');
  if (options.name) {
    nameEl.innerText = options.name;
    nameEl.style.color = "red";
  }
  template.content.querySelector('.message__bubble').innerText = options.content;
  const clone = document.importNode(template.content, true);
  const messageEl = clone.querySelector('.message');
  const messagesEl = document.querySelector('.messages');
  messagesEl.appendChild(clone);
}
```