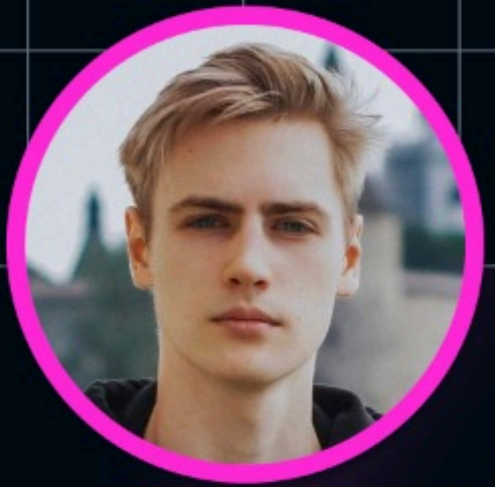
HOLY JS/...

Создание массовых звонков и трансляций с помощью WebRTC



Константин Сарвилов

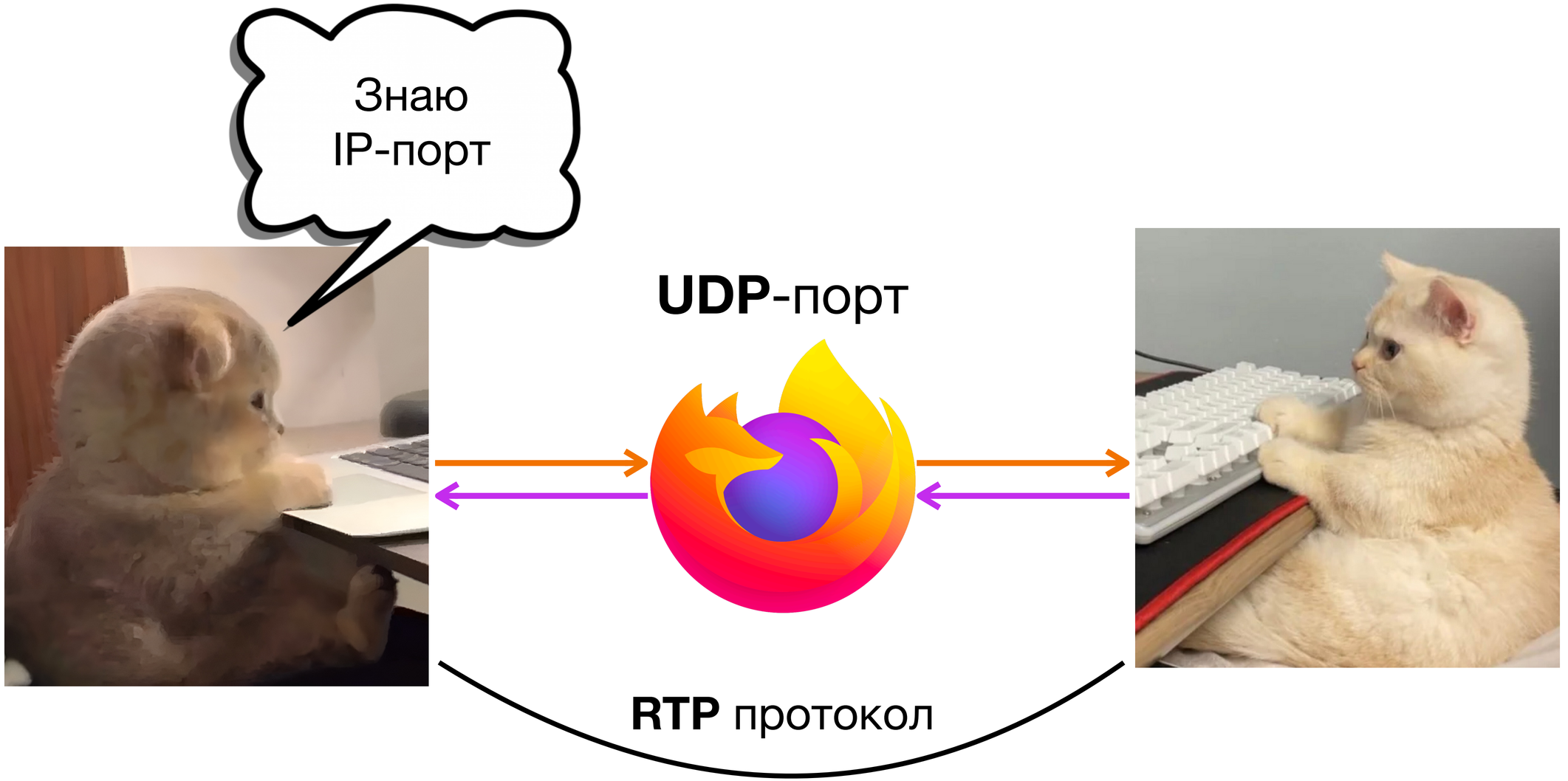
Совкомбанк Технологии

W3C — разработка стандартов для веба

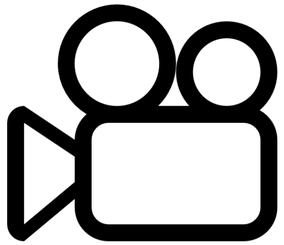
IETF — разработка стандартов на уровне транспорта

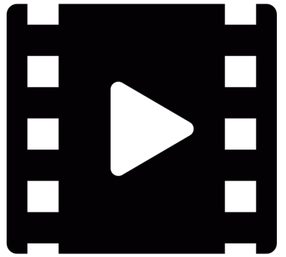

НАСТАЛО ВРЕМЯ
УДИВИТЕЛЬНЫХ
ИСТОРИЙ

В чем суть?


Знаю
IP-порт
UDP-порт
RTP протокол

Простой пример





H.264

VP8

AV1

Codec

UDP

TCP, UDP

...

А разница то в чем?

TCP

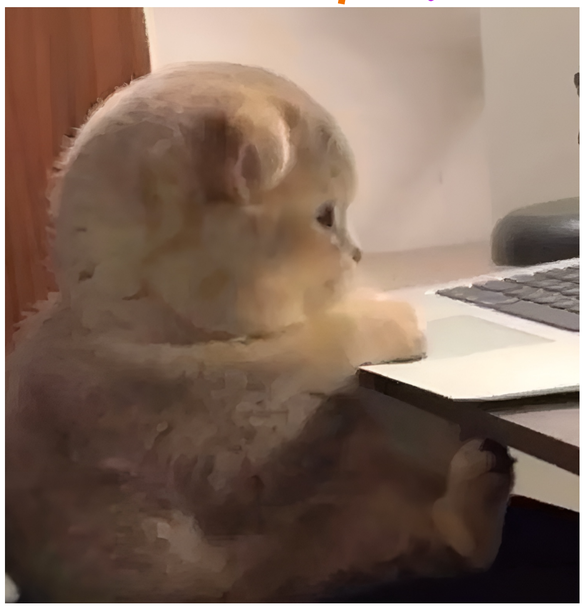
Stateful,
гарантии

Кидаюсь
пакетами
(быстро)

UDP

Как соединить двух участников?

Signaling Server

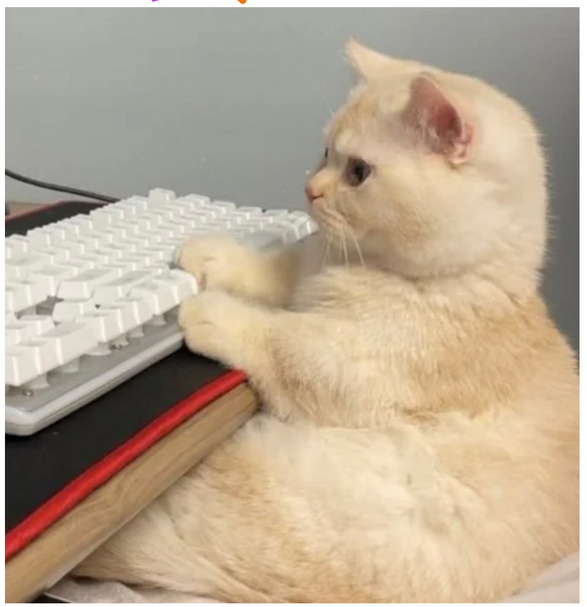


new RTCPeerConnection(…)
createOffer()

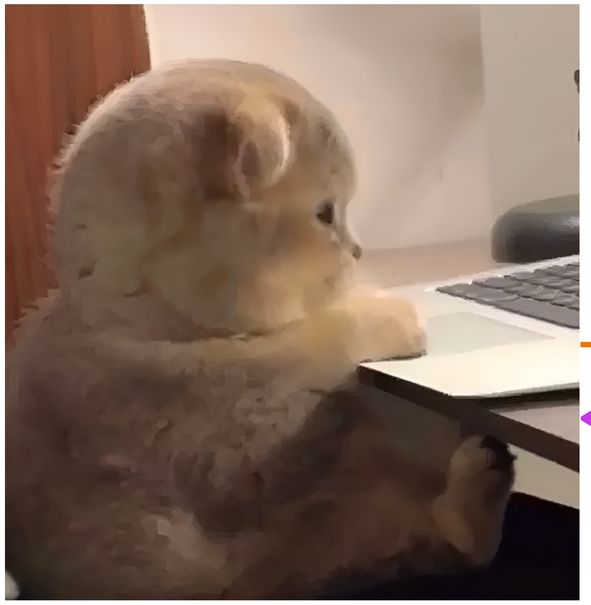
Соединение установлено!

…

Или нет?



new RTCPeerConnection(…)
createAnswer()



new RTCPeerConnection(…)
createOffer()

Signaling Server

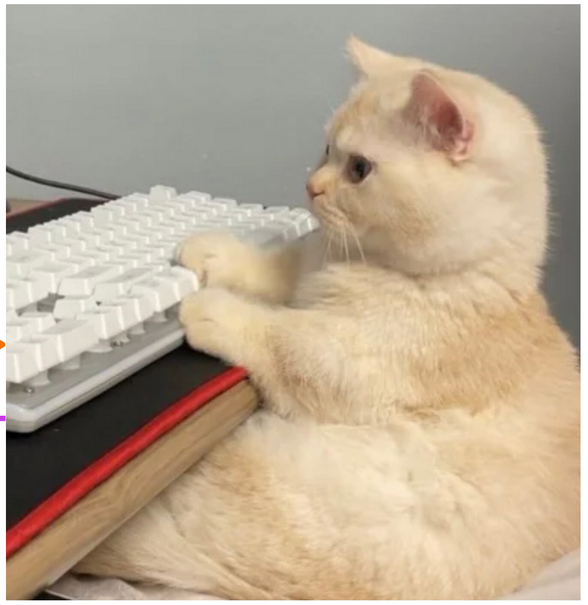
TURN Server

NAT

Кто я?

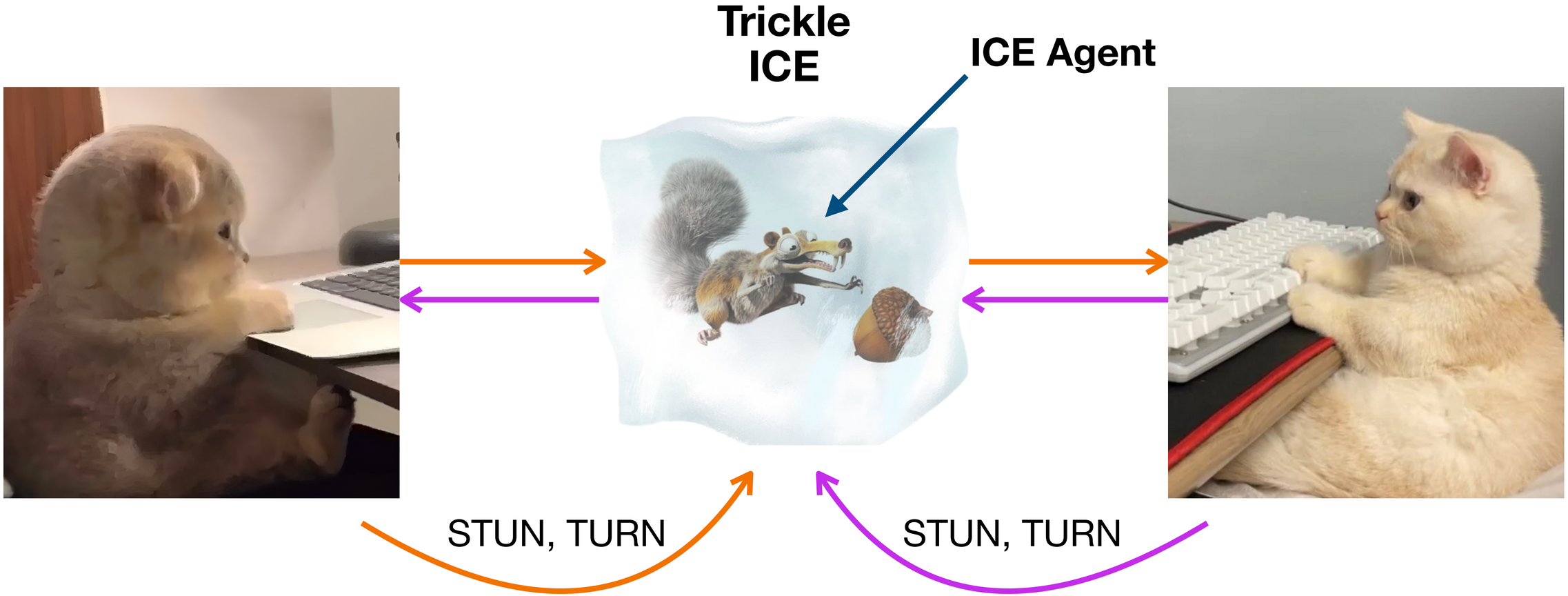
Ты котик

STUN Server



new RTCPeerConnection(…)
createAnswer()

Сложно, непонятно


Trickle
ICE
ICE Agent
STUN, TURN
STUN, TURN

Безопасность и шифрование

DTLS

Медиа

SRTP — данные

SRTCP — управление сессией

Бинарники

SCTP

Кодирование, кодеки

IETF — стандарт

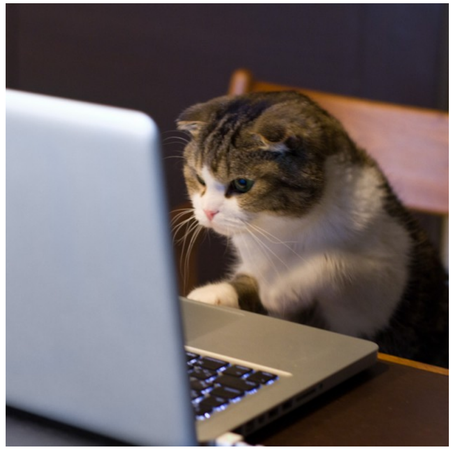
H.264/AVC

VP8

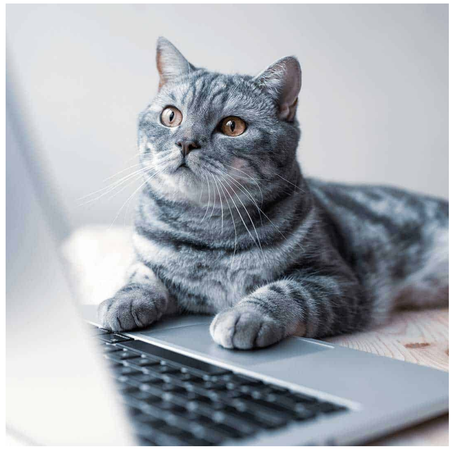
AV1

Simulcast — посылаем несколько видео-потоков разного качества

Создание массовых звонков

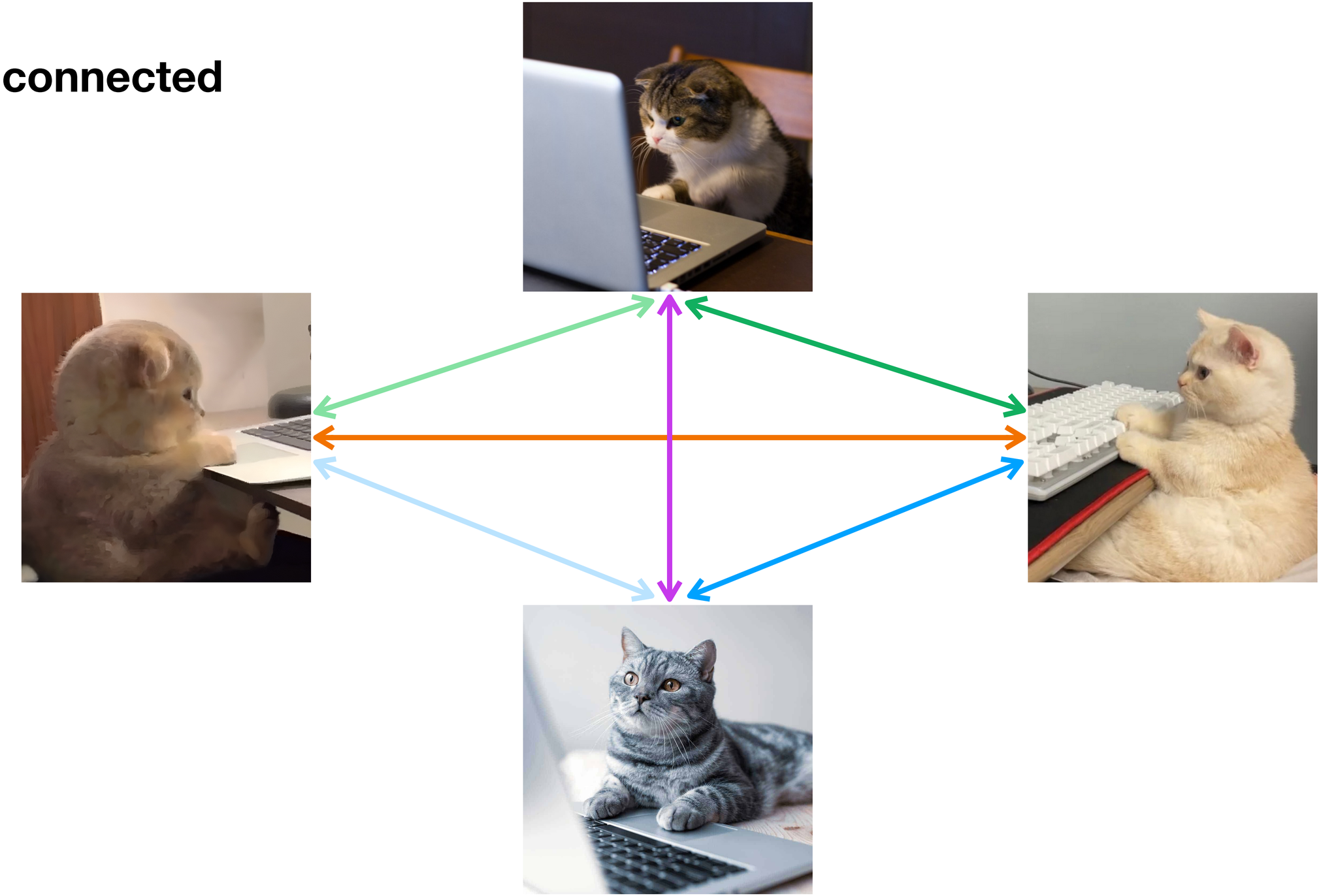


Как соединить участников?

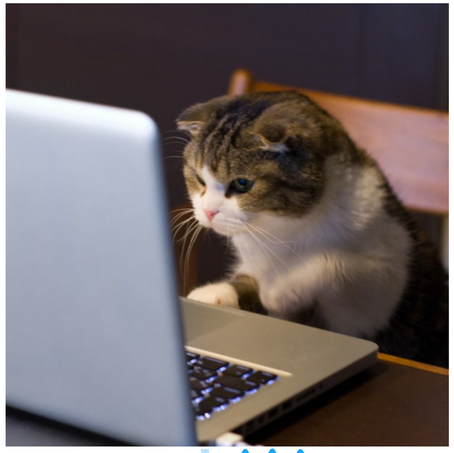


Архитектуры видеоконференций

Fully connected



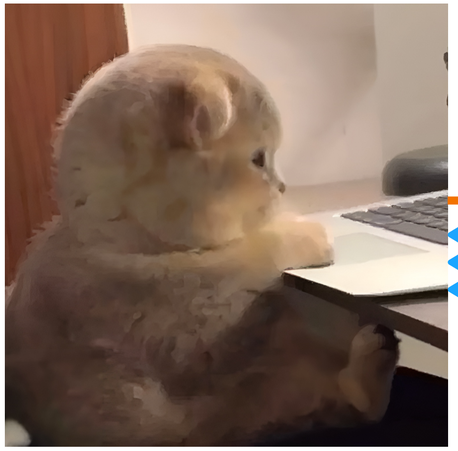
Selective Forwarding Unit

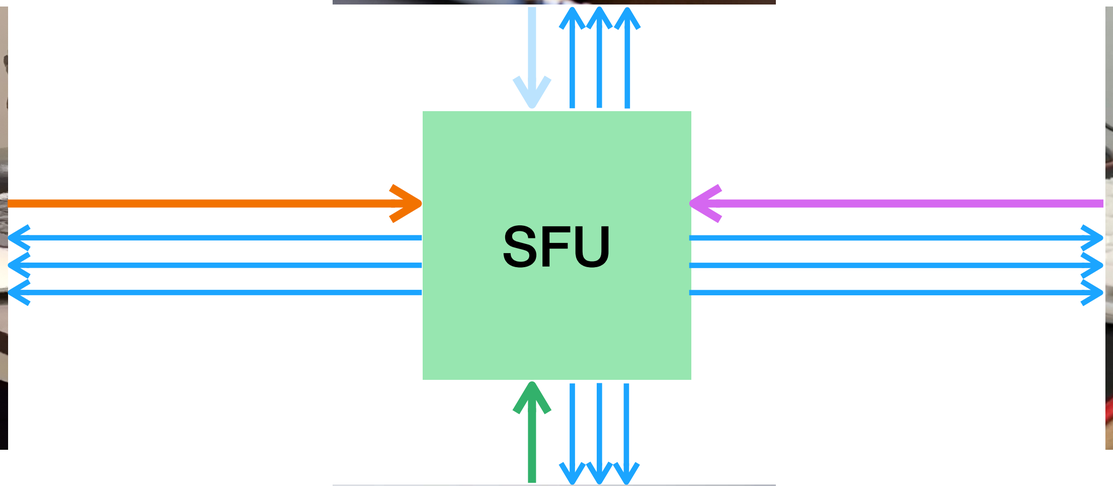


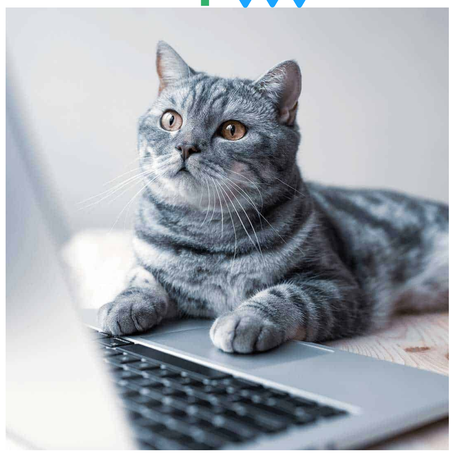
Клиент отправляет один пакет, а получает несколько

SFU пересылает полученный пакет другим клиентам

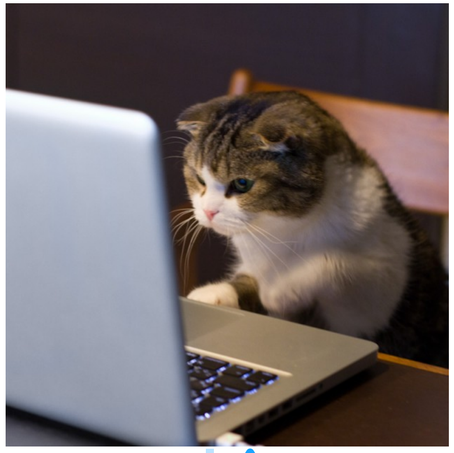
Нет кодирования / декодирования на SFU





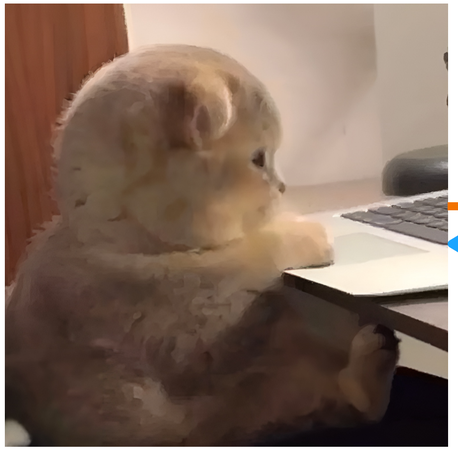


Multipoint Control Unit

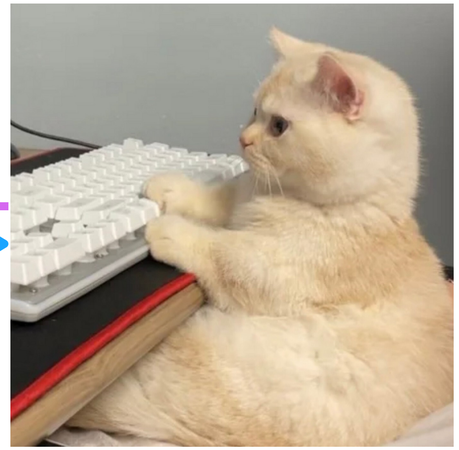


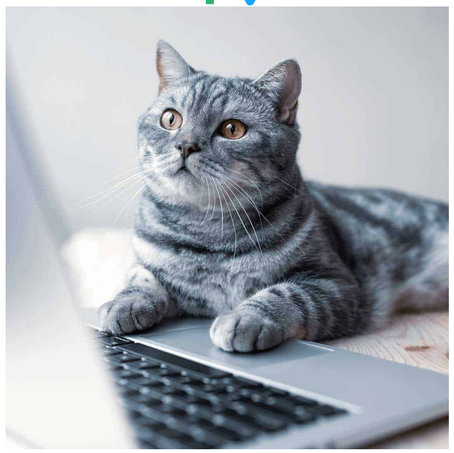
Клиент отправляет / получает один пакет в / из MCU

Объединяет несколько входящих пакетов в один исходящий



MCU





Что и когда выбрать?

P2P

До 5 участников

SFU

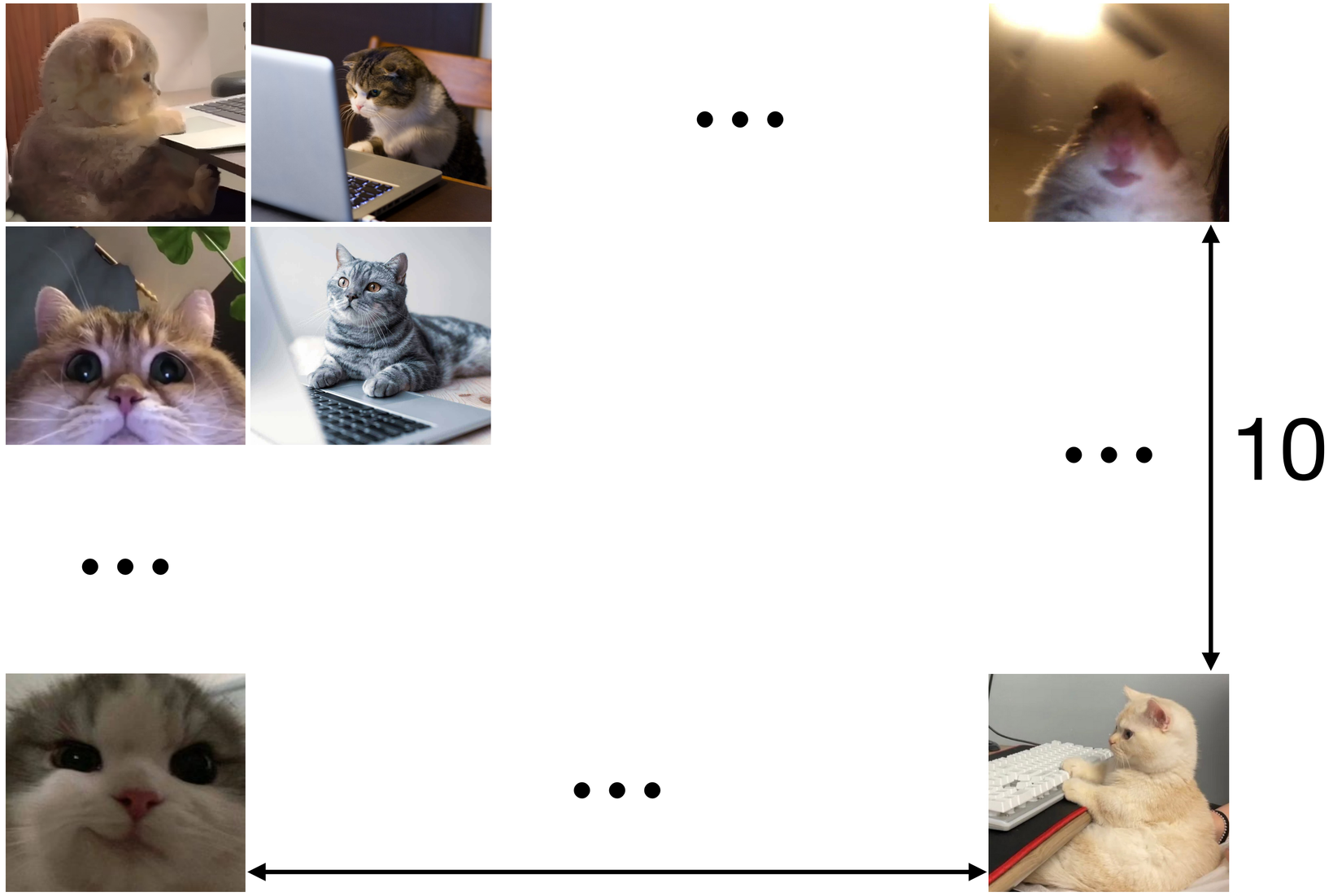
До 25 участников

MCU

25+ участников

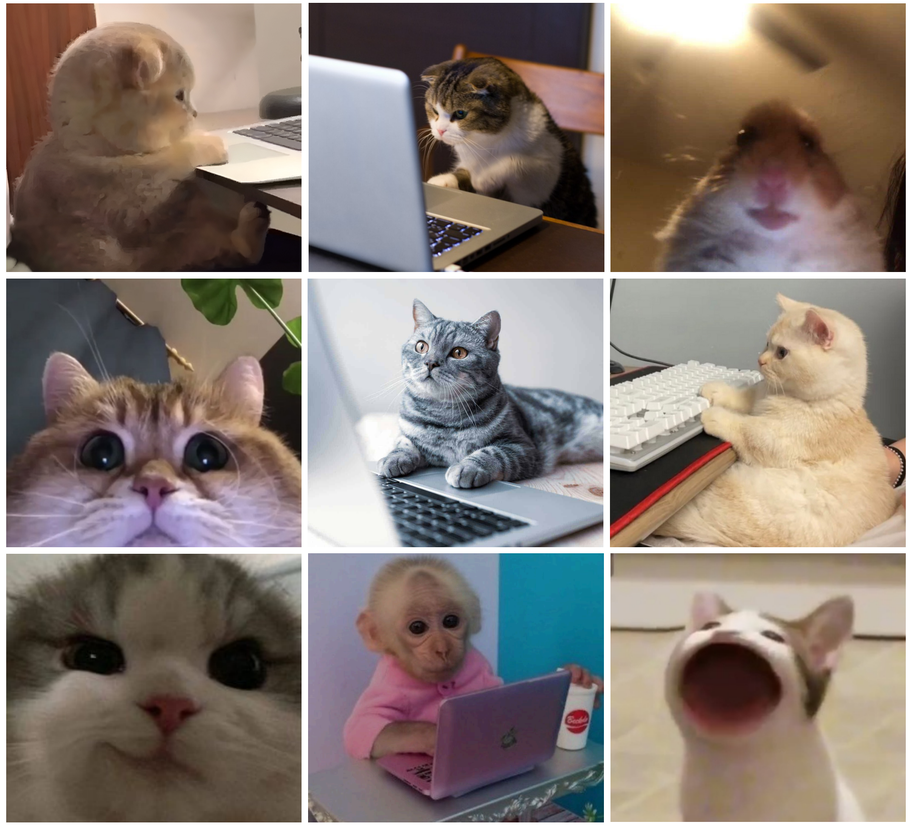
Массовые звонки готовы?!

Неа :)


10
20

Оптимизация!

Пагинация и slow start



←

→

Высокое качество только для основного говорящего

←

→

Особое построение SFU

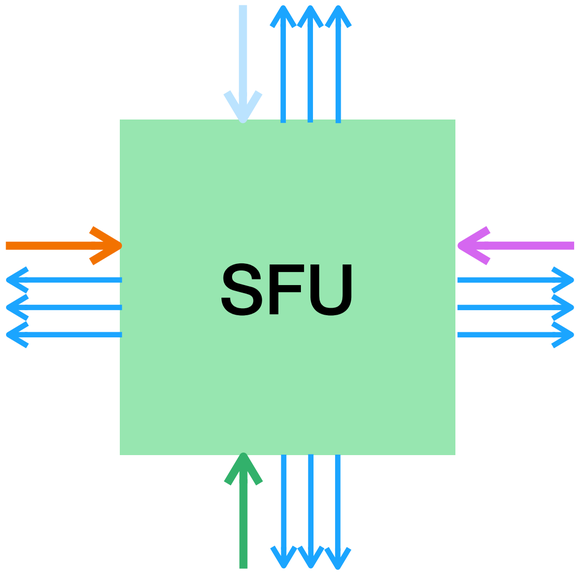
←

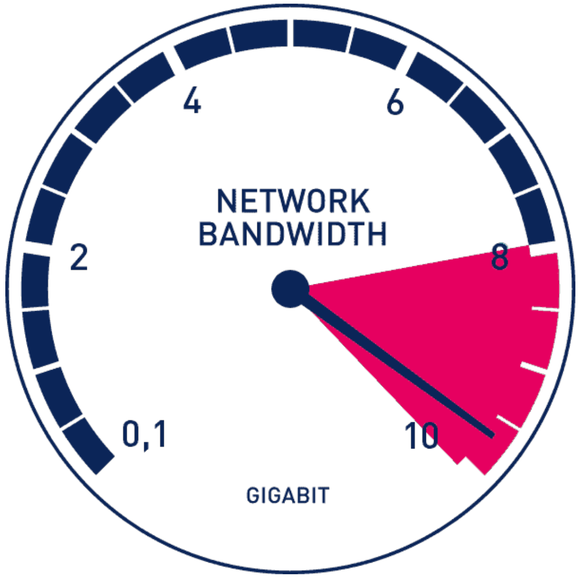
→

А как масштабировать?

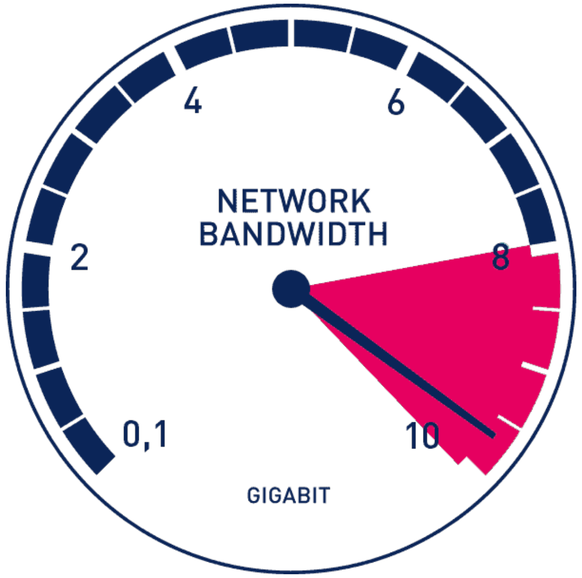
На основе CPU

В зависимости от пропускной способности



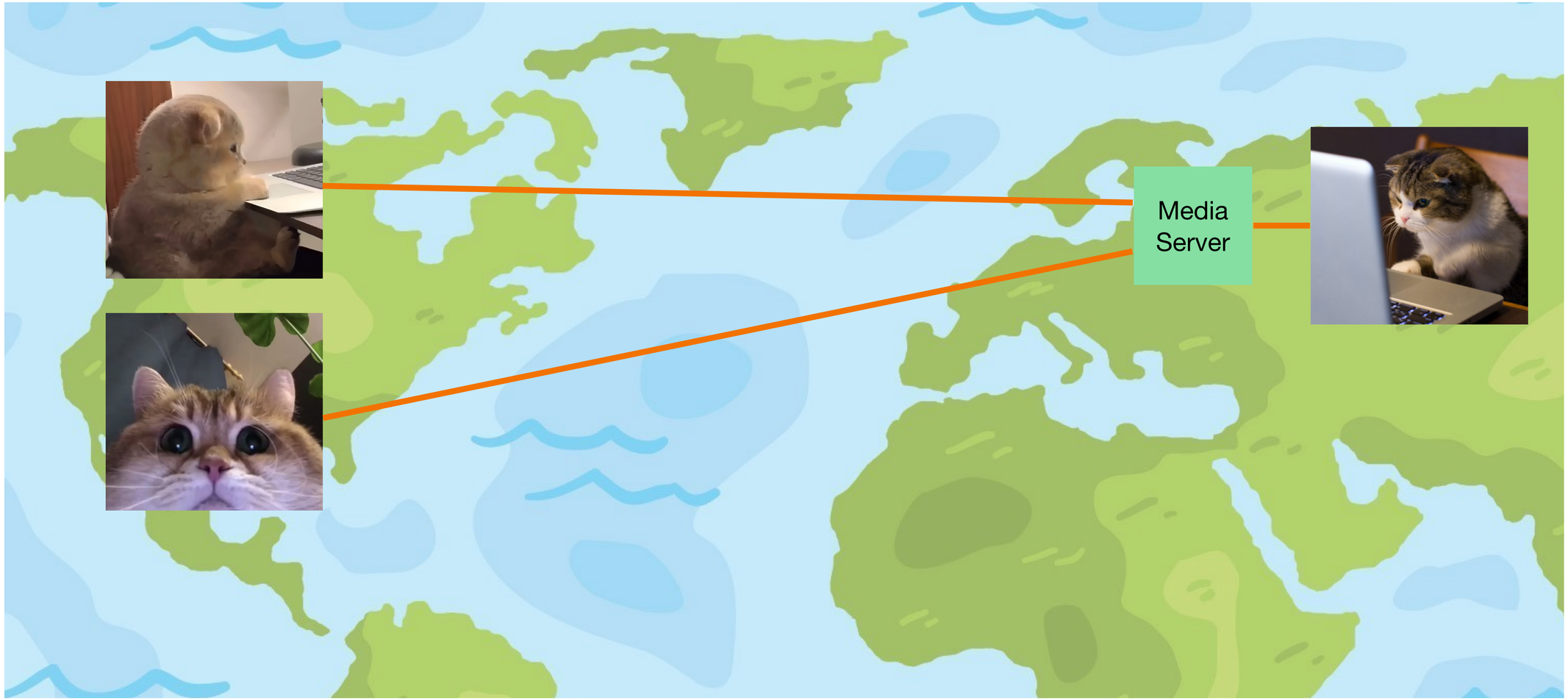


На основе потоков

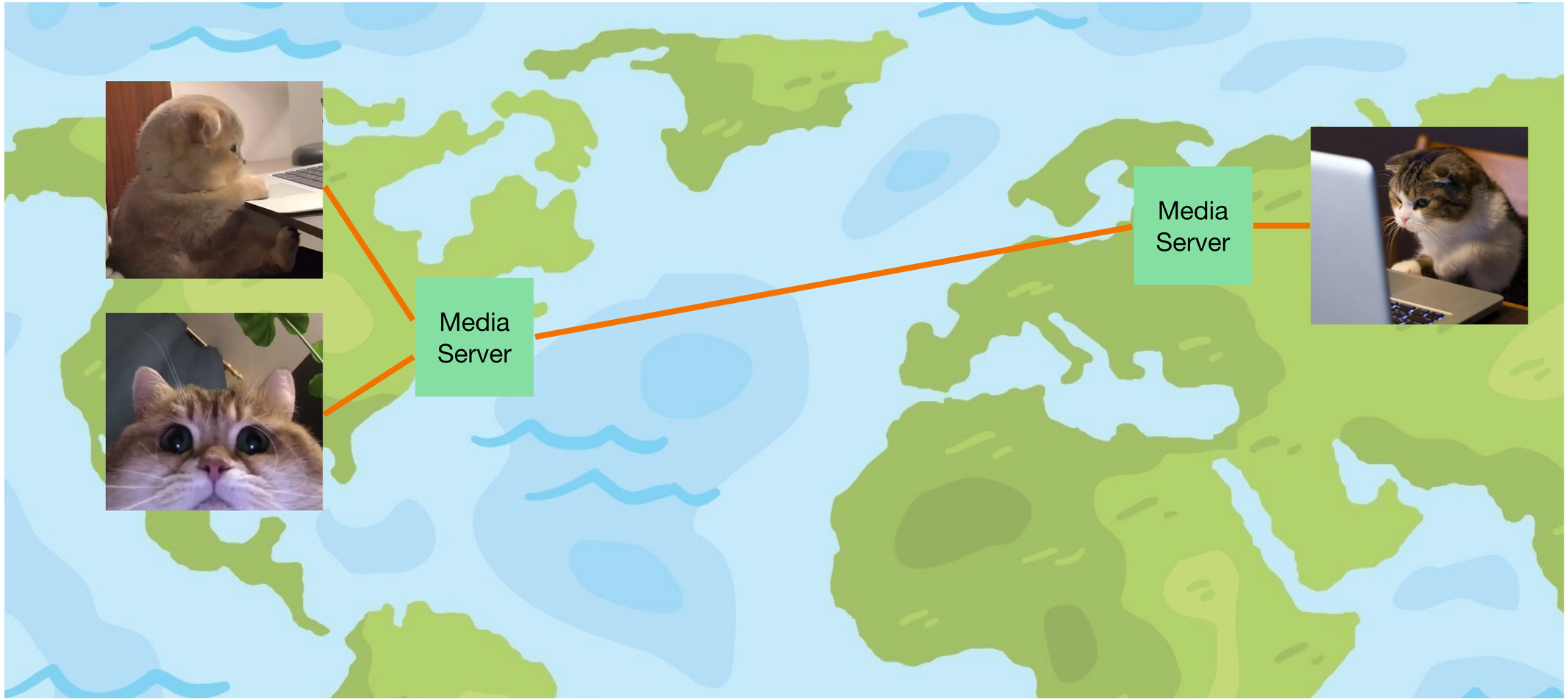


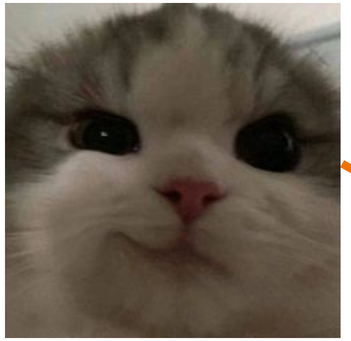
+ правила, основанные на количестве потоков

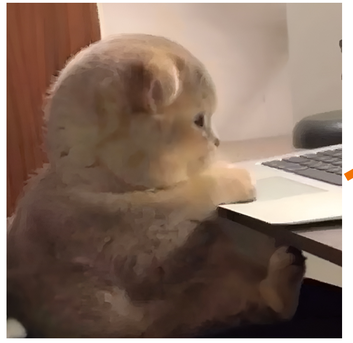
Каскадирование сеансов



Каскадирование сеансов







Signaling Server

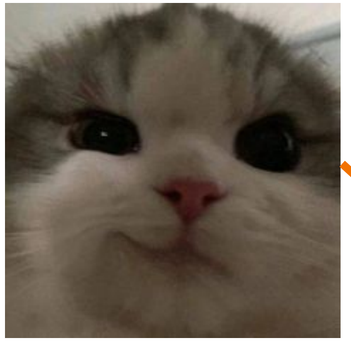
Media Requets Broker

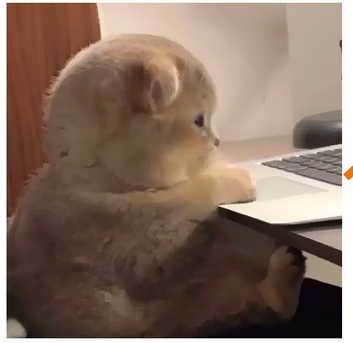
Media Server

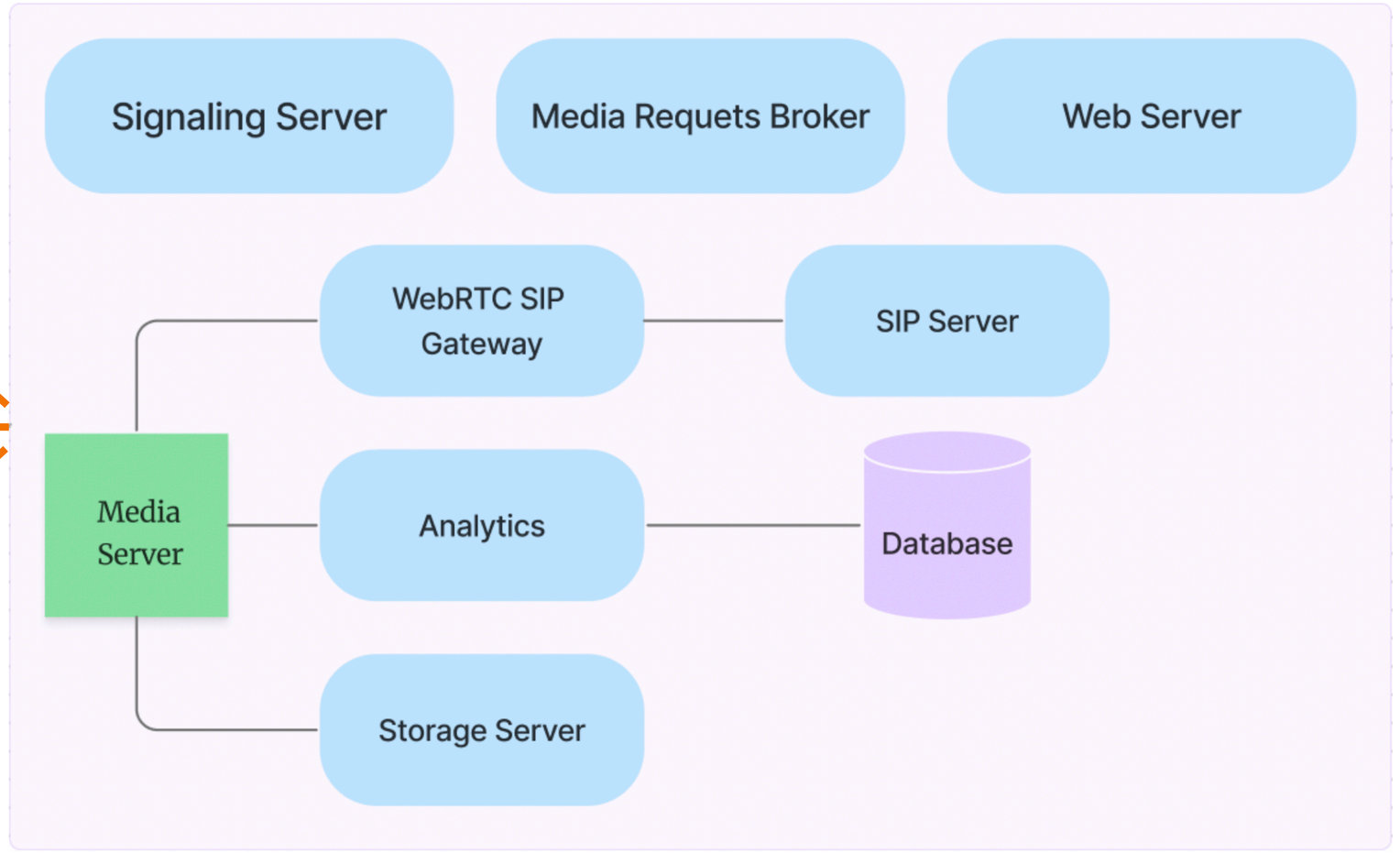
Media Server

Media Server

Media Server






Signaling Server
Media Requets Broker
Web Server
WebRTC SIP Gateway
SIP Server
Media Server
Analytics
Database
Storage Server

Создадим свое приложение!
React + Express

Создание веб-сервера

server.js > ...

```
1   const path = require("path");
2   const express = require("express");
3
4   const app = express();
5   const server = require("http").createServer(app);
6
7   const io = require("socket.io")(server);
8
9   const PORT = process.env.PORT || 3001;
10
11  server.listen(PORT, () => {
12    console.log("Server started!");
13  });
14
```

Подключение сокетов на клиенте

src > socket > JS index.js > ...

```
1   import { io } from "socket.io-client";
2
3   const options = {
4     "force new connection": true,
5     reconnectionAttempts: "Infinity",
6     timeout: 10000,
7     transports: ["websocket"],
8   };
9
10  const socket = io("http://localhost:3001", options);
11
12  export default socket;
13
```

Проверяем, что сокет подключился

server.js > shareRoomsInfo

```
1   const path = require("path");
2   const express = require("express");
3
4   const app = express();
5   const server = require("http").createServer(app);
6
7   const io = require("socket.io")(server);
8
9   const PORT = process.env.PORT || 3001;
10
11  io.on("connection", (socket) => {
12    console.log("Socket connected!");
13  });
14
15  server.listen(PORT, () => {
16    console.log("Server started!");
17  });
18
```

Получение комнат

server.js > ...

```
1   const path = require("path");
2   const express = require("express");
3   const { version, validate } = require("uuid");
4
5   const ACTIONS = require("./src/socket/actions");
6
7   const app = express();
8   const server = require("http").createServer(app);
9
10  const io = require("socket.io")(server);
11
12  const PORT = process.env.PORT || 3001;
13
14  function getClientRooms() {
15    const { rooms } = io.sockets.adapter;
16
17    return Array.from(rooms.keys()).filter(
18      (roomID) => validate(roomID) && version(roomID) === 4
19    );
20  }
21
22  function shareRoomsInfo() {
23    io.emit(ACTIONS.SHARE_ROOMS, {
24      rooms: getClientRooms(),
25    });
26  }
27
28  io.on("connection", (socket) => {
29    shareRoomsInfo();
30  });
31
32  server.listen(PORT, () => {
33    console.log("Server started!");
34  });
35
```

Возвращает массив всех комнаты

Получение массива комнат

Шарим доступные комнаты

server.js > ...

```
28  io.on("connection", (socket) => {
29    shareRoomsInfo();
30
31    socket.on(ACTIONS.JOIN, (config) => {
32      const { room: roomID } = config;
33      const { rooms: joinedRooms } = socket;
34
35      if (Array.from(joinedRooms).includes(roomID)) {
36        return console.warn(`Already joined to ${roomID}`);
37      }
38
39      const clients = Array.from(io.sockets.adapter.rooms.get(roomID) || []);
40
41      clients.forEach((clientID) => {
42        io.to(clientID).emit(ACTIONS.ADD_PEER, {
43          peerID: socket.id,
44          createOffer: false,
45        });
46
47        socket.emit(ACTIONS.ADD_PEER, {
48          peerID: clientID,
49          createOffer: true,
50        });
51      });
52
53      socket.join(roomID);
54      shareRoomsInfo();
55    });
56
```

Подключение к комнате

Массив подключенных юзеров

Подключенным юзерам не надо создавать offer

Новый юзер должен создать offer

Подключаемся и шарим информацию о всех комнатах

server.js > ...

```
57  function leaveRoom() {
58    const { rooms: joinedRooms } = socket;
59
60    Array.from(joinedRooms)
61      .filter((roomID) => validate(roomID) && version(roomID) === 4)
62      .forEach((roomID) => {
63        const clients = Array.from(io.sockets.adapter.rooms.get(roomID) || []);
64
65        clients.forEach((clientID) => {
66          io.to(clientID).emit(ACTIONS.REMOVE_PEER, {
67            peerID: socket.id,
68          });
69
70          socket.emit(ACTIONS.REMOVE_PEER, {
71            peerID: clientID,
72          });
73        });
74
75        socket.leave(roomID);
76      });
77
78    shareRoomsInfo();
79  }
80
81  socket.on(ACTIONS.LEAVE, leaveRoom);
82  socket.on("disconnecting", leaveRoom);
83 })
84
```

Выход из комнаты

Массив подключенных юзеров

Юзеры отключают нас

Мы отключаем юзеров

Выходим из комнаты

Шарим информацию о всех комнатах

Подключение к комнатам

```
8   const Main = () => {
9     const navigate = useNavigate();
10    const [rooms, setRooms] = useState([]);
11
12    useEffect(() => {
13      socket.on(ACTIONS.SHARE_ROOMS, ({ rooms } = []) => setRooms(rooms));
14    }, []);
15
16    const joinRoomHandler = (roomID) => {
17      navigate(`/room/${roomID}`);
18    };
19    const createNewRoomHandler = () => {
20      navigate(`/room/${v4()}`);
21    };
22
23    return (
24      <div>
25        <h1>Available rooms</h1>
26        <ul>
27          {rooms.map((roomID) => (
28            <li key={roomID}>
29              {roomID}
30              <button onClick={() => joinRoomHandler(roomID)}>Join room</button>
31            </li>
32          ))}
33        </ul>
34        <button onClick={createNewRoomHandler}>Create new room</button>
35      </div>
36    );
37  };
```

Стейт с комнатами

Получение массива доступных комнат

Выводим доступные комнаты на страницу

Создание компонента Room

src > pages > room > Room.jsx > ...

```jsx
import React from "react";
import { useParams } from "react-router";

import useWebRTC from "../../hooks/useWebRTC";

const Room = () => {
  const { id: roomID } = useParams();
  const {} = useWebRTC(roomID);

  return <div>Room</div>;
};

export default Room;
```

Здесь основная логика

Подключение к комнате

src > hooks > JS useWebRTC.js > ...

```
1   import { useRef } from "react";
2
3   import { useStateWithCb } from "./useStateWithCb";
4
5   export const LOCAL_VIDEO = "LOCAL_VIDEO";
6
7   export default function useWebRTC(roomID) {
8     const [clients, updateClients] = useStateWithCb([]);
9
10    const peerConnections = useRef({});
11    const localMediaStream = useRef({});
12    const peerMediaElements = useRef({
13      [LOCAL_VIDEO]: null,
14    });
15
```

Массив клиентов

Все пир соединения

Трансляция нашего видео и аудио

Ссылки на все медиа элементы
на странице

Реализация хука

src > hooks > JS useStateWithCb.js > ...

```
1   import { useCallback, useEffect, useRef, useState } from "react";
2
3   export const useStateWithCb = (initialState) => {
4     const [state, setState] = useState(initialState);
5     const cbRef = useRef(null);
6
7     const updateState = useCallback((newState, cb) => {
8       cbRef.current = cb;
9       setState((prev) =>
10        typeof newState === "function" ? newState(prev) : newState
11      );
12    }, []);
13
14    useEffect(() => {
15      if (cbRef.current) {
16        cbRef.current(state);
17        cbRef.current = null;
18      }
19    }, [state]);
20
21    return [state, updateState];
22  };
```

Сохраняем callback

Обновляем state

Вызываем callback после обновления стейта

src > hooks > JS useWebRTC.js > ...

```
19  const addNewClient = useCallback(
20    (newClient, cb) => {
21      if (!clients.includes(newClient))
22        updateClients((prevClients) => [...prevClients, newClient], cb);
23    },
24    [clients, updateClients]
25  );
26
27  useEffect(() => {
28    async function startCapture() {
29      localMediaStream.current = await navigator.mediaDevices.getUserMedia({
30        audio: true,
31        video: {
32          width: 1280,
33          height: 720,
34        },
35      });
36
37      addNewClient(LOCAL_VIDEO, () => {
38        const localVideoElement = peerMediaElements.current[LOCAL_VIDEO];
39
40        if (localVideoElement) {
41          localVideoElement.volume = 0;
42          localVideoElement.srcObject = localMediaStream.current;
43        }
44      });
45    }
46
47    startCapture()
48      .then(() => socket.emit(ACTIONS.JOIN, { room: roomID }))
49      .catch((e) => console.error("Error getting userMedia", e));
50  }, [roomID]);
51
```

Подключение к комнате

Получаем медиа

Добавляем юзера в массив клиентов

Достаем медиа

Убираем звук и в src тега <video/> устанавливаем медиа

Вызываем экшн JOIN

Если юзер не согласен, кидаем ошибку

Подключение к комнате

src > hooks > JS useWebRTC.js > ...

```
52    const provideMediaRef = useCallback((id, node) => {
53      peerMediaElements.current[id] = node;
54    }, []);
55
56    return { clients, provideMediaRef };
57  }
```

Выведем клиентов на экран

src > pages > room > Room.jsx > ...

```jsx
import React from "react";
import { useParams } from "react-router";

import useWebRTC, { LOCAL_VIDEO } from "../../hooks/useWebRTC";

const Room = () => {
  const { id: roomID } = useParams();
  const { clients, provideMediaRef } = useWebRTC(roomID);

  return (
    <div>
      {clients.map((clientID, index) => (
        <div key={clientID}>
          <video
            width="100%"
            height="100%"
            ref={(instance) => provideMediaRef(clientID, instance)}
            autoPlay
            playsInline
            muted={clientID === LOCAL_VIDEO}
          />
        </div>
      ))}
    </div>
  );
};

export default Room;
```

Достаем массив клиентов и функцию для получения доступа к <video/>

Выводим видео на страницу

Получаем доступ к элементу

Выход из комнаты

```
47      startCapture()
48        .then(() => socket.emit(ACTIONS.JOIN, { room: roomID }))
49        .catch((e) => console.error("Error getting userMedia", e));
50
51      return () => {
52        if (localMediaStream.current.getTracks) {
53          localMediaStream.current.getTracks().forEach((track) => track.stop());
54          socket.emit(ACTIONS.LEAVE);
55        }
56      };
57    }, [roomID]);
```

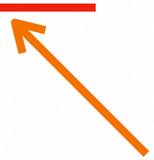
Останавливаем захват медиа

Вызываем экшн LEAVE

Добавление нового пира

server.js > ...

```
28  io.on("connection", (socket) => {
29    shareRoomsInfo();
30
31    socket.on(ACTIONS.JOIN, (config) => {
32      const { room: roomID } = config;
33      const { rooms: joinedRooms } = socket;
34
35      if (Array.from(joinedRooms).includes(roomID)) {
36        return console.warn(`Already joined to ${roomID}`);
37      }
38
39      const clients = Array.from(io.sockets.adapter.rooms.get(roomID) || []);
40
41      clients.forEach((clientID) => {
42        io.to(clientID).emit(ACTIONS.ADD_PEER, {
43          peerID: socket.id,
44          createOffer: false,
45        });
46
47        socket.emit(ACTIONS.ADD_PEER, {
48          peerID: clientID,
49          createOffer: true,
50        });
51      });
52
53      socket.join(roomID);
54      shareRoomsInfo();
55    });
```



Добавление нового пира

src > hooks > JS useWebRTC.js > ...

```
59    useEffect(() => {
60  >   const addNewPeerHandler = async ({ peerID, createOffer }) => {…
104     };
105
106     socket.on(ACTIONS.ADD_PEER, addNewPeerHandler);
107   }, []);
108
```

Добавление нового пира

src > hooks > JS useWebRTC.js > ...

```
59  useEffect(() => {
60    const addNewPeerHandler = async ({ peerID, createOffer }) => {
61      if (peerID in peerConnections.current) {
62        return console.warn(`Already connected to peer ${peerID}`);
63      }
64
65      peerConnections.current[peerID] = new RTCPeerConnection({
66        iceServers: freeice(),
67      });
68
69      peerConnections.current[peerID].onicecandidate = (e) => {
70        if (e.candidate) {
71          socket.emit(ACTIONS.RELAY_ICE, { peerID, iceCandidate: e.candidate });
72        }
73      };
74
75      let tracksNumber = 0;
76      peerConnections.current[peerID].ontrack = ({
77        streams: [remoteStream],
78      }) => {
79        tracksNumber++;
80
81        if (tracksNumber === 2) {
82          addNewClient(peerID, () => {
83            peerMediaElements.current[peerID].srcObject = remoteStream;
84          });
85        }
86      };
```

Создаем RTCPeerConnection

Обработчик на вызов setLocalDescription

Вызываем экшн RELAY_ICE с peerID и ice кандидатом

Обработчик на появление медиа трека

Пришло видео и аудио

Добавляем клиента и транслируем медиа

Добавление нового пира

src > hooks > JS useWebRTC.js > ...

```
88        localMediaStream.current.getTracks().forEach((track) => {
89          peerConnections.current[peerID].addTrack(
90            track,
91            localMediaStream.current
92          );
93        });
94
95        if (createOffer) {
96          const offer = await peerConnections.current[peerID].createOffer();
97          await peerConnections.current[peerID].setLocalDescription(offer);
98
99          socket.emit(ACTIONS.RELAY_SDP, {
100           peerID,
101           sessionDescription: offer,
102         });
103       }
104     };
105
106     socket.on(ACTIONS.ADD_PEER, addNewPeerHandler);
107   }, []);
```

Добавляем треки из localMediaStream к peerConnections

Создаем offer

Прикрепляем offer

Вызываем экшн RELAY_SDP с SDP данными

Добавление нового пира

server.js > ...

```
85    socket.on(ACTIONS.RELAY_SDP, ({ peerID, sessionDescription }) => {
86      io.to(peerID).emit(ACTIONS.SESSION_DESCRIPTION, {
87        peerID: socket.id,
88        sessionDescription,
89      });
90    });
91
92    socket.on(ACTIONS.RELAY_ICE, ({ peerID, iceCandidate }) => {
93      io.to(peerID).emit(ACTIONS.ICE_CANDIDATE, {
94        peerID: socket.id,
95        iceCandidate,
96      });
97    });
98  });
```

src > hooks > JS useWebRTC.js > ...

Добавление нового пира

```
107  useEffect(() => {
108    async function setRemoteMedia({
109      peerID,
110      sessionDescription: remoteDescription,
111    }) {
112      await peerConnections.current[peerID]?.setRemoteDescription(
113        new RTCSessionDescription(remoteDescription)
114      );
115
116      if (remoteDescription.type === "offer") {
117        const answer = await peerConnections.current[peerID].createAnswer();
118
119        await peerConnections.current[peerID].setLocalDescription(answer);
120
121        socket.emit(ACTIONS.RELAY_SDP, {
122          peerID,
123          sessionDescription: answer,
124        });
125      }
126    }
127
128    socket.on(ACTIONS.SESSION_DESCRIPTION, setRemoteMedia);
129
130    return () => {
131      socket.off(ACTIONS.SESSION_DESCRIPTION);
132    };
133  }, []);
```

Прикрепляем SDP данные

Создаем answer

Прикрепляем anser

Вызываем экшн RELAY_SDP с SDP данными

Добавление нового пира

src > hooks > JS useWebRTC.js > ...

```
135   useEffect(() => {
136     socket.on(ACTIONS.ICE_CANDIDATE, ({ peerID, iceCandidate }) => {
137       peerConnections.current[peerID].addIceCandidate(
138         new RTCIceCandidate(iceCandidate)
139       );
140     });
141   }, []);
142
```

Удаление пира

server.js > ...

```
57  function leaveRoom() {
58    const { rooms: joinedRooms } = socket;
59
60    Array.from(joinedRooms)
61      .filter((roomID) => validate(roomID) && version(roomID) === 4)
62      .forEach((roomID) => {
63        const clients = Array.from(io.sockets.adapter.rooms.get(roomID) || []);
64
65        clients.forEach((clientID) => {
66          io.to(clientID).emit(ACTIONS.REMOVE_PEER, {
67            peerID: socket.id,
68          });
69
70          socket.emit(ACTIONS.REMOVE_PEER, {
71            peerID: clientID,
72          });
73        });
74
75        socket.leave(roomID);
76      });
77
78    shareRoomsInfo();
79  }
80
81  socket.on(ACTIONS.LEAVE, leaveRoom);
82  socket.on("disconnecting", leaveRoom);
83 })
84
```

Удаление пира

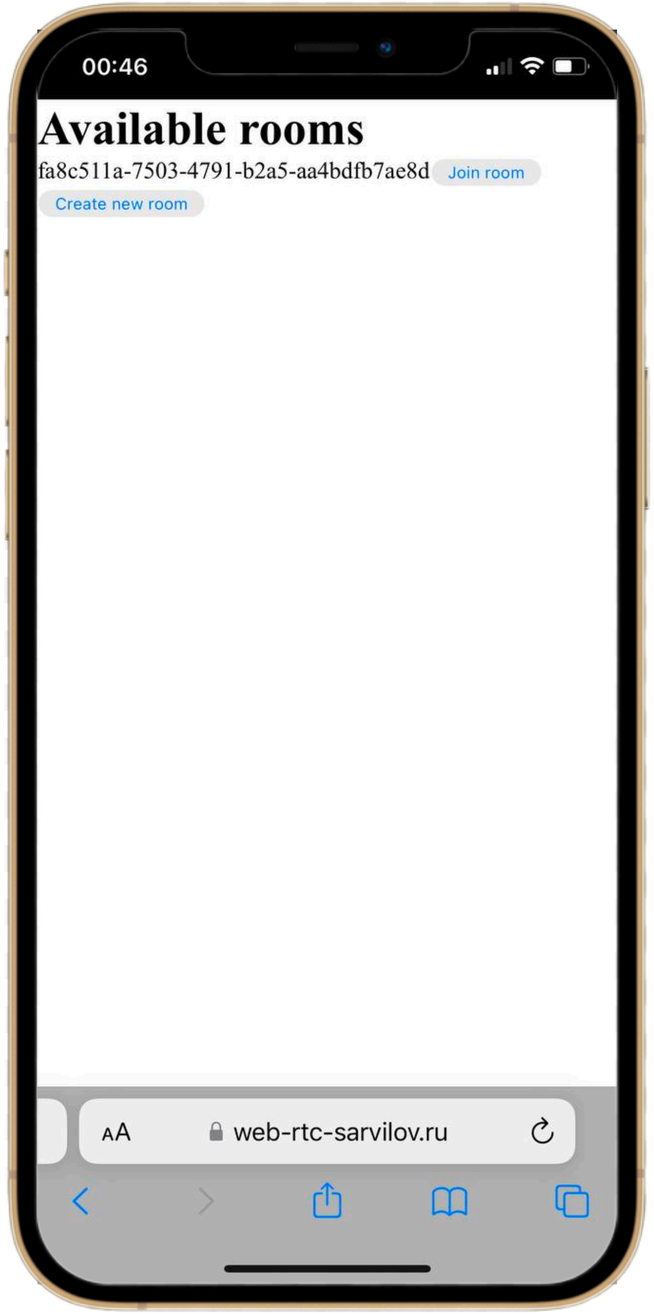
src > hooks > JS useWebRTC.js > ...

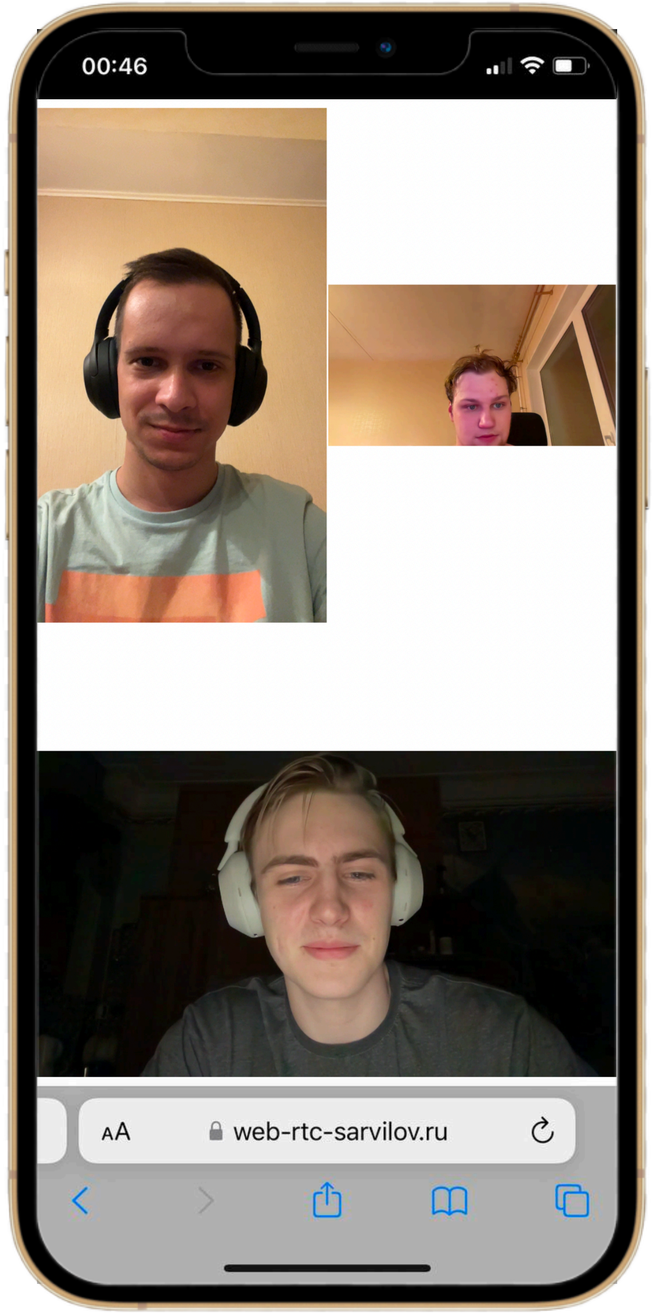
```
145   useEffect(() => {
146     const removePeerHandler = ({ peerID }) => {
147       if (peerID in peerConnections.current) {
148         peerConnections.current[peerID].close();
149       }
150       delete peerConnections.current[peerID];
151       delete peerMediaElements.current[peerID];
152       updateClients((prevClients) =>
153         prevClients.filter((client) => client !== peerID)
154       );
155     };
156
157     socket.on(ACTIONS.REMOVE_PEER, removePeerHandler);
158   }, []);
```

Закрываем соединение

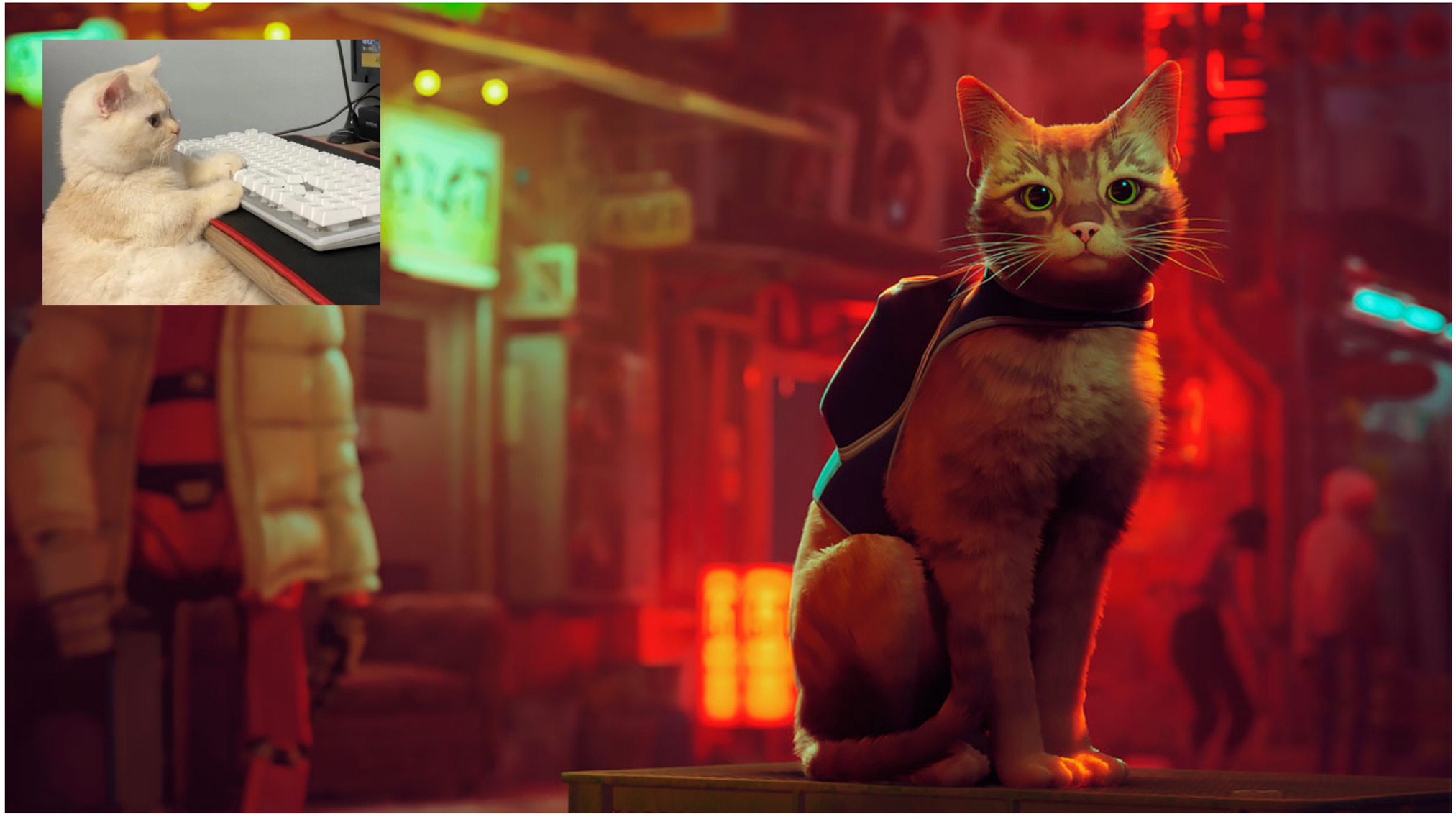
Удаляем из peerConnections и peerElements

Обновляем клиентов

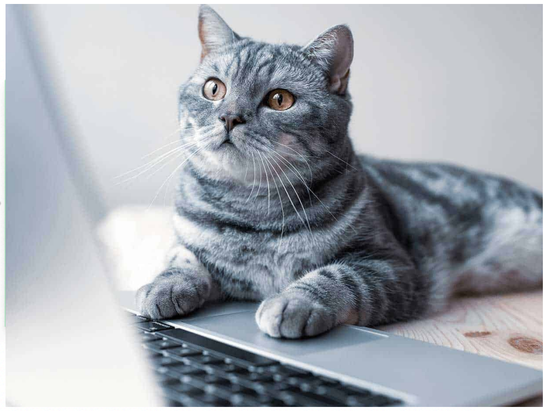

00:46
Available rooms
fa8c511a-7503-4791-b2a5-aa4bdfb7ae8d
Join room
Create new room
AA
web-rtc-sarvilov.ru


00:46
AA
web-rtc-sarvilov.ru

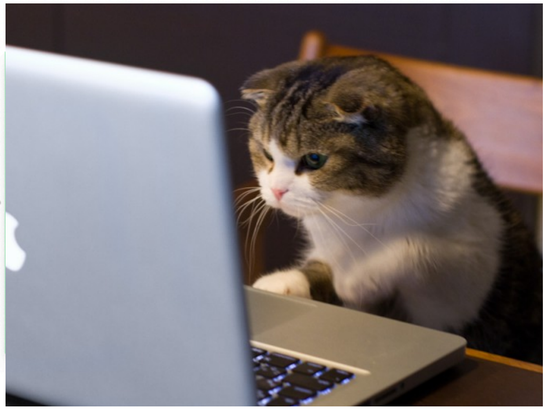
Создание трансляций



Media
Server



100



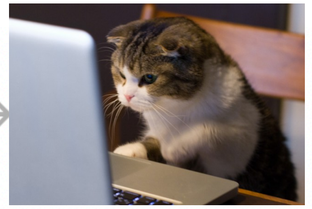
А если больше зрителей?

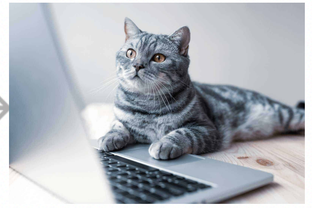
Media Server

Media Server

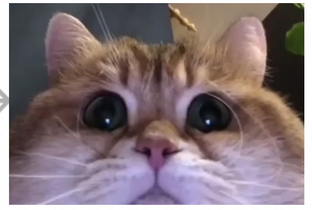
Media Server

100





100



100

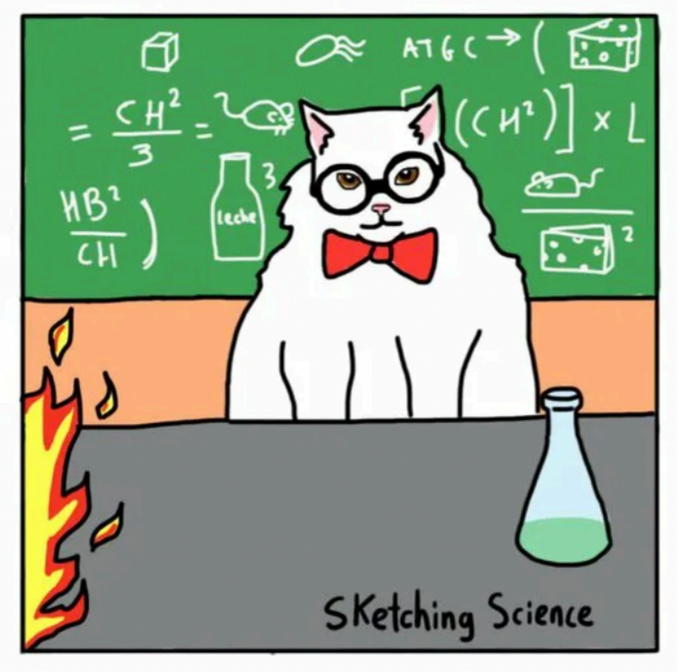
Как обрабатывать сетевые ошибки?

Jitter buffer

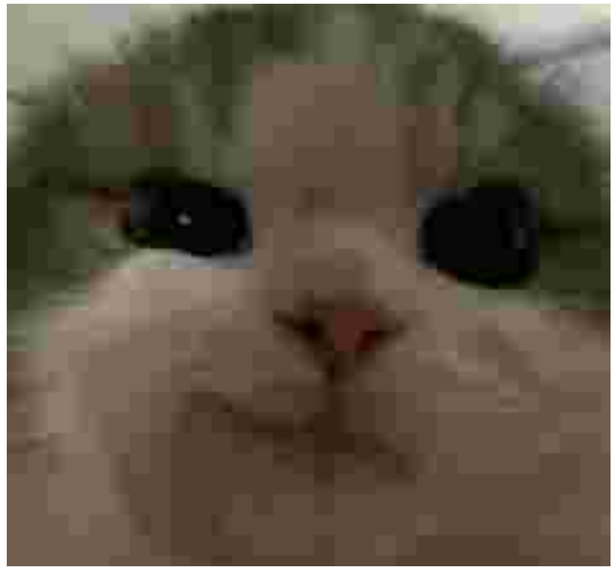
Трафик

Юзер

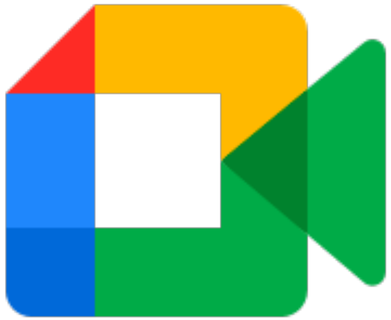
Forward Error Correction



Уменьшение битрейта



Примеры



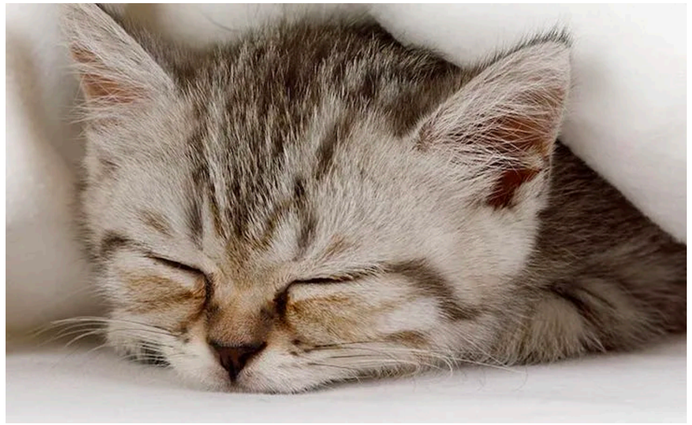
STADIA

WebRTC не магия!

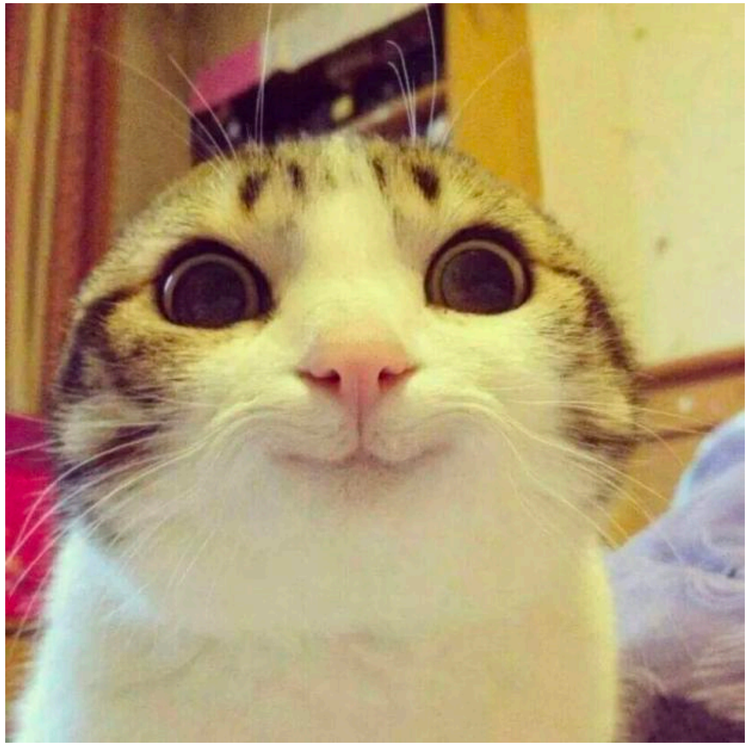
RTP, TCP, UDP, NAT, STUN, TURN, ICE — страшные аббревиатуры, но их не стоит бояться!

Вкатиться в технологию не сложно

Можно создавать полезные приложения



Спасибо!



Telegram

@k0ct9lhbi4

Email

sarvilov.k@mail.ru