SFU Two Way Streaming

Пример демонстрирует публикацию одного или нескольких потоков в нескольких качествах в одном WebRTC соединении, и проигрывание этих потоков. Единицей публикации считается комната, то есть зрители, подключившись к этой комнате, получают все опубликованные в ней потоки.

На скриншотах ниже:

- Server url - Websocket URL WCS сервера
- Room name - имя комнаты
- Publisher - имя пользователя, который публикует потоки

SFU Two-way Streaming

Server url

ws://localhost:8080

Room name

ROOM1-6f4b

Publisher

Publisher1-846d Stop

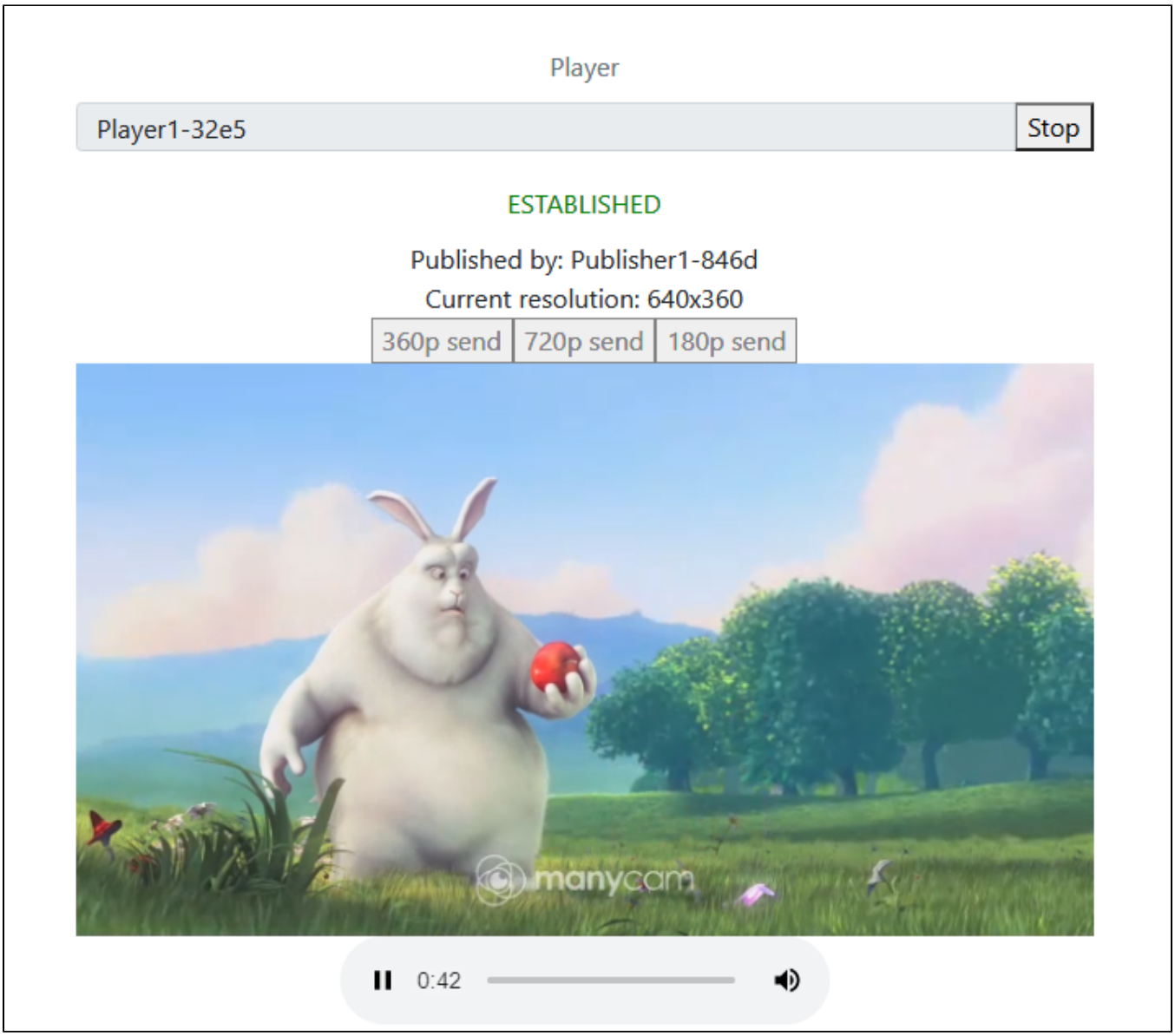
ESTABLISHED

Name: Publisher1-846d

Max.resolution: 1280x720

Mute

- Player - имя пользователя, который играет потоки
- 360p, 720p, 180p send - кнопки переключения принимаемого качества



Обратите внимание, что аудио потоки проигрываются в отдельных элементах

Исходный код примера

Исходный код разбит на следующие модули:

- two-way-streaming.html - HTML страница
- two-way-streaming.css - стили HTML страницы
- two-way-streaming.js - основная логика приложения
- config.json - файл конфигурации клиента, содержит описание публикуемых потоков

Анализ исходного кода

Для работы с исходным кодом примера возьмем версию файла two-way-streaming.js, доступнуюздесь

1. Локальные переменные

Объявление локальных переменных для работы с константами, SFU SDK, для отображения видео и работы с конфигурацией клиента

code

```
const constants = SFU.constants;
const sfu = SFU;
let mainConfig;
let localDisplay;
let remoteDisplay;
let publishState;
let playState;
const PUBLISH = "publish";
const PLAY = "play";
const STOP = "stop";
const PRELOADER_URL="../commons/media/silence.mp3"
```

2. Конфигурация по умолчанию

Объявление конфигурации комнаты и публикации потоков по умолчанию, на случай, если нет файла конфигурации config.json

code

```
const defaultConfig = {
    room: {
        url: "wss://127.0.0.1:8888",
        name: "ROOM1",
        pin: "1234",
        nickName: "User1"
    },
    media: {
        audio: {
            tracks: [
                {
                    source: "mic",
                    channels: 1
                }
            ]
        },
        video: {
            tracks: [
                {
                    source: "camera",
                    width: 640,
                    height: 360,
                    codec: "H264",
                    encodings: [
                        { rid: "360p", active: true, maxBitrate: 500000 },
                        { rid: "180p", active: true, maxBitrate: 200000, scaleResolutionDownBy: 2 }
                    ]
                }
            ]
        }
    }
};
```

3. Объект для хранения текущего состояния публикации/проигрывания

Хранит данные Websocket сессии, WebRTC соединения и комнаты, формирует идентификаторы элементов на странице для доступа к ним

code

```
const CurrentState = function(prefix) {
    let state = {
        prefix: prefix,
        pc: null,
        session: null,
        room: null,
        timer: null,
        set: function(pc, session, room) {
            state.pc = pc;
            state.session = session;
            state.room = room;
        },
        clear: function() {
            state.stopWaiting();
            state.room = null;
            state.session = null;
            state.pc = null;
        },
        waitFor: function(div, timeout) {
            state.stopWaiting();
            state.timer = setTimeout(function () {
                if (div.innerHTML !== "") {
                    // Enable stop button
                    $("#" + state.buttonId()).prop('disabled', false);
                }
                else if (state.isConnected()) {
                    setStatus(state.errInfoId(), "No media capturing started in " + timeout + " ms, stopping",
"red");
                    onStopClick(state);
                }
            }, timeout);
        },
        stopWaiting: function() {
            if (state.timer) {
                clearTimeout(state.timer);
                state.timer = null;
            }
        },
        buttonId: function() {
            return state.prefix + "Btn";
        },
        buttonText: function() {
            return (state.prefix.charAt(0).toUpperCase() + state.prefix.slice(1));
        },
        inputId: function() {
            return state.prefix + "Name";
        },
        statusId: function() {
            return state.prefix + "Status";
        },
        formId: function() {
            return state.prefix + "Form";
        },
        errInfoId: function() {
            return state.prefix + "ErrorInfo";
        },
        is: function(value) {
            return (prefix === value);
        },
        isActive: function() {
            return (state.room && state.pc);
        },
        isConnected: function() {
            return (state.session && state.session.state() == constants.SFU_STATE.CONNECTED);
        }
    };
    return state;
}
```

4. Инициализация

init()code

Функция init() вызывается после того, как страница загрузится:

- инициализирует объекты состояния
- загружает config.json или конфигурацию по умолчанию
- инициализирует поля ввода

```
const init = function() {
    let configName = getUrlParam("config") || "./config.json";
    ...
    publishState = CurrentState(PUBLISH);
    playState = CurrentState(PLAY);
    $.getJSON(configName, function(cfg){
        mainConfig = cfg;
        onDisconnected(publishState);
        onDisconnected(playState);
    }).fail(function(e){
        //use default config
        console.error("Error reading configuration file " + configName + ": " + e.status + " " + e.statusText)
        console.log("Default config will be used");
        mainConfig = defaultConfig;
        onDisconnected(publishState);
        onDisconnected(playState);
    });
    $("#url").val(setURL());
    $("#roomName").val("ROOM1-"+createUUID(4));
    $("#publishName").val("Publisher1-"+createUUID(4));
    $("#playName").val("Player1-"+createUUID(4));
}
```

5. Соединение с сервером

connect(), SFU.createRoom() code

Функция connect() вызывается по нажатию кнопки Publish или Play:

- создает объект PeerConnection
- очищает отображение статуса предыдущей сессии
- настраивает конфигурацию комнаты и создает Websocket сессию
- подписывается на события Websocket сессии

```
const connect = function(state) {
    //create peer connection
    pc = new RTCPeerConnection();
    //get config object for room creation
    const roomConfig = getRoomConfig(mainConfig);
    roomConfig.pc = pc;
    roomConfig.url = $("#url").val();
    roomConfig.roomName = $("#roomName").val();
    roomConfig.nickname = $("#" + state.inputId()).val();
    // clean state display items
    setStatus(state.statusId(), "");
    setStatus(state.errInfoId(), "");
    // connect to server and create a room if not
    const session = sfu.createRoom(roomConfig);
    session.on(constants.SFU_EVENT.CONNECTED, function(room) {
        state.set(pc, session, room);
        onConnected(state);
        setStatus(state.statusId(), "ESTABLISHED", "green");
    }).on(constants.SFU_EVENT.DISCONNECTED, function() {
        state.clear();
        onDisconnected(state);
        setStatus(state.statusId(), "DISCONNECTED", "green");
    }).on(constants.SFU_EVENT.FAILED, function(e) {
        state.clear();
        onDisconnected(state);
        setStatus(state.statusId(), "FAILED", "red");
        setStatus(state.errInfoId(), e.status + " " + e.statusText, "red");
    });
}
```

6. Запуск публикации или проигрывания при установке соединения

onConnected() code

Функция onConnected():

- настраивает действия по нажатию кнопки Stop
- подписывается на события об ошибках комнаты
- вызывает функцию публикации или проигрывания

```
const onConnected = function(state) {
    $("#" + state.buttonId()).text("Stop").off('click').click(function () {
        onStopClick(state);
    });
    ...
    // Add errors displaying
    state.room.on(constants.SFU_ROOM_EVENT.FAILED, function(e) {
        setStatus(state.errInfoId(), e, "red");
        stopStreaming(state);
    }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
        setStatus(state.errInfoId(), e.operation + " failed: " + e.error, "red");
        stopStreaming(state);
    });
    startStreaming(state);
}
```

7. Публикация потоков

publishStreams(), SFURoom.join() code

Функция publishStreams():

- инициализирует базовый элемент для отображения локального видео
- получает доступ к локальным медиа потокам согласно файлу конфигурации
- добавляет медиа дорожки в WebRTC соединение
- входит в комнату на сервере
- запускает таймер ожидания успешной инициализации локальных видео элементов

```
const publishStreams = async function(state) {
    if (state.isConnected()) {
        //create local display item to show local streams
        localDisplay = initLocalDisplay(document.getElementById("localVideo"));
        try {
            //get configured local video streams
            let streams = await getVideoStreams(mainConfig);
            let audioStreams = await getAudioStreams(mainConfig);
            if (state.isConnected() && state.isActive()) {
                //combine local video streams with audio streams
                streams.push.apply(streams, audioStreams);
                let config = {};
                //add our local streams to the room (to PeerConnection)
                streams.forEach(function (s) {
                    //add local stream to local display
                    localDisplay.add(s.stream.id, $("#" + state.inputId()).val(), s.stream);
                    //add each track to PeerConnection
                    s.stream.getTracks().forEach((track) => {
                        if (s.source === "screen") {
                            config[track.id] = s.source;
                        }
                        addTrackToPeerConnection(state.pc, s.stream, track, s.encodings);
                        subscribeTrackToEndedEvent(state.room, track, state.pc);
                    });
                });
                state.room.join(config);
                // TODO: Use room state or promises to detect if publishing started to enable stop button
                state.waitFor(document.getElementById("localVideo"), 3000);
            }
        } catch(e) {
            console.error("Failed to capture streams: " + e);
            setStatus(state.errInfoId(), e.name, "red");
            state.stopWaiting();
            if (state.isConnected()) {
                onStopClick(state);
            }
        }
    }
}
```

7.1. Добавление медиа дорожек в WebRTC соединение

addTrackToPeerConnection(), PeerConnection.addTransceiver() code

```
const addTrackToPeerConnection = function(pc, stream, track, encodings) {
    pc.addTransceiver(track, {
        direction: "sendonly",
        streams: [stream],
        sendEncodings: encodings ? encodings : [] //passing encoding types for video simulcast tracks
    });
}
```

7.2. Подписка на событие остановки потока

subscribeTrackToEndedEvent(), MediaTrack.addEventListener(), SFURoom.updateState() code

```
const subscribeTrackToEndedEvent = function(room, track, pc) {
    track.addEventListener("ended", function() {
        //track ended, see if we need to cleanup
        let negotiate = false;
        for (const sender of pc.getSenders()) {
            if (sender.track === track) {
                pc.removeTrack(sender);
                //track found, set renegotiation flag
                negotiate = true;
                break;
            }
        }
        if (negotiate) {
            //kickoff renegotiation
            room.updateState();
        }
    });
};
```

8. Проигрывание потоков

playStreams(), SFURoom.join() code

Функция playStreams():

- инициализирует базовый элемент для отображения входящих медиа потоков
- настраивает

```
const playStreams = function(state) {
    if (state.isConnected() && state.isActive()) {
        //create remote display item to show remote streams
        remoteDisplay = initRemoteDisplay({
            div: document.getElementById("remoteVideo"),
            room: state.room,
            peerConnection: state.pc
        });
        state.room.join(state.pc);
    }
    $("#" + state.buttonId()).prop('disabled', false);
}
```

9. Остановка публикации

unPublishStreams(), localDisplay.stop() code

```
const unPublishStreams = function(state) {
    if (localDisplay) {
        localDisplay.stop();
    }
}
```

10. Остановка проигрывания

stopStreams(), remoteDisplay.stop() code

```
const stopStreams = function(state) {
    if (remoteDisplay) {
        remoteDisplay.stop();
    }
}
```

11. Действия по нажатию кнопки Publish/Play

onStartClick(), playFirstSound(), connect() code

Функция onStartClick():

- проверяет правильность заполнения полей ввода
- перед стартом воспроизведения, в браузере Safari вызывает функцию playFirstSound() для автоматического проигрывания аудио
- вызывает функцию connect()

```
const onStartClick = function(state) {
    if (validateForm("connectionForm") && validateForm(state.formId())) {
        $("#" + state.buttonId()).prop('disabled', true);
        if (state.is(PLAY) && Browser().isSafariWebRTC()) {
            playFirstSound(document.getElementById("main"), PRELOADER_URL).then(function () {
                connect(state);
            });
        } else {
            connect(state);
        }
    }
}
```

12. Действия по нажатию кнопки Stop

onStopClick(), Session.disconnect() code

Функция onStopClick():

- останавливает публикацию или воспроизведение
- разрывает Websocket сессию

```
const onStopClick = function(state) {
    $("#" + state.buttonId()).prop('disabled', true);
    stopStreaming(state);
    if (state.isConnected()) {
        state.session.disconnect();
    }
}
```

13. Действия при разрыве Websocket сессии

onDisconnected() code

Функция onDisconnected():

- настраивает действия по нажатию кнопки Publish/Play
- открывает доступ к полям ввода Server url и Room name, если нет параллельной сессии

```
const onDisconnected = function(state) {
    $("#" + state.buttonId()).text(state.buttonText()).off('click').click(function () {
        onStartClick(state);
    }).prop('disabled', false);
    $("#" + state.inputId()).prop('disabled', false);
    // Check if other session is active
    if ((state.is(PUBLISH) && playState.session)
       || (state.is(PLAY) && publishState.session)) {
        return;
    }
    $('#url').prop('disabled', false);
    $("#roomName").prop('disabled', false);
}
```

14. Вспомогательные функции

14.1. Запуск публикации или проигрывания

startStreaming() code

```
const startStreaming = function(state) {
    if (state.is(PUBLISH)) {
        publishStreams(state);
    } else if (state.is(PLAY)) {
        playStreams(state);
    }
}
```

14.2. Остановка публикации или проигрывания

stopStreaming() code

```
const stopStreaming = function(state) {
    state.stopWaiting();
    if (state.is(PUBLISH)) {
        unPublishStreams(state);
    } else if (state.is(PLAY)) {
        stopStreams(state);
    }
}
```