Streaming Auto Restore

Пример стримера с автоматическим восстановлением публикации /проигрывания

Данный пример показывает различные способы восстановления публикации и проигрывания:

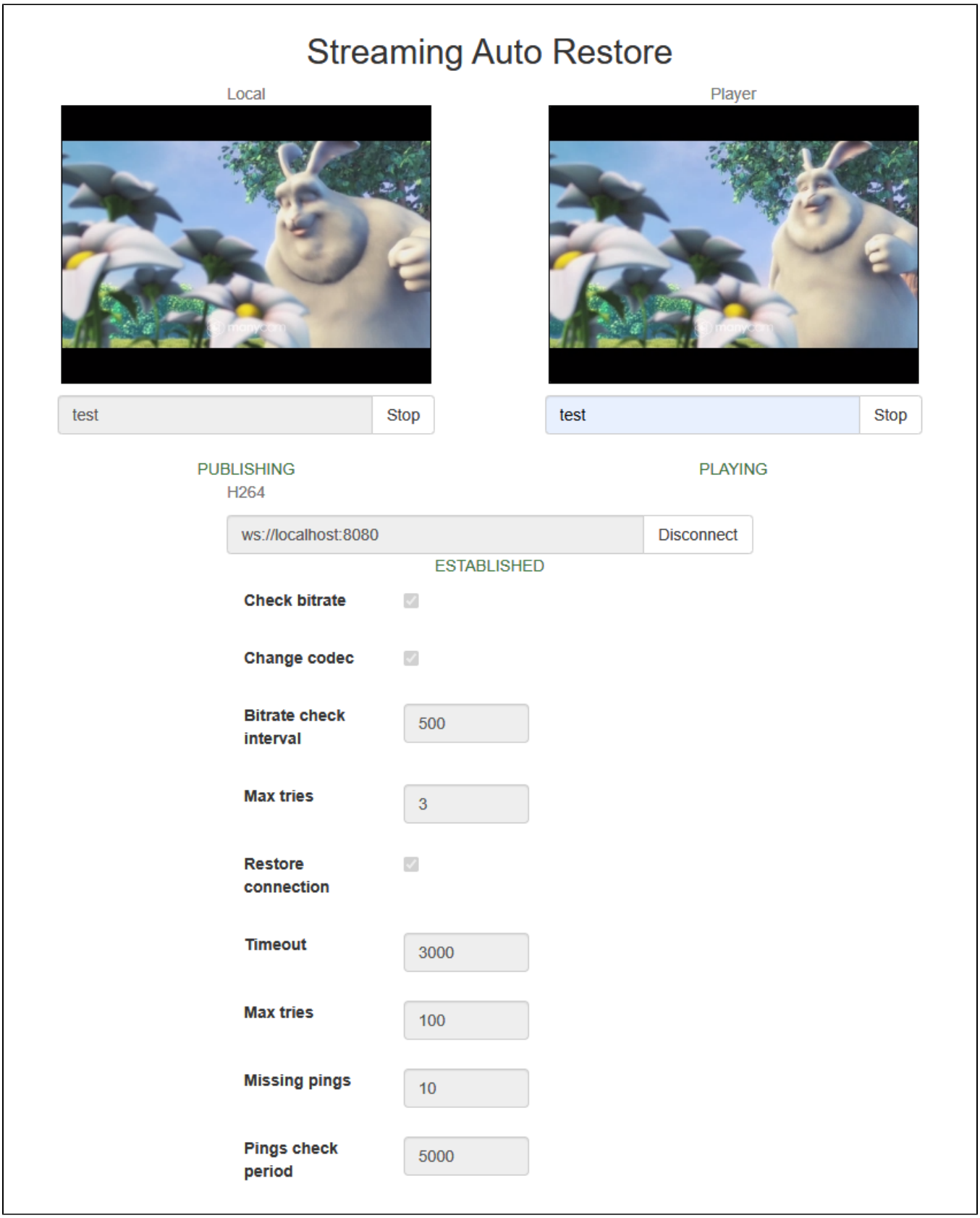
- при проблемах с публикацией H264 потока из браузера с заменой кодека на VP8
- при остановке отправки медиапакетов, с контролем по снижению битрейта публикации до 0
- при переключении с одной сети на другую (с Wi-Fi на LTE и обратно)
- разрыве и последующем восстановлении соединения (в том числе при остановке публикации играемого потока или при перезапуске сервера)

Параметры контроля битрейта

- Check bitrate - контролировать снижение битрейта публикации до 0
- Change codec - менять кодек с H264 на VP8 при остановке публикации по снижению битрейта до 0
- Bitrate check interval - интервал проверки текущего значения битрейта публикации
- Max tries - максимальное количество последовательных попыток проверки битрейта, при котором зафиксировано снижение до 0

Параметры восстановления соединения

- Restore connection - восстанавливать соединение при обрыве сессии или при остановке проигрывания потока с признаком ошибки
- Timeout - интервал между попытками восстановить соединение
- Max tries - максимальное количество попыток восстановить соединение
- Missing pings - максимальное количество ping пакетов, не полученных от сервера (0 отключает проверку)
- Pings check period - интервал проверки количества пропущенных ping пакетов



Код примера

Код данного примера находится на WCS-сервере по следующему пути:

/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/stream-auto-restore

stream-auto-restore.css - файл стилей
stream-auto-restore.html - страница клиента
stream-auto-restore.js - скрипт, обеспечивающий работу примера

Тестировать данный пример можно по следующему адресу:

https://host:8888/client2/examples/demo/streaming/stream-auto-restore/stream-auto-restore.html

Здесь host - адрес WCS-сервера.

Работа с кодом примера

Для разбора кода возьмем версию файла stream-auto-restore.js с хешем 2035db9, которая находитсяздесьи доступна для скачивания в соответствующей сборке 2.0.209.

1. Действия при открытии страницы

1.1. Инициализация API.

Flashphoner.init() code

```
Flashphoner.init();
```

1.2. Инициализация объектов для хранения текущего состояния сессии и публикуемого/играемого потоков

code

```
    currentSession = sessionState();
    streamPublishing = streamState();
    streamPlaying = streamState();
```

1.3. Инициализация объекта для контроля битрейта

code

```
    h264PublishFailureDetector = codecPublishingFailureDetector();
```

1.4. Инициализация объекта для восстановления соединения

code

объекту передается функция, которая должна выполниться при срабатывании интервального таймера восстановления соединения

```
    streamingRestarter = streamRestarter(function() {
        if (streamPublishing.wasActive) {
            onPublishRestart();
        }
        if (streamPlaying.wasActive && streamPlaying.name != streamPublishing.name) {
            onPlayRestart();
        }
    });
```

1.5. Запуск детектора изменения сети

code

```
    networkChangeDetector();
```

2. Действия при подключении к серверу/отключении от сервера

2.1. Подключение к серверу.

Flashphoner.createSession() code

При создании сессии передаются следующие параметры:

- url - Websocket URL сервера
- receiveProbes - максимальное количество ping пакетов, не полученных от сервера (0 отключает проверку)
- probesInterval - интервал проверки количества пропущенных ping пакетов (0 отключает проверку)

```
    Flashphoner.createSession({
        urlServer: url,
        receiveProbes: receiveProbes,
        probesInterval: probesInterval
    }).on(SESSION_STATUS.ESTABLISHED, function (session) {
        ...
    }).on(SESSION_STATUS.DISCONNECTED, function () {
        ...
    }).on(SESSION_STATUS.FAILED, function () {
        ...
    });
```

2.2. Получение от сервера события, подтверждающего успешное соединение.

ConnectionStatusEvent ESTABLISHEDcode

При успешной установке соединения:

- запоминаются параметры текущей сессии для последующего восстановления
- если в предыдущей сессии был опубликован или игрался поток, публикация/проигрывание перезапускается

```
    Flashphoner.createSession({
        urlServer: url,
        receiveProbes: receiveProbes,
        probesInterval: probesInterval
    }).on(SESSION_STATUS.ESTABLISHED, function (session) {
        setStatus("#connectStatus", session.status());
        currentSession.set(url, session);
        onConnected(session);
        if(restoreConnection) {
            if(streamPublishing.wasActive) {
                console.log("A stream was published before disconnection, restart publishing");
                onPublishRestart();
                return;
            }
            if(streamPlaying.wasActive) {
                console.log("A stream was played before disconnection, restart playback");
                onPlayRestart();
            }
        }
    }).on(SESSION_STATUS.DISCONNECTED, function () {
        ...
    }).on(SESSION_STATUS.FAILED, function () {
        ...
    });
```

2.3. Закрытие соединения при нажатии на кнопку Disconnect

session.disconnect()code

```
function onConnected(session) {
    $("#connectBtn").text("Disconnect").off('click').click(function () {
        $(this).prop('disabled', true);
        currentSession.isManuallyDisconnected = true;
        session.disconnect();
    }).prop('disabled', false);
    ...
}
```

2.4. Получение события при закрытии соединения

ConnectionStatusEvent DISCONNECTED code

Если соединение было закрыто вручную при нажатии на кнопку Disconnect:

- объекты состояния очищаются
- таймер восстановления соединения останавливается

```
    Flashphoner.createSession({
        urlServer: url,
        receiveProbes: receiveProbes,
        probesInterval: probesInterval
    }).on(SESSION_STATUS.ESTABLISHED, function (session) {
        ...
    }).on(SESSION_STATUS.DISCONNECTED, function () {
        setStatus("#connectStatus", SESSION_STATUS.DISCONNECTED);
        onDisconnected();
        // Prevent streaming restart if session is manually disconnected
        if (currentSession.isManuallyDisconnected) {
            streamPublishing.clear();
            streamPlaying.clear();
            streamingRestarter.reset();
            currentSession.clear();
        }
    }).on(SESSION_STATUS.FAILED, function () {
        ...
    });
```

2.5. Получение события при разрыве соединения

ConnectionStatusEvent FAILED code

Если на момент разрыва соединения публиковался или проигрывался поток, запускается таймер восстановления соединения

```
    Flashphoner.createSession({
        urlServer: url,
        receiveProbes: receiveProbes,
        probesInterval: probesInterval
    }).on(SESSION_STATUS.ESTABLISHED, function (session) {
        ...
    }).on(SESSION_STATUS.DISCONNECTED, function () {
        ...
    }).on(SESSION_STATUS.FAILED, function () {
        setStatus("#connectStatus", SESSION_STATUS.FAILED);
        onDisconnected();
        if(restoreConnection
           && (streamPublishing.wasActive || streamPlaying.wasActive)) {
            streamingRestarter.restart($("#restoreTimeout").val(), $("#restoreMaxTries").val());
        }
    });
```

3. Действия при публикации видеопотока

3.1 Публикация видеопотока

session.createStream(), publish()code

При создании передаются:

- имя видеопотока streamName
- localVideo - div-элемент, в котором будет отображаться видео с камеры
- stripCodecs - кодек, который нужно исключить, если при предыдущей публикации битрейт упал до 0

```
session.createStream({
    name: streamName,
    display: localVideo,
    cacheLocalResources: true,
    receiveVideo: false,
    receiveAudio: false,
    stripCodecs: stripCodecs
    ...
}).publish();
```

3.2. Получение от сервера события, подтверждающего успешную публикацию потока.

StreamStatusEvent PUBLISHING code

При успешной публикации:

- запускается контроль битрейта
- запоминаются текущие параметры публикуемого потока для последующего восстановления публикации
- останавливается таймер восстановления соединения
- если в предыдущей сессии игрался поток, запускается проигрывание этого потока

```
    session.createStream({
        ...
    }).on(STREAM_STATUS.PUBLISHING, function (stream) {
        setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
        onPublishing(stream);
        streamPublishing.set(streamName, stream);
        streamingRestarter.reset();
        if ($("#restoreConnection").is(':checked')
           && streamPlaying.wasActive) {
            console.log("A stream was played before, restart playback");
            onPlayRestart();
        }
    }).on(STREAM_STATUS.UNPUBLISHED, function () {
        ...
    }).on(STREAM_STATUS.FAILED, function (stream) {
        ...
    }).publish();
```

3.3. Запуск контроля битрейта при успешной публикации

code

```
function onPublishing(stream) {
    ...
    // Start publish failure detector by bitrate #WCS-3382
    if($("#checkBitrate").is(':checked')) {
        h264PublishFailureDetector.startDetection(stream, $("#bitrateInteval").val(), $("#bitrateMaxTries").
val());
    }
}
```

3.4. Остановка публикации при нажатии на Stop

stream.stop()code

```
function onPublishing(stream) {
    $("#publishBtn").text("Stop").off('click').click(function () {
        $(this).prop('disabled', true);
        streamPublishing.isManuallyStopped = true;
        stream.stop();
    }).prop('disabled', false);
    ...
}
```

3.5. Получение от сервера события, подтверждающего успешную остановку публикации потока при нажатии на Stop

StreamStatusEvent UNPUBLISHED code

При успешной остановке публикации:

- останавливается контроль битрейта
- останавливается таймер восстановления соединения
- очищается объект состояния публикуемого потока

```
    session.createStream({
        ...
    }).on(STREAM_STATUS.PUBLISHING, function (stream) {
        ...
    }).on(STREAM_STATUS.UNPUBLISHED, function () {
        setStatus("#publishStatus", STREAM_STATUS.UNPUBLISHED);
        onUnpublished();
        if (!streamPlaying.wasActive) {
            // No stream playback< we don't need restart any more
            streamingRestarter.reset();
        } else if (streamPlaying.wasActive && streamPlaying.name == streamPublishing.name) {
            // Prevent playback restart for the same stream
            streamingRestarter.reset();
        }
        streamPublishing.clear();
    }).on(STREAM_STATUS.FAILED, function (stream) {
        ...
    }).publish();
```

3.6. Получение от сервера события об остановке публикации в связи с ошибкой

StreamStatusEvent FAILED code

При остановке публикации в связи с ошибкой

- останавливается контроль битрейта
- если не зафиксирована локальная ошибка в браузере (например, по недоступности устройств захвата), запускается таймер восстановления соединения

```
    session.createStream({
        ...
    }).on(STREAM_STATUS.PUBLISHING, function (stream) {
        ...
    }).on(STREAM_STATUS.UNPUBLISHED, function () {
       ...
    }).on(STREAM_STATUS.FAILED, function (stream) {
        setStatus("#publishStatus", STREAM_STATUS.FAILED, stream);
        onUnpublished();
        if ($("#restoreConnection").is(':checked') && stream.getInfo() != ERROR_INFO.LOCAL_ERROR) {
            streamingRestarter.restart($("#restoreTimeout").val(), $("#restoreMaxTries").val());
        }
    }).publish();
```

3.7. Остановка контроля битрейта при остановке публикации

code

```
function onUnpublished() {
    ...
    h264PublishFailureDetector.stopDetection(streamPublishing.isManuallyStopped || currentSession.
isManuallyDisconnected);
    ...
}
```

4. Действия при воспроизведении видеопотока

4.1. Воспроизведение видеопотока

session.createStream(), play() code.

При создании передается имя видеопотока streamName (в том числе, это может быть имя потока, опубликованного выше), а также remoteVideo - div-элемент, в котором будет отображаться видео.

```
    session.createStream({
        name: streamName,
        display: remoteVideo
        ...
    }).play();
```

4.2. Получение от сервера события, подтверждающего успешное воспроизведение потока.

StreamStatusEvent PLAYING code

При успешном воспроизведении потока:

- запоминаются текущие параметры потока для последующего восстановления
- останавливается таймер восстановления воспроизведения

```
    session.createStream({
        name: streamName,
        display: remoteVideo
    }).on(STREAM_STATUS.PENDING, function (stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function (stream) {
        setStatus("#playStatus", stream.status());
        onPlaying(stream);
        streamingRestarter.reset();
        streamPlaying.set(streamName, stream);
    }).on(STREAM_STATUS.STOPPED, function () {
        ...
    }).on(STREAM_STATUS.FAILED, function (stream) {
        ...
    }).play();
```

4.3 Остановка воспроизведения видеопотока при нажатии на Stop

stream.stop() code

```
function onPlaying(stream) {
    $("#playBtn").text("Stop").off('click').click(function(){
        $(this).prop('disabled', true);
        stream.stop();
    }).prop('disabled', false);
    $("#playInfo").text("");
}
```

4.4. Получение от сервера события, подтверждающего успешную остановку воспроизведения потока.

StreamStatusEvent STOPPED code

При успешной остановке воспроизведения:

- останавливается таймер восстановления соединения
- очищается объект состояния играемого потока

```
session.createStream({
    name: streamName,
    display: remoteVideo
}).on(STREAM_STATUS.PENDING, function (stream) {
    ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
    ...
}).on(STREAM_STATUS.STOPPED, function () {
    setStatus("#playStatus", STREAM_STATUS.STOPPED);
    onStopped();
    streamingRestarter.reset();
    streamPlaying.clear();
}).on(STREAM_STATUS.FAILED, function (stream) {
    ...
}).play();
```

4.5. Получение события об ошибке воспроизведения потока

StreamStatusEvent FAILED code

При остановке воспроизведения в связи с ошибкой запускается таймер восстановления соединения

```
session.createStream({
    name: streamName,
    display: remoteVideo
}).on(STREAM_STATUS.PENDING, function (stream) {
    ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
    ...
}).on(STREAM_STATUS.STOPPED, function () {
    ...
}).on(STREAM_STATUS.FAILED, function (stream) {
    setStatus("#playStatus", STREAM_STATUS.FAILED, stream);
    onStopped();
    if ($("#restoreConnection").is(':checked')) {
        streamingRestarter.restart($("#restoreTimeout").val(), $("#restoreMaxTries").val());
    }
}).play();
```

5. Контроль битрейта и возобновление публикации при снижении битрейта до 0

5.1. Получение WebRTC статистики от браузера, определение текущего кодека и битрейта публикации, остановка публикации при устойчивом падении битрейта до 0

code

```
stream.getStats(function(stat) {
    let videoStats = stat.outboundStream.video;
    if(!videoStats) {
        return;
    }
    let stats_codec = videoStats.codec;
    let bytesSent = videoStats.bytesSent;
    let bitrate = (bytesSent - detector.lastBytesSent) * 8;
    if (bitrate == 0) {
        detector.counter.inc();
        console.log("Bitrate is 0 (" + detector.counter.getCurrent() + ")");
        if (detector.counter.exceeded()) {
            detector.failed = true;
            console.log("Publishing seems to be failed, stop the stream");
            stream.stop();
        }
    } else {
        detector.counter.reset();
    }
    detector.lastBytesSent = bytesSent;
    detector.codec = stats_codec;
    $("#publishInfo").text(detector.codec);
});
```

5.2. Остановка таймера проверки битрейта

code

```
if (detector.publishFailureIntervalID) {
    clearInterval(detector.publishFailureIntervalID);
    detector.publishFailureIntervalID = null;
}
```

5.3. Перезапуск публикации

code

```
if (detector.failed) {
    $("#publishInfo").text("Failed to publish " + detector.codec);
    if($("#changeCodec").is(':checked')) {
        // Try to change codec from H264 to VP8 #WCS-3382
        if (detector.codec == "H264") {
            console.log("H264 publishing seems to be failed, trying VP8 by stripping H264");
            let stripCodecs = "H264";
            publishBtnClick(stripCodecs);
        } else if (detector.codec == "VP8") {
            console.log("VP8 publishing seems to be failed, giving up");
        }
    } else {
        // Try to republish with the same codec #WCS-3410
        publishBtnClick();
    }
}
```

6. Восстановление соединения

6.1. Запуск таймера восстановления соединения

code

Таймер вызывает функцию, в которой выполняются необходимые действия по восстановлению

```
            restarter.restartTimerId = setInterval(function(){
                if (restarter.counter.exceeded()) {
                    logger.info("Tried to restart for " + restartMaxTimes + " times with " +restartTimeout + "
ms interval, cancelled");
                    restarter.reset();
                    return;
                }
                onRestart();
                restarter.counter.inc();
            }, restartTimeout);
```

6.2. Остановка таймера восстановления соединения

code

```
            if (restarter.restartTimerId) {
                clearInterval(restarter.restartTimerId);
                logger.info("Timer " + restarter.restartTimerId + " stopped");
                restarter.restartTimerId = null;
            }
            restarter.counter.reset();
```

6.3. Создание новой сессии

code

```
    let sessions = Flashphoner.getSessions();
    if (!sessions.length || sessions[0].status() == SESSION_STATUS.FAILED) {
        logger.info("Restart session to publish");
        click("connectBtn");
    } else {
        ...
    }
```

6.4. Повторная публикация

code

```
        let streams = sessions[0].getStreams();
        let stream = null;
        let clickButton = false;
        if (streams.length == 0) {
            // No streams in session, try to restart publishing
            logger.info("No streams in session, restart publishing");
            clickButton = true;
        } else {
            // If there is already a stream, check its state and restart publishing if needed
            for (let i = 0; i < streams.length; i++) {
                if (streams[i].name() === $('#publishStream').val()) {
                    stream = streams[i];
                    if (!isStreamPublishing(stream)) {
                        logger.info("Restart stream " + stream.name() + " publishing");
                        clickButton = true;
                    }
                    break;
                }
            }
            if (!stream) {
                logger.info("Restart stream publishing");
                clickButton = true;
            }
        }
        if (clickButton) {
            click("publishBtn");
        }
```

6.5. Повторное проигрывание

code

```
        let streams = sessions[0].getStreams();
        let stream = null;
        let clickButton = false;
        if (streams.length == 0) {
            // No streams in session, try to restart playing
            logger.info("No streams in session, restart playback");
            clickButton = true;
        } else {
            // If there is already a stream, check its state and restart playing if needed
            for (let i = 0; i < streams.length; i++) {
                if (streams[i].name() === $('#playStream').val()) {
                    stream = streams[i];
                    if (!isStreamPlaying(stream)) {
                        logger.info("Restart stream " + stream.name() + " playback");
                        clickButton = true;
                    }
                    break;
                }
            }
            if (!stream) {
                logger.info("Restart stream playback");
                clickButton = true;
            }
        }
        if (clickButton) {
            click("playBtn");
        }
```

7. Контроль смены сети

7.1. Обработка события, сигнализирующего о смене сети на устройстве

connection.onchange code

```
    if (Browser.isChrome() || (Browser.isFirefox() && Browser.isAndroid())) {
        connection = navigator.connection || navigator.mozConnection || navigator.webkitConnection;
        if (connection) {
            connectionType = connection.type;
            if (Browser.isFirefox()) {
                connection.ontypechange = onNetworkChange;
            } else {
                connection.onchange = onNetworkChange;
            }
        }
    }
```

7.2. Закрытие соединения при смене сети

code

```
        if (isNetworkConnected() && connection.type != connectionType) {
            if (currentSession.getStatus() == SESSION_STATUS.ESTABLISHED) {
                let logger = Flashphoner.getLogger();
                logger.info("Close session due to network change from " + connectionType + " to " + connection.
type);
                currentSession.sdkSession.disconnect();
            }
        }
```