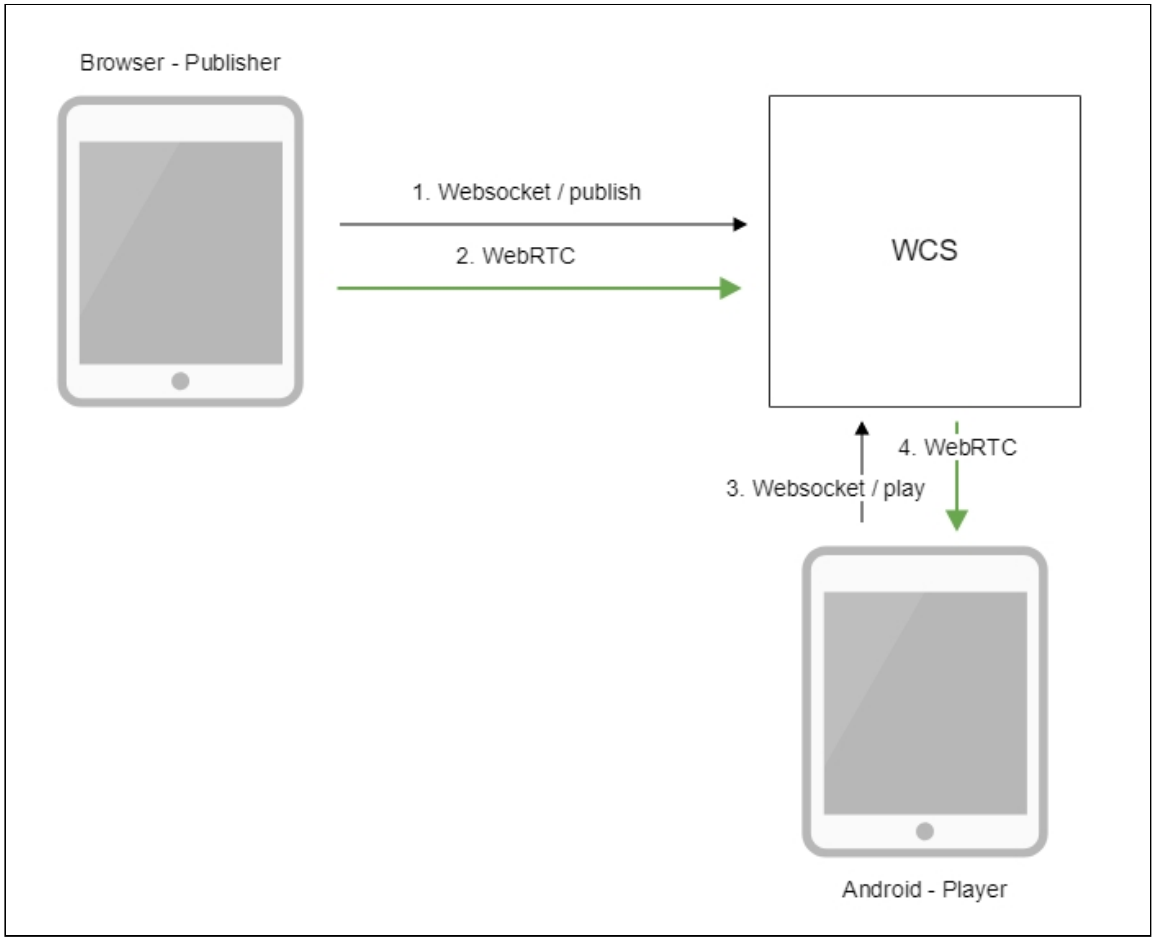
В мобильном приложении Android по WebRTC

Описание

WCS предоставляет SDK для разработки клиентских приложений на платформе Android

Схема работы

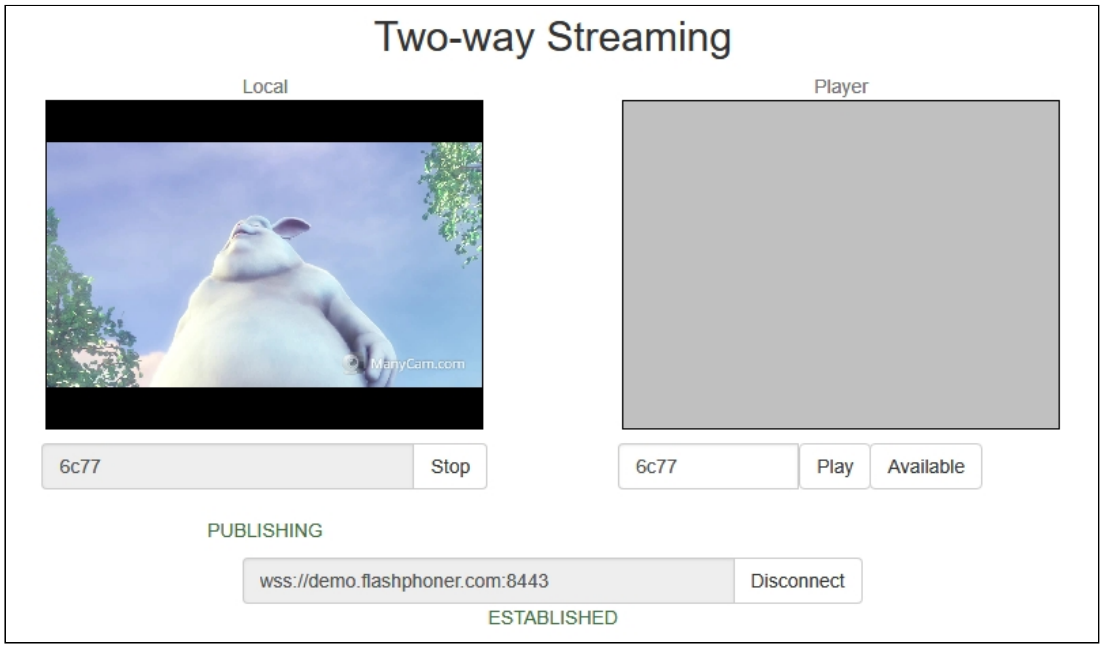


1. Браузер соединяется с сервером по протоколу Websocket и отправляет команду `publishStream`.
2. Браузер захватывает микрофон и камеру и отправляет WebRTC поток на сервер.
3. Android-устройство соединяется с сервером по протоколу Websocket и отправляет команду `playStream`.

4. Android-устройство получает WebRTC поток от сервера и воспроизводит в приложении.

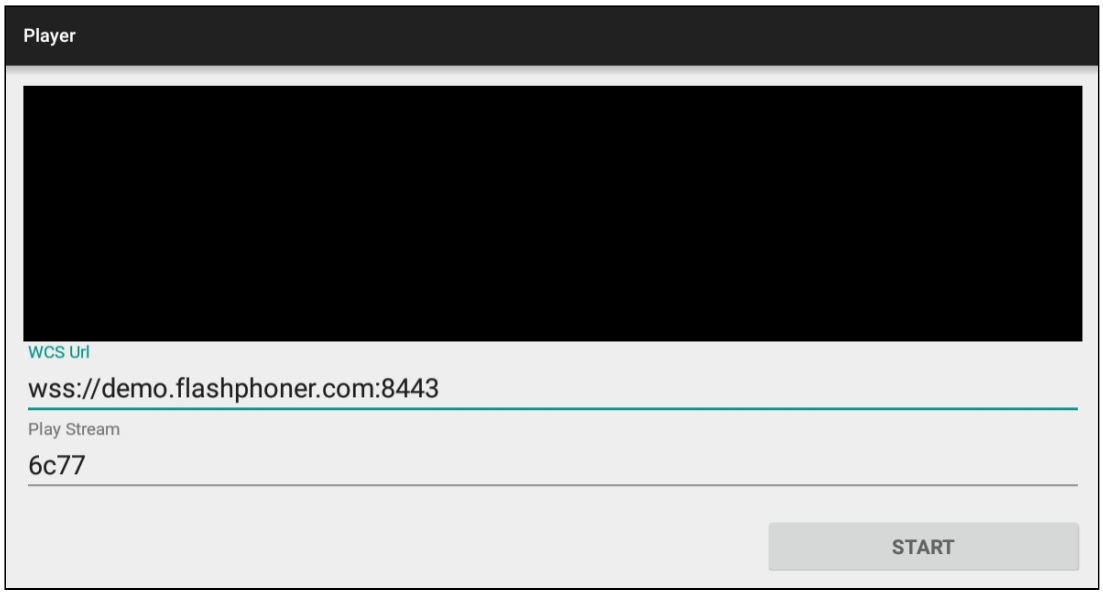
Краткое руководство по тестированию

1. Для теста используем:
2. демо-сервер `demo.flashphoner.com`;
3. веб-приложение Two Way Streaming для публикации потока;
4. мобильное приложение Player (Google Play) для воспроизведения потока
5. Откройте веб-приложение Two Way Streaming. Нажмите `Connect`, затем `Publish`. Скопируйте идентификатор потока:

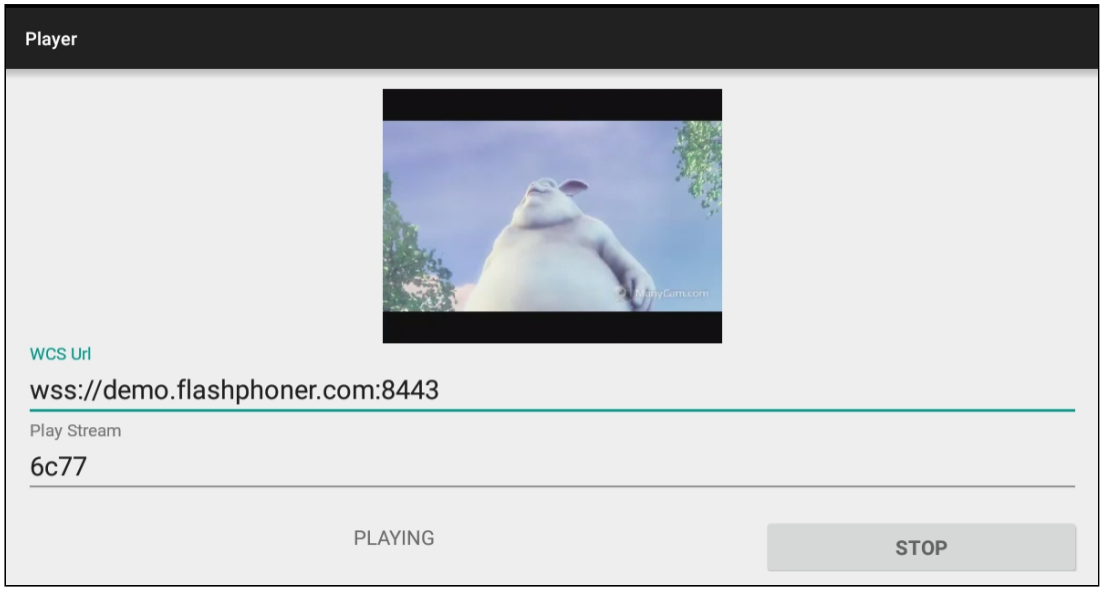


6. Установите на устройство Android мобильное приложение Player из Google Play. Запустите приложение на устройстве, введите в поле `WCS url` адрес WCS-сервера в виде `wss://demo.flashphoner.com:8443`, в поле `Play Stream` - идентификатор

видеопотока:



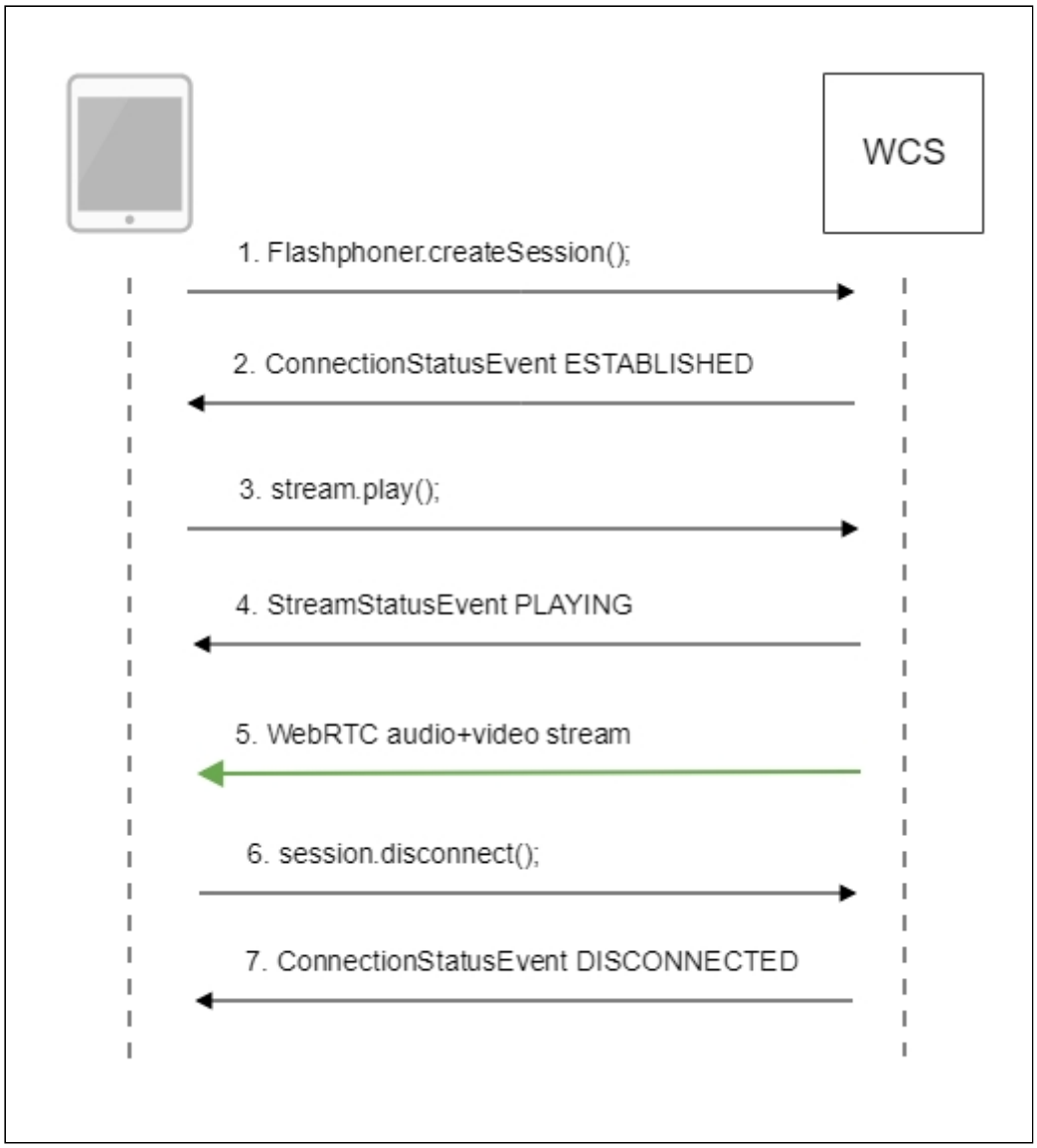
7. Нажмите кнопку `Start`. Начнется воспроизведение видеопотока:



Последовательность выполнения операций (Call flow)

Ниже описана последовательность вызовов при использовании примера Player для воспроизведения потока

PlayerActivity.java



1. Установка соединения с сервером

`Flashphoner.createSession()` code

```
/**
* The options for connection session are set.
* WCS server URL is passed when SessionOptions object is created.
* SurfaceViewRenderer to be used to display the video stream is set with
method SessionOptions.setRemoteRenderer().
*/
SessionOptions sessionOptions = new
SessionOptions(mWcsUrlView.getText().toString());
sessionOptions.setRemoteRenderer(remoteRender);

/**
* Session for connection to WCS server is created with method
createSession().
*/
session = Flashphoner.createSession(sessionOptions);
```

2. Получение от сервера события, подтверждающего успешное соединение
`Session.onConnected()` code

```
@Override
public void onConnected(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mStartButton.setText(R.string.action_stop);
            mStartButton.setTag(R.string.action_stop);
            mStartButton.setEnabled(true);
            mStatusView.setText(connection.getStatus());

            /**
             * The options for the stream to play are set.
             * The stream name is passed when StreamOptions object is
created.
             */
            StreamOptions streamOptions = new
StreamOptions(mPlayStreamView.getText().toString());

            /**
             * Stream is created with method Session.createStream().
             */
            playStream = session.createStream(streamOptions);
            ...
        }
        ...
    });
    ...
}
```

3. Запуск воспроизведения потока
`Stream.play()` code

```
/*
 * Method Stream.play() is called to start playback of the stream.
 */
playStream.play();
```

4. Получение от сервера события, подтверждающего успешное воспроизведение потока
`StreamStatus.PLAYING` code

```
/**
 * Callback function for stream status change is added to display the
status.
 */
playStream.on(new StreamStatusEvent() {
    @Override
    public void onStreamStatus(final Stream stream, final StreamStatus
streamStatus) {
        runOnUiThread(new Runnable() {
            @Override
```

```
            public void run() {
                if (!StreamStatus.PLAYING.equals(streamStatus)) {
                    Log.e(TAG, "Can not play stream " + stream.getName() +
" " + streamStatus);
                    mStatusView.setText(streamStatus.toString());
                } else if
(StreamStatus.NOT_ENOUGH_BANDWIDTH.equals(streamStatus)) {
                    Log.w(TAG, "Not enough bandwidth stream " +
stream.getName() + ", consider using lower video resolution or bitrate. "
+
                            "Bandwidth " +
(Math.round(stream.getNetworkBandwidth() / 1000)) + " " +
                            "bitrate " +
(Math.round(stream.getRemoteBitrate() / 1000)));
                } else {
                    mStatusView.setText(streamStatus.toString());
                }
            }
        });
    }
});
```

5. Прием аудио-видео потока по WebRTC
6. Остановка воспроизведения потока

   `Session.disconnect()` code

```
if (mStartButton.getTag() == null ||
Integer.valueOf(R.string.action_start).equals(mStartButton.getTag())) {
    ...
} else {
    mStartButton.setEnabled(false);

    /**
    * Connection to WCS server is closed with method Session.disconnect().
    */
    session.disconnect();
}
```

7. Получение от сервера события, подтверждающего остановку воспроизведения потока

   `Session.onDisconnection()` code

```
@Override
public void onDisconnection(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mStartButton.setText(R.string.action_start);
            mStartButton.setTag(R.string.action_start);
            mStartButton.setEnabled(true);
            mStatusView.setText(connection.getStatus());
        }
    });
}
```