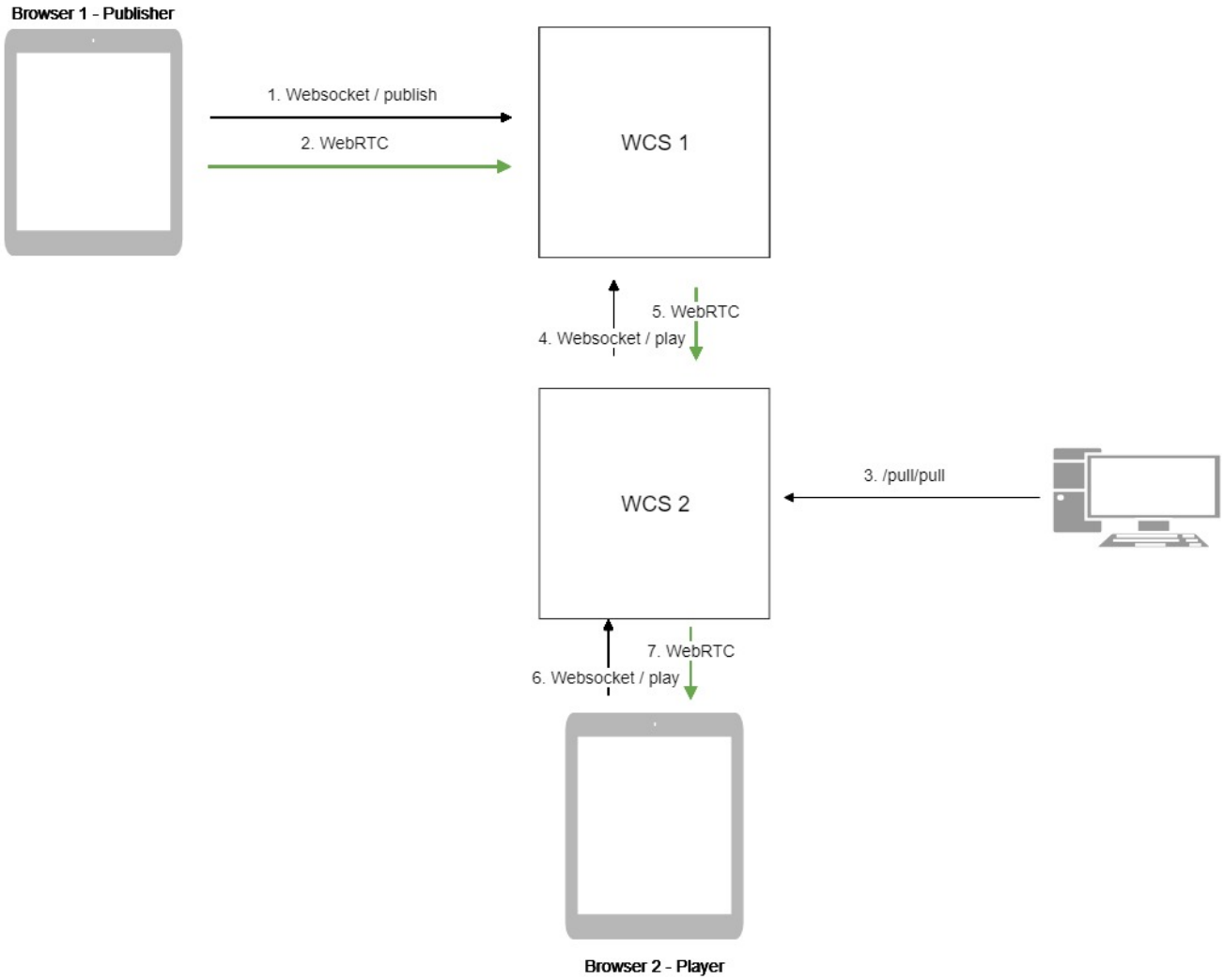
From another WCS server via WebRTC

Overview

On demand, WCS can capture a WebRTC video stream published by another WCS server. The captured stream can be than broadcast toany of supported platformsusingany of supported technologies. Managing of WebRTC stream capturing is performed usingREST API.

Operation flow chart



1. The browser connects to the WCS1 server via Websocket and sends the publish command.
2. The browser captures the microphone and the camera and sends the WebRTC stream to the server.
3. The REST client sends to the WCS2 server the /pull/pull query.
4. WCS2 requests the stream from WCS1.
5. WCS2 receives the WebRTC stream from WCS1.
6. The second browser establishes a connection to the WCS2 server via Websokcet and sends the play command.
7. The second browser receives the WebRTC stream and plays this stream on the page.

REST-queries

REST-query must be an HTTP/HTTPS POST request as follows:

- - HTTP:http://test.flashphoner.com:8081/rest-api/pull/rtmp/pull
- - HTTPS:https://test.flashphoner.com:8444/rest-api/pull/rtmp/pull

Where:

- -test.flashphoner.com- is the address of the WCS server
- - 8081- is the standard REST / HTTP port of the WCS server
- -8444- is the standard HTTPS port
- -rest-api- is the required part of the URL
- - /pull/rtmp/pull- is the REST method used

REST-methods and response statuses

REST-method	Example of REST-query	Example of REST response	Response status	Description

/pull/pull	<pre>{
	"uri":"wss://demo.
	flashphoner.com:8443",
	"localStreamName":
	"testStream",
	"remoteStreamName":
	"testStream"
	}</pre>
			409 - Conflict
			
			500 - Internal error
				Pull the WebRTC stream at the specified URL

/pull
/find_all
		<pre>{
		"localMediaSessionId": "5a072377-
		73c1-4caf-abd3",
		"remoteMediaSessionId": null,
		"localStreamName": "testStream",
		"remoteStreamName": "testStream",
		"uri": "wss://demo.flashphoner.
		com:8443",
		"status": "NEW"</pre>
			200 – streams are found
			
			500 - Internal error
				Find all pulled WebRTC streams

/pull
/terminate
	<pre>{
	"uri":"wss://demo.
	flashphoner.com:8443",
	"localStreamName":
	"testStream",
	"remoteStreamName":
	"testStream"
	}</pre>
			200 - stream terminated
			
			500 - Internal error
				Terminate the pulled WebRTC stream

Parameters

Parameter name	Description	Example
uri	Websocket URL of WCS server	wss://demo.flashphoner.com:8443
localMediaSessionId	Session identifier	5a072377-73c1-4caf-abd3
remoteMediaSessionId	Session identifier on the remote server	12345678-abcd-dead-beaf
localStreamName	Local name assigned to the captured stream. By this name the stream can be requested from the WCS server	testStream
remoteStreamName	Captured stream name on the remote server	testStream

status	Current stream status	NEW

Configuration

By default, WebRTC stream is pulled over unsecure Websocket connection, i.e. WCS server URL has to be set as ws://demo.flashphoner.com:8080. To use Secure Websocket, the parameter must be set in file flashphoner.properties

```
wcs_agent_ssl=true
```

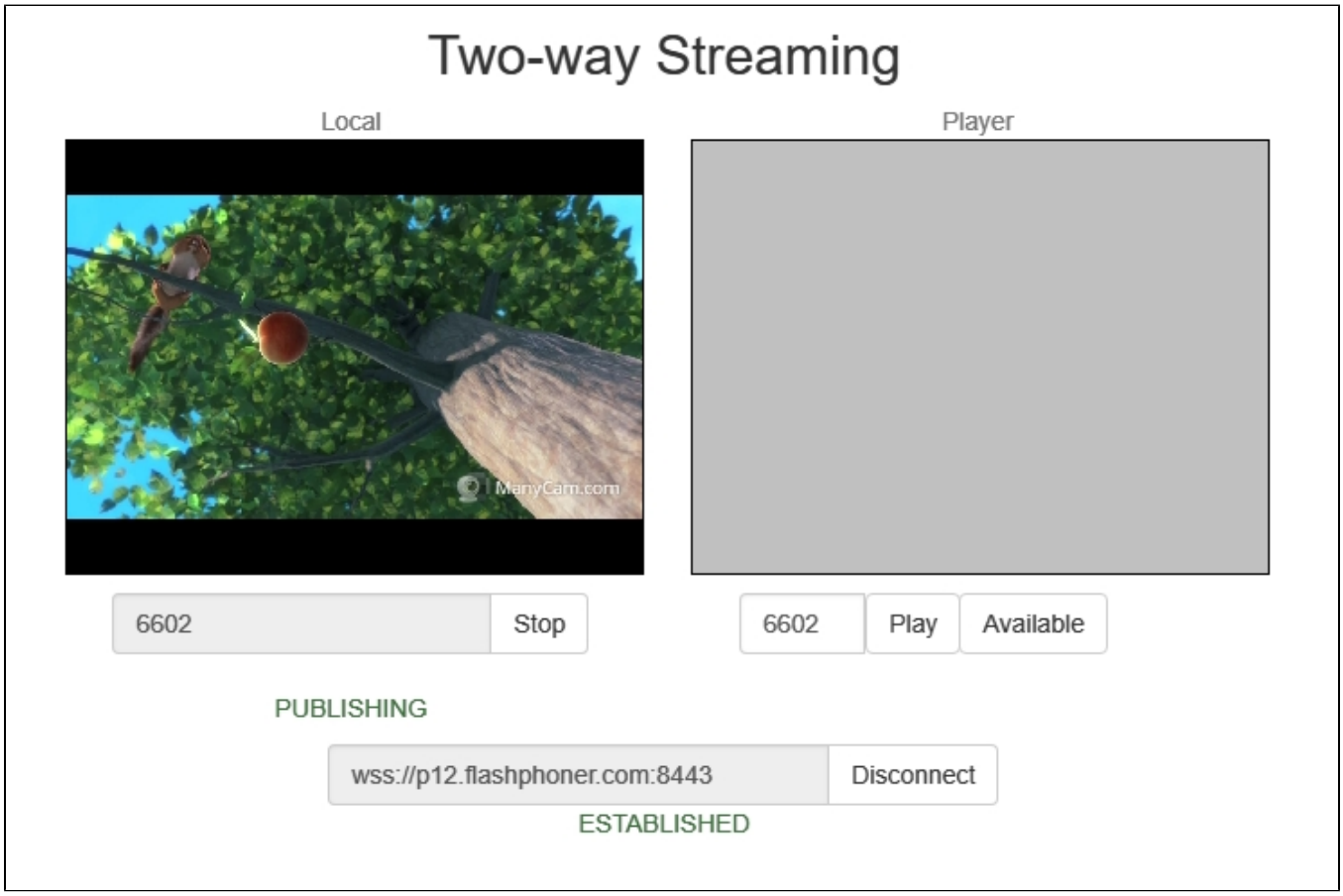
This change has to be made on both WCS servers: the server that publishes the stream and the server the stream is pulled to.

Quick manual on testing

1. For this test we use:

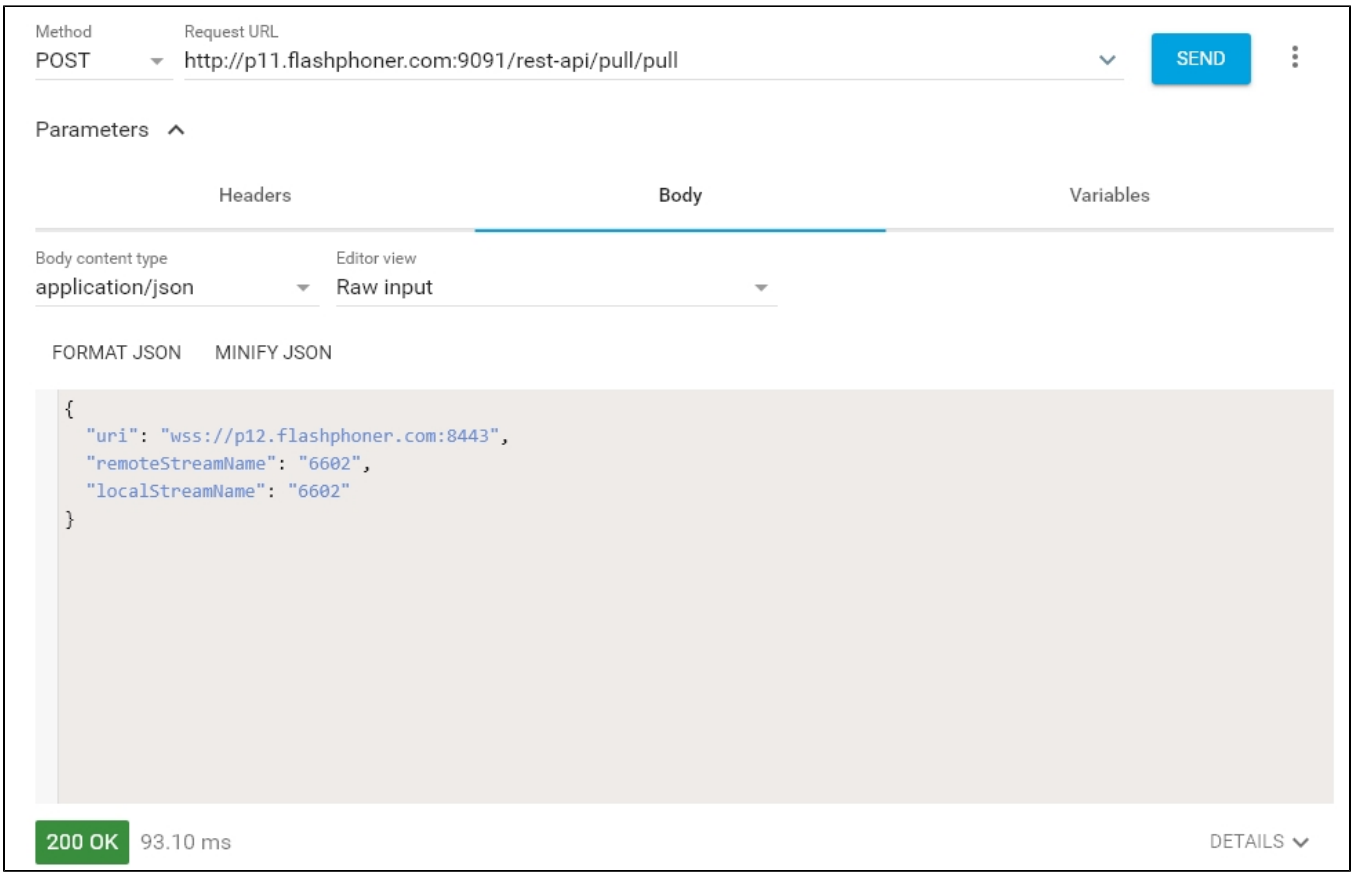
- two WCS servers;
- the Chrome browser and aREST-clientto send queries to the server;
- theTwo Way Streamingweb application to publish the stream;
- thePlayerweb application to play the captured stream in the browser.

2. Open the Two Way Streaming web application and publish the stream on the server

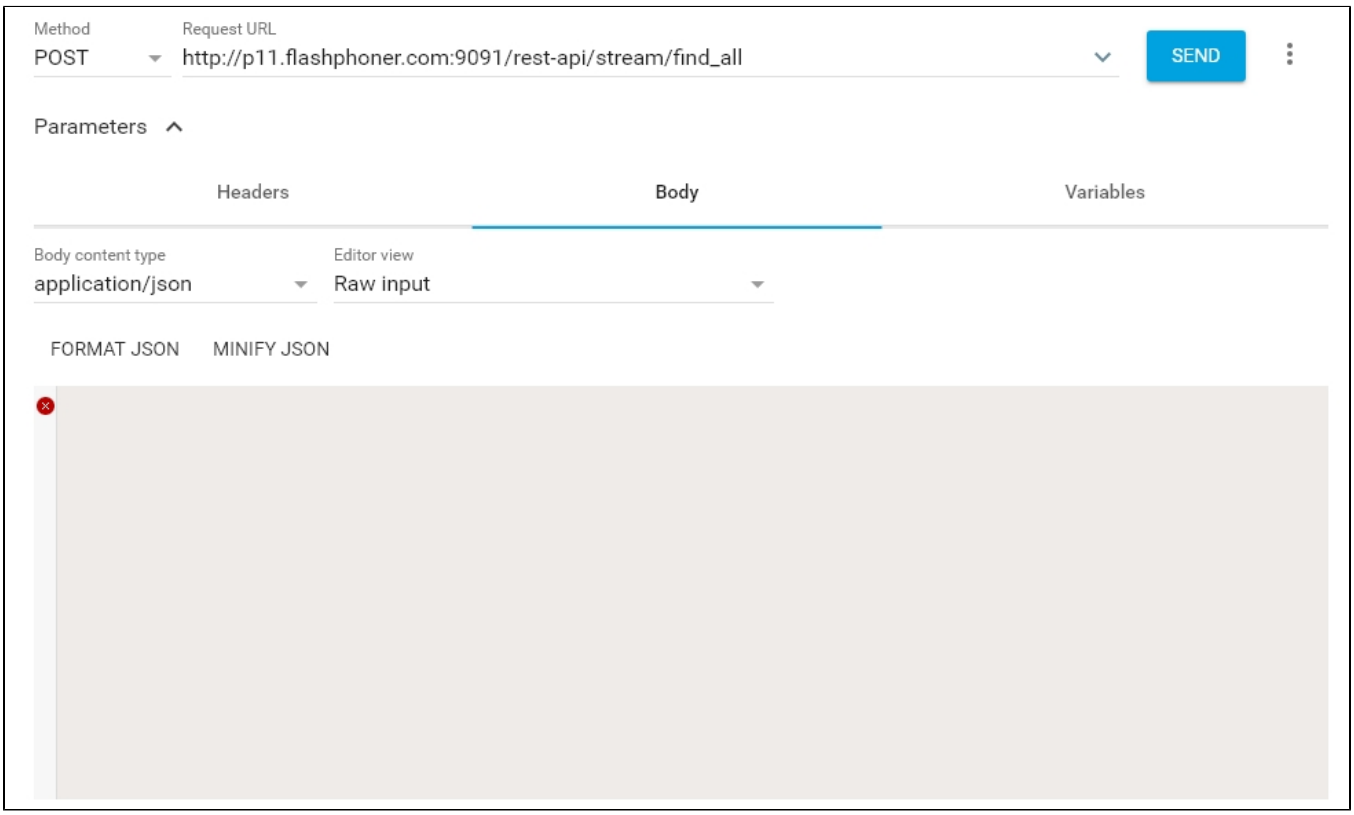


3. Open the REST client. Send the /pull/pull query and specify the following parameters:

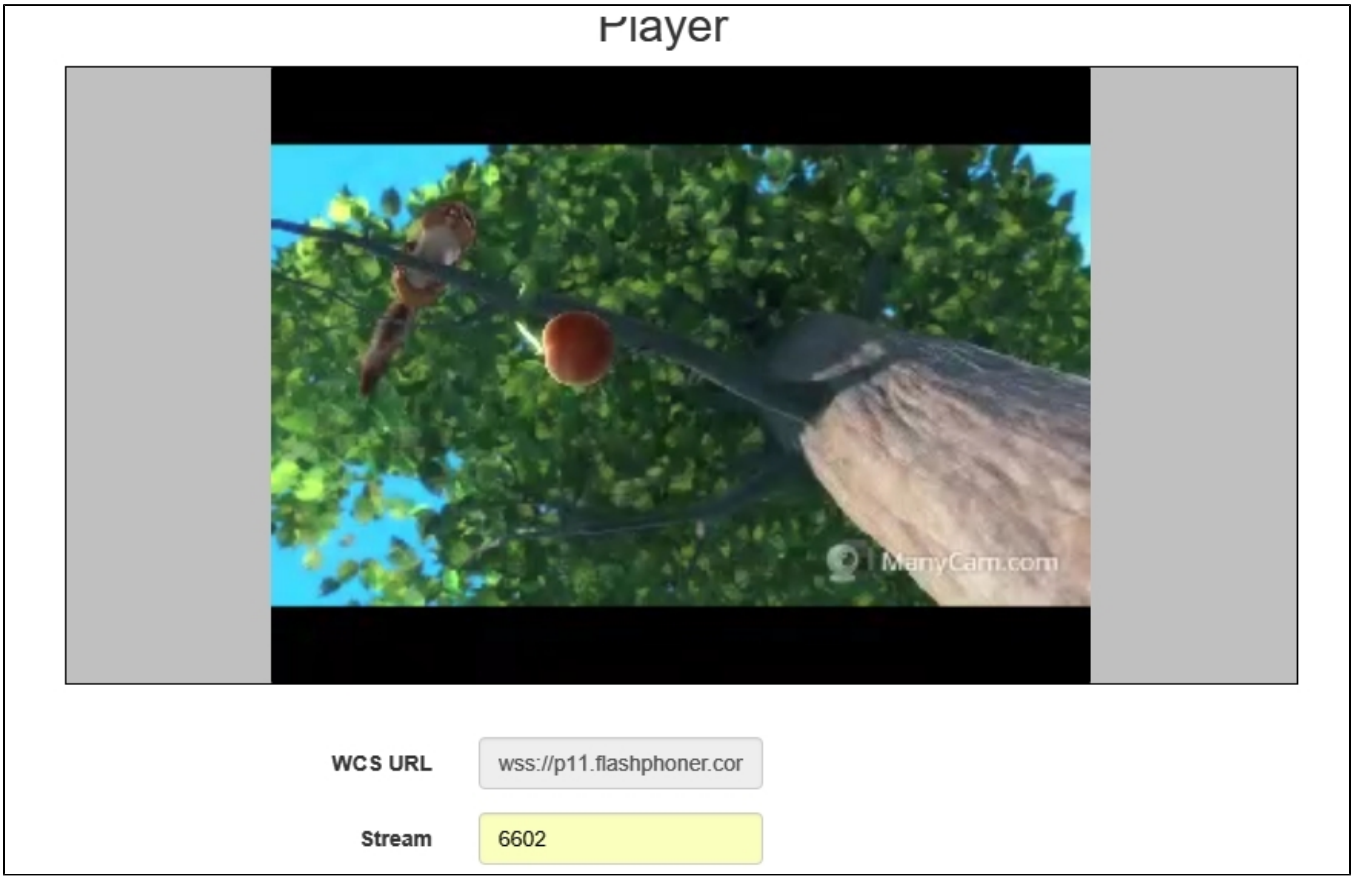
- URL of the WCS server the stream is captured from;
- stream name published on the server
- local stream name



4. Make sure the server captured the stream. To do this, send the /pull/find_all query:



5. Open the Player web application and put in the local stream name into the Stream field, then click Start



Call flow

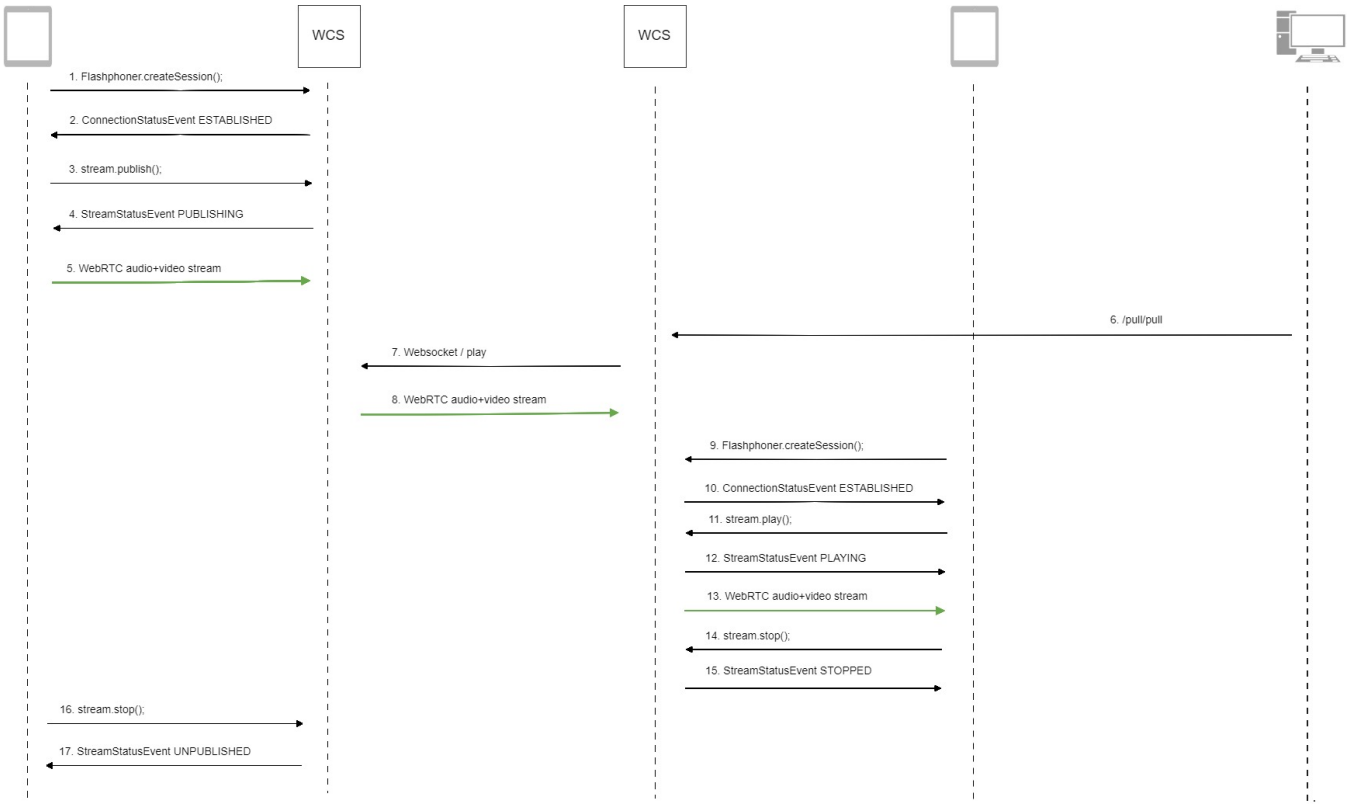
Below is the call flow when using the Two Way Streaming example to publish a stream on one WCS server and playing that stream on another WCS server

two_way_streaming.html

two_way_streaming.js

player.html

player.js



1. Establishing connection to the server.

Flashphoner.createSession();code

```
    Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function (session) {
        setStatus("#connectStatus", session.status());
        onConnected(session);
    }).on(SESSION_STATUS.DISCONNECTED, function () {
        setStatus("#connectStatus", SESSION_STATUS.DISCONNECTED);
        onDisconnected();
    }).on(SESSION_STATUS.FAILED, function () {
        setStatus("#connectStatus", SESSION_STATUS.FAILED);
        onDisconnected();
    });
```

2. Receiving from the server an event confirming successful connection.

ConnectionStatusEvent ESTABLISHEDcode

```
    Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function (session) {
        setStatus("#connectStatus", session.status());
        onConnected(session);
    }).on(SESSION_STATUS.DISCONNECTED, function () {
        ...
    }).on(SESSION_STATUS.FAILED, function () {
        ...
    });
```

3. Publishing the stream.

stream.publish();code

```
session.createStream({
    name: streamName,
    display: localVideo,
    cacheLocalResources: true,
    receiveVideo: false,
    receiveAudio: false
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
    ...
}).on(STREAM_STATUS.UNPUBLISHED, function () {
    ...
}).on(STREAM_STATUS.FAILED, function () {
    ...
}).publish();
```

4. Receiving from the server an event confirming successful publishing of the stream.

StreamStatusEvent, status PUBLISHINGcode

```
session.createStream({
    name: streamName,
    display: localVideo,
    cacheLocalResources: true,
    receiveVideo: false,
    receiveAudio: false
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
    setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
    onPublishing(stream);
}).on(STREAM_STATUS.UNPUBLISHED, function () {
    ...
}).on(STREAM_STATUS.FAILED, function () {
    ...
}).publish();
```

5. Sending the audio- video stream via WebRTC to the server

6. Sending the /pull/pull REST query to the second server

7. Requesting the stream from the first server

8. Sending the audio- video stream via WebRTC to the second server

9. Establishing connection to the second server.

Flashphoner.createSession();code

```
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function(session){
    setStatus(session.status());
    //session connected, start playback
    playStream(session);
}).on(SESSION_STATUS.DISCONNECTED, function(){
    setStatus(SESSION_STATUS.DISCONNECTED);
    onStopped();
}).on(SESSION_STATUS.FAILED, function(){
    setStatus(SESSION_STATUS.FAILED);
    onStopped();
});
```

10. Receiving from the server and event confirming successful connection.

ConnectionStatusEvent ESTABLISHEDcode

```
    Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function(session){
        setStatus(session.status());
        //session connected, start playback
        playStream(session);
    }).on(SESSION_STATUS.DISCONNECTED, function(){
        ...
    }).on(SESSION_STATUS.FAILED, function(){
        ...
    });
```

11. Requesting to play the stream.

stream.play();code

```
    stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
        var video = document.getElementById(stream.id());
        if (!video.hasListeners) {
            video.hasListeners = true;
            video.addEventListener('playing', function () {
                $("#preloader").hide();
            });
            video.addEventListener('resize', function (event) {
                var streamResolution = stream.videoResolution();
                if (Object.keys(streamResolution).length === 0) {
                    resizeVideo(event.target);
                } else {
                    // Change aspect ratio to prevent video stretching
                    var ratio = streamResolution.width / streamResolution.height;
                    var newHeight = Math.floor(options.playWidth / ratio);
                    resizeVideo(event.target, options.playWidth, newHeight);
                }
            });
        }
        ...
    });
    stream.play();
```

12. Receiving from the server an event confirming successful capturing and playing of the stream.

StreamStatusEvent, status PLAYINGcode

```
    stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function(stream) {
        $("#preloader").show();
        setStatus(stream.status());
        onStarted(stream);
    }).on(STREAM_STATUS.STOPPED, function() {
        ...
    }).on(STREAM_STATUS.FAILED, function(stream) {
        ...
    }).on(STREAM_STATUS.NOT_ENOUGH_BANDWIDTH, function(stream){
        ...
    });
    stream.play();
```

13. Sending the audio- video stream via WebRTC

14. Stopping playback of the stream

stream.stop();code

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

15. Receiving from the server an event confirming successful unpublishing of the stream.

StreamStatusEvent, status STOPPEDcode

```
    stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function(stream) {
        ...
    }).on(STREAM_STATUS.STOPPED, function() {
        setStatus(STREAM_STATUS.STOPPED);
        onStopped();
    }).on(STREAM_STATUS.FAILED, function(stream) {
        ...
    }).on(STREAM_STATUS.NOT_ENOUGH_BANDWIDTH, function(stream){
        ...
    });
    stream.play();
```

16. Stopping publishing the stream.

stream.stop();code

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

17. Receiving from the server an event confirming successful unpublishing of the stream.

StreamStatusEvent, status UNPUBLISHEDcode

```
   session.createStream({
        name: streamName,
        display: localVideo,
        cacheLocalResources: true,
        receiveVideo: false,
        receiveAudio: false
    }).on(STREAM_STATUS.PUBLISHING, function (stream) {
        ...
    }).on(STREAM_STATUS.UNPUBLISHED, function () {
        setStatus("#publishStatus", STREAM_STATUS.UNPUBLISHED);
        onUnpublished();
    }).on(STREAM_STATUS.FAILED, function () {
        ...
    }).publish();
```