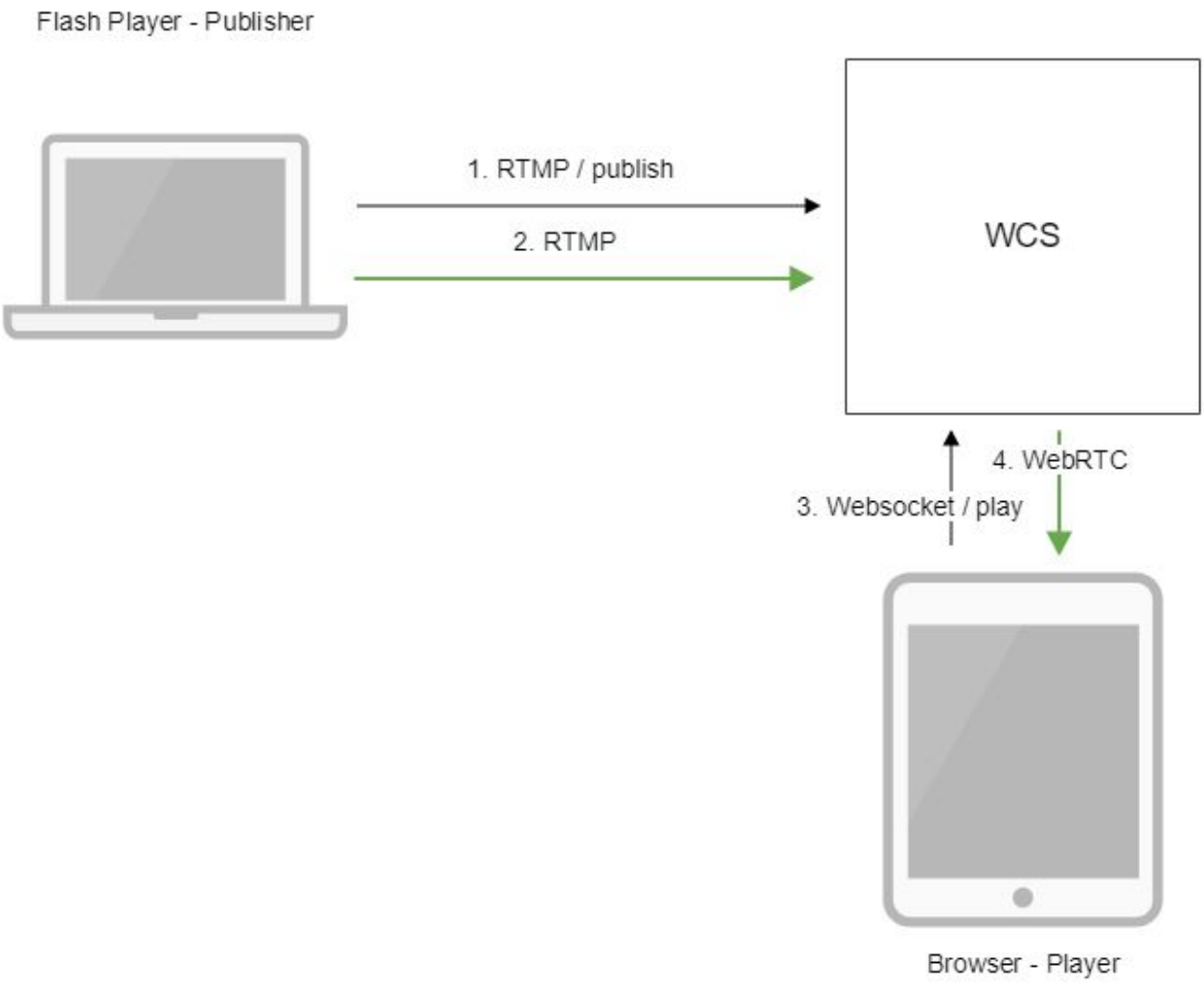
By means of Flash Player via RTMP

Overview

Supported platforms

	Adobe Flash
Windows	+
Mac OS	+
Linux	+

Operation flowchart



1. Flash Player connects to the server via the RTMP protocol and sends the publish command.

2. Flash Player captures the microphone and the camera and sends the RTMP stream to the server.

3. The browser establishes a connection via Websocket and send the play command.

4. The browser receives the WebRTC stream and plays that stream on the page.

Quick manual on testing

Capturing a video stream from the web camera and preparing to publishing

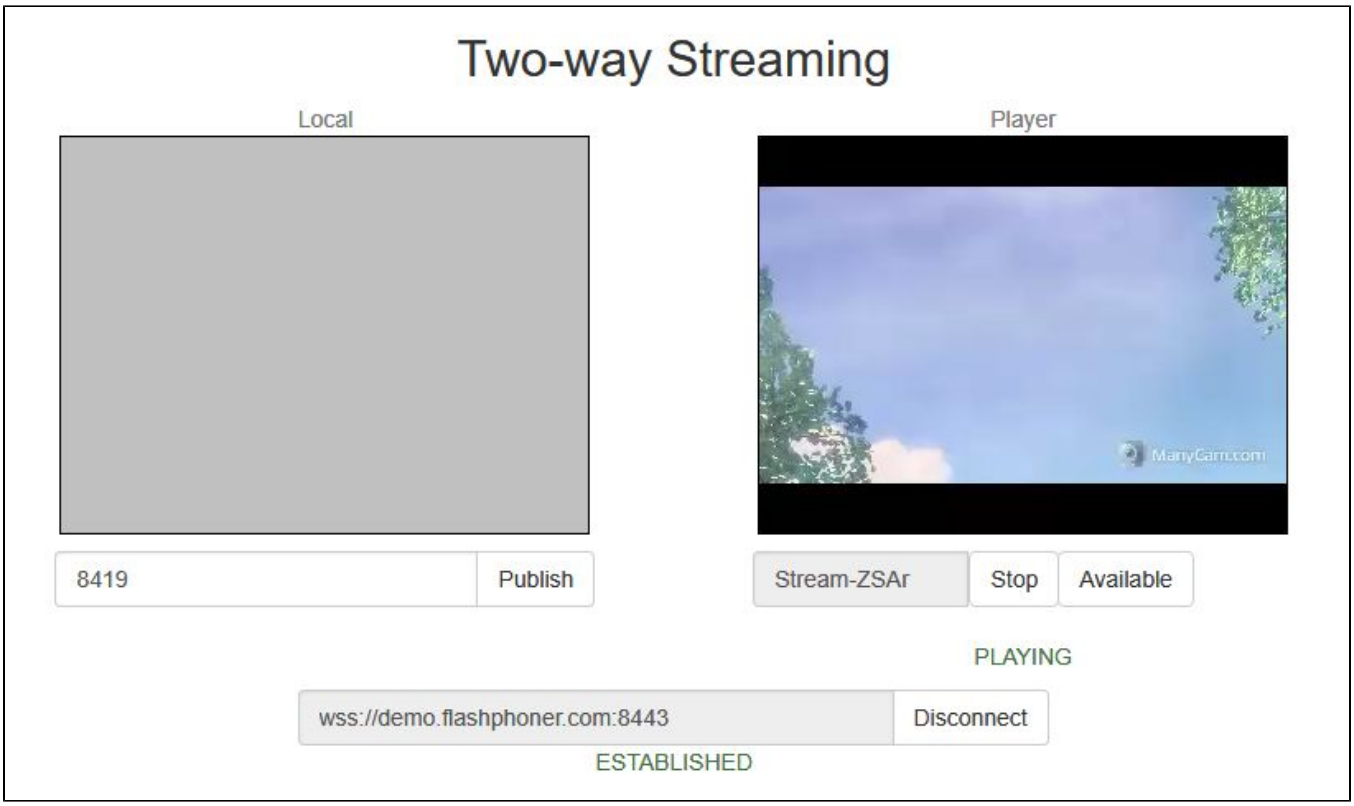
1. For this test we use the demo server at demo.flashphoner.com and the Flash Streaming web application in the Internet Explorer browser

https://demo.flashphoner.com/client2/examples/demo/streaming/flash_client/streaming.html

Install Flash Player. Open the page of the web application and allow running Flash in a browser:

2. Click the "Login" button. When the "Connected" label appears, click the Start button next to the Publish field:

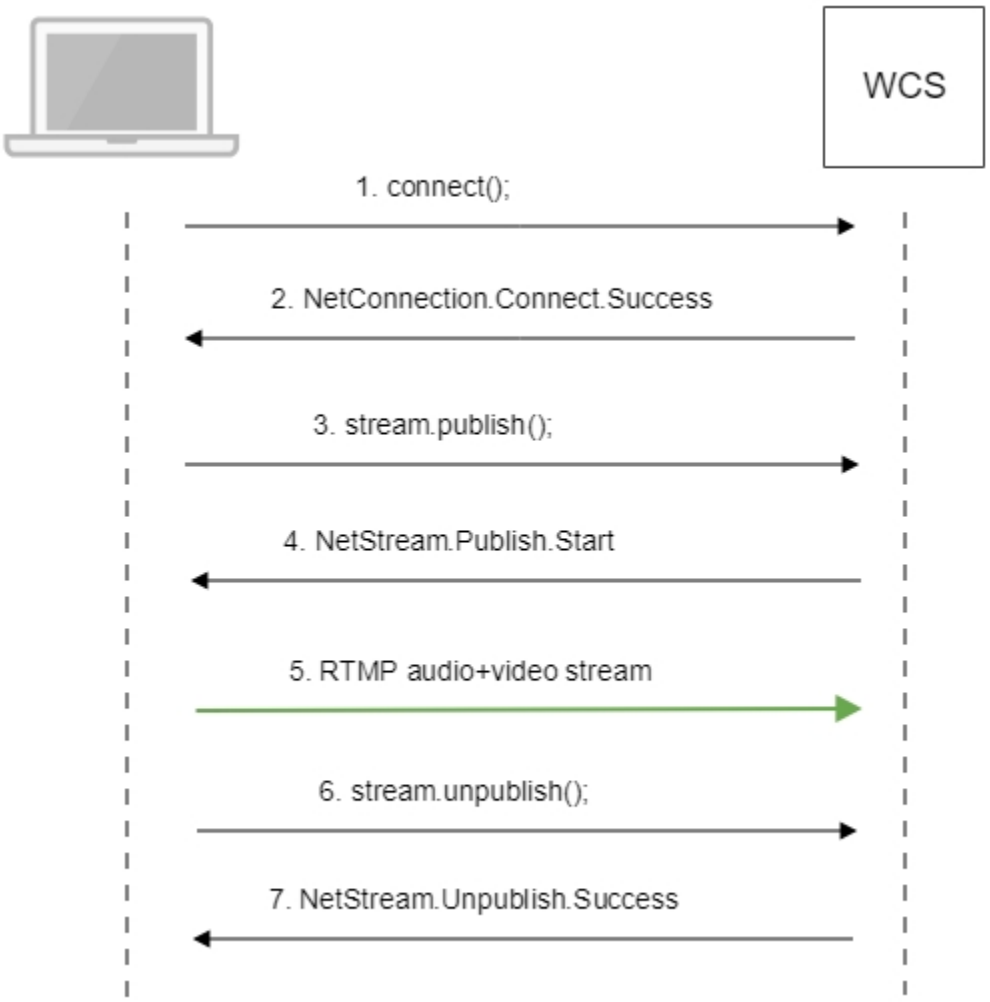
3. To make sure the broadcasting runs properly, open the Two Way Streaming application in a new window, click Connect and specify the stream identifier, then click Play



Call flow

Below is the call flow when using the Flash Streaming example

streaming.mxml



1. Establishing a connection to the server.

connect();code

```
        private function connect():void{
                var url:String = StringUtil.trim(connectUrl.text);
                Logger.info("connect " + url);
                nc = new NetConnection();
                //if (url.indexOf("rtmp") == 0){
                //        nc.objectEncoding = ObjectEncoding.AMF0;
                //}
                nc.client = this;
                nc.addEventListener(NetStatusEvent.NET_STATUS,
handleConnectionStatus);
                var obj:Object = new Object();
                obj.login = generateRandomString(20);
                obj.appKey  = "flashStreamingApp";
                nc.connect(url,obj);
        }
```

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

NetConnection.Connect.Successcode

```
        private function handleConnectionStatus(event:NetStatusEvent):void{
                Logger.info("handleConnectionStatus: "+event.info.code);
                if (event.info.code=="NetConnection.Connect.Success"){
                        Logger.info("near id: "+nc.nearID);
                        Logger.info("far id: "+nc.farID);
                        Logger.info("Connection opened");
                        disconnectBtn.visible = true;
                        connectBtn.visible = false;
                        playBtn.enabled = true;
                        publishBtn.enabled = true;
                        setConnectionStatus("CONNECTED");
                } else if (event.info.code=="NetConnection.Connect.Closed" || event.info.code=="NetConnection.
Connect.Failed"){
            ...
                }
        }
```

3. Publishing the stream.

stream.publish();code

```
        private function addListenerAndPublish():void{
                publishStream.videoReliable=true;
                publishStream.audioReliable=false;
                publishStream.useHardwareDecoder=true;
                publishStream.addEventListener(NetStatusEvent.NET_STATUS, handleStreamStatus);
                publishStream.bufferTime=0;
                publishStream.publish(publishStreamName.text);
        }
```

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

NetStream.Publish.Startcode

```
        private function handleStreamStatus(event:NetStatusEvent):void{
                Logger.info("handleStreamStatus: "+event.info.code);
                switch (event.info.code) {
            ...
                        case "NetStream.Publish.Start":
                                setPublishStatus("PUBLISHING");
                                publishBtn.visible = false;
                                unpublishBtn.visible = true;
                                break;
                }
        }
```

5. Sending the audio-video stream via RTMP

6. Stopping publishing of the stream.

stream.unpublish();code

```
        private function unpublish():void{
                Logger.info("unpublish");
                if (publishStream!=null){
                        publishStream.close();
                }
                videoFarEnd.clear();
        }
```

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

NetStream.Unpublish.Successcode

```
        private function handleStreamStatus(event:NetStatusEvent):void{
                Logger.info("handleStreamStatus: "+event.info.code);
                switch (event.info.code) {
            ...
                        case "NetStream.Unpublish.Success":
                                publishStream.removeEventListener(NetStatusEvent.NET_STATUS,
handleStreamStatus);
                                publishStream=null;
                                setPublishStatus("UNPUBLISHED");
                                publishBtn.visible = true;
                                unpublishBtn.visible = false;
                                break;
            ...
                }
        }
```

Setting a server application while RTMP stream publishing

While publishing RTMP stream to WCS server, a server application can be set that will be used to backend server interaction. It can be done with parameter in stream URL:

```
rtmp://host:1935/live?appKey=key1/streamName
```

Where

- host is WCS server;
- key1 is application key on WCS server;
- streamName is stream name to publish

By default, if application key parameter is not set, the standard application`flashStreamingApp`will be used.

Besides, an application can be explicitly specified as stream URL part. To do this, the following parameterinflashphoner.propertiesfile should be set

```
rtmp_appkey_source=app
```

Then application key must be set in stream URL as

```
rtmp://host:1935/key1/streamName
```

In this case, `live`is also an application name, therefore when stream is published with URL

```
rtmp://host:1935/live/streamName
```

`live` application must be defined on WCS server.

Known issues

1. When audio only stream is published, and this stream is played in browser via WebRTC, no sound is played.

Symptoms: there is no sound when playing a stream published with Flash client.

Solution: change SDP setting for the streams published from Flash clients in file flash_handler_publish.sdpto be audio only.

```
v=0
o=- 1988962254 1988962254 IN IP4 0.0.0.0
c=IN IP4 0.0.0.0
t=0 0
a=sdplang:en
m=audio 0 RTP/AVP 97 8 0
a=rtpmap:97 SPEEX/16000
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=sendonly
```

2. When RTMP stream is published with Flash Streaming, then it is played in iOS Safari browser via WebRTC, and another stream is published form iOS Safari via WebRTC, sound stops playing in RTMP stream.

Symptoms:

a) The stream1 stream is published from Flash Streaming web application in Chrome browser on Windows
b) The stream1 stream is played in Two Way Streaming web application in iOS Safari browser. Sound and video play normally.
c) The stream2 stream is published from Two Way Streaming web application in iOS Safari browser.Sound stops playing.
d) Stop publishing stream in iOS Safari. Sound of stream1 plays again.

Solution: switch Avoid Transcoding Alhorithm off on the server using the following parameter in flashphoner.properties file

```
disable_rtc_avoid_transcoding_alg=true
```

3. Parsing stream URL parameters does not work for streams published from Flash clients/