CDN 1.0

CDN based on WCS servers can be organized in two ways:

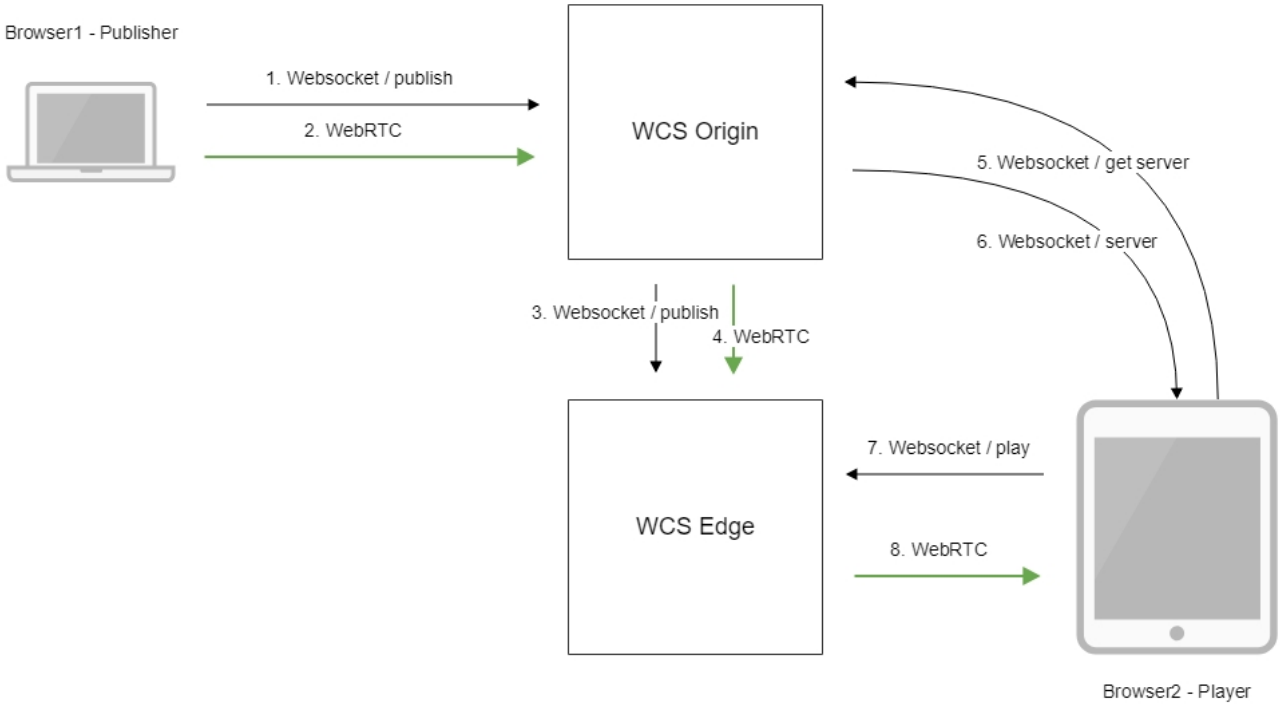
1. Static CDN with pre-configured nodes. Changing CDN configuration requires restarting the server(s), because in this scheme the server(s) are the source of streams in the network. Such CDN can be organized based on the load balancing function.
2. Dynamic CDN with dynamically changing nodes. To include/exclude a node to/from such CDN, restarting of this node only is required.

In this section we describe static CDN based on the load balancer.

Overview

Let's take a look at the simplest configuration that includes one server to publish the stream (Origin) and one to play it (Edge)

Operation flowchart



1. The browser connects to the Origin server via the Websocket protocol and sends the publish command.
2. The browser captures the microphone and the camera and sends the WebRTC stream to the server.
3. The Origin server connects to the Edge server via the Websocket protocol and sends the publish command.
4. The Origin server sends the WebRTC stream to the Edge server.
5. The second browser requests the stream from the Origin server.
6. The Origin server returns to the browser the address of the Edge server where the stream can be fetched.
7. The second browser establishes a connection to the Edge server via Websocket and sends the play command.
8. The second browser receives the WebRTC stream and plays this stream on the page.

Configuration

1. In the loadbalancing.xml settings file of the Origin server you need to specify the way to broadcast the stream and the server that will receive the broadcast:

```
<loadbalancer stream_distribution="webrtc">
        <node id="1">
                <ip>edge.flashphoner.com</ip>
        </node>
</loadbalancer>
```

Here:

- stream_distribution="webrtc" specifies that broadcasting is done via WebRTC (the other possible alternative is RTMP)
- edge.flashphoner.com - is the address of the Edge server that will receive the stream

2. In the flashphoner.properties settings file enable the load balancing mode:

```
load_balancing_enabled=true
```

Quick manual on testing

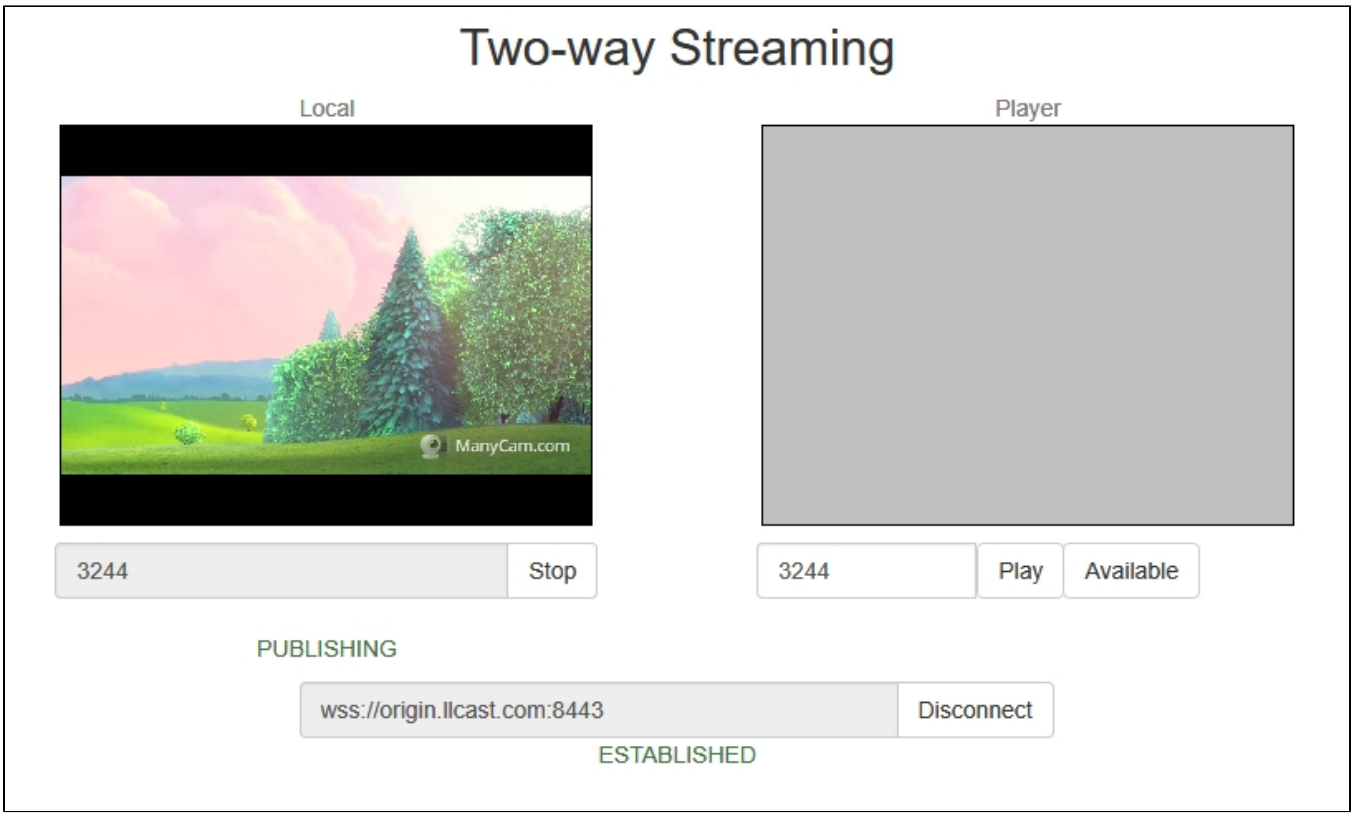
Running a broadcast via WebRTC on the Origin server

1. For this test we use the demo server atorigin.llcast.comand the Two Way Streaming web application

https://origin.llcast.com:8888/client2/examples/demo/streaming/two_way_streaming/two_way_streaming.html

2. Establish a connection to the server using the Connect button.

3. Click Publish. The browser captures the camera and sends the stream to the server.

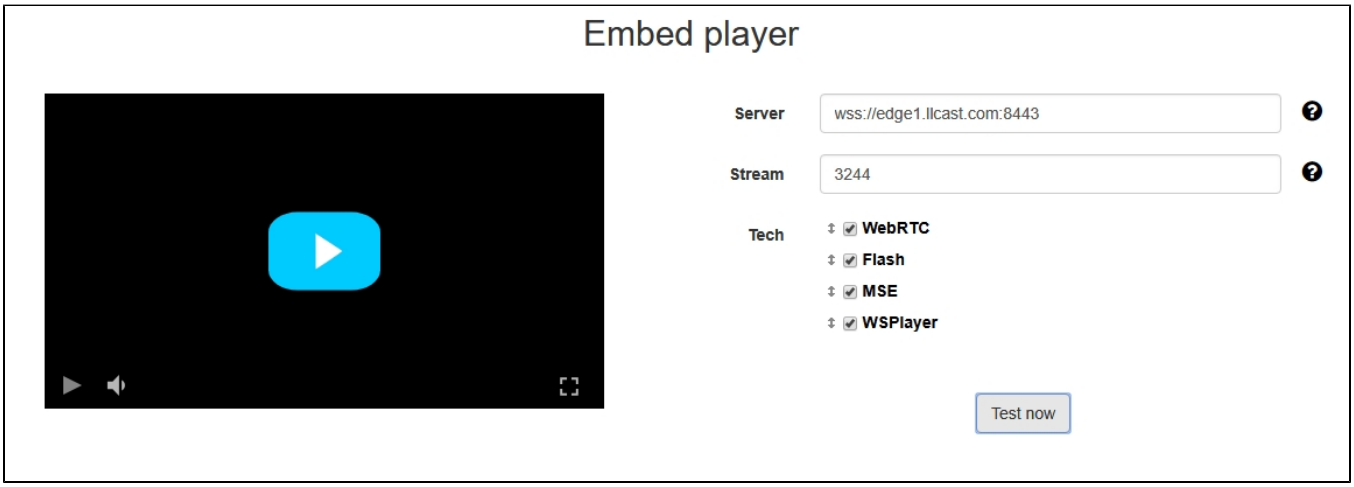


Playing the stream on the Edge server

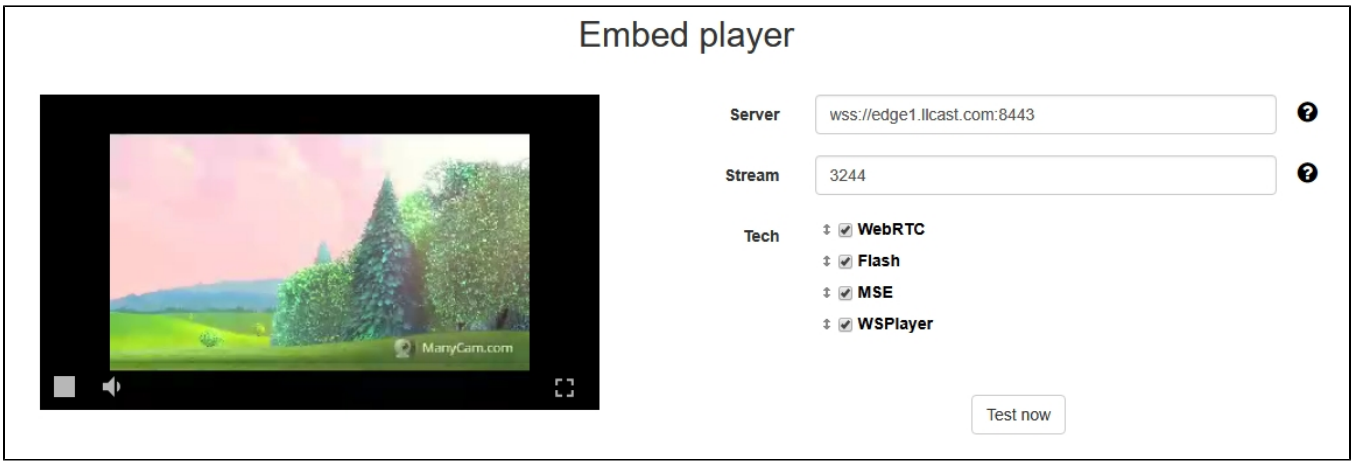
4. For the test we use the demo server at edge1.llcast.com and the Embed Player web application.

https://edge1.llcast.com:8888/client2/examples/demo/streaming/embed_player/sample.html

5. In the "Stream" field specify the identifier of the stream published on the Origin server, then click the "Test now" button.



6. Click the play button in the player window:



Call flow

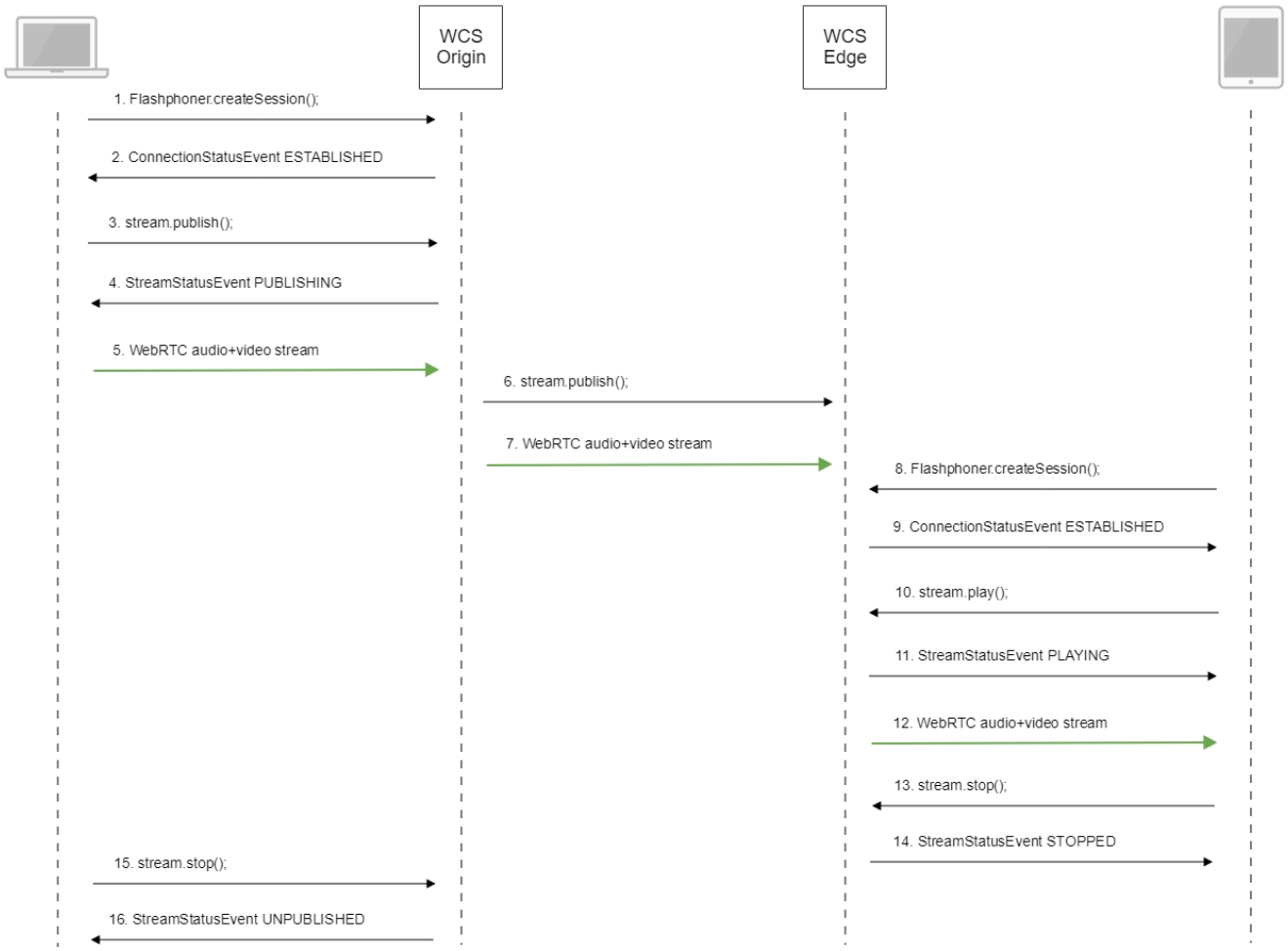
Below is the call flow when using the Two Way Streaming example to publish the stream on the Origin server and the Embed Player example to play the stream on the Edge server

two_way_streaming.html

two_way_streaming.js

player.html

player.js



1. Establishing a 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
        ...
    }).publish();
```

4. Receiving form 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 Origin server

6. Publishing the stream on the Edge server.

7. Sending the audio-video stream via WebRTC to the Edge server.

8. Establishing a connection to the server.

Flashphoner.createSession();code

```
    Flashphoner.createSession({urlServer: urlServer, mediaOptions: mediaOptions}).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();
    });
```

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

ConnectionStatusEvent ESTABLISHEDcode

```
    Flashphoner.createSession({urlServer: urlServer, mediaOptions: mediaOptions}).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 () {
        ...
    });
```

10. Requesting playback of the stream.

stream.play();code

```
    stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
        ...
    });
    stream.play();
```

11. 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) {
        setStatus(stream.status());
        onStarted(stream);
    }).on(STREAM_STATUS.STOPPED, function () {
        ...
    }).on(STREAM_STATUS.FAILED, function () {
        ...
    }).on(STREAM_STATUS.NOT_ENOUGH_BANDWIDTH, function (stream) {
        ...
    });
    stream.play();
```

12. Sending the audio-video stream via WebRTC from the Edge server to the browser

13. Stopping the playback of the stream.

stream.stop();code

```
    $('#play').on('click', function() {
        if (!$('.play-pause').prop('disabled')) {
            if (stopped) {
                ...
            } else {
                if (stream) {
                    stream.stop();
                }
                ...
            };
        };
    });
```

14. Receiving form the server an event confirming the playback of the stream is stopped.

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 () {
        ...
    }).on(STREAM_STATUS.NOT_ENOUGH_BANDWIDTH, function (stream) {
        ...
    });
    stream.play();
```

15. 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("");
}
```

16. Receiving from the server an event confirming unpublishing 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();
```