CDN 1.0

- Описание
  - Схема работы
- Настройка
- Краткое руководство по тестированию
  - Запуск трансляции по WebRTC на Origin-сервере
  - Воспроизведение потока на Edge-сервере
- Последовательность выполнения операций (Call flow)

CDN на базе WCS-серверов может быть организована двумя способами:

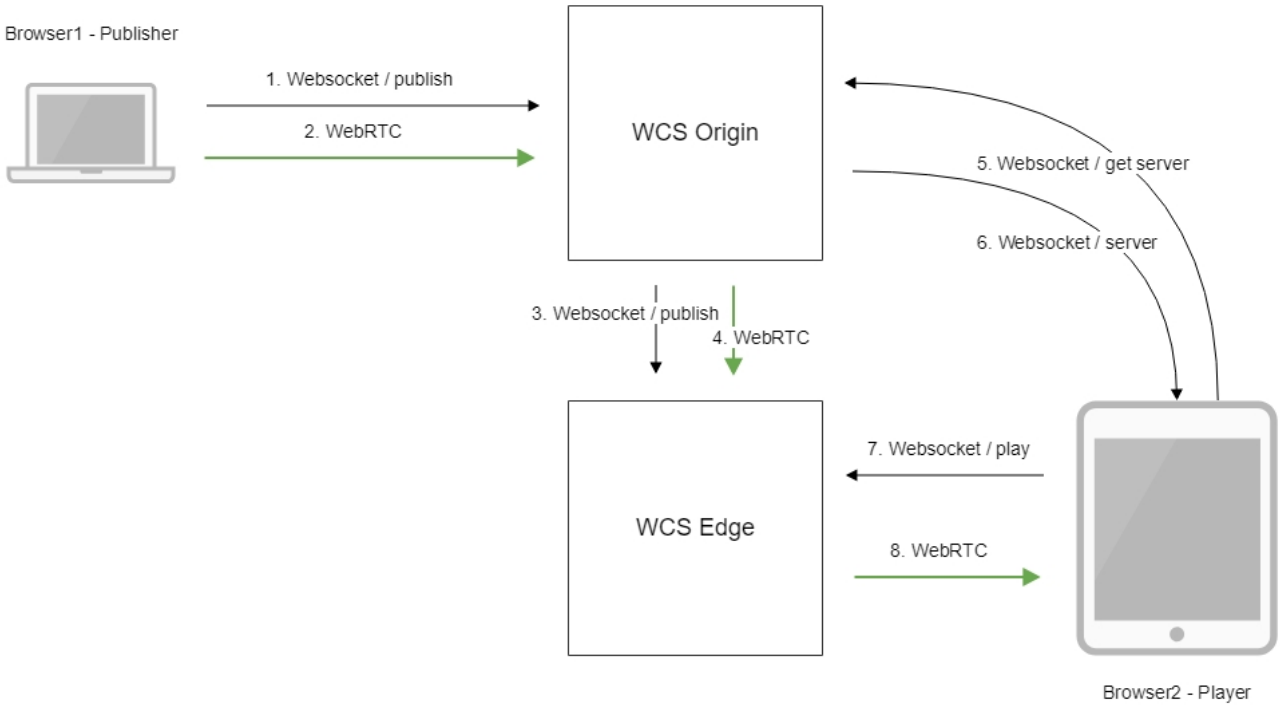
1. Статическая CDN, набор узлов в которой жестко определяется на этапе настройки, и для изменения конфигурации CDN необходимо перезапустить сервер(а), которые являются источниками потоков в данной сети. Такая CDN организуется на базе функции балансировки нагрузки.
2. Динамическая CDN, набор узлов в которой может изменяться на ходу. Для включения/исключения узла из такой CDN достаточно перезапустить только этот узел.

В данном разделе рассмотрим статическую CDN на базе балансировщика нагрузки

Описание

Рассмотрим простейшую конфигурацию с одним сервером для публикации потока (Origin) и одним для воспроизведения (Edge)

Схема работы



1. Браузер соединяется с Origin сервером по протоколу Websocket и отправляет команду publish.
2. Браузер захватывает микрофон и камеру и отправляет WebRTC поток на сервер.
3. Origin сервер соединяется с Edge сервером по протоколу Websocket и отправляет команду publish.
4. Origin сервер отправляет WebRTC поток на Edge сервер.
5. Второй браузер запрашивает у Origin сервера поток.
6. Origin сервер возвращает браузеру адрес Edge сервера, с которого можно забрать поток.
7. Второй браузер устанавливает соединение с Edge сервером по Websocket и отправляет команду play.
8. Второй браузер получает WebRTC поток и воспроизводит этот поток на странице.

Настройка

1. В файле настроекloadbalancing.xmlOrigin-сервера необходимо указать способ ретрансляции и сервер, который будет принимать трансляцию:

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

Здесь:

- stream_distribution="webrtc" указывает, что ретрансляцию нужно проводить по WebRTC (другой возможный вариант - RTMP)
- edge.flashphoner.com - адрес Edge-сервера, который будет принимать поток

2. В файле настроекflashphoner.propertiesвключить режим балансировки нагрузки:

```
load_balancing_enabled=true
```

Краткое руководство по тестированию

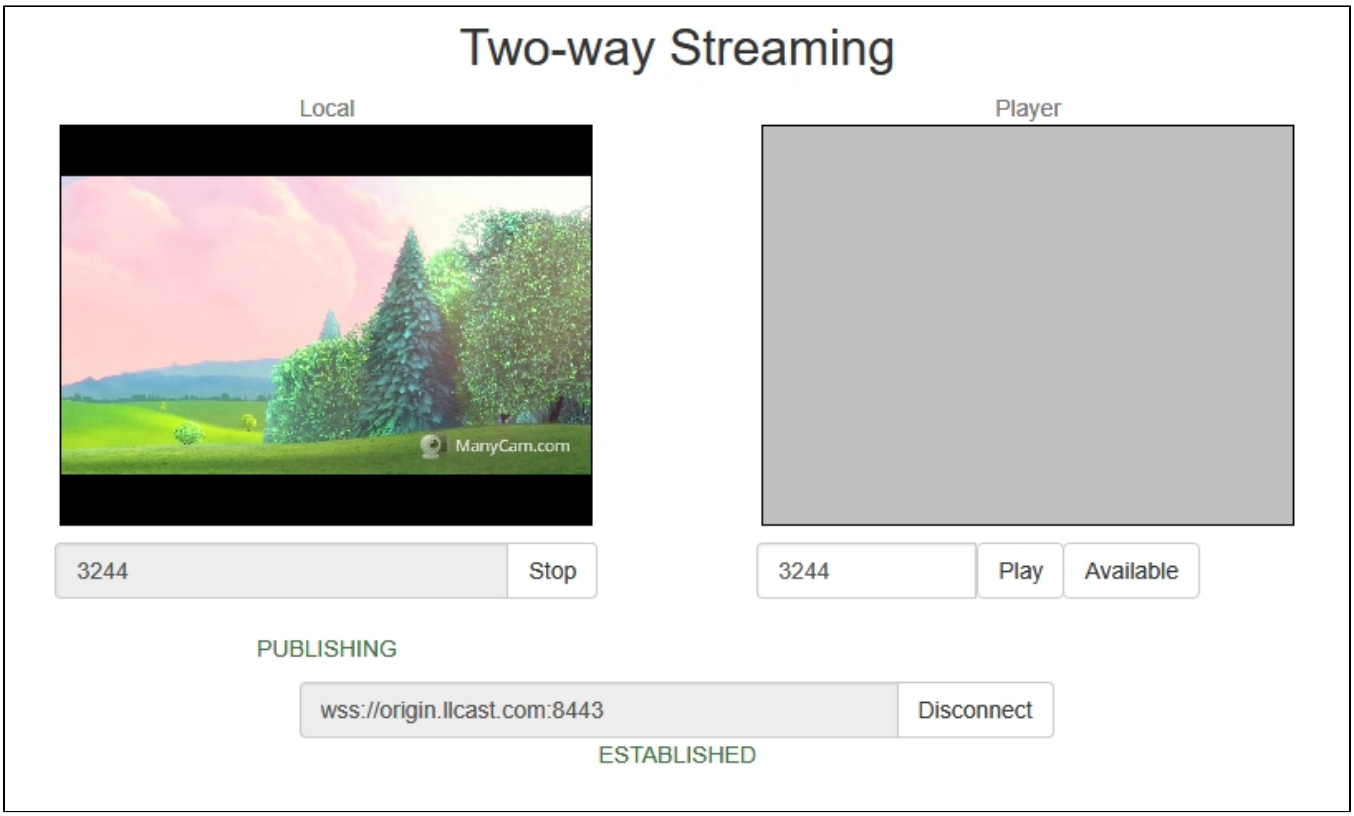
Запуск трансляции по WebRTC на Origin-сервере

1. Для теста используем демо-сервер origin.llcast.com и веб-приложение Two Way Streaming

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

2. Установите соединение с сервером по кнопке Connect

3. Нажмите Publish. Браузер захватывает камеру и отправляет поток на сервер.

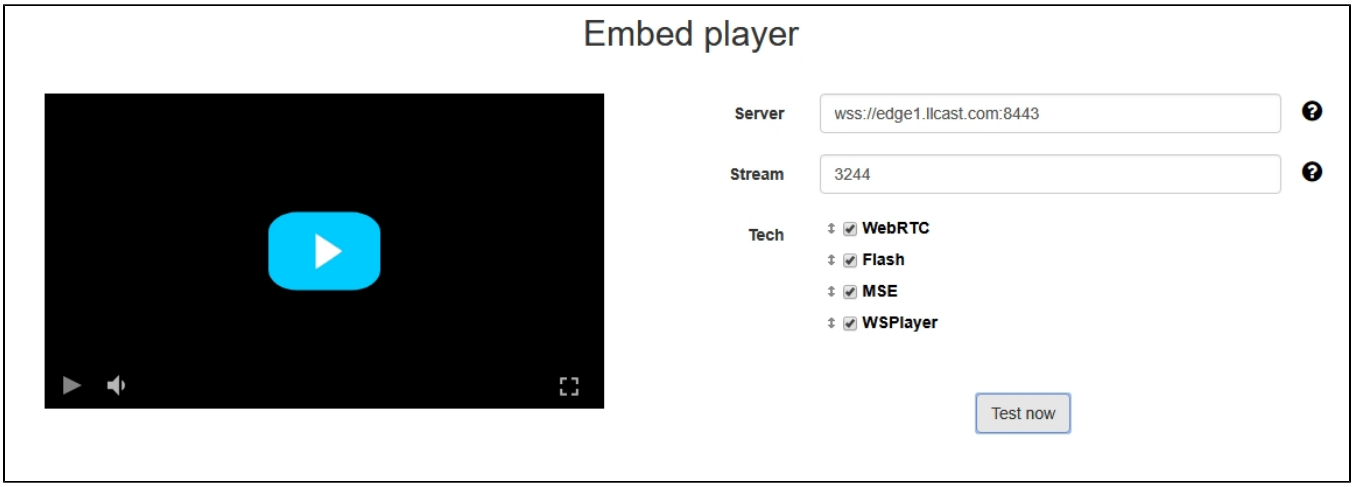


Воспроизведение потока на Edge-сервере

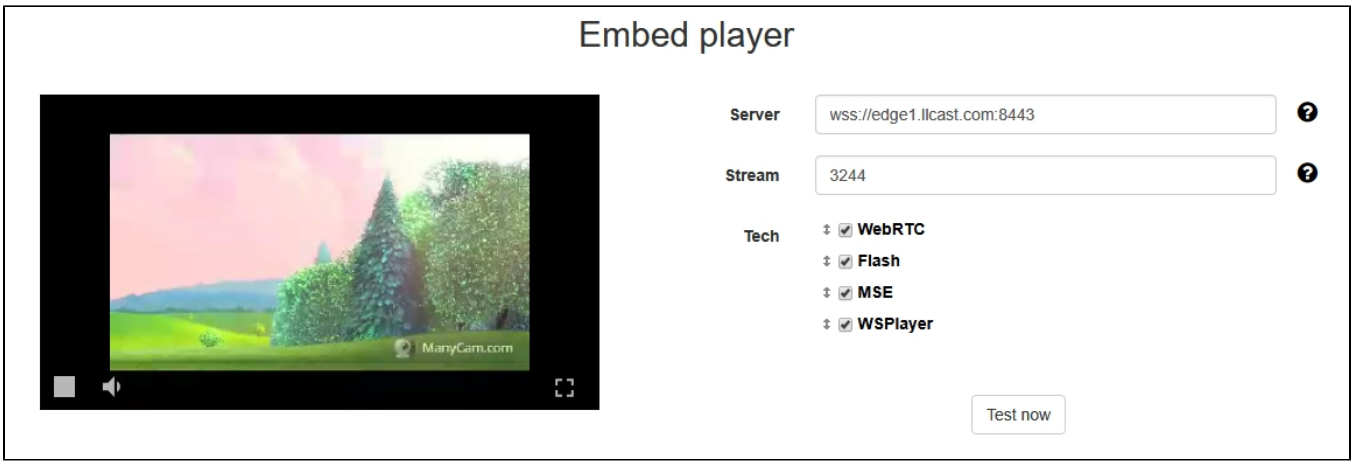
4. Для теста используем демо-сервер edge1.llcast.com и веб-приложение Embed Player

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

5. Укажите в поле "Stream" идентификатор потока, опубликованного на Origin-сервере, и нажмите кнопку "Test now"



6. Нажмите кнопку воспроизведения в окне плеера:



Последовательность выполнения операций (Call flow)

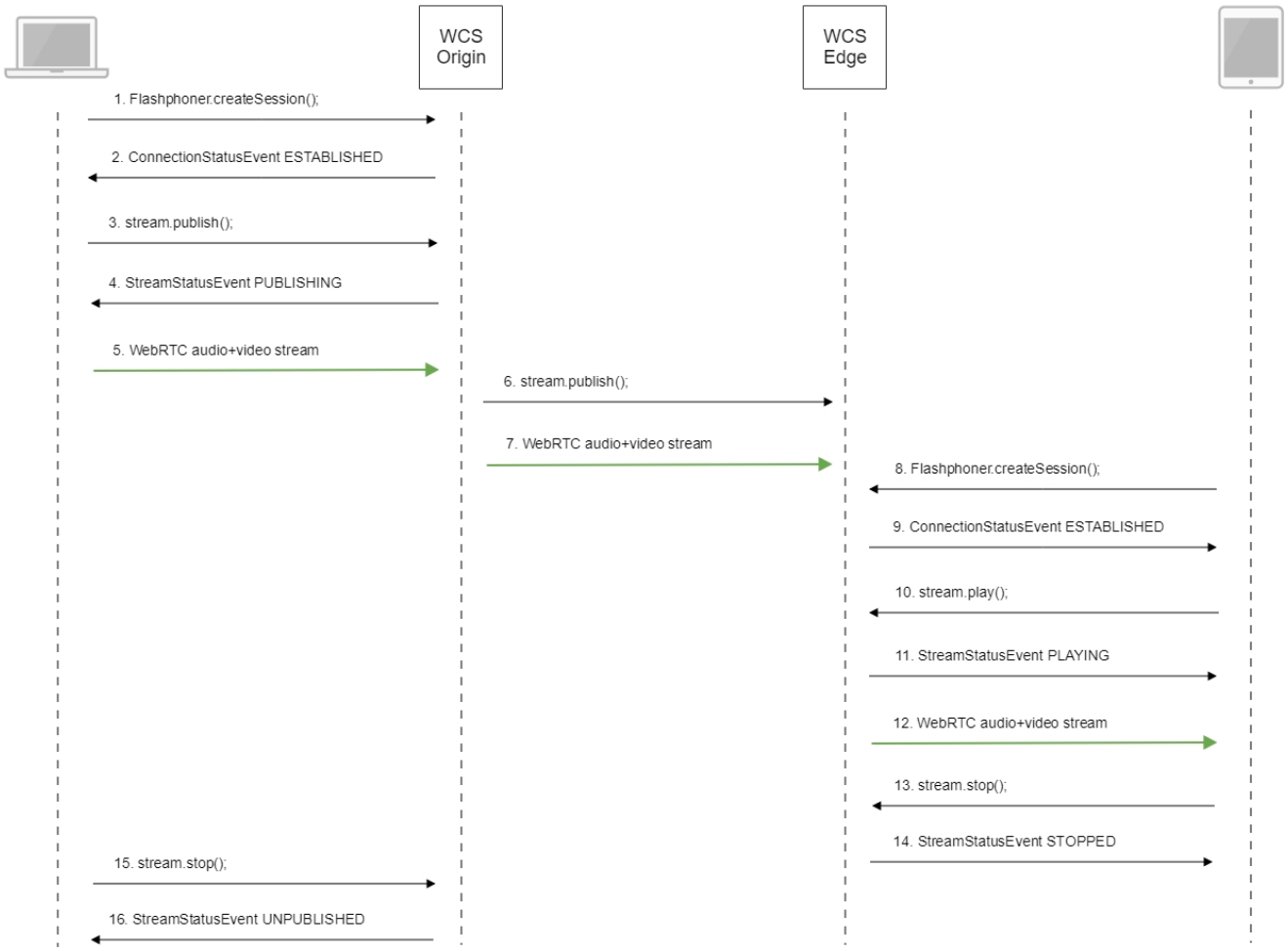
Ниже описана последовательность вызовов при использовании примера Two Way Streaming для публикации потока на Origin-сервере и Embed Player для воспроизведения потока на Edge-сервере

two_way_streaming.html

two_way_streaming.js

player.html

player.js



1. Установка соединения с сервером.

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. Получение от сервера события, подтверждающего успешное соединение.

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. Публикация потока.

stream.publish();code

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

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

StreamStatusEvent, статус 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. Отправка аудио-видео потока по WebRTC на Origin-сервер

6. Публикация потока на Edge-сервере

7. Отправка аудио-видео потока по WebRTC на Edge-сервер

8. Установка соединения с сервером.

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. Получение от сервера события, подтверждающего успешное соединение.

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. Запрос на воспроизведение потока.

stream.play();code

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

11. Получение от сервера события, подтверждающего успешный захват и проигрывание потока.

StreamStatusEvent, статус 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. Отправка аудио-видео потока по WebRTC с Edge-сервера в браузер

13. Остановка воспроизведения потока.

stream.stop();code

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

14. Получение от сервера события, подтверждающего остановку воспроизведения потока.

StreamStatusEvent, статус 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. Остановка публикации потока.

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. Получение от сервера события, подтверждающего остановку публикации потока.

StreamStatusEvent, статус 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();
```