Mixer load testing

Overview

If server use case involves stream mixing, mixer load testing may be necessary before you put server into production. The mixer testing is carried out as follows:

1. The required number of streams are published on server (at least one stream per mixer)
2. The specified number of audiomixers are created, and streams are fed to mixers input. One stream can be fed to one mixer input only
3. Mixers work for desired time then they will be destroyed and created again until the test is finished.

The server behavior can be observed with monitoring tools while testing.

To manage mixel load testing the special REST API queries are used.

REST API

A REST-query must be an HTTP/HTTPS POST query in the following form:

- HTTP: `http://streaming.flashphoner.com:8081/rest-api/mixer/test/start`
- HTTPS: `https://streaming.flashphoner.com:8444/rest-api/mixer/test/start`

Here:

- `streaming.flashphoner.com` - is the address of the WCS server
- `8081` - the standard REST / HTTP port of the WCS server
- `8444` - the standard HTTPS port
- `rest-api` - the required prefix
- `mixer/test/start` - the REST-method used

REST methods

/mixer/test/start

Start the test

REQUEST EXAMPLE

```
POST /mixer/test/start HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
    "feedingStreams": [
        "s1",
        "s2",
        "s3",
        "s4"
    ],
    "mixerCount": 2,
    "streamsInMixer": 2,
    "intervalInSeconds": 60
}
```

RESPONSE EXAMPLE

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
```

RETURN CODES

Code	Reason
200	OK
500	Internal error

/mixer/test/stop

Stop the test

REQUEST EXAMPLE

```
POST /mixer/test/stop HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
}
```

RESPONSE EXAMPLE

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
```

RETURN CODES

Code	Reason

Code	Reason
200	OK
404	Mixer not found
500	Internal error

/mixer/test/get_start_example

Return JSON object sample to pass to `/mixer/test/start` method

REQUEST EXAMPLE

```
POST /mixer/test/get_start_example HTTP/1.1
Host: localhost:8081
Content-Type: application/json

{
    "feedingStreams": [
        "stream1",
        "stream2",
        "stream3"
    ],
    "mixerCount": 3,
    "streamsInMixer": 1,
    "intervalInSeconds": 60
}
```

RESPONSE EXAMPLE

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
```

RETURN CODES

Code	Reason
200	OK
500	Internal error

Parameters

Parameter name	Description	Example
feedingSteams	Stream published list to participate in test	`["s1","s2","s3","s4"]`

Parameter name	Description	Example
mixerCount	Number of mixers created	2
streamsInMixer	Number of streams fed to each mixer input	2
intervalInSeconds	Interval in seconds to destroy mixers and create them again	60

Configuration

To test mixer perfomance under high load, asynchronous media session disconnection (which is enabled by default) should be disabled

```
handler_async_disconnect=false
```

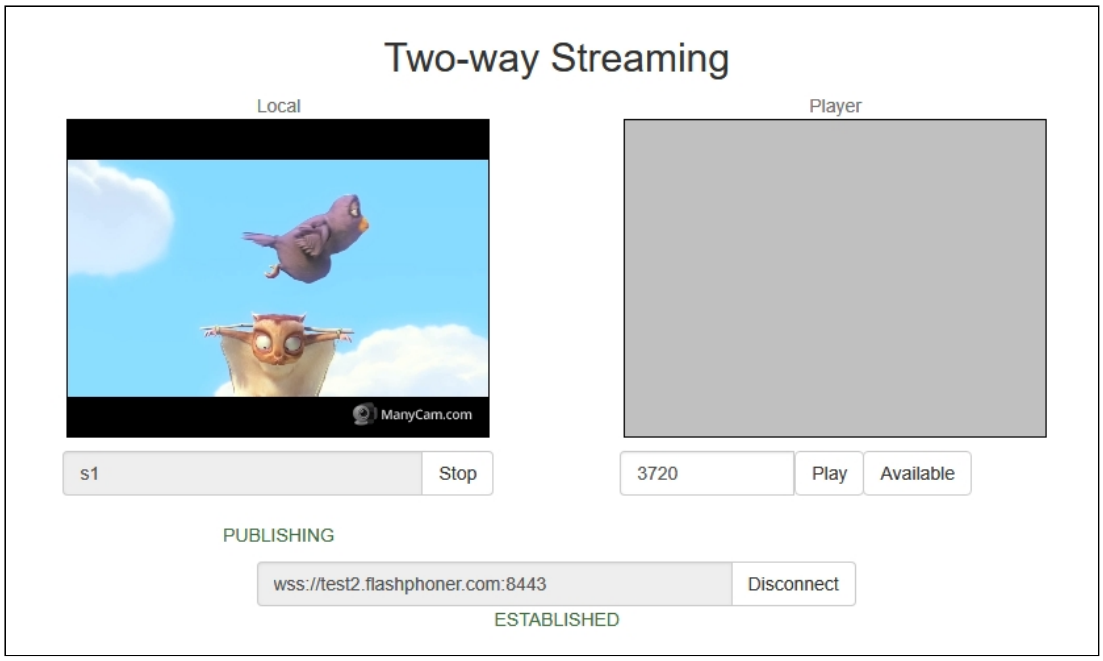
WCS should be restarted to apply.

When test is finished, this setting must be removed from configuration file.

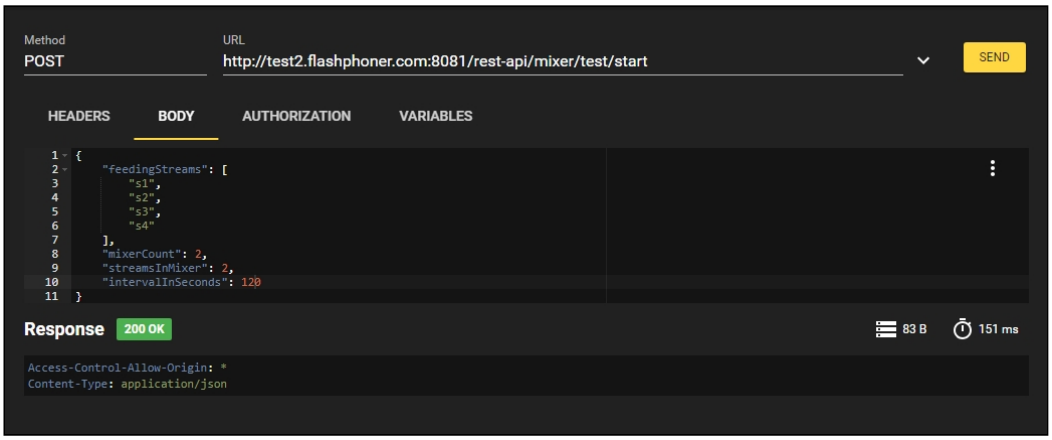
Testing

1. For test we use:
    - WCS server
    - Chrome browser and REST-client to send queries
    - `Two Way Streaming` web application to publish streams

2. Publish streams named `s1`, `s2`, `s3`, `s4`



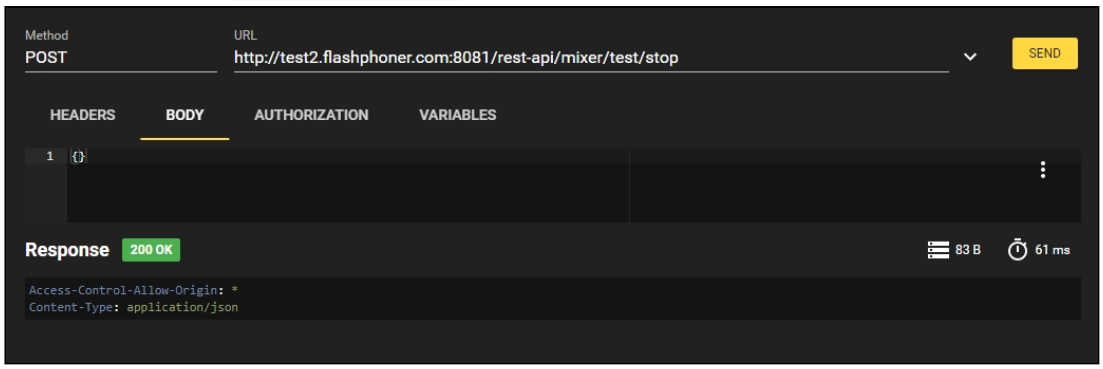
3. Open REST client. Send `/mixer/test/start` query with the following parameters:
   - streams published list: `s1, s2, s3, s4`
   - number of mixers: `2`
   - number of streams per mixer: `2`
   - mixer work interval: `120` seconds



4. Make sure mixers are created sending `/mixer/find_all` query

Two mixers `mixer0` and `mixer1` should be found

5. Server resource consumption can be observed while testing with Java Mission Control, load and resource usage information and error information pages, and server logs.
6. Stop the test with `/mixer/test/stop` query



Tuning recommendations

1. If large CPU load was detected during testing, follow server tuning recommendations.
2. If resource leak was detected during testing, send detailed test description and report collected to support@flashphoner.com