Streaming Auto Restore

A streamer example with publishing/playback automatic restore

This example shows how to restore stream publishing/playback automatically:

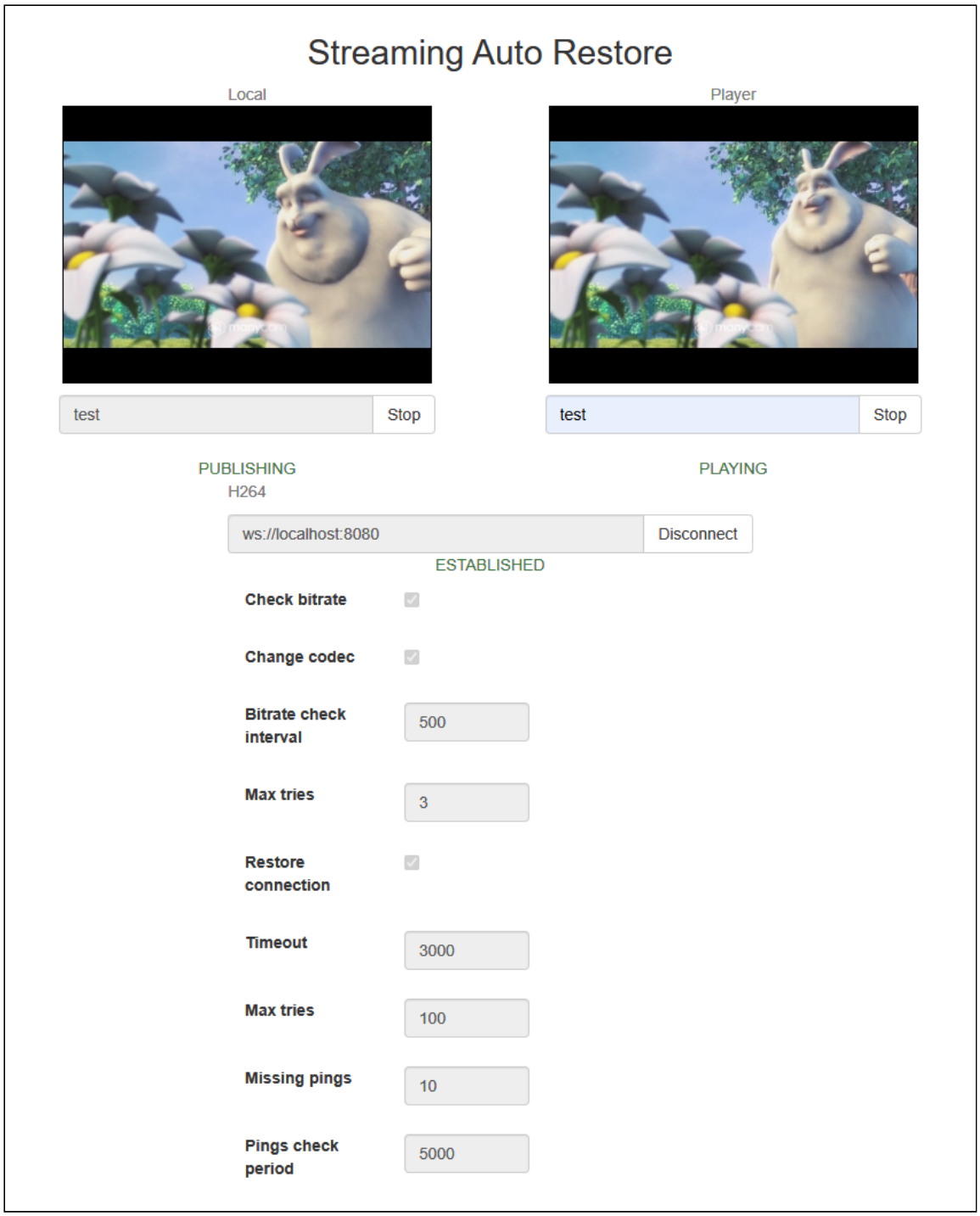
- changing publishing codec to VP8 when H264 publishing fails
- republish a stream if browser stops sending media packets (this is detected by video bitrate drop to 0)
- when network is changing (form Wi-Fi to LTE and vice versa)
- when streaming fails due to server connection breaking (including server restart) or due to stream publishing stopping by other side

Bitrate checking parameters

- Check bitrate - check if bitrate drops to 0
- Change codec - change H264 codec to VP8 if bitrate drop is detected
- Bitrate check interval - publishing bitrate checking interval
- Max tries - maximium number of subsequent bitrate drops to 0

Connection restoring parameters

- Restore connection - restore connection if session breaks or publishing/playback is failed
- Timeout - connection restore tries interval
- Max tries - maximum number of connection restore tries
- Missing pings - maximum number of subsequent missing pings from server (0 disables ping checking)
- Pings check period - pings checking interval (0 disables ping checking)



Code of the example

The example code is available on WCS server by the following path:

/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/stream-auto-restore

- stream-auto-restore.css - styles file
- stream-auto-restore.html - client page

- stream-auto-restore.js - main script to work

The example can be tested by the following URL:

https://host:8888/client2/examples/demo/streaming/stream_filter/stream-auto-restore.html

Where host - WCS server address.

Analyzing the code

To analyze the code take the file `stream-auto-restore.js` version with hash `2035db9` which is available here and can be downloaded with SDK build 2.0.209.

1. Page loading actions

1.1. API initialization

`Flashphoner.init()` code

```
Flashphoner.init();
```

1.2. Session and publishing/playing streams state objects initialization

code

```
currentSession = sessionState();
streamPublishing = streamState();
streamPlaying = streamState();
```

1.3. Bitrate checking object initialization

code

```
h264PublishFailureDetector = codecPublishingFailureDetector();
```

1.4. Connection restore object initialization

code

The function should be passed to the object to execute it when restore connection timer is fired

```
streamingRestarter = streamRestarter(function() {
  if (streamPublishing.wasActive) {
      onPublishRestart();
  }
  if (streamPlaying.wasActive && streamPlaying.name != streamPublishing.name)
```

```
{
      onPlayRestart();
  }
});
```

1.5. Network change detector start

code

```
networkChangeDetector();
```

2. Server connection/disconnection actions

2.1. Connecting to the server

`Flashphoner.createSession()` code

The following parameters are passed when session is created:

- `url` - server Websocket URL
- `receiveProbes` - maximum number of subsequent missing pings from server (0 disables ping checking)
- `probesInterval` - pings checking interval (0 disables ping checking)

```
Flashphoner.createSession({
    urlServer: url,
    receiveProbes: receiveProbes,
    probesInterval: probesInterval
}).on(SESSION_STATUS.ESTABLISHED, function (session) {
    ...
}).on(SESSION_STATUS.DISCONNECTED, function () {
    ...
}).on(SESSION_STATUS.FAILED, function () {
    ...
});
```

2.2. Receiving the event comfirming successful connection

`ConnectionStatusEvent.ESTABLISHED` code

When xconnection is established:

- session parameters are stored in session state object
- stream publishing/playback is restarted if stream was published/played in previous session

```
Flashphoner.createSession({
    urlServer: url,
    receiveProbes: receiveProbes,
```

```
    probesInterval: probesInterval
}).on(SESSION_STATUS.ESTABLISHED, function (session) {
    setStatus("#connectStatus", session.status());
    currentSession.set(url, session);
    onConnected(session);
    if(restoreConnection) {
        if(streamPublishing.wasActive) {
            console.log("A stream was published before disconnection, restart
publishing");
            onPublishRestart();
            return;
        }
        if(streamPlaying.wasActive) {
            console.log("A stream was played before disconnection, restart
playback");
            onPlayRestart();
        }
    }
}).on(SESSION_STATUS.DISCONNECTED, function () {
    ...
}).on(SESSION_STATUS.FAILED, function () {
    ...
});
```

2.3. Connection closing by clicking `Disconnect` button

`Session.disconnect()` code

```
function onConnected(session) {
    $("#connectBtn").text("Disconnect").off('click').click(function () {
        $(this).prop('disabled', true);
        currentSession.isManuallyDisconnected = true;
        session.disconnect();
    }).prop('disabled', false);
    ...
}
```

2.4. Receiving the connection closing event

`ConnectionStatusEvent.DISCONNECTED` code

If connection is closed manually by clicking Disconnect:

- state objects are cleared
- connection restore timer is stopped

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

```
    onDisconnected();
    // Prevent streaming restart if session is manually disconnected
    if (currentSession.isManuallyDisconnected) {
        streamPublishing.clear();
        streamPlaying.clear();
        streamingRestarter.reset();
        currentSession.clear();
    }
}).on(SESSION_STATUS.FAILED, function () {
    ...
});
```

2.5. Receiving the connection failure event

`ConnectionStatusEvent FAILED` code

Connection restore timer is starting if a stream was published or played befor connection is failed

```
Flashphoner.createSession({
    urlServer: url,
    receiveProbes: receiveProbes,
    probesInterval: probesInterval
}).on(SESSION_STATUS.ESTABLISHED, function (session) {
    ...
}).on(SESSION_STATUS.DISCONNECTED, function () {
    ...
}).on(SESSION_STATUS.FAILED, function () {
    setStatus("#connectStatus", SESSION_STATUS.FAILED);
    onDisconnected();
    if(restoreConnection
       && (streamPublishing.wasActive || streamPlaying.wasActive)) {
        streamingRestarter.restart($("#restoreTimeout").val(),
$("#restoreMaxTries").val());
    }
});
```

3. Stream publishing actions

3.1 Stream publishing

`Session.createStream()`, `Stream.publish()` code

The following parameters are passed while stream creation:

- `streamName` - stream name to publish
- `localVideo` - `div` element to display local video
- `stripCodecs` - codec to exclude if codec changing option is active

```
session.createStream({
    name: streamName,
    display: localVideo,
```

```
    cacheLocalResources: true,
    receiveVideo: false,
    receiveAudio: false,
    stripCodecs: stripCodecs
    ...
}).publish();
```

3.2. Receiving the stream publishing event

`StreamStatusEvent PUBLISHING` code

When stream is publishing successfully:

- bitrate checking timer starts
- the stream parameters are stored in publishing stream state object
- connection restore timer stops
- stream playback starts if stream was played previously

```
session.createStream({
    ...
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
    setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
    onPublishing(stream);
    streamPublishing.set(streamName, stream);
    streamingRestarter.reset();
    if ($("#restoreConnection").is(':checked')
       && streamPlaying.wasActive) {
        console.log("A stream was played before, restart playback");
        onPlayRestart();
    }
}).on(STREAM_STATUS.UNPUBLISHED, function () {
    ...
}).on(STREAM_STATUS.FAILED, function (stream) {
    ...
}).publish();
```

3.3. Bitrate checking timer startup

code

```
function onPublishing(stream) {
    ...
    // Start publish failure detector by bitrate #WCS-3382
    if($("#checkBitrate").is(':checked')) {
        h264PublishFailureDetector.startDetection(stream,
$("#bitrateInteval").val(), $("#bitrateMaxTries").val());
    }
}
```

3.4. Publishing stopping by clicking `Stop` button

`Stream.stop()` code

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

3.5. Receiving stream publishing stoppin event

`StreamStatusEvent.UNPUBLISHED` code

When stream is successfully stopped:

- bitrate checking timer stops
- connection restore timer stops
- publishing stream state object is cleared

```
session.createStream({
    ...
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
    ...
}).on(STREAM_STATUS.UNPUBLISHED, function () {
    setStatus("#publishStatus", STREAM_STATUS.UNPUBLISHED);
    onUnpublished();
    if (!streamPlaying.wasActive) {
        // No stream playback< we don't need restart any more
        streamingRestarter.reset();
    } else if (streamPlaying.wasActive && streamPlaying.name ==
streamPublishing.name) {
        // Prevent playback restart for the same stream
        streamingRestarter.reset();
    }
    streamPublishing.clear();
}).on(STREAM_STATUS.FAILED, function (stream) {
    ...
}).publish();
```

3.6. Receiving stream publishing failure event

`StreamStatusEvent.FAILED` code

When stream publishing fails:

- bitrate checking timer stops
- connection restore timer starts unless local browser error is detected (media devices unavailable for example)

```
session.createStream({
    ...
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
```

```
    ...
}).on(STREAM_STATUS.UNPUBLISHED, function () {
   ...
}).on(STREAM_STATUS.FAILED, function (stream) {
    setStatus("#publishStatus", STREAM_STATUS.FAILED, stream);
    onUnpublished();
    if ($("#restoreConnection").is(':checked') && stream.getInfo() !=
ERROR_INFO.LOCAL_ERROR) {
        streamingRestarter.restart($("#restoreTimeout").val(),
$("#restoreMaxTries").val());
    }
}).publish();
```

3.7. Bitrate checking timer stopping

code

```
function onUnpublished() {
    ...

h264PublishFailureDetector.stopDetection(streamPublishing.isManuallyStopped
|| currentSession.isManuallyDisconnected);
    ...
}
```

4. Stream playback actions

4.1. Stream playback

`Session.createStream()`, `Stream.play()` code

The following parameters are passed while stream creation:

- `streamName` - stream name to play
- `remoteVideo` - `div` element to display remote video

```
session.createStream({
    name: streamName,
    display: remoteVideo
    ...
}).play();
```

4.2. Receiving the stream playback event

`StreamStatusEvent PLAYING` code

When stream is successfully playing:

- the stream parameters are stored in playing stream state object
- connection restore timer stops

```
session.createStream({
    name: streamName,
    display: remoteVideo
}).on(STREAM_STATUS.PENDING, function (stream) {
    ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
    setStatus("#playStatus", stream.status());
    onPlaying(stream);
    streamingRestarter.reset();
    streamPlaying.set(streamName, stream);
}).on(STREAM_STATUS.STOPPED, function () {
    ...
}).on(STREAM_STATUS.FAILED, function (stream) {
    ...
}).play();
```

4.3 Stream playback stopping by clicking `Stop` button

`Stream.stop()` code

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

4.4. Receiving the stream playback stopping event

`StreamStatusEvent STOPPED` code

When stream playback is successfully stopped:

- connection restore timer stops
- playing stream state object is cleared

```
session.createStream({
    name: streamName,
    display: remoteVideo
}).on(STREAM_STATUS.PENDING, function (stream) {
    ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
    ...
}).on(STREAM_STATUS.STOPPED, function () {
    setStatus("#playStatus", STREAM_STATUS.STOPPED);
    onStopped();
    streamingRestarter.reset();
    streamPlaying.clear();
}).on(STREAM_STATUS.FAILED, function (stream) {
    ...
}).play();
```

4.5. Receiving the stream playback failure event

`StreamStatusEvent FAILED` code

Connection restore timer starts is stream playback fails

```
    session.createStream({
        name: streamName,
        display: remoteVideo
    }).on(STREAM_STATUS.PENDING, function (stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function (stream) {
        ...
    }).on(STREAM_STATUS.STOPPED, function () {
        ...
    }).on(STREAM_STATUS.FAILED, function (stream) {
        setStatus("#playStatus", STREAM_STATUS.FAILED, stream);
        onStopped();
        if ($("#restoreConnection").is(':checked')) {
            streamingRestarter.restart($("#restoreTimeout").val(),
$("#restoreMaxTries").val());
        }
    }).play();
```

5. Bitrate checking and stream republishing if bitrate drops to 0

5.1. Receiving browser WebRTC statistics, codec and bitrate detection, publishing stopping if bitrate drops to 0

code

```
stream.getStats(function(stat) {
    let videoStats = stat.outboundStream.video;
    if(!videoStats) {
        return;
    }
    let stats_codec = videoStats.codec;
    let bytesSent = videoStats.bytesSent;
    let bitrate = (bytesSent - detector.lastBytesSent) * 8;
    if (bitrate == 0) {
        detector.counter.inc();
        console.log("Bitrate is 0 (" + detector.counter.getCurrent() + ")");
        if (detector.counter.exceeded()) {
            detector.failed = true;
            console.log("Publishing seems to be failed, stop the stream");
            stream.stop();
        }
    } else {
        detector.counter.reset();
    }
    detector.lastBytesSent = bytesSent;
    detector.codec = stats_codec;
    $("#publishInfo").text(detector.codec);
});
```

5.2. Bitrate checking timer stopping

code

```
if (detector.publishFailureIntervalID) {
    clearInterval(detector.publishFailureIntervalID);
    detector.publishFailureIntervalID = null;
}
```

5.3. Stream republishing

code

```
if (detector.failed) {
    $("#publishInfo").text("Failed to publish " + detector.codec);
    if($("#changeCodec").is(':checked')) {
        // Try to change codec from H264 to VP8 #WCS-3382
        if (detector.codec == "H264") {
            console.log("H264 publishing seems to be failed, trying VP8 by
stripping H264");
            let stripCodecs = "H264";
            publishBtnClick(stripCodecs);
        } else if (detector.codec == "VP8") {
            console.log("VP8 publishing seems to be failed, giving up");
        }
    } else {
        // Try to republish with the same codec #WCS-3410
        publishBtnClick();
    }
}
```

6. Connection restoration

6.1. Connection restore timer launching

code

The timer invokes a function to perform an ctions needed

```
restarter.restartTimerId = setInterval(function(){
    if (restarter.counter.exceeded()) {
        logger.info("Tried to restart for " + restartMaxTimes + " times with
" +restartTimeout + " ms interval, cancelled");
        restarter.reset();
        return;
    }
    onRestart();
    restarter.counter.inc();
}, restartTimeout);
```

6.2. Connection restore timer stopping

code

```
if (restarter.restartTimerId) {
    clearInterval(restarter.restartTimerId);
    logger.info("Timer " + restarter.restartTimerId + " stopped");
    restarter.restartTimerId = null;
}
restarter.counter.reset();
```

6.3. New session creation if previous session is failed or disconnected

code

```
let sessions = Flashphoner.getSessions();
if (!sessions.length || sessions[0].status() == SESSION_STATUS.FAILED) {
    logger.info("Restart session to publish");
    click("connectBtn");
} else {
    ...
}
```

6.4. Republishing

code

```
let streams = sessions[0].getStreams();
let stream = null;
let clickButton = false;
if (streams.length == 0) {
    // No streams in session, try to restart publishing
    logger.info("No streams in session, restart publishing");
    clickButton = true;
} else {
    // If there is already a stream, check its state and restart publishing
if needed
    for (let i = 0; i < streams.length; i++) {
        if (streams[i].name() === $('#publishStream').val()) {
            stream = streams[i];
            if (!isStreamPublishing(stream)) {
                logger.info("Restart stream " + stream.name() + "
publishing");
                clickButton = true;
            }
            break;
        }
    }
    if (!stream) {
        logger.info("Restart stream publishing");
        clickButton = true;
    }
}
if (clickButton) {
    click("publishBtn");
}
```

6.5. Replaying

code

```
let streams = sessions[0].getStreams();
let stream = null;
let clickButton = false;
if (streams.length == 0) {
    // No streams in session, try to restart playing
    logger.info("No streams in session, restart playback");
    clickButton = true;
} else {
    // If there is already a stream, check its state and restart playing if
needed
    for (let i = 0; i < streams.length; i++) {
        if (streams[i].name() === $('#playStream').val()) {
            stream = streams[i];
            if (!isStreamPlaying(stream)) {
                logger.info("Restart stream " + stream.name() + " playback");
                clickButton = true;
            }
            break;
        }
    }
    if (!stream) {
        logger.info("Restart stream playback");
        clickButton = true;
    }
}
if (clickButton) {
    click("playBtn");
}
```

7. Network change actions

7.1. Network change event handling

`NetworkInformation.onchange` code

```
if (Browser.isChrome() || (Browser.isFirefox() && Browser.isAndroid())) {
    connection = navigator.connection || navigator.mozConnection ||
navigator.webkitConnection;
    if (connection) {
        connectionType = connection.type;
        if (Browser.isFirefox()) {
            connection.ontypechange = onNetworkChange;
        } else {
            connection.onchange = onNetworkChange;
        }
    }
}
```

7.2. Closing the connection if network is changed

code

```
if (isNetworkConnected() && connection.type != connectionType) {
    if (currentSession.getStatus() == SESSION_STATUS.ESTABLISHED) {
        let logger = Flashphoner.getLogger();
        logger.info("Close session due to network change from " +
connectionType + " to " + connection.type);
        currentSession.sdkSession.disconnect();
    }
}
```