Ceeblue Streaming Cloud

Signalling

Danny Burns

Version 1.4.1, 22 Feb 2021

Table of Contents

Changes

- **1.4.1 Feb 22 2021** Add peer configuration setter for **WebRTCStreamer** and **WebRTCPlayback** (*The options include ICE server and transport settings and identity information.*)

Chapter 1. Signalling protocol

1.1. Overview

WebRTC uses RTCPeerConnection to communicate streaming data between browsers (aka peers) but also needs a mechanism to coordinate communication and to send control messages, a process known as signalling.

We are using WebSocket signalling between WebRTC clients and the server. WebSockets enables opening up a session from the client to the webserver and then leaving it open for messages from both directions.

We are using the JSON messages format. To start signalling, clients need to open a WebSocket connection with the server.

WebSocket URL format

```
wss://[hostname and port]/webrtc/[stream name]
```

Example

```
wss://ice.ceeblue.tv:4433/webrtc/as+c94ca3ca-4f4b-415d-9467-7ae2fb9bf2d2
```

1.2. Requests

1.2.1. Common

Session Description Protocol (SDP offer)

Path	Type	Required	Description
type	const string	yes	Request type `offer_sdp`
offer_sdp	string	yes	SDP offer (plain text)

Example:

```
{
  "type": "offer_sdp",
  "offer_sdp": "v=0\r\no=- 724438709080065200 2 IN IP4 127.0.0.1\r\n...a=rtpmap:123
ulpfec/90000\r\n"
}
```

1.2.2. Playback

Switch track

In the case of a multi quality stream, the user can select (switch) the tracks (qualities) that he wants to receive.

Path	Type	Required	Description
type	const string	yes	Request type `tracks`
audio	number	no	Audio unique track id (see `CeeblueStreamingCloud Stream metadata`)
video	number	no	Video unique track id (see `CeeblueStreamingCloud Stream metadata`)

Examples

```
{
  "type": "tracks",
  "video": 2
}
```

```
{
  "type": "tracks",
  "audio": 0
}
```

1.2.3. Streaming

Set video bitrate limit

It causes the server to send a WebRTC command telling the browser to limit the max video bitrate to that value.

Path	Type	Required	Description
type	const string	yes	Request type `video_bitrate`
video_bitrate	number	yes	Target video bitrate

```
{
  "type": "video_bitrate",
  "video_bitrate": 1600000
}
```

1.3. Responses

1.3.1. Common

Session Description Protocol (SDP answer)

Path	Description
type	Response types `on_answer_sdp`
result	Initialization result. (**true**: internal checks succeeded, and the media transfer is ready)
answer_sdp	SDP answer (plain text)

Example

```
{
  "type": "on_answer_sdp",
  "result": true,
  "answer_sdp": "v=0\r\no=- 1588582671271257 0 IN IP4 0.0.0.0\r\n...\r\na=end-of-
candidates\r\n"
}
```

1.3.2. Streaming

Set video bitrate

Happens in case of successful setting of the video bitrate or change by the server.

Path	**Description**
type	Response types `on_video_bitrate`
result	Command result (**true**: accepted)
video_bitrate	Video bitrate
video_bitrate_constraint	Video bitrate constraint

Example

```
{
  "result": true,
  "type": "on_video_bitrate",
  "video_bitrate": 2100000,
  "video_bitrate_constraint": 2100000
}
```

RTP props

On RTP settings change

Path	Description
type	Response types `on_rtp_props`
result	Command result (**true**: accepted)
drop	Amount of packet any track will wait for a packet to arrive before considering it loss
nack	Amount of packet any track will wait for a packet to arrive before NACKing it

Example

```
{
  "result": true,
  "type": "on_rtp_props",
  "drop": 30,
  "nack": 5
}
```

Media

Periodic report on the status of streaming quality stats.

Path	Description
type	Response types `on_media_receive`
millis	Amount of media received in ms
jitter_ms	Average jitter between video packets (before reordering)
loss_num	Total packets lost since start of broadcast (never resets)
nack_num	Total packets NACKed since start of broadcast (never resets)
loss_perc	Percentage of lost in the last ~1 second (resets once per second)

Example

```
{
  "type": "on_media_receive",
  "millis": 13900,
  "stats": {
    "jitter_ms": 3.0552641779,
    "loss_num": 0,
    "loss_perc": 0.0000000000,
    "nack_num": 0
  },
  "tracks": [ "H264", "opus" ]
}
```

1.3.3. Playback

On time

Periodic report on the status of playing.

Path	Description
type	Response types `on_time`
begin	Upper bound of the playback buffer
current	Current playback position
end	Lower bound of the playback buffer
tracks	List of playing track ids

Example

```
{
  "type": "on_time",
  "begin": 2057477,
  "current": 2110997,
  "end": 2111317,
  "tracks": [
    1,
    6
  ]
}
```

On stop

This happens when a live stream ends

Path	Description
type	Response types `on_stop`

Example

```
{
  "type": "on_stop",
  "begin": 0,
  "current": 41084,
  "end": 0
}
```

Chapter 2. Streaming overview

2.1. Server-side

2.1.1. Video bitrate

The server accepts a video bitrate `video_bitrate` (default value 6000000). The setting is used to notify a sender of the total estimated available bit rate on the receiving side.

2.1.2. RTP properties

The server accepts `drop` and `nack`. These settings allow you to tune how quickly the server will respond to network problems.

- `drop` - Amount of packet any track will wait for a packet to arrive before considering it loss (default 30)
- `nack` - Amount of packet any track will wait for a packet to arrive before NACKing it (default 5)

2.1.3. Error handling

In case of packet loss, the server reduces video bitrate `video_bitrate_constrain`.

Once per second:

- > 50% loss, constrain previous value by 9%
- > 10% loss, constrain previous value by 5%
- > 5% loss, constrain previous value by 1%
- < 5% loss, do not constrain further

2.2. Client-side

The description of the code used on the demo push (streaming) page. It consists of three modules that can be worked together.

Push page example: https://ice.ceeblue.tv/push.html

2.2.1. Signalling

The *webrtc-streamer.js* **WebRTCStreamer** class.

Responsible for the signalling protocol (communication with the server), which is necessary for the initialization and control of the WebRTC connection.

The constructor requires a server endpoint, a stream name, an access token (optional).

Example

```
let signalling = new WebRTCStreamer('ice.ceeblue.tv:4433', 'as+083563ed-9bf9-4d5a-
90b9-aee35afda3bf');
```

Events:

- `started` - on streaming started
- `stopped` - on streaming stopped
- `error` - on errors happened
- `rtp_props` - on RTP properties set
- `stats` - on server stats received
- `video_bitrate` - on video bitrate set/change

Example

```
signalling.addEventListener('started',   onStreamingStarted);
signalling.addEventListener('stopped',   onStreamingStopped);
signalling.addEventListener('error',     onStreamingError);
signalling.addEventListener('rtp_props', onRtpProps);
```

Methods:

- `setMediaStream` - set media stream for streaming
- `setPeerConfiguration` - set an RTCConfiguration dictionary providing options to configure the new connection
- `start` - start streaming
- `stop` - stop streaming
- `setRTPProps(nack, drop)` - set RTP properties
- `setTargetVideoBitrate` - set target video bitrate
- `getStats(callback)` - get peer connection statistics

Example

```
signalling.setPeerConfiguration({
  iceServers: [{
    urls: ['turn:....ceeblue.tv:3478?transport=tcp','turn:....ceeblue.tv:3478'],
    username: 'xxx', credential: 'yyy',
  }]
});
signalling.setMediaStream(stream); // Set media stream
signalling.start(); // Start streaming
```

2.2.2. Adaptive input manager

The *webrtc-adaptive-input.js* **WebRTCAdaptiveInputManager** class.

Responsible for video bitrate adapting to the variability of the Internet bandwidth. The module is optional.

The constructor requires a signalling instance WebRTCStreamer.

Example

```
let adaptiveInput = new WebRTCAdaptiveInputManager(signalling);
adaptiveInput.setVideoBitrateLimit(3000000); // Maximum bitrate
adaptiveInput.setStartupVideoBitrate(3000000 * 0.7); // Let start from 2 Mbps
```

Default workflow

Manager collects server media stats, and monitors the value of lost packets for the last 5 seconds. It also collects target video bitrate for the last 60 seconds, when the stream was lossless.

When video bitrate less than the limit, then the manager checks the possibility to recover bitrate to reach video bitrate limit. It happens at intervals of 2,5 to 60 seconds, depending on the success of the recovery attempt. The recovery interval is halved if the previous attempt to increase bitrate was successful (min 2.5 seconds). The recovery interval is doubled if the previous attempt to increase the bitrate failed (maximum 60 seconds). The attempt is considered unsuccessful if after increasing the bitrate there are packet losses.

On a streaming startup, the manager sets the target video bitrate value to 2Mbit/s. (The limit is 3Mbit/s). If during the first 10 seconds of streaming, there is a loss of packets, then the video bitrate is halved.

The manager increases the bitrate by 5% with each attempt, when target bitrate is less than average and has not been packet loss in the last 5 seconds. It increases the target bitrate by 0.5% if the value is above average bitrate (slow increase, to avoid bitrate change jumps).

If packet loss does not exceed 5%, the manager reduces the bitrate by 1%. (As we are approaching the bandwidth)

If the bitrate is less than 1.2Mbps, and the image size is about HD, then the manager will halve the image size.

2.2.3. Metrics monitoring

The *webrtc-monitoring.js* **WebRTCStatsMonitoring** class.

esponsible for collecting peer statistics, etc, and sending it to the Ceeblue server. The module is optional.

The constructor requires a signalling instance WebRTCStreamer.

Example

```
let monitoring = new WebRTCStatsMonitoring(signalling);
```

Chapter 3. Playback overview

3.1. Client-side

The description of the code used on the demo playback page.

Play page example: https://ice.ceeblue.tv/play.html

3.1.1. Signalling

The *webrtc-playback.js* **WebRTCPlayback** class.

Responsible for the signalling protocol (communication with the server), which is necessary for the initialization and control of the WebRTC connection.

The constructor requires a server endpoint, a stream name, an access token (optional).

Example

```
let signalling = new WebRTCPlayback('ice.ceeblue.tv:4433', 'as+083563ed-9bf9-4d5a-
90b9-aee35afda3bf');
```

Events:

- `connected` - on signalling started
- `started` - on playback started
- `stopped` - on playback stopped
- `time` - on playback position update
- `error` - on errors happened

Example

```
signalling.addEventListener('started',   onPlaybackStarted);
signalling.addEventListener('stopped',   onPlaybackStopped);
signalling.addEventListener('error',     onPlaybackError);
```

Methods:

- `setVideoElement` - set video (DOM) element for image display
- `setPeerConfiguration` - set an RTCConfiguration dictionary providing options to configure the new connection
- `start` - start playback
- `stop` - stop playback
- `audioTrack` - set audio track index

- `videoTrack` - get video track index

Example

```
signalling.setPeerConfiguration({
  iceServers: [{
    urls: ['turn:....ceeblue.tv:3478?transport=tcp','turn:....ceeblue.tv:3478'],
    username: 'xxx', credential: 'yyy',
  }]
});
signalling.setVideoElement(stream); // Set video element
signalling.start(); // Start streaming
```