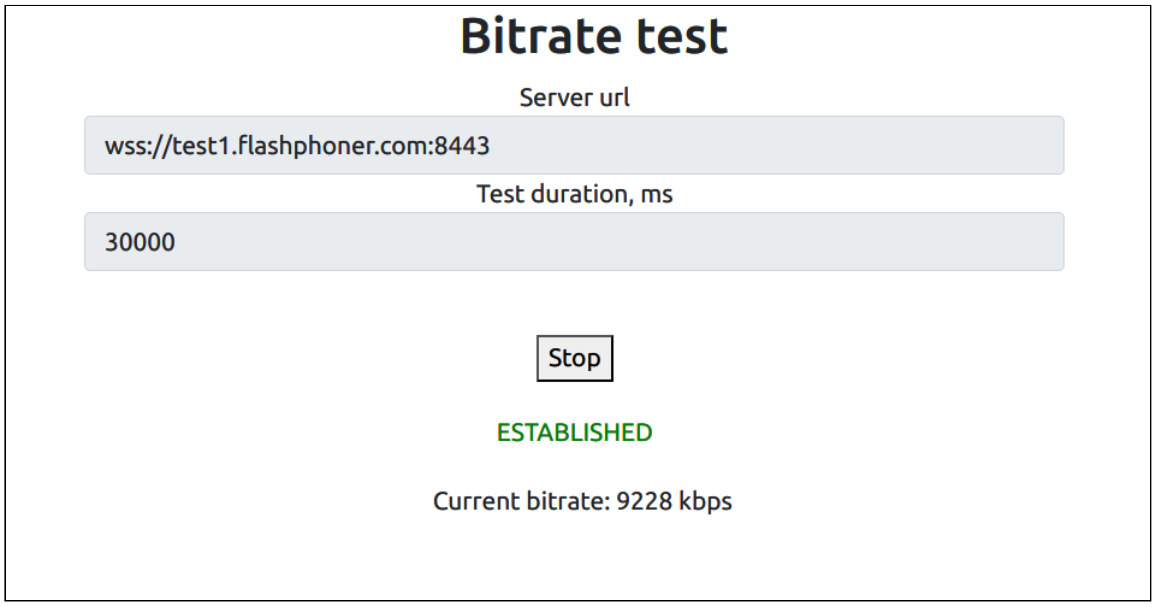
SFU Bitrate Test

The example shows how to test a publishing channel bandwidth.

On the screenshot below

- `Server url` - Websocket URL of WCS server
- `Test duration` - test maximum duration in ms
- `Current bitrate` - a current measured data sending bitarte in kbps



Example source code

The source code consists of the following modules:

`bitrate_test.html` - HTML page `bitrate_test.css` - HTML page style `bitrate_test.js` - main application logic

Analyzing the code

To analyze the example source code, take the file `bitrate_test.js` available here.

1. Local variables

Local variables declaration to work with constants, SFU SDK, to store a current state and to work with a test room

code

```
const constants = SFU.constants;
const sfu = SFU;
let bitrateTestState;

const BITRATE_TEST = "bitrateTest";
const TEST_DURATION = 30000;
```

2. Object to store current testing state

The object should keep Websocket session data, WebRTC connection data, room data and object to launch the test

code

```
const CurrentState = function (prefix) {
    let state = {
        prefix: prefix,
        pc: null,
        session: null,
        room: null,
        bitrateController: null,
        set: function (pc, session, room) {
            state.pc = pc;
            state.session = session;
            state.room = room;
        },
        clear: function () {
            state.room = null;
            state.session = null;
            state.pc = null;
            state.bitrateController = null;
        },
        durationId: function () {
            return state.prefix + "Duration";
        },
        buttonId: function () {
            return state.prefix + "Btn";
        },
        statusId: function () {
            return state.prefix + "Status";
        },
        errInfoId: function () {
            return state.prefix + "ErrorInfo";
        },
        currentStateId: function () {
            return state.prefix + "CurrentState";
        },
        getBitrateController: function () {
            return state.bitrateController;
```

```
        },
        setBitrateController: function (controller) {
            state.bitrateController = controller;
        },
        isConnected: function () {
            return (state.session && state.session.state() ===
constants.SFU_STATE.CONNECTED);
        }
    };
    return state;
}
```

3. Initialization

code

The `init()` function is called on page load and:

- initializes state objects
- initializes input fields

```
const init = function () {
    bitrateTestState = CurrentState(BITRATE_TEST);
    $("#" + bitrateTestState.buttonId()).prop('disabled', true);
    $("#url").prop('disabled', true);
    onDisconnected(bitrateTestState);
    $("#url").val(setURL());
    $("#" + bitrateTestState.durationId()).val(TEST_DURATION);
}
```

4. Establishing server connection

`RTCPeerConnection()`, `SFU.createRoom()` code

The `connect()` function is called by Start button click:

- creates `PeerConnection` object
- cleans the previous session state displayed
- sets up room configuration and creates Websocket session
- subscribes to Websocket session events

```
const connect = async function (state) {
    //create peer connection
    const pc = new RTCPeerConnection();
    //get config object for room creation
    const roomConfig = getRoomConfig(defaultConfig);
    roomConfig.url = $("#url").val();
    roomConfig.roomName = "ROOM1-" + createUUID(4);
    roomConfig.nickname = "User1" + createUUID(4);
    // clean status display items
```

```
    setStatus(state.statusId(), " ");
    setStatus(state.errInfoId(), " ");
    // clean bitrate display item
    $("#" + state.currentStateId()).val("");
    // connect to server and create a room if not
    try {
        const session = await sfu.createRoom(roomConfig);
        // Set up session ending events
        session.on(constants.SFU_EVENT.DISCONNECTED, function () {
            onStopClick(state);
            onDisconnected(state);
            setStatus(state.statusId(), "DISCONNECTED", "green");
        }).on(constants.SFU_EVENT.FAILED, function (e) {
            onStopClick(state);
            onDisconnected(state);
            setStatus(state.statusId(), "FAILED", "red");
            if (e.status && e.statusText) {
                setStatus(state.errInfoId(), e.status + " " + e.statusText,
"red");
            } else if (e.type && e.info) {
                setStatus(state.errInfoId(), e.type + ": " + e.info, "red");
            }
        });
        // Connected successfully
        onConnected(state, pc, session);
        setStatus(state.statusId(), "ESTABLISHED", "green");
    } catch (e) {
        onDisconnected(state);
        setStatus(state.statusId(), "FAILED", "red");
        setStatus(state.errInfoId(), e, "red");
    }
}
```

5. Starting the channel bandwidth test after session establishing

code

The `onConnected()` function:

- sets up Stop button click actions
- subscribes to room error events
- calls the testing function

```
const onConnected = function (state, pc, session) {
    state.set(pc, session, session.room());
    $("#" + state.buttonId()).text("Stop").off('click').click(function () {
        onStopClick(state);
    }).prop('disabled', false);

    $('#url').prop('disabled', true);
    $("#" + bitrateTestState.durationId()).prop('disabled', true);
    // Add errors displaying
    state.room.on(constants.SFU_ROOM_EVENT.FAILED, function (e) {
        setStatus(state.errInfoId(), e, "red");
```

```
        onStopClick(state);
    }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
        onOperationFailed(state, e);
    }).on(constants.SFU_ROOM_EVENT.ENDED, function () {
        setStatus(state.errInfoId(), "Room " + state.room.name() + " has
ended", "red");
        onStopClick(state);
    }).on(constants.SFU_ROOM_EVENT.DROPPED, function () {
        setStatus(state.errInfoId(), "Dropped from the room " +
state.room.name() + " due to network issues", "red");
        onStopClick(state);
    });
    startBitrateTest(state);
}
```

6. Channel testing

`SFURoom.join()`, `BitrateTest.setListener()`, `BitrateTest.test()` code

The `startBitrateTest()` function:

- joins SFU room and negotiates WebRTC connection
- gets access to `BitrateTest` object
- adds onStateChange event handler function to `BitrateTest` object
- launches the channel testing
- displays the testing results

```
const startBitrateTest = async function (state) {
    if (state.room) {
        await state.room.join(state.pc, null, {});
        const stateSelector = $("#" + state.currentStateId());
        stateSelector.attr("style", "display:inline-block;margin-left:
10px");
        try {
            const bitrateTest = state.room.getBitrateTest();
            state.setBitrateController(bitrateTest);
            bitrateTest.setListener({
                onStatusUpdate(bitrateKbps) {
                    stateSelector.text("Current bitrate: " + bitrateKbps + "
kbps");
                }
            });
            bitrateTest.test($("#" +
bitrateTestState.durationId()).val()).then((bitrateKbps) => {
                stateSelector.text("Test is finished, last measured bitrate:
" + bitrateKbps + " kbps");
                state.setBitrateController(null);
                onStopClick(state);
            });
        } catch (e) {
            if (e.type === constants.SFU_ROOM_EVENT.OPERATION_FAILED) {
                onOperationFailed(state, e);
```

```
            } else {
                console.error("Failed to start bitrate test: " + e);
                setStatus(state.errInfoId(), e.name, "red");
                onStopClick(state);
            }
        }
    }
}
```

7. Test stopping

`BitrateTest.stop()` code

```
const stopBitrateTest = function (state) {
    const controller = state.getBitrateController();
    if (controller) {
        controller.stop();
    }
}
```

8. Start click action

`connect()` code

The `onStartClick()` function:

- validates input fields
- calls the `connect()` function

```
const onStartClick = function (state) {
    if (validateForm("connectionForm", state.errInfoId())) {
        $("#" + state.buttonId()).prop('disabled', true);
        connect(state);
    }
}
```

9. Stop click action

`Session.disconnect()` code

The `onStopClick()` function:

- stops the test if running
- disconnects the Websocket session

```
const onStopClick = async function (state) {
    if (state.isConnected()) {
        stopBitrateTest(state);
        await state.session.disconnect();
```

```
            onDisconnected(state);
        }
    }
```

10. Websocket session disconnection action

code

The `onDisconnected()` function:

- sets up `Start` click action
- enables `Server url` and `Test duration` fields access

```
const onDisconnected = function (state) {
    state.clear();
    $("#" + state.buttonId()).text("Start").off('click').click(function () {
        onStartClick(state);
    }).prop('disabled', false);
    $('#url').prop('disabled', false);
    $("#" + bitrateTestState.durationId()).prop('disabled', false);
}
```