Bandwidth Connector

Developer's Guide

Bandwidth

CSC 492 Team 10

Nicholas Bragdon
Andrew Ferko
Nathan Fuchs
Brandon Walker

North Carolina State University Department of Computer Science

December 14, 2013

Table of Contents

1 Introduction

The following document contains all knowledge needed to use and expand BWC's APIs and major functional files.

2 Server API

2.1 Websocket API

The following section contains the Websocket API calls used for BWC. The general format used in all messages to and from the server include a "type" and "message" that is used to route and contain data, respectively, within the system. All calls use the '/websocket' URI as configured in Server.js,

2.1.1 System Calls

The following Websocket API calls are used to interact the system in order to register or retrieve user availability statuses.

2.1.1.1 Connect <connect>

The purpose of the <connect> api is to act as a login system for websockets. The call establishes the user's presence as available to those who wish to call them, as well as those who retrieve their status via <checkUserStatus> (2.1.1.3).

NOTE - There is no Disconnect API call, as the user is automatically disconnected once their socket has closed as covered in 2.1.1.4.

Example Request:
{"type" : "connect", "message" : {"cookie" : "sessionID=0"}}

2.1.1.2 Switch to Busy <busy>

The purpose of the <busy> api is to manually mark a user as busy within the system. The call establishes the user's presence as busy to those who wish to call them, as well as those who retrieve their status vis <checkUserStatus> (2.1.1.3).

NOTE - For users receiving calls via <start_call>, this is not necessary as their "room" will have 2 occupants and will be marked as busy automatically.

Example Request:
{"type" : "busy", "message" : {"cookie" : “sessionID=0”}}

2.1.1.3 Check User Status <checkUserStatus>

The purpose of <checkUserStatus> is to retrieve the status of a user within the system. Once requested, the server will send a response back via websockets that includes the original request with an additional “status” field in the message.

Status codes: 0 = offline, 1 = busy, 2 = available

Example Request:
{"type" : "checkUserStatus", "message" : {"user" : “bpwalker@ncsu.edu”}}

Example Response:

```
{
        "type" : "checkUserStatus",
        "message" : {
                "user" : data.message.user,
                "status" : 0
        }
}
```

2.1.1.4 Socket Disconnect

Unlike the <connect> message (2.1.1.1), there is no “disconnect” message for the server. Instead, after a socket connection is closed, the server removes the user’s busy status (if busy) and turns the user’s availability status to offline.

NOTE - If the user was currently in the call, all other members in the call will receive the following message:
{"type" : "disconnect", "message" : {"disconnect" : "disconnect"}}

2.1.2 Browser to Browser

The following Websocket API calls are used to interact with the system to establish Browser to Browser WebRTC connections.

2.1.2.1 Start Call <start_call>

The following Websocket API call is used to initiate a Video WebRTC chat from “remoteUserName” to another user, denoted as “user” and their email. The request either results in the message being passed to the intended user, or a response back containing one of several possible failure conditions.

NOTE - “remoteUserName” can be either be of JSON format:

- **{"name" : "anonymous"}** (for anonymous calls)

or a JSON containing the cookie for the user:

- **{"cookie" : "sessionID=0"}** (for user to user calls).

NOTE - "type":"offer" is manually added to the sdp information, so make sure that only the sdp information is being sent in the "sdp" field.

Example Request:

```
{
        "type" : "start_call",
        "message": {
                "start_call" : {
                        "sdp": "...",
                        "type":"offer"
                },
                "user" : "bpwalker@ncsu.edu",
                "remoteUserName" : {
                        "name" : "anonymous"
                }
        }
}
```

Example Failure Responses:

```
{
        "type" : "busy_notification",
        "message" : {
                "call_notification" : "Sorry, this user is currently busy. Please try
                again later."
        }
}

{
        "type" : "offline_notification",
        "message" : {
                "call_notification" : "Sorry, this user is offline. Please try again
                later."
        }
}

{
        "type" : "dne_notification",
        "message" : {
                "call_notification" : "Sorry, this user does not exist. Please try again
```

```
                later."
        }
}
```

2.1.2.2 Accept Call <accept_call>

The following Websocket call simply transmits the accepting users SDP information to the calling party who initiated the request from <start_call> (2.1.2.1).

NOTE - “type”:”answer” is manually added to the sdp information, so make sure that only the sdp information is being sent in the “sdp” field.

Example Request:

```
{
        "type" : "accept_call",
        "message" : {
                "accept_call" : {
                        "sdp": “...”
                        "type": "answer"
                }
        }
}
```

2.1.2.3 Decline Call <decline>

The following Websocket API call simply sends a decline event and message to the calling party who initiated the request from <start_call> (2.1.2.1).

Example Request:

```
{"type" : "decline", "message" : {"decline" : “reason”}}
```

2.1.2.4 Hangup Call <hangup>

The following Websocket API call ends a call in progress by forwarding the message to the other user and removing any busy status for that messenger.

NOTE - Hangup does not necessarily need to be called by all users, as if the user disconnects as covered in 2.1.1.4, the same process as <hangup> occurs.

Example Request:

```
{
        "type" : "hangup",
        "message" : {
                "hangup" : "hangup"
```

```
        }
}
```

2.1.2.5 Send WebRTC Candidate <candidate> *Deprecated*

The following Websocket API call is currently no longer in use, but is intended to support trickle candidates following initial WebRTC SDP exchange in <start_call> (2.1.2.1) and <accept_call> (2.1.2.2). The message is simply forwarded to the other caller in the room as to be handled by their browser.

NOTE: Since there is no current use of the call, there is no example to be shown. Again, any information sent via this Websocket API is simply forwarded to the other participant in the call in the exact format sent to the server.

2.1.3 Browser to Phone

The following Websocket API calls are used to interact with the system to establish Browser to Phone WebRTC connections via the SPIDR API provided by Bandwidth.

2.1.3.1 Phone Call <phoneCall>

The following Websocket API call is used to initiate a phonecall to a remote user while they are offline. The call starts a priority based walkthrough of the remote user's configured phone numbers (if any), and attempts to call each on a 15 second timer. If the user was not reached, a response is indicated by the server. This call is functionally congruent to <start_call> (2.1.2.1), however it does not support anonymous calling.

NOTE - Once a call has started, the server will notify the caller of 3 possible messages: **Accept Phone Call**, **Phone Call Failed**, **Hangup** (if in-call). These are covered below in more detail.

Example Request:

```
{
        "type" : "phoneCall",
        "message":{
                "sdp" : "...",
                "cookie" : "sessionID=0",
                "user" : "bpwalker@ncsu.edu"
        }
}
```

Example Failure Response:

```
{
```

```
        "type" : "phonecall_not_reached",
        "message" : {
                "phonecall_not_reached" : "phonecall_not_reached"
        }
}
```

Accept Phone Call - “accept_phonecall”
This message is sent to the user once the remote phone has answered. The “accept_call” variable contains the sdp information with type to use for setting up WebRTC on the browser. “user” contains the calling user’s information, including their email and alias (the number calling the remote phone), as well the session for the call. The session is used for <abort_phonecall> (2.1.3.3), <decline_phonecall> (2.1.3.5), and <hangup_phonecall> (2.1.3.6).

Example:

```
{
        "type" : "accept_phonecall",
        "message" : {
                "accept_call" : {
                        "sdp": “...”,
                        "type":"answer"
                },
                "user" : {
                        "email" : “bpwalker@ncsu.edu”,
                        "alias" : “8042233016”,
                        "session" : “...”
                }
        }
}
```

Phone Call Failed - “phonecall_failed”
This message is sent to the user in the case that the call failed. This can be for a number of reasons, but most likely indicated a non-existent or malformed number.

Example:

```
{
        "type" : "phonecall_failed",
        "message" : {
                "phonecall_failed" : "phonecall_failed"
        }
}
```

Hangup - “hangup”
This message is sent while a phone call is active, and works exactly the same as <hangup> in 2.1.2.4.

Example:

```
{
        "type" : "hangup",
        "message" : {
                "hangup" : "hangup"
        }
}
```

2.1.3.2 Start Phone Call <start_phoneCall>

The following Websocket API call is used to initiate a phone call to an arbitrary 10 digit phone number. This call will continue for as long as the Bandwidth SPIDR API allows, or until the user aborts the call <abort_phonecall> (2.1.3.3). This call is functionally congruent to <start_call> (2.1.2.1), however it does not support anonymous calling.

NOTE - Once a call has started, the server will notify the caller of 3 possible messages: **Accept Phone Call**, **Phone Call Failed**, **Hangup** (if in-call). These are covered below in more detail.

Example Request:

```
{
        "type" : "start_phoneCall",
        "message": {
                "sdp" : “...”,
                "cookie" : “sessionID=0”,
                "number" : “8042233016”
        }
}
```

Accept Phone Call - “accept_phonecall”
This message is sent to the user once the remote phone has answered. The “accept_call” variable contains the sdp information with type to use for setting up WebRTC on the browser. “user” contains the calling user’s information, including their email and alias (the number calling the remote phone), as well the session for the call. The session is used for <abort_phonecall> (2.1.3.3), <decline_phonecall> (2.1.3.5), and <hangup_phonecall> (2.1.3.6).

Example:

```
{
```

```
        "type" : "accept_phonecall",
        "message" : {
                "accept_call" : {
                        "sdp": "...",
                        "type":"answer"
                },
                "user" : {
                        "email" : "bpwalker@ncsu.edu",
                        "alias" : "8042233016",
                        "session" : "..."
                }
        }
}
```

Phone Call Failed - "phonecall_failed"
This message is sent to the user in the case that the call failed. This can be for a number of reasons, but most likely indicated a non-existent or malformed number.

Example:

```
{
        "type" : "phonecall_failed",
        "message" : {
                "phonecall_failed" : "phonecall_failed"
        }
}
```

Hangup - "hangup"
This message is sent while a phone call is active, and works exactly the same as <hangup> in 2.1.2.4.

Example:

```
{
        "type" : "hangup",
        "message" : {
                "hangup" : "hangup"
        }
}
```

2.1.3.3 Abort Phone Call <abort_phonecall>

The following Websocket API call is used to stop calling a phone number after a <phoneCall> (2.1.3.1) or <start_phoneCall> (2.1.3.2) request was made.

Example Request:

```
{
        "type" : "abort_phonecall",
        "message" : {
                "cookie" : “sessionID=0”
        }
}
```

2.1.3.4 **Accept Phone Call** <accept_phonecall>

This Websocket API call is sent to the server to exchange the WebRTC SDP information after an inbound call from a phone comes to the browser through the “start_phonecall” Websocket event from the server.

First step - Accept Phone Call to Browser

The following “start_phonecall” JSON is sent from the server to signal an incoming call. The format and description is similar to the “accept_phonecall” state in <phoneCall> (2.1.3.1) and <start_phoneCall> (2.1.3.2):

```
{
        "type" : "start_phonecall",
        "message" : {
                "start_phonecall" : {
                        "sdp": “...”,
                        "type":"offer"
                },
                "user" : {
                        "alias" : “8042233016”,
                        "session" : “...”
                },
                "callingNumber" : “8042232928”
        }
}
```

Second step - Accept Phone Call to Server

After the user accepts the incoming call, <accept_phonecall> is called to exchange their WebRTC SDP information with the phone in the “accept_phonecall” variable.

NOTE - the user variable sent to the browser during the first step, is reused in the second step, meaning only the SDP in the accept_phonecall variable must be added.

```
{
"type" :
        "accept_phonecall",
        "message" : {
                "accept_phonecall" : “<SDP>...”,
                "user" : {
                        "alias" : “8042233016”,
                        "session" : “...”
                }
        }
}
```

2.1.3.5 Decline Phone Call <decline_phonecall>

The following Websocket API is called by a user receiving an inbound call in order to decline it. The “user” variable is simply a reference to the user object that is passed along with the first step, “start_phonecall”, message outlined in 2.1.3.4.

Example Request:

```
{
        "type" : "decline_phonecall",
        "message" : {
                "user" :  {
                        "alias" : “8042233016”,
                        "session" : “...”
                }
        }
}
```

2.1.3.6 Hangup Phone Call <hangup_phonecall>

The following Websocket API is called by a user during a phone call that was answered via <accept_phonecall> in 2.1.3.4. The “user” variable is simply a reference to the user object that is passed along with the first step, “start_phonecall”, message outlined in 2.1.3.4.

Example Request:

```
{
        "type" : "hangup_phonecall",
        "message" : {
                "user" :  {
                        "alias" : “8042233016”,
                        "session" : “...”
                }
```

```
        }
}
```

2.2 RESTful API

The following sections include the REST APIs used for BWC.

2.2.1 Users

The following REST APIs are used to Add, Edit, and Remove users within the BWC system.

2.2.1.1 POST /user

This api adds a user to the database after ordering an alias number from the IRIS API. If there are no available alias numbers or the email already exists, an error code is sent back as a response. A “1” signifies that a user with that email address already exists in the database, while a “2” denotes that there are no aliases to assign to users and that new aliases need to be added to the database.

Example body:

```
{
        email: "bpwalker@ncsu.edu",
        firstName: "Brandon",
        lastName: "Walker",
        password: "password",
        "contactDevices": {}
}
```

2.2.1.2 PUT /user

This API call updates a user that currently exists in the database. The format of the body needs the same information fields as the GET /user calls . In the case that the email in the JSON provided does not match a currently existing email in the database, the system will return a “1” in the return body of the call.

Example body:

```
{
        "_id":"dfd1f118-9b63-4830-945b-cca35bd0a33f",
        "email":"bpwalker@ncsu.edu",
        "firstName":"Brandon",
        "lastName":"Walker",
        "password":"password",
        "alias":"8042233019",
        "contactDevices": {
                "Cell": {
```

```
                "deviceType":"phone",
                "deviceValue":"5555555555",
                "priority":0
                }
        },
        "sessionID":"f96uons45ifutkwqxwuki"
}
```


2.2.1.3 DELETE /user/sessionID/<sessionID>

This API call deletes a user from the database that has the current sessionID.

2.2.1.4 GET /user/userId/<userId>

This API call retrieves a user from the database by the id specified in the URI which maps to the id for the user in MongoDB. In order to retrieve all users from the system except yourself, include a Cookie in the request header with the sessionID.

2.2.1.5 GET /user/sessionID/<sessionId>

This API call retrieves a user from the database via the sessionID specified in the URI.

2.2.1.6 GET /user/email/<email>

This API call retrieves a user from the database via the email specified in the URI.

2.2.1.7 POST /user/addFriend

This API call adds a friend to the user's profile. For this call, include a Cookie in the request header with the sessionID. In the case where the sessionID could not be found for the request, a "1" is returned in the response body.

Example Body:
{"friendEmail": ["bpwalker@ncsu.edu"] }

Example Success Response:
{status: "success", message: "Friend Successfully Added"}

Example Failure Response:
{status: "error", message: "Friend Already Exists"}

2.2.1.8 POST /user/deleteFriend

This API call removes a friend from the user's profile. For this call, include a Cookie in the request header with the sessionID. In the case where the sessionID could not be found for the request, a "1" is returned in the response body.

Example Body:

{“friendEmail”: [“bpwalker@ncsu.edu”] }

Example Success Response:
{status: "success", message: "Friend Successfully Deleted"}

Example Failure Response:
{status: "error", message: "Cannot delete friend that doesn't exist"}

2.2.2 Aliases

The following API calls are related to Aliases in BWC. An alias is a phone number ordered from Bandwidth and stored in a pool to assign to new user accounts as they are created.

2.2.2.1 POST /alias

This API is used to add aliases after they have been ordered from Bandwidth’s IRIS API and registered within the Genband Gateway. A static password is required in all requests and can be configured within the server if needed in the future. The API supports adding multiple aliases at a time, as shown in the example JSON in the “aliases” variable. Syntactically valid aliases include the alias number itself, the password configured in SPIDR for the alias, and an availability flag (Either “1” or “0”) depending on if the alias should be assigned to new users creating accounts.

Example Body:

```
{
        "adminPassword":"bwc1234",
        "aliases":[
                {"alias":"8042232838", "password":"bwc1234", "available":"1"},
                {"alias":"8042232839", "password":"bwc1234", "available":"1"}
        ]
}
```

2.2.3 Login

The following API is used for users to logging into their account.

2.2.3.1 POST /login

This API call sends the users credentials to the server and allows the user to get to the dashboard page if the credentials are verified.

Example Body:

```
{
        email: “bpwalker@ncsu.edu”
        password: “password”
```

}

3 Functional Walkthrough

3.1 Server.js

The following walkthrough is for Server.js and should contain a basic description of how the Server starts up and functions during operation.

3.1.1 Startup

Once the server starts up, the following steps are taken.

3.1.1.1 MongoDB is Started

The Vert.x module defined as “'io.vertx~mod-mongo-persistor~2.0.0-final'” in the project is started using the configuration settings in Config.js as “mongoDBConfig”.

3.1.1.2 Alias Subscription

After MongoDB has started, all the aliases in the database are pulled and then subscribed to using the SPIDR API call location in subscribeAlias(). This is done so that the phone call portion of the server will work as a current subscription is needed to receive call information such as inbound calls, remote calls being answered, remote calls hanging up, etc. The most important part about subscribeAlias() besides the actual subscription channel monitoring is that the subscription will only stay open for the amount of seconds as defined in Config.js as “subscribeExpireSeconds”. After this time expires and the websocket subscription closes, the subscription is deleted, and then reestablishes. In the case that the subscription max is hit (5 concurrent connections), a notification will appear in the server console and the server will try to reestablish the subscription every 5 seconds until a subscription is obtained.

3.1.1.3 Server Started

Here both the Vert.X HTTP and SockJS Websocket servers are created and started on the port defined in Config.js as “serverPort”. The Websocket uri is defined in the installApp method as ‘/websocket’. This can be changed, but must also be updated on any SockJS connection used on the site. An important note for this step is that a requestInterceptor and websocketInterceptor are used to handle inbound requests for the HTTP server and Websocket listener respectively. These are simple closures defined at the bottom of the Server.js file and follows a map of strings (URIs or Websocket types as defined in the

Websocket API) to functions that execute on the event.

3.1.1.4 Rooms

Rooms were the original idea put forth before the phone call system was added to BWC. These are simple JS objects that map a user socket (owner) to all other sockets in the room. Essentially the existence of a room for a user means they are online, and if another user is in there room, then that user is in a call and marked as “busy”. This paradigm was quickly outdated once the SPIDR api was added but still serves as a useful mechanism of linking sockets together and managing availability states.

3.2 Call.html

The Call.html page serves as the launching point for initiating calls to users or phones.

3.2.1 Page Initialization

When the page is first loaded and JQuery $.onReady() fires, the first event besides initializing visual JS elements, is to determine whether or not a user is anonymous. This is determined by the presence of a cookie session id in the user’s browser. Next the type of call and remote user is determined via the url which is either /call/user or /call/phone/number. Once the type of call is established, the SockJS connection to the server is opened and the rest of the visual JS elements are initialized.

3.2.2 SockJS onopen and Call Initialization

Once the SockJS connection has been established, two paths can be taken by the page depending on the call type.

In the case of a phone call (anonymous users can not make it to this point), intializePhoneCall() is called. Here audio and video are requested via getUserMedia() and once approved, begins the phone call process via the Websocket API defined in section 2.

In the case of a browser call, there are two possibilities. In the case of an anonymous user, a prompt is given to the user for a display name. Authenticated users automatically use their sessionID in their API calls and thus don’t need this information. Once this is collected or the session id is found, initialize() is called which requests audio and video via getUserMedia. Once approved, the call process begins via the Websocket API defined in section 2.

3.2.3 SockJS onmessage

Depending on the call flow as defined in 3.2.2, one of the following websocket

messages could be invoked via the server.

accept_call - Invoked after a browser call is accepted by a remote user. It reveals the video call modal so that the users can see each other, initializes some of the tools for it, and adds the remote users WebRTC SDP description information to begin the call.

accept_phonecall - Invoked after a remote phone accepts a call by the user. It reveals the video call modal, sets the remote description for the call. It also sets flags that are used to control the hangup flow.

candidate *DEPRECATED* - This method is not currently used, but is valid for trickle candidates that are sent after WebRTC information is exchanged.

hangup - This is invoked when a remote user hangs up. The message is repeated and the call is ended. A note here that when a phone call fails, it also sends a hangup message, however this signifies that the priority system is heading to the next number and thus the call is not ended.

disconnect - This is invoked when a remote user disconnects either via a dropped connection or leaving the page. It immediately ends the call.

decline - Notifies the user if their call was declined by the remote user.

offline_notification - Notifies the user if the remote user they were trying to reach is offline. In the case of an authenticated user, this prompts for the priority rotary system to start on the server.

dne_notification - Notifies the user that they attempted to call a user that does not exist.

busy_notification - Notifies the user that the remote user is currently in a call.

phonecall_failed - Notifies the user that their call failed, either due to a malformed number in the remote users configuration or the call dropped.

phonecall_not_reached - Notifies the user that the priority rotary system finished without successfully reaching the remote user.

3.3 Dashboard.html

The Dashboard.html page serves two main purposes. The first is to act as the homepage of BWC where users are given a directory of users, profile access, etc. However, it also serves as the contact point for receiving calls from users or

phones.

3.3.1 Page Initialization

When the page is first loaded and JQuery $.onReady() fires, the first event besides initializing visual JS elements, is to establish the SockJS connection to the server is opened and the visual JS elements are initialized.

3.3.2 SockJS onopen

Once the SockJS websocket connection is opened, the page sends the session id through the <connect> websocket API call (defined in Section 2) in order to register the user as online. Then the page initializes the user directory with the online status of all the users by sending a <checkUserStatus> request through websockets.

3.3.3 SockJS onmessage

After the SockJS websocket connection is established with the server, any of the following websocket messages could be triggered depending on the state.

start_call - Invoked when a remote user video calls. The initialize() function is called, setting up and displaying the receiving call page as well as notifying the user. The remote session description is set temporarily to handle after the user accepts/declines.

candidate *DEPRECATED* - This method is not currently used, but is valid for trickle candidates that are sent after WebRTC information is exchanged.

hangup - This is invoked when a remote user hangs up. The message is repeated and the call is ended.

disconnect - This is invoked when a remote user disconnects either via a dropped connection or leaving the page. It immediately ends the call.

accept_call - This is invoked as confirmation from the remote user that the WebRTC information was exchanged and the call has started. The call modal is displayed and the users can see each other.

checkUserStatus - This is invoked as the callback from a <checkUserStatus> event (Defined in Section 2). Here the resulting message is parsed and the user directory is updated appropriately.

start_phonecall - Invoked when a remote user initiates a phone call to the user. The session is stored, and the intializePhoneCall() function is called, setting up and displaying the receiving call page as well as notifying the user. The remote

session description is set temporarily to handle after the user accepts/declines.

---

4 Final Thoughts and Concerns

4.1 Security

Currently, one of the most overlooked parts of BWC is the complete lack of security in both server side validation, and over-exposed information.

4.1.1 Server Side Validation

Currently there is no validation for information entered in both User Registration and User Profile Editing. Not only does this mean that a user could potentially use the available user RESTful APIs to bypass client side validation, but cross-site scripting is currently possible as no information is validated before being inserted into MongoDB and served to others when viewing the profile page. Currently there are no obvious flaws for unauthorization with the Websocket APIs, as most require session validation, however messages passed between users are also unvalidated and could be maliciously exploited.

4.1.2 Lack of HTTPS and WSS

Out of development practicality, BWC does not currently use HTTPS or WSS. Ideally, HTTPS could be implemented to not only ensure user privacy when browsing, but to save choices on the site such as allowing video/audio from getUserMedia(). WSS would also be ideal to use as identifying session ids are sent across Websockets unencrypted and could be hijacked.

4.1.3 Over-exposed Information

Currently, when user information is retrieved via User RESTful API calls, such as when viewing user profile information or loading the directory, all information about the user is sent to client. This includes passwords, session ids, and private configuration settings. This is a tremendously large security risk and requires modification of what data is exposed in these API calls.

4.2 Hard coded credentials

Currently there are many hard coded credentials in BWC including:

- The password for the Alias RESTful API is bwc1234 and should be changed to an actual admin account in MongoDB.

- All sip phone calls API calls to SPIDR use “@bconn-1.oscar.ncsu.edu” ending for the “to” field in the calls. These should be configured either in the Config.js file or in MongoDB.
- The IRIS API used to order and disconnect aliases uses a static username and password that is hard coded into the server. This should be stored in either Config.js or MongoDB.

4.3 Unfinished Alias API

Currently the only Alias RESTful API is for adding aliases to the system. This means there is no easy or standard way to modify aliases, delete them, or otherwise manipulate them in the system. This can be tiresome and encumbering, especially in situations of database wipes where they must be repopulated. Ideally a better API (with a potential bootstrap method) would exist that would easy the burden on developers.