Evaluating Framework for Monitoring and Analyzing WebRTC Peer-to-Peer Applications

Marco Schindler, Christian von Harscher, Johannes Kinzig, Sergej Alekseev
Computer Science and Engineering
Frankfurt University of Applied Sciences
Frankfurt am Main, Germany
{mschindl|harscher|kinzig}@stud.fra-uas.de, alekseev@fb2.fra-uas.de

Abstract—Web Real Time Communication (WebRTC) technology becomes more and more popular, because it enables communication between web browsers and mobile applications without the need for plug-ins or other apps. This makes it interesting for a wide range of use cases such as call center or online webshop. Our expectation is, that in the near future a lot of commercial and free applications based on this technology will be developed.

This paper presents an environment for prototyping of p2p WebRTC based applications, monitoring and analysis of their behaviour. The basic idea of this environment is to provide a possibility to visualize connection graphs containing connected peers of a WebRTC session and to collect statistics for monitoring and analysis.

Index Terms—peer-to-peer (P2P), Web Real Time Communication (WebRTC), Connection Graph, Monitoring

I. INTRODUCTION

Web Real-Time Communication (WebRTC) [1] is a new standard and industry effort that extends the web browsing model. It provides the ability of putting real-time communication capabilities such as audio, video and data communications into web browsers without the need of installing additional software or plug-ins. Undoubtedly, one of the most common and frustrating issues of companies, who offer products that make use of real-time communication, are missing diagnostic tools for testing and troubleshooting the WebRTC connectivity. The WebRTC services are typically p2p applications and thus the troubleshooting process is even more difficult.

In this paper we present a framework for prototyping of WebRTC based applications, monitoring the WebRTC connectivity by collecting the analysis of various statistics. Furthermore the framework visualizes all gathered information including the connection states, statistic and geographical location of involved peers.

II. RELATED WORKS

Considerable prior work has been done on the subject of WebRTC state analyzing in [2]. We are motivated by the fact that this cited work described how to analyze WebRTC connections by looking at therefor collected events. These events are generated by a state change of a WebRTC peer concerned in the connection process. The essential approach of this cited work is rest upon finite-state machines in accordance with the WebRTC specification. We are using this state analyzing concept to monitor and evaluate WebRTC sessions. Additionally we gather various statistics including the peer topology.

It is important to mention that an API named getStat() for getting WebRTC statistics exist. This API can be used within JavaScript code to return several WebRTC statistics such as round-trip-time, packets-send, packets-lost etc.

The API is used by several projects (on github, etc.) which claim to be "analyzing libraries" for WebRTC. These libraries mostly lack the infrastructure component to collect and maintain the statistics at a local place for error-handling and analyzing. An example for such a library can be found under https://github.com/muaz-khan/getStats.

There exist one more framework named "callstats.io" which offers the infrastructure to collect and analyze the statistics in one single place but this is a proprietary framework. They offer client libraries for integration in the application. The library then takes care for transmitting the WebRTC statistics to the callstats.io’s servers. Customers then can access their gathered statistics through a web interface. (cf. [4])

In contrast to the already existing possibilities offers our solution a complete framework (based on open source tools and libraries) which can easily be installed on a linux server. The framework does not only collect the statistics, it additionally collects the WebRTC events generated by the clients.

III. SYSTEM OVERVIEW AND GENERAL STRUCTURE OF THE PROPOSED FRAMEWORK

Figure 1 shows the overall system architecture of the proposed framework. All software components are running in a linux server environment. Nginx is used as a proxy to the backend applications such as application and signaling server. The application server includes a PHP and MySQL installation to run the Moodle Platform (see section III-A) and the WebRTC Monitor (see section VI). For the server side implementation of P2P algorithms two signaling services are provided, one for running the algorithms written in Java and one for running the algorithms written in JavaScript (see section III-C for further details).

A. Application Server

The basis for the evaluating and benchmarking framework is the Moodle eLearning Platform [5]. The reason for implementing the benchmarking framework partly as a Moodle Plugin is
the easy administration of users and sessions. The evaluation of the different P2P algorithms was done with real users all over the world. Therefore it was necessary to have a simple solution for adding and maintaining users and "classrooms".

The Moodle Plugin basically provides the client functionality for the P2P algorithm framework and the WebRTC Monitor application, which is responsible for displaying the collected data and statistics.

Our framework provides several interfaces for implementing P2P algorithms in Java and JavaScript.

B. WebRTC Signaling Server

WebRTC enables peer to peer communication between browsers, but also needs a mechanism to coordinate communication. This process is known as signaling. Signaling methods and protocols is not part of the WebRTC API [1].

There are several commercial cloud messaging platforms that provides an API for WebRTC signaling such as Pusher 1, Kaazing ², PubNub ³, and vLine ⁴. There are also several open source solutions based on Socket.io [7] such as webRTC.io, easyRTC and Signalmaster.

We decided to implement our own WebSocket based signaling server running in node.js [6] rely on channels. A Channel is a namespace which allows broadcasting messages to make sure the signaling data is exchanged among relevant users.

C. P2P Algorithm Server - RSWebSocket

In our proposed solution, two different algorithm servers are implemented, but both are based on the Web-socket principle. They differ in the programming language, the way how they handle the communication and they accept different types of incoming requests. One is implemented in Java and the other one in JavaScript.

IV. SOFTWARE INTERFACES AND PROTOCOL DESCRIPTION

A. Signaling Server for Java based algorithms - RSWebSocket

For evaluating algorithms written in Java the RSWebSocket has an interface called IP2PJoinAlgorithm which declares the methods needed for implementing a P2P joining algorithm. The interface can be seen in the following listing:

```java
IP2PJoinAlgorithm.join
IP2PJoinAlgorithm.peerLeave
IP2PJoinAlgorithm.getSearchTreeGraph
IP2PJoinAlgorithm.getSearchTreeRoot
IP2PJoinAlgorithm.getP2pTreeGraph
IP2PJoinAlgorithm.getPeer
```

The `join` method takes two arguments, as first argument an object of the type TopologicalCoordinate and as second argument peerId of type int.

The TopologicalCoordinate is an object which consists of the following data and types:

- TopologicalCoordinate.continent
- TopologicalCoordinate.country
- TopologicalCoordinate.city
- TopologicalCoordinate.ip1xxx
- TopologicalCoordinate.ip12xx
- TopologicalCoordinate.ip123x

The returned element by the `join` method is an object of the type JoinPeerResult, which holds all the information necessary for a peer to be attached to or included in the peer tree structure. The JoinPeerResult object holds an ArrayList which contains Reconnect objects.
The Reconnect object has two integer attributes, peerId and reconnectId. In case a peer leaves the session or a new peer is inserted between two peers, this reconnect list is transmitted to the clients. The clients then do a reconnect and update the tree structure because peers were added or removed from the recent structure. Additionally JoinPeerResult contains an integer object called parentJoinId, which holds the peerId where the new peer should connect to.

The third method is named peerLeave, it takes the peerId as an input argument and returns an object of the type PeerLeaveResult. The peerId is the peer which is going to be disconnected from the tree structure. This peer's children then need to be reconnected to the remaining peers. The type PeerLeaveResult holds an array list with Reconnect objects (as described above).

The fourth method is called getSearchTreeGraph, it takes no arguments and returns an object of the type SearchTreeGraph. The SearchTreeGraph object represents the whole search tree and implements several methods such as getAsDot which allows to print the search tree in the dot language format.

The fifth method is named getSearchTreeRoot, it does not take any arguments and returns an object of the type TVertex. The TVertex class is used to represent the nodes in the search tree.

The sixth method is called getP2pTreeGraph and returns an object of the type P2PTreeGraph, it does not take any arguments as input. The P2PTreeGraph class is used to represent the P2P tree.

The last method is called getPeer and takes the peerId as an input. The return type is an object of the PeerVertex class, it represents a node in the P2P topology.

B. Message Protocol Description - Implementing custom signaling server

When implementing a custom signaling server to work with the framework, the described messages in this section should be taken into account. Developing a custom signaling server becomes important when trying to evaluate algorithms written in another language than Java or JavaScript.

The signaling mainly takes place by exchanging messages between the client (session initiator, included as demo.js into the Moodle Plugin) and the signaling server. These messages are based on the JSON data format. JSON is widely used and can easily be processed by major programming languages and frameworks.

Generally the application is using two different JSON messages, one for each communication direction:
signaling server -> session initiator
signaling server <- session initiator

The important data fields for session initiator -> signaling server can be seen in this listing:
type: NewSession: [sessionId, algorithmID]
    join: [peerId]
onLeave: [peerId]

When a new websocket connection is opened by the session initiator the signaling server checks for the message type. In case it is NewSession a new session is internally added to the sessions map and the sessionId and algorithmID is taken from the JSON message. The algorithmID is only important for the testing architecture where you can chose between several algorithm implementations.

In case the message type is join the peerId is taken from the JSON message and the algorithm's join method is called. In case the message type is onLeave the peerId is taken from the JSON message to disconnect this peer from the structure and reconnecting the other peers which have been connect to this peer.

The data fields used for communicating in the opposite direction: signaling server -> session initiator (demo.js) are described in the following listing:
type: join_return:
    [peerId, connectTo, reconnectList]
LeaveResult: [peerId, reconnectList]
    reconnectList[peerId, reconnectId]

When the session initiator requests a join the signaling server answers with a join_return which contains the peerId, connectTo and reconnectList. The peerId is the current peer, connectTo is the peerId which the current client should connect to and the reconnectList contains the peers which need to reconnect in case the new peer is attached in between two other already connected peers. The reconnectList contains name value pairs of peerId:reconnectID.

C. JavaScript client side implementation

The client side implementation for the algorithm evaluating framework is part of the Moodle eLearning plugin (cf. section III-A). It is a JavaScript class for interacting with one of the signaling servers mentioned in section III-C. The class is implemented with the shown methods in the listing below:
P2PAlgorithm.prototype.init
P2PAlgorithm.prototype.join
P2PAlgorithm.prototype.onLeave

With these methods a point of intersection between the client and the algorithm is defined. Through this construction it is possible to use distributed algorithms running in the browser of each client as well as centralized algorithms running on the server. The communication of these algorithms is done over a WebSocket connection with a specific message format (cf. IV-B).

The init method is initializing the algorithm instance. The join method is called by clients entering the WebRTC session. The join method communicates the client's session-ID to the session initiator. The session initiator is then requesting the position for the new joining client from the signaling server. The client is joined after the session initiator receives the position information and extends the tree structure by the client's ID.
If a session is ending the `onLeave` method is called automatically by the corresponding client. This happens even if the browser window is closed by the user. The client is sending the `onLeave` information to the session initiator. The session initiator is then requesting the reconnection list from the server by sending the client-ID. On response the reconnection list is processed by the session initiator and the reconnect is executed.

V. GATHERING EVENTS AND STATISTICS

For the analysis of the WebRTC sessions the WebRTC states are gathered. The current system state of each WebRTC connection is represented through the states of the three following Finite State Machines. According to [2, section III] these finite state automatons are the signaling state machine, gathering state machine and connection state machine. For persisting the systems’ state we are using a MySQL database in combination with the WebRTC Analyzer mentioned in [2, abstract].

A. Raw events

The so called raw events are the WebRTC events reflecting the state of the connection. Not only the three states for signaling, gathering and connection finite-state automatons are displayed, but also the peer-id, the room-id of the virtual classroom, the session-id, the user-id and the method which fired the event. The given user-id is utilized for creating a link to the corresponding Moodle user. An database excerpt with sample events are displayed in table I.

B. WebRTC Statistics

The WebRTC traffic is transmitted over the best-effort IP network, which is inherently vulnerable to network congestion. Audio, video and data packets can be lost during transmission and congestion also increases the network latency and the delay of the transported packets. High packet loss and long delays affect the quality of the WebRTC media stream.

A statistics API of the WebRTC standard provides features to return peer connection stats. These statistics can be used to guarantee the best possible quality of WebRTC calls. The JavaScript library used in this environment, enables performance monitoring features for audio and video calls in WebRTC-based endpoints. By using this library, the following statistics of a client can be gathered.

```javascript
audio_video_out_RoundTripTime,
audio_video_out_packets_sent,
audio_video_out_bytes_sent,
audio_video_out_packets_lost,

audio_video_in_target_delay_ms,
audio_video_in_jitter,
audio_video_in_discarded_packets,
audio_video_in_packets_received,
audio_video_in_bytes_received
```

Table II shows the statistics gathered for session 10. This table shows an excerpt generated by the monitor (cf. VI) and displays values necessary to analyze the session’s quality.

C. Geolocation

Additionally to the gathered events described section V-A and V-B the geographical position of each peer is determined. When a peer enters a session the browser requests the current geographical location using the HTML5 geolocation service. This geolocation service replies with the geographical coordinates of the current location. The client then uses the Google Maps Geolocation API to translate the retrieved coordinates into city, and country. As soon as the client has received this information, it stores them in the database.

VI. MONITOR

Our gathered WebRTC events can be evaluated with the WebRTC Monitor module inside of Moodle. The module must be enabled for each Moodle course in which the virtual classroom is used as an activity module. The monitor features a selection dialog for each classroom. After selecting a classroom the WebRTC sessions are displayed. These sessions are adjustable in the settings of a classroom. The start and end time as well as the session-id are displayed in the session overview. In addition to that information buttons for various actions are displayed. These buttons provide links to the raw events, to the map, to the location and bandwidth information and to the browser based WebRTC statistics. A screenshot of the monitor can be seen in figure 3.

VII. CONCLUSION

Since WebRTC does not provide a standardized opportunity to monitor and evaluate sessions, this article describes a framework, introducing a solution for the missing analyzing
TABLE I
DATABASE EXCERPT WITH WebRTC RAW EVENTS GENERATED BY THE MONITOR

<table>
<thead>
<tr>
<th>UserID</th>
<th>PeerID</th>
<th>RoomID</th>
<th>Timestamp</th>
<th>sid</th>
<th>Eventhandler</th>
<th>Signaling State</th>
<th>Gathering State</th>
<th>Connection State</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>1443149221164_587393171153963</td>
<td>61</td>
<td>1443149276</td>
<td>10</td>
<td>start</td>
<td>stable</td>
<td>new</td>
<td>new</td>
</tr>
<tr>
<td>18</td>
<td>1443149221164_587393171153963</td>
<td>61</td>
<td>1443149276</td>
<td>10</td>
<td>onsighalingstatechange</td>
<td>have-local-offer</td>
<td>new</td>
<td>new</td>
</tr>
<tr>
<td>7</td>
<td>1443149221164_587393171153963</td>
<td>61</td>
<td>1443149276</td>
<td>10</td>
<td>start</td>
<td>stable</td>
<td>new</td>
<td>new</td>
</tr>
</tbody>
</table>

TABLE II
DATABASE EXCERPT WITH WebRTC STATISTICS

<table>
<thead>
<tr>
<th>course</th>
<th>time_stamp</th>
<th>sid</th>
<th>userid</th>
<th>peer_id</th>
<th>video_out_rtt</th>
<th>video_out_packets_sent</th>
<th>video_out_packets_lost</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1443149221164</td>
<td>61</td>
<td>1443149276</td>
<td>10</td>
<td>start</td>
<td>stable</td>
<td>new</td>
</tr>
<tr>
<td>18</td>
<td>1443149221164_587393171153963</td>
<td>61</td>
<td>1443149276</td>
<td>10</td>
<td>onsighalingstatechange</td>
<td>have-local-offer</td>
<td>new</td>
</tr>
<tr>
<td>7</td>
<td>1443149221164_587393171153963</td>
<td>61</td>
<td>1443149276</td>
<td>10</td>
<td>start</td>
<td>stable</td>
<td>new</td>
</tr>
</tbody>
</table>

option. This framework includes different possibilities to assess past sessions (see section VI). There are still some slight improvements of the monitor necessary, such as implementing the option to show the peer graph of a particular session (VIII). But all in all it is an operative framework to find errors and problems in WebRTC applications.

VIII. FUTURE WORK

The next steps for the project consist of defining algorithms for automated event examination and graphical displaying. As described in section VI the monitor is able to show the gathered events (cf. V) as a table. Important information such as the tree structure or the state of a peer can be determined by looking at the events but for a fast and easy evaluation a graphical solution would be helpful. At the current state of the project, these graphs were generated manually by examining the database relations. Results can be seen in figure 4. The graph shows the tree structure of the participating peers and important information about the connection between the peers.

Therefore the next steps include defining an algorithm which is able to draw a graph displaying the participants as nodes and the connection direction as edges. Additionally the statistics can be included in the graph such as showing the round trip time for packets from one peer to another, showing the video and audio packets send from one client to another and showing the lost packets between the peers.

REFERENCES