Adgt.js: A Web Application Framework for Peer-to-Peer Location-Based Services

Giacomo Brambilla(✉), Michele Amoretti, and Francesco Zanichelli

Dipartimento di Ingegneria dell'Informazione,
Università degli Studi di Parma, Parco Area delle Scienze 181a, 43124 Parma, Italy
giacomo.brambilla@studenti.unipr.it,
{michele.amoretti,francesco.zanichelli}@unipr.it
http://dsg.ce.unipr.it

Abstract. Mobile applications are increasingly taking advantage of user geographic location to provide sophisticated Location-Based Services (LBSs). Unfortunately, most LBSs rely upon centralized infrastructures, with serious problems as regards user privacy. For this reason, the research community has proposed a number of decentralized protocols and studied their effectiveness and efficiency by means of simulations.

In this paper, we describe Adgt.js, a truly cross-platform, WebRTC-based implementation of the ADGT georeferenced peer-to-peer overlay scheme. Moreover, we present a concrete LBS example, realized with Adgt.js, to illustrate how simple and powerful such a framework is.

Keywords: WebRTC · Peer-to-peer · Location-based service

1 Introduction

In recent years, there has been a growing attention to Location-Based Services (LBSs), *i.e.*, services that take advantage of user geographic location, especially owing to the expansion of the smartphone and tablet markets. LBSs allow, for example, to locate people on a map, discover nearby social events or receive geolocalized alerts (such as warnings of traffic jams along the user route).

If, on the one hand, large IT companies such as Google and Facebook are pushing more and more their LBSs without worrying too much about user privacy, on the other hand, researchers are investigating to provide such services while preserving user privacy. In particular, various peer-to-peer (P2P) overlay schemes that enable completely decentralized LBSs have been presented [1,2].

These P2P protocols, in addition to safeguard privacy of users inasmuch the data are not in the hands of a single possibly untrustworthy company, support the realization of bottom-up LBSs, not requiring large and expensive infrastructures. Despite the many benefits of a P2P approach, often these solutions have been studied only in simulative environment and truly usable implementations have never been released.

In this paper we present a working implementation of the ADGT overlay scheme [2]. The objective behind the development is the software interoperability between all possible and heterogeneous devices, to make sure that the adoption is high. For this reason, we turned to real cross-platform technologies, such as WebRTC, WebSocket and JavaScript to build a framework that supports the development of P2P-based LBSs. To the best of our knowledge, our implementation is the first of its kind in the area of P2P protocols for LBSs.

The paper is organized as follows. Section 2 provides an overview of the ADGT overlay scheme. Web technologies adopted for the cross-platform implementation of ADGT are described in Sect. 3. In Sect. 4 we describe how the ADGT has been implemented and in Sect. 5 it is explained how to realize LBSs using the developed framework, with reference to a concrete example. Related work are presented in Sect. 6. Finally, in Sect. 7, we present our conclusions and future work.

2 Adaptive Distributed Geographic Table (ADGT)

ADGT is a location-aware P2P overlay scheme designed with the objective to fully take into account peer mobility [2]. What mainly characterizes ADGT is its particular data structure for the management of neighborhood, based on the idea that a peer should be directly connected to those peers from which it is most likely to obtain contents of its interest, using an adaptive topology that reacts to peers' movements.

In the ADGT overlay scheme, the distance between two peers is evaluated as the great-circle distance, which is the shortest distance between two points on the surface of a sphere, measured along the surface of the sphere itself.

The neighborhood of a geographic location is defined as the set of peers that are geographically close to that specific location. In other words, those peers which are located inside a given surrounding region.

In the ADGT, each peer stores a set of lists of neighbors, called GeoBucket, each list being sorted according to the distance from the center that the GeoBuckets have in common. Such lists are regularly updated in order to have the latest peers' positions. As shown in Fig. 1, the shape of GeoBuckets is elliptical, where both the semi axes of the ellipse depend on the velocity of the peer, *i.e.*, depend both on the direction and speed of the peer.

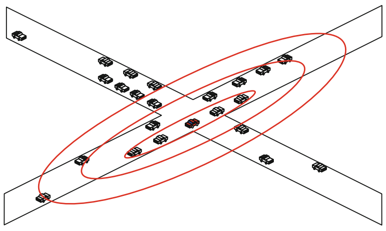

Fig. 1. ADGT GeoBucket example.

The idea behind such elliptical GeoBuckets is that the higher is the speed of the peer, the higher is the eccentricity of the ellipses: when the peer is stationary, its speed is 0 and the eccentricity of the ellipses is also 0, so the GeoBuckets are circular. On the other hand, when the peer reaches the maximum speed, the eccentricity of the ellipses is high, so the GeoBuckets have an elongated shape. Also, the direction of the semi-major axis coincides with the direction of movement of the peer.

3 Technologies

As the idea behind our implementation is the complete interoperability among devices as much as possible different – both from the hardware point of view, and in terms of installed software – we have turned to those technologies that constitute the Open Web Platform (OWP).[1] The OWP is a collection of open royalty-free Web technologies, such as HTML5 and JavaScript, developed by the World Wide Web Consortium (W3C) and other Web standardization bodies such as the Unicode Consortium, the Internet Engineering Task Force (IETF), and ECMA International, with the objective to obtain a platform that works on all browsers, operating systems and devices, without requiring any approvals or waiving license fees.

Although the standards of the Open Web Platform are at different maturity levels, and the development of most standards is still in progress, the web browser has become the main access interface to the Internet and has actually become synonymous with the Internet itself for a large portion of Internet users. While initially web browsers were designed only to display information provided by web servers, thanks to this standardization process, they are becoming the real cross-platform technology, being able to truly realize the "write once, run everywhere" unfulfilled promise of Java related technologies.

Among the many technologies that are encompassed under the umbrella of Open Web Platform, one of the most interesting definitions the W3C has worked on is the Web Real-Time Communication (WebRTC[2]), a free and open API that supports browser-to-browser applications for voice calling, video chat, and P2P data sharing without the need of either internal or external plugins. Its aim is to enable rich, high quality, real-time applications to be developed for browsers, mobile platforms, and IoT devices, allowing them all to communicate via a common set of protocols. WebRTC, WebSocket API[3], Geolocation API[4] and ECMAScript[5] are the OWP technologies we have embraced to implement the ADGT protocols.

[1] http://www.w3.org/blog/2014/10/application-foundations-for-the-open-web-platform.

[2] http://www.w3.org/TR/webrtc/.

[3] http://www.w3.org/TR/websockets/.

[4] http://www.w3.org/TR/geolocation-API/.

[5] http://www.ecma-international.org/ecma-262/6.0/index.html.

3.1 WebRTC

The Internet is no stranger to audio and video. Nowadays, speaking to someone over a video stream is a simple task for an everyday user, with technologies such as Apple FaceTime, Google Hangouts and Skype video calling. Together with these applications, a wide range of techniques and solutions to problems have been developed and engineered, such as packet loss, recovering from disconnections, and reacting to changes in network, to ensure a high quality of the communication.

The aim of Web Real-Time Communication (WebRTC) is to bring all of this technology into the browser. Differently from those solutions that require the installation of plugins which can be difficult to deploy, test and mantain, and may necessitate licensing fees from developers, WebRTC brings high-quality audio and video to the open Web [3].

Moreover, WebRTC supports data transfer: since a high-quality data connection is needed between two clients for audio and video, it also makes sense to use this connection to transfer arbitrary data. Indeed, WebRTC enables data streaming between browser clients without the need to install plugins or third-party software, implying a strong integration between the content presented by the browser and the real-time content. With WebRTC, web browsers become peers of a real P2P network, being capable to exchange data in an unmediated fashion.

To acquire and communicate streaming data, WebRTC implements the following APIs:

– `MediaStream`, which represents synchronized streams of media such as user's camera and microphone;
– `RTCPeerConnection`, which handles stable and efficient communication of streaming data between peers, with facilities for encryption and bandwidth management;
– `RTCDataChannel`, which enables P2P exchange of arbitrary data, with low latency and high throughput.

The `MediaStream` interface represents a stream of data of audio and/or video. A `MediaStream` may be extended to represent a stream that either comes from or is sent to a remote node, and not just the local camera. This API will not be detailed further here because it is not strictly relevant to the presented work.

The `RTCPeerConnection` interface represents a WebRTC connection between the local computer and a remote peer. It is used to handle efficient data streaming between the two peers.

Differently from most Web applications that choose the Transmission Control Protocol (TCP), WebRTC relies on User Datagram Protocol (UDP) as the default transport protocol. In fact, if on the one hand TCP guarantees delivery of data in the exact order and without duplication, on the other hand in streaming applications most data quickly become obsolete and, if any data were to be ensured in the reception, there would be a bottleneck in case of data loss. Since a completely reliable connection is not a requirement for audio, video and data streaming transmissions, while a very fast connection between the two browsers is highly desirable, UDP has been chosen as the default transport protocol in WebRTC. In particular, WebRTC transports audio and video streams using the Secure Real-Time Transport (SRTP) protocol, which is real-time, and provides encryption, message authentication and integrity to transmitted data. `RTCPeerConnection` hides all the complexities of WebRTC to web developers. WebRTC uses codecs and protocols to make real-time communication possible, even over unreliable networks, adopting techniques for packet loss concealment and noise reduction and suppression, in a completely transparent manner to developers.

Another feature that `RTCPeerConnection` offers to web developers is the Interactive Connectivity Establishment (ICE), a technique developed by the Internet Engineering Task Force [4] to overcome the complexities of real-world networking, where most devices live behind one or more NAT layers, some have anti-virus software that blocks certain ports and protocols, and many are behind proxies and corporate firewalls. First, ICE tries to make a connection using the host address obtained from the operating system and the network card. In case of failure, ICE uses the Session Traversal Utilities for NAT (STUN) [5] protocol to discover the public address of the device and then pass that on. If also this attempt fails and a direct communication between peers over UDP cannot be established, ICE falls back on Traversal Using Relays around NAT (TURN) [6], rerouting the traffic via a TURN relay server using TCP.

The `RTCDataChannel` interface allows us to transfer arbitrary data directly from one peer to another. `RTCDataChannel` works with the `RTCPeerConnection` API, which enables P2P connectivity with lower latency, and uses Stream Control Transmission Protocol (SCTP), allowing configurable delivery semantics: out-of-order delivery and retransmit configuration.

SCTP is a transport-layer protocol, serving in a similar role to the popular protocols TCP and UDP that provides some of the same service features of both: it is message-oriented like UDP and ensures reliable, in-sequence transport of messages with congestion control like TCP.

`RTCDataChannel` can work in either reliable mode (analogous to TCP) or unreliable mode (analogous to UDP). The first guarantees the transmission of messages and also the order in which they are delivered. This takes extra overhead, thus potentially making this mode slower. The latter does not guarantee every message will get to the other side nor what order they get there. This removes the overhead, allowing this mode to work much faster.

Furthermore, in the case of WebRTC, SCTP sits on top of the Datagram Transport Layer Security (DTLS) protocol, which is derivative of SSL, and provides communication security for datagram protocols. In particular, using DTLS, WebRTC guarantees that every peer connection is automatically encrypted and, in particular (Fig. 2):

– messages are not readable if they are stolen while in transit between peers;
– a third party cannot publish messages within the ADGT overlay network;
– messages can not be altered while in transit;
– the encryption algorithm is fast enough to support the highest possible bandwidth between peers.

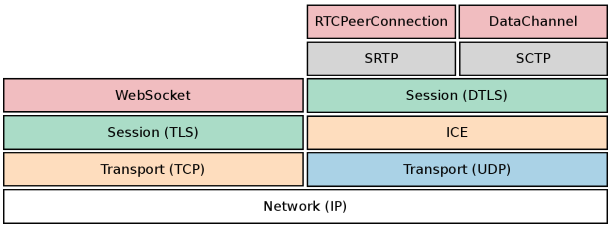

Fig. 2. WebRTC and WebSocket protocol stack.

3.2 WebSocket

The Web has been traditionally tied to the request/response paradigm of HTTP. Nevertheless, with the need to have a more and more dynamic web, new technologies such as AJAX have emerged. However, all of these technologies are not well suited for low latency applications, carrying the overhead of HTTP.

The WebSocket specification defines an API establishing an interactive communication session between a web browser and a server. With this API, the client and the server can make a persistent full-duplex connection between them and send data to each other at any time. The main advantage is that the client can send messages to a server and receive event-driven responses without having to poll the server for a reply.

3.3 Geolocation

The Geolocation API defines a high-level interface to location information associated with the device. The API itself is agnostic of the underlying location information sources: location can be indiscriminately obtained from a Global Positioning System (GPS), inferred from network signals such as IP address, RFID, Wi-Fi and Bluetooth MAC addresses, and GSM/CDMA cell IDs, as well as user input.

The API provides the location information represented by latitude and longitude coordinates. The API is designed to enable both "one-shot" position requests and repeated position updates, as well as the ability to explicitly query the cached positions.

3.4 ECMAScript

ECMAScript is a scripting language specification standardized by ECMA International. JavaScript is one of the most known implementation of the language.

The current version of the ECMAScript Language Specification standard is ECMAScript 2015 (6th Edition) and introduces language support for classes, constructors, and the `extend` keyword for inheritance. Moreover, it provides a way to load and manage module dependencies, new `Map` and `Set` objects, `Promise` objects and many other features.

4 Implementation

The ADGT protocol has been implemented using Open Web Platform technologies only. Our implementation, named Adgt.js, has been designed as an ECMAScript 6 software library that can be freely used for the realization of P2P-based LBSs, where it is important to discovery geographic neighbors and exchange messages with them using a technology that guarantees security and data encryption.

In particular, we have defined and written a JavaScript `Peer` class that represents the ADGT peer. This class is characterized by a `Descriptor`, *i.e.*, an unique identifier of the peer in the network and its geographic location. This latter implements the `Position` interface defined in the Geolocation API and represents the position of the peer at a given time, but also its altitude and its speed.

Furthermore, the `Peer` class contains a reference to a `GeoBucket` object that, as the name says, implements the peculiar routing table of the ADGT protocol. Our GeoBucket implementation consists of a wrapper of the new ECMAScript 6 `Set` class, whose elements are nodes of the network. The `GeoBucket` class, in addition to being a collection of nodes, presents functionalities for the management of geographic neighborhood, therefore to add and remove nodes that approach and move away from the peer, and to update the information about the geographic locations of the neighbors.

In Adgt.js, neighbors are represented by the `RemoteNode` class, which actually realizes the P2P connection with other peers, through WebRTC technologies. More specifically, this class allows to connect to another peer of the ADGT network using the `RTCPeerConnection` interface, and to directly send a message to it with the `DataChannel` interface. In this way, all data exchanges between network nodes—such as position updates as well as peer discovery messages—are realized using WebRTC.

`DataChannels` are also used as signaling channels. In fact, signaling methods and protocols, *i.e.*, the mechanisms required to coordinate communication and to send control messages, are not specified by WebRTC. WebRTC assumes the existence of a communication coordination process, allowing clients to exchange session control messages (outlined by the JavaScript Session Establishment Protocol [7]), error messages, media metadata such as codecs and codec settings, bandwidth and media types, key data, used to establish secure connections, and network data, such as a IP address and port, without placing constraints on the signaling technology.

Although a signaling service consumes relatively little bandwidth and CPU per client, signaling servers for a popular application may have to handle a lot of messages, from different locations, with high levels of concurrency. For this reason, we have decided to distribute the responsibility to act as signaling servers among all the peers of the network, using `DataChannels`. In particular, the peer discovery operation has been implemented in a way that when a peer receives the list of neighbors from the peer that has contacted, the latter acts also as a signaling server between the first and the possible peers which have to be contacted.

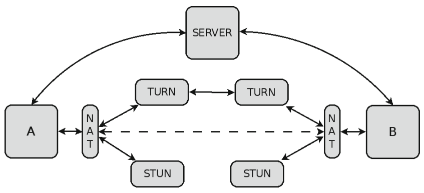

Fig. 3. Using ICE to cope with NATs and firewalls.

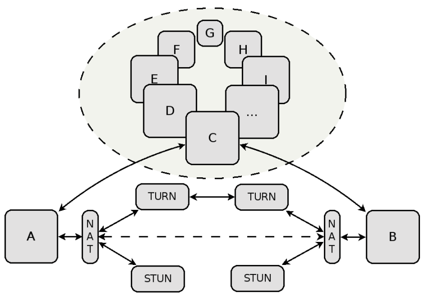

Fig. 4. Peers of the network act as signaling servers in our implementation.

A direct connection between two peers can be achieved by means of a signaling server that coordinates the communication. Actually, a signaling server by itself is not sufficient to overcome the complexities of real-world networking, that can be solved with the use of ICE technology, as shown in Fig. 3.

To increase system scalability, Adgt.js has been designed to be architecturally different from what depicted in Fig. 3, inasmuch the operations of signaling between two peers that are attempting to establish a connection are provided by an intermediary peer rather than from a centralized server. Figure 4 represents the architecture of our implementation.

In particular, the peer designed to act as a signaler between two other peers is the one that allowed the other two to get to know each other, at the end of the discovery process. Figure 5 shows the sequence of messages exchanged during a discovery operation, in the event that peer *A* wants to start a conversation with peer *B*, just discovered by means of peer *C*. After peers *A* and *C* have exchanged discovery messages, where *A* asks for a specific geographic location and *C* returns a list of known peers near the location indicated including *B*, if peer *A* wants to add peer *B* to its GeoBucket, peer *C* will be the signaler among them. The first message that *A* has to send to *B* through *C* is an Offer message, which is a serialized session description message, followed by an ICE message with

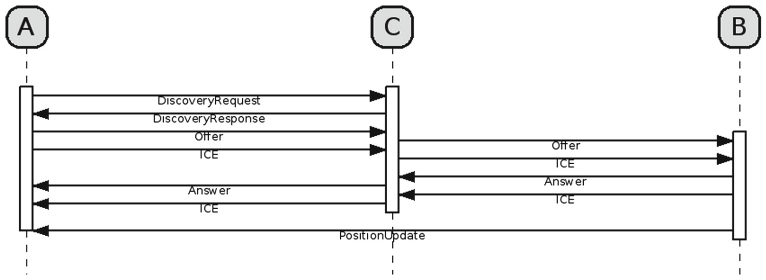

Fig. 5. Messages exchanged during discovery operation.

the information about network interfaces and ports. On the other side, when B receives the Offer message, it replies to A through C with an Answer message containing its session description, therefore with an ICE message. Finally, at the end of this initialization, A and B can directly exchange ADGT or application-specific messages, such as updates on their geographic location. Despite the complexity of the architecture, the implementation hides all these aspects to users, which do not have to worry how the connections are established.

Since it is not always possibile to use another peer as a signaling server, *e.g.*, when the peer joins the P2P network, each peer has a reference to a `BootstrappingNode` that is able to operate as a signaling server for those peers that log on to the network for the first time. We have implemented this kind of signaling server using the Node.js framework[6] and the WebSocket protocol. The choice of adopting Node.js, which is an open-source runtime environment based on Google's V8 JavaScript engine, has allowed to reuse most of the code written for the ADGT implementation. Furthermore, the WebSocket protocol allowed us to encrypt the signaling and negotiation communication like the standard HTTPS protocol works, ensuring that no one can intercept messages sent to the server to figure out which peers are talking to whom.

Adgt.js has been released online[7] with a free and open-source software license and can be used as a web application framework without restrictions.

5 Realizing LBSs with Adgt.js

To design and implement LBSs with Adgt.js is easy, not much different from making a simple web page. Moreover, being Adgt.js an implementation of a P2P protocol, it is affordable by whoever, as it may run over any type of device, not being particularly demanding in terms of computing and memory resources.

In order to use the Adgt.js web application framework, it is sufficient to include the JavaScript file in the HTML page using the `src` attribute in the `<script>` tag. Once the Adgt.js is included, it is already possible to create a new ADGT peer, as shown in Listing 1.1.

Listing 1.1. Creating a new ADGT peer.

```
var peer = new Peer(options);
peer.connect();
```

Two statements are enough: the first one actually creates an instance of the `Peer` class, while the second one starts the connection to the ADGT network. During the creation of the peer, it is possible to specify some options, such as the address of the bootstrapping node, the STUN and TURN servers, as well as some parameters of the ADGT protocol.

Adgt.js has been developed with an event-driven approach, thus allowing to define functions that are executed upon the occurrence of certain events, such as the reception of a message from another peer, or a change in the neighborhood.

6 https://nodejs.org.

7 https://github.com/brambilla/adgt.js.

In Listing 1.2, it is reported how to set listener functions for two kinds of events: `neighbors` and `data`. The first event occurs when the neighborhood of the peer changes, while the second one fires when the peer receives any type of data from near peers. The callback functions allow us to manage the set of neighbor `descriptors` and received `data`, respectively.

Listing 1.2. Setting listeners for peer events.

```
peer.on('neighbors', function(descriptors) {  });
peer.on('data', function(data) {  });
```

To send `data` to the geographic neighbors of the peer, it is sufficient to invoke the `send` method, as in Line 1 of Listing 1.3.

Listing 1.3. Sending data to neighbors and updating peer's position.

```
1 peer.send(data);
2 ...
3 peer.move(position);
```

In case the geographic location of the peer changes, it is sufficient to use the `move` method to automatically trigger the position update process, that involves the transmission of a message to the neighbors and the removal of peers no longer included in the GeoBucket (Line 3 of Listing 1.3).

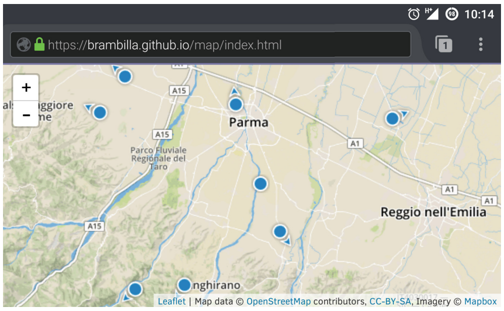
10:14
https://brambilla.github.io/map/index.html
Parma
Reggio nell'Emilia
Parco Fluviale Regionale del Taro
A15
A1
nghirano
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

Fig. 6. The LBS developed with ADGT.js, running on Firefox for Android.

We have implemented and published online[8] a simple LBS that illustrates the ease of use of Adgt.js and represents a building block for more sophisticated applications. The LBS shows on a map the peers connected to the network, *i.e.*, visitors of the web page, that are at a maximum distance of 40 km. Neighbor discovery and connection establishment is entrusted to Adgt.js. Regarding the map, we have adopted Leaflet.js[9], a widely used open-source JavaScript library used to build web mapping applications. Figure 6 is a screenshot taken from an Android smartphone running the LBS on Firefox for mobile.

[8] https://brambilla.github.io/map/index.html.
[9] http://leafletjs.com.

This LBS only requires to update the geographic location of the peer and change the marker on the map with the position of the browser obtained with the Geolocation API, as shown in Listing 1.4.

Listing 1.4. Managing current position of the device.

```
navigator.geolocation.watchPosition(function(position) {
  peer.move(position);
  var latLng = L.latLng(position.coords.latitude, position.coords.longitude);
  marker.setLatLng(latLng);
  if (position.coords.speed > 0) { marker.setIcon(icon_heading); }
  marker.setRotationAngle(position.coords.heading);
  map.panTo(latLng);
});
```

In addition, when a change in the neighborhood happens, all the markers on the map representing neighbors are updated as in Listing 1.5.

Listing 1.5. Managing changes of neighborhood.

```
peer.on("neighbors", function(descriptors) {
  markers.clearLayers();
  for (var index in descriptors) {
    var position = descriptors[index].position;
    var latLng = L.latLng(position.coords.latitude, position.coords.longitude);
    var rotationAngle = position.coords.heading;
    if (position.coords.speed > 0) {
      markers.addLayer(L.marker(latLng, {rotationAngle: rotationAngle, icon: icon_heading}));
    } else {
      markers.addLayer(L.marker(latLng, {rotationAngle: rotationAngle, icon: icon}));
    }
  }
});
```

6 Related Work

As WebRTC has reached a good level of maturity and has been adopted by major web browsers, researchers have started to investigate the potentiality of this technology. Tindall and Harwood presented an implementation of an unstructured P2P protocol using WebRTC [8]. Bevilacqua *et al.* described a network architecture for the development of browser-based P2P web applications [9]. Such a work has the disadvantage of requiring a nonstandard browser plugin for signaling. A framework for decentralized online social networks is presented by Disterhoft and Graffi [10]. The main differences between our implementation and the above-mentioned ones, apart from the fact that the P2P schemes are different, reside in the signaling process. Indeed, in Adgt.js, signaling is a distributed mechanism, not supplied by a single central server.

7 Conclusion

In this paper we have presented Adgt.js, a web application framework that enables the realization of completely decentralized LBSs, being a cross-platform implementation of the ADGT georeferenced P2P overlay scheme.

We have described the implementation and the architecture of the developed framework, paying particular attention to the adopted technologies. It has also been accurately described how to use the framework, with reference to a concrete example. Moreover, the framework has been published online, together with the example, which is freely usable.

Regarding future work, we will investigate the performance of the implemented framework, with respect to technological aspects such as battery drain of mobile devices, also compared with previous results obtained in simulation. Furthermore, we will evaluate the adoption of even more innovative technologies, such as Web Workers[10] and Object Real-Time Communications for WebRTC.[11]

References

1. Florian, M., Pieper, F., Baumgart, I.: Establishing location-privacy in decentralized long-distance geocast services. Ad Hoc Netw. **37**, 110–121 (2016)
2. Brambilla, G., Picone, M., Amoretti, M., Zanichelli, F.: An adaptive peer-to-peer overlay scheme for location-based services. In: IEEE 13th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, August 2014
3. Grigorik, I.: High Performance Browser Networking. O'Reilly Media, Sebastopol (2013)
4. Rosenberg, J.: Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, Internet Engineering Task Force (IETF). Request for Comments **5245**, April 2010
5. Rosenberg, J., Mahy, R., Matthews, P., Wing, D.: Session Traversal Utilities for NAT (STUN), Internet Engineering Task Force (IETF). Request for Comments **5389**, October 2008
6. Mahy, R., Matthews, P., Rosenberg, J.: Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN), Internet Engineering Task Force (IETF). Request for Comments **5766**, April 2010
7. Uberti, J., Jennings, C.: Javascript Session Establishment Protocol, Internet Engineering Task Force (IETF), Network Working Group, February 2012
8. Tindall, N., Harwood, A.: Peer-to-peer between browsers: cyclon protocol over WebRTC. In: IEEE International Conference on Peer-to-Peer Computing (P2P) (2015)
9. Bevilacqua, A., Boemio, P., Romano, S.P.: Introducing ufo.js: a browser-oriented P2P network. In: International Conference on Computing, Networking and Communications (ICNC) (2014)
10. Disterhoft, A., Graffi, K.: Protected chords in the web: secure P2P framework for decentralized online social networks. In: IEEE International Conference on Peer-to-Peer Computing (P2P) (2015)

[10] https://html.spec.whatwg.org/multipage/workers.html.
[11] https://www.w3.org/community/ortc/.