WebGC Gossiping on Browsers without a Server [Live Demo/Poster]

Raziel Carvajal-Gómez, Davide Frey, Matthieu Simonin, Anne-Marie Kermarrec

WebGC Gossiping on Browsers without a Server
[Live Demo/Poster]

Raziel Carvajal-Gómez, Davide Frey,
Matthieu Simonin and Anne-Marie Kermarrec
name.surname@inria.fr

INRIA Rennes, France

Abstract. [1]Decentralized social networks have attracted the attention of a large number of researchers with their promises of scalability, privacy, and ease of adoption. Yet, current implementations require users to install specific software to handle the protocols they rely on. The WebRTC framework holds the promise of removing this requirement by making it possible to run peer-to-peer applications directly within web browsers without the need of any external software or plugins. In this demo, we present WebGC, a WebRTC-based library that supports gossip-based communication between web browsers and enables them to operate with Node-JS applications. Due to their inherent scalability, gossip-based protocols constitute a key component of a large number of decentralized applications including social networks. We therefore hope that WebGC can represent a useful tool for developers and researchers.

1 Introduction

A number of authors have proposed the use of gossip-based protocols for the implementation of decentralized social networks [13, 9, 14], these protocols form an unstructured distributed system to disseminate information in a periodic way, they are easy to deploy and resilient to failures. Yet, like for other peer-to-peer (P2P) solutions, their implementations have always required users to install specific software to support decentralized protocols to enable users to access the social network from their web browsers. This constitutes a major show stopper for the adoption of decentralized social-network solutions.

In this demo, we present WebGC, a library for gossip-based communication between web browsers. Based on the WebRTC framework, WebGC has the potential to improve the applicability of decentralized user-centric applications like social networks by enabling them to run directly within web browsers. WebGC is a Javascript library to provide a simplified framework for building gossip-based applications. This includes the implementations of standard components such

[1] A previous version of this demo appeared in [8]. Since then, we have integrated the library with a new decentralized signaling service and introduced support for web workers.

as random-peer-sampling [11] and clustering [19] protocols. Moreover, it augments the WebRTC connection-initiation protocol by means of a decentralized signaling mechanism.

WebRTC's primary goal consists in enabling direct communication between two browsers for media and real-time communication. For this reason, WebRTC relies on a signaling server that makes it possible to establish a connection between two browsers that do not know each other. While this feature is very convenient for applications that need to establish a small number of connections, the signaling server quickly becomes a bottleneck in the context of P2P applications. WebGC's decentralized signaling service establishes connections by exploiting the very operation of peer-sampling protocols.

The demo will present a running example of a WebGC-enabled application: a semantic overlay grouping users by interests. Attendees will be able to join the application using their own laptops and follow its evolution in real time on a web page.

2 WebGC Architecture

WebGC is a Javascript library and relies on two underlying frameworks: WebRTC, and SimplePeer. The former is directly provided by compatible web browsers and implements the low-level interaction primitives that make it possible to establish communication between browsers. The latter also takes the form of a Javascript library and provides a wrapper around WebRTC that simplifies the establishment of data connections between peers. The left diagram in Figure 1 places WebGC in the context of these two libraries.

The figure also shows that WebGC comprises a decentralized signaling service that replaces the centralized signaling server used in WebRTC applications. This makes it possible to eliminate bottlenecks when operating in environments that do not involve NAT and firewalls. When these are present, WebGC relies on ICE, STUN, and TURN like standard WebRTC applications. However, we are currently considering augmenting the library with NAT traversal solutions such as Nylon [12] or Croupier [10]. In the following, we describe the gossip-based framework provided by our library and detail its decentralized signaling service.

WebGC Internals. The right diagram in Figure 1 depicts WebGC's architecture. Its core consists of a Coordinator object that instantiates the gossip protocols and acts as a communication broker dispatching incoming messages to the various protocols. The library currently includes the implementation of two peer sampling protocols, Cyclon [18] and the generic protocol suite from [11], as well as a clustering protocol [19, 7]. All protocols implement a GossipProtocol "interface"—since Javascript does not natively support interfaces, we adopt the interface pattern described in [15]. The Coordinator makes it possible to stack these protocols on top of each other [7] to implement applications.

The GossipProtocol interface follows the scheme proposed in the literature [17, 16, 11] and defines the high-level operations that constitute a gossip-protocol. Developers can use the protocols provided by the library, but they

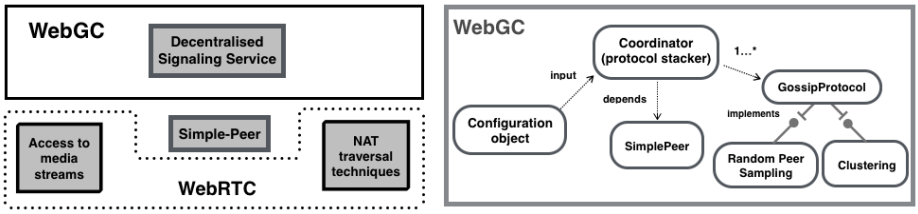

Fig. 1. General architecture

can also implement the operations in GossipProtocol to define new protocols. Finally, the latest version of WebGC also includes support for web workers. This makes it possible to run complex CPU-intensive tasks without blocking the browser's user interface.

Decentralized Signaling. A Signaling Service acts a mediator between two peers, say P_1 and P_2, by assisting them during the establishment of a mutual connection. Let us assume P_1 wants to initiate a connection to P_2. WebRTC does not allow nodes to contact each other using their IP addresses. So P_1 sends its connection request to P_2 via the signaling service. P_2 responds by sending a reply also through the signaling service, and finally P_1 initiates a direct link with P_2 thanks to the data (IP address, communication protocol, etc.) contained in the request and reply messages. WebRTC does not require a specific signaling server but most existing solutions rely on a centralized architecture. However, a centralized signaling server would easily become a bottleneck in a peer-to-peer setting, particularly when using gossip-based protocols, which frequently need to establish new connections.

While other decentralized signaling libraries exist [1], WebGC integrates signaling within the operation of peer-sampling and clustering protocols. In both these types of protocols, which we refer to as overlay protocols, nodes maintain data structures called *views* that contain references to other nodes, and periodically exchange messages that contain subsets of their views.

Our decentralized signaling solution maintains an additional routing table that contains an entry for each of the node references that appear in any of the views of running overlay protocols. Each such entry contains the node reference of a mediator node, i.e. a node that has an active connection with the node in the entry. When a node needs to establish a connection to another node in its table, it simply contacts the node's mediator and uses it as a signaling server.

WebGC maintains its routing tables by augmenting the messages sent by the overlay protocols it hosts. Specifically, for each message sent by one such protocol, WebGC also sends a routing table update that tells the receiving node how to establish a connection which each of the nodes referenced in the overlay message. Consider a node P_j sends to node P_i a message containing a reference to node P_k. If P_j has an open connection with P_k, then P_j itself can act as a mediator between P_i and P_k, so it sends a routing table entry $< P_k, P_j >$. Otherwise, P_j simply forwards the information about its current mediator for

P_k and thus sends an entry $< P_k, R_j[k] >$, where $R_j[k]$ is the mediator for P_k stored in P_js routing table.

3 Demo Timeline

Our demo will present a simple WebGC-based application consisting of two stacked overlay protocols and a chat service. We will provide one or two demo machines, but users will also be able to join the demo using their laptops. The application's web interface will require each user to specify his/her own interests in the form of keywords. This will organize users into a semantic overlay thereby identifying groups of users with similar interests.

Each of them will be able to follow the application's evolution on the web interface in the form of graphs that will display their RPS [11] and a clustering [19] views. In addition, users will be able to broadcast messages to each of the users in either view. Thanks to this feature, the demo might also provide workshop attendees with a way to exchange messages during the workshop.

4 Related work

The introduction of WebRTC [2] has motivated a number of developers to design in-browser peer-to-peer solutions. PeerJS [3], SimplePeer [4], and P [1] all seek to offer a simplified API to program peer-to-peer applications on top of WebRTC. The previous version of WebGC [8] was built on top of PeerJS and exploited PeerServer, its associated signaling server. Since then, we have performed a complete refactoring to build our decentralized signaling solution, and to support multi-threading by means of web workers. In the process, we also migrated from PeerJS to SimplePeer, which simplified the library's requirements and made it possible to run WebGC applications without a browser on NodeJS [5].

While we designed our decentralized signaling solution to work in conjunction with gossip-based overlay maintenance, others have considered the idea of having a decentralized signaling server. P [1], for example, uses servers only for bootstrapping and then each peer can take up the role of a mediator. Unlike WebGC, however, P does not integrate signaling with gossip-based overlay maintenance. Finally, our work on WebGC is closely related to past efforts dedicated to the modular implementation of gossip protocols [16, 11, 17, 6].

5 Conclusions

We presented WebGC, a library that simplifies the development of gossip-based applications based on the WebRTC framework. Built on top of SimplePeer [4], WebGC includes a decentralized signaling service as well as the implementation of several standard gossip protocols. Our demo demonstrates the effectiveness of our library in a real context, with a gossip-based chat application.

References

1. https://github.com/unsetbit/p/
2. http://www.webrtc.org/
3. http://peerjs.com/
4. https://github.com/feross/simple-peer/
5. https://nodejs.org/
6. http://gossiplib.gforge.inria.fr/
7. Bertier, M., Frey, D., Guerraoui, R., Kermarrec, A., Leroy, V.: The gossple anonymous social network. In: Middleware 2010, Bangalore, India, 29/11 - 3/12, 2010. pp. 191–211 (2010)
8. Carvajal-Gomez, R., Frey, D., Simonin, M., Kermarrec, A.: Webgc: browser-based gossiping. In: Proceedings of the Middleware '14 Posters & Demos Session, Bordeaux, France, December 8-12, 2014. pp. 13–14 (2014), http://doi.acm.org/10.1145/2678508.2678515
9. Datta, A., Sharma, R.: Godisco: Selective gossip based dissemination of information in social community based overlays. In: Aguilera, M., Yu, H., Vaidya, N., Srinivasan, V., Choudhury, R. (eds.) Distributed Computing and Networking, Lecture Notes in Computer Science, vol. 6522, pp. 227–238. Springer Berlin Heidelberg (2011)
10. Dowling, J., Payberah, A.H.: Shuffling with a croupier: Nat-aware peer-sampling. In: 2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China, June 18-21, 2012. pp. 102–111 (2012), http://dx.doi.org/10.1109/ICDCS.2012.19
11. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM TOCS 25(3) (2007)
12. Kermarrec, A., Pace, A., Quéma, V., Schiavoni, V.: Nat-resilient gossip peer sampling. In: 29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009), 22-26 June 2009, Montreal, Québec, Canada. pp. 360–367 (2009)
13. Mega, G., Montresor, A., Picco, G.: Efficient dissemination in decentralized social networks. In: Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on. pp. 338–347 (Aug 2011)
14. Nilizadeh, S., Jahid, S., Mittal, P., Borisov, N., Kapadia, A.: Cachet: A decentralized architecture for privacy preserving social networking with caching. In: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies. pp. 337–348. CoNEXT '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2413176.2413215
15. Osmani, A.: Learning JavaScript Design Patterns. JavaScript and jQuery developer's guide, O'Reilly Media, Inc. (2012), http://books.google.fr/books?id=JYPEgK-1bZoC
16. Rivière, E., Baldoni, R., Li, H., Pereira, J.: Compositional gossip: a conceptual architecture for designing gossip-based applications. SIGOPS Operating Systems Review (2007)
17. Taïani, F., Lin, S., Blair, G.S.: Gossipkit: A unified componentframework for gossip. IEEE Trans. Software Eng. 40(2), 123–136 (2014), http://doi.ieeecomputersociety.org/10.1109/TSE.2013.50
18. Voulgaris, S., Gavidia, D., van Steen, M.: CYCLON: inexpensive membership management for unstructured P2P overlays. J. Network Syst. Manage. 13(2), 197–217 (2005), http://dx.doi.org/10.1007/s10922-005-4441-x
19. Voulgaris, S., van Steen, M.: Epidemic-style management of semantic overlays for content-based searching. pp. 1143–1152. EuroPar 2005, Lisbon, Portugal

As Scalable As Possible: Foundations of large-scale dynamic distributed systems

Team Leader: François Taiani
Web site: http://www.inria.fr/en/teams/asap
[1]email: name.lastname@inria.fr

WebGC Gossiping on Browsers without a Server

Raziel Carvajal-Gómez[1] - Davide Frey[1] - Matthieu Simonin[1] - Anne-Marie Kermarrec[1]

Introduction

In one hand, the inclusion of Real Time Communication (RTC) API's such as WebRTC in web browsers has renewed the interest in peer-to-peer (P2P) applications. In the other hand, epidemic (gossip) protocols have always played a prominent role in P2P applications due to their inherent scalability, resilience to failures and wide applicability. Nowadays, having gossiping in real time communications implies the next considerations:

Pros (left) & Cons (right) of Gossip Protocols

- **Easy to deploy:** P2P applications, Mobile Computing, Data Centers
- **Versatile:** Overlay Networks, Broadcast services, Video Streaming Network statistics, Recommendation
- **Self-adaptive:** Resilient to churn, catastrophic failures

- **Software portability:** existing gossip-based applications require users to install specific software to manage gossip exchanges. In a world where more and more cutting-edge applications run within web browsers, this represents a clear disadvantage for epidemic protocols.

WebGC: browser-based gossip library

We propose WebGC, a library for gossip-based communication between web browsers.

- It is build on the top of WebRTC, it contains 3 gossip protocol implementations.
- Protocols' specification is done via a JSON object.
- Few JavaScript lines are required to extend web applications with gossip techniques.
- A decentralised signaling service, to create new peer connections, is offered.
- WebGC runs on NodeJS too and relies relies on the simple-peer project [3].

WebGC Architecture

The library consists of a COORDINATOR and a set of GOSSIP PROTOCOLS.

- The COORDINATOR instantiates gossip protocols, manages peer connections and links dependencies between protocols. The library currently contains an implementation of the peer sampling protocol [5], as well as a clustering protocol [4] implementation.
- All protocols implement a common GOSSIPPROTOCOL "interface"—since JavaScript does not natively support interfaces, we adopt the interface pattern described in [6].
- One CONFIGURATIONOBJECT is red to define a gossip application, this makes possible to stack a combination of gossip protocols to achieve new complex applications.
- Connections between peers are used by the decentralised signalling service to create new connections.

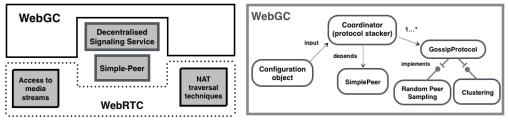

Figure 1: WebGC architecture

WebGC In Action

The next two figures show how easy it is to set up the library. Figure 2 contains an example of the configuration object: a JSON object that defines the settings and the dependencies between gossip protocols. Figure 3 shows the interface that enables users to extend WebGC with other implementations.

```
var configurationObj = {
  peerId: 'myId',
  peerJsOpts: {
    host: '131.254.213.43', //PeerServer address
    port: 9000,             //PeerServer port
    /* ... */
  },
  gossipAlgos: {
    rps_1: {
      class: 'RPS', viewSize: 10, fanout: 5,
      periodTimeOut: 10000,
      propagationPolicy: { push: true, pull: true }
    },
    clustering_1: {
      class: 'Clustering', viewSize: 5, fanout: 3,
      periodTimeOut: 20000,
      propagationPolicy: { push: true, pull: true },
      selectionPolicy: 'biased', //random OR biased OR agr-biased
      similarityFunction: 'DumbProximityFunc',
      attributes: { rpsView: 'rps_1.view' }
    }
  },
  plotter: true, //NodeJS server for graphs
  loggingServer: {
    host: '131.254.213.42', port: 9001 //Debugging server
  }
}
```

Figure 2: Configuration object

```
(function(exports){
  function GossipProtocol(opts){
    this.class = opts.class;
    this.view = {};
    this.loop = 0;
    /* ... */
  }

  GossipProtocol.prototype.increaseAge = function(){ };
  GossipProtocol.prototype.selectPeer = function(){ };
  GossipProtocol.prototype.getItemsToSend = function
    (thisId, dstPeer, thread){ };
  GossipProtocol.prototype.selectItemsToKeep = function
    (thisId, rcvCache){ };
  GossipProtocol.prototype.initialize = function(keys){ };

  exports.GossipProtocol = GossipProtocol;
})(this)
```

Figure 3: Methods to implement of the GossipProtocol interface

Demonstration

Follow the next steps for trying WebGC:

- Open a web browser, just Chrome 37+ or Firefox 32+ is supported
- Connect to URL given by the expositor
- Type an identifier (typically your name)
- Select your profile (set of topics in computer science)
- Click on the start button
- Chat with your neighbours

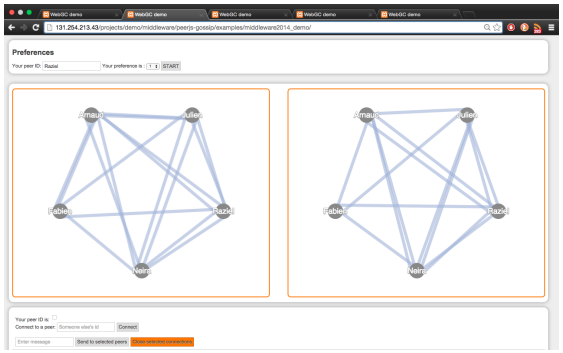

Figure 4: Screenshot of the WebGC demonstration

Figure 4 shows you what you will see in the demonstration and it is divided in three sections:

- **User Profile (top):** users select a list of topics in computer science. Through the use of two gossip protocols (RPS and Clustering), WebGC is going to find other peers with similar profiles
- **Graph visualisation (middle):** the overlay of two gossip protocols are showed in this section.
- **Application (bottom):** chat with peers you have a connection with

Do not reinvent the wheel

WebGC relies on the recent introduction of WebRTC, and two projects:

- **Simple-Peer** is a Peer implementation to create connections between browsers and it is compatible with NodeJS too
- **PeerServer** is a centralised signalling service. WebGC uses a customised version of PeerServer [2] to bootstrap incoming peers.

Experimental Results

We evaluated the properties of the graphs generated by the RPS and Clustering protocols along the lines of [5].

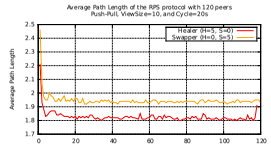

Figure 5: Average Path Length of RPS

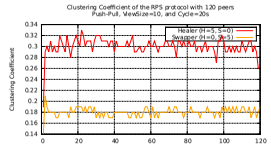

Figure 7: Clustering Coefficient of RPS

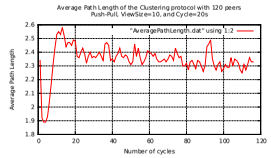

Figure 6: Average Path Length of Clustering

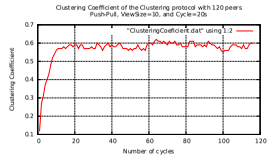

Figure 8: Clustering Coefficient of Clustering

Conclusions and ongoing work

We plan to test WebGC with a certain degree of churn.

References

[1] http://www.webrtc.org/.

[2] https://github.com/peers/peerjs-server.

[3] https://github.com/feross/simple-peer.

[4] M. Bertier, D. Frey, R. Guerraoui, A. Kermarrec, and V. Leroy. The Gossple anonymous social network. In *Middleware 2010, Bangalore, India*

[5] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. *Middleware 2004* Toronto, Canada. October 2014.

[6] A. Osmani. *Learning JavaScript Design Patterns.* JavaScript and jQuery developer's guide. O'Reilly Media, Inc., 2012.