Addressing challenges in browser based P2P content sharing framework using WebRTC

Shikhar Vashishth[1] Yash Sinha[2], K Hari Babu[3]
Department of Computer Science and Information Systems
Birla Institute of Technology and Science
Pilani, India.
{f2012436[1], f2012365[2], khari[3]}@pilani.bits-pilani.ac.in

***Abstract*— Most of the content sharing applications use the client/server model in which all of group managements are done by the server and this sometimes becomes a communication bottleneck. Installing specialized software for different purposes such as file sharing, video conferencing etc., becomes a barrier for the user. Recent technologies like NodeJs and Socket.io have fostered new ideas the ways web browsers can be used. Moreover, the emerging standards of WebRTC open up new paradigm of direct communication channel between web browsers without relaying the data through a web server. But there are certain issues such as lack of full-fledged threading/concurrency support in the JavaScript language, reliance on synchronous loading etc. that restricts modern day browsers to take full advantage of current multiprocessing capabilities. Although, on one hand there are advantages of using web browsers, such as no requirement of specialized software, benefits of emerging technologies etc.; the aforementioned issues pose challenges in implementation in certain areas.**

In this paper, we have tried to couple the benefits of peer-to-peer (P2P) architecture (elimination of centralized dependency, better scalability, shareability etc.) along with the advantages of recent web technologies (NodeJs, WebRTC etc.) by designing and implementing a browser based P2P content sharing framework. We have addressed the aforementioned challenges of a browser based P2P architecture by providing a mechanism to exchange messages asynchronously and facilitating new peer joins via existing peers in the network, thus reducing the dependency on bootstrap server. Our prototypical implementation demonstrates the feasibility, efficiency and scalability of this lightweight framework, on the top of which a variety of applications can be added as a layer of functionality.

Keywords— peer-to-peer, signalling, WebRTC, DHT, NodeJs

I. INTRODUCTION

With rapidly growing need and dependence of society on data sharing and communication, the networks today are growing exponentially with time. The client-server architecture doesn't seem to be a feasible solution as much as peer-to-peer (P2P) architecture because often too many requests to the server leads to congestion and if it fails, whole network goes down. Further cost and maintenance issues are also there.

On the other hand, P2P architecture is more reliable as central dependency is eliminated and resource sharing is easy. Moreover, P2P is scalable because with increase in number of participants the storage capacity, computation power and bandwidth of the network also increases as the resources are shared among others.

Upcoming web technologies like NodeJs and Socket.io have fostered new innovations is usability of web browsers. The new paradigm of WebRTC has enabled a direct communication channel between web browsers without relaying data through a server. WebRTC is supported by most of the popular browsers. And this technology is getting more secured and fast everyday. Also, WebRTC cannot be monitored by third party which makes it more secure. Therefore, instead of bugging the user to download a new specialised software for each of his different needs, we have attempted to assimilate the benefits of P2P architecture implemented using recent web technologies on top of web browsers; thus helping him use an already installed software for more diverse purposes. Much work has been done that emphasises that a variety of activities and purposes which had previously required specialized software such as file sharing between peers without a server to relay the files, video and audio chat without the use of proprietary 3rd-party plugins, and multimedia conferencing without the need for proprietary, platform-dependent 3rd-party applications; can be now built with ease using WebRTC.

But there are certain issues such as lack of threading/concurrency support in the JavaScript language, reliance on synchronous loading, single main thread (event loop) & unavailability of functionality to create new threads etc., that restricts all capabilities of P2P networks be fully implemented on modern day browsers.

II. BACKGROUND AND RELATED WORK

Chord protocol is one of the most prominent, simple and effective DHT technique; however it is designed for languages which supports synchronous calls therefore cannot be used in asynchronous environment [Chord paper]. [1] proposes a model for building DHT based CDN on top of webrtc which gives an abstract picture of how peers can join the network and can exchange messages between each other. WebRTC-Chord [David Dias] is an asynchronous implementation of DHT which is designed for web browsers. This suffers from over dependence on centralized server which becomes a vulnerable point of failure and a communication bottleneck for the whole network. [online social network] presents a design and implementation of social networking application which can build on top of WebRTC-based p2p framework. [sharefest] is another application to allow browsers to share files using

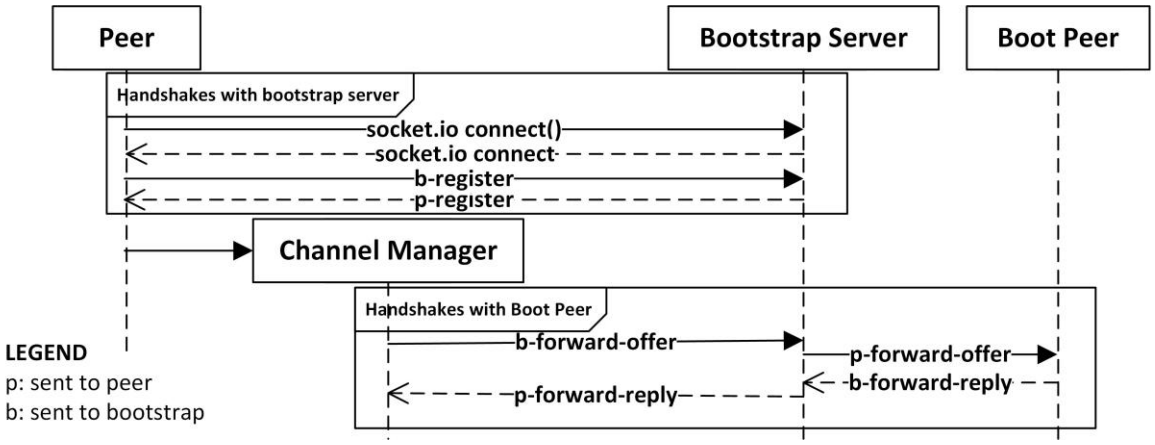

Figure 1. Join Boot Peer via Bootstrap server

proposed framework. CHEWBACCA (CHord, Enhanced With Basic Algorithm Corrections and Concurrent Activation) [CHEWBACCA] is a P2P network framework in Java using sockets based messaging which uses synchronous calls thus, is not suitable for event driven, non-blocking systems like web-browsers.

III. Challenges in Implementation

Here we discuss the challenges faced in course of implementation of the framework:

A. Lack of full-fledged threading/concurrency support in the Javascript language

JavaScript historically suffers from an important limitation: all its execution process remains inside a unique thread. This JavaScript limitation implies that a long-running process freezes the main window. The user is unable to interact with the application and user experience becomes unpleasant. The user may decide to kill the tab or the browser instance. The join operation to the network requires peers to be discovered and peer to peer connections to be established which involves a number of procedures like contacting the bootstrap server, handshakes between connecting peers and several message forwarding operations depending on the network size. Moreover, sequential execution of the procedure is required because next procedure requires successful completion of the preceding procedure.

If this operation is implemented in traditional synchronous way, it is required that the thread sleeps (or waits) till the previous procedure executes successfully. There is no provision for sleep in JavaScript, and busy waiting, in the worst case, will freeze the main window.

Further, because the web workers operate independently of the main thread, they cannot access many of its objects. They cannot access the DOM, so they cannot read or modify the HTML document. In addition, they cannot access any global variables or some special objects like the window, parent and the document. Because the communication with web worker is based on messaging, sequential execution cannot be guaranteed.

B. Dependency on bootstrap server

In order for a WebRTC application to set up a P2P connection, its clients need to exchange information such as session control messages, error messages, media metadata, key data (for secure connections), and network data. [cite http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/] This signalling process needs a way for clients to pass messages back and forth. To avoid redundancy and to maximize compatibility with established technologies, signalling methods and protocols are not specified by WebRTC standards (as outlined by JSEP). [cite http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03#section-1.1]. The main challenge here is signalling servers may have to handle a lot of messages, from different locations, with high levels of concurrency. This signalling server is generally also called the bootstrap server.

In current implementations [cite David Dias], bootstrap server is also actively involved in connection of new peers to the existing network, facilitation of handshakes and network stabilization. This defeats the purpose of using a P2P architecture as the bootstrap server becomes a communication bottleneck as well as a single point of failure.

C. Using ICE to cope with NATs and firewalls: STUN and TURN servers

In real life scenarios most devices function behind one or more layers of NAT. Some may have anti-virus software that blocks certain ports and protocols, and others may be behind proxies and corporate firewalls. A firewall and NAT can be implemented by the same device, such as a home wifi router. So ICE framework is required to overcome the complexities of real-world networking.

ICE first tries to make a connection using the host address obtained from a device's operating system and network card; if that fails (which it will for devices behind NATs) ICE obtains an external address using a STUN server, and if that fails, traffic is routed via a TURN relay server. A STUN server is used to

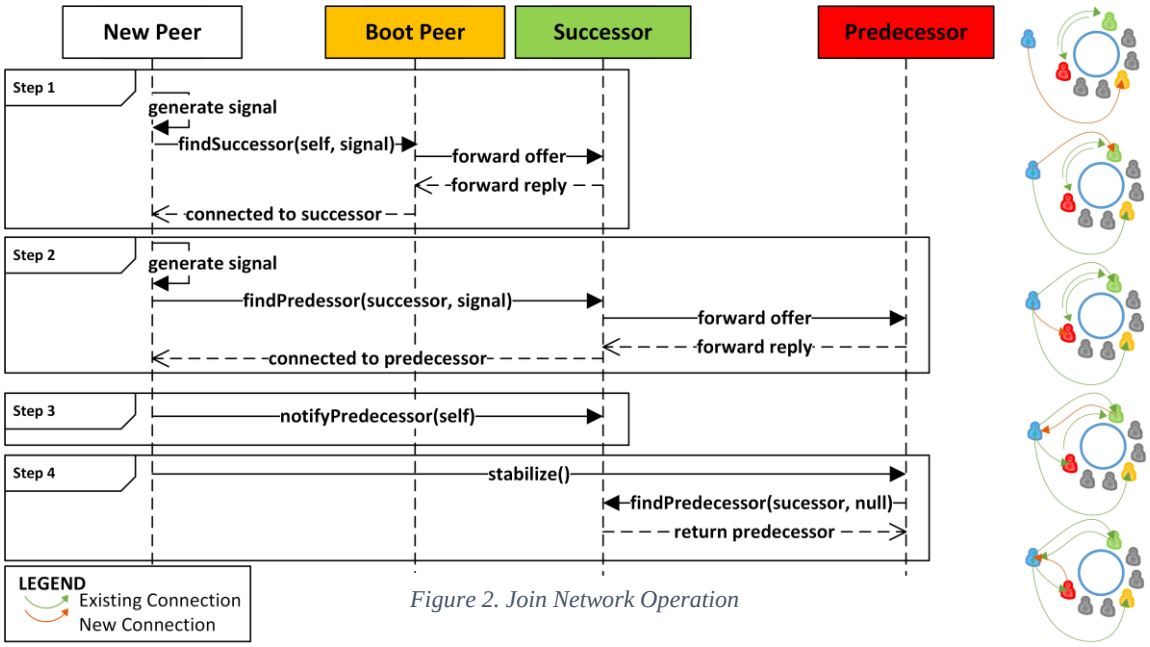

Figure 2. Join Network Operation

get an external network address whereas TURN servers are used to relay traffic if direct (peer to peer) connection fails. ICE servers may have to handle a lot of messages, so high levels of concurrency is required.

IV. Design and Implementation of Prototype

A. Components

1) Peer

a) Channel Manager

b) Node Details

2) Bootstrap

3) ICE Server

4) Message Format

B. Functionality (Sequence Diagram)

1) Join Boot Peer via Bootstrap server

2) Join Network

Two approaches to make find successor request
Including signal with the request
Generating

V. Proposed Solutions to the Challenges

A. Division of join network operation into asynchronous procedures

We divided the join network operation into separate procedures that can be called asynchronously. Sequential execution is guaranteed with the help of the response table, which stores the results of preceding procedures based on message ids and *function* parameter of the procedures that indicates which procedure is to be called next once this procedure executes successfully.

B. Reducing dependency on bootstrap server

C. Addressing load on STUN and TURN servers

We use multiple STUN servers to balance load of ICE candidate requests on them. We use TURN server to route traffic via relay server. Further, periodically we run a check for existing connections which are not required, and destroy them.

VI. Emulation Environment

A. Stunman, Google ICE server

B. listPeers(), getMsgCount(), fixAllFingers()

VII. Evaluation

A. Message count to join network O(n) vs O(log n)

B. Message count to fix fingers

C. Time to join network

D. Max peers vs number of stun servers

E. Chrome vs Firefox vs Others (Opera, Android, Spartan etc.)

F. Visualisation of network (at Bootstrap server, GUI)

G. Comparison of two approaches to make find successor request: tradeoff between signal generation time and message floating time Discuss

H. Churn rate Discuss

I. Local STUN vs Google join time. Discuss

VIII. Future Scope

Because JavaScript engines have only a single thread, asynchronous events are forced to be queued for execution. Queueing in JavaScript varies from browser to browser. Since the network is dynamic, messages may be delivered out of order to a peer and this can lead to errors.

When nodes leave, bootstrap server is required to stabilise the network. This may also be achieved using peers in the network only and the stabilisation via bootstrap server be used as fall back, if it fails. This will reduce load on bootstrap server.

Security issues have not been investigated.

IX. Conclusion

X. References

There are no sources in the current document.