Decentralized Bootstrapping for WebRTC-based P2P Networks

Dennis Boldt, Felix Kaminski and Stefan Fischer
Institute of Telematics
University of Lübeck
Lübeck, Germany
Email: {boldt,kaminski,fischer}@itm.uni-luebeck.de

***Abstract*—Around the millennium, peer-to-peer (P2P) networks were standalone applications, where users had to configure their firewalls properly to be able to connect to the network. Nowadays, P2P connections between browsers are possible without the need for any user-side configuration. This can be achieved with the Web Real-Time Communication protocol stack (WebRTC). Recent research showed that WebRTC-based peer-to-peer networks can work as a decentralized, redundant and encrypted storage for user data. However, all existing networks employ a centralized bootstrapping infrastructure, which still is a single point of failure. In this paper, we present a decentralized architecture to handle bootstrapping for a WebRTC-based peer-to-peer network, completely removing the need for a central instance. The architecture uses the highly decentralized Domain Name System (DNS), combined with so-called Master-Peers. Our evaluation shows that the architecture scales well and reduces the bootstrapping time remarkably.**

Keywords–Peer-to-Peer; Decentralized; Bootstrapping; WebRTC; Chord; DNS

I. INTRODUCTION

The Internet changed in the last 20 years. In the 1990s, the Internet was a consume-only network, where content providers produced the content, and people consumed it. Starting in the 2000s, the Internet changed to a user-generated content network. Thus, new web technologies and protocols were designed to support that shift. In 2006, the first technology was the XMLHttpRequest (XHR) [1]. XHR provides asynchronous HTTP requests without page reloading. To get real-time updates from the server, a client needs to perform XHR requests frequently. In 2009, a first draft of the WebSockets Protocol was published [2]. The main benefit of the WebSockets Protocol is that connections are bidirectional. It can be used by a server to push data to a client immediately.

To enable browser-to-browser communication, Web Real-Time Communication (WebRTC) was developed in recent years [3], [4] (see Section II-C).

The main challenge of every P2P network is *bootstrapping*, which is the initial creation of the network itself and new peers joining it. This is still a challenge for classical P2P networks [5], [6], thus we review some typical bootstrapping methods in Section II-A. As bootstrapping is a core functionality inherent to P2P, it is also required for WebRTC-based P2P networks.

Our contribution in this paper is a decentralized architecture to handle bootstrapping for a WebRTC-based P2P network, which avoids having a single point of failure (SPOF). WebRTC still needs servers to exchange meta data and to cope with Network Address Translators (NATs) [7]. We achieve the bootstrapping by deploying these servers behind DNS-based load balancers around the world. These servers are used in combination with a geolocation approach.

The remainder of this paper is organized as follows: Section II gives an overview on the fundamentals. Related work on browser-based P2P networks is presented in Section III, our bootstrapping architecture is presented in Section IV, and Section V shows experimental results. Finally, the paper ends with a conclusion and future work in Section VI.

II. FUNDAMENTALS

If a component is both a consumer and a producer, it needs both client and server functionality and is then called a peer. A network which connects directly multiple peers is called a P2P network. In such a network, peers can communicate directly without the need for a central server. P2P networks are so-called *overlay networks*. They can be classified as *structured* or *unstructured* [8]. Unstructured networks establish connections between peers in an arbitrary manner, e.g., Gnutella [9] or Kazaa [10]. Here, the location of data is not known. To find data, a peer has to flood the entire network. In structured networks, peers have unique identifiers and the location of data can be associated with a specific peer, e.g., Kademlia [11] or Chord [12]. Because of this, the network behaves like a Distributed Hash Table (DHT). Structured networks organize themselves in topologies:

Full mesh: In a full mesh topology, every peer is connected to every other peer. A joining peer needs to establish a connection to every peer in the network.

Star: In a star topology, every peer is connected to a central peer. The problem of the star topology is that all messages are routed through the central peer which conforms to the classical client-server architecture.

Ring: A common P2P topology is the ring topology. The number of connections in the network depends on the ring algorithm. In Chord, every peer maintains a so-called finger table of size $log_2(N)$.

Tree: A binary tree is another common topology for P2P networks. Examples are Kademlia [11] or P-Grid [13].

A. Bootstrapping Methods

Research in the area of bootstrapping was done by GauthierDickey and Grothoff [15], Cramer et al. [5] or Knoll et al. [16]. They divide the existing bootstrapping methods into *peer-based* and *mediator-based* approaches:

1) Peer-based approaches: These approaches involve technologies where peers can find other peers without the need for a third party.

a) A *Peer Cache* is a database of peers to which a peer was connected previously. To join the network, a peer

 ISBN: 978-1-61208-557-9

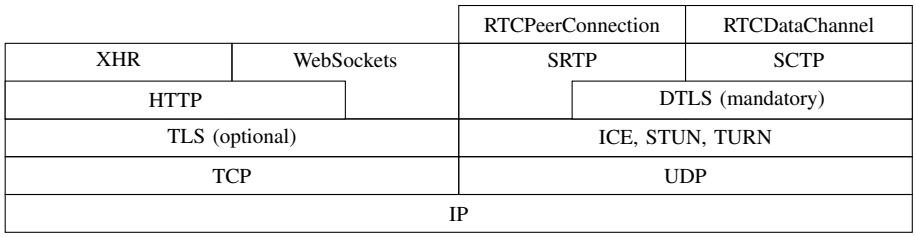


Figure 1. WebRTC protocol stack (Based on [14]).

tries to connect to one peer in its cache. If that fails, the joining peer tries the next peers from the cache. If all peers fail, or a peer joins the P2P network for the first time, a fallback is needed, e.g., a rendezvous server (see below).

b) *Multicast/Broadcast*: Peers in a Local Area Network (LAN) join a multicast group. A joining peer sends a request to that group. As a reply, the peer receives some peers to which it can connect. This approach works well for Universal Plug and Play (UPnP) or zero-configuration networking (Zeroconf) within a LAN, but it does not scale for world-wide networks.

c) *Random IP Probing*: A joining peer randomly selects Internet Protocol (IP) addresses [17] and tries to connect to a specific port. If a host is a peer of the network already, it answers according to the bootstrapping protocol. If not, the joining peer tries another IP address. Obviously, this does not scale on the Internet.

2) Mediator-based approaches: A mediator is a so called *well-known entry point*, which must be known prior to bootstrapping. This entry point should never change and must always be available. Mediators can be peers of the network or separate hosts.

a) *Rendezvous Server*: A rendezvous server is a central server to which a joining peer can connect. The rendezvous server returns a list of active peers, or it forwards the joining request to a peer in the network. This approach is used by Napster [18] as a central index, and by BitTorrent [19], where it is called tracker.

b) *Internet Relay Chat*: Knoll et al. [16] propose to use the Internet Relay Chat (IRC) [20] for bootstrapping, because it is a highly decentralized architecture. A joining peer connects to an IRC channel, where it can communicate with all peers. IRC is a widely distributed and failure tolerant decentralized network, thus it is quite reliable.

B. Bootstrapping Requirements

Bootstrapping can be organized centralized or decentralized. If bootstrapping is centralized, it uses a single central bootstrapping server following the client-server approach, which is a SPOF. The opposite is decentralized bootstrapping, where all components of the bootstrapping are decentralized. Knoll et al. [16] propose five requirements for a decentralized bootstrapping architecture:

1) Robustness against Failure: All components of the bootstrapping should be completely decentralized and the bootstrapping should not exhibit a SPOF, e.g., no central bootstrapping server.

2) Robustness against Security Appliances: Users should not have trouble with their NATs, e.g., users cannot connect to the P2P network without configuring port forwarding.

3) Robustness against External Inference: All components of the bootstrapping should be decentralized. No entity is able to shut down elemental components of the bootstrapping. Additionally, the bootstrapping should still work if the initiator leaves, thus other peers take over the tasks of the initiator.

4) Efficiency: The bootstrapping should be fast and lightweight. This could be the number of messages exchanged for bootstrapping or joining of a peer should happen in a reasonable amount of time.

5) Scalability: The bootstrap must scale with the number of peers in the network.

C. WebRTC

Web Real-Time Communication (WebRTC) does not reinvent the wheel, it is a protocol stack (see Figure 1) based on existing protocols [3] and combines them into the corresponding WebRTC API [4]. WebRTC is used to establish direct P2P connections between two browsers, based on *RTCMediaStreams* [21] and *RTCDataChannels* [22]. Most WebRTC projects are focused on multimedia applications such as telephone conferences and use RTCMediaStreams. RTCDataChannels are a universal channel type in binary-format. They are implemented through encapsulating the Stream Control Transmission Protocol (SCTP) over the User Datagram Protocol (UDP) [23] (SCTP-over-UDP), which allows the configuration of reliability and in-order-delivery without using the Transmission Control Protocol (TCP) [24].

In order to establish a browser-to-browser P2P connection, WebRTC uses the well-known *Offer/Answer Model* [25] from the Session Initiation Protocol (SIP) [26]. This includes *offer* and *answer* messages serialized with the Session Description Protocol (SDP) [27], which contain media capabilities, e.g., for the Real-Time Transport Protocol (RTP) [28]. Additionally, WebRTC collects connectivity information with Interactive Connectivity Establishment (ICE) [29] to search for the most efficient connection between two peers. This is done in three steps: Gathering ICE candidates, exchange offer/answer messages and ICE candidates, and finally running connectivity checks.

In the first step, ICE gathers so-called *ICE candidates*, which are transport addresses (e.g., IP/port tuples) of three types:

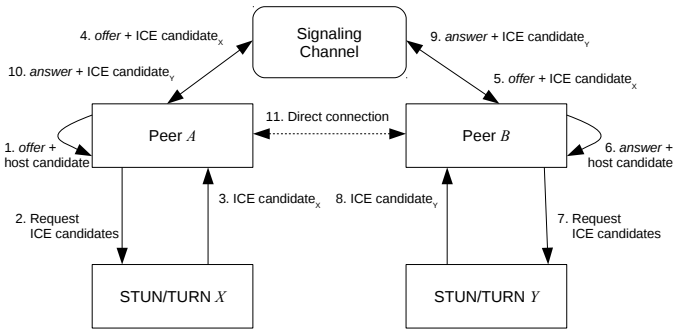


Figure 2. WebRTC Offer/Answer Model, including the offer/answer messages, ICE candidates and the signaling channel.

1) Host Candidates: They contain the local transport address and are obtained directly from the local network interface.

2) Server Reflexive Candidates: They contain the public transport address from the public side of a NAT, which is usually not known to peers behind a NAT. They are obtained by using Session Traversal Utilities for NAT (STUN) [30]. STUN connects via UDP to an external STUN-server, which returns the public transport address. STUN can also be used to keep a public transport address binding alive.

3) Relay Candidates: They contain an external transport address from a publicly available relay server. They are obtained by using Traversal Using Relays around NAT (TURN) [31], which connects to a TURN server, which itself binds a transport address and returns that to the peer.

In the second step, offer/answer messages and ICE candidates are exchanged between two peers through a so-called *Signaling Channel*, which implementation is not specified in WebRTC, so it could be XHR or WebSockets (left side of Figure 1).

Finally, the third step pairs the candidates and both peers perform connectivity checks with all ICE candidate-pairs to see, which pairs work:

1): ICE tries a direct connection with the Host Candidates, e.g., within LANs.

2): If that fails, ICE tries to connect to the Server Reflex Candidates. In case of a Full Cone NAT [32], any peer can send packets to that public transport address, which forwards them to the local transport address of the peer behind the NAT.

3): If this fails as well, i.e., a peer is behind a Symmetric NAT [32], only the initial external host (e.g., the STUN server) can send a packet back to the peer behind the NAT. The fallbacks are the Relay Candidates.

Figure 2 shows all required steps to establish a direct P2P connection with the WebRTC Offer/Answer Model: Peer A creates an offer and initiates a communication with a STUN/-TURN server, and as a result, A receives ICE candidates (steps 1 to 3). Both the offer and the ICE candidates are sent to peer B through the signaling channel (steps 4 and 5). As soon as B receives the offer, it creates an answer and performs the same process (steps 6 to 8) and sends its connectivity information through the same signaling channel back to A (steps 9 and 10). After the signaling is complete, both peers have all connectivity information: answer, offer and both ICE candidates. Finally, both peers perform connectivity-checks with all ICE candidate-pairs to establish a direct connection to each other (step 11).

Observation 1: WebRTC-based P2P networks always require two components for bootstrapping: A signaling mechanism to exchange the offer/answer messages and the ICE candidates (e.g., a signaling server), and a STUN/TURN server to establish connections from/to peers behind NATs. Consequently, WebRTC-based P2P networks are always hybrid P2P networks.

III. Related work

Vogt et al. presented BOPlish in 2013 [33], [34]. They claim that their approach is not supposed to operate on Internet-scale. They use one central bootstrap server, which holds a number of WebSocket connections to peers that have recently joined the network. Thus, a joining peer is able to perform the signaling over these WebSocket connections. If all recently joined peers have left the network, a joining peer will not be able to join the network anymore. Their future work included a JavaScript implementation of Chord.

In 2014, we presented a browser-based P2P network [35]. Our network was based on a so-called *WebSocket SOCKS5 Proxy* (WSSP). Browsers are connecting to the WSSP with WebSockets, where they use the SOCKS5 protocol, Like this, a browser is able to connect to other browsers through the WSSP. We also use the WSSP as the central bootstrapping component. because it is used to handle bootstrapping and to proxy connections. We implemented Chord in JavaScript to create a P2P ring topology. Our future work included a WebRTC-based network.

In parallel, Vogt et al. continued their work on BOPlish which was presented in 2014 [36]. Like our approach, they use Chord to create a P2P ring topology. They refer to a *bootstrapping component* which handles WebRTC specifics like offers and answers. Unfortunately, they do not explain protocol details.

Desprat et al. presented a hybrid client-server and P2P network for a collaborative Computer Aided Design (CAD) in 2015 [37]. Their approach is not designed to operate on Internet-scale. It is made for a small group of peers (max. 7-8 users). They create a WebRTC-based P2P full mesh topology between all peers. The P2P network is used to distribute real-time updates, and a client-server architecture (based on XHR) to persist the CAD data. The bootstrapping uses Peer.js [38]: A new peer connects to a central server which returns a list of Identifiers (IDs) of all existing peers in the network. Now, the new peer connects to a signaling server via WebSockets and gets an ID. The new peer can initiate the signaling and connects to all peers.

Disterhöft and Graffi proposed a WebRTC-based P2P network for social networks in 2015 [39]. Their implementation is based on the Google Web Toolkit (GWT) [40], which allows to create web applications in Java. In Java they used OpenChord [41] to create a ring topology. OpenChord uses native TCP/IP connections, thus they modified it to use Peer.js. Peer.js is used to perform the signaling and to create the connections. As before, the drawback of this approach is the central Peer.js server to handle the bootstrapping, which again is a SPOF.

 ISBN: 978-1-61208-557-9

Bille et al. published RTCSS in 2016 [42]. RTCSS provides an API to create objects that are synchronized between browsers, using a publish/subscribe pattern. For bootstrapping, they use a centralized socket.io-based signaling server [43]. When a new peer connects to this server, it receives its ID and a list of peer IDs existing in the network (similar to Peer.js). Then, the new peer connects to each peers using signaling. Consequently, the peers form a full mesh network. This is unsuitable for very large WebRTC-based networks, as the number of WebRTC connections per browser is limited.

***Observation 2**:* All existing WebRTC-based P2P networks exhibit one SPOF, by using a central bootstrapping.

IV. Bootstrapping Architecture

This section presents a decentralized bootstrapping architecture for WebRTC-based P2P networks, which resolves Observation 2. We start by analyzing the bootstrapping methods from Section II-A, to figure out which methods do work in the context of WebRTC:

Using a *Peer Cache* requires peers to exchange connectivity information to reconnect to a previously known peer. A WebRTC peer has no way to initiate this exchange, once all WebRTC connections are closed. *Multicast/Broadcast* relies on packet forwarding, which is usually not enabled on Internet routers. Consequently, broadcast is not suited for WebRTC. *Random IP Probing* requires peers to listen on a specific port for incoming connections, which is not possible with WebRTC. *IRC* is a protocol built on top of TCP. Using IRC is not feasible in an WebRTC environment, since clients cannot connect directly to TCP sockets. A *Rendezvous Server* acts as a third party, which a browser can reach by using XHR or WebSockets.

As required in *Observation 1*, WebRTC needs a signaling server and a STUN/TURN server. Thus, the only possible bootstrapping method for WebRTC-based P2P networks are Rendezvous Servers. Knoll et al. [16] use the IRC for bootstrapping, because it is an existing, highly decentralized architecture. Since we cannot use IRC, our goal is to use an existing, highly decentralized architecture for bootstrapping as well: The Domain Name System (DNS) [44].

A. Slave Peers and Master Peers

We define that a peer can operate in two ways, *Master* and *Slave*:

Slave Peers are regular peers in the Chord ring. They don't have any special functionalities, except those which are needed for the Chord protocol. Thus, they are perfectly suited for browser environments.

Master Peers are regular peers and act as STUN/TURN servers and as signaling servers, with access to the Chord ring via RTCDataChannels. For STUN/TURN, they need to listen on a fixed port. Consequently, they can only run in a server environment.

B. Signaling Channel

To join the network, peer A connects via WebSockets to a Master Peer M and sends a *findSuccessor*-request, which M forwards in-network via WebRTC to the corresponding peer B. Both, the WebSocket connection between peer A and M as well as the in-network WebRTC connections are the signaling channel to be used for the Offer/Answer Model as explained in Section II-C. A *Signaling Channel* is shown in Figure 3. Having a higher Round Trip Time (RTT) between A and M results in a higher bootstrapping time.

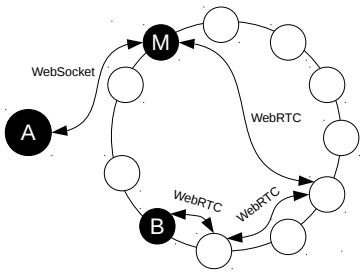


Figure 3. Signaling Channel.

Therefore, our goal is to reduce the RTT of this connection in the next subsections.

C. Scalable Bootstrapping

To allow a scalable way of managing signaling server addresses, we use a load balancer which uses DNS Round Robin Load Balancing [45]. The load balancer only serves as a public DNS entry (i.e., a domain) for some associated Master Peers (see Figure 4). Joining peers do not need to know IP addresses of any specific Master Peer, but only the domain of the load balancer. The DNS lookup returns the IP address of one associated Master Peer and a new peer can join the network.

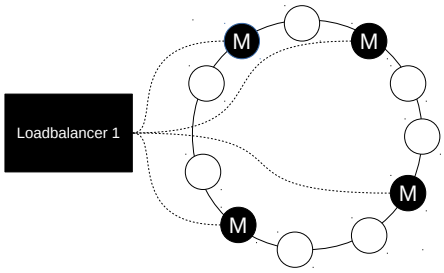


Figure 4. Scalable Bootstrapping.

D. Multiple Load Balancers

As it can be seen in Figure 4, the load balancer is now the SPOF. Therefore, we extend our architecture with multiple DNS-based load balancers, each one with a set of associated Master Peers. This can be seen in Figure 5 (other peers exists between the Master Peers, but are not shown). The Master Peers are located geographically close to the load balancers. Thus, the RTT to the Master Peers is similar to the RTT to the load balancer.

E. Geolocated Bootstrapping

To reduce bootstrapping times, we also use a geolocation-based approach with a distributed set of load balancers around the world. The associated Master Peers are still geographically

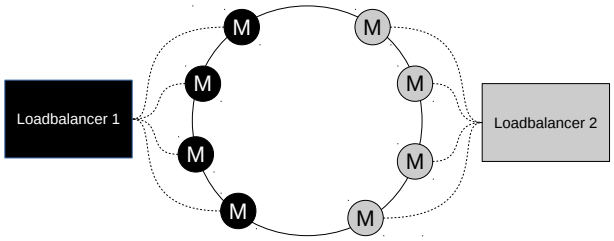


Figure 5. Multiple Load Balancers.

close to the load balancers. The P2P application itself is distributed with a load balancer list with their corresponding latitude- and longitude-coordinates. Given two load balancers and a peer A wants to join the network. It uses the native Geolocation API to retrieve its approximate latitude- and longitude-coordinates. By knowing the coordinates of the load balancers, A connects to the closest available load balancer.

The geolocated bootstrapping only reduces the WebSocket RTT and the STUN/TURN RTT to the Master Peer of the bootstrapping process. Given a global Chord ring, one cannot guarantee that a peer connects to a successor close to it, e.g., the same country or the same continent.

V. Evaluation

A. Experimental Setup and Results

To evaluate our architecture, we use *Amazon Web Services – Elastic Compute Cloud* (AWS EC2) instances and AWS Elastic Load Balancers, which use DNS Round Robin Load Balancing. Each Master Peer can be started through Amazon Machine Images (AMIs) and can be automatically registered to the load balancer in an AWS Auto Scaling Group. As a STUN/TURN server, we use coTURN [46]. We only measure the bootstrapping time from a joining peer A in central Europe (located in Lübeck) to one Master Peer M, deployed in each AWS region. Like this, we prevent in-network messages, which can span the whole globe. We divide the measurement into three steps:

1) TCP and WebSocket handshake time,
2) Chord time, e.g., from *findSuccessor*-Request until the successor is returned, and
3) WebRTC process time, until the connection is opened (incl. offer, answer, ICE).

Figure 6 shows our result, which is an average value for 25 tests per region. It can be seen, that the bootstrapping increases with the distance to the Master Peer. Bootstrapping with a Master Peer in region *eu-central-1* (Frankfurt) needs 778ms, while bootstrapping with a Master Peer in region *ap-southeast-2* (Sydney) needs 3777ms. This shows, that our decentralized bootstrapping reduces the bootstrapping time remarkably. The box plots in Figure 7 show the detailed distribution of the three evaluated steps.

26 messages are exchanged between a joining peer A and a Master Peer M during the bootstrapping process:

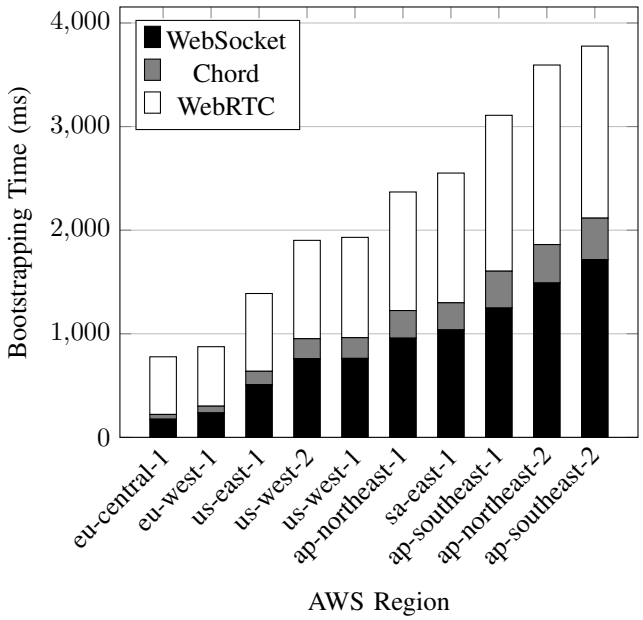


Figure 6. Average value for 25 tests per AWS region.

TCP and WebSocket handshake (4 messages)
A sends a TCP-SYN to M receives a TCP-ACK from M
A sends WebSocket-request to M and receives the response

Chord (2 messages)
A sends the findSuccessor-request to M and receives the findSuccessor-response

WebRTC (20 messages)
A sends a STUN request to M and receives a Server Reflexive Candidate
A sends a TURN request to M and receives a Relay Candidate
A sends the offer to M and receives an acknowledgement
A sends the Host Candidate to M and receives an acknowledgement
A sends the Server Reflexive Candidate to M and receives an acknowledgement
A sends the Server Relay Candidate M and receives an acknowledgement
A receives the answer from M and sends an acknowledgement
A receives the Host Candidate to M and sends an acknowledgement
A receives the Server Reflexive Candidate to M and sends an acknowledgement
A receives the Server Relay Candidate M and sends an acknowledgement

This number of messages is the smallest possible with WebRTC.

B. Bootstrapping Requirements

Finally we evaluate our bootstrapping wrt. to Section II-B:

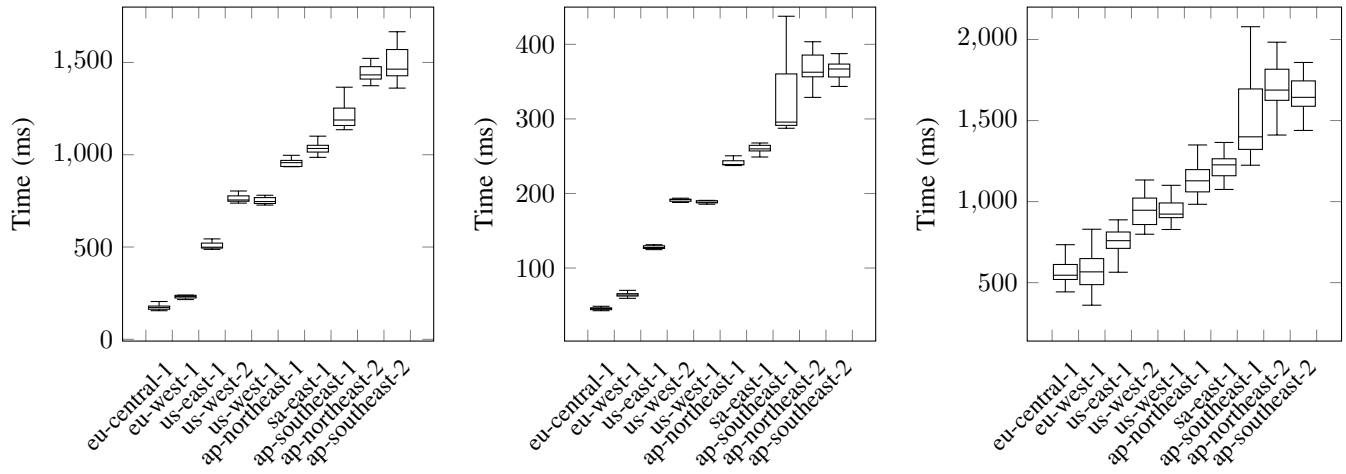
1) Robustness against Failure: We have a geolocated bootstrapping with multiple load balancers and multiple Master Peers. Thus, we do not have a SPOF.

2) Robustness against Security Appliances: WebRTC uses STUN/TURN and ICE, which handles peers behind NATs. Therefore, user do not have to configure any port forwarding.

3) Robustness against External Inference: All components of the bootstrapping are decentralized. Even if the initial Master Peer leaves the Auto Scaling Group, other Master Peers will be available.

4) Efficiency: Because of the geolocated bootstrapping, a peer always selects the closest load balancer. The RTT to a Master Peer is small and the bootstrapping happens in the shortest possible time (see Section V-A). The number of messages exchanged is the smallest possible with WebRTC.

 ISBN: 978-1-61208-557-9



(a) TCP and WebSocket handshake time. (b) Chord time. (c) WebRTC process time.

Figure 7. Detailed distribution of the three evaluation steps.

5) Scalability: If the load gets high, additional Master Peers associated with the load balancer will be started automatically by the Auto Scaling Group. If the load gets low, Master Peers are removed. Thus, our bootstrapping scales. Additionally, the DNS is a well-working decentralized and scalable system, thus we rely on it.

All requirements are covered. So, we achieved a decentralized bootstrapping architecture for a WebRTC-based P2P network. Note that just the bootstrapping is decentralized. WebRTC-based P2P networks itself are still hybrid P2P networks, because they require a signaling server and a STUN/-TURN server. These servers are addressed with well-known domains.

VI. Conclusion and Future work

In this paper, we described a decentralized architecture to handle the bootstrapping for WebRTC-based P2P networks. Our architecture is based on Master Peers associated with a DNS-based load balancer. We have chosen DNS, because it is an existing, highly decentralized architecture which is used by browsers. Additionally, we employed a geolocation-based bootstrapping to reduce the bootstrapping time.

Our future work is mainly focused on security. Feher et al. [47] provide an overview on the security of WebRTC, which can be a starting point in order to improve our bootstrapping architecture. Since we use the DNS Round Robin Load Balancing for bootstrapping, Domain Name System Security Extensions (DNSSEC) [48] must be taken into account. Because STUN and TURN support TLS-over-TCP, a secure connection to the STUN/TURN server can be used by ICE to collect the connectivity information. WebSockets support TLS-over-TCP as well, thus the connection to the Master Peer can be secured too. For TLS it is possible to use certificates issued by Let's Encrypt [49], since they support the Automatic Certificate Management Environment (ACME) protocol [50]. The established WebRTC connections between the peers are secured per specification of the WebRTC protocol. Even all connections are secured, intermediate nodes in the P2P network are still able to eavesdrop and to modify the bootstrapping messages (i.e., connectivity information, offer and answer messages). Thus, a confidential and authenticated end-to-end connection to exchange the bootstrapping messages is required.

References

[1] A. van Kesteren, J. Aubourg, J. Song, and H. Steen, "XMLHttpRequest Level 2," W3C Working Group Note, Nov. 2014. [Online]. Available: http://www.w3.org/TR/XMLHttpRequest2/ [retrieved: April, 2017]

[2] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455 (Proposed Standard), Internet Engineering Task Force, Dec. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6455.txt [retrieved: April, 2017]

[3] H. Alvestrand, "Overview: Real time protocols for browser-based applications," Internet Engineering Task Force, Internet Draft, draft-ietf-rtcweb-overview-18, Mar. 2017. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-overview-18 [retrieved: April, 2017]

[4] A. Bergkvist, D. C. Burnett, C. Jennings, A. Narayanan, and B. Aboba, "WebRTC 1.0: Real-time communication between browsers," World Wide Web Consortium, Working Draft, WD-webrtc-20170313, Mar. 2017. [Online]. Available: https://www.w3.org/TR/2017/WD-webrtc-20170313/ [retrieved: April, 2017]

[5] C. Cramer, K. Kutzner, and T. Fuhrmann, "Bootstrapping locality-aware p2p networks," in Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on, vol. 1. IEEE, 2004, pp. 357–361.

[6] J. Dinger and O. P. Waldhorst, "Decentralized bootstrapping of p2p systems: a practical view," in NETWORKING 2009. Springer, 2009, pp. 703–715.

[7] S. Dutton, "Webrtc in the real world: Stun, turn and signaling," html5rocks.com, Jul. 2012. [Online]. Available: https://www.html5rocks.com/en/tutorials/webrtc/basics/ [retrieved: April, 2017]

[8] V. Vishnumurthy and P. Francis, "A comparison of structured and unstructured p2p approaches to heterogeneous random peer selection." in Usenix Annual Technical Conference, 2007, pp. 309–322.

[9] E. Adar and B. A. Huberman, "Free riding on gnutella," First monday, vol. 5, no. 10, 2000, pp. 1–22. [Online]. Available: http://firstmonday.org/issues/issue5_10/adar/index.html [Retrieved: April, 2017]

[10] J. Liang, R. Kumar, and K. W. Ross, "Understanding kazaa," Manuscript, Polytechnic Univ, 2004, p. 17.

 ISBN: 978-1-61208-557-9

[11] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in Peer-to-Peer Systems. Springer, 2002, pp. 53–65.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in ACM SIGCOMM Computer Communication Review, vol. 31, no. 4. ACM, 2001, pp. 149–160.

[13] K. Aberer et al., "P-grid: a self-organizing structured p2p system," ACM SIGMOD Record, vol. 32, no. 3, 2003, pp. 29–33.

[14] I. Grigorik, "High performance browser networking," 2013. [Online]. Available: https://hpbn.co/webrtc/ [retrieved: April, 2017]

[15] C. GauthierDickey and C. Grothoff, "Bootstrapping of peer-to-peer networks," in Applications and the Internet, 2008. SAINT 2008. International Symposium on. IEEE, 2008, pp. 205–208.

[16] M. Knoll, A. Wacker, G. Schiele, and T. Weis, "Decentralized bootstrapping in pervasive applications," in Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on. IEEE, 2007, pp. 589–592.

[17] J. Postel, "Internet Protocol," RFC 791 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1349, 2474, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt [retrieved: April, 2017]

[18] P. Mahlmann and C. Schindelhauer, Peer-to-Peer-Netzwerke: Algorithmen und Methoden. Springer, Jul. 2007, ISBN: 978-3-540-33992-2.

[19] B. Cohen, "The bittorrent protocol specification," Jan. 2008. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html [retrieved: April, 2017]

[20] C. Kalt, "Internet Relay Chat: Client Protocol," RFC 2812 (Informational), Internet Engineering Task Force, Apr. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2812.txt [retrieved: April, 2017]

[21] C. Perkins, M. Westerlund, and J. Ott, "Web real-time communication (webrtc): Media transport and use of rtp," Internet Engineering Task Force, Internet Draft, draft-ietf-rtcweb-rtp-usage-26, Mar. 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-rtp-usage-26 [retrieved: April, 2017]

[22] R. Jesup, S. Loreto, and M. Tuexen, "Webrtc data channels," Internet Engineering Task Force, Internet Draft, draft-ietf-rtcweb-data-channel-13, Jan. 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-13 [retrieved: April, 2017]

[23] J. Postel, "User Datagram Protocol," RFC 768 (INTERNET STANDARD), Internet Engineering Task Force, Aug. 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt [retrieved: April, 2017]

[24] ——, "DoD standard Transmission Control Protocol," RFC 761, Internet Engineering Task Force, Jan. 1980. [Online]. Available: http://www.ietf.org/rfc/rfc761.txt [retrieved: April, 2017]

[25] J. Rosenberg and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)," RFC 3264 (Proposed Standard), Internet Engineering Task Force, Jun. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3264.txt [retrieved: April, 2017]

[26] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol," RFC 2543 (Proposed Standard), Internet Engineering Task Force, Mar. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2543.txt [retrieved: April, 2017]

[27] M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC 4566 (Proposed Standard), Internet Engineering Task Force, Jul. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4566.txt [retrieved: April, 2017]

[28] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550 (INTERNET STANDARD), Internet Engineering Task Force, Jul. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt [retrieved: April, 2017]

[29] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," RFC 5245 (Proposed Standard), Internet Engineering Task Force, Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5245.txt [retrieved: April, 2017]

[30] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389 (Proposed Standard), Internet Engineering Task Force, Oct. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5389.txt [retrieved: April, 2017]

[31] R. Mahy, P. Matthews, and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)," RFC 5766 (Proposed Standard), Internet Engineering Task Force, Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5766.txt [retrieved: April, 2017]

[32] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489 (Proposed Standard), Internet Engineering Task Force, Mar. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3489.txt [retrieved: April, 2017]

[33] C. Vogt, M. J. Werner, and T. C. Schmidt, "Content-centric user networks: WebRTC as a path to name-based publishing," in Network Protocols (ICNP), 2013 21st IEEE International Conference on, Oct 2013, pp. 1–3.

[34] ——, "Leveraging WebRTC for P2P content distribution in web browsers," in Network Protocols (ICNP), 2013 21st IEEE International Conference on, Oct 2013, pp. 1–2.

[35] D. Boldt and S. Fischer, "Return the Data to the Owner: A Browser-Based Peer-to-Peer Network," The Ninth International Conference on Internet and Web Applications and Services, Jul. 2014, pp. 140–146. [Online]. Available: http://www.thinkmind.org/download.php?articleid=iciw_2014_7_30_20082 [retrieved: April, 2017]

[36] M. J. Werner, C. Vogt, and T. C. Schmidt, "Let our browsers socialize: Building user-centric content communities on webrtc," in Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on. IEEE, 2014, pp. 37–44.

[37] C. Desprat, H. Luga, and J.-P. Jessel, "Hybrid client-server and P2P network for web-based collaborative 3D design," in Conference on Computer Graphics, Visualization and Computer Vision, 2015. WSCG 2015. World Society for Computer Graphics, Jun. 215, pp. 229–238.

[38] M. Bu and E. Zhang, "PeerJS – Simple peer-to-peer with WebRTC." [Online]. Available: http://peerjs.com [retrieved: April, 2017]

[39] A. Disterhoft and K. Graffi, "Protected chords in the web: secure p2p framework for decentralized online social networks," in Peer-to-Peer Computing (P2P), 2015 IEEE International Conference on. IEEE, 2015, pp. 1–5.

[40] "Google Web Toolkit." [Online]. Available: http://www.gwtproject.org/ [retrieved: April, 2017]

[41] "OpenChord." [Online]. Available: http://open-chord.sourceforge.net/ [retrieved: April, 2017]

[42] R. J. Bille, Y. Lin, and S. K. Chalup, "Rtcss: a framework for developing real-time peer-to-peer web applications," in Proceedings of the Australasian Computer Science Week Multiconference. ACM, 2016, p. 56.

[43] "Socket.io." [Online]. Available: http://socket.io [retrieved: April, 2017]

[44] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt [retrieved: April, 2017]

[45] T. Brisco, "DNS Support for Load Balancing," RFC 1794 (Informational), Internet Engineering Task Force, Apr. 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1794.txt [retrieved: April, 2017]

[46] "coturn TURN server project." [Online]. Available: https://github.com/coturn/coturn [retrieved: April, 2017]

[47] B. Feher, L. Sidi, A. Shabtai, and R. Puzis, "The security of webrtc," arXiv preprint arXiv:1601.00184, Jan. 2016. [Online]. Available: https://arxiv.org/abs/1601.00184 [retrieved: April, 2017]

[48] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033 (Proposed Standard), Internet Engineering Task Force, Mar. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4033.txt [retrieved: April, 2017]

[49] "Let's Encrypt – Free SSL/TLS Certificates." [Online]. Available: https://letsencrypt.org/ [retrieved: April, 2017]

[50] R. Barnes, J. Hoffman-Andrews, and K. J., "Automatic Certificate Management Environment (ACME)," Internet Engineering Task Force, Internet Draft, draft-ietf-acme-acme-06, Mar. 2017. [Online]. Available: https://tools.ietf.org/html/draft-ietf-acme-acme-06 [retrieved: April, 2017]

 ISBN: 978-1-61208-557-9