Decentralized Communications: Trustworthy Interoperability in Peer-To-Peer Networks

Paulo Chainho*, Steffen Drüsedow†, Ricardo Lopes Pereira§,
Ricardo Chaves§, Nuno Santos§, Kay Haensge†, Anton Roman Portabales¶

*Altice Labs, †Deutsche Telekom AG, §INESC-ID / IST, University of Lisbon, ¶Quobis

***Abstract*—This paper introduces a new communication paradigm called *Decentralized Communications*, which enables cross-domain communication services to trustfully use peer-to-peer networks. Decentralized Communication services are inherently inter-operable without the need to standardize protocols or service APIs. This property is achieved by using the Protocol-on-the-fly concept and the Reporter-Observer communication pattern. Users can select Identity Providers to mutually authenticate themselves and secure communications, independently of the Communication Service Provider. Decentralized Communications can be applied to any kind of communication, including human-to-human, human-to-things, and things-to-things communication. The reTHINK project has developed an Open Source reference implementation of a Decentralized Communications framework, which was successfully used in various challenging scenarios. The results of this project demonstrate the feasibility to enhance interoperability and users' privacy, and the freedom to select whom to trust, without slowing down ICT innovation pace. It is expected that Decentralized Communications will impact the design of more agile Service Delivery frameworks for Internet of Things and 5G networks.**

Keywords*—Decentralized communication, peer-to-peer, hyperlinked entities, protocol on-the-fly, identity management, WebRTC.

I. Introduction

The impressive fast pace of innovation of the Web is fostered by Global Web Players. Although the benefits to consumers are countless, they should be protected from the abuse of dominant market positions. Web Monopolies are natural, but it means Web Communication Services only work when users use the same Service [1]. The positive network effect requires a critical mass of hundreds of millions of peers, as the value of a product to one user depends on how many other users there are. The more peers are available to communicate with, the more valuable the Service becomes.

With Web Monopolies, the fundamental users' freedom to select whom to trust their data and select which Services to be consumed are curtailed. Users are forced to consume Services from the same organization in order to communicate or share any kind of resource to each other. If users want to be in contact and reachable from different Services, they have to manage different Accounts and Applications. Before a communication is accomplished, the user has to discover which Service is used by the recipient. The data that is generated from the Service consumption as well as a user's own identity, are managed by the very same Service provider, giving no choice to users as to whom to trust. The same problem pattern applies to communication between humans and things (Internet of Things). However, the scale and the complexity of the problem for IoT communications is much higher.

Usually these kinds of problems are addressed with strongly regulated and standardized services, as the ones delivered by Telecommunication operators, including GSMA (Global System Mobile Association) mobile telephony and SMS (Short Message Service) [2]. Services are standardized to ensure they inter-operate with each other regardless of the Telecom Operator they are subscribed from. However, reaching agreements on standards is a very complex activity. Standards and rules in general (including regulating policies) in a fast moving area as the ICT (Information and Communication Technology) constraints stakeholders' freedom to innovate with alternative technologies or processes.

Decentralized Communications introduces a novel approach to solve these issues with non-standards and a full decentralized architecture. A Decentralized Communication infrastructure enables a Web of native inter-operable Services that are able to securely communicate in a peer to peer mode without having to agree on common network protocols or service APIs, dramatically reducing the standardization effort.

The remainder of this paper is organized as follows. The conceptual foundations underlying the design of Decentralized Communications, including the Protocol on-the-fly and the Reporter - Observer communication pattern concepts, are described in Section II. A reference implementation of a Decentralized Communications framework, developed in the reTHINK Project, is described in Section III, that was evaluated as described in section IV. The paper concludes with related work in Section V and conclusions in Section VI.

II. Decentralised Communication Concepts

The conceptual foundations of Decentralized Communications are illustrated in Figure 1, which includes a *Decentralized Messaging Framework* providing message delivery in a peer-to-peer mode, the *Protocol on-the-fly* mechanism providing transport interoperability without requiring to standardise messaging protocols, the *Reporter – Observer data stream synchronisation* communication pattern that enables semantic interoperability, and the *Decentralised Trust Framework* that enables trustworthy interoperability by using independent Identity Providers. These concepts are further detailed in the

978-1-5386-3873-6/17/$31.00

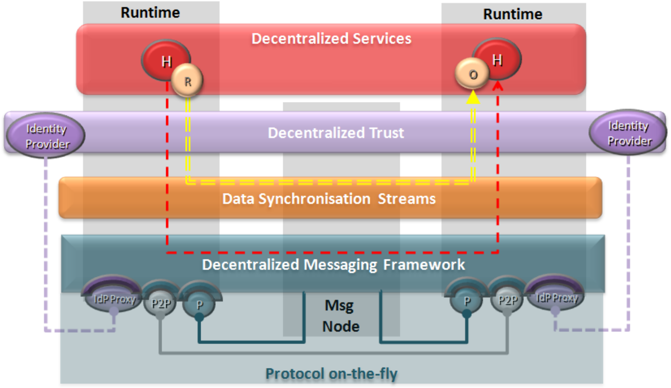

Fig. 1. Decentralized Communication Concepts.

following sections and a reference implementation is discussed in section III.

A. Decentralized Messaging Framework

Decentralised Communications are built on top of a Resource Oriented Messaging model that supports publish/subscribe as well as request/response messaging patterns. Messages are used to perform CRUD (Create, Read, Update, Delete) operations on resources handled by communication endpoints, for example to create or update a WebRTC connection.

The Message delivery is based on a network of routers where each router only knows adjacent registered routers or end-points. The routers forward messages to all registered listeners, which can be other routers or the final recipient end-point. Listeners are programmatically registered and unregistered by Routing Management functionalities, which take their decisions according to a higher level view of the Routing Network.

There are different layers of routers in the Decentralized Communication architecture. The lower layers are responsible for the communication between components on a single end-point. The upper layers - the Message Nodes - route messages between end-points in the same service provider domain or even between different domains. The Decentralized Messaging Framework is protocol agnostic, i.e. domains with different internal messaging protocols (e.g. SIP, Matrix) can exchange message with each other. This is achieved by Message Nodes supporting the Protocol on-the-fly concept, which is described in the next section.

B. Protocol on-the-fly (Protofly)

Protocol on-the-fly extends the usage of signalling on-the-fly concept [5] to ensure messaging protocol interoperability between any distributed software Services. Protocol on-the-fly leverages the code on-demand support by runtime engines (e.g. JavaScript) to dynamically select, load, and instantiate the most appropriate protocol stack at run-time. This feature enables protocols to be selected at run-time and not at design time, which brings several benefits, namely: enables protocol interoperability among distributed Services, promotes loosely coupled Service architectures, makes platform updates much easier, and minimizes standardization efforts and optimizing resources spent. These benefits stem from the fact that Protocol Gateways can be avoided in Services' middleware. The implementation of the protocol stack (e.g. as a JavaScript file) which is dynamically loaded and instantiated at run-time is called a ProtoStub.

C. Semantic Interoperability: Reporter - Observer data stream synchronisation

While the Protofly provides transport interoperability without requiring the standardisation of messaging protocols, the Reporter – Observer communication pattern enables semantic interoperability between Services without having to standardize Service APIs. This pattern extends existing Observable communication patterns by using a P2P data stream synchronization solution for programmatic Objects, e.g. JSON Objects, hereafter simply called Data Objects [6]. To avoid concurrency inconsistencies between peers, only one peer is granted writing permissions to the Data Object – the Reporter service. All the other service instances have permissions only to read the Data Object – the Observers. As soon as the Reporter performs changes to Data Objects, they are immediately propagated to any authorized Observer by using the messaging framework. In this way, the Data Object monitored by the Observer is always synchronized with the Data Object owned by the Reporter. Full interoperability is achieved between two service instances by having to agree only on the usage of common formats for the Data Objects.

To be noted that, conceptually, more complex semantic interoperability and data synchronization technologies, such as Semantic Web and Operational Transformation, can be used.

D. Decentralized Trust

With Decentralized Communications, services are securely associated to User Identities that are managed by independent Identity Providers. The end-user is empowered to decide about which Identity Provider to trust, i.e. User Identities are decoupled from Service Providers. Users are human beings (including group of human beings e.g. corporation) or things (including group of things and physical spaces, e.g. a smart home or smart building). The Decentralized Communication Trust Model extends the WebRTC Identity model where Identity tokens are generated, inserted in intercepted Messages sent by Services, and validated before being delivered to the target Service [4].

E. Decentralized Microservices: Hyperties

Hyperties is a new Service paradigm designed according to Decentralised Communication principles. It follows Microservices architectural patterns, i.e. Hyperties are independently deployable components. Each Hyperty provides a small set of business capabilities using the smart endpoints and dumb pipes philosophy. Hyperties also follow emerging Edge and Fog computing paradigms [7] as opposed to more popular Cloud Computing and they tend to be executed as much as possible in end-users devices. Hyperties are dynamically loaded from a Catalog Support Service and instantiated in the runtime environment, when required by Applications. Each Hyperty instance is registered in a Registry Support Service to become discoverable.

F. Quality of Service

Providing dedicated quality of service for real time communications may complement decentralized communication services.

One approach is based on a *Last Hop Connectivity Broker* (LHCB) which is able to select the best network to attach before the actual communication starts. The LHCB aims at providing means to obtain information on the various alternative networks available at a client for communication, as well as on QoS parameters of those.

A second approach is network-based. It relies on traffic management in the network during communications and covers several loosely coupled schemes between Communication Service Provider, Network Service Provider and their end-users. Leveraging the network capabilities, a brokering instance may provide the most feasible communication path for an optimal real-time communication set-up. Another approach for network-based quality support is to use policies, predefined inside the network to support communication stability as well as quality enhancements. This mechanism is possible in enterprise environments where the managing IT unit is able to provide such rule-sets on the network level.

III. Implementation

The reTHINK Project has developed a reference implementation of a Decentralized Communication framework which is available as Open Source (Apache 2.0 license) in reTHINK Github repositories [8]. The reference implementation of the Hyperty Runtime and of the Decentralized Messaging framework are described in the following sections.

A. Hyperty Runtime

The Hyperty Runtime supports the execution of Hyperties providing all required functionalities to securely manage its life-cycle, only consuming back-end support Services when strictly required. Thus, the Runtime features a catalogue functionality from where Hyperties source packages are deployed, as well as a registry functionality to handle the registration of Hyperty instances in order to make the Hyperties reachable within the runtime.

The Runtime design enables the reuse of the core runtime components through different platforms including Browsers, Standalone Mobile Application, Network Side Application Servers and more constrained M2M/IoT standalone devices.

The Hyperty Runtime is designed to support the execution of multiple untrusted software components, namely Applications, Hyperties, ProtoStubs, and Hyperty Core Runtime components. Therefore, to provide for the overall system security, it is necessary to ensure that different components execute in isolation from each other and to restrict their communication path to secure channels.

To enforce isolation, the Hyperty Runtime implements a sandboxing mechanism which confines components downloaded from different Service Providers to independent sandboxes. Applications and Hyperties may or may not run within the same sandbox depending on their trust level. If they are downloaded from the same Service Provider, it is assumed that they trust each other and that they can share the same sandbox. Otherwise, Hyperties and Application run in different sandboxes. On the other hand, Protocol Stubs and Hyperties live in separate sandboxes even if they are distributed by the same Service Provider. To preserve compatibility with existing device native runtimes, Hyperty Core Runtime components are downloaded from the Hyperty Runtime Service Provider and executed in a sandbox named Core Sandbox. The Core Sandbox is responsible for the deployment, execution, and supervision of components downloaded from Service Providers.

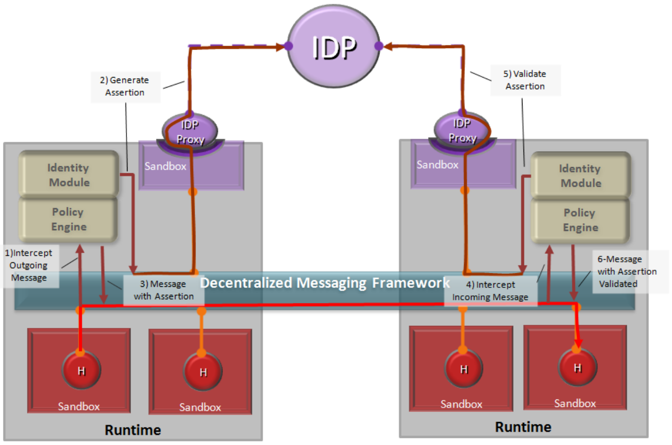

Fig. 2. Decentralized Trust implementation.

Communication between components residing in different sandboxes is possible only through messages exchanged via the Message Bus component located in the Core Sandbox. To communicate with a Service Provider, a Protocol Stub playing the role of a bridge between the Hyperty Runtime and the Service Provider, is used. If Hyperty and Application share the same sandbox, they can communicate directly through a local API, otherwise they have to exchange messages through the Message Bus.

As illustrated in Figure 2, the Policy Engine uses the Identity Module to enforce user trust policies, e.g. identity assertions and communication encryption:

1) Outgoing messages are intercepted by the Policy Engine,
2) The Policy Engine requests the Identity Module to generate an Identity Assertion through the IdP Proxy,
3) The Policy Engine embeds the Identity Assertion into message body before it is routed outside the runtime,
4) In the recipient runtime, the incoming message is intercepted by the Policy Engine,
5) The Policy Engine requests the Identity Module to validate the Identity Assertion from the message sender through the IdP Proxy,
6) If successful, the Policy Engine authorizes the delivery of the message to the recipient with validated identity assertion.

This mechanism is used to support Mutual Authentication and to generate symmetric keys to encrypt communications.

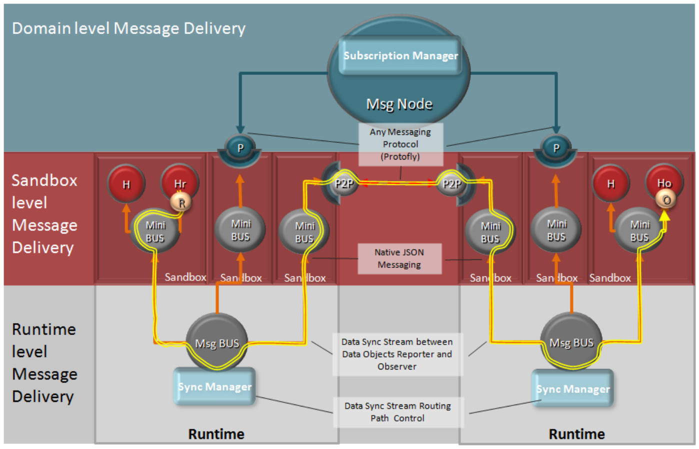
Domain level Message Delivery
Subscription Manager
Msg Node
Any Messaging Protocol (Protofly)
P
P
Sandbox level Message Delivery
H
Hr
R
Mini BUS
Mini BUS
Mini BUS
P2P
P2P
Mini BUS
Mini BUS
H
Ho
O
Mini BUS
Native JSON Messaging
Sandbox
Sandbox
Sandbox
Sandbox
Sandbox
Sandbox
Runtime level Message Delivery
Msg BUS
Data Sync Stream between Data Objects Reporter and Observer
Msg BUS
Sync Manager
Data Sync Stream Routing Path Control
Sync Manager
Runtime
Runtime

Fig. 3. Implementation of the Messaging Framework and Reporter Observer Data Sync Stream.

B. Messaging Framework implementation

The Decentralized Messaging Framework was implemented at three layers (Figure 3): (1) at the Runtime Sandbox level, where Services are executing, messages are delivered by a minibus, (2) at the Runtime level, where Sandboxes are hosted (e.g. in a Browser or in a NodeJS instance), messages are delivered by a Message Bus, and (3) at Domain Level, where message delivery is provided by the Message Node functionality using the Protofly mechanism, i.e. communication between Message Bus and Message Nodes and among Message Nodes are protocol agnostic. This also means that the Message Node can be provided by any Messaging solution as soon as there is a ProtoStub available.

The Data synchronization streams established between Data Object Reporters and Observers (see example in Figure 3) are managed at the Runtime level by the Sync Manager component and at Domain Level by the Message Node Subscription Manager functionality. Services use a Syncher library to manage the Data Objects Reporters and Observer, which handles the required exchange of messages with the Sync Manager component to control the Data Syncronization stream. The Sync Manager exchanges messages with the Message Node Subscription Manager to control the Data Synchronization Stream between Runtime and Domain Message Node.

Message Nodes are responsible for routing messages between Hyperty Runtimes on intra- and inter-domain level. The main tasks of Messages Nodes are:

- Assignment of unique addresses to entities (Hyperties and Data Objects),
- Management of routing paths between entities,
- Gateway to support Services in a Service Provider domain,
- Enforcement of provider policies concerning the message routing.

Message Nodes can either be implemented from scratch, as a stand-alone solution, or they can make use of existing message routing systems and enrich them with support Services for the tasks listed above. It is the ProtoStub that acts as the "glue" between the reTHINK runtime and the Message Node. Therefore, each Message Node and its corresponding ProtoStub forms a unit that must be designed and implemented together. All ProtoStubs must handle a common message format, but the transport protocol between the ProtoStub and the Message Node as well as internally in the Message Node are implementation specific.

Depending on the complexity and flexibility of the Message Node, different ProtoStub models are possible. In one extreme, the ProtoStub might just be a pipe that transports messages completely un-touched. The other extreme is a ProtoStub that already analyses and translates reTHINK messages to another format suitable for the specific Message Node.

ReTHINK provides reference implementations of Message Nodes based on Vert.x [9], NodeJS [10], and Matrix.org [11].

ProtoStub are also used for interworking with any open legacy Communication Services using different types of network protocols avoiding the need to use Interworking Gateways. Interworking ProtoStubs were developed for IMS telephony using SIP over Websocket protocol [12] and for Slack Group Chat using Slack REST API [13].

IV. Evaluation

A set of Hyperties and Applications were implemented to evaluate Decentralized Communication reference implementation, including two WebRTC Hyperties providing Audio and Video communications, two Group Chat Hyperties, a Mybracelet Hyperty to collect and publish data from a connected bracelet and two context management Hyperties to manage user's location and availability.

These Hyperties were used to implement several scenarios, including a Call Center Application and a Smart Contextual Assistance application both featuring chat, audio and video communication, from different Hyperties. Interoperability tests were successfully performed between these two applications, each one provided from different domains. It was also successfully tested interoperability with telephony in IMS and Group Chat in Slack, demonstrating that Decentralized Communications are inter-operable with any open legacy Communication Services, avoiding the Service silos. The legacy Service only has to provide an Inter-working ProtoStub and an appropriate Inter-working IdP Proxy in order to achieve trustworthy interconnections with non-Decentralized Communication endpoints.

Preliminary performance tests are also very promising. For example, around 5000 messages / sec are currently supported when exchanged between Hyperties while data synchronization for 200 observers take around 400ms.

V. Related Work

Decentralized Communications takes advantage of previous work done in several areas to introduce a novel communication paradigm.

Decentralized network topologies were introduced in Paul Baran's 1962 seminal Work [14], providing the foundations to design the Internet and the Web with no single point of control and failure. More recently, a few initiatives are ongoing to address privacy, security and preservation by default for web content [15].

The *signalling on-the-fly* concept was proposed in 2015 [5] to ensure interoperability between any WebRTC administrative domains without the need to use a standardized signalling protocol such as Session Initiation Protocol (SIP). Decentralized Communication extends the usage of signalling on-the-fly by introducing the Protocol on-the-fly concept to ensure messaging protocol interoperability between any distributed software Services.

With respect to the runtime security, different architectures have been proposed in the literature. JSand [16] is a sandboxing mechanism that isolates third-party Javascript code from surrounding browser components. Isolation is enforced through *code wrappers* that prevent unauthorized accesses to the environment according to user-defined access control policies. ConScript [17] and WebJail [18] provide isolation not through code wrappers but by *native sandboxing mechanisms* available on the browser. Whereas in ConScript the security policies that control execution of the scripts apply to single web page components, WebJail provides richer policy support, allowing multiple components of web mashups to be supervised. reTHINK takes one step further by providing a flexible sandboxing model for isolation and access control of Hyperty and application code. This flexibility is achieved by allocating these components into independent native sandboxes and routing all messages through a Message Bus where security policies can be centrally enforced.

WebRTC technology enabled browsers to establish direct communications, enabling the use of the Peer-to-Peer concept, opening the door to a whole new range of opportunities that were previously limited to native applications. However, WebRTC does not define how to handle the peer discovery and the exchange of capabilities. The latter requires the two browsers to exchange information before being able to establish the P2P communication channel. Different approaches have been followed to tackle these issues. Most rely on the use of centralized servers [19], which have scalability and privacy issues. Furthermore, collaboration is limited to the users of the same server. An alternative approach relies on a centralized server only to establish a global P2P network among browsers, which is later used for discovery and exchange of capabilities [20]. While this approach reduces privacy concerns, latency and peer churn limit its performance. Decentralized Communications uses a hybrid approach, that separates peer discovery from capabilities exchange. Peers access to the Registry Support Service, a scalable hybrid P2P DHT and REST server architecture, where all services providers participate in the DHT (Global Registry) and each run their own server (Local Registry) [3]. After a peer has been discovered, communication between the local ProtoStub and the remote Hyperty is performed using the appropriate Message Node, which provides the means for the exchange of the signaling messages necessary to establish a P2P communication channel.

VI. Conclusion

Decentralized Communications have been discussed in this paper as a new Communication paradigm to address a highly fragmented market of mostly isolated communication platforms that restrict free, open, and inter-operable communication flows. Decentralized Communications leverage the best from Telecommunication federated model and from Web Players Walled Garden model to create a new trusted worldwide communication model. Decentralized Communications address Service scenarios not only between human entities but all possible variations between human and non-human entities. The reTHINK results pave the way for the design of new Service Delivery Frameworks that are more agile, protect users' freedom to select whom to trust and which Services to be consumed, without slowing down ICT innovation pace.

Acknowledgment

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 645342; project reTHINK.

References

[1] *Everybody wants to rule the world*, http://www.economist.com/news/briefing/21635077-online-businesses-can-grow-very-large-very-fastit-what-makes-them-exciting-does-it-also-make, last accessed February 20th, 2017.

[2] *Global System Mobile Association: Universal Profile*, http://www.gsma.com/network2020/universal-profile/, 2017.

[3] Friese, Ingo and Beierle, Felix and Copeland, Rebecca and Lopes Pereira, Ricardo and Göndör, Sebastian and Crom, Jean-Michel, *Cross-Domain Discovery of Communication Peers*, In European Conference on Networks and Communications, 2017

[4] Bergkvist, Adam and Burnett, Daniel C and Jennings, Cullen and Narayanan, Anant, *WebRTC 1.0: Real-Time Communication between Browsers*, Working draft, W3C, 2012.

[5] Chainho, Paulo and Haensge, Kay and Drüsedow, Steffen and Maruschke, Michael, *Signalling-On-the-fly: SigOfly*, International Conference on Intelligence in Next Generation Networks, 2015.

[6] *Reactivex: An API for asynchronous programming with observable streams*, http://reactivex.io/, last accessed February 20th, 2017.

[7] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta and A. Neal, *Mobile-Edge Computing Introductory Technical White Paper* ETSI, 2014

[8] *reTHINK Github Repositories*, https://github.com/reTHINK-project, last accessed February 20th, 2017.

[9] *VERT.X*, http://vertx.io/, last accessed February 20th, 2017.

[10] *Node.js*, https://nodejs.org/en/, last accessed February 20th, 2017.

[11] *Matrix.org*, https://matrix.org/, last accessed February 20th, 2017.

[12] *RFC 7118: The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)*, https://tools.ietf.org/html/rfc7118/, IETF, January, 2014.

[13] *Slack API*, https://api.slack.com/, last accessed April 28th, 2017.

[14] *On Distributed Communications*, http://www.rand.org/pubs/research_memoranda/RM3420.html, 1962, last accessed February 20th, 2017.

[15] *Decentralized Web Summit*, https://www.decentralizedweb.net/, last accessed February 20th, 2017.

[16] P. Agten, S. Acker, Y. Brondsema, P. Phung, L. Desmet, and F. Piessens, *JSand: Complete Client-side Sandboxing of Third-party JavaScript Without Browser Modifications.* In ACSAC, 2012.

[17] L. Meyerovich and B. Livshits, *ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser.* In IEEE Security and Privacy, 2010.

[18] S. Van Acker, P. De Ryck, L. Desmet, F. Piessens, and W. Joosen, *WebJail: Least-privilege Integration of Third-party Components in Web Mashups.* In ACSAC, 2011.

[19] *The PeerJS library*, http://peerjs.com/, last accessed April 28th, 2017.

[20] Jung Ha Paik and Dong Hoon Lee, *Scalable signaling protocol for Web real-time communication based on a distributed hash table* In Elsevier Computer Communications, 70, 2015