Exploring a social-aware WebRTC-based P2P architecture for live video streaming

Franco Tommasi
University of Salento
Dept. of Engineering of Innovation
Lecce, Italy
franco.tommasi@unisalento.it

Alessandro De Donno
University of Salento
Dept. of Engineering of Innovation
Lecce, Italy
alessandro.dedonno@unisalento.it

Fulvio Irno Consalvo
University of Salento
Dept. of Engineering of Innovation
Lecce, Italy
fulvio_ic@yahoo.it

ABSTRACT
In the present work a feasibility study of a novel peer-to-peer architecture for live video streaming is proposed. It leverages both recent web technologies for peer implementation and overlay management, and the use of online social networks as integral part of the architecture itself.

The key points of the architecture are: (1) the use of WebRTC to setup and manage a P2P overlay, and to deliver audio/video streams; (2) the implementation of P2P clients as HTML5/Javascript web applications, without involving other centralized entities acting as servers, according to a nearly serverless model; (3) the use of online social networks to retrieve user information and relationships between them, in order to improve overlay and stream management, and as a place to announce their setup and broadcasting, and to increase the audience; (4) the use of social networks to implement the WebRTC signaling.

In this paper, after a brief *excursus* of the involved technologies, the proposed architecture will be introduced, and a first prototype will be presented with the aim of demonstrating the feasibility of the project. The authors will also introduce the elements of the architecture on which future efforts for subsequent enhancements and extensions will be focused.

CCS Concepts
• Software and its engineering→Peer-to-peer architectures • Information systems→Web applications • Information systems→Web conferencing • Networks→Online social networks • Computer systems organization→Peer-to-peer architectures.

Keywords
Live video streaming; P2P overlay; WebRTC; social networks; web-based applications; serverless architectures.

1. INTRODUCTION

ICETC 2017, December 20–22, 2017, Barcelona, Spain

ACM ISBN 978-1-4503-5435-6/17/12…$15.00
https://doi.org/10.1145/3175536.3175548

In recent years online social networks have become very popular and they have been experiencing a relentless exponential growth [1]: currently there are a number of different platforms which allow their users to create communities around specific topics, sharing contents (text, images, videos, etc.) and establishing friendships. In addition, all major social networks provide Web APIs that give third-party developers access to their resources (user data, friendships, messaging systems, etc.), thus allowing the interoperability between social networks and external applications.

At the same time, the Web has become a full-fledged software platform thanks to new standards such as HTML5 and WebRTC. HTML5 [21] is the latest HTML standard published by the World Wide Web Consortium (W3C), and provides advanced APIs such as built-in audio/video support, caching and local storage capabilities for offline usage, 2D/3D graphics support via WebGL standards, Web Workers that allow spawned scripts execution in parallel to the main web page. Moreover in recent years Javascript has become more and more a general-purpose programming language, suitable for writing complex applications: as a matter of fact HTML5, Javascript and CSS3 [22] are promoting a paradigm shift [2] from traditional desktop applications towards web-based ones. WebRTC [23] standard adds real-time communication to the Web through a set of APIs and protocols that allow direct peer-to-peer connections between browsers and, consequently, between web applications [3], without external plugins.

The present work introduces a preliminary decentralized software architecture for large scale live video distribution, based on the aforementioned web technologies: it mainly consists of a web application, acting as a node of a P2P overlay, that interacts with both a social network service for user authentication, WebRTC signaling and overlay management, and a list of public STUN servers for NAT-traversal setup.

2. RELATED WORK
The process of WebRTC standardization by the W3C and the IETF [24] is going ahead (the latest *Draft 05* has been released in June 2017), and nowadays all major browsers support it natively, both desktop and mobile ones. As WebRTC introduces real-time communication between browsers, in recent years research has begun to explore its potential in structured and unstructured peer-to-peer architectures.

In [4], Rhinow *et al.* take into account the use of WebRTC (wrapped up in a modified version of PeerJS Javascript Library, [25]) to implement a pull-based protocol for live video streaming, where video content is provided by a *source publisher* and the overlay is maintained by a *WebRTC coordinator*. Peers of the network assist the *source publisher* in content distribution. However, when the paper was presented in 2014, WebRTC

suffered from limitations such as poor browser implementations and interoperability issues between browsers, because of its early stage in standardization. The authors then concluded that an implementation of such protocols for large-scale systems was not yet fully feasible.

Another WebRTC-based peer-to-peer protocol is *Cyclon.p2p* [5], based on *Cyclon* [6], a gossip-based peer sampling protocol that provides a stream of peer descriptors to its upper layers. The authors claim *Cyclon.p2p* should provide a P2P topology management for further implementation of web applications.

In [7], Zhang *et al.* introduced *Maygh*, an architecture that helps a traditional web site to serve its static content by building a content distribution network (CDN) through the browsers of the users who visit the web site. Even if *Maygh* isn't a P2P overlay and requires a set of centralized coordinators to work, it makes use of WebRTC for establishing peer-to-peer communication channels between browsers.

Tommasi *et al.* in [8] introduce *WebCHARMS*, a new version of their *CHARMS architecture* [9]-[13], for relaying real-time video streams. *CHARMS* is an hybrid system where entire streams are served, as in CDNs (whereas in P2P systems they are splitted in chunks), and at the same time its cooperative overlay topology is built in a P2P fashion. In *WebCHARMS* the authors move towards a WebRTC-based architecture by converting the clients from traditional desktop applications to web applications, and by redesigning the overlay management. The architecture is still not fully decentralized, as it requires at least an *overlay coordinator* for nodes management and arrangement.

Disterhoft and Graffi in [14] build a P2P framework for a decentralized secure social network. As in [4], they use the PeerJS library as a wrapper around WebRTC APIs.

In parallel, research has shown interest in the social networking phenomenon: on the one hand, people around the world are become familiar and interested in sharing their interests and establishing friendships each other; on the other hand, all major online social networks make available some of their users' information via-APIs. As a consequence, projects that tried to merge social networking logics and P2P networks emerged, with the purpose to improve P2P overlay management and to limit churn rates. In [15], Nguyen *et al.* introduce *Stir*, a peer-to-peer streaming architecture that also implements social logics between its users: in this way, peer connections between "friends" are preferred above "strangers" ones. In [16], Abboud *et al.* suggest to build their streaming system on top of online social networks, instead of adding social logics to an existing P2P network.

However, to the best of authors' knowledge, the present work is one of the first attempts to deeply integrate WebRTC technologies and online social network services to build a nearly-serverless architecture for live video streaming.

3. ARCHITECTURE OVERVIEW

The proposed architecture leverages both social network APIs and WebRTC ones to build P2P overlays for live video stream distribution. Peers are implemented through a web application running on user browsers, thus deploying a nearly serverless architecture. Taking into account a specific overlay, the main actors are:

- a bootstrap node that initializes a P2P overlay;
- one or more sources that provide live video streams;
- one or more consumers, i.e. peers interested to watch streams. They also act as relayers for the joined streams;
- a web server where to retrieve the web application.

Bootstrap node, sources and consumers share the same code, and then each node can act as bootstrap, source and consumer, depending on the user's choices.

Moreover, each node needs to interact with two external services to work properly:

- a social network that makes its resources and services available through RESTful WebAPIs;
- one or more public STUN/TURN servers, freely available on the Internet.

3.1 Application overview

The main components of the web application are shown in Fig. 1.

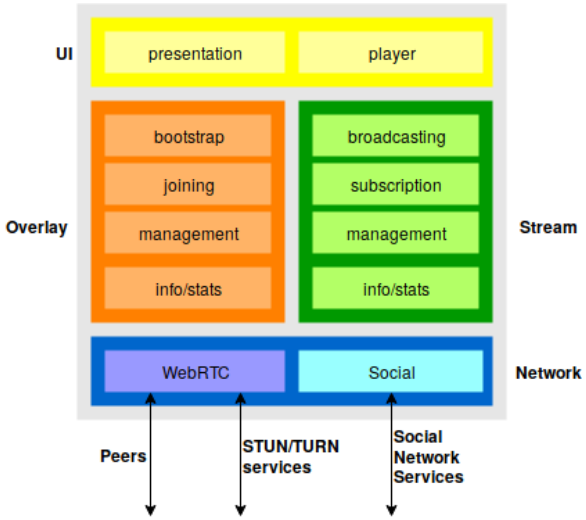

Figure 1. High-level design of the web application.

3.1.1 Overlay module

It manages overlay setup and management. Any node is able to initialize an overlay, as well as to join/leave an existing one. It interacts with the social sub-module to "announce" the newly created overlay, and makes use of WebRTC during the overlay lifecycle.

3.1.2 Stream module

This component is responsible for stream management and distribution tree via WebRTC APIs. It's involved when a source wants to broadcast its stream, and a consumer subscribes/unsubscribes to it. It announces stream availability to the social network through the social sub-module.

3.1.3 Network module

This layer consists of two main sub-modules, a wrapper around standard WebRTC APIs and a "social" sub-module. The latter manages the communication with the social network for user authentication, overlay and stream announcements, WebRTC signaling via private messages, user relationships.

3.1.4 User Interface module

It implements the web application's presentation logic.

3.2 High-level workflow

3.2.1 User authentication

Once the application is loaded into the user's browser, the user authentication is delegated to the OAuth2 social network service. The social network must obviously offer such service, as explained in the next section, where the architecture requirements and constraints are detailed. The user must authorize the application to access his account and then to retrieve his information. The main information required by the application are: (1) a user id that univocally identifies the user within the social network; (2) his friendships for further overlay optimization and distribution tree management.

3.2.2 Bootstrapping an overlay

As soon as a user is logged in, he can initialize an overlay (or he can join an existing one, as described in the next paragraph) through the application's UI. Under the hood, the overlay module is initialized, and the newly-created overlay is announced to the social network's wall. The public message contains at least an hardcoded *hashtag* that identifies the proposed architecture, and the overlay id. The user can also add a textual comment.

3.2.3 Joining an overlay

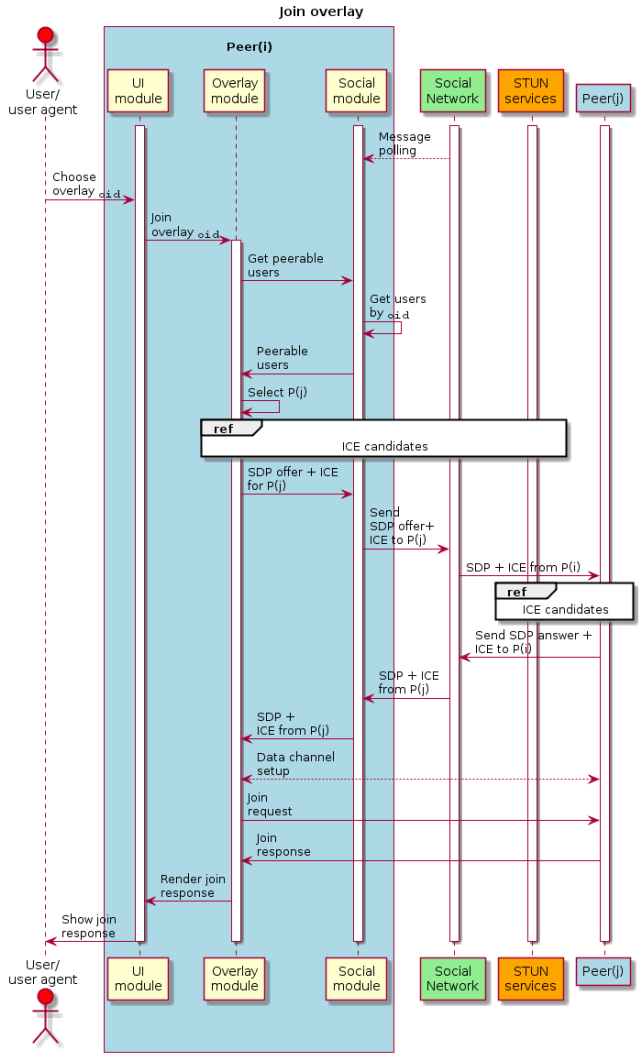

Figure 2. The "join overlay" procedure.

Immediately after user authentication, the application listens to social network for public messages containing both the architecture hashtag and an overlay id, so an overlay list is presented to the user (as well as a per-overlay list of available live streams). The user can choose an overlay to join from that list: under the hood, the application exploits the private messaging system of the social network to establish a WebRTC `RTCDataChannel` with one of the peers of the overlay (Fig. 2).

Currently the peer is randomly chosen, and the details of the operation are described in the next section. Once the overlay is joined, the application sends an announcement message to the public wall.

3.2.4 Broadcasting a live stream

Any user joining an overlay may decide to broadcast a live video stream. Through the WebRTC MediaStream API he gets access to his camera and microphone, and a message announcing the stream is sent to the overlay. The stream list of each peer is then updated. Additionally, the social sub-module announces the stream to the social network's public wall.

3.2.5 Subscribing a live stream

Each user can see an updated list of per-overlay available streams, even if he hasn't already joined any overlay, because new streams are announced to social network's public wall, other than via-overlay. When a user wants to subscribe to a stream, he just selects it from the stream list: the stream is then relayed by a subscriber according to the distribution tree, as explained in the next section. If the user hasn't joined the overlay hosting the stream, first the application automatically joins the overlay.

4. ARCHITECTURE REQUIREMENTS AND CONSTRAINTS

4.1 Social network

So far, reference has been made to a generic online social network service although, in fact, the proposed architecture is not bound to a specific one. However, for a social network to be used, some requirements must be met. Specifically, a social network must implement a set of WebAPIs that allow third-party applications to:

- delegate user authentication to the social network via-OAUTH2 [17]. Depending on the authorization grant type provided by the social network OAUTH-API, the architecture can even work without a web server (it is the case of the *implicit* grant type);
- send and receive public messages to/from the social network's public wall. They are needed for a peer to announce both a new overlay and a new stream, and to receive similar announces from other peers;
- send and receive private messages to/from other users. They act as duplex communication channel for WebRTC signaling;
- retrieve user's friendships. Currently the authors are working on an improved version of the architecture that will leverage friendships to optimize both the overlay management and the distribution tree.

As an example, two of the most popular social networks like Facebook and Twitter already meet these requirements, and in the early stages of the present work a first attempt to integrate Twitter REST APIs [26] and Streaming APIs [27] was successfully done; however, as explained in the next section, an internal social network simulator has been developed for testing purposes.

Future versions of the architecture could be able to integrate more than one social network, although interoperability issues must be furtherly explored.

4.2 Overlay

In the proposed architecture any user, once authenticated for the social network, can initialize an overlay. His web application acts as bootstrap node for incoming peers. A peer P_i that wants to join the overlay, selects it from the overlay list provided by the application's UI and updated by polling the social network's public messages containing the overlay announces sent by the bootstrap node and by the peers that already joined the overlay. When selecting the overlay through the UI, the application silently chooses a random peer of the overlay as its bootstrap node P_{boot} and starts the join procedure:

- P_i sends a `initJoinRequest` private message to P_{boot} containing its SDP-offer and ICE candidates;
- P_{boot} answers with `ainitJoinResponse` private message to P_i containing its SDP-answer and ICE candidates. P_{boot} acts as predecessor (P_{pred}) for P_i. The message also contains the social network user id of the peer that will act as successor (P_{succ}). If there is one only peer in the overlay, P_{pred} and P_{succ} are the same node;
- P_i sends a `endJoinRequest` private message to P_{succ} containing its SDP-offer and ICE candidates;
- P_{succ} answer with a`endJoinResponse` private message to P_i containing its SDP-answer and ICE candidates.

Once the overlay is joined, social network private messages are removed.

In this way P_i establishes two RTCDataChannels with its predecessor and successor and it can build its own routing table, depending on the underlying DHT algorithm: the authors are exploring the use of *Chord* [18] because of its low computation cost and bandwidth consumption [19]. As WebRTC data channels are bidirectional, an improved topology of Chord modeled as a bidirectional graph (*Bichord*, [20]) will be explored too. However, since the research present focus is on messaging and stream exchange and not on overlay topological structure, in the current feasibility study of the proposed architecture, the application prototype makes use of a ring-based topology and overlay messages are sent to both predecessor and successor nodes.

Three types of overlay messages are currently implemented: `joinOverlay` and `leaveOverlay`, which are broadcast messages to announce the join/leave of a peer, and a `toPeer` message for unicast communication, without an established RTCDataChannel between the involved peers.

Apart from the aforementioned overlay messages, the current prototype implementation manages a few stream-related messages:

- `newStream`, to announce the broadcasting of a new stream;
- `streamRelaying`, to announce a peer subscription to a stream and its availability to act as relayer for other peers;
- `endSlots`, sent to other peers when a node has no more bandwidth to relaunch a stream;
- `connStream`, to manage `RTCDataChannel`'s setup between a relayer and a subscriber, by carrying SDP messages and ICE candidates.

Once the feasibility of the architecture is verified, users' friendships provided by social networks will be leveraged to improve the distribution tree performances and to limit churning: e.g. connections among "friends" sharing common interests could be deemed more reliable then random ones.

5. "STREAMAPP", AN EXPERIMENTAL PROTOTYPE

The proposed architecture has been tested by developing *StreamApp*, an experimental web application written in HTML5/Javascript with the help of AngularJS [28], a well known client-side open source web framework mainly developed and maintained by Google, that follows the model-view-controller (MVC) and model-view-viewmodel (MVVM) principles. HTML5 provides two important features used in the prototype, among others: *Web Workers*, that allow the execution of Javascript code in parallel with the main page, and *IndexedDB*, an embedded transactional database, available inside the browser.

5.1 Social network simulator

Furthermore, a social network simulator has been implemented. Initially the prototype was linked to Twitter through its REST APIs, however the APIs present some limitations (as an example, currently the same API method can be invoked 15 times every 15 minutes at most): although they don't compromise nor alter the application workflow, they could be annoying during development and testing stages. The simulator is written in JavaScript and it runs on a Node.JS [29] web server. Data are stored on *MongoDB* [30], a *NoSQL* document-oriented database. The OAuth protocol has been also implemented, to fully simulate standard social network services.

5.2 Web Workers

Web Workers are used to communicate with social network, to find and post messages, to "polling" for incoming ones and to classify them (Fig. 3).

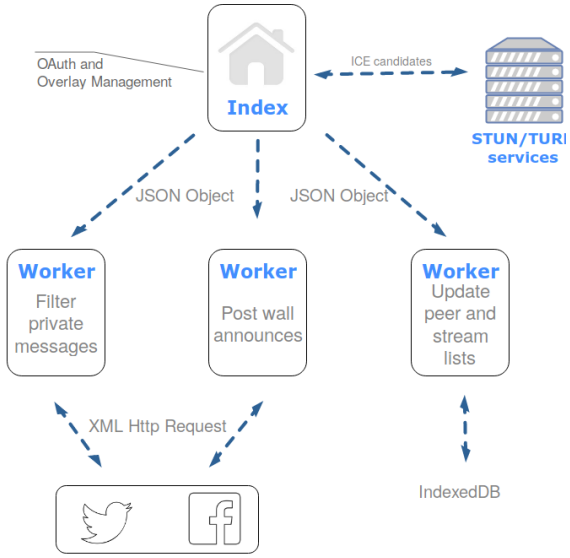

Figure 3. Web workers implemented in the application prototype.

Information about peers and available streams is locally saved via-IndexedDB, and then retrieved to prioritize and to choose the best peer for establishing a connection, or to subscribe a stream.

Web Workers present an important limitation that must be managed: they cannot directly manipulate the DOM elements and cannot use some default methods and properties of the window object. Additionally there is no access to non-threadsafe components of the DOM.

5.3 Messages

Messages coming from the social network wall can be overlay announcements or stream announcements. They are filtered by *hashtag* and have the following syntax:

```
#STREAMPP_id:{overlayId}_p:{peerCount}
```

for overlay announcements, and

```
#STREAMPP_id:{overlayId}_p:{peerCount} \
_s:{streamId}_t:{streamDesc}
```

for stream announcements.

Private messages have the following syntax:

```
TS:{timestamp}|MSG:{text}
```

They are used by peers to exchange SDP-messages and ICE candidates for establishing an `RTCDataChannel` and then to join an overlay. Once a peer joins an overlay, it uses the WebRTC data channel to communicate with others peers. For example, if it wants to subscribe a stream, the SDP and ICE candidates are sent to relayer node via-overlay: the social network is not involved.

6. CONCLUSION AND FUTURE WORK

In this paper the authors examine the feasibility of a WebRTC-based P2P architecture for live video streaming, that leverages and integrates social network services for overlay management. As it mainly consists of a web application acting as client of the P2P network, with no need of centralized servers but the web server running the application, the architecture is nearly serverless: depending on the authorization grant type provided by the chosen social network APIs, it can evolve in a fully serverless architecture.

The authors are currently focusing on improvements to overlay and stream management, starting with a reliable and scalable DHT algorithm, as well as an efficient distribution tree. Social user's information like relationships will be fully integrated in the overlay management.

7. ACKNOWLEDGMENTS

The authors would like to thank Catiuscia Melle for her advices during the early stages of the project, and Ivan Taurino for his technical support.

8. REFERENCES

[1] Julia Heidemann, Mathias Klier, and Florian Probst 2012. Online social networks: A survey of a global phenomenon. *Computer Networks*, 56(18):3866–3878.

[2] Antero Taivalsaari, Tommi Mikkonen, Matti Anttonen, and Arto Salminen 2011. The death of binary software: End user software moves to the web. In *Creating, Connecting and Collaborating through Computing (C5), 2011 Ninth International Conference on*, pages 17–23. IEEE.

[3] Christian Vogt, Max Jonas Werner, and Thomas C Schmidt 2013. Leveraging webrtc for p2p content distribution in web browsers. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–2. IEEE.

[4] Florian Rhinow, Pablo Porto Veloso, Carlos Puyelo, Stephen Barrett, and Eamonn O Nuallain 2014. P2p live video streaming in webrtc. In *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*, pages 1–6. IEEE.

[5] Nick Tindall and Aaron Harwood 2015. Peer-to-peer between browsers: cyclon protocol over webrtc. In *Peer-to-Peer Computing (P2P), 2015 IEEE International Conference on*, pages 1–5. IEEE.

[6] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen 2005. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217.

[7] Liang Zhang, Fangfei Zhou, Alan Mislove, and Ravi Sundaram 2013. Maygh: Building a cdn from client web browsers. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 281–294. ACM.

[8] Franco Tommasi, Catiuscia Melle, and Alessandro De Donno 2016. Towards a decentralized overlay for real-time live streaming: Charms meets the web. In *SAI Computing Conference (SAI), 2016*, pages 815–822. IEEE.

[9] Franco Tommasi, Simone Molendini, Elena Scialpi, and Catiuscia Melle 2010. Charms: Cooperative hybrid architecture for relaying multicast satellite streams to sites without a satellite receiver. In *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on*, pages 269–276. IEEE.

[10] Franco Tommasi and Catiuscia Melle 2011. Large-scale terrestrial relaying of satellite broadcasted real-time multimedia streams. *International Journal of Computer Networks & Communications*, 3(3).

[11] Franco Tommasi, Catiuscia Melle, and Valerio De Luca 2014. Opensatrelaying: a hybrid approach to real-time audio-video distribution over the internet. *Journal of Communications*, 9(3):248–261.

[12] Franco Tommasi, Valerio De Luca, and Catiuscia Melle 2014. Are p2p streaming systems ready for interactive e-learning? In *Education Technologies and Computers (ICETC), 2014 The International Conference on*, pages 49–54. IEEE.

[13] F. Tommasi, C. Melle, A. De Donno, and I. Taurino 2016. Charms for e-learning: A case study. In *INTED2016 Proceedings, 10th International Technology, Education and Development Conference*, pages 3960–3967. IATED, 7-9 March, 2016.

[14] Andreas Disterhoft and Kalman Graffi 2015. Protected chords in the web: secure p2p framework for decentralized online social networks. *In Peer-to-Peer Computing (P2P), 2015 IEEE International Conference on*, pages 1–5. IEEE.

[15] Anh Tuan Nguyen, Baochun Li, Michael Welzl, and Frank Eliassen 2011. Stir: Spontaneous social peer-to-peer streaming. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 816–821. IEEE.

[16] Osama Abboud, Thomas Zinner, Eduardo Lidanski, Konstantin Pussep, and Ralf Steinmetz 2010. Streamsocial: A p2p streaming system with social incentives. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, pages 1–2. IEEE.

[17] Dick Hardt 2012. The oauth 2.0 authorization framework.

[18] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan 2003. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32.

[19] Jinyang Li, Jeremy Stribling, Robert Morris, M Frans Kaashoek, and Thomer M Gil 2005. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 225–236. IEEE.

[20] Junjie Jiang, Ruoyu Pan, Changyong Liang, and Weinong Wang 2005. Bichord: An improved approach for lookup routing in chord. In *Advances in Databases and Information Systems*, pages 338–348. Springer.

[21] https://www.w3.org/TR/html5/.

[22] https://www.w3.org/Style/CSS/.

[23] https://webrtc.org.

[24] https://w3c.github.io/webrtc-pc/.

[25] http://peerjs.com/.

[26] https://dev.twitter.com/rest/public.

[27] https://dev.twitter.com/streaming/overview.

[28] https://angularjs.org/.

[29] https://nodejs.org/en/.

[30] https://www.mongodb.com/.