2014 IEEE 34th International Conference on Distributed Computing Systems Workshops

Let Our Browsers Socialize: Building User-centric Content Communities on WebRTC

Max Jonas Werner, Christian Vogt, Thomas C. Schmidt
Department of Computer Science
Hamburg University of Applied Sciences, Hamburg, Germany
{maxjonas.werner,christian.vogt}@haw-hamburg.de, t.schmidt@ieee.org

***Abstract*—The emerging Internet standards of WebRTC open up a new paradigm of direct browser interconnects. Users are thus enabled to build personal communities by simply loading Web pages, provided an appropriate software is at hand. In this paper, we present a software architecture that provides the core concepts and components for content-centric communities on a pure P2P basis. Starting from generic use cases, we develop an information-centric overlay that naturally supports user requirements. Our prototypical implementation and evaluation demonstrate the feasibility of this light-weight approach.**

***Keywords*—P2P networks; browser-based communities; information-centric networking; user mobility; content offloading;**

I. Introduction

The World Wide Web has largely shaped the advancement of the Internet and browsers have become the dominant user interface to networked applications. Over the last decade, use cases have transformed from the mere consumption of content to a platform for sharing content between users. People use the Web to exchange files, chat or collaborate on content creation, all this unified in the browser environment. The common technologies deployed are based on the traditional client/server paradigm which did not change since the beginning of the Web. In consequence, users eager to publish content on the Web still need to rely on infrastructure of their own or others.

With the creation of the Web Real-Time Communication (WebRTC) specifications – jointly defined by the World Wide Web Consortium (W3C, [1]) and the Internet Engineering Task Force (IETF, [2]) – a major paradigmatic change in Web technologies arrived. For the first time, WebRTC enables a browser to establish a direct connection to another browser without the need for additional software or services. This opens the Web platform for a wide range of user-centric use cases, given an appropriate architecture is designed and implemented that facilitates these visions.

In this paper, we introduce such an architecture built on top of WebRTC transports. Our concept and prototypic implementation leverages WebRTC for spanning P2P content networks between browsers. We specify a rich set of use cases and discuss the corresponding requirements. In our evaluation we show how this architecture naturally supports the various use cases, easily matches the requirements, and present selected application examples. We specify a URI-based naming scheme for content along with appropriate semantics. This URI scheme revolves around a unique user name identified by an identity provider. By adapting ICN naming techniques and leveraging the broad deployment of Web technology, our solution enables user-centric P2P interaction between Web users that does not rely on centrally controlled infrastructure.

Our contribution starts from the perspective of the publish/subscribe paradigm [3] and takes inspiration from the emerging ideas of Information-centric Networking (ICN) [4]. ICN decouples content from location and builds name resolution and replication into the network layer itself. Our Browser-based Open Publishing (BOPlish) approach, on the other hand, builds user-centric naming and content routing on a P2P layer that is maintained solely among browsers. We put a strong focus on concepting a naming scheme syntax and semantics.

In the remainder, we discuss ICN approaches and WebRTC (Section II), describe the use cases we identified that demand for a P2P user-centric Web architecture (Section III) and introduce BOPlish as a bootstrap architecture for developing P2P Web applications (Section IV). We evaluate our architecture in Section V and conclude with an outlook in Section VII.

II. Background and Related Work

A. Publish/Subscribe Networking

The Internet revolution started after the World Wide Web had introduced a uniform, simple architecture of separating content publication and provisioning from content retrieval. The decoupling of publishing information from its consumption in space and time is a core element of the rich publish/subscribe paradigm [3]. In recent years, (proprietary) Content Delivery Networks have shifted this server-centric approach to the network that mirrors one-to-many communication for which the initial Internet architecture has not been built [5].

The ideas of Information-centric Networking (ICN) [4] have taken up the well-established concept of in-network storage and replication towards end-user communities, while adding the core objective of an open future Internet design. The latter requires resolution of the three major challenges *naming, security,* and *routing* [6]. In ICN, the underlying network layer must be capable of directing a named data request to a location completely transparent to the requesting client, and it must provide an independent verification of the supplied content. As a result the location of data becomes irrelevant, making it simple to introduce caches distributed throughout the network. Many such architectures have been introduced, prominent examples being DONA [7] and NDN [8]. Independent solutions have been designed that differ in naming, security, and routing, but all show a high interdependency among these three [9].

1545-0678/14 $31.00 © 2014 IEEE
DOI 10.1109/ICDCSW.2014.35

1) Naming and Security Binding: Unlike in HTTP, ICN uses names that are independent of a location or server instance. Two competing approaches exist, hierarchical or flat (e.g., hashed) names (Figure 1).

```
ndn://alice/images/image.png
dona://134(...)0f6:dfe(...)164
```

Fig. 1. NDN hierarchical identifiers and flat identifiers in DONA

Hierarchical names have the benefit that they can be aggregated, provided name prefixes and content locations coincide. When routing on the names itself, it is preferable to reduce routing table sizes by aggregating names. NDN uses such hierarchical names. Aggregation could be performed at the ISP level (with ISPs assigning prefixes to their customers), but this reintroduces a binding to location. The existence of the location-identity binding is the main argument for flat names (as used in DONA), which allow for a complete decoupling of location and identity but cannot easily be aggregated. Coping with a huge amount of unaggregatable identifiers requires either huge routing tables or external infrastructure. Finding a scheme that allows for both, effective aggregation and location-independence of the system without bloating routing tables is still subject to research activities [9].

Another aspect of the debate between flat and hierarchical names is the decision between human-readableness and cryptographic expressions for self-certification. While DONA can use a cryptographic hash of the content as its identifier and thus offers implicit content certification, NDN requires an external trust mechanism as described in [10].

2) Name Resolution and Data Routing: Each content request in ICN should be directed to a nearby surrogate in the network. When a location of the content is found, it has to be transferred to the requester. Therefore, data routing is used to find a path over which the actual content is transferred. Depending on the ICN implementation, routing is performed directly on names (e.g., NDN) or decoupled by some resolution service (e.g., DONA). In a coupled approach, the data forwarding follows the reverse path identified from the name resolution. In a decoupled approach, data is forwarded independently of content routing paths.

Coupling the data routing means to either a) store routing states in the intermediate hops traveled by the name resolution query or b) integrate this information into the content query packets on the way. Decoupled approaches allow for more flexibility, as control and data flows can be separated. On an Internet scale, both approaches must be seen as a severe challenge [6]. Our community size system does not face severe scalability issues and is built on a decoupled name resolution concept.

B. User-centric Naming and Networking

Our concept of user-centric content networks revolves around the idea that every participant in a specific P2P browser network is able to name and publish content. All of the (static or dynamic) content a user wishes to publish is assigned a URI that is derived from the user's unique name.

In [11], Allman describes the concept of a "personal namespace". The author lays out several problems with current naming systems such as DNS and URLs: Names are location-bound as is the case with URLs, where the hostname is resolved to a specific location on the network. Additionally, e.g. domain names are mentioned as ambigious so that users do not actually know by the domain name who the owner of the domain might actually be. The author distinguishes three different parties that are involved in creating and accessing a name for a content item: the consumer, the content provider (e.g. a user who shares a file) as well as the service provider (e.g. Flickr or Facebook). Typical current naming systems derive names from parts under the service providers' control so that migration from content to another provider will lead to a name change.

The "pnames" system proposed in [11] acts as an indirection between personal names assigned by a specific user and actual names like URLs or host aliases.This enables users to reference e.g. Bob's e-mail address as `Bob:mail`. For sharing such pnames the author proposes the usage of a DHT to resolve the flat pname identifiers.

In a follow-up to "pnames" the authors provide the outline and a prototypical implementation of a more abstract idea that is based on the concept of storing and referencing arbitrary content's meta data [12]. That system is called Meta-Information Storage System (MISS). MISS is meant to be operated on a global scale at ISP level. All MISS servers are interconnected in a global DHT that is used to find the MISS server that holds a specific information item. The authors thus introduce a lookup layer for retrieving meta-information of content.

A high-level description of user-centric networking is presented in [13]. The authors start with the idea that each user in a specific interest group offers a set of services to the group. For interconnecting users the authors propose to leverage existing social networks such as Twitter or Facebook to retrieve unique user identifiers. This way it is possible to leverage existing relationships between persons. A tuple of `(user name, service name)` is proposed to address services offered by a specific user. This makes it possible to decouple the service name from the host that offers the service while at the same time coupling the service with the user offering it (e.g. to ensure authenticity).

The IETF is currently engineering an Internet Draft for a user-centric SIP (Session Initiation Protocol) approach [14] that is based on RELOAD specified in [15]. RELOAD defines a powerful framework for P2P storage and messaging, including a security model, NAT traversal and a pluggable topology mechanism (with a Chord variant as default topology plug-in). RELOAD is designed so that specific overlay applications are to be implemented on top of a RELOAD network. One such application is the SIP usage specified in [14]. This usage employs RELOAD to establish SIP sessions via the P2P overlay and defines a naming scheme, eventually defining a completely user-centric distributed telephony service. Users store a mapping from their AOR (e.g. `alice@dht.example.org`) to their node ID in the P2P network. This mapping is then used by others to retrieve the node to connect to for a specific AOR.

C. WebRTC

WebRTC is a protocol suite that allows two Web browsers to communicate directly over a UDP-channel [2], paired with a JavaScript API for Web applications [1]. WebRTC allows for transferring a/v data via the Secure Real-time Transport Protocol (SRTP) as well as generic binary and textual data via the Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS). Because most browsers are expected to operate behind a NAT, Session Traversal Utilities for NAT (STUN) are natively provided. WebRTC is limited in the way that it allows two browsers to interconnect and exchange data. The standards neither include topology- nor routing-related topics.

Current research based on the WebRTC technology is mostly conducted in the multimedia conferencing and CDN context. The authors of [16] present a media server component that exploits the expected broad deployment of WebRTC to converge multimedia conferencing on different devices like smartphones and desktops. They provide an architecture based on open source software that handles media mixing, transcoding and filtering for group communication use cases.

Maygh [17] is a WebRTC-based system that facilitates P2P content distribution among participating clients. Maygh uses a centralized P2P lookup system with a coordinator node to store mappings between content and clients that already downloaded specific content. Succeeding requests from other clients can than be answered by peers that already downloaded the content with the help of the coordinator.

We build our architecture for application networking on the emerging WebRTC standards, which are under active implementation in several browsers.

III. The Case for Browser-based Publishing

We now focus on the basic conditions for our design. First we identify use cases that are inspired from the current Web or explicitly outlined for ICN (e.g. in [18]). Second we derive the corresponding requirements.

A. Application Use Cases

1) Document Sharing And Search: Current file sharing applications can be divided into two main groups: Server-based and P2P-based. On the Web, file sharing is implemented using the server-based approach. The drawback here is the reliance on a centralized service or the requirement to setup a custom server. Moreover, users have to trust the service provider with regards to content integrity and privacy concerns. A user-centric approach would counter these disadvantages in the following ways: Users share documents directly from one browser to another. The publication of a document does not rely on setting up a Web server, uploading the document to a central instance or changing DNS entries. Similarly, it shall be possible for users to search for content on other peers.

To share a document, a user registers for an account, uploads a specific document from the filesystem to 'the Web' and grants either public access or to a group of collaborators. Typically, this is done by sharing an identifier via an external channel or by using a front end to invite other registered platform members.

TABLE I. Requirements derived from Use Cases

Use Case	Requirement
Document Sharing And Search	Unique Identity, Unique Identifier
Conversational Apps	Unique Identity
Group Collaboration and Gaming	Unique Identity
Mobility and Offloading	Unique Identity, Transparent Handover, Replicas, Location-independent Identifier
Replication and Synchronization	Unique Identity, Multi-presence, Replicas, Location-independent Identifier
Privacy	Unique Identity

2) Conversational Apps: Real-time text or audio/video chats are of growing popularity. The increasing usage of social collaboration tools in the private as well as the enterprise context serve as a ground for conversational apps. On the Web, users employ centralized applications provided by service operators.

In a group chat context, all users of a community need to call and establish sessions. A single user must be able to open a chat room and invite other users (in some way connected to this user, e.g. via a friendship relation). The network transport must provide appropriate real-time services.

3) Group Collaboration and Gaming: In group collaboration applications and multiplayer games users collaborate in self-defined groups. Such applications allow for the creation of groups, varying memberships and the invitation of participants. In addition, the application state (e.g., the position of players in a game or the content of documents in a groupware) must be transparently accessible for all members of a group.

B. System Use Cases

1) Mobility and Offloading: More and more people use Internet services from mobile devices like smartphones or tablets. Under mobility, the user network must be able to cope with frequent network address changes. Publishers can accomplish this goal by offloading content to (stationary) third parties that promise better connectivity. Consumers can partially mitigate rapid address changes by pre-fetching content.

2) Replication and Synchronization: Users generally demand high content availability and fast access. Replication accomplishes this by storing multiple copies of the content (replicas) on different, independent systems. Synchronization implies that the content is being kept up-to-date between the replicas at a reasonable time-scale and logic.

3) Privacy: There are increasing demands for privacy by Internet users. The benefits of these applications are that they only use encrypted transports and encrypt the data that is stored at third parties. This makes it more unlikely that eavesdroppers with access to transport routes or community infrastructure are able to reveal sensitive data.

C. Requirements

We now can derive a set of requirements that a user-centric solution should fulfill.

1) Unique Identity: The first one is that every user of a community must be uniquely identifiable so that others are able to verify the identity of the remote peer.

2) Multi-presence: As a second requirement, it should be possible for a user to stay online in the community with several clients at the same time (multiple presence), so that content can be served from different hosts belonging to one user.

3) Unique Identifier: The third requirement is that content must be uniquely identifiable in the sense that a document must get a unique, persistent handle to be shared between users. Such an identifier must be uniformly constructed and generic enough to support the use cases described here as well as future usages.

4) Location-independent Identifier: Fourth, the identifier must be constructed in a way that is not tied to a specific host. It should rather be host-independent to be able to shift content between hosts and decouple the content names from the underlying infrastructure as introduced by ICN. Content can then be published independently of the actual peer serving it, thus enabling flexible offloading approaches.

5) Transparent Handover: Fifth, a solution is required to take into account peers changing networks quickly. Web applications need to implement transparent handovers. Since the identifiers are host-independent, the host resolution must be flexible enough to support quick updates to host locations.

6) Replicas: The final and sixth requirements shall enable replicas. Content may be shared among peers but must be available through one identifier, which shall be resolvable to more than one host address.

IV. Browser-based Open Publishing

The solution concept and the software architecture proposed here are designed to comply with all the requirements stated above. The primary assumption is that an implementation of this concept runs in a Web browser without the need for additional plugins. Our approach is called BOPlish, short for Browser-based Open Publishing.

A. Solution Concept

The solution presented here focuses on the idea that users have complete control over naming and providing content and eventually introduces a user-centric naming and networking concept, similar to those presented in [11], [12] or [13].

A user community consists of a number of peers that are connected to each other via a P2P network. A Web server delivering a BOPlish application serves as bootstrap component for joining one specific community. A user can join the P2P network and may close the connection to the Web server without losing any functionality provided by BOPlish. Prior to joining a user community a peer has to acquire a unique peer ID (its address) and the user has to authenticate at an identity provider. The combination of peer address and username is then used to join the user community and stored in a Distributed Hash Table (DHT) with the username as key and a reference to this peer as value.

When the user community is established, content can be published, accessed and shared among the peers. For this purpose, the design of content identifiers is a key ingredient to our solution. We start from URIs, the common meta-scheme for Web resources. For the further specification, we follow three steps. First, we build on the recent Common API for (multicast) publish/subscribe [19]. RFC 7046 provides a standard syntax for an identifier of the form `id@instantiation` along with security credentials. Second, we center IDs around users that are 'instantiated' by identity providers. Third and last, we add the name of the application-layer protocol (instead of ports) to facilitate a transparent communication context.

In summary, our proposal for a uniform content naming reads:

```
bop:username@idp:protocol
                    [/path[?parameters]]
```

These content URIs are comprised of the scheme `bop` and a hierarchical component further built from a unique `username` verified by an identity provider `idp`, followed by a `protocol` and `path` specifier and optional `parameters` that can include security credentials. The protocol specifier is used for setting different usages in one community, e.g., a chat service and a document sharing service. A peer uses that identifier to pass the URI to different modules of the application. This puts part of the application-specific semantics into the URI, with the consequence that not every BOPlish application may be able to serve every URI. The advantage of this design is that BOPlish URIs are flexible and extensible enough to easily reflect future use cases. Such a URI is generated for every published item and is shared to other users. The sharing itself is done as with HTTP URLs, e.g., via XMPP or e-mail. These are some examples of BOPlish URIs:

```
bop:alice@example.org:document/img/
    images.png?sha-256;1234abc...

bop:bob@example.com:search/Music/*tomte*
```

BOPlish URIs guarantee a location-independence by employing the username instead of a specific host identifier. The actual address of a peer responsible for a specific user is resolved via the user community itself (using the DHT). A query for one key in the DHT may result in a list of peer addresses, reflecting the currently available content publishers.

Our reference implementation of this concept consists of a JavaScript library that can be included in web applications either by running directly in the browser or potentially on a server using a JavaScript runtime environment like Node.js[1]. A user navigates to a web page and automatically joins the user community. After the user has joined the overlay network, he can request content or publish content himself. This overlay could even span across web sites so that a user that joined from `example.org` can communicate with a user from `example.com`. This allows for a decentralized, domain independent content distribution which is not tied to central services. BOPlish uses WebRTC as its transport mechanism, allowing for direct peer-to-peer connections between the clients' browsers.

Figure 2 shows an overview of the current BOPlish architecture first described in [20], [21]. It consists of three components: a) The Name Resolver API that allows for the resolution of location-independent identifiers; b) The Content API that is queried by a remote host to access the content

[1] http://js-platform.github.io/node-webrtc/

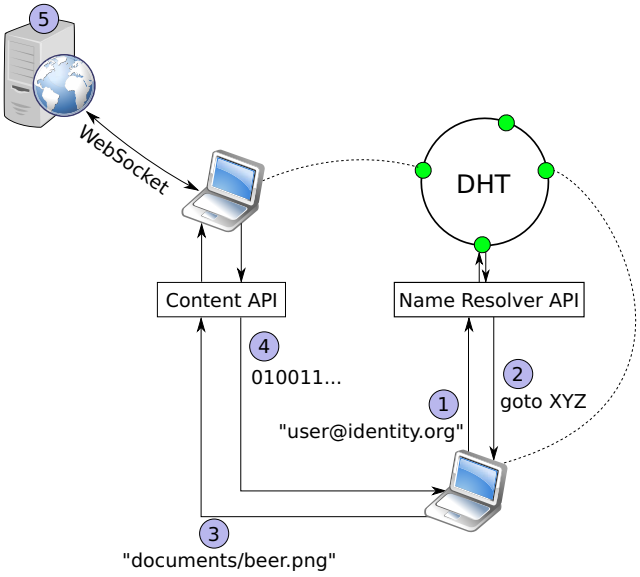

Fig. 2. Example for the steps involved in retrieving the content addressed by the URI `bop:user@identity.org:documents/beer.png`. Nodes in BOPlish retrieve content by issuing a lookup of the content's user namespace to the underlying DHT (1) which returns a pointer to the actual node that holds the content (2). This pointer is then used to open a WebRTC Data Channel to the peer, query for the content (3) and transfer it (4). At least one node maintains a WebSocket connection to the bootstrap server (5) so that new peers can join the community via this node.

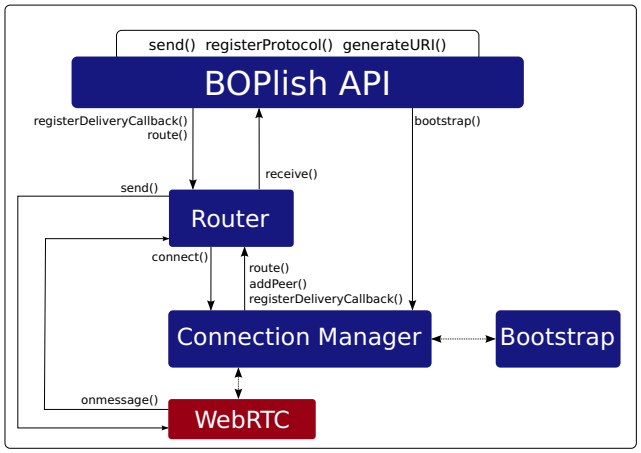

Fig. 3. Outline of all BOPlish components and their interaction with one another. Each component is strictly defined by a set of API calls and loosely coupled to the other components. This enables us to easily replace parts of the functionality without having to refactor reliant code.

announced by the publisher via its URI; c) A bootstrap component for joining an existing network.

B. Software Architecture

The design of our architecture is presented in Figure 3. At the very top sits the BOPlish API which is the entry point for all applications that make use of the content sharing facility. This is the developer facing part that exposes an API for sending and receiving data. Below that part, we encapsulated a Router, a Connection Manager and a Bootstrapping mechanism.

The Router component is responsible for deciding where to forward messages to and thus maintains a routing table, eventually forming a peer in the DHT. The Connection Manager component is responsible for handling WebRTC specifics like creating offers and answers, keeping track of open connections and handling glare. To be able to join a P2P network, a node has to know at least one other node already part of that network. This is where the Bootstrap component comes into play. It encapsulates the functionality for discovering an initial node to connect to. Since this process is very tightly bound to the generic connection establishment in our WebRTC-based implementation, we included this component into the Connection Manager.

C. Name Resolution and Data Routing in BOPlish

Instead of operating on the network layer like ICN, we introduce an overlay network which is capable of unbinding the relation between location and content identifier. This mechanism is realized by using a DHT which uses a hash of the identifiers as key and returns a pointer to the node that holds the content as value. This indirection allows the system to handle names and locations separately which we identified as a requirement for a content-centric architecture above. Because only the name-location resolution depends on a DHT and we limit the scale to specific interest groups instead of the Internet as a whole, the DHT can be designed to be highly churn-resistant and redundant. This is a crucial requirement as DHT implementations tend to be fragile when peers join/leave the network in a high frequency [22].

Data Routing in the BOPlish architecture is decoupled from the name resolution. Instead of using the reverse path of the name resolution, BOPlish opens a direct WebRTC connection between the content receiver and one or more of the publishers. Coupling the data routing with the name resolution is also possible but routing the content through the DHT would impose unnecessary load, leading to poor performance regarding the name lookup. Moreover, depending on the DHT implementation, the overlay path can be disadvantageous because it is not aware of geographical and performance properties of the overlay hops. The reference to a location is obtained by using the return value of the DHT name resolution procedure. If the connection to the publisher fails, the content receiver can always re-query the DHT to find the updated location information. This allows for mobility of both, the content receiver and the publisher because the DHT entry can easily be updated without requiring a name change of the content's identifier.

D. Envisioning Pub/Sub in BOPlish

BOPlish is a user-centric approach to content distribution in user communities. In such a system, peers need to be loosely coupled to gain stability and performance despite unreliable entities. As the recent success of ICN indicate, the pub/sub paradigm is a good fit for such large-scale applications and a pub/sub interface would be a valuable addition to the current

BOPlish implementation. To identify the characteristics of pub/sub systems, [3] factors out a common denominator: the decoupling of the communicating entities in time and space. In this section, we show how BOPlish can be extended to fulfill the requirements introduced by [3].

TABLE II. EXTENDED NAME RESOLUTION DATA STRUCTURE

Key	Value	
	Publishers	Subscribers
$h(bob@idp)$	$[CurrentPublishers]$	$[(CurrentSubscribers, RequestedURI)]$

Table II shows an extended data structure that aims to provide the necessary decoupling. This is achieved by (a) adding a subscriber reference to the hash table's value. This allows a subscriber to issue an interest in content even when no publisher is available at that time and therefore allows for asynchronous notification mechanisms. Alongside the subscriber's reference, the requested URI is also kept in the DHT (b). In this way, publishers can serve certain subscribers depending on the requested URI (e.g., chat from pub1, documents from pub2). After extending the data structure, we can revisit the requirements of a pub/sub system:

1) Spatial decoupling: Spatial decoupling in BOPlish is feasible by letting the name resolution service select the appropriate publisher for a requesting subscriber. As such, producers and consumers both have a limited view upon the current state instead of requiring full knowledge of each other.

2) Temporal decoupling: As the name resolution is realized as a DHT and therefore distributed among the participating peers, it can be viewed as a persistent storage entity. Even when no publisher is available, subscribers can issue an interest in the content by adding themselves to the appropriate DHT entry. Conversely, the subscriber can get notified by the name resolution service when a publisher is available.

Extending the data scheme that is used by the name resolver results in a flexible approach that fulfills the requirements for a pub/sub system introduced by [3]. On the other hand, this approach burdens load on the DHT because updates to the hash table's entries have to be made for each participating subscriber.

V. EVALUATION

We evaluate the BOPlish architecture by showing its theoretical applicability on the use cases defined in III-A and III-B. We developed demo applications[2] on top of our BOPlish implementation to evaluate the practical consequences of our design decisions. Finally, the security aspects are discussed.

A. Applicability On Use Cases

Every application that runs on top of a BOPlish platform instance registers one or more protocols it claims responsibility for; this is the protocol specifier of the URI (see IV-A). The BOPlish API offers the method `registerProtocol()` to achieve this. After registration, messages for this specific protocol identifier can be sent (`send()`) and asynchronously received by the application running on top of BOPlish. Figure 4 and 5 show examples of such applications [20].

[2]https://github.com/boplish/

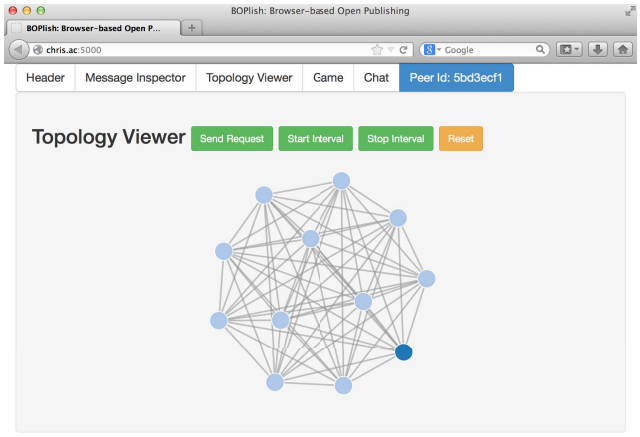

Fig. 4. Application running on top of the BOPlish architecture exhibiting the network topology where each vertex represents a peer in the community.

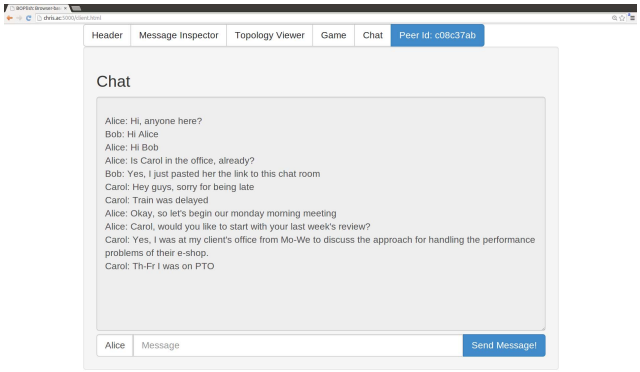

Fig. 5. Simple BOPlish group chat application.

1) Document Sharing and Search: Since JavaScript code running in the browser does not have access to the file system, a document sharing application is responsible for giving users the possibility to provide access to a specific file. This can be done by drag'n'drop mechanisms or with a file selection HTML element. Once the application has access to the file it can read its contents and meta data and store it in memory or a more persistent space like the browser's local storage.

After registering the protocol-identifier (e.g., `documents`), a unique URI is generated for every document. This URI is then shared by the end user in the community. Searching for documents is implemented using a wildcard syntax in the identifiers. The application can implement the search function locally and returns matches (e.g., a JSON list of `documents` URIs).

2) Conversational Apps: The use case of enabling real-time chats between two or more users in a community is achieved by registering for an appropriate protocol specifier like `chat` and generating a URI for a specific room:

```
bop:alice@example.org:chat/nightOut
```

This URI denotes a chat room "nightOut" with the host of the room being the user "alice@example.org". When Bob wants to join that room, his client resolves Alice's peer ID via

the name resolver, connects to it and receives a list of peer IDs that already joined the room. For every ID, Bob's application generates a WebRTC offer, sends it via Alice's peer to the specified host and receives an answer, eventually resulting in a WebRTC connection to every peer in the chat room.

3) Group Collaboration and Gaming: Real-time collaboration and multiplayer games in small scale can be implemented by directly connecting all participating peers. Extending the name resolver mechanism as described in Section IV-D benefits group communication claims by adding asynchronous notifications (e.g., when a player gets available). The SCTP transport of the DataChannel allows for high-bandwidth and low-latency connections.

4) Mobility And Offloading: Mobility implies rapid changes in network connectivity (e.g., a user traveling by train). To mitigate frequent address changes in BOPlish, the name-resolution mechanism must be fast enough so that the key-value pair of (username, peer ID) remains current. This is done by updating the corresponding DHT value, which must be implemented accordingly.

Moving content between peers (offloading) without changing its URI is possible when peers with two different IDs but the same user identity can be online at the same time. This implies that the DHT stores a one-to-many mapping between username and peer IDs. Thus, querying the DHT for a key `alice@example.org` may result in a list of IDs. Each BOPlish client must be able to query each host in that list for the desired content (given by its URI) until either the content has been retrieved or every host has denoted that it does not hold that specific content.

5) Replication and Synchronization: When a peer closes the browser or the tab that runs a BOPlish application, the WebRTC connection to the user community is lost. As a result, the shared content from this peer is not available anymore. To keep content highly available, a user may want to run a stationary device (such as a home NAS server) that's always connected to the user community.

To implement such a client one major requirement must be fulfilled: The client must have access to the filesystem. This is possible when the application runs on a Node.js server because Node.js provides ways to access the filesystem directly. Such an application could provide an administrative interface listing all the available files and corresponding BOPlish URIs that the user may then share to her fellows in the community. In our current solution concept, every application is responsible for synchronizing replicas because the core BOPlish library does not provide such mechanisms.

6) Privacy: Every peer has complete control over how content is shared in the community. Peers are able to encrypt content prior to delivering it to other peers and restrict access to certain remote users. BOPlish enables this use case and leaves it up to the application developers to protect content access; something, that centralized web services usually do not offer.

B. Security

Every peer in a structured P2P network has to acquire a unique ID. In the BOPlish name resolver implementation, this ID is used to identify the publisher and to determine the storage range. The assignment process needs to be secure in the sense that no user can (intentionally or unintentionally) be assigned to an ID that already refers to another user. Likewise, transport security is required, but WebRTC offers all functionality needed.

Approaches to secure ID assignment and identity verification in structured P2P networks have been previously proposed, e.g., in RELOAD [15]. This concept could also be applied to BOPlish. A current shortcoming of browsers are missing functions in public crypto, e.g., signature verification or encryption using an asymmetric key pair. The W3C is working on an API to make these available to Web applications [23]. In the meantime, the missing functionality can be included by incorporating JavaScript implementations of cryptography functions into BOPlish. However, it has to be noted that these cannot provide the same security as a native API because browsers lack access to e.g., a cryptographically secure random number generator.

VI. Discussion

BOPlish is a user-centric content sharing facility. The vision of our architecture differs from today's use of the Internet. Instead of handing over the content to the service provider (or the CDN provider), users keep control over the content shared within a BOPlish interest group. This is possible by employing a self-organized infrastructure centered around the user instead of using third-party services. Resources are accessible via a user prefix that is not bound to a provider. This prefix is resolved to the content location using an domain-independent overlay name resolution. The resulting overlay has to ensure reliability and availability characteristics comparable to current services.

Reliability in a BOPlish user community is largely influenced by the P2P algorithm in use. BOPlish currently uses the Chord protocol to maintain the network topology. DHTs tend to be fragile in regards to performance and have to be carefully designed. Finding beneficial parameters or even exchanging the algorithm with, e.g., a mesh-based solution are possible encounters to handle frequent joins/leaves of peers and therefore boost the reliability of the system to a sufficient level.

Content in a BOPlish user community is available as long as the content owner is reachable. As such, the user (instead of the service provider) has to make sure that a copy of the content stays available in the BOPlish network. As the BOPlish library is not limited to the browser environment, it could potentially also be run on NAS-like devices or servers that provide the necessary availability for a specific kind of content. Moreover, a service running on top of BOPlish could cache content at other peers for increased performance or offload content to increase availability. The ICN-inspired location-independent identifiers used by BOPlish to address content allow for flexible cache placement by extending the name resolution service.

Traditional services on the Web relay content over their own infrastructure. This allows the service provider to access and monitor all bypassing content. In contrast, a BOPlish user community does not comprise any central authorities and content is always transferred directly from one peer to another

over a secure DTLS-encrypted channel. Central monitoring is therefore much more complex to implement. As content is stored at peers that are under control of the content owner, the owner can at all times withdraw the content from the network by deleting the mapping in the name resolution service.

We evaluated the theoretical applicability on the defined use cases in section V. However, quantitative results from measurements are currently not conducted. Possible shortcomings like performance problems of the name resolution service might thus require changes to the architecture. Introducing caches could benefit performance and, equally important, reliability throughout the system. On the other hand, caching data at foreign users also means handing over data to peers that are not directly controlled by the content owner.

VII. Conclusions and Outlook

We presented BOPlish, our user-centric architecture that enables name-based publishing with the help of a browser-to-browser overlay network. Inspired by ICN ideas (but without changes to the network-layer) BOPlish facilitates various name-based content sharing applications. An easy implementation of various key use cases allows to distribute content among users of multiple websites in various application contexts. An evaluation and an open implementation of the network core along with some demo applications complement our idea.

Still, open issues remain that have not been dealt with. An appropriate DHT implementation is an important issue for the name resolution performance and therefore the whole system. Scalability of the system is also largely influenced by the used P2P topology. Thus, measurements of a larger scale, by using an emulation or simulation component, need to be conducted for detecting possible shortcomings of the BOPlish architecture. Also, security aspects have to be investigated further.

Since BOPlish focuses on Web browser clients we expect high churn rates in the communities. We plan on conducting the adaptability of our prototypical implementation to different levels of churn depending on the communities' size, delays between clients and rates of joins/leaves.

To take our approach one step further we constantly evaluate additional use cases. Such usages could include efficient multicast applications as well as the design of a simple to use API for developers that would like to make use of BOPlish communities.

References

[1] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers," http://www.w3.org/TR/2013/WD-webrtc-20130910/, World Wide Web Consortium, W3C Working Draft, 2013.

[2] H. Alvestrand, "Overview: Real Time Protocols for Brower-based Applications," IETF, Internet-Draft – work in progress 09, February 2014.

[3] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, June 2003.

[4] G. Xylomenos, C. Ververidis, V. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. Katsaros, and G. Polyzos, "A Survey of Information-Centric Networking Research," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–26, 2013.

[5] M. Handley, "Why the Internet Only Just Works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, Jul. 2006.

[6] D. Kutscher, S. Eum, K. Pentikousis, I. Psaras, D. Corujo, D. Saucez, T. C. Schmidt, and M. Wählisch, "ICN Research Challenges," IRTF, IRTF Internet Draft – work in progress 02, February 2014.

[7] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A Data-Oriented (and beyond) Network Architecture," *SIGCOMM Computer Communications Review*, vol. 37, no. 4, pp. 181–192, 2007.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, and M. F. Plass, "Networking Named Content," in *Proc. of the 5th Int. Conf. on emerging Networking EXperiments and Technologies (ACM CoNEXT'09)*. New York, NY, USA: ACM, Dec. 2009, pp. 1–12.

[9] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in Content-oriented Architectures," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ser. ICN '11. New York, NY, USA: ACM, 2011, pp. 1–6.

[10] D. Smetters and V. Jacobson, "Securing network content," PARC, Tech. Rep., Oct. 2009.

[11] M. Allman, "Personal Namespaces," in *Proc. of the 6th ACM Workshop on Hot Topics in Networks (HotNets-VI)*. New York, NY, USA: ACM, 2007.

[12] T. Callahan, M. Allman, M. Rabinovich, and O. Bell, "On Grappling with Meta-information in the Internet," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 5, pp. 13–23, Oct. 2011.

[13] R. Marques and A. Zuquete, "User-Centric, Private Networks of Services," in *2013 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, vol. 01, June 2013, pp. 1–5.

[14] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, H. Schulzrinne, and T. Schmidt, "A SIP Usage for RELOAD," IETF, Internet-Draft – work in progress 12, January 2014.

[15] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol," IETF, RFC 6940, January 2014.

[16] L. Lopez Fernandez, M. Paris Diaz, R. Benitez Mejias, F. Lopez, and J. Santos, "Kurento: a media server technology for convergent WWW/mobile real-time multimedia communications supporting WebRTC," in *Proc. of 14th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'13)*, June 2013, pp. 1–6.

[17] L. Zhang, F. Zhou, A. Mislove, and R. Sundaram, "Maygh: Building a CDN from Client Web Browsers," in *Proc. of 8th ACM European Conference on Computer Systems (EuroSys'13)*. New York, NY, USA: ACM, 2013, pp. 281–294.

[18] K. Pentikousis, B. Ohlman, D. Corujo, G. Boggia, G. Tyson, E. Davies, A. Molinaro, and S. Eum, "Information-centric Networking: Baseline Scenarios," IETF, Internet-Draft – work in progress 02, March 2014.

[19] M. Wählisch, T. C. Schmidt, and S. Venaas, "A Common API for Transparent Hybrid Multicast," RFC Editor, RFC 7046, December 2013.

[20] C. Vogt, M. J. Werner, and T. C. Schmidt, "Leveraging WebRTC for P2P Content Distribution in Web Browsers," in *21st IEEE Intern. Conf. on Network Protocols (ICNP 2013), Demo Session*. Piscataway, NJ, USA: IEEEPress, Oct. 2013, ICNP Best Demo Award.

[21] ——, "Content-centric User Networks: WebRTC as a Path to Name-based Publishing," in *21st IEEE Intern. Conf. on Network Protocols (ICNP 2013), PhD Forum*. Piscataway, NJ, USA: IEEEPress, Oct. 2013.

[22] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[23] D. Dahl and R. Sleevi, "Web Cryptography API," http://www.w3.org/TR/2013/WD-WebCryptoAPI-20130625/, World