Usages for the BOPlish Content Sharing Facility

Max Jonas Werner
Department of Computer Science
Hamburg University of Applied Sciences
Hamburg, Germany
maxjonas.werner@haw-hamburg.de

ABSTRACT

The Browser-based Open Publishing content sharing facility (BOPlish) enables users to publish content directly from within their Web browsers, leveraging a new set of technologies called Web Real-time Communication (WebRTC). BOPlish thus encompasses the vision of a server-less Web where users are not required to setup Web servers or register DNS names in order to share content within a group of interest. BOPlish provides programmers with the needed infrastructure to resolve content names (URIs) to actual locations within a user network (Resolver API) and to acquire content from a specific host (Content API). On top of this infrastructure a wide range of novel use cases can be implemented. This paper describes in detail a selection of these usages that BOPlish enables. It characterizes the different requirements that each use case poses (such as peer identity verification, authentication, data integrity protection, availability/replication and fragmentation), outlines how these are met within the BOPlish infrastructure and gives an evaluation of how BOPlish must be improved to support them.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; D.1 [**Programming Techniques**]: General

General Terms

Design, Experimentation, Security

Keywords

Peer-to-Peer systems, Information-centric networking, Content dissemination

1. INTRODUCTION

Browser-based Open Publishing (BOPlish) envisions a new paradigm of content dissemination on the World Wide Web [9]. The concept is based on the idea that an ordinary Web application served by a Web server and running in a client browser uses WebRTC to establish Peer-to-Peer (P2P) connections to other instances of that application running on other browsers of the same User Community. A User Community is comprised of all users currently running the specific application inside their browsers.

BOPlish applications are initially served by a Web server over HTTP/HTTPS. The aim is to eventually form a P2P network between all browsers running that specific application [10]. Having joined the network a user is not dependant on the availability of the Web server anymore; in fact the server may be shut down without affecting the functionality of the network as such. Thus, BOPlish enables Web application developers to implement applications on top of that P2P network.

For this to work, BOPlish provides a platform consisting of basically two API layers as indicated in Figure 1. Every application using the JavaScript BOPlish library (or a compatible different library) can make use of those two API layers: The Name Resolver API is used to resolve a host-independent identifier in the form of the authority part of a URI to a specific host ID (a host ID can be interpreted as its unique address). This host ID is then used to directly connect to the given host and query content from it using the Content API. The latter is only roughly defined in [9] and thus a more detailed specification of the capabilities and constraints of the Content API are provided in this paper. Users share BOPlish-specific URIs that address a certain piece of content such as a file or a chat group. In essence BOPlish provides a location/identifier split, a vital part of Information-centric Networking (ICN, [1]).

The decision of how these URIs are structured has a high impact on the capabilities of the underlying architecture. One central benefit of Content/Information-centric Networking is that the names used to address content decouple the actual content from its location. In BOPlish this is done by a layer of indirection: The URI is resolved by the Name Resolver (making use of a DHT) to a location-specific name.

The charm of BOPlish lies in the fact that it is instantly deployable (by including the JavaScript code in an application) and provides users a content publishing solution without having to install any infrastructure or additional software. Each BOPlish instance makes up a so called User Community (or User Network) which is a network of individual users sharing a certain interest. The reasoning behind

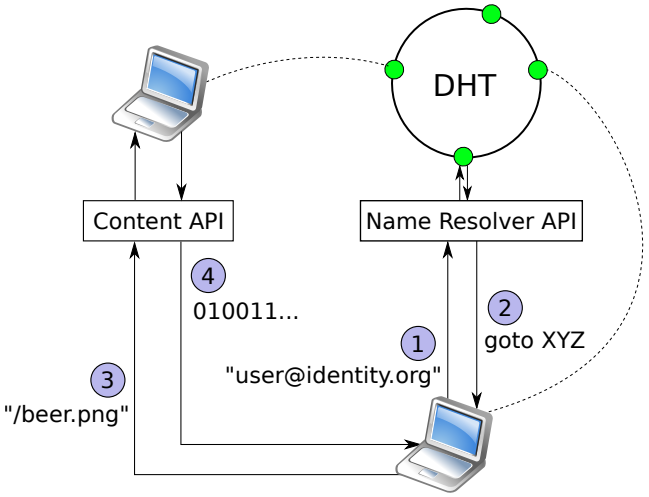

Figure 1: Nodes in BOPlish retrieve content by issuing a lookup of the content's user ID to the underlying DHT (1) which returns a pointer to the actual node that holds the content (2). This pointer is then used to open a WebRTC Data Channel to the peer, query for the content (3) and transfer it (4).

this approach is that every instance may run different applications, have different access privileges and, most importantly, scale well since they do not need to handle a large number of users as opposed to approaches that are to be rolled out internet-wide.

This paper discusses the implications of trying to implement use cases on top of BOPlish that are common to the Web as of today but also use cases that have been envisioned for ICN-like architectures. It starts with an outline of the problem space and an overview of related work in section 2. Section 3 then outlines different scenarios where BOPlish could be employed and ends with a discussion of the findings in 4 as well as an outlook into future work to make BOPlish a viable solution. As such, this paper is to be understood as an analysis of recent published work as well as a source for improvements to BOPlish.

2. PROBLEM STATEMENT AND RELATED WORK

These are the questions that this paper is ought to answer:

- Which refining do the Name Resolver and Content APIs need, if any?
- Does the joining and URI sharing model need further elaboration? (e.g. how does a user join multiple P2P networks? Is there one user network per server?)
- Which security constraints/requirements have to be modeled? How are DHT entries authenticated? How is namespace ownership guaranteed?
- Can the BOPlish architecture be used to demonstrate real-world use cases?

Evaluating the applicability of BOPlish for certain use cases implies an analysis of the employed naming scheme that has a direct consequence on usability, flexibility and security. BOPlish provides a hierarchical naming scheme based on URIs:

bop://<user>@<domain>/<path>?<sec-credentials>

Each URI's *authority* part (<*user*>@<*domain*>) is used as input into the Name Resolver API, eventually resulting in the caller to retrieve a concrete host ID that is associated with that authority. The *path* and *query* parts are application-specific parameters for the Content API. How URIs are published to the User Network and DHT entries are created, authenticated and verified is not mentioned in [9] and is part of the evaluation in this paper.

To determine which features and characteristics a Content-centric platform must provide one has to differentiate between features that are a direct property of the network infrastructure and those that are end-to-end features of each individual peer. This is especially important in the context of security. In [3] Ghodsi et al. define the three security features of data integrity, confidentiality and source authentication (provenance) as properties of end hosts and availability as a property of the network. With regards to availability they make the distinction between general underlay network availability (which has no association with the overlay architecture) and availability as a kind of denial-of-service countermeasure where the naming scheme plays a significant role.

The authors state that hierarchical names do not provide an intrinsic binding between a user's identity and the name as opposed to flat names where the name usually consists of a public key – which allows for the verification of the content's owner – and a label. Thus hierarchical names will have to rely on a different method for verification (Public Key Infrastructure, central third-party, etc.).

Usages for ICN approaches are described and evaluated in [6]. The intent of that draft is to establish a common understanding of possible ICN scenarios and to be able to compare different approaches with one another. The authors also discuss general properties such as performance, security, mobility or caching.

3. BOPLISH USAGES

This section sheds light on the different scenarios that are to be enabled by BOPlish. The documented use cases are either based on current usages of the centralized Web or influenced by usages explicitly outlined for ICN (e.g. in [6]). One of the examples for the latter is defined in [5]. The authors define a content-centric API based on location awareness. Peers can leave notes in the areas they are physically staying in or request notifications when entering specific geographical locations.

The ICN-specific use cases shall demonstrate whether the general BOPlish approach is powerful enough to mimic these (and thus provide the benefits of ICN as such) and how much the limited Web runtime (in comparison to native applications) is able to offer developers.

3.1 Prerequisites

When evaluation the use cases below we identified necessary properties that are common among all usages of BOPlish. This is especially true for peer identities. Every peer in a structured P2P network has to acquire a unique ID. This is used in the BOPlish DHT implementation to route traffic to that peer and to determine the storage range.

The user thus has to be assigned an ID unique to the specific User Network she wants to join. The assignment process needs to be secure in the sense that no user can (intentionally or unintentionally) be assigned an ID that's already assigned to another user. The ID is used to identify the user's peer instance in the DHT. Above that there is the need to identify real-world persons so that other users are able to make an assertion about the document's origin.

There are many approaches to secure ID assignment in structured P2P networks as well as to identity verification. RELOAD employs a central trust anchor that's responsible for assigning user names and IDs jointly [4]. Much of the concept of this process described by RELOAD could also be applied to BOPlish. For this to make sense the Web runtime – that has a very limited set of APIs – would need to offer strong cryptography functions to JavaScript applications. Currently no browser offers functions that would suffice for 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 and we are closely following the progress there [2]. In the meantime the missing functionality can be included by incorporating JavaScript implementations of cryptography functions into BOPlish. It has to be noted, however, that these cannot provide the same security as a native API because they do not have access to e.g. a cryptographically secure random number generator.

After the name acquiring process is done the user has a name and an ID the application can commence the join procedure for joining the overlay network that is already specified in [9].

3.2 Document Sharing

One major use case on the Web is that of sharing static documents between interest groups. There are very many services deployed on the Web to facilitate this usage, e.g. Dropbox[1], Google Docs[2] or Scribd[3]. The open-source software ownCloud [4] is an example of a service that users can host themselves. Generally users can upload documents to these platforms and grant access either publicly or with a defined group of collaborators. This is done by sharing an HTTP URL with them or by inviting registered platform members. The major drawback here is the reliance on a centralized service or the requirement to setup a custom server and handle propagation of DNS records. Also users have to trust the service provider with regards to content integrity, i.e. that the provider does not modify the uploaded content. A possible countermeasure is to attach digital signatures to the content or to provide checksums as is commonly done with software downloads.

The benefit of a User-centric approach is that every user has inherent control over a custom namespace and does not rely on central infrastructure. Once attached to the User Network one is able to publish content immediately. Thus, users could be enabled to share documents without having to rely on central service providers like the ones mentioned above. One scenario specific to BOPlish is that a user may want to provide a group of friends access to documents directly from her computer. The first step in this scenario would be to point the browser to an HTTP URL that retrieves the BOPlish-enabled sharing application.

After the joining procedure has succeeded she can actually start to publish the document to her interest group. This implies that the document is first made available to the Web application because browser's offer no way of accessing the file system directly. There are APIs for reading files that the user has explicitly made available to the application, though. The application then creates a BOPlish URI that denotes that document. An example URI looks like this:

bop://max@example.org/family.png?csum=sha256:a2bd...

This link is displayed to the user and ready to be shared. The sharing itself is done as with HTTP URLs, e.g. via XMPP or email. Using the Name Resolver API the others are then able to find out the ID of the host providing the document under the given name (*max@example.org*), connect to that host and employ the Content API to query for the document. The *csum* query parameter of the URI is used on the consumer side to verify the integrity of the document. This way the producer as well as all consumers can assure data integrity end-to-end, a property that is not provided by commonly used cloud providers today.

3.3 Content Search

Similar to the document sharing usage is that of searching for content on one particular peer. By changing the *path* part of the BOPlish URI in a way that allows for wildcards a search URI could look like this:

bop://max@example.org/Music/*johnossi*

Handing this path to the Content API would result in a list of files that match the given query and allow for retrieving a specific file.

3.4 Multi-protocol Usage

Since BOPlish just lays the ground for different use cases it makes sense to use one User Community for different purposes. A user may want to join a User Community and then share documents but at the same time initiate a text chat (see below). The URI syntax and semantics defined in [9] does not account for this fact because the URI has no way of specifying which use case the URI is meant to serve. Thus, it is advisable to include this information in the URI itself. Figure 2 shows a slightly different approach to the syntax.

Here, there is not authority part but rather a path component immediately following the scheme name `bop`. This path component is further built from the username serving

[1]https://www.dropbox.com
[2]https://docs.google.com
[3]http://www.scribd.com
[4]https://owncloud.org

as unique namespace (the old authority part) followed by a protocol identifier and optional protocol-specific parts. The protocol identifier now enables different usages in one community, e.g. a chat service and a document sharing service. One peer can then use that identifier to pass the URI to different modules of the application. Such a URI is generated for every published item.

```
(a) bop:namespace:protocol
      [/protocol-specific[?parameters]]
(b) bop:alice:document/img/images.png?
      csum=sha1:1234abc...
(c) bop:bob:search/Music/*johnossi*
(d) bop:carol:chat/room1
```

Figure 2: Enhanced BOPlish URI syntax (a) and example URIs of BOPlish content addresses for document retrieval (b), search (c) and text chat (d). The `csum` parameter in (b) is used for checking the integrity of the retrieved document. This way the producer as well as all consumers can assure data integrity end-to-end, a property that is not provided by cloud providers today.

3.5 Content Offloading

One of the main benefits of location-independent naming schemes is that content can easily be shifted from one host to another without losing accessibility of the content. This has various real-world applications; consider a user that has just published a document from her computer and is about to leave the house. She wants the content to remain available even after turning the computer off. In BOPlish this can be achieved with a three-step process:

1. Copy the content to another device
2. Join the User Network from that device
3. Update the name pointer of the affected content in the DHT

Step 1 can be seen as part of the BOPlish architecture but does not need to be. A user could leverage the document sharing capabilities outlined above transfer the content in question via a simple network transfer (e.g. using SCP) or a portable disk. Step 2 is done by starting a browser on the device and pointing it to the URL of the application serving as entry point into the User Network. Since [10] mentions the development of a head-less WebRTC component that will be able to run WebRTC applications it is also imaginable that the user fires up a BOPlish client on a server.

Joining the User Network implies that the new peer will be assigned a unique ID and – depending on the application running on top of BOPlish – supply authentication credentials. For being able to update the name pointer of the affected content the network must be able to determine whether the new user is allowed to change that specific DHT entry. Thus, two authentication schemes are necessary: One for joining the network and one for updating DHT entries.

For actually modifying the name pointer the Name Resolver API must provide a proper method. Handling secure addition and updating of key/value pairs in a DHT is covered by [8]. The authors propose an API where in a simple case the hash value of a secret is stored together with the actual value (in our case the name pointer). The secret is then obligatory to be able to remove that entry. An update is carried out by first removing an entry and then adding a new one. More sophisticated methods also described by the authors allow for the storage and retrieval of authenticated data using public/private key pairs. The Internet-Draft for RELOAD [4] proposes a similar approach in that each client may only store values that are authorized by its peer certificate.

3.6 Mobility

Being able to publish and consume content on the go would be a major benefit for BOPlish users. ICN promises to eliminate problems with mobility such as changing IP addresses and in a way BOPlish is able to provide similar "always-on" capabilities for content. The key factor when talking about mobility is "a host must be able to publish and/or consume content even when its network address changes (frequently)".

ICN approaches that are to be rolled out on the network layer, replacing IP, offer these capabilities by making content totally host-unaware. Since BOPlish works on the application layer and thus on top of IP it suffers from the same problems that IP suffers from when it comes to mobility (e.g. changing IP addresses).

The two usages of content offloading and content replication can help to counter these problems. When a user publishes content from a device and then is about to enter an area with limited connectivity she can offload the content to another host that ensures connectivity to the P2P network. Replication, on the other hand, solves the problem by disseminating the same content onto different peers. One problem that BOPlish faces here is that URIs are identity-driven which means that even if two different peers host the same content this content is not available under the same URI. For replication to work in BOPlish it is thus necessary to provide content for the same URI from different peers.

3.7 Real-time Text and A/V Chat

The name Web Real-Time Communication suggets that real-time messaging is already built into WebRTC. This is true in the sense DataChannels as well as audio/video (media) channels can be established between two peers using WebRTC. The tricky part, however, lies in addressing and reaching peers. Currently many solutions are being developed to enable real-time chat with WebRTC, e.g. EasyRTC[5], TokBox[6] and Vidyo[7]. What these providers have in common is that signaling (i.e. call setup) and addressing are centralized through their servers.

With BOPlish each user is able to get a unique address (its BOPlish URI) that may be used to initiate a call to that user. Let's consider the use case flow of two BOPlish

[5]http://easyrtc.com
[6]https://opentokrtc.com
[7]http://www.vidyo.com

users wanting to establish an audio/video chat between each other:

User Alice and user Bob both join a common User Network determined by the bootstrap server the employ. Alice tells Bob the name she uses on the network so that Bob can enter that name into the application's UI and hit a *call* button. Offers and answers are exchanged between the two browsers and eventually a WebRTC media session is established.

When looking at the process in detail there is nothing that has to be changed in the BOPlish architecture as of now (the joining is covered in the *Prerequisites* part of this section). What is different in comparison to the two use cases outlined above is that the URI does not have to contain a specific *path* part. It suffices for Bob to enter the URI bop://alice@example.org into his application. After hitting the *call* button the application exploits the Name Resolver API to retrieve Alice's ID. This ID serves as the anchor for routing the offer to Alice's browser which then sends out an answer and the call is established in a P2P manner.

What can be especially useful here is the feature of identity providers that is to be built into WebRTC implementations. The Rtcweb working group has specified the working of such a feature in [7]. The main building blocks are that Alice and Bob both log into a third-party service like an OpenID provider or Facebook. This provider then generates an assertion that is merged into the offer's and answer's SDP. On the remote side a peer sends this assertion to the identity provider used for generating the assertion. The provider then can verify it and inform the browser of the result. If it is positive the browser displays the remote peer's identity together with the video stream. This way BOPlish users have a standardized way of establishing authenticated media streams.

4. CONCLUSIONS AND FUTURE WORK

The use cases envisioned in this paper allow us to conclude that BOPlish is in principle able to provide the underlying architecture for applications that were not possible before on the Web. We have elaborated and analyzed specific usages so that we are now able to counter BOPlish's shortcomings and buff up the parts that are only roughly defined in [9].

One key aspect we will have to investigate further is that of name and ID assignment and verification which is a subject of vast research in P2P networks as a whole. In specifying a secure name/ID assignment process (most possibly using asymmetric cryptography) a ground would be laid for applications that are far superior to the centralized Web islands today.

The questions stated in 2 can now be answered more precisely with the help of the analysis provided in this paper. It proposes concrete steps for refining the BOPlish API layers and gives insights into improvement strategies regarding security constraints.

5. REFERENCES

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A Survey of Information-Centric Networking. *IEEE Communications Magazine*, 50(7):26–36, July 2012.

[2] D. Dahl and R. Sleevi. Web Cryptography API. W3C Working Draft, World Wide Web Consortium.

[3] 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*, ICN '11, pages 1–6, New York, NY, USA, 2011. ACM.

[4] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD) Base Protocol. Internet-Draft – work in progress 26, IETF, February 2013.

[5] J. Ott and J. Kangasharju. Opportunistic content sharing applications. In *Proceedings of the 1st ACM Workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, NoM '12, pages 19–24, New York, NY, USA, 2012. ACM.

[6] K. Pentikousis, B. Ohlman, D. Corujo, G. Boggia, G. Tyson, E. Davies, A. Molinaro, and S. Eum. Information-centric Networking: Baseline Scenarios. Internet-Draft – work in progress 01, IETF, October 2013.

[7] E. Rescorla. WebRTC Security Architecture. Internet-Draft – work in progress 07, IETF, July 2013.

[8] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: A public dht service and its uses. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 73–84, New York, NY, USA, 2005. ACM.

[9] C. Vogt, M. J. Werner, and T. C. Schmidt. 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, Oct. 2013. IEEEPress.

[10] 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, Oct. 2013. IEEEPress. ICNP Best Demo Award.