Persistent Naming for P2P Web Hosting

Md. Faizul Bari*, Md. Rakibul Haque*, Reaz Ahmed†, Raouf Boutaba* and Bertrand Mathieu‡
*David R. Cheriton School of Computer Science, University of Waterloo
†Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology
‡France Telecom, France
Contact: mfbari@uwaterloo.ca

***Abstract*—The peer-to-peer paradigm has great potential of contributing to the next generation web-hosting infrastructure. Profound advancements in P2P technologies in the last decade have proven their capability to provide the same functionality as traditional client–server systems at a much larger scale and much lower cost. Existing centralized website hosting technology has a number of inherent deficiencies, including: scalability issues; single point of failure; administrative overhead; and hosting expenses. P2P Web hosting can effectively address these problems and open a new era for the next generation web technology. Unlike the current web however, peer availability and content placement are highly dynamic in P2P networks. This dynamism raises a number of research challenges related to naming, indexing, searching and hosting in P2P environments. In this paper we identify the practical requirements for devising a persistent naming scheme for P2P web-hosting on top of highly dynamic non-persistent P2P networks and present a novel naming architecture for satisfying these requirements.**

I. Introduction

Peer-to-Peer (P2P) computing represents the concept of sharing resources available at the edges of the Internet [1]. The P2P paradigm dictates a fully distributed, cooperative network, where nodes collectively form a system without any supervision. Its advantages include robustness to failures, resource sharing, self organization, load balancing, data persistence, publisher anonymity, etc. Since the initial success of Napster [2], P2P technology has enjoyed ever growing popularity for online file sharing, amongst a large community of Internet users. Profound advancements in P2P technologies in the last decade have proven their capability to provide the same functionality as traditional client–server systems at a much larger scale and much lower cost. From the perspective of World Wide Web, P2P technology has good potential of becoming the next generation web hosting infrastructure as well. This paper is a first step towards the realization of a P2P network for server less web hosting. The contribution of this paper is a novel naming scheme crucial for realizing server less web hosting over structured P2P networks.

Contemporary web hosting technology is built around client–server architecture, which has a number of inherent deficiencies including scalability issues, a single point of failure, administration overhead and hosting expenses. If the websites were distributed over a P2P network then there would be virtually no limitation on the number of users who can access a popular website. Even with high-end servers and geographically distributed content distribution systems, flash crowds is still an inherent problem in the Internet. With a P2P based web hosting architecture there will be no single point of failure, as many peers will be hosting a website. Content caching will be built intrinsically into the system. A network composed of contemporary machines will be able to host web pages without the need for expensive high-end web servers. There will be no need for dedicated website administrators and experts for web server maintenance. Finally, such a system will provide an uncensored platform that will promote freedom of speech. That can have significant social and political impact on the society. However, P2P technology can not be directly applied for server less web publishing for two main reasons. Firstly, the participating peers in a P2P system join and leave the network very frequently, whereas a web server remains up for years continuously. Secondly, a shared content in P2P systems usually keeps moving from one peer to another, while web pages do not usually change location within the Internet. Due to this dynamism in the peers and the shared contents, a number of naming related research challenges need to be addressed before successful realization of server less web hosting over P2P networks. Some of these challenges are as follows: location and time independent naming of web pages, dynamic namespace management, persistent hyperlink references, and distributed name registration and resolution. Here we have proposed a persistent naming scheme that addresses all these issues.

The authors in [3], [4] presented a DHT based routing and indexing technique called Plexus for structured P2P networks which provides efficient mechanisms for partial keyword based content advertisement and discovery. Plexus also provides a robust routing protocol built on top of the concept of error correcting codes. In this paper we leverage the properties of Plexus to build our P2P web hosting stack. We primarily focus on the architecture of the naming system required for P2P web hosting. Our aim is to devise a location and time independent naming scheme, persistent linking of web documents, and distributed name registration and resolution in dynamic P2P environment. These properties are crucial for achieving a complete architecture for server less web hosting over structured P2P networks.

The rest of the paper is organized as follows: the research challenges faced in designing our naming system are presented in Section II. Section III describes some preliminaries essential for a better understanding this paper. A brief description of our P2P web hosing architecture is presented in Section V and our proposed naming scheme is presented in Section VI. Experimental results and evaluations are reported in Section VII and finally we conclude in Section VIII.

II. Challenges

This section highlights the research challenges for devising a naming scheme for serverless P2P web hosting.

Web documents are identified using Unified Resource Locators (URLs), which form the hyper-link structure of the World Wide Web. However, URLs are not suitable for naming in the P2P context, due to the dynamism in peer population. Recall that the domain name part of a URL essentially specifies the server that hosts the document in the Internet. But, in P2P environment there is no guarantee of a stable location for a document. A Domain Naming System (DNS) resolver maps URLs to server IP addresses, which allows site relocation and replication without affecting the URL. Site relocation is relatively less frequent than P2P content dynamism, and DNS updates are more static compared to P2P index updates. Hence, URL based naming and hierarchical DNS lookup are not suitable for P2P Web hosting.

A suitable naming system for P2P web hosting should support names that are independent of spatial and temporal scope so that any peer holding a valid copy (replica) can serve as a source. In the Internet DNS resolves the domain name part of a URL to a web server IP address. While this strategy worked successfully for the Internet, it is not feasible for the P2P environment for obvious reasons. In a P2P environment peer uptime is dynamic and unpredictable, so if we want to provide 24/7 uptime for a website then the website must be replicated to multiple peers. We can not bind names to peers, instead names must be bound to contents. A single name should be able to identify the same content hosted on any peer in the network.

Names in a P2P web hosting environment should be flexible and human–friendly. While a hierarchical naming structure facilitates efficient routing and namespace management mechanisms, it makes the names hard to remember. If we look at P2P file sharing networks like BitTorrent and eMule, the users are accustomed to a flat namespace with no predefined structure. Though DNS provides strictly structured names for the Internet, we choose to provide flexible, human–friendly, flat and unstructured namespace for P2P web hosting for seamless user experience in the P2P community.

Providing support for persistent bookmarking in P2P web hosting environment is not trivial. Whenever a user visits a website he may want to bookmark it for later visits. Implementing bookmarking in the P2P environment is not so easy as there is no one peer hosting the website 24/7. Websites keep moving between peers and the name resolution system has to cope with such dynamism.

Finally the name registration and resolution systems must be distributed, fault-tolerant, and scalable. This implies that the responsibilities of name assignment and storage of name mappings are distributed among the peers, which mandates the use of sophisticated techniques for responsibility assignment, load balancing, data synchronization, management of stale data, and handling registration expiries.

III. Preliminaries

In this section we introduce some terminologies and concepts that are necessary for better understanding of the material presented in this paper.

A. Linear Covering Codes

A linear covering code $(n, k, d)f$, over $\mathbb{F}_2^n$, is a *subspace* $\mathcal{C} \subset \mathbb{F}_2^n$. Each element in $\mathcal{C}$ is a *codeword*. $|\mathcal{C}| = 2^k$ and d is the minimum Hamming distance between any two codewords. The *covering radius* f is the smallest integer such that every vector $P \in \mathbb{F}_2^n$ is covered by at least one $B_f(c_i)$. Here, $B_f(c_i) = \{P \in \mathbb{F}_2^n | d(P, c_i) \leq f\}$ is the *Hamming sphere* of radius f centered at codeword c_i.

B. Reed-Muller Codes

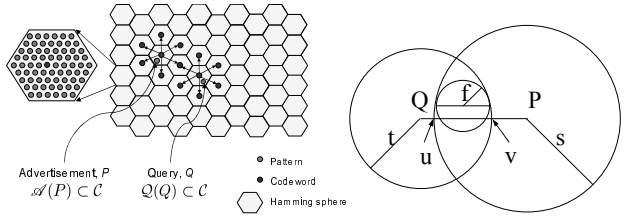
Reed-Muller (RM) codes are a family of linear error correcting codes. The r-th order RM code is denoted by $RM(r, m)$, which is a vector subspace of length $n = 2^m$ over $\mathbb{F}_2^n$, for some positive integers r and m. Minimum distance d, between any pair of codewords is 2^{m-r} and the number of codewords is 2^k, where $k = \sum_{i=0}^{r} \binom{m}{i}$ is the dimension of the code. In this paper, we use $RM(2, 7)$ codes, where the length of a codeword is 128 bits, minimum hamming distance between any two codewords is 32, and the number of total codewords is 2^{29} (more than 5×10^8).

C. List Decoding

Let C be a error correcting code (n, k, d) and x be a pattern(binary) of length n, then a list decoding algorithm provide a set of codewords $X = X_1, X_2, \ldots, X_m$ where $X_i \in C$ and Hamming distance from x to each X_i is at most e (error-bound). List decoding algorithms are widely used in data communication where a set of valid codewords of used correcting code are generated from a received erroneous message.

D. Plexus Content Advertisement and Discovery

In Plexus [4] keywords are mapped to bloom filters [5] (or bit-vectors) and a novel Hamming distance based technique derived from the theory of *Linear Covering Codes* is used for routing. The keyword to bloom filter mapping process retains the notion of similarity between keywords, while Hamming distance based routing delivers deterministic results and efficient bandwidth usage.



(a) Hamming distance based indexing (b) Bound on $d(P, Q)$

Fig. 1. Core concepts in Plexus

In Plexus, advertisements and queries are routed to two different sets of peers in such a way that the queried set

of peers and the advertised set of peers have at least one peer in common, whenever a query pattern is a subset of the advertised pattern. As explained in 1(a), a linear covering code ($\mathcal{C}$) partitions the entire pattern space $\mathbb{F}_2^n$ into Hamming spheres, represented by the hexagons in the figure. A codeword ($c_i \in \mathcal{C}$) is selected as the unique representative for all the patterns within its Hamming sphere. The challenge is then to map a query pattern Q to a set of codewords ($\mathcal{Q}(Q) \subset \mathcal{C}$) and to map an advertised pattern P to another set of codewords ($\mathscr{A}(P) \subset \mathcal{C}$), such that $\mathcal{Q}(Q)$ and $\mathscr{A}(P)$ have *at least one* codeword in common whenever the 1-bits of Q constitute a subset of the 1-bits in P. Mathematically,

$$Q \subseteq P \implies \mathcal{Q}(Q) \cap \mathscr{A}(P) \neq \emptyset \tag{1}$$

Given a code $\mathcal{C}$, a trivial way of computing $advSet$ and $qSet$ is to use list decoding; i.e. $\mathscr{A}(P) = B_s(P) \cap \mathcal{C} = \{Y | Y \in \mathcal{C} \wedge d(Y,P) \leq s)\}$ and $\mathcal{Q}(Q) = B_t(Q) \cap \mathcal{C} = \{Y | Y \in \mathcal{C} \wedge d(Y,Q) \leq t)\}$, for positive integers s and t. The goal is to compute the minimum $d(P,Q)$ without violating Equation (1). If the covering radius of $\mathcal{C}$ is f then the Hamming sphere of radius f around any arbitrary point should contain at least one codeword. Hence, according to Figure 1(b): $d(P,Q) \leq s + t - 2f$. In other words, if advertisement is done to all the codewords in $B_s(P) \cap \mathcal{C}$ and search is done on all the codewords in $B_t(Q) \cap \mathcal{C}$ then any subset Q of P within distance $d(P,Q) \leq s + t - 2f$ can be discovered.

E. Generating UUIDs

Universally Unique Identifier (UUID) [6] is an identifier standard used to enable distributed systems to uniquely identify resources without significant central coordination. One can generate a UUID for identifying an object with reasonable confidence that the same identifier will not be used by someone else to identify another object. A UUID is a 128-bit character sequence. Number of theoretically possible UUIDs is about 3×10^{38}. Typically a UUID consists of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters (32 digits and 4 hyphens). For example: 580f8760-e09b-42d4-a716-876543450000. Here the word unique means practically unique rather than guaranteed unique. Since the identifiers have a finite size it is possible for two different objects to share the same UUID. Random UUID generated by the `java.util.UUID` class have 122 random bits. The remaining 6 bits are used for version and variant number. With random UUIDs, the chance of two UUIDs having the same value can be calculated by using the Birthday paradox. Which yields the following probability:

$$p(n) \approx -e^{-\frac{n^2}{2x}}, \text{ where } x = 2^{122} \tag{2}$$

Using Plexus with $RM(2,7)$ codes gives us a maximum network size of 2^{29}. Then according to equation 2 the probability of a single UUID clash in this network is 3×10^{-20}.

F. Replication Groups

As pointed out in Section II providing 24/7 uptime for a website in a P2P environment is not a trivial task. To ensure content persistence we need to provide a persistent storage over non-persistent P2P network. In [7], a mathematical model for measuring time-based and presence-based availability has been presented. In [8], peers are grouped based on the probability of being online and erasure codes are used to improve content availability. In [9], a replication strategy for improving availability in structured P2P network has been proposed. That scheme is based on the mean-time-to-failure of a replica. In [10], [11] gossip based protocol has been used to replicate content based on tentative lifetime of the peers. A replication mechanism based on the diurnal availability pattern of the peers participating from different time zones across the Globe is presented in [12] which work across temporal and spatial dimensions. Peers from different time zones form a group for hosting web sites in turn, round the clock. We use this replication grouping mechanism for providing content persistence and primarily focus on the details of the naming system in this paper.

IV. Related Work

The challenge of achieving persistent names for shared content in a P2P network has not been addressed so far. File sharing P2P systems (e.g., Gnutella, Kaaza, Morpheus etc.) use keyword lists for content identification and randomly selected temporary identifiers for peer names. Web browsing over P2P networks is offered by FreeNet [13], FlashBack [14] and Web2Peer [15], under the assumptions that only static webpages are hosted and page replicas are independent. Webpage availability is increased through replication over the P2P network. In all of these systems, the P2P network is used for locating and caching the webpages and Internet Web servers still serve as the source of webpages cached in the P2P system.

BitTorrent [16] uses SHA-1 hash values for assigning ids to file chunks, while peers are identified by IP:Port values. Though it is quite successful in distributing large media files, it is not suitable for hosting regular websites. Web sites typically contain images that are quite small. BitTorrent is designed for handling large files, which require minutes or hours rather than seconds. BitTorrent uses a significant amount of time to try out and compare different connections which introduces large amount of latencies. This latency may be negligible for large media files, but it may surplus the actual time required to download small images embedded in web pages.

Persistent naming is also being investigated for supporting movement of electronic documents in the Internet . Whenever a document's URL is changed, all pages referencing that document need to be updated. Which is a cumbersome and error prone process. To solve this problem the Digital Object Identifier (DOI) system [17] provides persistent and unique DOI links for electronic documents. But the architecture of DOI is centralized and requires human intervention for maintaining the DOI links upto date. A number of research projects, including DONA [18], NetInf [19] are proposed as next generation architectures for the Internet. They propose to use self-certifying names to identify and authenticate objects in the Internet. Typically hash of public keys are used as content

names. Though these schemes provide security against name thefts, names are no longer human-friendly.

DONA proposes a clean slate redesign of the Internet protocol stack and proposes to replace DNS names with flat, and self-certifying names having the form $P : L$, where P is the cryptographic hash of the Principal's (content owner) public key and L is a label chosen by the principal. For name resolution, DONA uses the route-by-name paradigm. Resolution infrastructure consists of *Resolution Handlers* (RH) similar to the hierarchical architecture of DNS following the same parent-child paradigm and each domain must have one logical RH. NetInf [19] follows the same naming scheme as DONA but the current open source implementation of NetInf called *OpenNetInf*, resolve NetInf identifiers into locators (e.g. URLs) using a P2P-based Name Resolution Service with three accessible nodes. The naming schemes proposed by DONA and NetInf can not be adapted for pWeb for the following reasons: (i) due to dynamism of content location in a P2P networks, establishing a DONA like hierarchically organized RH infrastructure will be very expensive in-terms of computation and network usage, (ii) the P2P-based Name Resolution Service provided by NetInf is essentially a centralized cluster of three nodes which is separate from the rest of the network providing the content, so there is a large architectural difference between NetInf and pWeb. NetInf names actually resolves to URLs in the Internet, which does not solve the problem of location independent content naming in pWeb.

In existing P2P systems (structured and unstructured) [20] peers are considered to be memoryless, i.e., peers are not assumed to retain knowledge about the overlay network from their previous sessions. Hence, the requirement for assigning persistent names to peers and content is not important. But for P2P web hosting we have to ensure persistent names for peers and websites. A number of research works, including [21]–[24], focus on implementing DNS lookup using P2P systems. All of these works use DHT-based techniques for P2P lookup and support exact name matching. P2P web hosting requires partial name resolution along with a way to map a name to the address of a live peer in the replication group.

V. System Architecture

Our P2P web hosting stack in built on top of Plexus routing and indexing framework. Though in [4] Plexus used Golay Codes for routing, here we use the Plexus variant using Reed-Muller Codes introduced in [25]. Using Reed-Muller codes, Plexus can scale to millions of peers without using subnets or super-peers. This way the overlay network can have a flat and symmetric structure. On top of this overlay network, replication groups are formed based on uptime history of peers as discussed in Section III-F. The functional dependencies between the architectural components have a direct impact on the naming scheme and vice versa. For a better understanding of the requirements of the naming system three crucial processes namely: Advertisement, Peer Join and Group Maintenance, and Query are presented below:

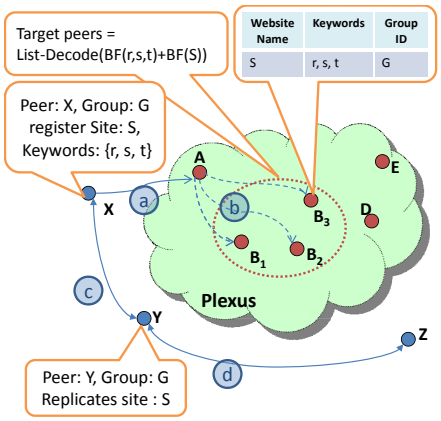


Fig. 2. Advertisement of a Site

1) Advertisement Process: To facilitate efficient search of web sites we use a distributed Hamming distance based indexing mechanism on top of Plexus routing. As depicted in Fig. 2, the advertisement process consists of the following four steps:

- Step a: Each peer in the system will belong to a replication group. Suppose peer X belongs to group G and wants to advertise site S. The search keywords (or other meta information) related to site S are r, s and t. Peer X sends this information to a peer A in the Plexus indexing framework.
- Step b: In this step, peer A creates two advertisement patterns (basically Bloom Filters). One from the meta information (i.e., r, s and t) and the other from the website name. Then peer A list decodes the patterns and computes the set of codewords within a pre-specified Hamming distance from the advertised patterns. Then it uses the Plexus routing mechanism to multi-cast the site index to the peers (B_1, B_2, B_3) responsible for the codewords. The indices stored in the indexing peers (i.e., B_i's) are in the form of <Website Name, Keywords, Group ID> triplets.
- Step c and d: Newly hosted sites or updates to existing sites are propagated to all the members (i.e., peers Y and peer Z) of the hosting peer's (i.e., X's) replication group (i.e., group G). This replication takes place whenever a group member rejoins the network. More detail on the peer join and replication process is given next.

2) Peer Join and Group Maintenance Process: In order to maintain diurnal availability, peers collaborate in small groups in such a way that at any given time instance at least one peer from a group is online with very high probability. Content in each peer is fully replicated and synchronized. To ensure replica consistency, whenever a peer joins or returns to the system it finds active group members, updates its online status and replication contents in the following manner:

- Step α: As depicted in Fig. 3, peer Z, a member of group G, becomes online after being offline for a while. Once online peer Z requests a peer, say C, in the Plexus indexing framework to find the members of its own group G. In case of a new peer, its replication group is determined based on its uptime history.

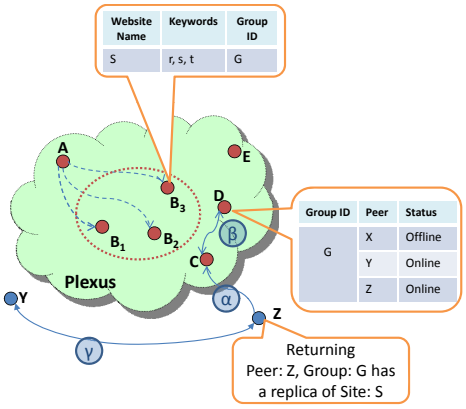


Fig. 3. Accommodating new/returning peers

- Step β: Peer C, constructs a pattern (Bloom Filter) from the group ID G, decodes the pattern to find the closest codeword, and routes the query to the responsible peer (here D) using Plexus routing. Upon receiving the query, peer D updates the current status of peer Z to online, records its IP:port and returns the IP:port list of all of the online members of group G.
- Step γ: After learning about the current online members of group G, peer Z synchronizes website replicas with other members.

3) Query Process: There can be two types of queries: a) keyword search and b) name search. The second type is more straight forward and a subset of the first type. Hence we explain here the first type only, i.e. query by keywords. As depicted in Fig. 4, the keyword search process can be performed in the following four steps:

- Step 1: In the example scenario peer W is searching for the sites that have keyword r. It first sends the query to a random peer, say E.
- Step 2: Upon receiving the query, peer E constructs a query pattern (a Bloom filter) from the query keywords and uses list decoding to find the codewords within a pre-specified Hamming distance. Then it uses the Plexus routing to forward the query to the peers that are likely to have the meta-information on sites with the queried keywords. In the example, peer B_3 responds with the site name, keywords and the group ID (G) of the group hosting the site.
- Step 3: Once peer E receives the group ID G, it queries the Plexus network (similar to the rejoin process) to find the list of currently online members of group G. In this instance the IP:Port of peer Z will be discovered and returned to the original querying peer W.
- Step 4: Now peer W can directly browse the site from peer Z.

These processes are at the core of our P2P web hosting framework. The structure of names for web contents, peers, and replication groups are presented in the next section along with mechanisms for name registration and resolution.

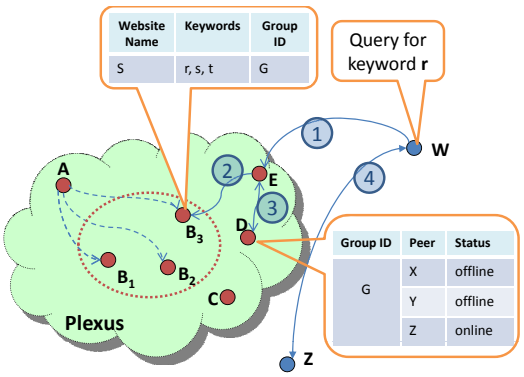


Fig. 4. Keyword-based content searching

VI. Proposed Naming Scheme

In all subsequent sections we refer to our P2P web hosting framework as pWeb. Naming in the pWeb architecture requires a flexible scheme that is human friendly, persistent and independent of spatial and temporal scope. The pWeb naming system is composed of two components: pWeb naming authority and pWeb name resolution system. Both of these components are distributed, highly scalable, and fault-tolerant. We propose a multi-faced naming called pRLs, which stand for pWeb Resource Locator. Figure 5 shows the structure of a pRL. The *NSID* part of the pRL is used to distinguish between the supported naming schemes in pWeb. It gives us the flexibility to incorporate future changes in the naming scheme. The *WebID* is the name of the website which can be assigned by the user or system. It can be a human-friendly or a secure (Hash of public key) name assigned by the user or system generated UUID . A user can assign any valid sequence of characters or an ontological name structured according to predefined categories or hash of his public key as the *WebID*. Pages within a website are referred by the *ObjectID* part of pRL which will be explained shortly. Metadata is associated with each website in pWeb, which facilitates attribute–value and keyword based search. The metadata also stores security related information such as public keys and digital signatures required for secure name publishing.

Before going into further details we would like to describe Zooko's Triangle [26], which identifies the possible tradeoffs between three properties of a naming scheme for a distributed system. It states that names can simultaneously have any two of the following properties:

- **Secure:** A name surely addresses only a given content
- **Memorable:** A human can remember a name
- **Decentralized:** A name can be chosen distributedly

For example, self-certifying names presented in DONA [18] and NetInf [27] are secure and decentralized but not memorable. Nick-Names used by Skype are memorable and decentralized but not secure. DNS names are memorable and secure but centralized. In pWeb, names must be distributed, so we have to choose between secure and memorable names. As already mentioned in Section II, we prefer human-friendly (memorable) names for web contents for providing better user experience. However if a website publisher wants to provide secure names, then he can create a self-certifying name by

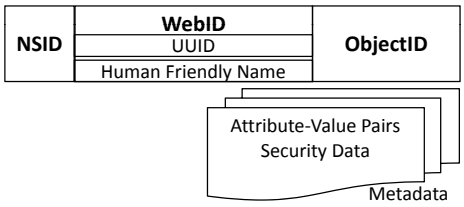


Fig. 5. Structure of a pRL

attaching the hash of his public key with the website name. Cryptographically constructed self-certified name can be used to securely verify if the associated content is the original one or not. A publisher having public key P and secret key S, can securely publish a website with the name *Hash(P):L*, where L is a locally unique label within the publisher's domain. The digital signature (content hash signed by S) is stored in the metadata so that no one can forge a name without knowing the secret key S. This way we have memorable-distributed and secure-distributed naming schemes and it is up to the publisher to choose the one suitable for his/her website. From the perspective of name assignment and name resolution both naming schemes are handled in a similar manner, as will be explained in the next section. Now, we introduce the entities that need to be named in pWeb along with their naming requirements.

A. Entities and Requirements

To achieve persistent object naming on non-persistent and transient P2P networks we need to name four entities in the pWeb system. These entities are: Websites, Web Contents, Peers, and Replication Groups. We describe each of these entities and their naming requirements below.

1) Website Naming: pRLs are used to refer to websites in pWeb just like URLs are used in the Internet. As pRLs are assigned to websites instead of peers, name persistence is supported across peer sessions and any peer with a valid copy (replica) of a website can serve as a source. To separate a website from its hosting peer, we use separate namespaces for websites, peers, and replication groups. We allow multiple websites to have the same name for user assigned unsecure names. In case of user assigned secure names and system assigned UUID website names are unique across the whole overlay network and this uniqueness is guaranteed by the pWeb naming system as described in Section VI-B. Whenever a website is replicated to a particular group the group leader generates an UUID for the website which is unique within the group and stores it in the metadata along with the replication group's UUID. The system uses these UUIDs for disambiguating between multiple websites having the same name. The users of the system can disambiguate between websites by using the associated attribute–value pairs and keywords.

2) Web Content Naming: For naming webpages or other files within a website we use relative naming and this information is stored in the *ObjectID* field of pRL. It is the responsibility of the website publisher to make *ObjectID*s unique within his domain. Metadata is also associated with each webpage or file for providing attribute–value and keyword based search. Some example pRLs are shown below.

- `ptp://wc.v1:bobshome`
- `ptp://wc.v1:bobshome/about`
- `ptp://wc.v2:sci:net:p2p:bobshome`
- `ptp://wc.v2:sci:net:p2p:bobshome/about`
- `ptp://wc.v3:f47ac10b-58cc-4372-a567-0e02b2c3d479/bobshome`
- `ptp://wc.v4:2fd4e1c6.7a2d28fc.ed849ee1.bb76e739.1b93eb12/bobshome`

Here `ptp` stands for pWeb Transport Protocol and `wc` means that this name refers to a web content. The `v1`, `v2`, `v3`, and `v4` in the pRL are the *NSID*s, which identify the version of the naming scheme in use. Here `v1` represents names without any ontological structure and `v2` refers to names with ontological structure. `con.v1:bobshome` and `con.v2:sci:net:p2p:bobshome` represent the *WebID* and `about.html` is the *ObjectID* part of the pRL. `v3` and `v4` represent system generated UUID and user assigned secure names respectively. UUIDs are generated as described in Section III-E. The SHA-1 hashing algorithm is used by the secure naming scheme for generating *WebID* from publisher's public key. Human-friendly and distributed names are provided by version `v1` and `v2` naming schemes. Version `v3` and `v4` provide secure and distributed names.

3) Peer Naming: Each peer in the pWeb overlay network is assigned an unique system generated identifier called *peerUUID*. When a peer P wants to join the pWeb overlay network, it generates a random UUID (as described in Section III-E) and sends a join request to any random peer R. Now peer R forwards this message to the pWeb naming authority for name registration. Peer P is then notified by the pWeb naming authority whether name registration was successful or not. For returning peers, a challenge/response mechanism [28] is used for reclaiming previously registered *peerUUID*s. The details about how pWeb naming authority performs these operations will be presented in Section VI-B.

4) Replication Group Naming: Each replication group is assigned an unique system generated identifier called *groupUUID*. We do not require the group identifiers to be human-friendly, since these identifiers are used internally by the name resolution mechanism for locating the currently active (i.e., alive) replica of a website. So like peers, replication groups are also assigned UUIDs. *groupUUID*s are registered by the first peer forming the group and the registration process follows the same steps as peer name registration.

All four entities discussed above register their names through the pWeb naming authority. Name resolution is done by the pWeb name resolution system. Given any pRL, it first resolves that pRL to a replication group's *groupUUID* and in the next step the *groupUUID* is resolved to the IP:Port pair of an active peer (group leader) who is currently responsible for representing that group. These mechanisms are described in greater details in the following sections.

B. pWeb Naming Authority

The core functionality of the pWeb naming authority is to register names and preserve name uniqueness. It uses Plexus

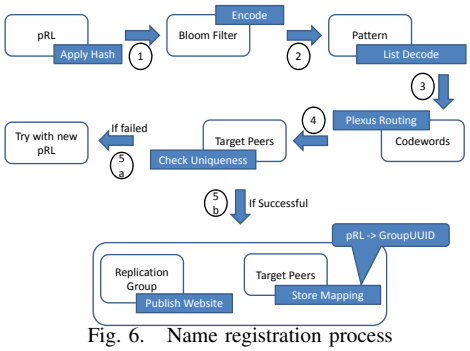


Fig. 6. Name registration process

routing for forwarding name registration messages. It is fully distributed and the namespace is partitioned among the peers. Each peer is responsible for maintaining only a portion of the namespace. It utilizes the Plexus's linear covering code based partitioning technique for partitioning the namespace among the peers. Name registrations are handled much like Plexus content advertisement described in Section III-D.

Let us assume that a peer A wants to publish a website with pRL S. First A creates a bloom filter BF from S by applying multiple hash functions as explained in Section **??**. Then by using the list decoding function of the underlying error correcting code, A generates a set of codewords $LD(BF)$ that fall within the hamming ball centered at the codeword representing BF. After that A uses Plexus routing to send name registration messages to the peers that are responsible for these codewords. Let the set containing these peers be represented as $P_{LD(BF)}$. At this point the peers in $P_{LD(BF)}$ check whether pRL S is unique or not. However if the website name follows user assigned unsecure naming scheme (`v1` or `v2`) then this checking is skipped. The peers can determine whether to perform uniqueness check from the *NSID* field of the pRL. If any of the peers in $P_{LD(BF)}$ finds pRL S to be not unique then it sends a "pRL Registration Failed" message to A and A retries with a new pRL S'. When pRL S is found to be unique, the peers in $P_{LD(BF)}$ notify peer A that its pRL registration was successful. Peer A now publishes its website and pRL to the pWeb network. The website is replicated at multiple peers in its replications group G and the associated mapping from pRL S to replication group's *groupUUID* is stored in all peers in $P_{LD(BF)}$. The group leader L of replication group G becomes the primary contact point for this website. These steps are shown in Figure 6.

As peer population in a P2P network is highly dynamic, it may happen that all peers indexing the mapping data for pRL S go offline. In such a situation no one will be able to browse S (though it may be alive in its replication group), until the original peer comes back online and re-registers it. To solve this problem we adopt a simple and effective data refresh strategy. As already mentioned, any published content gets replicated within a particular replication group and the group forming process ensures that at least one peer will be alive at any time. We leverage this for refreshing a pRL mapping. One thing that needs to pointed out here is that, a peer may or may not participate in replicating its own content. Currently the peer with the smallest IP:Port address is chosen as the

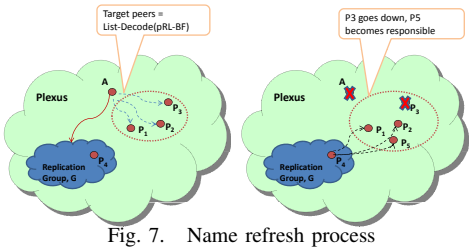


Fig. 7. Name refresh process

group leader. Every peer in a replication group keeps track of all other alive peers in the same group which makes the group leader selection process trivial.

This situation is depicted in Figure 7. When peer A publishes its website, the content is replicated in replication group G and its pRL to *groupUUID* mapping is stored in $P1$, $P2$, and $P3$ (Figure 7 (a)). Here pRL-BF denotes the bloom filter that is generated from the pRL of A's website. The group leader of group G is peer $P4$. After some time peer A goes offline. Now it is the responsibility of peer $P4$ to keep peer A's pRL mapping data refreshed. So $P4$ keeps refreshing A's pRL mapping after a predetermined time interval. In Figure 7 (b) both peer A and $P3$ have gone offline and the responsibility of $P3$'s namespace is now assigned to another peer say $P5$. Now $P5$ is also responsible for storing A's name mappings. Group leader $P4$ refreshes A's pRL mapping which gets stored in peer $P5$. This mechanism fails in the rare situation when all members of a replication group go offline. In this case the pRL will become available only when the original peer A comes back online and refreshes its pRLs.

Peer UUIDs are published just like pRLs where the bloom filter is created from the *peerUUID*. The mapping from *peerUUID* to the peer's IP:Port pair is stored in a tuple like ⟨*peerUUID*, IP:Port⟩ in the indexing peers. The responsibility for periodically refreshing this mapping data is assigned to the corresponding peer. Replication group *groupUUID*s are published in the same way but here the IP:Port pair refers to the current group leader. The responsibility for periodically refreshing mapping data is also assigned to the group leader. The *groupUUID* must always resolves to the IP:Port pair of an active peer. So when ever the group leader changes (due to peer failure/churn), the new group leader updates the corresponding *groupUUID* mapping data with its own IP:Port pair.

C. pWeb Name Resolution System

Its main purpose is to resolve a pRL to a currently alive peer's IP:Port pair. For this purpose it uses the same mechanism as pWeb naming authority. Figure 8 shows the steps required for resolving a website pRL. Suppose peer A wants to resolve a pRL S. In step 1, peer A creates a bloom filter from S. This responsibility can also be handed over to another peer as Shown in Figure 4. In Step 2 peer A encodes the bloom filter to get a valid codeword C. Next in step 3 peer A list decodes C to get a list of codewords LD within a hamming ball of predetermined radius centered at C. In step 4, the peers responsible for the codewords in LD are contacted for the *groupUUID* of Group G hosting the website. After getting the *groupUUID*, another lookup is performed by peer

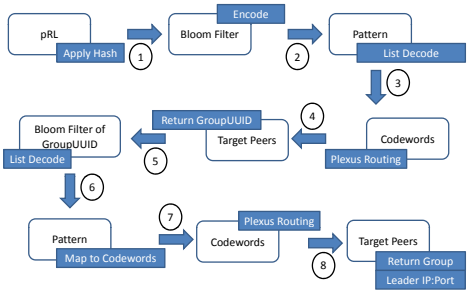


Fig. 8. Name resolution process

A in the same manner to find the IP:Port pair of the current group leader of replication group G. After that peer A can browse the website from the returned IP:Port pair. We have implemented A browser plug-in that enables a browser (e.g. Firefox, Chrome) to browse contents from pWeb. This plug-in enables the browser to handle pRLs just like URLs. Whenever a user clicks on a pRL, the plug-in resolves it by invoking the pWeb name resolution system bypassing the DNS and then the desired contents are fetched from the hosting peer(s).

VII. Experimental Evaluation

In this section we evaluate the performance of our proposed naming system on three different aspects: scalability, fault-tolerance, and effects of refresh rate.

A. Simulation Setup

We have developed a multi-threaded simulator in Java for performing our experiments. In our experiments, we start with a few peers and gradually grow the network to our desired size. Peer arrival and departure events are timed by a Poisson process, making them uncorrelated and bursty. Poisson means for peer arrivals and departures vary between 20 to 100 milliseconds which corresponds to 2 to 7 peers arriving or departing per second in a network of 20,000 peers. Peer uptime follows a power-law distribution where most of the peers have a very short uptime but the rest of them have larger uptimes. Similar peer churn and uptime models have been used in [29], [30]. For scalability related experiments, once a target network size is reached peer arrival and departure Poisson means are adjusted so that the network size keeps varying between a predetermined maximum and minimum limit over time. During this period, name registrations and resolutions are performed and performance related statistics are collected. For experiments related to fault-tolerance, we start with a network of 20,000 peers and gradually decrease its size to 5,000 by setting peer departure rate much higher than the peer arrival rate. Network statistics are collected throughout this period. Peer arrival and departure events are handled by separate threads for making these events independent of other operations. A third thread is used to capture system snapshots for collecting statistical data.

B. Scalability

For measuring scalability of our system we have measured three metrics: the average number of name mappings stored per peer, average hop counts per name registration and average hop counts per name resolution. For all these experiments,

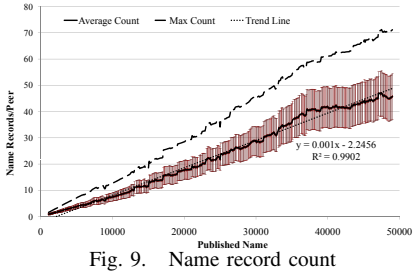
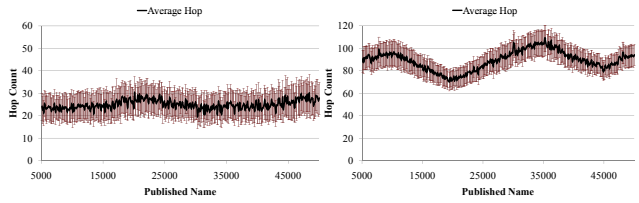


Fig. 9. Name record count

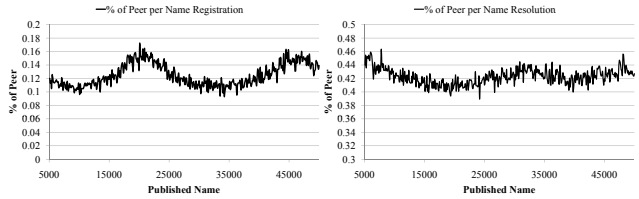
we start with a network of few ($\sim$500) peers and then grow the network to 20,000 peers. After that, the number of peers keeps varying between 17,500 to 22,500. Names are published incrementally and statistical data is collected. We start with 5,000 names and gradually increase the number to 50,000. Among the published names, 40% are human friendly names (generated using a syllable based name generator), 30% are peer or group UUIDs, and the remaining 30% are SHA-1 hash based secure names. All reported measurements are averaged over 100 independent runs.

In Figure 9, the solid line represents the average number of name mappings stored per peer against the total number of published names. The vertical lines indicate the standard deviation from the average. Clearly the storage load scales linearly with the number of published names. A linear trend line fits quite well (R^2 = 0.9902) with the data. The upper dashed line represents the maximum number of name mapping tuples stored in a peer over all runs. The maximum count varies from the average due to the skewness in the hash functions used for generating bloom filters. But the amount of skewness is quite low. For 50,000 names the difference between the average and maximum count is only around 25. So by using error correcting code based name mapping distribution we have achieved a namespace partitioning that is quite close to uniform. With an average network size of 20,000 peers and 50,000 published names there should have been only 2.5 names per peer. But in our simulation each peer has 50 names. This reason behind this can explained as follows: each peer in Plexus has one replica as explained in Section III-D and our list decoding algorithm outputs 10$\sim$12 codewords on average. So each name mapping gets stored in 20 peers which causes the name mapping count to be around 50 in our simulations. While this scheme increases the storage load, it provides robustness against peer churns. Performance of our system during high rate of peer churn is studied in the next section.

Figure 10(a) and 10(b) show the average hop count (along with the standard deviation) per name registration and resolution respectively against the total number of published names. Hop count for name resolution is higher than that of name registration for the following reasons: (i) name resolution queries include two resolution steps and (ii) spans more peers that registration. Hop counts for both name registration and resolution are quite high in our experiments due to a lacking in our implementation of the list-decoding algorithm. In Plexus [3], [4] extended Golay codes were used for routing where the number of maximum hops is bounded by 6. But unfortunately

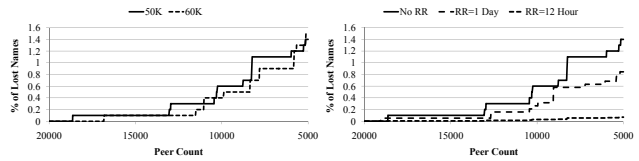


(a) Name registration hop count (b) Name resolution hop count



(c) % of peer per registration (d) % of peer per resolution

Fig. 10. Name registration and resolution hop count



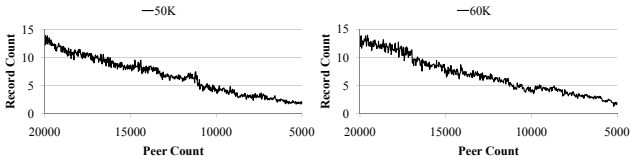
(a) (b)

Fig. 11. Percentage of lost names

there is no list decoding algorithm for the Golay codes. So we had to use the Reed-Muller codes in all our experiments. There are many theoretical papers on list decoding of Reed-Muller codes but there is still no implementation available. In our experiments we have used a brute-force implementation of the list-decoding algorithm which caused the high number of hop counts for both name registration and resolution. We are working on a more efficient implementation which will improve our results. Theoretically, if we keep the decoding radius within the Johnson Bound [31] (which will be the case in our approach) the maximum number of hops will be bounded by 30 and the actual average hop count per name registration and resolution will be much lower (in our current implementation hop count is bounded by 250). As expected these metrics are not dependent on the number of published names. They depend solely on the number of alive peers in the network. Figure 10(c) and 10(d) show the percentage of total peers accessed per name registration and name resolution respectively. Accessing a peer and its replica are considered as separate events in these experiments. Surprisingly enough, at most $\sim$0.17% and $\sim$0.46% peers are accessed per name registration and name resolution respectively.

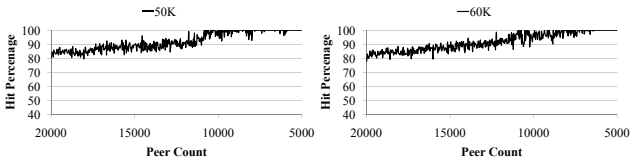
C. Fault Tolerance

For testing the robustness of our naming scheme against peer failure we start with a network of 20,000 peers, publish names and then gradually decrease the network size to 5,000. We measured the percentage of lost names, average number of records found per name resolution, and hit percentage. We have performed these experiments without name mapping republishing by group leaders. Evaluation of the effect of name



(a) Record count for 50K names (b) Record count for 60K names

Fig. 12. Record count per name resolution



(a) Hit percentage for 50K names (b) Hit percentage for 60K names

Fig. 13. Hit percentage per name resolution

republishing is examined in the next section. However we used the built-in replication mechanism of Plexus, where each peer has a single replica that stores the same information as the original peer and can be reached in just one hop. Figure 11(a) reports the percentage of name loss against peer count. We report a name as lost whenever a random peer fails to resolve that name. We report data for 50,000 and 60,000 names, which are shown as the solid and dotted lines respectively. By only utilizing the one node replication scheme of Plexus, our system is quite robust against peer failures. A network with 60,000 names lost only 1.5% names, where peer population dropped to one–forth of its initial size.

Figure 12(a) and 12(b) show the average number of name records found per name resolution for 50,000 and 60,000 names respectively. With a peer population of 20,000 each name resolution finds around 12$\sim$14 records. When the population falls to 5,000, still around 2 records are found on average per name resolution. With the decrease in peer population, amount of storage required per peer to store naming data increases. Peers begin to index larger portions of the namespace and as a result the number of peers storing the same name mapping begins to decrease. But with decreasing peer count, name resolution queries start to span more and more peers and discover almost all alive peers storing a particular name mapping. This is shown in Figure 13(a) and 13(b). Here hit percentage is the percentage of discovered peers among the peers actually storing a particular name mapping. As peer population decreases hit percentage increases from 80% to 100%. Since only 1.5% names are lost, our name resolution process can ensure almost 100% search completeness.

D. Effects of Refresh Rate

In our naming system group leaders periodically republish name mappings. The periodicity of this operation is termed as the refresh rate. The value of refresh rate provides a tradeoff between name availability and message overhead. A small value lead to an increase in name availability but at the cost of increased message overhead. On the other hand a large refresh rate decreases message overhead but increases the risk of name mapping losses. As was shown in Figure 11(a)

without any sort of republishing around 1.5% names are lost when a network is reduced from 20,000 to 5,000 peers. In Figure 11(b) we perform the same experiment with two different refresh rate values. When names are republished once per day the percentage of lost names drops to 0.8% which is close to half of the name loss percentage with no republishing ($\sim$1.5%). When refresh rate is equal to 12 Hours, the name loss percentage drops below 0.1%. Now lowering the value even further does not provide much improvement. Because in our experiments there is no peer rejoins. So, once a name mapping gets lost there is no one in the network who can republish it.

VIII. Conclusion

In this paper we have presented a scalable, fault-tolerant, distributed, and persistent naming scheme for structured P2P networks. Our naming scheme provides a flat namespace with no explicit structure. Users can have both human-friendly and secure names in our system. Through simulations we have shown that, error correcting code based namespace partitioning achieves uniform distribution of naming data across peers. At most 0.17% and 0.46% peers are accessed per name registration and name resolution respectively, which makes our system extremely scalable. The robustness of our system was also demonstrated through experiments. By using only the single peer replication scheme built in Plexus our system looses only $\sim$1.5% names in the face of 75% peer failure and without any name republishing. Simulation results also show that, with a moderate refresh rate value of 12 hours, the name loss percentage becomes negligible ($\sim$0.1%). The originality of our work lies in the application of a structured routing and indexing framework like Plexus for achieving a persistent naming scheme for P2P networks. Though our naming scheme is designed for a P2P web hosting platform, we belief that this approach can have significant impact on other existing P2P applications. The naming scheme also had application in the emerging filed of content centric networks. As a future work we plan to evaluate the performance of our naming system using real P2P overlay traces and more realistic testbeds such as Planetlab or ModelNet.

References

[1] Peer-to-peer, wikipedia, the free encyclopedia. [Online]. Available: http://en.wikipedia.org/wiki/Peer-to-peer

[2] Napster website. [Online]. Available: http://www.napster.com.

[3] R. Ahmed and R. Boutaba, "Distributed pattern matching: a key to flexible and efficient p2p search," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 73–83, 2007.

[4] ——, "Plexus: a scalable peer-to-peer protocol enabling efficient subset search," *IEEE/ACM Trans. Netw.*, vol. 17, pp. 130–143, February 2009.

[5] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, July 1970.

[6] Universally unique identifier, wikipedia, the free encyclopedia. [Online]. Available: http://en.wikipedia.org/wiki/Universally_unique_identifier

[7] O. B. Karimi, S. Yousefi, M. Fathy, and M. Mazoochi, "Availability in peer to peer management networks," in *LNCS: Challenges for Next Generation Network Operations and Service Management*. Springer, Oct. 2008, vol. 5297/2008, pp. 552–555.

[8] T. Schwarz, Q. Xin, and E. Miller, "Availability in global peer-to-peer storage systems," in *Proceedings of the 2nd IPTPS*, July 2004.

[9] K. Kim, "Time-related replication for p2p storage system," in *Proceedings of International Conference on Networking*, Apr. 2008, pp. 351–356.

[10] J. Sacha, J. Dowling, R. Cunningham, and R. Meier, "Discovery of stable peers in a self-organising peer-to-peer gradient topology," in *Proceedings of the 6th IFIP Int. Conference on Distributed Applications and Interoperable*, 2006.

[11] S. Blond, F. Fessant, and E. Merrer, "Finding good partners in availability-aware p2p networks," in *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems(SSS)*, 2009.

[12] N. Shahriar, M. Sharmin, R. Ahmed, and R. Boutaba, "Diurnal availability for peer-to-peer systems," *CoRR*, vol. abs/1101.4260, 2011.

[13] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," *Lecture Notes in Computer Science (LNCS)*, vol. 2009, pp. 46–66, 2001.

[14] M. Deshpande, A. Amit, M. Chang, N. Venkatasubramanian, and S. Mehrotra, "Flashback: A peer-to-peer web server for flash crowds," in *Proceedings of the 27th Int. Conference on Distributed Computing Systems*, 2007.

[15] H. B. Ribeiro, L. C. Lung, A. O. Santin, and N. L. Brisola, "Implementing a peer-to-peer web browser for publishing and searching web pages on internet," in *Proceedings of International Conference on Advanced Information Networking and Applications*, 2007, pp. 754–761.

[16] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6. Citeseer, 2003, pp. 68–72.

[17] N. Paskin, "Digital object identifier (DOI®) system," *Forthcoming publication in the third edition of the Encyclopedia of Library and Information Sciences (Taylor & Francis Group). Final reviewed and corrected text February*, 2009.

[18] 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 Comput. Commun. Rev.*, vol. 37, pp. 181–192, August 2007.

[19] C. Dannewitz, "NetInf: An Information-Centric Design for the Future Internet," in *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, 2009.

[20] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys Tutorials, IEEE*, vol. 7, no. 2, pp. 72 – 93, quarter 2005.

[21] S. Ajmani, D. E. Clarke, C.-H. Moh, and S. Richman, "Conchord: Cooperative sdsi certificate storage and name resolution," in *LNCS: Peer-to-Peer Systems*. Springer, Jan 2002, vol. 2429/2002, pp. 141–154.

[22] R. Cox, A. Muthitacharoen, and R. T. Morris, "Serving dns using a peer-to-peer lookup service," in *LNCS: Peer-to-Peer Systems*. Springer, Jan 2002, vol. 2429/2002, pp. 155–165.

[23] X. Li and C. G. Plaxton, "On name resolution in peer-to-peer networks," in *Proceedings of ACM international workshop on Principles of mobile computing*, 2002, pp. 82–89.

[24] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 73–86, 2002.

[25] M. Haque, R. Ahmed, and R. Boutaba, "Qpm: Phonetic aware p2p search," in *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, sept. 2009, pp. 131 –134.

[26] B. Wilcox-OHearn. (2003, September) Names: Decentralized, secure, human-meaningful: Choose two. [Online]. Available: http://zooko.com/distnames.html

[27] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, march 2010, pp. 1 –6.

[28] Challenge response authentication, wikipedia, the free encyclopedia. [Online]. Available: http://en.wikipedia.org/wiki/Challenge-response_authentication

[29] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 189–202.

[30] S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht (awarded best paper!)," in *USENIX Annual Technical Conference, General Track*. USENIX, 2004, pp. 127–140.

[31] S. Johnson, "A new upper bound for error-correcting codes," *IEEE Transactions on Information Theory*, vol. 8, pp. 203–207, 1962.