A Naming Scheme 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
[mfbari | m9haque | r5ahmed | rboutaba]@uwaterloo.ca
†Orange Labs, Lannion, France
bertrand2.mathieu@orange.com

***Abstract*—The peer–to–peer paradigm has great potential of providing the next generation Web hosting infrastructure. Profound advancements in P2P technology in the last decade have proven its capability to provide functionality similar to traditional client–server systems at a much larger scale with relatively lower cost. Existing centralized website hosting technology has a number of inherent deficiencies including scalability, single point of failure, administration overhead, hosting expenses *etc*. P2P Web hosting can effectively address these problems and hence open a new era for next generation Web hosting. However peer availability and content location are highly dynamic in a P2P network. This dynamism raises a number of research challenges related to naming, addressing, indexing, and searching in a P2P environment. In this paper we identify the practical requirements for devising a secure, persistent and human friendly naming scheme for P2P Web hosting and propose a novel naming scheme that satisfies all these requirements. We also present extensive simulation results validating the accuracy, scalability and fault-resilience of the proposed naming scheme.**

I. Introduction

Profound advancements in P2P technology in the last decade have proven its capability to provide functionality similar to traditional client–server systems at a much larger scale with relatively lower cost. Content distribution, on-demand video streaming and distributed computing are among the application domains where P2P technology has been successfully applied. However, the applicability of P2P systems for Web hosting is still a largely unexplored area.

Contemporary Web hosting technology is built around a client–server architecture, which has a number of inherent deficiencies including scalability, single point of failure, administration overhead, hosting expenses *etc*. If websites were distributed over a P2P network then there would be virtually no limitation on the number of users who could concurrently access a popular website. Flash crowd is still an inherent problem for the Web. This problem can be mitigated with a P2P architecture as many peers will be hosting a website and content caching will be done intrinsically by the system. Moreover, such a system will provide an uncensored platform that will promote freedom of speech, which can have significant societal and political impact. Content Distribution Networks (CDNs) attempt to overcome some of these problems by storing the same content at geographically distributed locations around the globe. However, the number of locations where a content can be stored is limited by the span of the CDN and this kind of service is only available for a fee. In turn, a P2P Web hosting platform can easily span the entire globe without deploying many expensive high–end servers. It can provide the same level of service by disseminating content over geographically distributed peers in the network at virtually no cost.

Various research challenges including naming, name resolution, indexing, searching, content placement, availability *etc*. need to be solved before P2P technology can be applied for server–less Web hosting. In this work we mostly focus on naming and name resolution. Later, we plan to incorporate works on the other related problems and build those solutions around the naming scheme presented in this paper. A suitable naming scheme for P2P web hosting should support names that are persistent and independent of spatial and temporal scope. Names should directly reference the content instead of depending on the peer that hosts it. After a name is attached to a content it should remain valid as long as the underlying content is available. Existing references should not be broken when a content is moved between domains or network locations. Security should be attached to the content it-self instead of depending on the peer hosting it. A user receiving a content should be able to ensure the authenticity and integrity of the content from the associated security information without requiring to trust the peer. This will enable any peer holding a valid copy (replica) of a content to serve as a valid source. A related research issue is providing persistent bookmarking in a P2P environment. Whenever a user visits a website he may want to bookmark it for later visits. Implementing bookmarking in the P2P environment is not trivial as websites can be replicated over multiple peers. The name resolution process should be able to identify content replicas hosted on multiple peers using the same name. The name registration and resolution system should be distributed, fault–tolerant, secured, and scalable. This mandates the use of sophisticated techniques for responsibility assignment, load balancing, data synchronization, authentication, authorization, and management of stale content.

Our earlier work in [1], proposed a naming scheme that supports persistent names which are either secured or human friendly but not both. Here we have extended that work in several ways. Our major contributions in this paper are the design of a scalable naming scheme that supports names that are persistent, secure, and human friendly all at the same time and the flexible embedding of organizational structure in a name to permit seamless content movement between domains. We have also proposed a detailed architectural design with streamlined overlay and system level APIs along with two generic performance enhancement mechanisms namely group information caching and message aggregation for reducing routing overhead.

The rest of the paper is organized as follows. Some preliminaries essential for a better understanding of this work are presented in Section II. Section III provides a brief description of our P2P web hosing architecture. Our naming scheme is presented in Section IV and principle system processes are described in Section V. Section VI discusses the rationale behind the major design decisions made in the system. Experimental results and evaluations are reported in Section VII. Section VIII presents related work and finally we conclude in Section IX.

II. Background

The name resolution mechanism presented in this work uses Plexus [2], [3]. Like other Distributed Hash Table (DHT) techniques Plexus supports efficient routing which scales logarithmically with network size. In addition, support for approximate matching is built into the Plexus routing mechanism, which is not easily achievable by other DHT techniques. To cope up with churn in P2P systems, Plexus supports multipath routing and efficient replica placement. Plexus delivers a high level of fault-resilience by using replication and redundant routing paths. Because of these advantages we have incorporated Plexus routing at the core of our naming system. In this section we present the basic concepts of Plexus indexing and routing mechanism.

Plexus Content Advertisement and Discovery: In Plexus keywords are mapped to Bloom filters [4] (or bit-vectors). A Hamming distance based technique derived from the theory of *Linear Covering Codes* (LCC) [5] 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.

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 within a pre–specified Hamming distance of an advertised pattern. As explained in Fig. 1(a), a linear covering code ($\mathcal{C}$) partitions the entire pattern space $\mathbb{F}_2^n$ into Hamming spheres, represented by hexagons. A codeword ($c_i \in \mathcal{C}$) is selected as the unique representative for all the patterns within its Hamming sphere. To facilitate approximate matching in Plexus, an advertisement pattern, say P; is mapped to all codewords, denoted by $\mathscr{A}(P)$, that are within a pre–specified Hamming distance, say s, from P. Mathematically $\mathscr{A}(P)$ can re represented as, $\mathscr{A}(P) = B_s(P) \cap \mathcal{C} = \{Y | Y \in \mathcal{C} \wedge d(Y,P) \leq s\}$, where $B_s(P)$ is the Hamming sphere of radius s centred at P and $d(Y,P) = |Y \oplus P|$ is the Hamming distance between Y and P. Similarly, a query pattern, say Q, is mapped to a set of codewords $\mathcal{Q}(Q) = B_t(Q) \cap \mathcal{C}$, for some pre–specified Hamming distance t. It is shown in [2] that there will be at–least one common codeword in $\mathscr{A}(P)$ and $\mathcal{Q}(Q)$, if $d(P,Q) \leq s + t - 2f$, where f is the covering radius of $\mathcal{C}$. In other words, by looking into the codewords in $\mathcal{Q}(Q)$, one should be able to find all advertised patterns within Hamming distance $s + t - 2f$ from Q.

In a complete Plexus network, each peer is responsible for one codeword. Thus the maximum number of peers allowed in a Plexus network is bounded by the number of codewords in the linear code used for deploying the network. In this work, we use the second-order Reed Muller code, $RM(2,6)$, which can support about four million (2^{22}) peers in the overlay.

Plexus Routing: Consider a (n,k,d) linear covering code $\mathcal{C}$ with generator matrix $G_{\mathcal{C}} = [g_1, g_2, \ldots, g_k]^T$. To route using this code, a peer responsible for codeword X, has to maintain links to $(k+1)$ peers with codewords $X_1, X_2, \ldots, X_{k+1}$, computed as follows:

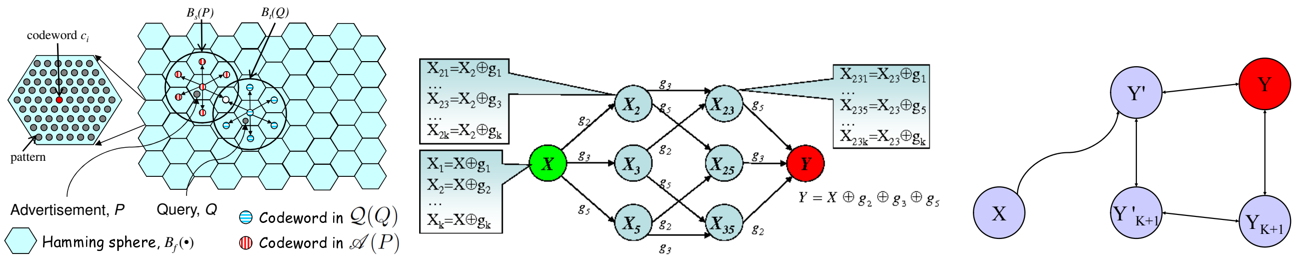
$$X_i = \begin{cases} X \oplus g_i & 1 \leq i \leq k \\ X \oplus g_1 \oplus g_2 \oplus \ldots \oplus g_k & i = k+1 \end{cases} \quad (1)$$

Now, the routing process in Plexus can be best explained by the example in Fig. 1(b), which shows the possible routes from peer X to peer $Y = g_2 \oplus g_3 \oplus g_3$ (any codeword Y can be generated from any other codeword X as follows: $Y = (X \oplus g_{i_1} \oplus g_{i_2} \oplus \ldots \oplus g_{i_t})$, where $g_{i_1}, g_{i_2}, \ldots g_{i_t} \in G$ and $\oplus$ is bitwise XOR operation). Peer X will forward the message to any of $X_2 (= X \oplus g_2)$, $X_3 (= X \oplus g_3)$ or $X_5 (= X \oplus g_5)$, which are one hop nearer to Y than X. If the message is forwarded to X_2 then X_2 can route the message to Y via $X_{23} (= X \oplus g_2 \oplus g_3)$ or $X_{25} (= X \oplus g_2 \oplus g_5)$. In such an overlay, it is possible to route a query from any source to any destination codeword in $\frac{k}{2}$ or fewer routing hops [2].

In Plexus protocol, a peer say Y replicates its indices to peer Y_{K+1}. In presence of failure a peer's replica can be reached in just 2 extra hops, which can be explained using the example of Fig. 1(c). Here peer X is attempting to route a query to peer Y, which has failed. When a neighbour (Y') of Y detects the failure, it forwards the query to its own replica Y'_{K+1} in one hop. Next peer Y'_{K+1} forward the query to peer Y's replica Y_{K+1} in one hop.

III. Architecture

Our P2P web hosting system is based on a super peer based architecture, where regular peers provide disk space for content storage and super peers index meta information. Figure 2 presents an overview of our three tier architecture. Tier–1 is the hosting network (*i.e.*, the Internet). Tier–2 is the group overlay, where peers are grouped together based on their diurnal connectivity patterns. These groups provide persistent storage for Web content. Tier–3 is the Plexus overlay, which provides the Plexus routing protocol and full–text indexing support of content and metadata. The Plexus overlay also provides registration and resolution services for three types of names: Peer ID (pID), Group ID (gID), and P2P Resource Locator (pRL) to reference peers, groups, and websites respectively. Both pIDs and gIDs are unique and remain unchanged across connectivity sessions. When a peer joins the system for the first time, the user provides his public key and the hash of his public key is used as the pID, which makes it unique depending on the fact that each user has a unique public key. For groups, randomly generated Universally Unique Identifiers



(a) Hamming distance based indexing (b) Possible routing paths between peer X and Y (c) Routing under failure

Fig. 1. Core concepts in Plexus

(UUIDs) are used as gIDs, which are generated and managed by the group overlay. Our Web hosting system provides the following interface to the end users:

- **Publish:** Used by end users to publish a website.
- **Search:** Provides keyword based search functionality.
- **Resolve:** Used to resolve a pRL (bookmark).
- **Refresh:** Resets expiry timers associated with a website.
- **Remove:** Removes a website from the network.

Expiry timers are associated with each index and when the timer expires the corresponding index is removed from the network. These timers are reset whenever the corresponding website is accessed by any peer.

A. Group Overlay

Regular peers with intermittent connectivity and relatively low connection speed collaborate with each other in small groups. These groups are formed based on the diurnal connectivity patterns of the peers. Peers having complementary or mutually exclusive (possibly with small overlap) uptime patterns are placed in the same group. As a result, at least one peer is always online for each group and can serve the websites published by the members of its group. This peer is called the group leader and if multiple peers in the same group are online at the same time then the peer having the smallest pID is chosen as the group leader. The responsibility of a group leader is two–fold: (i) ensure content availability while the publishing peer is offline and (ii) keep the published content and associated index uptodate by periodically refreshing them using the *refresh* function provided by the Plexus overlay. The mechanism for group formation, maintenance, and content replication between groups is beyond the scope of this paper. Our ongoing work on diurnal availability based group management can be found in [6]. The group overlay supports the following functions:

- **Store:** Used to store Web content in a group.
- **Find:** Retrieves Web content associated with a pRL.
- **Remove:** Takes a pRL as input and removes the associated Web content.

B. Plexus Overlay

Peers with relatively longer uptime and higher bandwidth are promoted to Super Peers. The super peers participate in group formation and website hosting similar to regular peers. In addition, super peers collaborate with each other using the Plexus routing protocol (Section II) to index website keywords and meta information for responding to future search queries.

The Plexus overlay maintains five kinds of indices, which are described below:

- **keyword $\rightarrow$ pRLs:** When publishing a website, a publisher may provide a list of keywords. The Plexus overlay maintains keyword to pRL mapping for each such keyword. Websites with the same keyword are grouped together, which results in each keyword mapping to a list of relevant pRLs.
- **pRL $\rightarrow$ Metadata:** Metadata associated with each website is indexed against its pRL. Metadata contains security related information that is used for content authentication and authorization.
- **pID $\rightarrow$ gID:** While joining the network a peer also joins an availability group. The group stores and provides access to a peer's content while it is offline. To keep track of which peer belongs to which group, each peer's group ID (gID) is indexed against its pID.
- **gID $\rightarrow$ gInfo:** Group information (gInfo) such as list of group members, storage capacity, currently alive peers, *etc.* is indexed against each group's gID. This information is used for selecting a group for new peers and finding currently alive peer (group leader) in a group.
- **pID $\rightarrow$ Uptime:** For adapting to the dynamism in peer connectivity patterns, uptime of each peer is tracked and indexed against its pID. This information is used to reorganize peers among the availability groups to ensure 24x7 availability.

The Plexus overlay uses these indices to provide various functionality required for Web hosting. A brief description of these functions is provided below:

- **Register:** Used to register a pRL with our system.
- **Resolve:** given a pRL finds out the IP:port pair of the currently alive peer hosting the Web content.
- **Advertise:** given a set of keywords and pRL, publishes keyword to pRL mappings in the Plexus overlay.
- **Query:** takes a set of keywords as input and returns a set of related pRLs.
- **Refresh:** used to refresh the expiry timer of the metadata associated with a pRL.
- **Remove:** used to remove an index from the overlay.

To summarize, 24x7 content availability is ensured by the group overlay and the Plexus overlay provides routing, indexing, name registration and resolution services. While a

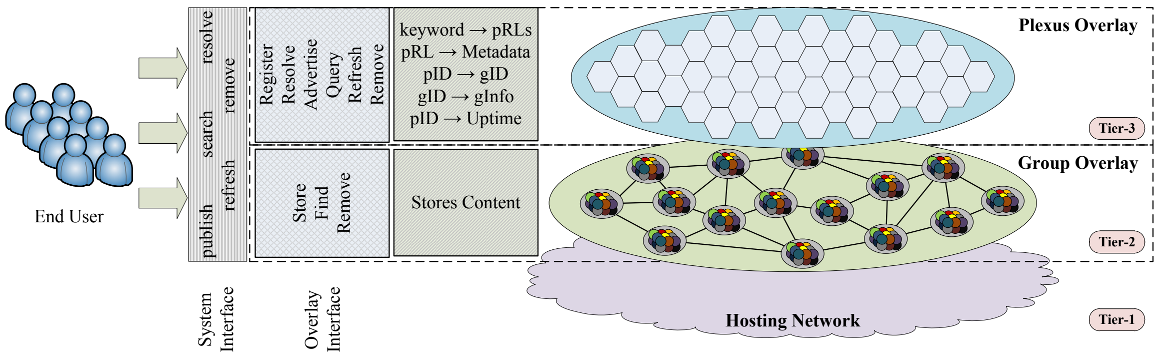


Fig. 2. System Architecture

peer is offline, its group leader takes on the responsibility to serve its published content and keep the data and associated index fresh in the network.

IV. Naming Scheme

Website names or pRLs are human friendly, persistent, and secured. These features are achieved by utilizing the metadata associated with each content (Figure 3). Content combined with metadata is called a "Content Package" and a requesting peer receives this package as a unit. A content package contains web data, pRL, metadata, and content signature as shown in Fig. 3. The pRL is composed of two parts: the first part is the pID (hash of publisher's pubic key, P_{PK}); and the second part is the hierarchical and human friendly label assigned by the publisher. Three SHA–1 digests namely: Data Digest, Name Digest, and Meta Digest are created by hashing the website data, pRL, and metadata respectively. These digests are then XOR'd to create the content digest, which is then signed by the publisher's secret key (P_{SK}) to create the content signature. This is then used to verify the authenticity and integrity of the content. The rest of this section describes how various information stored in the content package is used to support human friendly, persistent, and secured pRLs.

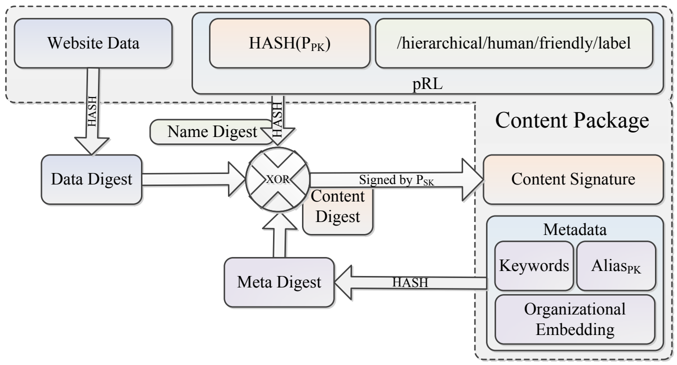


Fig. 3. Structure of a Content Package

Human Friendly Naming: As shown in Figure 3, the second part of a pRL is always human friendly. The first part is the hash of the publisher's public key and therefore is not human friendly. However, when our client application shows the pRL to an end user it replaces the public key hash with the $Alias_{PK}$ provided in the metadata to make the whole pRL human friendly. The end user also has the option to replace the publisher provided alias with an alias of his own choice. For example, the actual pRL for one of Bob's webpage is `9f9...9d8/how-to-assign-a-page-name`. Now Alice's client application replaces Bob's public key hash with the provided alias `Bob` (or any other alias configured by Alice), so that his webpage pRL reads `Bob/how-to-assign-a-page-name`. Our application saves the mapping between the hash and the alias in both ways so that Alice can type in the alias instead of the hash in the browser address bar. This aliasing scheme opens up opportunities for name forgery, which can be prevented by consulting a Public Key Infrastructure (PKI) to get the true identity of the publisher. Aliasing can also cause name conflicts between legitimate parties. In that case, Alice can determine which pRL she wants to browse by looking at the additional metadata (*e.g.*, keywords, organizational information *etc*.) associated with the content.

Persistent Naming: In our naming system pRLs are independent of the hosting peer. A site can be hosted by any peer in the network and still be identified by the same pRL. Furthermore, pRLs are not bounded by organizational boundaries. Organizational embedding of a pRL is encoded in the metadata (Figure 3), while the pRL itself is attached to the owner of the content. The organizational embedding field also contains public key chaining information to attach a pRL to its owner as well as to owner's organization. For example, Bob works at "Alice Inc." and has a website with pRL `9f9...9d8/how-to-assign-a-page-name`. This information is embedded in the metadata (Fig. 4). When Bob moves to "Trudy Inc.", he retains his original website pRL and only changes the organizational embedding field in the metadata as shown in Fig. 4.

Secured Naming: Both authenticity and integrity of a content can be verified by a receiver using the associated content signature under the assumption that an out–of–band Global PKI or Web of Trust (WoT) system exists which can be used to retrieve and verify public keys. Keeping the public key management system separate from the Web hosting system makes it possible for both systems to evolve without being affected by one another. Upon receiving a content, a receiver

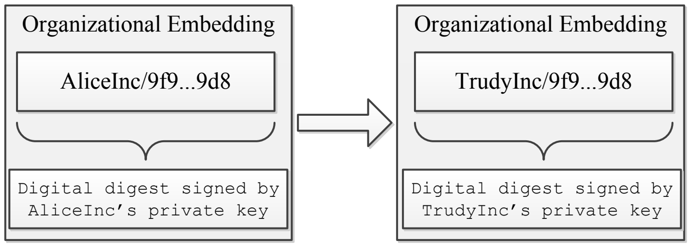

Organizational Embedding
AliceInc/9f9...9d8
Digital digest signed by AliceInc's private key
Organizational Embedding
TrudyInc/9f9...9d8
Digital digest signed by TrudyInc's private key

Fig. 4. Organizational embedding of a website

computes the content digest using the public key and content signature. He also computes the content digest from the pRL, Web data, and metadata as shown in Figure 3. If the two content digests match, the content is ensured to be both authentic and unaltered. The authenticity of the content is ensured by the public/secret key based cryptography system, while the integrity is ensured by the collision resistance properties of the hash function. However, malicious content modifications will be undetectable by this system if the publisher's secret key is compromised. Issues like secret key compromises and revocation mechanisms are out of the scope of this paper and hence not discussed here.

V. System Processes

Our Web hosting architecture provides five methods (*i.e.*, publish, search, resolve, refresh, and remove) for the end users to interact with the system. These methods use various functions provided by the overlays (Fig. 2) to perform their tasks. Interactions between overlays and operation sequences of these methods are shown in Fig. 5. Here, peer A, B, and C are super peers in the Plexus overlay. Peer B, D, and E are group leaders of three different availability groups in the group overlay. Peer B is participating in both Plexus and group overlay and peer X is the initiator in our examples. In-detail description of these methods is given below:

A. Publish

Suppose Peer X belongs to availability group G and wants to publish a website S with keywords r and s and metadata M.

- **Register:** *(Step a)* Peer X contacts a super peer A and provides the website name S_{pRL} and metadata M to register. *(Step b)* Super peer A finds out the super peer (B) responsible (Section II) for indexing S_{pRL} and instructs it to store the S_{pRL} to metadata mapping ($S_{pRL} \rightarrow M$) in its index store. *(Step c)* Successfully storing the mapping, super peer B returns a positive acknowledgement to peer X.
- **Advertise**: *(Step d)* Peer X contacts super peer A and provides the website name S_{pRL} and associated keywords r and s to advertise. *(Step e)* Super peer A finds out the super peers (B and C) who are responsible for indexing the keywords r and s respectively. Then it instructs them to store the $r \rightarrow S_{pRL}$ and $s \rightarrow S_{pRL}$ mappings respectively. *(Step f)* Upon successfully storing these mappings, super peer B and C return positive acknowledgements to X.
- **Store**: *(Step g)* Peer X requests a super peer A for the IP:port pair of group G's leader. *(Step h)* Super peer A requests super peer (C), who is responsible for indexing group G's group information (G_{gInfo}). *(Step i)* Receiving the request, super peer C looks up the IP:port pair of the group leader (here peer E) in its local index store and returns this information to peer X. *(Step j)* In this step, peer X directly transfers its Web content to peer E.

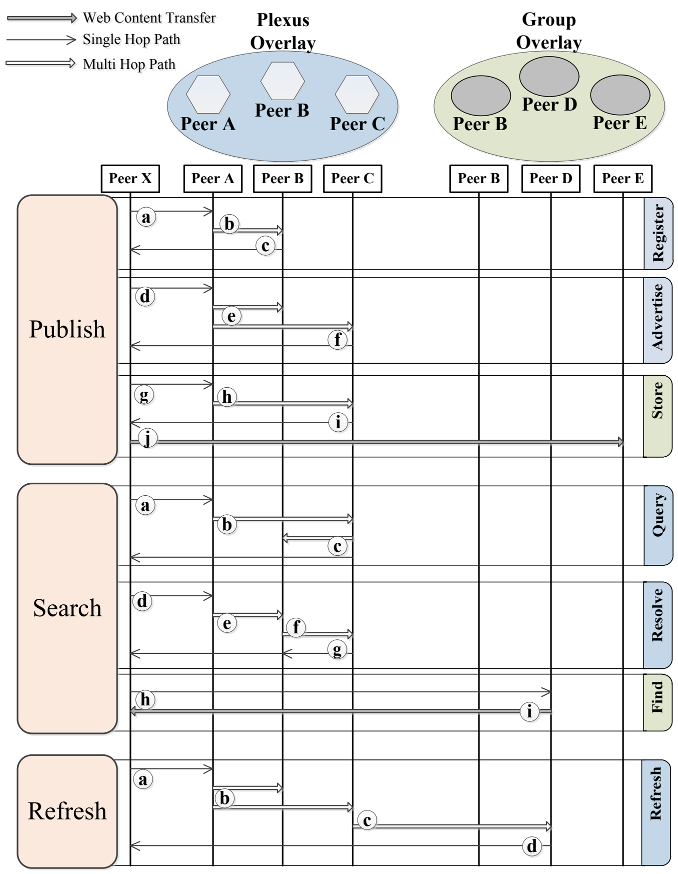


Fig. 5. System Processes

B. Search

Suppose Peer X is searching for a website with keyword r.

- **Query:** *(Step a)* Peer X contacts a super peer A and provides the keyword r it wants to search. *(Step b)* Super peer A finds the super peer (C) responsible for indexing keyword r and forwards the query request to C. *(Step c)* Upon receiving this request, super peer C constructs a list of relevant pRLs. Then it looks up the metadata associated with each pRL (shown as a single request to super peer B for simplicity) and returns a list of (pRL, Metadata) pairs to peer X.
- **Resolve:** *(Step d)* Peer X selects one of the pRLs from the query result and contacts super peer A to resolve the selected pRL (S_{pRL}). *(Step e)* Super peer A extracts the pID (S_{pID}) of the publishing peer P from S_{pRL} and finds out the super peer (B) who is responsible for

indexing S_{pID}. Then super peer A requests super peer B for the IP:port pair of peer P's group leader. *(Step f)* Upon receiving this request, super peer B looks up the group ID (P_{gID}) for peer P. Then B finds out the super peer (C) responsible for indexing P_{gID} and requests C for the IP:port pair of the group leader (here peer D). *(Step g)* Super peer C returns the IP:port pair of peer D to peer B, which then caches and forwards this information to peer X.
- **Find:** *(Step h)* Peer X requests D for desired content. *(Step i)* Peer D directly transfers the requested Web content to peer X.

C. Refresh

Suppose that Peer X belongs to availability group G and published a website S with keywords r and s and metadata M. Now, peer X wants to refresh its published indices.

(Step a) Peer X contacts a super peer A and provides it with the website name S_{pRL} it wants to refresh. *(Step b)* Super peer A finds the super peers (B and C) responsible for indexing all related information (*i.e.*, metadata, keywords, uptime, gID, and gInfo) to S_{pRL} and forwards the refresh request to them. *(Step c)* Upon receiving this request, super peer B and C refresh all expiry timers and super peer C (responsible for indexing gID) forwards the refresh request to group leader D who is storing the Web content. *(Step d)* After successfully completing the above operations B, C, and D return positive acknowledgement to peer X.

To avoid repetition the **Resolve** and **Refresh** methods are not discussed in detail. The system level **Resolve** method is just a wrapper for the *Resolve* method provided by the Plexus overlay. This method is used to resolve a given pRL (previously saved bookmarks) to the current group leader's IP:port pair. The **Remove** method follows the same steps as the **Refresh** method, but it instructs peers to remove indices instead of refreshing them.

VI. Design Rationale

Namespace Management: Three separate namespaces are managed in our system, namely pID, gID, and pRL. The pID namespace is managed by the Plexus overlay, which does not require to perform any explicit checks to ensure pID uniqueness. A pID is inherently unique as it is generated by hashing its publisher's public key. Group overlay has the responsibility to manage the gID namespace. Group IDs are basically UUIDs, which are generated by the group leaders when a new group is formed. Finally, the pRL namespace is managed in a distributed manner by the super peers participating in Plexus routing. The first part of each pRL is the pID (which is unique) and the second part is the human friendly label, which is unique within a publisher's domain. So, the overall pRL is always unique and there is no need to perform any uniqueness checks during name registration.

Diurnal Connectivity Based Grouping: Contemporary P2P replication mechanisms attempt to ensure content availability by creating multiple replicas and distributing them over the network. All of these mechanisms involve a trade–off between content availability and efficient storage utilization. Diurnal connectivity based peer grouping provides a replication scheme where replicas are hosted by peers with disjoint uptime distributions. At any given time there will be only a predetermined number of replicas in the network. It enables us to maximize content availability while minimizing replication overhead. Our ongoing work in this regard can be found in [6].

Group Information Caching: Group information is cached for speeding up the name resolution process. In the *resolve* function (Fig. 5), when super peer B receives the response for the gID resolution operation from super peer C (Step **g**) it caches the IP:port pair of the group leader against the corresponding pID. So that subsequent resolution requests for the same pID can be served from cache. This process reduces the number of hop counts for name resolution by half. Cache entries are associated with expiry timers and a cache entry is purged whenever the timer expires or the requesting peer fails to contact the group leader with the returned IP:port pair. In Fig. 5, when peer X fails to contact peer D it informs peer C and then peer C performs Step **f** again to obtain the IP:port of the current group leader and returns it to peer X.

Message Aggregation: Besides offering approximate matching and efficient routing, Plexus has the inherent capability of path aggregation for multicast routing. Here the path aggregation capability of Plexus is extended from single source multicasting to the multi–source case. This extension can be explained using an airport analogy. Each airport works as a hub. Transit passengers from different sources gather at an airport and depart on different outgoing flights matching their destinations. Similarly, each Plexus peer acts as a routing hub. Each Plexus message contains a number of target codewords. These codewords are used to route the messages to the appropriate receiver. As explained in Fig. 1(b), the number of alternate paths between a pair of source and target peers is combinatorially related to the distance between them. After receiving an incoming message a peer accumulates it in a message queue for a very small period of time instead of instantly forwarding it. Target codeword lists of the messages in the queue are combined to form a master target list. Then Plexus routing is applied to select the next hop neighbours and the targets in the master list are distributed over the selected neighbours. This approach significantly reduces the number of messages in the network.

Index Refresh: Expiry timers are associated with each index and whenever a timer expires the corresponding index is removed from the network. An expiry timer can be refreshed in two ways: (i) whenever any peer looks up that index and (ii) explicit refresh performed by the publishing peer itself or by the leader of the availability group where the associated Web content is stored. Publishing peer is primary responsible for this task. However the responsibility is delegated to the group leader when the publishing peer goes offline.

Security Model: Our security model revolves around the publisher. Authentication and authorization schemes are tied to the public/private key pair of the publisher. This model removes the dependency on a single host to perform operations on the content. The publisher can login from any host to publish new content or modify old ones. Security information is attached to the content itself, which makes it possible to serve the content from any host without introducing security holes. Our public key based naming system provides the split between content identifier and location, while placing the content at the core of the security model allows a user to place his trust directly on the content instead of the peer actually hosting it. A PKI based solution may suffer from performance issues due to computational and network overhead for signature verification. However, these problems can be avoided by using a Simple Distributed Security Infrastructure (SDSI) [7] or a Web of Trust (WoT) [8] based security model. We can also utilize other variations of PKI [9] that impose less resource requirements on the participating devices.

VII. Experimental Evaluation

A. Simulation Methodology

We have measured various performance metrics under diverse network conditions to evaluate the scalability, routing performance, and fault-tolerance capabilities of our naming system. Measurements have been taken under varied network size, number of published names, name resolution rate, peer session time, peer failure rate, and index refresh period. All experiments are done using a queuing model based cyclic simulator, where each peer is explicitly modelled using a message queue. In each cycle every peer gets his fair chance to process its message queue in parallel with other peers. A syllable based name generator is used to create a synthetic collection of human friendly names, which are used as pRLs in our experiments. The distribution of name resolution requests follows a Zipf distribution, which is frequently used to model distribution of content popularity in the Internet [10].

B. Measurement Matrix

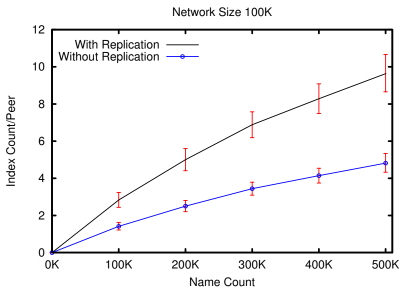
To evaluate the scalability of our naming system three performance metrics are measured: (i) number of indices stored per peer, (ii) effect of group information caching and message aggregation, and (iii) average routing hop required per name registration and resolution. The first measure shows us the uniformity of storage load distribution over the peers. The second and third measures highlight the performance of Plexus routing and effectiveness of the proposed optimization schemes: group information caching and message aggregation. The fault tolerance of our naming system is measured under two failure models: (i) peer churn, and (ii) peer failure. We have measured the average number of routing hops and percentage of successful name resolutions under both failure models.

C. Scalability

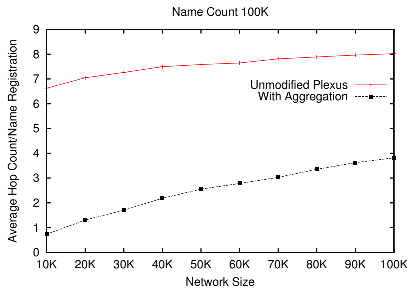
1) Index Load: Fig. 6(a) shows the average number (with 99% confidence intervals) of pRL indices stored per peer with and without replication. For this experiment, network size is kept fixed at 100K and number of published names is increased from zero to 500K. It is evident from the figure that the average number of indices per peer varies linearly with the number of published names and the distribution is uniform. The error bars also confirm the uniform distribution of indices over the peer. For example, with 400K names the average index count is approximately 8 and we can say with 99% confidence that it varies by at most 1. Replication doubles the storage load per peer as each index gets stored in two peers but this scheme provides fault–tolerance against peer churn and failure.

2) Message Aggregation: In this experiment we measure the impact of group information caching and message aggregation on name resolution. Fig. 6(d) shows the average hop count per name resolution where the network size is kept fixed at 100K peers and the number of name resolutions is varied from 10K to 100K. Hop counts for four different cases are shown: (i) unmodified Plexus, (ii) Plexus with caching, (iii) Plexus with message aggregation, and (iv) Plexus with both message aggregation and caching. Caching (Section VI) reduces hop count to almost half, as group information can be served from cache instead of performing a Plexus lookup. The hop counts for unmodified Plexus and Plexus with caching do not vary with number of names, because the number of routing hops depends only on network size and not on the number of published names. As explained in Section VI, with message aggregation average hop count drops significantly and keeps dropping with increasing name resolution rate. As name resolution rate increases so does the amount of accumulated messages in a peer's queue. A peer can apply aggregation on a larger population of messages, which eventually results in a reduction in hop count. Combining aggregation with caching further decreases resolution hop count. However the reduction is not significant because aggregation packs together lookup messages for both group ID and group leader, which get processed in parallel.

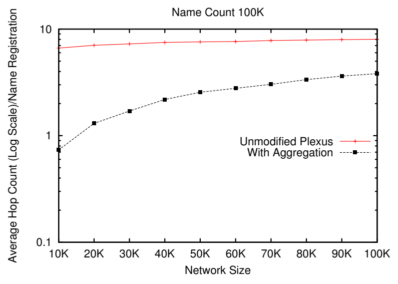
3) Routing Hop: Fig. 6(b) and Fig. 6(c) show the average number of hops required per name registration in normal and log scale, respectively. Here the network size is increased from 10K to 100K while publishing 100K names. From Fig. 6(b) we can conclude that average hop count for unmodified Plexus does not increase significantly with increasing network size. In fact it increases logarithmically with network size as explained in Section II and shown in Fig. 6(c). The curves with caching are omitted for name registration as each name is registered exactly once and caching has no impact. With message aggregation average hop count is significantly reduced as explained before but it seems to increase sublinearly with increasing network size. Hop count for name resolution is almost twice the hop count of name registration, as name resolution involves two lookup steps: (i) peer ID to group ID and (ii) group ID to group leader's IP:port pair. On the contrary name registration requires to route only one message to store the peer ID to group ID mapping at the target peer. Fig. 6(e) and Fig. 6(f) show the average number of hops required per name resolution in normal and log scale, respectively. Here the network size is increased from 10K to 100K while publishing 100K names. Hop counts for four different cases are shown: (i) unmodified Plexus, (ii) Plexus with caching, (iii) Plexus with message aggregation, and (iv) Plexus with both message aggregation and caching. Hop counts for unmodified Plexus and Plexus with caching increase logarithmically with network size as shown in Fig. 6(f). Message aggregation significantly



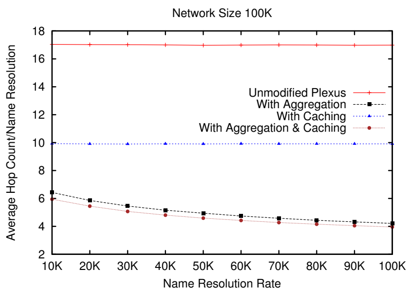
(a) Index count per peer



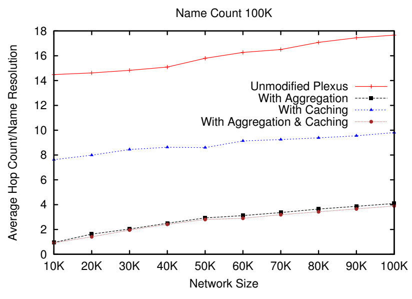
(b) Name registration hop count



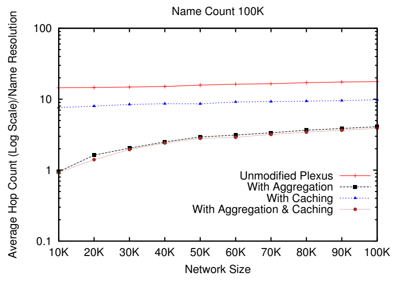
(c) Name registration hop count



(d) Name resolution hop count



(e) Name resolution hop count



(f) Name resolution hop count

Fig. 6. Index load and hop count for name registration and resolution

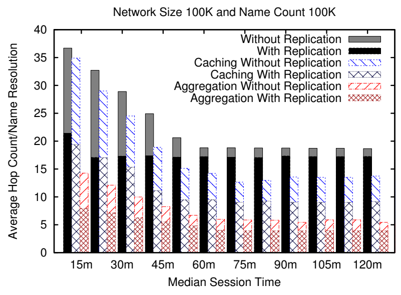
reduces the name resolution hop count while showing similar characteristics as the name registration case with increasing network size. With both message aggregation and caching, hop count is reduced but not as significantly as before.

D. Fault Tolerance

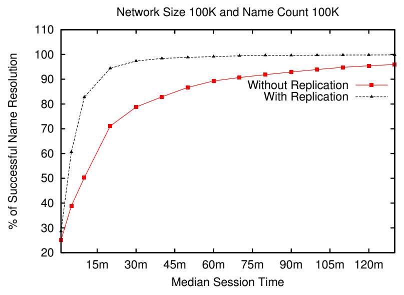
To measure the fault tolerance of our naming system experiments are done under two failure models: (i) peer churn, and (ii) peer failure. Under the peer churn model, peer session times (time between a join and leave event) are modelled according to Weibull distribution and peers join and leave events are timed according to a Poisson process [11]. Median session times between 15 minutes to 120 minutes are considered for performance evaluation following real network measurement data presented in [11], [12]. In the peer failure model, peers go offline uniformly at random locations in the network without any peer joins. During these experiments indices are not refreshed to assess the impact of multipath route selection mechanism of Plexus. All experiments are performed with and without replication to evaluate the fault–resilience gained by using a replica. Effect of refresh rates on percentage of successful name resolutions is also measured along with its messaging overhead.

1) Performance Under Churn: Fig. 7(a) shows the average hop count per name resolution against median session time, where session time represents the time between a peer's joining and leaving the network. We have shown the average hop counts for six different cases (as listed in the figure) while varying the median session time from 15 to 120 minutes. As median session time increases the network becomes more and more stable and as a result the average hop count decreases. It is clear from the figure that the network reaches steady state when median session time is 60 minutes. For median session time less than 30 minutes average hop count is high because of two reasons: reduced number of alternate routing paths and many hops being spent in locating lost names. The group information caching approach is mostly affected by small median session time as cached information becomes quickly unavailable in the network. The impact of caching keeps decreasing with reduced median session time. The hop count for unmodified Plexus and Plexus with message aggregation is increased by approximately 4 hops even with median session time of 15 minutes, which can be explained from Fig. 1(c). Plexus can route to the replica of a peer in 2 additional hops. As name resolution involves two Plexus lookups the average hop count is increased by at most 4. Both caching and aggregation improves average hop count but the impact of aggregation is most significant. The impact of replication is evident from Fig. 7(a). Replication significantly reduces the average hop count in all cases (as explained in Section II). Even without replication both caching and message aggregation improves average hop counts. However message aggregation achieves significantly better performance than caching. Fig. 7(b) shows the percentage of successful name lookups under churn with and without replica. Without replication, resolution success rate never reaches 100% even with a median session time of 120 minutes. However with replication, the success rate reaches very close to 99% with a median session time around 30 minutes and almost 100% with a median session time of 40∼45 minutes. We obtain optimal performance when replication is used and median session time is around 60 minutes.

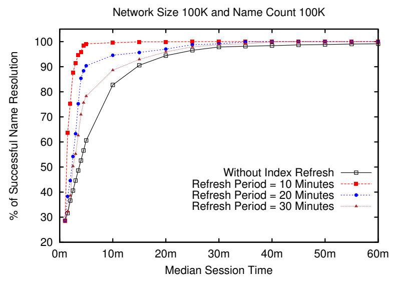
2) Performance Under Failure: Fig. 7(d) shows the average hop count for six different cases (as listed in the figure) per



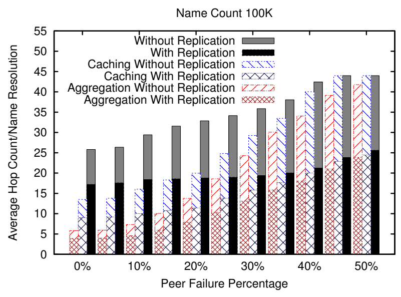
(a) Avg. hop count/resolution (churn)



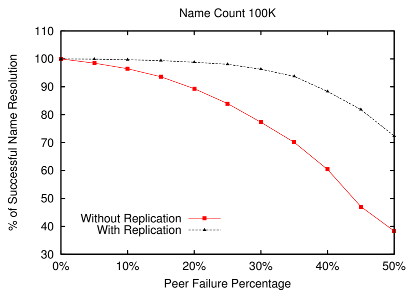
(b) % of successful resolution (churn)



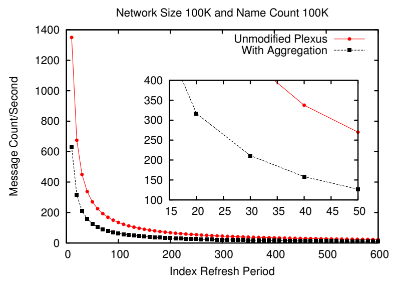
(c) % of successful resolution



(d) Avg. hop count/resolution (failure)



(e) % of successful resolution (failure)



(f) Message overhead for index refresh

Fig. 7. Name resolution under peer churn and failure

name resolution against percentage of failed peers. We have failed zero to 50% peers in this experiment. As more peers fail average hop count keeps increasing as number of alternate routing paths is reduced and many hops are spent in locating lost names. However the network can perform satisfactorily even in presence of 30% peer failure. Success rate remains close to 100% (Fig. 7(e)) though at the expense of increased routing overhead. The impact of caching and message aggregation is quite significant up to 30% peer failures, while message aggregation outperforms caching in all cases. The impact of replication is best reflected in presence of failures (Fig. 7(d)). Replication significantly reduces the average hop count in all cases. With replication average hop count is almost constant up to 40% failures because the replica of a failed node can be reached in only two extra hops as explained in Fig. 1(c). Percentage of successful name resolution under continuous peer failure is shown in Fig. 7(e). Using replication the success rate falls slowly and even with 40% peer failures around 90% success is achieved. However without replication, success rate falls sharply dropping below 90% even with just 20% failures.

3) Effect of Index Refresh: As explained in Section VI, the index refresh operation is used to replenish expired indices from the network. This feature provides a trade–off between name availability and message overhead for performing refresh. A small refresh period increases both name availability and message overhead. Whereas a large value decreases message overhead but increases percentage of lost names. Fig. 7(c) and Fig. 7(f) show the effect of refresh rate on percentage of successful name resolution and incurred message overhead, respectively. In these experiments both the network size and number of published names are kept fixed at 100K. From Fig. 7(c) we can see that with a refresh period of 10 minutes percentage of successful name resolution reaches very close to 100% very quickly. Even with a median session time of just 5 minutes our name resolution system in able to obtain approximately 100% success rate. For refresh periods of 20 and 30 minutes the rate of improvement is relatively slower and they achieve close to 100% success rate around median session time of 30 and 40 minutes, respectively. Fig. 7(f) shows the number of messages that need to be exchanged over the whole network per second while varying refresh period from 10 to 600 minutes. It is evident from the figure that message load drops sharply with increasing refresh period up to 100 minutes and after that it starts to level–off. It is also clear from the figure that message aggregation almost halves the message overhead. From the zoomed portion of the figure we can see that if we move from refresh period of 10 minutes to 20 minutes, the message overhead drops from ∼650 to ∼300 messages per second. A refresh period of 20 minutes also provides satisfactory performance is terms of percentage of successful name resolutions. This seems to be a good operating point for our system under current conditions.

VIII. Related Work

Existing P2P systems (*e.g.*, BitTorrent [13], Kazaa [14] *etc.*) either use randomly generated IDs or host IP, Port pairs to reference peers. In most of these systems, peers are considered to be memoryless, *i.e.*, peers are not assumed to retain any knowledge about the overlay network from their previous sessions. Therefore, the requirement for assigning persistent names to peers is not important. But for P2P web hosting we have to ensure persistent names for both peers and websites. None of the existing P2P name assignment schemes can ensure persistent IDs across connectivity sessions [15], whereas our naming scheme ensures persistent and secured IDs for both content and peers. Several research works, including [7], [16], focus on implementing DNS lookup using P2P systems. All of these works use DHT-based techniques for P2P lookup for conventional websites hosted on the Internet. However, P2P web hosting requires to resolve a name to the address of a live peer currently hosting the website in the network.

Hash based naming is also used by BitTorrent to identify its content chunks. Our architecture however uses public key hashes as name prefixes and ensures content and index availability by utilizing diurnal uptime patterns of peers instead of using trackers. Besides, BitTorrent is not suitable for hosting regular websites as Web objects are quite small compared to large media files typically shared by BitTorrent. It is efficient in handling files that require minutes or hours rather than seconds. BitTorrent uses a significant amount of time to try out and compare different connections which introduces significant latency. While this latency is negligible for large media files, it may overwhelm the download time required for small images embedded in webpages.

Web browsing over P2P networks is investigated by FreeNet [17], FlashBack [18], and Web2Peer [19] 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 webpages and Internet Web servers still serve as the source of webpages cached in the P2P system. The challenge of achieving persistent names for content shared in a P2P network has not been addressed so far.

Digital Object Identifier (DOI) [20] provides persistent and unique DOI links for electronic documents on the Internet. However the architecture of DOI is centralized and requires human intervention for maintaining the DOI links up–to–date. Data Oriented Network Architecture (DONA) [21] proposes to replace DNS names with flat, self–certifying names. Name resolution is done by the route–by–name paradigm using DNS like hierarchically organized *Resolution Handlers* (RHs). Network of Information (NetInf) [22] follows the same naming scheme as DONA but instead of replacing DNS, it resolves NetInf names to DNS names (URLs). Though these schemes provide security against name–thefts, names are no longer human-friendly. The naming schemes proposed by DONA and NetInf can not be adapted for P2P web hosting 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 overhead, (ii) DONA proposes to replace DNS whereas NetInf proposes to build a persistent and secured naming layer on–top of existing DNS. Neither of these schemes is suitable for resolving a name to the address of a live peer currently hosting the website in the network. While we borrowed the idea of using self–certifying names from DONA, our naming scheme is different from both DONA and NetInf in the following ways: (i) our naming scheme supports human–friendly names, (ii) name resolution is performed by peers within the same network without depending on a secondary resolution infrastructure (*e.g.*, DNS), and (iii) name persistence is maintained by peers with intermittent uptimes without using any stable servers (*e.g.*, RHs in DONA). A secured naming scheme for Information Centric Networking (ICN) has been proposed in [23]. This scheme forwards each name to a designated authority for verifying name to IP mapping. This type of name to authority relationship is impossible to maintain in a P2P network due to content and peer dynamism.

IX. Conclusion

In this paper we have presented a secure, persistent, and human friendly naming scheme for P2P Web hosting. Our proposed naming system is distributed, efficient, scalable, and fault–tolerant. Through simulations we have shown that our index publishing scheme achieves close to uniform load distribution. We have also introduced two optimizations to the original Plexus protocol: caching and message aggregation to reduce the number of routing hops. We have also investigated the fault–resilience of our naming system under two failure models and showed that our naming system performs reasonably well under realistic operational conditions. While we address the issues related to naming and name resolution here, successful realization of a P2P Web hosting infrastructure requires a number of research problems to be addressed including efficient full-text indexing, intelligent searching, distributed ranking, security, privacy, ensuring availability, *etc*. In our future research we intend to address these problems. We have developed our naming system based on a content centric security model, which intrinsically supports verification of publisher authenticity and content integrity. A very interesting research direction can be the exploration of the interplay between content based security and existing P2P security mechanisms in the literature. Though our naming scheme is designed for P2P Web hosting, we believe that it can be extended for naming any persistent service over P2P networks.

References

[1] M. F. Bari, M. R. Haque, R. Ahmed, R. Boutaba, and B. Mathieu, "Persistent Naming for P2P Web Hosting," in *IEEE International Conference on Peer-to-Peer Computing (P2P)*, August 2011, pp. 270–279. [Online]. Available: http://dx.doi.org/10.1109/P2P.2011.6038745

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

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

[5] G. Cohen, *Covering codes*. North Holland, 1997, vol. 54.

[6] N. Shahriar, M. Sharmin, R. Ahmed, and R. Boutaba, "Diurnal Availability for Peer-to-Peer Systems," *CoRR*, vol. abs/1101.4260, 2011.

[7] 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.

[8] Explanation of the web of trust of PGP. [Online]. Available: http://www.rubin.ch/pgp/weboftrust.en.html

[9] Y. Lee, J. Lee, and J. Song, "Design and implementation of wireless PKI technology suitable for mobile phone in mobile-commerce," *Comput. Commun.*, vol. 30, no. 4, pp. 893–903, Feb. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2006.10.014

[10] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An analysis of Internet content delivery systems," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 315–327, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844158

[11] S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *USENIX Annual Technical Conference, General Track*. USENIX, 2004, pp. 127–140.

[12] 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.

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

[14] K. Ross, J. Liang, and R. Kumar, "Understanding kazaa," Technical report, Polytechnic University Brooklyn, Tech. Rep., 2004.

[15] 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.

[16] 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.

[17] 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.

[18] M. Deshpande, A. Amit, M. Chang, N. Venkatasubramanian, and S. Mehrotra.

[19] H. B. Ribeiro, L. C. Lung, A. O. Santin, and N. L. Brisola.

[20] N. Paskin, "Digital object identifier (DOI®) system," *Encyclopedia of library and information sciences*, 2008.

[21] 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.

[22] 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.

[23] W. Wong and P. Nikander, "Secure naming in information-centric networks," in *Proceedings of the Re-Architecting the Internet Workshop*, ser. ReARCH '10. New York, NY, USA: ACM, 2010, pp. 12:1–12:6. [Online]. Available: http://doi.acm.org/10.1145/1921233.1921248