Handling Identity in Peer-to-Peer Systems*

Manfred Hauswirth, Anwitaman Datta, Karl Aberer
Distributed Information Systems Laboratory
École Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
{Manfred.Hauswirth, Anwitaman.Datta, Karl.Aberer}@epfl.ch

Abstract

As IP addresses have become a scarce resource most computers on the Internet no longer have permanent addresses. For client computers this is usually not a big problem but with the advent of P2P systems, where every computer acts both as a client and as a server, this has become increasingly problematic. In advanced P2P systems ad-hoc connections to peers have to be established, which can only be done if the receiving peer has a permanent IP address. In this paper we propose an approach for a completely decentralized, self-maintaining, light-weight, and sufficiently secure peer identification service that allows us to consistently map unique peer identifications onto dynamic IP addresses in environments with low online probability of the peers constituting the service. For security we apply a combination of PGP-like public key distribution and a quorum-based query scheme. We describe the algorithm as implemented in our P-Grid P2P lookup system and give a detailed analytical performance analysis demonstrating the efficiency and robustness of our approach. Our approach also can easily be adapted to other application domains, i.e., be used for other name services, because we do not impose any constraints on the type of mappings.

Keywords: Dynamic IP addresses, P2P systems, identity, name service, self-organization, self-maintenance

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

1 Introduction

In any distributed system where participants establish connections among each other identification mechanisms are prerequisite to consistently determine the endpoints of communication. For example, a virtually centralized system such as the WWW requires uniqueness of IP addresses (or hostnames) to operate in the expected way, meaning that the hostname part of a URL always points to the same host. Uniqueness of IP addresses is guaranteed by the network infrastructure and ISPs. Illegal use of an IP address can easily be detected and prevented. Also uniqueness of names can be guaranteed efficiently by DNS and owners of domains/hosts can be identified. However, this is not sufficient for e-commerce, so additional identification via certificates and security by encryption (SSL) were added such that the user can be sure to communicate with the party he/she intended to. The required security level in the identification process is determined by the requirements of the applications (simple web site, e-commerce, etc.), the architecture of the system (centralized, decentralized, etc.) and the communication patterns used (static connection with known parties, ad-hoc connections).

In systems with higher degrees of decentralization, mobility, and ad-hoc connections such as P2P systems, ad-hoc networks, or systems that support mobility, the situation is not as well-covered because the dynamic properties of these systems are not addressed by current Internet technologies which focus on "static" systems. In advanced P2P systems ad-hoc connections to peers have to be established which is only possible if the receiving peer has a permanent IP address. However, since IP addresses have become a scarce resource, most computers have dynamic, i.e., changing, IP addresses. Thus a new means for identifying peers and consistently mapping these identifications onto IP addresses are required.

In this paper we present a completely decentralized, self-maintaining, and sufficiently secure peer identification service that facilitates the consistent mapping of globally unique peer identifications onto dynamic IP addresses. Peers generate universally unique identifications locally and store them along with their public key, their current IP address and a cryptographic signature in our P-Grid P2P lookup system [2] on a certain number of peers. Mapping an id onto an IP address then is done by querying P-Grid using the receiver's id as the key. If a certain quorum of identical answers is returned the mapping is considered trustworthy and the peer is contacted. If contacting the peer fails then the peer is either offline or has changed its IP address (this cannot be distinguished). The requester can now either assume that the peer is offline and give up or, in the latter case, submit a new query to determine the new IP address. If contacting the peer succeeds in either case, its public

key is used to determine whether the contacted peer really is the one identified by the mapping or whether a different peer reuses the address or a malicious peer tries an impersonation attack. The security concept of our approach is a combination of PGP-like public key distribution and a quorum-based query scheme.

We describe the algorithm as implemented in our P-Grid P2P system and give a detailed analytical performance analysis demonstrating the efficiency and robustness of our approach. The algorithm is designed to operate in environments with low online probabilities of the peers constituting the identification service. We apply the our identification service in P-Grid itself to address the problem of changing IP addresses. Our approach also can easily be adapted to other application domains, i.e., be used for other name services, because we do not impose any constraints on the type of mappings.

The paper is organized as follows: We start with a short description of P-Grid whose requirements were the original motivation for this work and briefly discuss dynamic IP addresses, some scenarios, and some security implications in Section 2. Section 3 then describes our algorithmic approach to address the identification problem which is then illustrated by a detailed example in Section 4. Section 5 presents the pseudo-code of the required algorithms that will be analysed in Section 6. The results of the analysis will be discussed in Section 7. In Section 8 we position our approach regarding related work and give our conclusions in Section 9.

2 Problem statement and motivation

To motivate our approach and show its practical background we briefly present our P-Grid P2P system in this section. Then we discuss dynamic IP addresses in more detail and give some scenarios in which identification is required.

2.1 P-Grid in a nutshell

P-Grid [2] is a peer-to-peer lookup system based on a virtual distributed search tree: Each peer only holds part of the overall tree, which comes into existence only through the cooperation of individual peers. Searching in P-Grid is efficient and fast even for unbalanced trees [1] ($O(\log(n))$, where n is the number of leaves). Unlike many other peer-to-peer systems P-Grid is a truly decentralized system which does not require central coordination or knowledge. It is based purely on randomized algorithms and interactions. Also we assume peers to fail frequently and be online with a very low probability. Figure 1 shows a simple P-Grid.

Every participating peer's position is determined by its path, that is, the binary

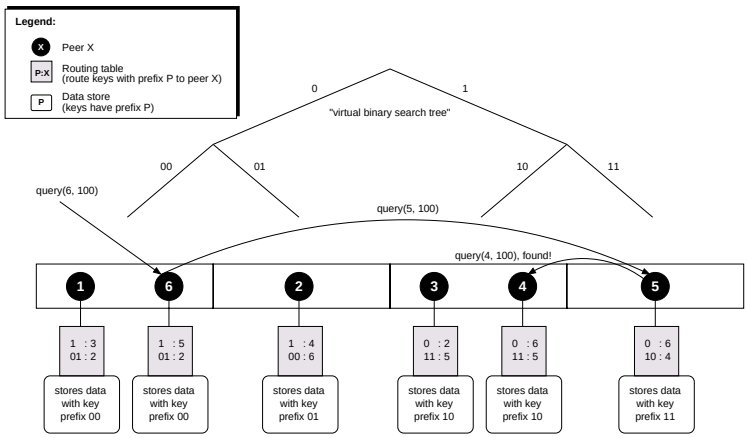


Fig. 1. Example P-Grid

bit string representing the subset of the tree's overall information that the peer is responsible for. For example, the path of Peer 4 in Figure 1 is 10, so it stores all data items whose keys begin with 10. For fault-tolerance multiple peers can be responsible for the same path, for example, Peer 1 and Peer 6. P-Grid's query routing approach is simple but efficient: For each bit in its path, a peer stores a reference to at least one other peer that is responsible for the other side of the binary tree at that level. Thus, if a peer receives a binary query string it cannot satisfy, it must forward the query to a peer that is "closer" to the result. In Figure 1, Peer 1 forwards queries starting with 1 to Peer 3, which is in Peer 1's routing table and whose path starts with 1. Peer 3 can either satisfy the query or forward it to another peer, depending on the next bits of the query. If Peer 1 gets a query starting with 0, and the next bit of the query is also 0, it is responsible for the query. If the next bit is 1, however, Peer 1 will check its routing table and forward the query to Peer 2, whose path starts with 01.

The P-Grid construction algorithm [2] guarantees that peer routing tables always provide at least one path from any peer receiving a request to one of the peers holding a replica so that any query can be satisfied regardless of the peer queried. Additionally it guarantees that a sufficient number of replicas exist for any path and that the peers representing a certain path also know their replicas. Thus the routing tables will hold also multiple references for each level which the routing algorithm selects randomly [2].

P-Grid supports updates of the stored, replicated data via a push/pull strategy with probabilistic guarantees [6] which allows us to store and change the mappings required for handling dynamic IP addresses in P-Grid itself. To improve efficiency each peer stores the mappings it knows in a cache which is consulted before issuing

a query. If a cache entry has become stale this can be detected by the signature mechanism we propose. Although storing the mappings inside P-Grid for use inside P-Grid itself looks like a typical hen-egg problem, we show in the following sections that our approach will work provided that no catastrophic failure occurs that would render the system dysfunctional anyway. We will show how we use the same mechanisms for implementing our identification approach and for "self-healing" the directory holding the identification mappings.

2.2 Dynamic IP addresses

The available IPv4 address space is fairly limited and fragmented inefficiently. Although approximately 2^{32} addresses are available, in reality this number is considerably lower for historical reasons. The address space is divided into subranges (class A, B, and C networks) that are assigned to one authority. Since most of the networks have already been assigned but by far not all of the IP addresses are used, a shortage of IP addresses has occured. Additionally IP addresses for special use (e.g., broadcast addresses) and multicast addresses further diminish the available addresses. IPv6 would solve the address shortage problem because it offers an address space of 2^{128}. The current address shortage was one of the driving forces to develop the Dynamic Host Configuration Protocol (DHCP) and Network Address Translation (NAT).

DHCP [7] was originally developed to automatically configure the networking software of workstations. Assuming that the stations are not online all the time this functionality can also be used to exploit a limited pool of IP addresses more efficiently. The simplified working pattern is that a DHCP server maintains available IP addresses. Typically upon boot-up a client sends out a DHCP broadcast message to find a DHCP server and requests an IP address and the DHCP server sends the assigned IP address possibly along with some additional information such as network mask, default gateway, and nameserver. The client may then use the IP address for a time period defined by the server (lease time). If the lease time expires the client has to send a new request and may either get a new lease time for the same address or a new IP address. Most DHCP servers can be configured to assign the same IP address for certain Ethernet addresses so that clients can have a permanent address. However, the standard case is that the IP address changes.

Network Address Translation [10] attacks the address problem more directly. The basic concept is that a NAT router maps non Internet-routable IP addresses[1]

[1] 10.0.0.0 – 10.255.255.255 (class A), 172.16.0.0 – 172.31.255.255 (class B), 192.168.0.0 – 192.168.255.255 (class C)

onto routable IP addresses back and forth. The most frequently used configuration is that the NAT router has an official IP address and all the computers in the local network have non-routable ones. The router then assigns a unique port to each of the non-routable addresses thus making it possible to route packets coming in from the Internet to the correct local address. This works nicely if the connections to the Internet are initiated by the local computers. However, it is not possible for a peer on the Internet to initiate a connection to another peer that is behind a NAT for the obvious reason that the addressee does not have a routable address so Internet routers just drop the received packets.

Many ISPs combine both technologies. In the following we focus on the problem of dynamic routable addresses. NAT is beyond the scope of this paper. NAT is a general problem of P2P systems because it only supports uni-directional connection establishment. This must and can be addressed but is subject to future work.

Besides providing a larger address IPv6 also offers a solution for the issue of mobility (i.e., dynamic addresses) which we address in our approach. However, IPv6 is not in place at the moment and it is rather unclear when it will be. Also MobileIP, an IPv4-based solution for mobility which would remedy many problems is not deployed yet (and possibly will never be).

Consequently, although enabling technologies exist but are not deployed, we have to address the issue of identity and dynamic IP addresses in P2P systems.

2.3 Scenarios

Unlike many popular peer-to-peer systems such as Gnutella [5] (constrained broadcast, breath-first) or Freenet [4] (depth-first), P-Grid has a much more "directed" search strategy to minimize efforts. Particularly it uses randomized, near optimal routing tables for storing its distributed access structure. The routing tables and the distributed index hold unique peer identifications. Ultimately, these peer identifiers have to be mapped onto IP addresses consistently. If all peers have static IP addresses this is no problem because the IP addresses (or the corresponding hostnames) can be used. However, this is not realistic in a real-world setting and motivated the approach presented in the following sections. Typically nodes have a temporary network address assigned by DHCP [7] or do not have a routable IP address at all in the case network address translation (NAT) is used. Therefore it is necessary to address to following problems:

1. How can universally unique identifiers be mapped onto physical addresses in a secure, decentralized, and efficient way?

2. With the possibility of changes of physical addresses a peer must be able to detect whether it is still talking to the same entity it intends to talk with. For P-Grid this translates into:

   (a) Peer p_1 goes offline and a different peer p_2 gets p_1's IP address. The other peers must be able to detect this change and react accordingly.

   (b) A peer goes online again with a new IP address. The other peers must be able to detect this, update their routing tables accordingly, and check identification before downloads of indexed data.

Gnutella is not affected by the first problem since peers actively announce their availability but at the cost of high bandwidth consumption because of its constrained broadcast approach. Freenet and any other system that uses ad-hoc connections suffers from the same problems of dynamic IP addresses as P-Grid.

However, any peer-to-peer system actually should address the second problem for security reasons. Otherwise it is very simple to de-facto shutdown a system by a variation of rather simple denial-of-service (DOS) attacks. For such a DOS attack a malicious node could impersonate another peer or provide wrong query hits or routing information thus putting high load on other peers or simply not route messages at all. These are only some security problems of current systems. A comprehensive discussion, however, is beyond the scope of this paper.

The following sections will discuss our approach and provide an analysis of its performance.

3 Approach

Each peer p is uniquely identified by a universally unique identifier (UUID) Id_p. This identifier is generated once at installation time by applying a cryptographicly secure hash function to the concatenated values of the current date and time, the current IP address $addr_p$ and a large random number. At bootstrap each peer p also generates a private/public key pair D_p/E_p once.

In P-Grid routing tables and the index hold only these identifiers. Each peer p additionally has a cache of mappings $(Id_i, addr_i, TS_i)$ (TS_i denotes a timestamp) that it already knows.

Then the algorithm for handling dynamic IP addresses works as follows (inserts and updates are done according to the algorithm presented in [6]):

Bootstrap

1. p generates Id_p, D_p/E_p.

2. Upon the first startup p determines its current IP address.[2]

3. p inserts the tuple $(Id_p, addr_p, E_p, TS_p, D_p(Id_p, addr_p, E_p, TS_p))$ into P-Grid using Id_p as the key (TS_p prevents replay attacks). Inserting in P-Grid means that the request is routed to a peer $R_i \in \Re_p$. $\Re_p$ is the set of replicas responsible for the binary path using Id_p as the key value ($path(Id_p)$). If Id_p already exists in the P-Grid (though this is very unlikely) p is notified. If so, p generates a new Id_p and repeats this step.

4. The previous step is repeated R_{min} times and p waits for confirmation messages from R_{min} distinct peers to prevent a malicious peer in $\Re_p$ from distributing false data to the other replicas in $\Re_p$.

5. As a result of the previous 2 steps the mapping will be physically stored at peers in $\Re_p$. Based on the randomized algorithms that P-Grid uses we can assume that the individual replicas $R_i \in \Re_p$ are independent and they collude or behave Byzantine only to a degree that can be handled by existing algorithms.

Peer startup

1. p starts up and checks whether its $addr_p$ has changed. If not the algorithm terminates. Otherwise the following steps are taken.

2. p sends an update message $(Id_p, addr_p, TS_p, D_p(Id_p, addr_p, TS_p))$ to the P-Grid, i.e., a new mapping and a signature for this mapping.

3. Upon receiving the update request the R_i check the signature by verifying that $(E_p(D_p(Id_p, addr_p, TS_p)).Id_p = Id_p$ (thus only p can update its mapping) and $TS_{R_i} < TS_p$ (to prevent replay attacks). If yes, the new mapping is stored, otherwise an error message is returned.

Operation phase

This phase denotes the standard operation of P-Grid, i.e., p is up and running, has registered an up-to-date mapping $(Id_p, addr_p, TS_p)$ and is ready to process queries and update requests.

[2]The IP address must be routable and reachable, i.e., not behind a firewall.

1. p receives a request Q from a peer q.

2. In case p can satisfy Q the result is returned to q. Otherwise p finds out which peers p_f to forward the query to according to P-Grid's routing strategy. Then it checks its routing table and retrieves $(Id_{p_f}, addr_{p_f}, E_{p_f}, TS_{p_f})$ which had been entered during the construction of P-Grid.

3. p generates a random number r, contacts p_f and sends $E_{p_f}(r)$. As an answer p_f must send $(D_{p_f}(E_{p_f}(r)))$ and q can check whether $D_{p_f}(E_{p_f}(r)) = r$. If yes, p_f is correctly identified, i.e., p really talks to the peer it intends to, and Q is forwarded to p_f.

4. If not, then p_f has a new IP address (the case that somebody tries to impersonate p_f is covered implicitly by the signature check above) and p sends a query to P-Grid to retrieve the current $addr_{p_f}$ using Id_{p_f} as the key.

5. p collects all answers $t_i = (Id_{p_f}, E_{p_f}, addr_{p_f}, TS_{p_f}, D_{p_f}(Id_{p_f}, E_{p_f}, addr_{p_f}, TS_{p_f}))$ it receives from the $R_j \in \Re_{p_f}$) (if extended security is required then the R_j should sign their answers, i.e., send $(t_i, D_{R_j}(t_i))$). p has to collect at least R_{min} answers to detect misinformed or malicious peers, i.e., checks whether a certain quorum of the answers is identical (R_{min} is defined by each individual p according to its local requirements for trustworthiness of the reply). Otherwise the query is repeated a certain number of times before aborting.

   (a) As an optimization the quorum can be avoided under certain circumstances. If p already knows E_{p_f}, e.g., from the construction of the P-Grid or because it has already done a certain number of (quorum-based) queries for E_{p_f} that have resulted in identical answers, so that it can assume that its E_{p_f}, then it can immediately check the validity of the answer by $E_{p_f}(D_{p_f}(Id_{p_f}, E_{p_f}, addr_{p_f}, TS_{p_f})).Id_{p_f} = t_i.D_{p_f}$.

   (b) The scheme can be further optimized (and made more robust and secure) by having all peers store the E_p's that they receive.

6. Now p can proceed with step 3. And in case this is successful p enters $(Id_{p_f}, addr_{p_f}, E_{p_f}, TS_{p_f}))$ in its local cache.

4 Example

4.1 Example scenario

Consider an example P-Grid, as shown in Figure 2.

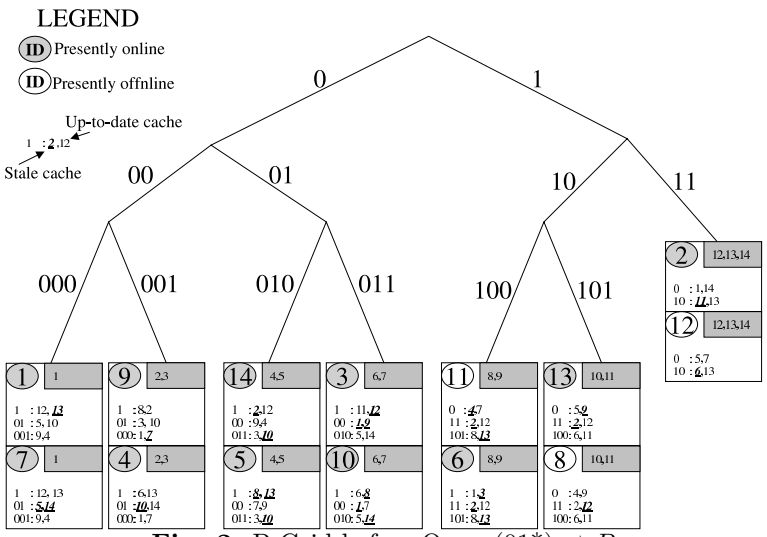


Fig. 2. P-Grid before Query(01*) at P_7

Peer P_i is denoted by i inside an oval. Currently online peers are indicated by shaded ovals, while the unshaded ovals represent off-line peers. We will consider a query $Q(01*)$ at P_7 for our example. P_7 holds the public key and latest physical address mapping about P_1 (updated by P_1). These are shown in the shaded rectangle in the upper-right corner. We assume that the peers are represented by paths of a length of 4 bits. For example, information about P_1 can be obtained by $Q(P_1)$, i.e. $Q(0001)$.

Since P_7 is responsible for the search path 000, it stores references for paths starting with 0, 01 and 001, so that queries with these prefixes may be forwarded to respective peers for further processing. These references to other halves of the P-grid subtrees at all depths form P_7's routing table. The cached physical address of these references may be up-to-date (for example P_{12}) or stale (for example $\underline{P_5}$). Stale cache entries are underlined. Peers, however, do not realize that the cached entries are useless until they try to use them. The public key infrastructure described in Section 3 is used to determine whether the contacted peer is indeed the peer which was intended to be contacted. This authentication scheme takes care both of stale caches as well as impersonation. The underlying assumption here is that the public key is generated once during the bootstrap and is not changed afterwards. The public key too may be revoked and/or renewed using messages signed with the last public key.

A peer P_q decides that it has failed to contact peer P_s if one of the following happens:

- No peer is available at the cached address. In this case, P_q trivially determines

that P_s is unavailable.

- The contacted peer fails in the authentication[16]. If any peer $P_{s'}$ is present at the physical address as cached by P_q for P_s, P_q will use P_s's public key to verify whether $P_{s'}$ is indeed P_s. If $P_{s'}$ fails the identity test, P_q concludes that it has failed to contact P_s. Since only P_s knows its private key, only P_s can pass the identity test. Vulnerability and possible attacks in using this scheme may arise from the weaknesses of a decentralized public key infrastructure. The in-depth security aspects of our public key infrastructure is subject to complimentary work and beyond the scope of this paper. However, similar systems such as PGP [12, 14] using transitively certified public keys have gained wide-spread acceptance. Those schemes provide probabilistic guarantees by the use of multiple paths, whose resilience against attacks is analogous to that of our approach of storing information at multiple replicas.

4.2 Progress of Query (Search)

While the P-Grid is in the state as shown in Figure 2, assume that P_7 receives a query $Q(01*)$. P_7 fails to forward it to P_5 and P_{14} since the cache entries are stale. The isolated query algorithm (Algorithm 1 in Section 5.2) fails immediately.

In the recursive query version (Algorithm 2 in Section 5.2), a peer that has failed to contact any of the peers to which it could forward the query, first tries to discover the latest addresses for those routing table entries. In our example, P_7 initiates $Q(P_5)$, i.e., $Q(0101)$, which needs to be forwarded to either P_5 or P_{14}. This fails again. P_7 may then initiate $Q(P_{14})$, i.e., $(Q(1110))$, which needs to be forwarded to P_{12} and (or) P_{13}. P_{12} is off-line, so irrespective of the cache being stale or up-to-date, $Q(P_{14})$ fails to be forwarded to P_{12}. P_{13} is online, and the cached physical address of P_{13} at P_7 is up-to-date, so $Q(P_{14})$ is successfully forwarded to P_{13}.

At P_{13}, P_{13} needs to forward $Q(P_{14})$ either to P_2 or P_{12}. It fails to forward it to P_{12}. Further, P_{13} fails to forward it to P_2 because its cached entry for P_2 is stale. P_{13} thus initiates $Q(P_2)$, i.e., $(Q(0010))$. It may initiate $Q(P_{12})$ as well.

¿From P_{13}, $Q(P_2)$ is forwarded to P_5. From P_5, $Q(P_2)$ is forwarded to one of P_7 and P_9. Assume P_9 replies (though in parallel it may be forwarded from P_7 to P_4, P_9 and then eventually be answered).

Thus P_{13} learns P_2's location and updates it in its cache. P_{13} also forwards $Q(P_{14})$ to P_2. P_2 provides P_{14}'s up-to-date address, and P_7 updates it in its cache (directly or via P_{13}, depending on the implementation).

P_7 now forwards $Q(01*)$ to P_{14}. In this case, not only is the original query satisfied, but also P_7 has an opportunity to learn and update P_5's physical address, since P_{14} is responsible for P_5's latest physical address. Thus, apart from successfully replying to the original query, P_7 updates the physical address for P_{14}, and possibly of P_5. Further, because of the initiated child queries, P_{13} updates its cache for P_2. The final state with several caches updated after the end of $Q(01*)$ at P_7, is shown in Figure 3.

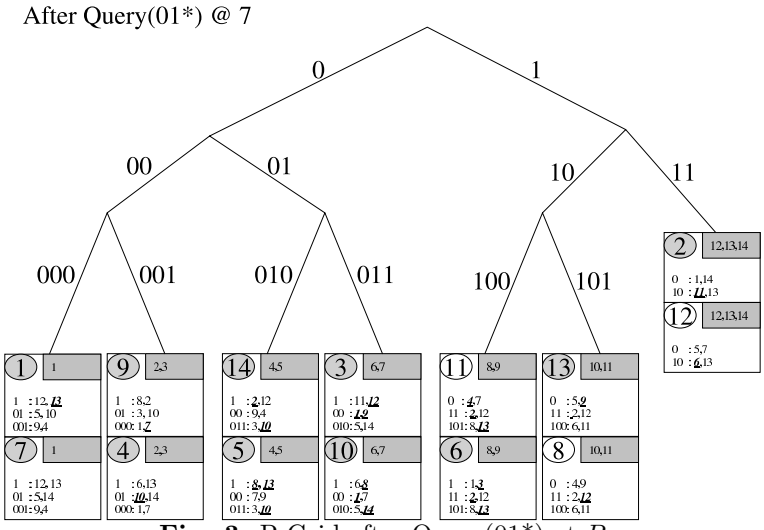


Fig. 3. P-Grid after Query(01*) at P_7

5 Query (Search) algorithms

5.1 Notation and primitives

The example in Section 4 has illustrated the strategies to react to stale entries in caches for routing table entries. The non-recursive version (called Isolated-Query) succeeds as long as at least one entry is up to date, and the concerned peer is online, when a query needs to be forwarded. Thus its functionality is dependent on the redundancy of the routing tables. The recursive version is meant to further enhance the robustness, and thus tries to find an online peer, even if the entry in the routing table is stale. Thus, intermediate failures trigger new queries to locate the peers (their latest address). Consequently, with the additional effort of recursion, it reduces the probability of failure of individual original queries and implicitly updates some of the stale cache entries (self-healing). These algorithms deal with dynamic physical addresses and offline/online behavior of peers, and are in effect extensions to the original search strategy of P-Grid described in [1, 2] (summarized

in Section 2.1).

In the following, $\Re_{P_i}$ denotes the set of peers (replicas) which has the result for a query seeking information about P_i. If a node P_q receives a query $Q(P_i)$ and $P_q \notin \Re_{P_i}$ it tries to route (forward) the query to some peers in its routing table. The set of corresponding entries in its routing table is then denoted as $\Re_{P_i,q}$.

The following section provides the pseudo-code of the algorithms to address the online/off-line behavior of peers and the issue of stale physical addresses cached for each entry in $\Re_{P_i,q}$ at P_q. We devise two algorithms: Isolated-Query (Algorithm 1) terminates if all cache entries are stale or the peers to contact are offline; Recursive-Query (Algorithm 2) in contrast triggers children queries upon failure at any stage, until it succeeds (in a real implementation, it is necessary to use a time-to-live, so that cyclic recursions do not perpetuate). The recursive query is not isolated because failures to forward searches at intermediate stages and subsequent children queries may lead to updates of stale caches at various peers, thereby self-healing the overall P-grid.

5.2 Pseudo-code of algorithms

For simplicity some details have been deliberately omitted from the pseudo-code: To return the final reply to the original requester directly its identifier and physical address can be included in the query; and to ensure that cycles created because of recursion terminate a TTL would be included in the recursive algorithm.

Algorithm 1 Isolated-Query(P_i) at P_q (analyzed in Section 6.1)

```
1: if P_q ∈ ℜ_{P_i} then
2:    return(success/reply to query)
3: else
4:    for all P_j ∈ ℜ_{P_i,q} do
5:       if P_j @ cached physical address for P_j then
6:          forward Isolated-Query(P_i) to P_j;
7:          break; {hop successful at P_q}
8:       end if
9:    end for
10:   return(failure in forwarding query);
11: end if
```

Algorithm 2 Recursive-query(P_i) at P_q (analyzed in section 6.2)

```
1: Isolated-Query(P_i);
2: if returned(failure) then
3:    for all P_s ∈ ℜ_{P_i,q} do
4:       Recursive-Query(P_s);
5:       if success then
6:          forward Recursive-Query(P_i) to P_s;
7:       end if
8:    end for
9:    if failure for all children queries then
10:      return(failure in forwarding query);
11:   end if
12: end if
```

6 Analysis

Table 1 summarizes the notations used in our analysis.

p_{on}	Probability of peers being online.
p_{dyn}	Probability of local cache entries being stale.
μ	Probability that an isolated attempt to contact any particular peer P_i by peer P_j using its local cache information fails.
ϵ_h	Probability of failure in P-Grid query forwarding from one peer to any other peer specialized for the other half of P-Grid search-subtree.
ϵ	Probability of failure of a query in P-Grid.
n	Number of leaves in P-Grid.
r	Number of references for the other half of the subtree in P-Grid routing tables for each depth (may be different at different depths and peers).
A (A_ϵ)	Expected number of attempts required for a query (along with the achieved error rate).
$\triangle_{p_{dyn-w/o}} = \eta_{w/o} p_{dyn}$	Increase in the probability that the local cache gets stale without any rectification method in place. $\eta_{w/o}$ is the fractional change with respect to the present p_{dyn} without rectification.

Table 1: Notation used in the analysis

6.1 Analysis of an isolated search/query in P-grid

In this subsection we analyze the effect of peers going off-line and then rejoining the P-Grid community with possibly a different physical address on P-grid searches. In

[1, 2] we analyzed the efficient search from an abstract logical level, ignoring the dynamics of underlying network topology which is taken into account here.

When a peer P_q needs to forward a query $Q(P_i)$, it may fail to do so because all the peers in $\Re_{P_i,q}$ to which the query may be forwarded are off-line or their cached physical address is stale (or both). If the overall offline probability of peers is $1-p_{on}$, and the probability that the cache at P_q for each peer entry is stale is p_{dyn}, then the probability that an isolated attempt at P_q to reach a particular peer in $\Re_{P_i,q}$ is successful (denoted by $1-\mu$) is $p_{on}(1-p_{dyn})$. Likewise, the failure probability of an isolated attempt to forward a query (denoted by μ) is $1-p_{on}(1-p_{dyn})$.

Thus μ represents the coupled probability that a peer is off-line and/or the physical address associated with any peer P_s cached in P_q has changed. Consequently when attempts are made to contact r random peers from the references $\Re_{P_i,q}$ at P_q, the probability that all r attempts fail, is μ^r. So, given a per-hop error tolerance ϵ_h, we need a minimum of r references to which a search may be forwarded, such that $\mu^r < \epsilon_h$. Then the expected number of attempts to achieve a failure probability less than or equal to ϵ_h for a single hop during the routing is A_{ϵ_h} such that $\mu^{A_{\epsilon_h}} < \epsilon_h$. Thus we need at least $A_{\epsilon_h} = \lceil \frac{log\epsilon_h}{log\mu} \rceil$ references for the other half of the P-Grid subtree (at any arbitrary depth to which the query has already propagated) to achieve a given ϵ_h (and vice versa).

With a ϵ_{h_i} failure probability for query routing (at hop i), the probability of successful routing to a desired leaf node is $\prod_{i=1}^{H}(1-\epsilon_{h_i})$ where H is the required number of hops to reach the particular leaf node in question. If there are at least A_{ϵ_h} references available at any hop then $\epsilon_h \geq \epsilon_{h_i} \forall i$, and thus ϵ_h determines the worst per-hop failure probability. We use this ϵ_h for all hops, thus determining a worst case average performance in the remaining analysis.

If ϵ_h is achievable at every hop (enough references available) then the the success probability is $1-\epsilon = (1-\epsilon_h)^H$ where H is the number of times the query needs to be forwarded to reach the leaf node. Thus, the expected value of the achievable success probability is $1-\epsilon = E_H[(1-\epsilon_h)^H]$. For a general P-Grid, distribution of H and thus the expectation is difficult to evaluate, but for a balanced P-Grid, H is binomially distributed, and we get $1-\epsilon = (1-\frac{\epsilon_h}{2})^{lg_2 n}$ The expected number of attempts then is $A = \frac{lg_2 n}{2} A_{\epsilon_h}$.

In the following we will use the notations ϵ and A for the failure probability of an isolated P-Grid query and the expected number of attempts (message exchanges) required for it to succeed.

6.2 Recursive search for peers

The analysis in this section is for a balanced P-Grid tree. Given the model as in the previous subsection, if there are r references to choose from, the probability that none of the r references are reachable occurs with a probability of μ^r. In such a case, P_q needs to initiate recursively a query for $P_s \in \Re_{P_i,q}$. If it discovers a physical addresses different from that in its cache, it updates its cache, and contacts the peer at the new address. It is of course possible that the peer is off-line, and thus not available even at the updated address. The probability that a query which needs to be forwarded further will trigger a recursive search is μ^r. For a random incoming query with b bits unresolved, the probability that it needs to be forwarded is $1-2^{-b}$ (probability under binomial distribution that it cannot be answered locally). The incoming queries may be unresolved with 0 bit to $lg_2 n$ bits (under assumption of load balancing, b is uniformly distributed), and thus the expected probability of forwarding an incoming query is

$$\begin{aligned} E_b[1-2^{-b}] &= \frac{\sum_{b=0}^{lg_2 n}(1-2^{-b})}{1+lg_2 n} \\ &= 1-\frac{2(1-0.5^{1+lg_2 n})}{1+lg_2 n} \end{aligned}$$

When recursive queries are initiated at P_q, these queries will have the following fate (as summarized in Figure 4):

1. The child queries will need the routing table at the same depth at which failures had occurred (thereby creating the child queries). This will happen for an expected $\frac{r}{1+lg_2 n}$ of the r child queries. These queries will fail definitely.
2. An expected $\frac{r}{1+lg_2 n}$ of the r children queries can be locally answered, and will thus always succeed. Still, with a $1-p_{on}$ probability, each of these peers may not be contactable, simply because they are offline.
3. The remaining $\frac{r(lg_2(n)-1)}{1+lg_2 n}$ queries need to be forwarded, and will each have a failure probability of ϵ. Even if the forwarded queries succeed, with a probability of $1-p_{on}$, each of these peers still may not be contactable.

Thus probability of failure of a single hop ϵ_h of the recursive version of the algorithm is:

$$\begin{aligned} \epsilon_h &= \mu^r(1-\frac{2(1-0.5^{lg_2 n+1})}{lg_2 n+1})(\frac{1}{1+lg_2 n}+\frac{1}{1+lg_2 n}(1-p_{on})^{\frac{r}{1+lg_2 n}} \\ &\quad +\frac{(lg_2(n)-1)}{1+lg_2 n}(\epsilon+(1-\epsilon)(1-p_{on}))^{\frac{r(lg_2(n)-1)}{1+lg_2 n}}) \end{aligned}$$

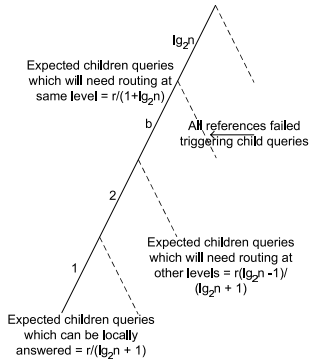


Fig. 4. Fate of children queries of recursive algorithm.

Solving $1 - \epsilon = (1 - \frac{\epsilon_h}{2})^{lg_2 n}$ for ϵ gives the probability of failure of a recursive query in a balanced P-Grid.

Now, at each hop, if r is the number of attempts (messages) made for an isolated query, then the expected number of attempts for a complete isolated query is $\frac{rlg_2 n}{2}$. In using recursion, at each hop we are using

$A_h = r + \mu^r(1 - \frac{2(1-0.5^{lg_2 n+1})}{lg_2 n+1})r\frac{lg_2 n-1}{lg_2 n+1}A$

attempts (ignoring the attempts which terminate locally), where A is the total effort needed for completing a recursive query. Thus $A = 0.5lg_2 nA_h$. Consequently, the additional effort for recursion compared to an isolated query is by a factor of $\frac{1}{1-0.5r\mu^r lg_2 n(1-\frac{2(1-0.5^{lg_2 n+1})}{lg_2 n+1})\frac{lg_2 n-1}{lg_2 n+1}}$.

An issue which we have ignored in the pseudo-code and analysis of the algorithm, is that of having orphan child processes, even when the original query has been satisfied. Cycles are also possible. However, these can easily be dealt with using a time-to-live, which is decremented whenever recursive child queries are initiated, thus ensuring that both loops and orphan queries die out.

6.3 Self-maintenance–a first order analysis

The salient aspect of the recursive algorithm is that failures trigger a process of self-healing of the whole P-Grid. In fact, even if the recursive algorithm is not used, isolated queries may be explicitly used to update the cached information about peers' latest physical addresses. A totally self-organizing system needs to be self-maintaining also, and with such a mechanism to update cached entries, P-Grid becomes self-healing. Here we give a first order analysis of this property. The analysis is first order because we ignore the overall dynamics of the system, and only focus on how to maintain p_{dyn} at a given value, assuming that over a period of time, it has a fixed rate of change. While we analyze the effect of cache

updates locally, we assume that there are no other queries or cache update processes running in parallel in the P-Grid (apart from possible child queries of the original ancestor query). As a matter of fact, such unrelated queries will in itself trigger self-maintenance, consequently improving the performance of individual queries.

¿From Section 6.2 we know that the probability of recursion is $\mu^r(1-\frac{2(1-0.5^{lg_2 n+1})}{lg_2 n+1})$. Probability of success of each of the child queries is $\frac{1}{1+lg_2 n} + \frac{(lg_2(n)-1)(1-\epsilon)}{1+lg_2 n}$. Assuming that the child queries are issued in parallel (without waiting for individual child queries to succeed or fail), probability of repair is $p_{dyn}\mu^r(1-\frac{2(1-0.5^{lg_2 n+1})}{lg_2 n+1})\frac{1+(lg_2(n)-1)(1-\epsilon)}{1+lg_2 n}$, where p_{dyn} is the fraction of references with stale cached physical addresses (other references are up-to-date and need not be repaired). Thus if $\frac{\triangle_{p_{dyn-w/o}}}{p_{dyn}} = \eta_{w/o} > 0$ is the rate at which p_{dyn} deteriorates with no rectification measures in place, then, because of the self-healing feature of our recursive algorithm, we obtain the following rate equation:

$$\frac{\triangle_{p_{dyn}}}{p_{dyn}} = \eta_{w/o} - \mu^r(1 - \frac{2(1-0.5^{lg_2 n+1})}{lg_2 n+1})\frac{1+(lg_2(n)-1)(1-\epsilon)}{1+lg_2 n}$$

For a sustainable (stable) system, we need $\frac{\triangle_{p_{dyn}}}{p_{dyn}}$ to be zero, or negative, determining the critical value of $\eta_{w/o}$ which can be tolerated by P-Grid. The Results will be elaborated in Section 7.

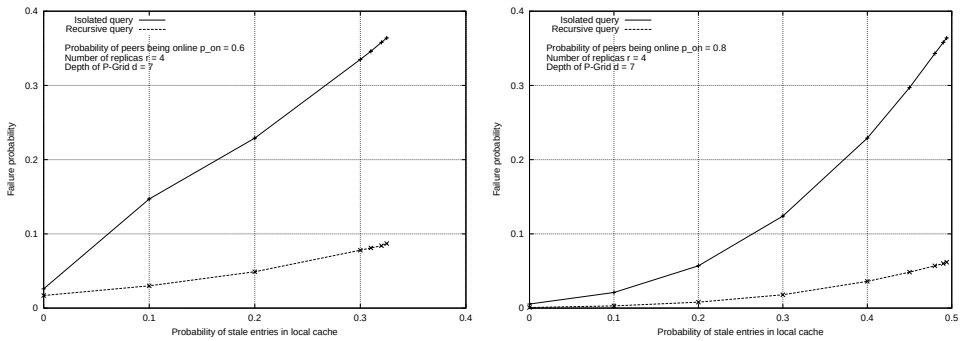
The analysis of this section is for the case when an answer from only one replica is obtained. However, for probabilistic reliability a quorum is required. The same query needs to be initiated multiple times in order to obtain replies from multiple replicas, in order to achieve a quorum, because the replicas use lazy update algorithms [6] and thus may not all be in a consistent state. From a security perspective, the quorum is more essential to thwart impersonation or denial of service attacks instigated by malicious peers. Use of a quorum can thus mitigate the effects of both individual and limited collaborative denial-of-service attacks, apart from uninformed peers. Analysis and results for creating a quorum under a model of replica consistency and peer maliciousness has been excluded from the paper purely because of space constraints, and will be included in complementary and future work.

7 Analytical results

We investigate the performance of queries without and with recursion, and study the improvement in the success rate and the additional effort incurred, as the system parameters change. Due to space limitation we give results for only the case where the P-grid tree has $n = 2^7$ leaves, and all peers have 4 references cached at any

depth. Further we consider only the cases when on an average 60% and 80% peers are online ($p_{on} = 0.6, 0.8$). We have varied p_{dyn} and observed the failure probability of non-recursive queries increases rapidly with an increase in p_{dyn}. With recursion, however, the failure probability is significantly lower, since intermediate failures trigger recursive queries, leading to self-healing effects, and thus to the eventual success of the original query. The additional effort required is marginal for even a moderately high value of μ (the effective probability that a peer is unavailable after an isolated attempt), but when μ is large (> 0.5), the effort starts increasing rapidly. Such a behavior is understandable, since the larger the number of recursive child queries, the greater is the chance of creating loops. Benefits of the recursion, however, are dual, since apart from reducing the probability of failure it induces self-healing. In the following we elaborate our results. In all the Figures of this section, the X-axis represents p_{dyn}, the probability of stale entries in the local cache.

In Figures 5(a) and 5(b) we compare the failure probability (Y-axis) of the isolated and recursive query algorithms with the variation of p_{dyn} for the two cases of p_{on} (0.6 and 0.8).



(a) (b)

Fig. 5. Probability of failure with variation of p_{on} and p_{dyn}

Figure 6(a) shows the additional effort incurred by the recursive version of the query (Y-axis) with variation of p_{dyn} for both the cases of p_{on} (0.6 and 0.8) studied here. The expected effort for the isolated-query in these cases is fixed, and equals 14 messages ($0.5 * r * lg_2(n)$).

Figure 6(b) shows the fraction of presently stale caches which will be updated as a consequence of the recursive queries (Y-axis) with variation of p_{dyn}, again for both the cases of p_{on} (0.6 and 0.8).

Figure 6(c) shows μ, the overall probability that any particular peer may not be available (either because it is offline, or has changed physical address) in an isolated attempt to contact it at a cached physical address.

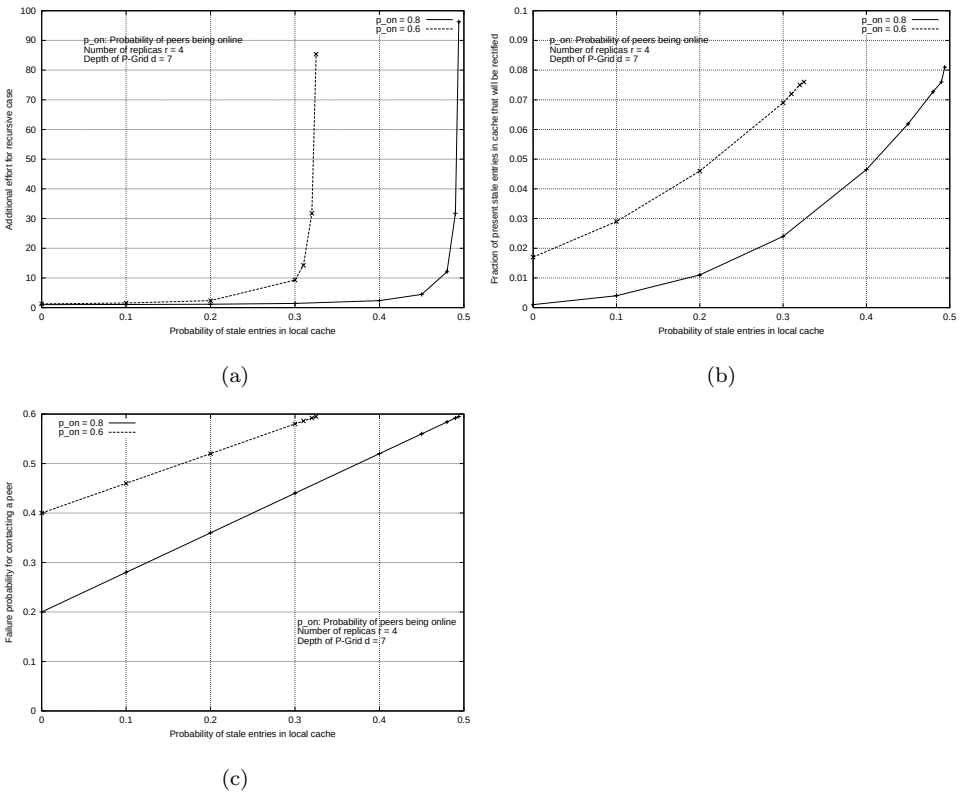


Fig. 6. Consequences of recursive version of the algorithm: additional effort and self-healing; and failure probability for contacting a peer

The results are intuitive. With higher p_{on} and lower p_{dyn}, failure rates are low, and the additional effort in using recursion is marginal. Self-healing is not critical, and thus a low fraction of stale caches being rectified does not effect the overall performance. With lower p_{on} and higher p_{dyn}, and thus higher μ, effort for recursion increases, but it manages to restrict the failure rate, unlike the non-recursive version (isolated query), where the effort is constant, but the failure rate increases rapidly. Thus even when μ is relatively low, the failure probability of the recursive version of the search is significantly lower than the non-recursive counterpart. For both the cases of p_{on}, we observe that even when μ is moderately high (0.5) ($p_{dyn} = 0.167$ and 0.375 for $p_{on} = 0.6$ and 0.8 respectively) (see Figure 6(c)), the failure probability of the non-recursive query is as high as 0.2-0.25, while the recursive version limits the failure to $0.03 - 0.05$ (shown in Figures 5(a) and 5(b)) with only an additional effort of 2 to 3 times (additional effort is shown in Figure 6(a)). However, when μ is very high (≈ 0.6 and larger) ($p_{dyn} = 0.33$ and 0.5 for $p_{on} = 0.6$ and 0.8 respectively), even then the recursive query can restrict the failure rate, but pays heavily in terms of additional effort (see Figure 6(a)). The effort rapidly grows, because of numerous child queries which potentially form loops. Such an avalanche can however be easily

restricted by the use of time-to-live for recursion, but that will increase the failure probability, and has been omitted from the algorithm and analysis for the sake of simplicity. Further, with the self-healing induced by recursion, as shown in Figure 6(b), the cache is updated. The self-healing property is also encouraging, since with larger μ, the chances of recursions increase, and so do the chances of rectifying stale cached physical addresses.

The initial efforts to manage dynamic addresses of peers with our approach is promising, not only because the mechanism works with a good success rate, but also because it has inherent self-healing features, which are essential for any self-organizing system to succeed.

8 Related Work

As mentioned earlier in Section 2, Gnutella [5] can handle dynamic IP addresses easily because peers actively announce their availability (IP address) and maintain a number of permanent connections to other peers. However, this comes at the expense of very high network traffic. For example, the number of messages caused by a single Gnutella message (ping) to join the Gnutella network and its replies (pong) for a standard Gnutella setup ($TTL = 7$, 4 connections C per peer, i.e., each peer forwards incoming messages to 3 other peers) can be calculated as $2 * \sum_{i=0}^{TTL} C * (C-1)^i = 26240$.

Freenet [4] originally proposed address resolution keys stored inside Freenet itself to cope with dynamic IP addresses: A mapping between a virtual address which is used in the routing tables and a real IP address is stored inside Freenet itself as so-called signed-subspace keys. However, no details on the concrete algorithm are available. Additionally, entries in a signed-subspace can only be changed by the owner of the subspace which seems to make it impossible that peers change their address in a secure way: either the owner can change the mapping or all peers have the necessary security credentials to do it but then the security provision of a signed subspace is futile. Alike, no analysis on the efficiency of this approach is available. According to the Freenet website [11] they suggest to use a dynamic DNS service such as DynDNS.org.

DynDNS.org [8] is a service that allows a user to map a dynamic address to a static hostname, i.e., a user selects a hostname in one of the supported domains, for example, *p-grid.homedns.org*, and enters the current IP address via a web interface that is protected by a username/password scheme (this can be automated in a simple way via a HTTP session done by a program). DynDNS.org then provides a

mapping from *p-grid.homedns.org* to the provided IP address on their DNS server. If the IP address changes the mapping can be updated. By using the chosen DynDNS hostname in routing tables and for downloads an up-to-date mapping is available. Besides DynDNS many similar services exist [17].

In theory even DNS [3] itself could now be used for maintaining dynamic IP addresses. [19] added support for dynamically updating a nameserver's database to the original design of DNS, e.g., for allowing a DHCP server to enter up-to-date mappings. Access to the database is based on the IP address of the requester. [18] added transactional signatures (TSIG) based on symmetric cryptography to make updates more secure and [9] finally introduced a fully-flexed infrastructure for secure DNS updates based on public-key cryptography. However, all these schemes require elaborate configurations, and are intended for communication among a number of dedicated servers, not for allowing the average user to change the DNS database (every peer would have to be granted access which would not scale at all). To get around these problems [13] describes an approach to make DNS self-configuring and self-administering. However, security is not addressed at all and unique identity of peers would have to be ensured by management processes outside the proposed system, i.e., consistent and secure assignment of up-to-date IP addresses to names and prevention of name reuse to ensure unique identity mappings are not addressed. It may even occur that hosts are renamed (merging of two previously unconnected networks with name conflicts) which renders this approach unusable for our purposes.

In order to handle dynamic physical addresses securely we introduced a self-organizing public key infrastructure. PGP [12] is comparable to our concepts because it offers a similar, decentralized approach. PGP exploits the small world phenomenon of social acquaintance to create a web of trust on peer's public key. Consequently, it obliterate the need of central authorities for a public key infrastructure, and has been an enormous success as a civilian purpose decentralized public key infrastructure. PGP uses transitivity of trust, whereby, if P_A trusts that K_B is P_B's public key, and also relies (personally determined) on P_B to certify a third party's public key, then P_A will use K_C as P_C's public key, if P_B certifies the same. The strength of a chain is determined by the weakest link, and hence a simple transitive trust is highly vulnerable. An extension of the approach has been to include multiple paths [14] in an effort to bolster authentication, but reliability of such approaches is limited because of intersecting paths among other reasons, and thus needs apart from the resource consuming effort to find multiple paths, authentication metrics [15] to quantify the reliability of such multiple paths. Thus it is

heavy on both network resources as well as computational resources. Further, both the multiple paths and the metrics need to be evaluated at each peer, and thus the effort is not shared. Our use of P-Grid virtual infrastructure to create a distributed public key infrastructure however is bereft of these drawbacks, since discovery of keys is done using efficient searches, and probabilistic guarantee (PGP and variants can also provide only probabilistic guarantee) can be obtained through replication, and using quorums to mitigate malicious behaviour. Thus the effort of storing as well as searching is distributed, and is not heavy either computationally or on network resources. Since a subset of peers (to which searches are efficiently routed) are responsible for a given key, it is also easy to either revoke or update public key using lazy update algorithms [6]. Thus, though we have used our approach for handling dynamic address of participants in a P2P system, the underlying public-key infrastructure we have proposed in itself merits further study, possibly in conjunction with a hybrid approach of PGP, and defines a direction of our future work.

9 Conclusions

This paper described a decentralized, self-maintaining, light-weight, and secure name service. We have demonstrated that our algorithm is robust and applicable in unreliable environments such as current peer-to-peer systems and operates well even if we assume low online probabilities. The service is based on our P-Grid P2P system and applied in P-Grid itself to remedy the problem of dynamic IP addresses of peers. However, it can be used for other applications that require constant server access despite changing IP addresses of the servers as well. Also the approach can easily be generalized to support other kinds of name services because we do not constrain the type of mappings stored. The service offers a sufficient level of security—deliberately balancing costs against application requirements—by combining a PGP-like approach for circulating public keys with a quorum-based query scheme that provides robustness against cheating peers. It is self-maintaining since it requires only little manual configuration and then operates without requiring further maintenance. To demonstrate the efficiency and applicability of our approach we have provided an analytical model and have evaluated our algorithm against this model. The analytical model is a significant contribution in contrast to most of the literature in this area which relies on empirical measurements only.

References

[1] Karl Aberer. Scalable Data Access in P2P Systems Using Unbalanced Search Trees. In *Proceedings of Workshop on Distributed Data and Structures (WDAS-2002), Paris, France*, 2002.

[2] Karl Aberer, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. Improving Data Access in P2P Systems. *IEEE Internet Computing*, 6(1), Jan./Feb. 2002. http://lsirpeople.epfl.ch/hauswirth/papers/w1058.pdf.

[3] Paul Albitz and Cricket Liu. *DNS and BIND*. O'Reilly & Associates, fourth edition, 2001.

[4] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, 2001. http://freenetproject.org/cgi-bin/twiki/view/Main/ICSI.

[5] Clip2. The Gnutella Protocol Specification v0.4 (Document Revision 1.2), Jun. 2001. http://www.clip2.com/GnutellaProtocol04.pdf.

[6] Anwitaman Datta, Manfred Hauswirth, and Karl Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. Technical Report IC/2002/47, École Polytechnique Fédérale de Lausanne (EPFL), 2002. http://lsirpeople.epfl.ch/hauswirth/papers/TR-IC-2002-47.pdf.

[7] R. Droms. *Dynamic Host Configuration Protocol*. Network Working Group, IETF, March 1997. http://www.ietf.org/rfc/rfc2131.txt.

[8] LLC Dynamic DNS Network Services. DynDNS website, Sep. 2002. http://www.dyndns.org/.

[9] D. Eastlake. *Domain Name System Security Extensions*. Network Working Group, IETF, Mar. 1999. http://www.ietf.org/rfc/rfc2535.txt.

[10] K. Egevang and P. Francis. *The IP Network Address Translator (NAT)*. Network Working Group, IETF, May 1994. RFC1631, http://www.ietf.org/rfc/rfc1631.txt.

[11] Freenet. Windows Snapshot, Sep. 2002. http://www.freenetproject.org/wiki/index.php?WindowsSnapshot.

[12] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1994.

[13] Paul Huck, Micheal Butler, Amar Gupta, and Michael Feng. A Self-Configuring and Self-Administering Name System with Dynamic Address Assignment. *ACM Transactions on Internet Technology*, 2(1):14–46, 2002.

[14] Michael K. Reiter and Stuart G. Stubblebine. Resilient authentication using path independence. *IEEE Transactions on Computers*, 47(12):1351–1362, 1998.

[15] Michael K. Reiter and Stuart G. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2):138–158, 1999. http://doi.acm.org/10.1145/317087.317088.

[16] Bruce Schneier. *Applied Cryptography*, chapter 21, "Identification Schemes". John Wiley & Sons, 1996.

[17] David E. Smith. Dynamic DNS, May 2002. http://www.technopagan.org/dynamic/.

[18] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington. *Secret Key Transaction Authentication for DNS (TSIG)*. Network Working Group, IETF, May 2000. http://www.ietf.org/rfc/rfc2846.txt.

[19] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. *Dynamic Updates in the Domain Name System (DNS UPDATE)*. Network Working Group, IETF, April 1997. http://www.ietf.org/rfc/rfc2136.txt.