Efficient, self-contained handling of identity in peer-to-peer systems[*]

Karl Aberer[†], Anwitaman Datta, Manfred Hauswirth
School of Computer and Communication Sciences
Swiss Federal Institute of Technology Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
Email: {karl.aberer, anwitaman.datta, manfred.hauswirth}@epfl.ch

Abstract

Identification is an essential building block for many services in distributed information systems. The quality and purpose of identification may differ, but the basic underlying problem is always to bind a set of attributes to an identifier in a unique and deterministic way. Name/directory services such as DNS, X.500, or UDDI are a well-established concept to address this problem in distributed information systems. However, none of these services addresses the specific requirements of peer-to-peer systems with respect to dynamism, decentralization and maintenance. We propose the implementation of directories using a structured peer-to-peer overlay network and apply this approach to support self-contained maintenance of routing tables with dynamic IP addresses in structured P2P systems. Thus we can keep routing tables intact without affecting the organization of the overlay networks, making it logically independent of the underlying network infrastructure. Even though the directory is self-referential, since it uses its own service to maintain itself, we show that it is robust due to a self-healing capability. 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 the P-Grid P2P lookup system (http://www.p-grid.org/) and give a detailed analysis and simulation results demonstrating the efficiency and robustness of our approach.

Keywords: C.2 [**Communication/Networking and Information Technology**]: Distributed Systems—*Peer-to-Peer Systems*; C.2.4 [**Distributed Systems**]: Distributed Databases—*Peer-to-Peer, Identity Handling*; C.2.4 [**Distributed Systems**]: Distributed Applications—*Self-Contained, Self-Maintaining, Decentralized Directory Service*; E.1 [**Data Structures**]: Distributed data structures—*Distributed Hash Tables*; H.3 [**Information Storage and Retrieval**]: Indexing Methods—*Distributed Hash Tables, Dynamic Resilience*

[*]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.

[†]Corresponding author: Karl Aberer

1 Introduction

Identification provides an essential building block for a large number of services and functionalities in distributed information systems. In its simplest form identification is used to uniquely denote computers on the Internet by IP addresses in combination with the Domain Name System (DNS) as a mapping service between symbolic names and IP addresses. Thus computers can conveniently be referred to by their symbolic names, whereas in the routing process their IP addresses must be used. Higher-level directories, such as X.500/LDAP, consistently map properties to objects which are uniquely identified by their *distinguished name* (DN), i.e., their position in the X.500 tree. Other directories, such as UDDI, map names onto service descriptions and vice versa. These are just a few examples among many others that map sets of attributes onto objects, and that are essential to provide basic functionalities, such as routing of IP packets, searching distributed databases and retrieving certificates from public key authorities to conduct secure e-commerce.

Although the quality and purpose of identification may differ in the various domains, due to varying requirements and levels of abstraction, the basic underlying problem is always the one of binding a set of attributes to an identifier in a unique and deterministic way. Name/directory services such as DNS, X.500, or UDDI are a well-established concept to address this problem in distributed information systems. Usually these services are optimized towards the targeted problem area and differ in the degree of (de-)centralization, security guarantees, descriptive power, and flexibility. However, none of these pre-existing services addresses the specific requirements of peer-to-peer systems. Peer-to-peer systems are inherently decentralized and thus identification management should be decentralized as well, to avoid scalability problems. For example, peer-to-peer systems are rather dynamic, with nodes frequently joining and leaving the system, and a centralized identification service may easily become a bottleneck. Additionally, it is favorable not to depend on a third-party infrastructure because if this external service ceases to exist, the peer-to-peer system would no longer be operable. Thus the peers should be able to manage identification issues themselves. This provides excellent scalability but introduces security problems that need to be addressed, for example, ensuring that entries are updated only by legitimate parties, being able to detect malicious use, and surviving attacks.

Peer-to-peer systems (also called overlay networks in the literature) such as Chord [29], CAN [24], Freenet [7, 8], Pastry [27], or P-Grid [1, 4] operate on top of a routing infrastructure based on a logical identification of the peers participating in the overlay. For routing this logical identification is mapped onto an IP address in the routing tables. Since IP addresses are scarce most peers will have dynamic IP addresses that may change over time. This problem would be solved if Mobile IP [21] or IPv6 [28] were in place already and available at a large scale, because they take into account mobility (dynamism) and offer a much larger address space. However, this requires considerable changes in the basic networking infrastructure of the complete Internet and it cannot be foreseen at the moment when this will happen. Our approach could bridge this gap but can also be applied in many other settings, for example, in mobile ad-hoc networks, because it is independent of the networking infrastructure. Additionally, peers can leave and join the overlay at any time. This dynamism introduces inconsistencies into the routing tables and the whole routing process, which may make correct routing impossible if it is not dealt with appropriately.

In Chord [29], peers that (re-)join the overlay with a new IP address, adopt a new identity and introduce themselves into the routing infrastructure like a completely new node. This is mainly due to the fact that in Chord the logical identifier depends on the IP address. To repair faulty entries in routing tables resulting from node departures, the approach in [17] devises a maintenance protocol. The execution of the periodic stabilization protocol is independent of changes to the network and the adaptation resulting from repairs may compromise structural properties of the routing infrastructure, which may have been established in order to address non-uniform workloads [23]. DKS(N,k,f) [6], a generalization of the Chord model, proposes a correction-on-use protocol to maintain routing tables. The authors show that their protocol is self-stabilizing and more efficient than the Chord maintenance protocol. In their approach peers can maintain logical identifiers with changing IP addresses, but routing tables need to be reorganized at all depths with the occurrence of every update.

In Pastry [27] nodes have an independent logical ID, and upon re-entering the overlay they may enter their new ID-to-IP binding into the routing tables of peers, encountered when executing the node join protocol. However, as these peers are typically different from those already storing such bindings, stale mappings will be encountered by other peers during query routing. These stale entries are replaced by new routing entries [19], irrespective of whether the peer identified by the stale entry has rejoined with a new IP address or not.

We see the management of dynamic IP addresses in overlay networks as an instance of the more general problem of identification. In this paper we will present our decentralized, self-maintaining approach to identification and prove its applicability and validity by applying it to the basic problem of changing IP addresses in P-Grid (http://www.p-grid.org/), our structured peer-to-peer system. In contrast to the approaches of Chord, Pastry, and DKS, our strategy can track changes in the mapping. This is important as soon as information on the characteristics of specific peers is exploited for routing purposes, such as their trustworthiness, quality of service or locality. Thus our approach is in particular relevant for applications such as e-commerce [11], trust management [3], and mobility management in ad-hoc networks. If information about peers is gathered from earlier interactions and the choice of routing table entries is made dependent on such properties, changes to the structure of the overlay network due to modification of routing tables should be avoided if possible (unless triggered by the application). This is in particular true for changes resulting from a peer's physical address, which may be completely independent of a peer's other properties. The approaches of changing the logical IDs or restructuring routing tables as a result of changes to the peers' physical IDs (Chord, Pastry) would incur a loss of information. Thus our approach makes the overlay network logically independent of the underlying physical network. This separation of concerns is an important step towards creating semantic overlay networks as a basic constituent in distributed information management. It is worth to mention that this functional advantage comes at no specific additional cost. All approaches (including the one we introduce here) incur message costs of order $O(\log^2 n)$ for maintenance.

Currently file-sharing is the most popular use of P2P systems. Here, user response time is a major issue and identification is viewed to be only of subordinate importance. However, this is already changing, since reputation, data authenticity, and fair use of resources have become major issues in P2P systems and require identification as a service to address them. Also, quickly finding the peer that offers information of interest will reduce response time. If peers have dynamic network addresses, this again requires an identification service as described in this paper.

To support dynamic IP addresses as an application of identification, it is necessary to address the following problems: (1) How can universally unique identifiers be mapped onto physical addresses in a secure, decentralized, and efficient way, and be maintained securely by the owner? (2) With the possibility of changes of the mapping, i.e., the physical addresses, how can a peer detect whether it is still talking with the intended entity? This means that, (a) if peer p_1 goes offline and a different peer p_2 gets associated with p_1's old IP address, the other peers in the system must be able to detect this change and react accordingly, and (b) if 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 further information transfers.

Unstructured P2P systems, for example, Gnutella [9], and super-peer P2P systems, for example, Kazaa (FastTrack), are able to address the first problem in a simpler manner than structured P2P systems but have other shortcomings, for example, excessive bandwidth consumption (Gnutella) or scalability limitations due to inherent centralization (FastTrack). However, any peer-to-peer system should actually address problem (2) for security reasons to avoid simple denial-of-service (DOS) attacks by registering false mappings.

In our approach, peers generate universally unique identifiers locally, independent of their IP address or any other global information, and store them along with their public key, their current IP address, and a cryptographic signature in the P-Grid P2P lookup system [1, 4] on a certain number of peers (replicas). Instead of IP addresses, the unique IDs are used for querying and routing. Mapping an ID onto an IP address in the routing process 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, the peer is either offline or has changed its IP address. 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 is really 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, elaborated in [11].

This strategy ensures secure mapping, but may seem like introducing a recursive hen-egg problem, as we use the mechanism (P-Grid) that depends on using the mappings, also for storing and maintaining the mappings. We show that, despite this recursive dependency, it is in fact possible to devise such a self-contained service which is completely decentralized, self-maintaining, and light-weight, by devising algorithms implementing self-healing capabilities. Decentralization may require additional security precautions, which we take into account, but it is of primary concern to avoid performance and especially administrative bottlenecks, i.e., no central authority is required to maintain mappings, but this service is provided securely by the users of the service themselves. In turn, self-maintenance must be addressed properly because in a decentralized system no central authority can enforce maintenance.

The paper is structured as follows: Section 2 introduces P-Grid, which we use to verify our approach. Section 3 then defines our identification protocol which is demonstrated in Section 4 by a simple example. Section 5 then provides the algorithms which are analyzed and evaluated in Sections 6 and 7. Section 8 presents related work and we draw our conclusions in Section 9.

2 The P-Grid data structure

Since the approach presented in this paper is based on and used in P-Grid as a proof of concept, we briefly introduce it. P-Grid is a distributed data structure based on the principles of distributed hash tables (DHT) [22]. As any DHT approach P-Grid is based on the idea of associating peers with data keys from a key space $\mathcal{K}$. Without constraining general applicability we will only consider binary keys in the following. In contrast to other DHT approaches we do not impose a fixed or maximal length on the keys, i.e., we assume $\mathcal{K} = \{0,1\}^*$.

In the P-Grid structure each peer $p \in Peers$ is associated with a binary key from $\mathcal{K}$. We denote this key by $path(p)$ and will call it the path of the peer. This key determines which data keys the peer has to manage, i.e., the keys in $\mathcal{K}$ that have $path(p)$ as prefix. In particular the peer has to store them. In order to ensure that the complete search space is covered by peers we require that the set of peers' keys is *complete*. The set of peers' keys is complete, if for every prefix s_{pre} of the path of a peer p there exists a peer p', such that $path(p') = s_{pre}$, or there exist peers p_0 and p_1, such that $s_{pre}0$ is a prefix of $path(p_0)$ and $s_{pre}1$ is a prefix of $path(p_1)$. Naturally, one of the two peers p_0 and p_1 will be p itself in that case. Completeness is guaranteed by P-Grid's construction algorithm. We do not exclude the situation where the path of one peer is a prefix of the path of another peer. This situation will occur during the construction and reorganization of a P-Grid. Ideally, this situation is avoided, since otherwise peers with shorter paths (prefixes) will have high storage loads and thus load balancing is compromised. Thus, any algorithm for maintaining a P-Grid should eventually converge to a state where the P-Grid is *prefix-free*, i.e., for peers p_0 and p_1 we have $path(p_0) \not\subseteq path(p_1) \wedge path(p_1) \not\subseteq path(p_0)$, where $s \subseteq s'$ denotes the prefix relationship among strings s and s'. An indexing structure based on similar principles as P-Grid which does not require the prefix-free property is described in [20].

We also allow multiple peers to share the same paths, in that case we call the peers replicas. The number of peers that share the same path is called the *replication factor* of the path. Replication is important to support redundancy and thus robustness of a P-Grid in case of failures and to distribute workload when searching in a P-Grid.

To be able to search in P-Grid, peers maintain *routing tables*. The routing tables are defined as (partial) functions $ref : Peers \times N \to \{Peers\}$ with the properties

1. $ref(p, l)$ is defined for all $p \in Peers$ and $l \in N$ with $1 \leq l \leq |path(p)|$
2. $ref(p, l) \subseteq Peers_{s_1 s_2 \ldots s_{l-1}(1-s_l)}$ with $path(p) = s_1 s_2 \ldots s_{l-1} s_l \ldots s_k$, $k \geq l$

where $Peers_t = \{p \in Peers | t \subseteq path(p)\}$ for $t \in \mathcal{K}$.

For the same association of peers with paths, different P-Grids can be obtained depending on the choice of $ref(p, l)$. Algorithms for construction and maintenance of a P-Grid have been introduced in [2].

Having multiple references at each level l again is necessary to guarantee robustness of the data structure. In the following, r denotes the maximum number of references maintained at each level. The search algorithm for locating data keys indexed by a P-Grid is defined as follows: Each peer $p \in Peers$ is associated with a location $loc(p)$ (IP address in the network). Searches can start at any peer. Peer p knows the locations of the peers referenced by $ref(p, l)$, but not of other peers. Thus the function $ref(p, l)$ provides the necessary routing information to forward search requests to other peers in case the searched key does not match the peer's path. Let $t \in \mathcal{K}$ be the searched data key and let the search start at $p \in P$. Algorithm 1 shows P-Grid's basic recursive search algorithm.

Algorithm 1 always terminates successfully, if the P-Grid is complete and all peers are reachable. Due to the definition of ref, $search(t, loc(p))$ will always find the location of a peer at which the search can continue (use of completeness). With each invocation of $search(t, loc(p))$

Algorithm 1 Search in P-Grid: search(t, loc(p))

```
1: if path(p) $\subseteq$ t then
2:     return(loc(p));
3: else
4:     determine maximal $l$ such that $t_1 \dots t_{l-1}(1-t_l) \subseteq path(p)$;
5:     r = randomly selected element from ref(p,l);
6:     search(t, loc(r));
7: end if
```

the length of the common prefix of $path(p)$ and t increases at least by one. Therefore the algorithm always terminates.

In case of an unreliable network, it may occur that a search cannot continue since the peer r selected from the routing table is not available. Then alternative peers can be selected from the routing table to continue the search.

3 The identification protocol

This section defines the algorithm and protocol for maintaining ID-to-IP mappings in P-Grid. Generalizing this specific example to other mappings is implicit and straight-forward.

Each peer p is uniquely identified by a universally unique identifier (UUID) Id_p. Each peer generates this identifier locally once in the bootstrap phase by applying a cryptographically secure hash function to the concatenated values of the current date and time, the current IP address of the peer $addr_p$ and a large random number. At bootstrap each peer p also generates a private/public key pair D_p/E_p once. In the following, we use the standard security notations $D_p(x)$ and $E_p(x)$, where $D_p(E_p(x)) = E_p(D_p(x)) = x$, to denote the application of the private and public keys in an asymmetric encryption scheme. Note that keys do not have to be certified since we do not need legally binding authentication guarantees as provided by certification authorities. Also, this would introduce centralization and limit scalability. Instead, our approach follows a PGP-like strategy of distributing signed mappings and public keys via independent paths, and we apply a quorum-based strategy to find trustful mappings. This provides a similar level of security, but, of course, is not legally binding.

In P-Grid routing tables and the index hold only these identifiers. Each peer p additionally has a cache of mappings $(Id_i, addr_i, TS_i)$ that it already knows. TS_i denotes a timestamp which must be included to prevent replay attacks, i.e., the recording of transmitted information by a malicious party and replaying it at a later time (the timestamp guarantees the freshness of messages).

The algorithm for handling dynamic IP addresses works as follows (inserts and updates are done according to the algorithm presented in [12]):

Bootstrap

1. Upon startup p determines its current IP address.
2. p generates Id_p, D_p/E_p.
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. 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. Note that the tuple contains a signed version of the information to be inserted, to assure that only the originator of the information can change it. This is a standard strategy used in security which we will use in all data manipulation operations in the following. How the checking process works, will be described in those steps of the algorithm, where it is required. If Id_p already exists in the P-Grid, p is notified. However, this is very improbable because of the randomization in the generation of Id_p and the application of a cryptographically secure hash function. Nevertheless, if it should occur, p generates a new Id_p and repeats this step.

4. The previous step is repeated a limited number of times, which we denote as R_{min} in the following, 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 two 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 in a Byzantine way 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_is 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. Note that this operation does not cause a chase, because the replicas responsible for storing the mapping already know E_p from the original insertion of the mapping in the bootstrap phase.

Operation phase
In the operation phase 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.

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 to which peers p_f it can forward the query 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 ρ, contacts p_f and sends $E_{p_f}(\rho)$. As an answer p_f must send $(D_{p_f}(E_{p_f}(\rho)))$ and p can check whether $D_{p_f}(E_{p_f}(\rho)) = \rho$. 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, it can assume that its E_{p_f} is correct. Thus it can immediately check the validity of the answer by $E_{p_f}(D_{p_f}(Id_{p_f}, E_{p_f}, addr_{p_f}, TS_{p_f})).Id_{p_f} = t_i.Id_{p_f}$.

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

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

4 Processing queries

Figure 1 shows a typical state of a P-Grid network.

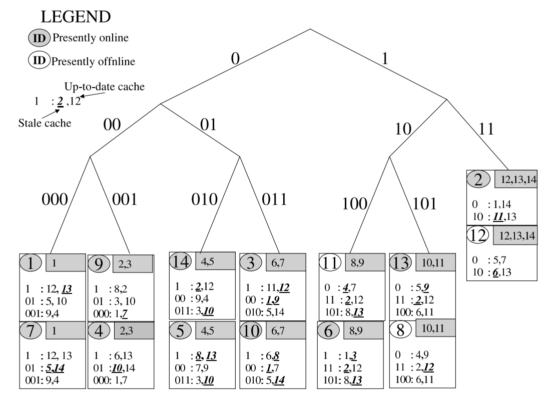


Figure 1: P-Grid before Query(01*) at P_7

Peer P_i is denoted by i inside an oval. Online peers are indicated by shaded ovals, offline peers by unshaded ovals. Peers under the same branch are replicas. For example, P_1 and P_7 are both responsible for paths starting with 000. Without loss of generality we assume that Id_p has a length of 4 bits. Thus P_7 holds the public key and latest physical address mapping about P_1 (updated by P_1) because P_7 is responsible for the paths 0000 and 0001. The shaded rectangle in the upper-right corner of each peer shows the peer IDs that a peer is responsible for, i.e., whose public key and physical address mapping it manages. Note that there exists no dependency between the peer identity ($id_{P_7} = 0111$) and the path it is associated with ($path(P_7) = 0000$). In its routing table P_7 stores references for paths starting with 1, 01 and 001, so that queries with these prefixes can be forwarded closer to the peers holding the searched information. The cached physical addresses of these references may be up-to-date (for example, P_{13}'s) or be stale (denoted by underlining, for example, $\underline{P_5}$).

A peer P_q decides that it has failed to contact a peer P_s, if one of the following happens: (1) No peer is available at the cached address (trivial case). (2) The contacted peer fails in the authentication as described in step 3 in the operation phase of the algorithm described in the previous section: P_q will use P_s's public key to verify P_s's identity. Since only P_s knows its private key which must have been used for the signature, it is the only peer that can pass the identity test. In either of the above two cases an up-to-date mapping must be obtained by querying the P-Grid. We have investigated two querying strategies:

Isolated-Query: Upon receiving a query a peer checks whether it can answer the request. If not, it forwards the query to at least one of the peers in its routing table according to P-Grid's routing algorithm. If none of these peers can be contacted, the query is abandoned and fails.

Recursive-Query: If a peer fails to contact peers in its routing table, it initiates a new query to discover the latest "identity-to-address" mapping of any of those peers. If this is successful it forwards the query.

Assuming this initial setup, the query processing based on the protocol of Section 3 will work as follows: While the P-Grid is in the state as shown in Figure 1, assume that P_7 receives a query $Q(01*)$. P_7 fails to forward the query to either of P_5 or P_{14} since their cache entries are stale. Here, the *Isolated-Query* algorithm fails immediately.

In contrast, the *Recursive Query* algorithm would try to discover the latest addresses for the stale entries. P_7 initiates *Recursive-Query*(P_5), i.e., $Q(0101)$, which needs to be forwarded to either P_5 or P_{14}. This fails again. P_7 then initiates *Recursive-Query*(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, the query cannot 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 the query is forwarded to P_{13}. P_{13} needs to forward $Q(P_{14})$ to either P_2 or P_{12}. Forwarding to P_{12} fails and so does the attempt to forward the query to P_2 because P_{13}'s cache entry for P_2 is stale. Thus P_{13} initiates another sub-query, *Recursive-Query*(P_2), i.e., $Q(0010)$. Additionally, it may initiate *Recursive-Query*(P_{12}). P_{13} sends $Q(P_2)$ to P_5 which forwards it to P_7 and/or P_9. Let us assume P_9 replies. Thus P_{13} learns P_2's address and updates its cache. P_{13} also starts processing and forwards the parent query *Recursive-Query*(P_{14}) to P_2. P_2 provides P_{14}'s up-to-date address, and P_7 updates its cache.

Having learned P_{14}'s current physical address, P_7 now forwards the original query $Q(01*)$ to P_{14}. This does not only satisfy the original query but P_7 also has the opportunity to learn and update physical addresses P_{14} knows and P_7 needs, for example, P_5's latest physical address (we assume that peers synchronize their routing tables during communication since this does not incur any overhead). In the end, the query $Q(01*)$ is answered successfully and additionally P_7 gets to know the up-to-date physical addresses of P_{14} and possibly of P_5. Furthermore, due to child queries, P_{13} updates its cached address for P_2. Figure 2 shows the final state of the P-Grid with several caches updated after the the completion of $Q(01*)$ at P_7.

We have not explicitly mentioned concurrency issues in the example above because those are either addressed by the networking layer or do not cause problems since we support only lazy

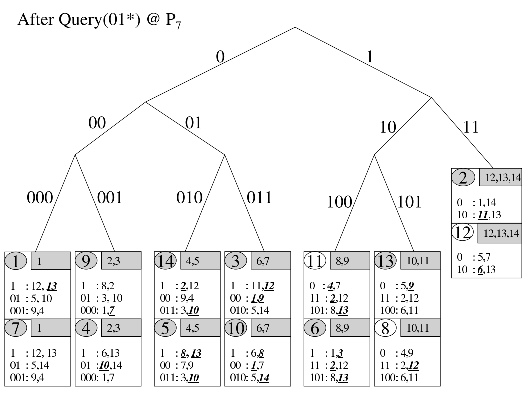


Figure 2: P-Grid after Query(01*) at P_7

consistency. For example, if the IP addresses change during authentication, the networking layer would drop the connection and the authentication would start anew without causing security or concurrency problems. If the IP address of a peer to be contacted would change after retrieving a mapping, i.e., a query would return a "stale" entry, this would be recognized, because the authentication would fail upon contacting the peer present at the retrieved address. Since only the owner can update a mapping, concurrency control is not needed here and if replicas have not been updated correctly when being queried, this would also be recognized upon contacting the peer and can be accounted for by re-issuing the same query. Additionally, we can assume that IP addresses do not change at a high frequency (normally once a peer goes online it keeps the same address for at least some hours in typical ISPs' DHCP setups). However, other "hidden costs," for example, updates to the mappings and rectifying stale cache entries (self-healing) need to be taken into account as discussed in the following sections.

5 Query (search) algorithm

Algorithm 2 shows the recursive query (search) algorithm in pseudo-code. In the following, $\Re_{path}$ denotes the set of peers (replicas) which has the result to a query for path $path$. If a peer P_q receives a query $Q(path)$ and $P_q \notin \Re_{path}$, then it tries to route (forward) the query to peers in its routing table according to P-Grid's routing strategy. The set of peers in its routing table to which the query can be forwarded, is denoted as $\Re_{path,q}$. P_s denotes a stale cache entry.

In fact, the pseudo-code summarizes three different variants of the algorithm which we will consider. The non-recursive variant (isolated query) provides a result for a query, if at least one routing entry is up-to-date for each routing step and the corresponding peers are online. Thus it only works, if the routing tables are sufficiently redundant. The two other variants (lazy repair and eager repair) are recursive and try to fix stale routing tables to increase the success probability of a query. They differ in their approach to repair stale routing entries. The eager strategy tries to repair any stale entry immediately (correction on use), whereas the lazy strategy starts the repair process only if all routing entries at one level of a routing table are stale (correction on failure). These two variants may trigger child queries at any stage in the routing process until they succeed. The recursive strategies affect the rectification of stale caches at various peers, thereby "self-healing" the overall P-Grid. To avoid cyclic recursions the queries bear unique identifiers and recursion does

Algorithm 2 Query P_q for $path$ (query($path$) at P_q)

```
1: if $P_q \in \Re_{path}$ then
2:   reply to query; {$P_q$ has the requested information}
3: else
4:   $\Re = \{P_j \in \Re_{path,q} | P_j$ is at cached address$\}$
5: else
6:   {the isolated query strategy does not fix stale cache entries as the lazy and eager repair strategies do}
7:   if ($\Re = \emptyset$ and lazy repair strategy) or eager repair strategy then
8:     $\mathcal{C}$ = {all cached peers at all levels};
9:     for all $P_s \in \Re_{path,q} \setminus \Re$ do
10:      forward query($path(P_s)$) to a non-stale entry from $\mathcal{C}$;
11:    end for
12:  end if
13:  $\Re = \{P_j \in \Re_{path,q} | P_j$ is at cached address$\}$
14:  if $\Re \neq \emptyset$ then
15:    forward query($path$) to a $P_j \in \Re$;
16:  else
17:    return failure;
18:  end if
19: end if
```

not occur when it would apply to a reference that is under repair in a parent query. Additionally, this helps to prevent concurrent repair requests for the same stale entry at one peer.

Deadlock situations, where none of the recursive queries terminates successfully and all entries at one level are stale, may occur. But experiments show that their influence on performance decreases with an increasing network size.

6 Analysis of the algorithms

In the analysis of the algorithms below we use the following notations: p_{on} denotes the probability of peers being online; p_{dyn} defines the probability that a randomly selected entry of a routing table is stale; μ is the probability that an isolated attempt to contact any particular peer P_i by peer P_j using its local cache information fails; ϵ_h denotes the failure probability of forwarding a query to any other peer specialized for the other half of the search-subtree; ϵ defines the failure probability of a query; n is the number of leaves; r is the number of references for the other half of the subtree in P-Grid routing tables for each depth. It is important to note, that we can assume these r references to be independent due to the randomized construction process of P-Grid. A (A_ϵ) denotes the expected number of attempts (message exchanges) required for a query (along with the achieved error rate).

6.1 Analysis of an isolated search/query

We first analyze the effect on P-Grid searches of peers going off-line and then rejoining the P-Grid with a possibly different physical address. 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 addresses are stale (or both). If the overall offline probability of peers is $1 - p_{on}$, and the probability that a cache entry at P_q 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 equals $1 - \mu = p_{on}(1 - p_{dyn})$. Likewise, the failure probability of an isolated attempt to forward a query equals $\mu = 1 - p_{on}(1 - p_{dyn})$.

Thus μ represents the coupled probability that a peer is off-line and/or the physical address associated with any peer P_s cached at 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 $\mu^r < \epsilon_h$. 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 depth 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 expected 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. We thus obtain the effort for one hop of the non-recursive version of the query as $A_h^{iso} = A_{\epsilon_h}$. The expected total cost to process a query in a balanced P-Grid is then $A_{iso} = \frac{\log_2 n}{2} A_h^{iso}$.

If ϵ_h is achievable at every hop (enough references available) then 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, the distribution of H and thus the expectation is difficult to evaluate, but for a balanced P-Grid, H is a binomial random variable of size $\log_2 n$ and parameter 0.5. Hence, $1 - \epsilon = (1 - \frac{\epsilon_h}{2})^{\log_2 n}$.

6.2 Recursive queries and dynamic equilibrium

While cached entries continuously get stale owing to network dynamics, they trigger recursive queries in order to update the stale mappings. Hence the recursive version of querying in P-Grid has an inherent self-healing property. With few stale mappings, there is hardly any deterioration in answering the queries, but as the stale entries accumulate over time, they lead to more frequent recursions. Thus it is expected that the system will reach a dynamic equilibrium, such that the rate of changes will equal the rate at which self-maintenance is done due to recursions. If the rate of changes in the system is very high, then the system's self-maintenance will be unable to catch up with the changes, and the system breaks down. In this section, we analyze if a dynamic equilibrium for a given rate of changes is achievable, thus determining the operational zone with respect to the rate of change, and if such an equilibrium is achievable, we evaluate the system behavior (dynamic resilience) at the equilibrium.

In the analysis and the results we quantify the rate of change considering two kinds of events in the network. With probability r_{up} an event consists of peers independently updating their address. With probability $1 - r_{up}$ an event consists of peers issuing queries. Since updates necessarily imply the execution of one query to locate a node to which the update is going to be stored, we assume that $r_{up} \leq 0.5$.

6.2.1 Eager recursion

In the eager recursion, all references are checked and the algorithm tries to rectify all stale cache entries immediately. The effort for a single hop is $A_h = A_h^{iso} + r\mu A_{rec}$ since an expected $r\mu$ recursions will be initiated at each hop, even if the original query is forwarded, where $\mu = 1 - p_{on}(1 - p_{dyn})$. The effort for a recursive query is $A_{rec} = \frac{\log_2 n}{2} A_h$. Thus the expected number of recursions $N_{rec} = \frac{A_{rec}}{A_h^{iso}} = \frac{1}{1 - \frac{\log_2 n}{2} r\mu}$.

A single hop fails when none of the references can be contacted, either because the recursive query fails or the concerned peer is offline. If ϵ_h is the probability of failure of a single hop (forwarding) in the querying process, then $1 - \epsilon_r = (1 - \frac{\epsilon_h}{2})^{\log_2 n}$ (as derived in Section 6.1). The probability that a single hop fails equals the probability that recursions are initiated ($p_{rec} = \mu$) and none of the r children responses are usable, either because the recursions themselves fail, or even if they do not fail, the concerned routing reference is offline. Thus $\epsilon_h = p_{rec}(1 - (1 - \epsilon_r)p_{on})^r$.

The dynamic equilibrium equation then is $(1 - r_{up})(N_{rec} - 1)p_{on}(1 - \epsilon_r) = r_{up}(1 - p_{dyn}) r \log_2 n$. The left hand side is the rate at which successful repairs of stale routing table entries occur, due to the $N_{rec} - 1$ additional recursive queries. Repairs can only lead to a successful update, if the peer to be repaired is online, therefore an additional factor of p_{on}. This reflects the underlying assumption that queries and thus repairs occur at a substantially higher rate than the peers are switching between offline and online state. The right hand side is the rate at which routing table entries turn stale due to changes.

We solve the above equations numerically for p_{dyn}, ϵ_r and N_{rec} to determine the system performance at the dynamic equilibrium given the system parameters p_{on}, r_{up}, r and n. For the case $p_{on} = 1$ it is also relatively easy to see from the equations that $N_{rec} \leq 1 + \frac{r_{up}}{1 - r_{up}} r \log_2 n$. This shows that eager recursion exhibits also excellent scalability with n.

6.2.2 Lazy Recursion

For lazy recursion we need to analyze the states of routing tables in more detail. There are r routing table entries at each peer at each depth. Let S_i denote the probability that i out of r routing table entries at a given depth are stale. In the following we will also say that a routing table at a given depth is in state S_i.

When using lazy recursion, queries are triggered if a peer cannot use any of its corresponding routing tables to forward a query. This happens either because the routing entries are stale and the latest ID-to-IP mappings have to be found (using recursive queries) or because the concerned peers are offline. We assume that in this case the peer issues parallel queries for all r references in order to minimize response time. Recursive queries occur with a probability $p_{rec} = \sum_{i=0}^{r} S_{r-i}(1 - p_{on})^i$ where p_{on} is the probability that any particular peer is online.

The effort to forward any query (one hop) is $A_h = A_h^{iso} + r p_{rec} A_{rec}$, where A_{rec} is the total effort in terms of number of messages for a newly issued recursive query. From the properties of P-Grid, any query requires $\frac{\log_2 n}{2}$ forwarding steps on an average to successfully answer the query. Hence, $A_{rec} = \frac{\log_2 n}{2} A_h$. Solving these recursive equations, we obtain the total number of queries including the original and children queries invoked per original (isolated) query $N_{rec} = \frac{1}{1 - \frac{r p_{rec} \log_2 n}{2}}$.

As mentioned earlier, r_{up} fraction of the network events are ID-to-IP mapping changes in comparison to $1 - r_{up}$ fraction of queries. Routing tables are created based on P-Grid's randomized construction algorithm, and each peer is equally likely to be a routing table entry for other peers. If there are N peers populating a P-Grid with n leaves (replication factor of N/n), then each peer has on an average $r \log_2 n$ routing references, r for each of the $\log_2 n$ depth of the search tree. Hence, each peer is used as a routing reference $\frac{Nr\log_2 n}{N} = r\log_2 n$ times in the whole P-Grid network. The probability that an update of a specific ID-to-IP mapping affects a routing table entry with i stale entries then is $r_{up} S_i \frac{r-i}{r} r \log_2 n$, since the original queries are uniformly distributed and $r-i$ out of the r entries are susceptible to become stale.

The self-healing comes into action while processing the original queries, each of which leads to $N_{rec}-1$ recursive queries on average, and each of these recursive queries tries to update one routing reference. When recursive queries are triggered in state S_i, which occurs with probability $S_i(1-p_{on})^{r-i}$, i out of the $N_{rec}-1$ recursive queries are initiated. Thus in state S_i, the probability that self-healing occurs due to an initial query is $(1-r_{up})\frac{1}{p_{rec}} S_i (1-p_{on})^{r-i}\frac{1}{i}(N_{rec}-1)$.

However, recursive queries may not always succeed. If the recursion is triggered when i cached references are stale and the $r-i$ others are offline, then i/r fraction of references can be updated at most. If ϵ_r is the probability that a recursive query fails, then $\binom{j}{i-j}(1-\epsilon_r)^{i-j}\epsilon_r^j$ is the probability that of the i possible repairs, only $i-j$ are successful, such that $j \leq i$ stale entries are left in the routing table even after recursive queries. $\epsilon_h = (\mu(1-(1-\epsilon_r)p_{on}))^r$ and $1-\epsilon_r = (1-\frac{\epsilon_h}{2})^{\log_2 n}$ are derived as in Section 6.2.1.

For the dynamic equilibrium, the inflow to any state S_i should equal the outflow from S_i. Hence, the dynamic equilibrium equation is

$$\begin{aligned} r_{up} \quad S_i & \quad \frac{r-i}{r} r \log_2 n \\ + & \quad (1-r_{up}) \sum_{j=i+2}^{r} \frac{1}{p_{rec}} S_j (1-p_{on})^{r-j} \frac{1}{j}(N_{rec}-1) \binom{i}{j-i-1} (1-\epsilon_r)^{j-i-1}\epsilon_r^{i+1} \\ = & \quad r_{up} S_{i+1} \frac{r-i-1}{r} r \log_2 + (1-r_{up}) \frac{1}{p_{rec}} S_{i+1} (1-p_{on})^{r-i-1} \frac{1}{i+1}(N_{rec}-1)(1-\epsilon_r^{i+1}) \end{aligned}$$

The left hand side of the equation is the inflow into state S_{i+1} from S_i as well as from $S_j \forall j > i+1$, because of partial repairs. The right hand side is the outflow from S_{i+1}. The outflow is caused by two factors: The first is because additional entries turn stale; the second occurs whenever recursions are initiated and at least one cached entry is repaired.

We solve the above equations numerically for S_i, N_{rec}, p_{dyn}, ϵ_r and thus determine the system's performance (dynamic resilience) at the dynamic equilibrium given the system parameters p_{on}, r_{up}, r and n.

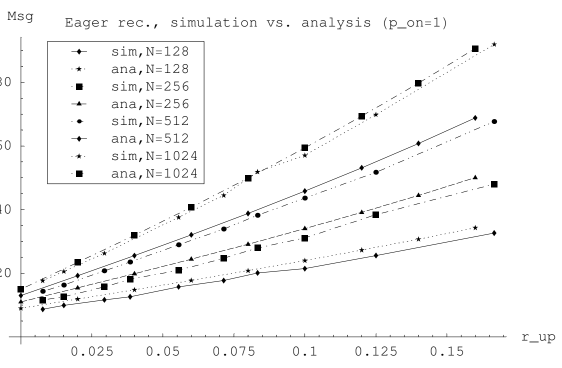
7 Analytical and simulation results

We implemented the algorithms described in this paper in order to verify the analytical models by simulation and to demonstrate their scalability. Due to space limitations, we primarily report on results where $p_{on} = 1$, i.e., peers only change their IP addresses but stay available, apart from some analytical results for the more general case of $p_{on} \leq 1$. As in the analysis, we do not consider the cost of establishing a quorum and refer the reader to [11].

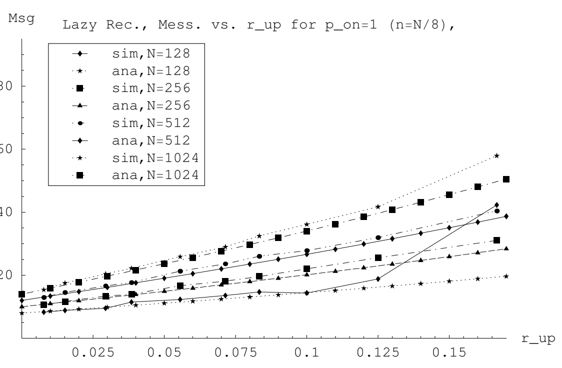
For the underlying storage, we construct P-Grid overlay networks of variable population sizes $N = 128, 256, 512,$ and 1024 peers. We set the average replication factor to 8, such that the paths have an average length of 4, 5, 6 and 7 ($\log_2 n$ where $n = N/8$). At each depth of the routing tables we maintain $r = 4$ references. The simulation is performed in rounds, where in each round we issue a random query at a random peer with probability $1 - r_{up}$, and with probability r_{up} we change the identity of a random peer and perform the necessary update. To reach a dynamic equilibrium state, we run a sufficiently large number of rounds (increasing from $25N$ to $100N$ for decreasing values r_{up}) and take the mean over the last $25N$ rounds to obtain values of the measured parameters.

For both the lazy and eager recursive query mechanisms we could show that the performance of the simulated system matches the predicted performance very well. We summarize the results in Figure 3, where we show the number of messages generated as a function of the frequency of updates, both when using the eager and the lazy algorithm.

We see that the message cost in the simulation is slightly higher than the predicted cost. This is due to the variation of the staleness of references. Since the message cost depends non-linearly on the staleness, variations inevitably lead to an increase as opposed to our average case analysis.



(a) Messages vs. r_{up}, eager algorithm



(b) Messages vs. r_{up}, lazy algorithm

Figure 3: Simulation results

We observe that for the lazy algorithm for small N ($N = 128$) the model starts to break down when the value of r_{up} grows. This is so because for very small networks combinatorial effects such as cycles and deadlocks, which are not accounted for in the analysis, start to take effect, thus making the model inaccurate. On the other hand, we see that for a larger network population the predictions are increasingly accurate, as it is the case for any statistical model. The message cost scales gracefully. Thus for large networks our analytical model can be used to reliably predict its performance. We also observe that the lazy algorithm overall consumes slightly fewer messages than the eager algorithm.

Figure 4 shows the analytical predictions and the observed p_{rec} values from simulations for varying r_{up}, which may also be used to verify the accuracy of the analytical model. For N=128, the probability of recursion p_{rec} starts to increase dramatically, which also implies increase in message cost, and the simulation results deviate from the analytical predictions. Even for moderately large network sizes (N = 256 and higher), the results obtained from both simulations and analysis match well, which shows that the independence assumptions, and statistical results of the analysis are correct, once the system has a moderately large peer population. This is as expected from any statistical model.

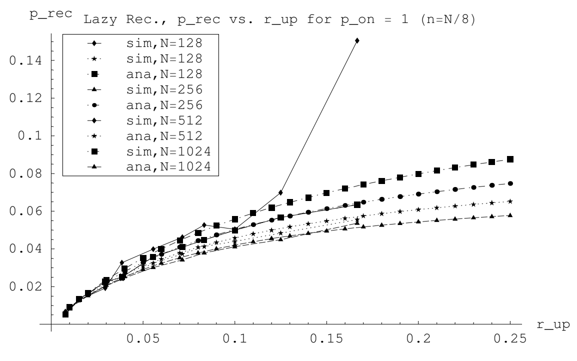


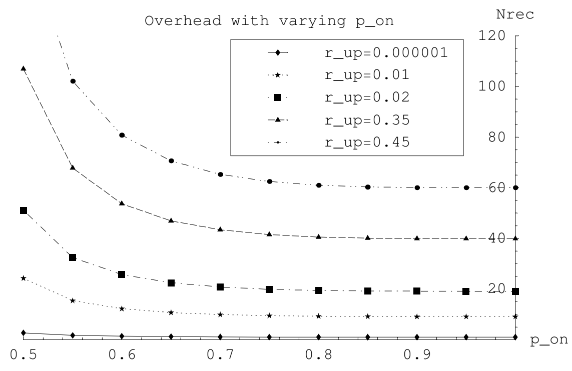
Figure 4: p_{rec} vs. r_{up}, lazy

We use the analytical model to further explore the properties of the system in dynamic equilibrium. In Figure 5(a) we show how N_{rec} varies with varying p_{on} for any r_{up} value when using lazy recursion. We observe that the algorithm is very robust, and the message overhead is stable for a wide range of p_{on} values. This is so because for lower p_{on} values, even fewer stale entries render the routing table unusable, and trigger recursions. The intuitive expectation will thus be an increase in N_{rec}. However, such recursions also have the effect of quickly repairing the routing table, such that fewer recursions are triggered later. These two opposite effects balance, hence the wide stretch of p_{on} values where the overhead stays stable.

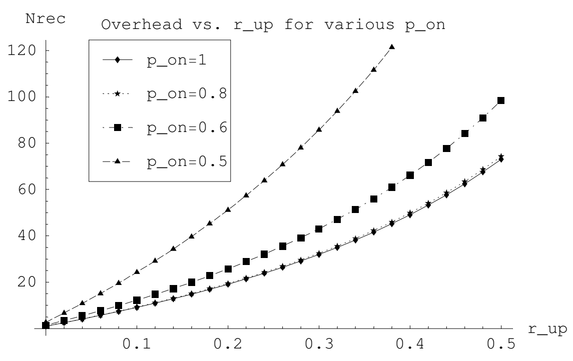
Figure 5(b) shows how the overhead varies with increasing network dynamics (increasing r_{up}), and we observe that it is more sensitive to r_{up} at lower p_{on} values.

While the use of recursion almost eliminates failures, tolerating even very low p_{on} values and moderately high network dynamics (high r_{up}), the incurred effort may not be affordable in a realistic network. In Figure 5(c) we thus provide contour maps corresponding to N_{rec} values, with p_{on} in the X-axis and r_{up} in the Y-axis. The interpretation of the plot is that if a system (participating peers) is willing to incur an N_{rec} fold increase of effort per query with respect to the ideal case ($p_{on} = 1$ and $r_{up} = 0$), the network will operate for all p_{on}, r_{up} combinations below the curve, with the success probability being 1. If the system is unwilling to use more than N_{rec} effort and if the system operates in the region above the curves of Figure 5(c), there is a non-zero failure probability, which starts increasing with the increase of distance from the curve. Figure 5(c) thus captures two important tradeoffs in the system. The first tradeoff is that of efficiency versus probabilistic success guarantee of queries. The second tradeoff is the system's resilience against the two "demons" of the network, the network dynamics r_{up} versus average availability of peers in the network p_{on}.

Finally, we analyze the dependency on varying values of p_{on}. In Figure 6 we show the number of messages for a fixed $r_{up} = 0.2$ and path length 5. We see that for networks with more peers being online, the lazy strategy is advantageous. The tradeoff is that the lazy strategy collapses earlier. Thus the eager strategy is more resilient in the case of low availability. This suggests that combined adaptive strategies with various degrees of "eagerness" are an interesting approach for environments with varying online characteristics.



(a) N_{rec} vs. p_{on}



(b) N_{rec} vs. r_{up}

(c) Contour maps for N_{rec}

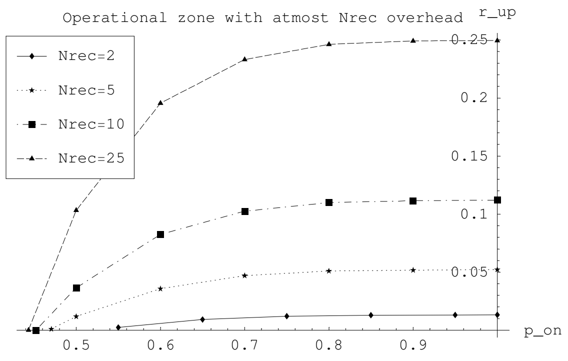


Figure 5: Analytical results

8 Related Work

For unstructured P2P systems such as Gnutella [9] and hierarchical systems such as FastTrack-based (http://www.fasttrack.nu/) systems like Kazaa, dynamic IP addresses are less of a problem. For example, Gnutella builds an unstructured graph of peers in which each peer typically has 4 permanent connections to other peers. In the case that a connection drops, a peer simply tries to reconnect or tries to connect to another peer, it has learned about implicitly through Gnutella's routing process. Since no routing tables are maintained no inconsistencies can occur. However, this comes at the expense of very high network traffic. In hierarchical systems "routing tables" (in

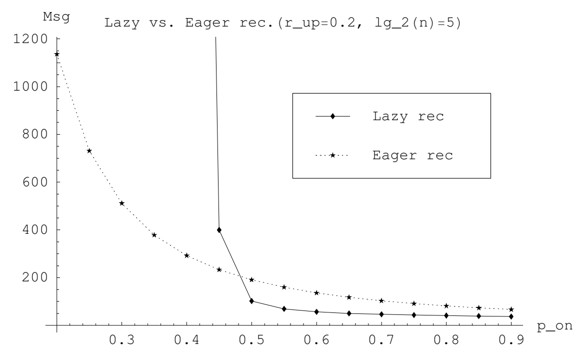


Figure 6: Dependency on p_{on}

fact they are rather simple) can become inconsistent but their scope is limited, so the effect can be compensated easily with existing methods.

Freenet [7, 8] suggests the use of a third-party DNS service that allows the peer to update its name-IP mapping in special DNS domains. However, this introduces a dependency on a third-party service and an element of centralization into the architecture which is in contrast to the principles of decentralization to ensure scalability.

The approaches of Chord [29], DKS [6] and Pastry [27] are the closest ones to ours and have already been discussed in detail and related to our work in Section 1. Their approaches are complementary to our work and further comparative studies on their performance are required.

The approach presented in [15] extends Tapestry [30] to address the joining and leaving of peers. In absence of self-healing, network maintenance is very expensive in this approach in terms of traffic (multicast-based partial flooding of the network), and there are no results on how the approach will cope with the degree of network dynamics (p_{on} and p_{dyn}).

DNS's original specification was extended by several RFCs (RFC2136, RFC2846, RFC2535), so that in theory it could maintain dynamic IP addresses through secure nameserver updates. However, this is very heavy-weight, requires very elaborate configurations, and is not intended for allowing a large number of peers to change the DNS database. Also an alternative DNS-based approach presented in [16] is still way too heavy for P2P systems and does not address security and unique identity of peers. To some extent our approach for recursive queries is similar to DNS's recursive lookup strategy which also updates caches during a name lookup. However, DNS's strategy is much simpler since DNS servers change their IP addresses very infrequently and thus the tree structure is basically static which simplifies routing a lot. Additionally, the number of participating DNS servers is considerably lower than the number of peers in a P2P system, the depth of the DNS tree is small, and, in contrast to DNS, our approach is self-contained, i.e., does not depend on a third-party infrastructure.

Other work using a DNS-like hierarchy without a single root has been done in the context of decentralized identification, such that some peers authorize other peers to use particular resources they provide. Any peer can authorize other peers to use its local resources as well as possibly delegate the authority to authorize other peers to do so. Systems following this approach are [10] and [5] which are based on [29].

For security we devise a self-organizing public key infrastructure [11] which is comparable to PGP [13] which uses a similar, decentralized approach. PGP uses transitivity of trust, whereby, if P_A trusts that K_B is P_B's public key, and also relies on P_B (personally determined) 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 it. The strength of such chains is determined by its weakest link and thus highly vulnerable. So [25] suggests to include multiple paths which, however, still offers only limited liability due to intersecting paths. Thus additionally authentication metrics [26] are required to quantify the reliability of such multiple paths. This approach, however, is heavy-weight, for example, finding multiple paths, loads the network considerably and both the multiple paths and the metrics need to be evaluated at each peer, and thus the effort is not shared. In contrast to that, our approach does not suffer from these problems at all. In P-Grid random (independent) peers replicate identity information (mappings) and thus our approach does not incur any costs in finding independent paths, the use of a quorum mitigates malicious behavior, and storage and search costs are distributed among the peers and require substantially lower computing and network resources. Additionally, since a subset of peers (to which searches are routed efficiently) are responsible for a given key, it is also simple to revoke or update mappings which is superior to PGP-based schemes. Further discussions are provided in [11].

9 Conclusions

This paper described a decentralized, self-maintaining, light-weight, and secure directory service based on secure identification. We have demonstrated that our algorithm is robust and applicable in unreliable environments such as current peer-to-peer systems and that it operates well, even if we assume low online probabilities. Our approach has five major contributions: (1) We separate identity from network properties and thus introduce the concept of logical independence into overlay networks, (2) we provide a general approach to identify entities and to bind arbitrary information to them, (3) we demonstrate that the approach does not corrupt structural properties of the used P2P system and retains existing knowledge and semantics, (4) we explicitly address security to guarantee the correctness of identities, and (5) we have explored a P2P system's dynamic resilience in the presence of changes in the underlying network, in contrast to other works that have only addressed static resilience of P2P systems [14, 18]. The service is based on the P-Grid P2P system and applied in P-Grid itself to mitigate the problem of dynamic IP addresses. To prove the efficiency and applicability of our approach we have provided an analytical model for the dynamic equilibrium case and have evaluated our algorithm based on this model. Additionally, we have provided simulation results to verify the correctness of the model. Our infrastructure 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. The presented approach is self-maintaining since it requires only little manual configuration and then operates without requiring further explicit maintenance mechanisms, since this is accomplished implicitly by using the network.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS)*, 2001.

[2] K. Aberer, A. Datta, and M. Hauswirth. The Quest for Balancing Peer Load in Structured Peer-to-Peer Systems. Technical Report IC/2003/32, EPFL, 2003.

[3] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the 10th International Conference on Information and Knowledge Management (2001 ACM CIKM)*. ACM Press, 2001.

[4] K. Aberer, M. Hauswirth, M. Punceva, and R. Schmidt. Improving Data Access in P2P Systems. *IEEE Internet Computing*, 6(1), 2002.

[5] S. Ajmani, D. E. Clarke, C.-H. Moh, and S. Richman. ConChord: Cooperative SDSI Certificate Storage and Name Resolution. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, number 2429 in LNCS. Springer, 2002.

[6] L. Onana Alima, S. El-Ansary, P. Brand, and S. Haridi. DKS(N,k,f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, 2003.

[7] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1), 2002.

[8] I. Clarke, O. Sandberg, B. Wiley, and T. 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*, 2001.

[9] Clip2. The Gnutella Protocol Specification v0.4 (Document Revision 1.2), Jun. 2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[10] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a Peer-to-Peer Lookup Service. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, number 2429 in LNCS. Springer, 2002.

[11] A. Datta, M. Hauswirth, and K. Aberer. Beyond "web of trust": Enabling P2P E-commerce. In *IEEE Conference on Electronic Commerce (CEC'03)*, 2003.

[12] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

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

[14] K. Gummadi, R. Gummadi, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of the ACM SIGCOMM*, 2003.

[15] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed Object Location in a Dynamic Network. In *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002.

[16] P. Huck, M. Butler, A. Gupta, and M. Feng. A Self-Configuring and Self-Administering Name System with Dynamic Address Assignment. *ACM Transactions on Internet Technology*, 2(1), 2002.

[17] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[18] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, 2003.

[19] R. Mahajan, M. Castro, and A. Rowstron. Controlling the Cost of Reliability in Peer-to-Peer Overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, LNCS. Springer, 2003.

[20] M. Aris Ouksel and Otto Mayer. A robust and efficient spatial data structure: The nested interpolation-based grid file. *Acta Informatica*, 29, 1992.

[21] C. E. Perkins, B. Woolf, and S. R. Alpert. *Mobile IP Design Principles and Practices*. Prentice Hall PTR, 1998.

[22] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distribute d Environment. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.

[23] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load Balancing in Structured P2P Systems. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, LNCS. Springer, 2003.

[24] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM*, 2001.

[25] M. K. Reiter and S. G. Stubblebine. Resilient authentication using path independence. *IEEE Transactions on Computers*, 47(12), 1998.

[26] M. K. Reiter and S. G. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2), 1999.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, 2001.

[28] P. H. Salus, editor. *Big Book of IPv6 Addressing RFCs (Big Book)*. Morgan Kaufmann, 2000.

[29] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM*, 2001.

[30] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-are location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

Karl Aberer is a full professor at EPFL since September 2000. There he is heading the Distributed Information Systems Laboratory of the School of Computer and Communications Sciences. His main research interests are on distributed information management, P2P computing, semantic web and the self-organization of information systems. He received his Ph.D. in mathematics in 1991 from the ETH Zürich. From 1991 to 1992 he was postdoctoral fellow at the International Computer Science Institute (ICSI) at the University of California, Berkeley. In 1992 he joined the Integrated Publication and Information Systems institute (IPSI) of GMD in Germany, where he became manager of the research division Open Adaptive Information Management Systems in 1996. He has published more than 80 papers on data management on the WWW, database interoperability, query processing, workflow systems and P2P data management. Recently he has been PC chair of ICDE 2005, DBISP2P 2003, RIDE 2001, DS-9, and ODBASE 2002. He is associate editor of SIGMOD RECORD and member of the editorial board of the VLDB Journal and Web Intelligence and Agent Systems.

Anwitaman Datta is a research assistant from the Distributed Information Systems Laboratory at the Swiss Federal Institute of Technology in Lausanne (EPFL). He holds a B.Tech. in Electrical Engineering from the Indian Institute of Technology (IIT Kanpur). Prior to joining EPFL, he worked at Transwitch India as an associate member of technical staff. Anwitaman is currently pursuing a Ph.D. under the supervision of Prof. Karl Aberer. His research interests include peer-to-peer systems, ad-hoc networks, epidemic algorithms and self-organization and resilience of decentralized systems. He is a member of the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS).

Manfred Hauswirth is a senior researcher at the Distributed Information Systems Laboratory of the Swiss Federal Institute of Technology in Lausanne (EPFL) since January 2002. He holds an M.S. (1994) and a Ph.D. (1999) in computer science from the Technical University of Vienna. Prior to his work at EPFL he was an assistant professor at the Distributed Systems Group at the TU Vienna where he still lectures courses on distributed systems. His main research interests are on peer-to-peer systems, distributed information management, self-organizing systems, semantic web, and security in distributed systems. He has published numerous papers and co-authored a book on distributed software architectures. He has served in numerous program committees of international scientific conferences and is a member of IEEE and ACM.