# Identifying peers using a self-contained directory*

Karl Aberer, Anwitaman Datta, Manfred Hauswirth

Distributed Information Systems Laboratory

Ecole Polytechnique Fédérale de Lausanne (EPFL)

CH-1015 Lausanne, Switzerland

{Karl.Aberer, Anwitaman.Datta, Manfred.Hauswirth}@epfl.ch

Contact Information: phone +41-21-693 4679; fax +41-21-693 8115

**Abstract**

PGP's "web of trust" approach provides basic identification facilities. One benefit of this approach is that it tries to lower the infrastructure requirements such that, for example, no certification authorities are required. In this paper we also pursue this objective but take it even further. We present a self-contained P2P-based directory that provides similar guarantees for identification as PGP, and like PGP requires no dedicated, third-party infrastructure. The participating peers provide the infrastructure along with their normal operation. But unlike PGP it does not require out-of-system mechanisms to disseminate the public key. Instead we exploit properties of the underlying P2P system to ensure correct key publication and retrieval by probabilistic quorums. We argue that this is an efficient and more reliable mechanism compared to web-of-trust approaches.

**Keywords:** Peer-to-peer, public key infrastructure, PGP

* 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

Current P2P systems fall into one of three groups: hierarchical, for example, Kazaa, unstructured, for example, Gnutella, and structured, for example, CAN, P-Grid, Chord and Pastry. At the moment only the first two groups are wide-spread. Structured P2P systems are still subject to research.

The key drawback of current hierarchical systems is their lack of scalability. Unstructured systems which are completely decentralized solve this, but at the price of high bandwidth consumption (flooding, constrained broadcast). Moreover, it is not clear how such unstructured P2P systems can support other services than file sharing. This explains the advent of structured P2P systems which at the moment mainly follow variants of the distributed hash table (DHT) approach. DHT systems delegate parts of the overall key space to particular peers and use proximity routing for queries. This avoids flooding and hence drastically lowers bandwidth consumption, latency, and computing requirements of the peers.

However, the tight association of key space with peers makes it important to identify the peers correctly in the routing process, particularly if peers may have dynamic IP addresses. Also Byzantine or malicious peers could collaborate to launch DOS attacks or try to impersonate others. Apart from just making P2P systems and their routing reliable and resilient, establishing peer identity can also provide security guarantees for the applications that are developed on top of such infrastructures (e-commerce, grid computing, web services). We can achieve authentication and other security aspects like authorization or reputation management if we can provide a secure mechanism to identify the participating peers. Traditionally, for identification, either username/password schemes or public key authentication have been used. Unfortunately username/password based schemes are unsuitable for P2P systems and standard public key schemes require directories and certification authorities which introduces centralization through the back door.

From the experiences with PGP's [3] web-of-trust model we know that it is possible to sustain a decentralized public key infrastructure (PKI). PGP depends on the small world phenomenon of social acquaintance that is observed in its trust (certificate) graph. In the web-of-trust model, a participant of the system (peer) knows the public keys of some other peers, and considers this knowledge sacrosanct. It also relies on some of these peers (with possibly varying degree of trust) to certify the public keys of other peers. The knowledge of peers' public key is obtained by finding a path in the peer acquaintance/trust graph. This forms the "web-of-trust:" If $P_A$ trusts that $K_B$ is $P_B$'s public key, and also relies on $P_B$ to certify a third party's public key, then $P_A$ will believe in $K_C$ being $P_C$'s public key if $P_B$ certifies it.

Approaches of this group, however, have certain peculiarities which make them unsuitable for a decentralized PKI as we envision it:

- Path discovery is inefficient because the effort is not shared, and has high, unbounded latency, since paths are explored by random walks, and thus do not exploit the graph's structural properties as

structured P2P systems would do.

- As the name PGP (Pretty Good Privacy) suggests, web-of-trust approaches are designed for privacy purposes. To completely believe in a person's public key, the information has to be obtained out-of-band. Otherwise only certifications provided by persons known in real life are accepted. Thus, identification is assumed implicitly. Particularly in the absence of any quantifiable guarantee, it is premature to say that the web-of-trust model will translate well from its traditional focus of confidential communication in email to the development of a public key infrastructure in a P2P system, where peers will interact with totally unknown peers, and peer identity is of principal concern.

- Web-of-trust models fail to use the collective knowledge of the whole society but use only information available within a small number of transitive hops in the trust graph. However, the ultimate purpose of a decentralized PKI should be to provide a way to establish identity of unknown peers beyond reasonable doubt. Web-of-trust models cannot guarantee that since transitive paths are not guaranteed.

- Finally, since the trust on other peers' certification is essentially ad-hoc, a web-of-trust is susceptible to the treachery of even one or very few trusted peers.

These reasons motivated us to explore the possibility to use a structured P2P system to store public keys, and use a quorum based approach to reliably find public keys of other peers, and hence establish peer identity. The question to investigate was whether a P2P system could provide a decentralized PKI without relying on any specialized roles like that of a trusted third party nor using out-of-system support like exchanging public keys offline. Besides this we had to take into account additional challenges of realistic P2P systems: Peers are often offline, their IP address (DHCP) may change, and peer may behave maliciously. So even for a P2P system's basic routing purposes, identification is already required.

Though this introduces a cyclic dependency—the system intended for identification depends on identification itself—we demonstrate analytically and with an example that a P2P system maintaining a decentralized PKI directory for its participating peers can indeed be realized and cannot only tolerate certain degree of inconsistency, but also has self-healing properties.

Our system is based on the P-Grid P2P system [1] (http://www.p-grid.org/). However, our approach is generic enough, such that it is realizable in other structured P2P systems as well. P-Grid is a completely decentralized and self-organizing index structure based on a distributed prefix tree that is constructed through a distributed, randomized interaction mechanism. Simplifying it is a binary tree distributed among the the participating peers where each peer (actually it is a set of replicating peers) is responsible for one path of the overall tree, i.e., peers are the leaves in this tree. Navigating a query in this tree is done by forwarding

it from peer to peer according to the peers' local routing tables until one of the responsible leaf peers is reached.

Searching in P-Grid is efficient and fast ($O(\log(n))$, where $n$ is the number of leaves). The P-Grid construction process guarantees that peer routing tables always provide at least one path from any peer receiving a request to at least one of the peers holding a replica of the requested index information, so that any query can be satisfied regardless of the peer queried. Additionally, it guarantees that a sufficient number of replicas exists for any path and that the peers representing a certain path also know their replicas. In contrast to most other structured P2P approaches (CAN is an exception), peers in P-Grid can negotiate and choose the key space they want to be responsible for irrespective of their own logical identifier (self-organization, separation of concerns). This implies better load balancing properties and resilience against attacks.

## 2 Locating and identifying peers

In P-Grid as in any other structured P2P system dynamic IP addresses (e.g, because of DHCP) are a key problem to be solved. The routing tables must be kept up-to-date with the current IP addresses of the peer population. In the query routing process the following problems can occur:

1. Peer $p$ goes offline and a different peer $q$ gets $p$'s IP address. The other peers wanting to route a query to $p$ must be able to detect this change.

2. A peer goes online again with a new IP address. The other peers must be able to detect this, check identification, and update their routing tables accordingly.

While this may sound simple, it is in fact a rather complex problem if it is to be solved inside a system without the help of trusted third parties because P2P systems are highly decentralized and no standard mechanisms exist to communicate such changes efficiently for large-scale systems (a comprehensive discussion of this problem is provided in [4]). MobileIP or IPv6 would solve this problem but they are not in place and it is unclear when they will be.

So, in fact this basic problem which needs to be solved for routing purposes is already a identification problem. We have to identify peers by universally unique identifiers and use these identifiers in the routing tables. For really contacting a peer in the routing process, however, we have to map this identifier onto the peer's current IP address. Informally our solution works the following way: If a peer's address changes it inserts its current ID-to-IP mapping into P-Grid so that other peers can retrieve it. The insertion uses a probabilistic quorum to ensure that the mapping is stored consistently at a number of replicating peers. Using the same method a peer also inserts its public key. We also assume that if a peer has interacted

with another peer in the past which will frequently be the case because that is required by the P-Grid construction process, then it has the peer's public key. While maintaining the P-Grid, the public key is needed to verify that it is the same peer with which a particular peer had interacted in the past. To prohibit malicious use all requests and answers additionally are signed. Finding an up-to-date ID-to-IP mapping means querying P-Grid using a peer's ID as the key. The correctness of the query result will again be verified by a probabilistic quorum.

This approach, however, has a subtle catch: On the one hand we use the P-Grid to index arbitrary information (application index) and on the other hand we use the same P-Grid as a directory service for also storing the ID-to-IP mappings for its participating peers, that is needed by the P-Grid itself to perform routing. In other words: Queries in P-Grid require routing decisions and during routing, queries to the same P-Grid for mappings occur. The approach is self-contained, but a legitimate question to ask is "Will it work in practise, since we are producing recursive queries (hen/egg problem)?". In fact, the good news is: Under realistic assumptions, using cached ID-to-IP mappings, and trying to update the cache entries upon routing failures, the approach works fine and exhibits acceptable performance [4].

Let us now get back to the problem of identification. If we can use P-Grid as a self-contained directory system for such ID-to-IP mappings we can also use it as a general, self-contained identification infrastructure: Along with the ID-to-IP mappings we can also store a peer's public key which can then also be retrieved with the same probabilistic guarantees as the ID-to-IP mapping and hence be used as the basis for identification of arbitrary peers, with similar guarantees as in PGP.

In the following sections we will give a more detailed look at the algorithm, give an explaining, concrete example, and provide some results from the analytical model we developed [4] to justify that our quorum-based P2P-PKI works despite the hen/egg problem mentioned above.

# 3 Basic identification algorithm

Upon installation each peer $p$ generates a universally unique id ($uuid$) by applying a cryptographicly secure hash function to the concatenated values of the current date and time, the peer's current IP address $addr_p$ and a large random number. For the algorithm we distinguish 3 phases: bootstrap, peer startup, and operations phase.

During bootstrap $p$ once generates a private/public key pair and inserts a mapping between its $uuid$ and its current IP address $addr_p$ along with its public key into P-Grid (inserts and updates are done according to the algorithm presented in [2]). This request is signed with the private key of $p$ and includes a timestamp to prevent replay attacks. If it is discovered during this step that a mapping for the proposed $uuid$ already exist (though this is very unlikely) a new $uuid$ is computed and the step is repeated. The insertion is

considered successful if a certain minimum number of replicas have confirmed to have stored the mapping (the responsible replicas are selected according to the $uuid$). It is thus implicitly ensured that the correct public key is replicated. This means other peers can now query P-Grid for the mapping and for the public key of $p$ and use a quorum to check whether they can rely on the result. The retrieved public key can be stored locally and used for authentication purposes in the future.

During every peer startup following the bootstrap, $p$ then checks whether its IP address has changed. If so, $p$ inserts into P-Grid its present ID-to-IP mapping (signed). Replay attacks are prevented by timestamps. From the properties of DHTs (key association) the relevant replicas responsible for storing the ID-to-IP mapping already have $p$'s public key.

These steps ensure the maintenance of correct ID-to-IP mappings in the P-Grid. In the operations phase the algorithm works as follows: $p$ needs to contact peer $q$. If it has not interacted with $q$ earlier and it does not have a cached ID-to-IP mapping and $q$'s public key, it queries the P-Grid directory using $q$'s $uuid$ as the key. As a result it will receive a number of signed answers from a number of replicas storing the mapping. If a certain quorum of the received public keys/mappings are identical and the signatures are correct, $p$ assumes that the mapping is correct and store $q$'s key for further requests. However, it still may be that $q$ just went offline and somebody else now uses $q$'s IP address (legitimately or not does not make a difference here). To cope with that $p$ uses a standard challenge/response using random numbers.

Having $q$'s public key speeds up the whole process in the future because $p$ does not need to find a quorum anymore if it requests $q$'s up-to-date ID-to-IP mapping. Since all requests are signed and it already has $q$'s public key it can immediately check the results it receives by checking the signature of the received data (since $q$ signs its ID-to-IP mapping during insertion/update).

In the above description we have assumed implicitly that during the routing of the original query no additional (recursive) queries are necessary. However, it may occur that at some peers all required entries in its routing table have become stale and a recursive query to the same P-Grid needs to be issued. This may seem like a hen/egg problem but as we show in the following it can be solved.

So far the query strategy works as follows: A peer tries to reply to a query if it has the results stored locally, or else 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 fails and is abandoned.

We can modify this query handling algorithm to take into account the need for recursive queries: If a peer fails to contact any of the peers in its routing table, it initiates a new query to discover the latest ID-to-IP mapping of the peers in the routing table, and if such a peer can be located, the cache is updated and the original query is then continued (forwarded). To avoid query loops and guarantee termination of the recursive algorithm a time-to-live count is used.

Under the assumptions that a certain minimum number of peers is online and a certain number of

replicas exist, it has been shown that the recursive version succeeds with high probability [4]. However, the possibility exists that a query terminates without success. The only remedy then is to repeat the query at a later time assuming that the peer population has changed (and alike the routing tables) and the query has a chance to succeed then.

Abstracting from the underlying P-Grid (or any other DHT), the routing of the queries can also be viewed as routing/navigating in a trust graph (similar to the web-of-trust model of PGP), but instead of performing a random walk, it is an efficient proximity routing in the structured P2P system, thus guaranteeing existence of paths to the information as well as the efficiency of such searches. Reliability is provided through replication and probabilistic quorum.

When interacting with any arbitrary peer for a purpose other than maintaining the directory itself, the same directory can be used to store and query information like certificates of peers and support applications which require an identification scheme. For many purposes even the identification required for P-Grid routing itself will be sufficient. The additional advantage of using P-Grid is that no additional specialized infrastructure is required because the participants of a system implicitly provide the infrastructure in a P2P way. Thus no third party infrastructure has to be in place before the system can operate and use identification information, nor out-of-system mechanisms are used. Hence our approach achieves a self-contained P2P directory system.

The following Section 4 will now give a concrete example to make our theoretical considerations easier to understand and Section 5 will provide a condensed, informal version of the results of the proof presented in [4] to justify this claim.

# 4 Example

This section gives an example of how the query algorithm described above work in a concrete setting. While the P-Grid is in the state as shown in Figure 1(a), assume that $p_7$ receives a query $Q(01*)$.

The query may be for searching any information in P-Grid, either information about some participating peer or any other information. In this example situation, $p_7$ fails to forward the query to $p_5$ and $p_{14}$ because the cache entries for ID-to-IP mapping are stale (underlined entries in the routing table). An isolated query would fail immediately.

However, a peer that has failed to contact all of the peers in its routing table either because they are offline or because their physical address is different from what is cached locally, first tries to discover the latest addresses for the entries (peers) in its routing table. In our example we assume that information about $p_{id}$ is inserted and queried using a 4-bit representation of the "id." Routing in P-Grid is done by finding the longest common prefix of the query with the paths in the routing table (left of the colon) and forwarding

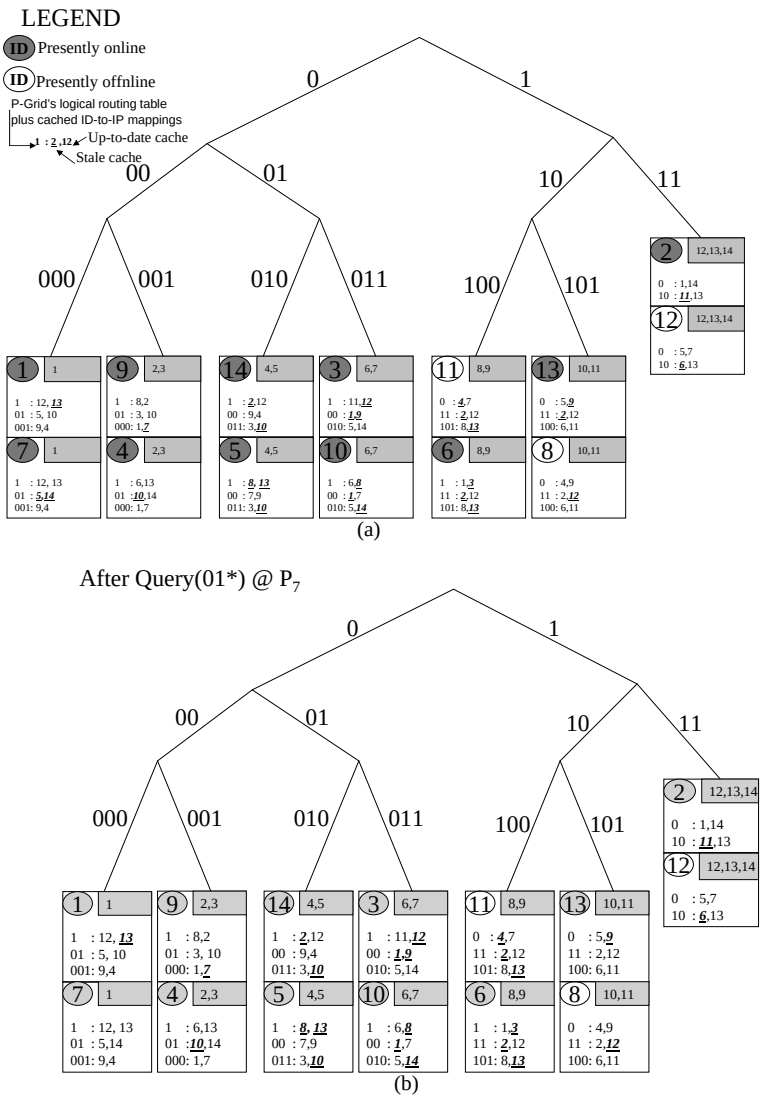



Figure 1: Before and after recursively querying P-Grid

the query to one of the peers given on the right side of the colon of that line.

So, $p_7$ initiates Recursive-Query$(p_5)$, i.e., $Q(0101)$, which needs to be forwarded to either $p_5$ or $p_{14}$. This fails again. $p_7$ may then initiate Recursive-Query$(p_{14})$, i.e., ($Q(1110)$), which needs to be forwarded to $p_{12}$ and (or) $p_{13}$. $p_{12}$ is offline (unshaded ovals), 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 $p_{13}$.

$p_{13}$ needs to forward $Q(p_{14})$ either to $p_2$ or $p_{12}$. It fails to forward it to $p_{12}$, since it is offline. Further, $p_{13}$ fails to forward it to $p_2$ because its cached entry for $p_2$ is stale. $p_{13}$ thus initiates another query, Recursive-Query$(p_2)$, i.e., ($Q(0010)$). It may initiate Recursive-Query$(p_{12})$ 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. Thus $p_{13}$ learns $p_2$'s latest physical address and updates the ID-to-IP mapping in its local cache. $p_{13}$ also starts processing and forwards the original Recursive-Query$(p_{14})$ 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).

Having learnt $p_{14}$'s current physical address, $p_7$ now forwards the original query $Q(01*)$ 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 that of $p_5$. Further, because of the initiated child queries, $p_{13}$ updates its cached address for $p_2$. The final state with several caches updated after the completion of $Q(01*)$ at $p_7$ is shown in Figure 1(b). This example thus demonstrates how a self-contained directory can work, tolerating not only a certain degree of inconsistent information, but also repairing some of them (self-healing).

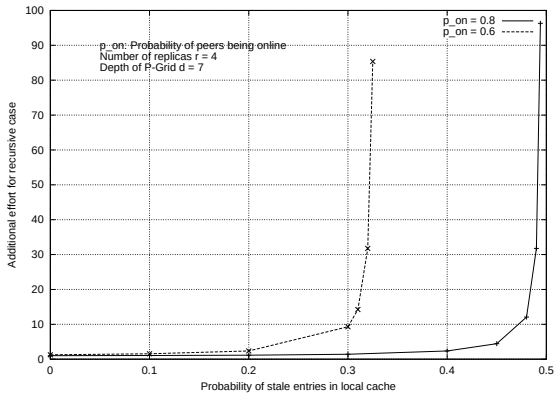
For the purpose of completeness, let us now consider P-Grid in terms of the peers' knowledge of each others' public keys. We elaborate how the P-Grid of Figure 1 evolves from an earlier state. Since P-Grid's access structure is constructed through random peer interaction [1], let us consider a time when peers $p_1$ and $p_9$, both responsible for path "00", meet and decide that $p_1$ will be responsible for "000" while $p_9$ will be responsible for "001" (or they can decide to be replicas). For routing to be reliable in P-Grid (as in any other DHT), if $p_1$ later receives a query for "001" it needs to forward it to $p_9$, and thus needs to locate and identify $p_9$. So during the P-Grid construction process, $p_1$ stores $p_9$'s public key. In addition $p_1$ locally caches the current ID-to-IP mapping of $p_9$. In the future, $p_1$ first tries to use this cached information, and only upon failure, issues a query to locate $p_9$ as elaborated in the above example, and verifies its identity using the locally stored public key that was obtained during P-Grid construction.

At a later time, $p_1$ may interact with $p_7$ and they decide to become replicas for "000". In that case $p_7$ may replicate $p_1$'s routing table and cache of ID-to-IP mappings. Additionally it may obtain $p_9$'s public key from $p_1$. However, if $p_7$ does not trust $p_1$'s certification for $p_9$'s public key, it can opt to query P-Grid for it (probabilistic quorum). This makes the approach more robust against individual or collaborative malicious behavior of peers, as compared to PGP. Since queries are efficient in P-Grid, the overhead of forming probabilistic quorum is insignificant to the random walks in the trust graph of web-of-trust based schemes.

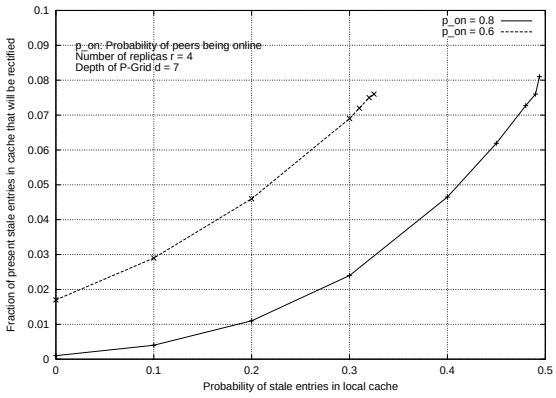
Thus, the P-Grid construction process ensures that each peer has the public key of other peers listed in its routing table which in contrast to the ID-to-IP mappings does not change.

# 5 Viability of a quorum-based P2P-PKI

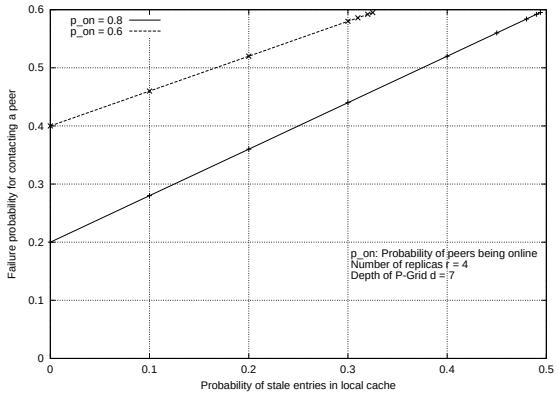
To evaluate the viability of such a self-contained directory, we developed an analytical model [4]. Figure 2 summarizes our observations for a realistic situation.



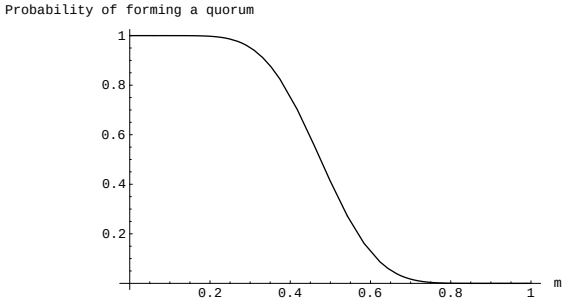

(a) Additional effort




(b) Self-healing




(c) Failure probability for contacting a peer




(d) Probability of successfully forming a quorum with varying percentage $m$ of malicious peers

Figure 2: Results for the algorithm to identify peers by recursive queries

The first three plots are for a balanced P-Grid of depth 7 with 4 replicas at each leaf. Thus there are 28 cached ID-to-IP mappings at each peer. The results are probabilistic and hence for a real system to show such statistical behavior, it would need a larger population of peers, such that there exists a moderately high number of replicas. Simulations presently underway also show self-healing characteristics, thus reinforcing the conclusions drawn from the analytical results.

If all local caches are consistent and all peers are online, then the expected number of messages to complete a query in P-Grid is $O(log(n))$ where $n$ is the number of leaves in P-Grid (even when the P-Grid

tree is unbalanced). However, when only a certain percentage of the peers is online (denoted by $p\_on$ in Figure 2) and physical address entries in the routing caches are possibly stale, the number of messages increases (by a factor) as shown in Figure 2(a). This is the overhead incurred in successfully locating a particular peer by recursive queries. As a consequence of such recursive queries, a fraction of the stale caches is updated (as was elaborated in the example). So if we assume only 60% of the peers being online and 25% of the cache entries being stale the performance still is very efficient, with a marginal overhead as compared to when all cached entries in P-Grid's routing tables at the individual peers are up-to-date.

Figure 2(b) quantifies the self-healing property, such that a fraction of the stale caches are updated while conducting the recursive querying. Here we observe that the fraction of cache entries that are updated increases with the percentage of stale entries. This is because isolated queries fail more frequently, initiating recursive queries. For example, with 60% of the peers being online and 25% of the cache entries being stale, about 6% of the stale entries will be rectified when there is a new query.

Recursive queries may still fail, either because the peer being looked for is offline or because the query enters a loop (which is detected). The resulting probability of failing to locate individual peers (which is lower bound by the probability of peers being offline) versus the percentage of caches being stale, is shown in Figure 2(c), for various values of $p\_on$. For example, if 80% of the peers are online and 20% of the caches are stale then the probability for not being able to contact a peer is around 37%.

Let us assume that there are 50 replicas (i.e, the number of nodes in each P-Grid leaf) and the probability of contacting a peer after recursively searching P-Grid is 40% (i.e., only 20 of them are expected to be found), then the probability of successfully forming a quorum of a minimum of 11 correct results (i.e., $> 50\%$) is shown in Figure 2(d), with varying percentage of malicious users who may individually try to impersonate, or collaborate to launch distributed denial of service attack against the directory system. As may be observed, the system can provide correct identification with very high probability (essentially 100%), tolerating attacks by up to 20% malicious peers.

These figures provide sufficient evidence to conclude that an efficient, resilient, self-contained, self-organizing and self-healing P2P directory is indeed realizable in a P2P system with predominantly cooperating participants.

# References

[1] Karl Aberer, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. Improving Data Access in P2P Systems. *IEEE Internet Computing*, 6(1), 2002.

[2] Anwitaman Datta, Manfred Hauswirth, and Karl Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *ICDCS*, 2003.

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

[4] Manfred Hauswirth, Anwitaman Datta, and Karl Aberer. Handling Identity in Peer-to-Peer Systems. Technical Report IC-2002-67, EPFL, 2002.