Attack Resistant Trust Metrics

Raph Levien, UC Berkeley

This document is a draft of Raph Levien's Ph.D. thesis in compact formatting, to minimize page count. It was generated from LaTeX sources July 31, 2004.

Abstract

This dissertation characterizes the space of trust metrics, under both the scalar assumption where each assertion is evaluated independently, and the group assumption where a group of assertions are evaluated in tandem. We present a quantitative framework for evaluating the attack resistance of trust metrics, and give examples of trust metrics that are within a small factor of optimum compared to theoretical upper bounds. We discuss experiences with a real-world deployment of a group trust metric, the Advogato website. Finally, we explore possible applications of attack resistant trust metrics, including using it as to build a distributed name server, verifying metadata in peer-to-peer networks such as music sharing systems, and a proposal for highly spam resistant e-mail delivery.

Chapter 1

Introduction

In today's world of open, decentralized networks, the question of *trust* is becoming increasingly relevant. Most existing Internet protocols implement a naive policy of providing a relatively limited set of services, but trusting all users with them. Given that a significant fraction of Internet users are not trustworthy, the inevitable result is spam, denial of service attacks, cracking, and a host of other ills far too familiar to legitimate Internet users.

A number of approaches have been developed and deployed over the years, with varying degrees of success. The most basic is the model of password-protected accounts. This mechanism is almost universally deployed, but suffers from serious limitations, including the need for servers to manage the accounts, the need for users to keep track of a large number of passwords, and the relative lack of security provided by this model. Thus, there has been a sustained interest in more sophisticated models.

One such approach is the *Public Key Infrastructure*, or PKI [6]. Briefly, a PKI consists of various *Certification Authorities* (or CA's) that issue *certificates* asserting a binding between a name and a public key. PKI's suffer from two fundamental problems: the lack of useful meaning in the PKI's underlying namespace, and the question of which CA to trust. Actual implementations of CA's have proved themselves not worthy of absolute trust. Further, as the number of CA's deployed scales up, the risk of any one of them being compromised scales accordingly. In part because of these two problems, PKI's have met with limited success at best.

Spurred on by these limitations, much recent attention has been focussed on the explicit encoding and evaluation of trust relationships. A pioneering work in this area is the PolicyMaker framework of Matt Blaze[5]. A particularly interesting branch of this work is the concept of *trust metrics,* which are the primary focus of this thesis.

Trust metrics are based on the principle of *local* encoding of trust relationships, but granting trust on a global scale. These assumptions closely parallel the philosophy of peer to peer networks[26].

While the literature is rich with design proposals for trust metrics, there has been relatively little analysis of how well they work in practice. In Chapter 2, we show that, under assumptions similar to the Internet, the entire category of "scalar" trust metrics fails to resist easily-mounted attacks. Attacks against these poor trust metrics dot the literature[31], and have been used to argue (incorrectly) that a centralized identity service is needed[13].

While the outlook for scalar trust metrics is indeed bleak, we propose a new class, which we call *group* trust metrics, so called both because the trust metric is very well suited to evaluating membership in a group, and because this evaluation is done over the entire group of nodes, rather than individually for each node. While the group trust metric cannot prevent individual hostile nodes from being accepted as group members, it can place strict bounds on the *number* of hostile nodes so accepted.

I built a community website for free software developers, called Advogato, that uses this group trust metric to determine who belongs to the community. Acceptance by the trust metric confers privileges to post articles and comments, and to edit project information. At the time of this writing, the site has been in operation for about 18 months, and has built a trust graph of over 1000 active nodes. Advogato is notable for the extremely low level of trolls, spam, and other forms of abuse common to bulletin board type systems, thus providing strong anecdotal evidence that the trust metric is effective. Advogato is described in more detail in Chapter 4.

One interesting application of trust metrics is secure registration and lookup of public keys bound to names, essentially the same problem addressed by Public Key Infrastructures (PKI). The attack resistant properties of the trust metric avoid the single point of vulnerability common to most PKI designs. We present a detailed design for such an attack-resistant name service in Chapter 5. The desire to build a "better PKI" was the original motivation behind the work of this thesis.

Because the group trust metric is effective in resisting attack, it has many other interesting applications. One such is an attack-resistant system for distributing metadata, for example opinions about songs. Attack resistance can be useful for ensuring high quality of the metadata, but becomes particularly important when there are financial implications to the metadata system, for example to identify recipients of voluntary donations to artists who post their music to the Net. We propose a design for an attack resistant metadata system in Chapter 6. This design combines ideas from both the Advogato group trust metric and the PageRank algorithm used in Google.

Another intriguing application for group trust metrics is to provide a spam resistant infrastructure for e-mail. Spam is a huge problem, causing lots of wasted time for many, and most e-mail users feel hopeless in the face of it. We present a partial design for a communications infrastructure resistant to spam, yet highly permeable to legitimate email. This design uses a capacity constrained flow network where "stamps" are the commodity of flow. Rather than explicitly computing a flow in the network, the flow results from local activity. When capacity is exhausted, e-mail is no longer deliverable. This design has many appealing properties, but several aspects remain as open problems, particularly scaling. Chapter 8 presents this design and discusses some open issues.

Chapter 2

Trust Metrics

In this chapter, we review the literature of trust metrics, present a quantitative framework for analyzing the attack resistance of trust metrics. Finally, we prove tight upper and lower bounds on the attack resistance of trust metrics given the usual scalar assumptions.

2.1 The simplest trust metric

In this section, we present the simplest possible trust metric, which forms the basis of other trust metrics in the literature.

There are three inputs to this trust metric: a directed graph, a designated "seed" node indicating the root of trust, and a "target" node. We wish to determine whether the target node is trustworthy.

Each edge from s to t in the graph indicates that s believes that t is trustworthy. The simplest possible trust metric evaluates whether t is reachable from s. If not, there is no reason to believe that t is trustworthy, given the data available.

In a cryptographic implementation, each node corresponds to a public key, and each edge from s to t corresponds to a digitally signed *certificate*. In the usual terminology, s is the *issuer*, t is the *subject*. The certificate itself is some string identifying t, along with a digital signature of this string generated by s.

The simplest trust metric is also the weakest with respect to attacks. If an attacker is able to generate an edge from any node reachable from the seed to a node under his control, then he can cause arbitrary nodes to be accepted. As the size of the reachable subgraph increases, the risk of any such attack increases as well. Thus, the primary focus in the literature is to present stronger trust metrics that (hopefully) resist attacks better, while still accepting most nodes that deserve trust.

Section 2.2, discusses a sampling of trust metrics previously published. Section 2.3, presents a quantitative notion of "attack resistance."

2.2 Survey of the literature

All trust metrics include the three inputs described above: the trust graph, the seed node, or *trust root*, and the target. Some trust metrics add more detail to these inputs. Trust edges may contain some conditions or restrictions, for example that t is himself trustworthy, but cannot be trusted to vouch for other nodes. In addition, the target may be a richer assertion than merely whether a certain node is trustworthy. For example, edges may additionally identify a maximum dollar value, and the target may be an assertion that the target node can be trusted with a transaction of some dollar value.

Finally, there may be additional policies to be enforced by the trust metric, or parameters supplied. Often, these parameters control the overall strictness or tolerance of the trust metric.

The most general form of "trust management" system is represented by PolicyMaker[5], as certificates and policies can represent arbitrary computations in Turing-complete language. Applications of PolicyMaker tend to focus on the language of assertions rather than trust computations over the graph, but the fully general nature of the system allows the the latter to be implemented.

Most trust metrics in the literature assume monotonicity. More precisely, if a monotonic trust metric accepts a node t as trustworthy when given a graph G, it will also accept node t when given a graph G' that contains all trust edges in G. A primary motivation for this assumption is scaling. In particular, it allows for highly efficient overall distributed architectures in which all participants need only access a small, local subset of the global trust graph. Further, under such an assumption, there is no special vulnerability to denial of service attacks. With the monotonicity assumption, if an attacker can withold trust edges (i.e. cause the verifier to evaluate a subset of the global trust graph), then it cannot cause nodes to be accepted other than those which would be accepted given the entire trust graph.

The simplest trust metric is clearly monotonic. Adding edges to the trust graph can only cause more nodes to be reachable.

Another relatively simple trust metric is similar to the simplest one, with the additional constraint that path lengths are bounded by some parameter k. Thus, all nodes within a distance of k edges from the seed are accepted. A variation of this trust metric is used in X.509 systems.

Perhaps the earliest published system resembling a modern trust metric is the Beth, Borcherding, and Klein[3]. This system contains a set of elaborate inference rules for deriving a "trustworthiness" value between 0 and 1, using both "direct trust" and "recommendation trust" relationships encoded on edges. However, further work by Reiter and Stubblebine[31] showed this trust metric to be no more secure than the simple reachability criterion described above. Note that the BBK trust metric is not monotonic.

A still earlier system is described by Tarah and Huitema[34], who suggest evaluating both certificates and "certificate paths." They suggest several possible simple trust metrics, including the length of the certification path and the minimum of involved trust values along the path. However, because it evaluates only paths, rather than general graphs, this system does not quite meet the definition of "trust metric".

Reiter and Stubblebine[30] presented the first trust metric with the ability to resist nontrivial attacks. Briefly, this trust metric counts the number of node-independent paths of bounded length from the seed to the target. Thus, both the path length and the threshold for the minimum number of such paths are tunable parameters of the metric. A simple characterization of its attack resistance follows easily: an attacker must add "bad" edges to nodes under his own control to at least as many nodes as the threshold for the number of independent paths. Even though evaluation of this trust metric is an NP-complete problem, the experimental PathServer implementation seems to perform reasonably well in practice.

Maurer presented an intriguing trust metric based on randomized experiments[21]. Very briefly, each edge in the network has an associated probability. The result of the trust metric is the probability that the target is reachable from the seed in a subgraph of the original graph where each node is present with the probability given in the original graph.

Both implementation and analysis of the Maurer trust metric are challenging. In addition, we present an analysis below (Section 2.6) suggesting that the Maurer metric is vulnerable to an easily-mounted attack. To do so, we will first need to prepare a framework for quantitatively evaluating the attack resistance of a trust metric.

2.3 Framework for analysis

What does it mean for a trust metric to be attack resistant? In this section, we present a quantitative framework for this question.

In any given attack, we assume that the attacker can add or delete edges from the legitimate part of the trust graph, pointing to arbitrary nodes. These edges may point directly to nodes under the attacker's control, or perhaps to other good nodes, in order to fool the trust metric.

We assign a *cost* to each such attack. A typical cost metric is to count the number of edges added. Assuming an attack of a given cost, what is the highest number of bad assertions the attacker can force to be accepted? If this number is limited, the trust metric is attack resistant. If it can grow to the same order as the number of good assertions even for a fairly low cost attack, then the trust metric suffers from catastrophic failure and is not attack resistant.

2.3.1 Cost metrics: node vs edge attacks

We consider two cost metrics. The simplest is to count the number of edges added. Another useful metric is to count the number of "attacked " nodes with added outedges, and to assume that any such node may have an arbitrary number of edges added. As we will see later, the attack-resistance properties of different trust metrics behave differently under these two cost assumptions. Note that, for any given attack, the number of nodes counted is no greater than the number of edges counted.

An edge attack corresponds to fooling a victim into generating an edge. In many cases, mounting such an attack is straightforward. For example, the attacker may send a forged email pretending to be a friend of the victim, asking for a cert. Many such electronic means of transmitting key material suffer from lack of authentication.

Similarly, a node attack corresponds to taking over the victim's ability to create certificates. Such an attack is generally harder than an edge attack, but still feasible. For example, anyone with physical access to the victim's machine can recover the private key used to sign certificates, for example by using a keyboard sniffer to recover the encryption key used to protect the private key.

Edge attack: number of certs. Corresponds to fooling the victim once.

Protection against node attack is a stronger property than protection against edge attack. Attack of a single node can correspond to an arbitrary number of edges.

Another factor in trust metric: how many certs are needed? At simplest, indegree of target node.

We analyze two classes of attacks: one in which the attacker is able to choose the victims, and another in which the victims are chosen randomly. The former class of attack is at least as effective as the latter, and, not surprisingly, we find that it in most cases it is considerably more so. This result parallels the literature in scale-free networks, in which removing the "hubs" in a scale-free network with hub-and-spoke topology is far more effective in fragmenting the network than simply removing random nodes.

2.4 Analysis: upper bounds

Strict upper bounds on effectiveness of trust metric. For node attack, if n nodes are attacked, entire system falls, where n is minimum indegree of target node.

Proof: attacker chooses target that is accepted (with minimum indegree), and chooses its predecessors as victims. Since indegree is n, attack is n nodes. Removing the original target, the graph is identical to the one in which the original target is accepted, so the trust metric can't distinguish.

For edge attack, if n^2 nodes are attacked, entire system fails. Choose target as above, then spoof inedges of predecessors of target.

2.5 Analysis: lower bounds

This section analyzes existing trust metrics.

Present simple flow-based metrics, and prove (using min-cut) that they almost exactly meet the upper bounds given above.

2.6 Analysis of Maurer's trust metric

Analysis of Maurer: follow [20].

2.7 Discussion

Several conclusions follow from the analysis presented in preceding sections. First, the scalar trust metrics place a very low upper bound on the attack resistance possible to achieve. Second, we demonstrate that simple trust metrics based on maximum network flow meet these upper bounds. Thus, within these assumptions, there is little if any room to design improved trust metrics.

In particular, analysis of Maurer's trust metric shows its attack resistance to be both disappointing and equivalent to much simpler metrics.

A reasonable conclusion, then, is that it is the monotonicity assumption itself which is flawed, and that it is not possible to achieve good attack resistance by verifying a small, local subset of the trust edges comprising the global trust metric. Indeed, in the next chapter, we show that by relaxing the monotonicity assumption, a reasonably simple trust metric also based on maximum network flows can acheive dramatically better attack resistance.

Chapter 3

Group Trust Metrics

This chapter presents the concept of *group trust metrics,* which have significantly better attack resistance properties than trust metrics under the scalar assumption.

The key feature of a group trust metric is that it calculates a trust value for all the nodes in the graph at once, rather than calculating independently the trust value independently for each node. Thus, a successful attack causing one bad node to be trusted need not necessarily result in catastrophic compromise of the trust metric. Indeed, in this chapter we will present a simple trust metric, also based on max-flow over the trust graph, which has significantly better attack resistance than the theoretical best-case for scalar trust metrics.

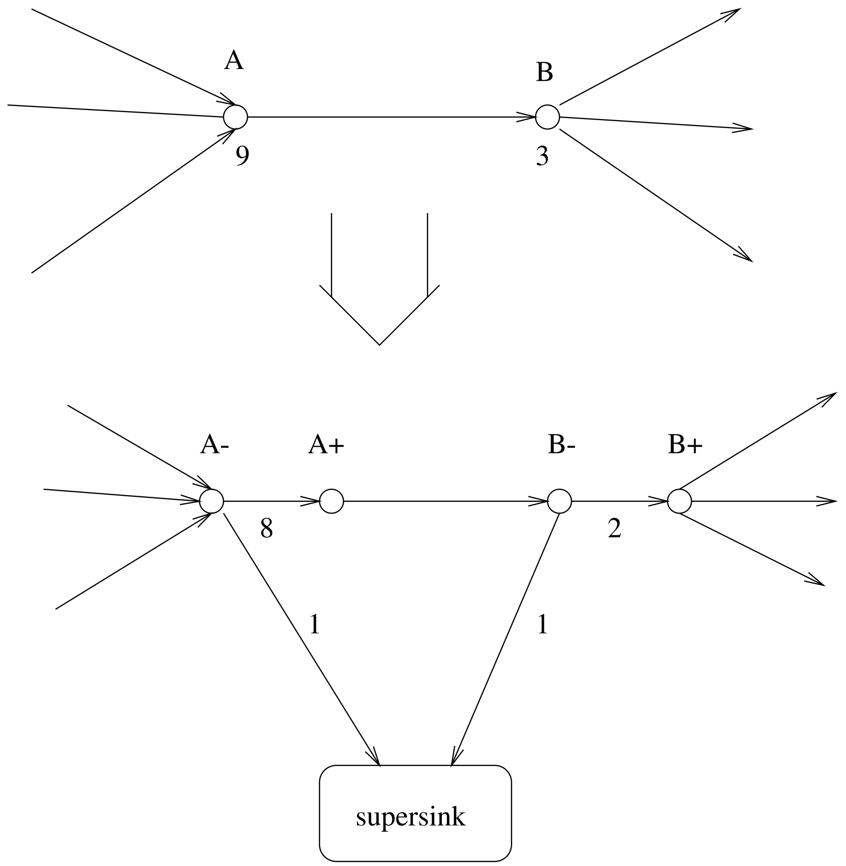
To achieve attack-resistance, a group trust metric must sacrifice the monotonicity property - if the algorithm accepts a set of nodes S for graph G, then for a graph consisting of G plus additional edges, the set of nodes accepted may not be a superset of S. Otherwise, the attack mentioned in Section **??** would be feasible.

However, for the trust metric presented below, a weaker monotonicity property does hold. As edges are added to the graph, the *number* of nodes accepted increases monotonically. As we will show in Section **??**, this weaker monotonicity property fits some applications well, particularly those based on finding a majority or other consensus from the nodes accepted by the trust metric.

3.1 The Advogato network-flow trust metric

Capacity constrained flow network.

Capacities of nodes are set as a function of distance from seed.



3.2 Comparison with Flake's Self-Organization work

The Advogato trust metric is quite similar to the Exact-Flow-Commnuity algorithm designed to discover communities existing on the Web[15]. Similarities include infinite flow to the seeds, finite, bounded capacities for the intermediate graph, unit capacity edges from all nodes to a supersink, and reaping of flow through these edges after a max-flow computation to determine membership in a community. The differences are worth noting:

- Exact-Flow-Community makes all edges bidirectional.
- Advogato places capacity constraints on nodes, while Exact-Flow-Community places capacity constraints on edges.
- Advogato uses distance from the seed to assign capacities, while Exact-Flow-Community uses a single constant capacity k for all intermediate graph edges.

The goals of Exact-Flow-Community and Advogato are somewhat different. The former is intended primarily to identify existing communities, based on patterns in Web links. Advogato also defines a community, but with the specific goal of attack resistance, in other words ensuring that an attacker cannot add an arbitrary number of false nodes to the true community.

For this goal, preserving the directionality of edges is crucial. Edges from bad nodes to good may be under the attacker's control, but edges from good nodes to bad are assumed not to be. Failure to distinguish between the two clearly leads to a loss of attack resistance.

Similarly, as described in Section 2.3, constraining node capacities yields better attack resistance than constraining edge capacities.

The relative significance of the two capacity assignment strategies is harder to determine. Advogato uses capacities dependent on the distance from the seed to accept a large number of nodes, even when the number of seed nodes (and their total outdegree) is small. The Exact-Flow-Community algorithm does not have this property. Thus, the authors propose an Approximate-Flow-Community algorithm that heuristically adds more seeds, then computes Exact-Flow-Community over this augmented seed set. It is quite plausible the results are similar to Advogato's, but we have not tested this hypothesis.

Chapter 4

Advogato

We have implemented the group trust metric described in Chapter 3, and deployed it as the key part of the Advogato website. In this chapter, we describe the implementation, analyze the data collected, and present some observations on lessons learned.

The Advogato website launched in November 1999, and has been in continuous operation since. It serves as a community message board for free software developers. The heart of Advogato is the certification mechanism, essentially a straightforward implementation of the group trust metric.

4.1 Implementation overview

Advogato is implemented as a C-language Apache module, available under the GNU General Public License.

Internally, mod_virgule is heavily based on XML. All of the account information (including the peer certifications), diary entries, articles, responses, and project information is stored in the filesystem in XML format. To handle a request, mod_virgule dynamically generates HTML from these XML files. The content varies slightly if the user is logged in, as determined by the standard HTTP cookie mechanism.

Advogato extends the yes/no evaluation of the trust metric presented in Chapter 3 to three levels, entitled Apprentice, Journeyer, Master. Each certification edge has an associated level. The underlying trust metric is run once for each level. For a given level, edges below that level are discarded. For example, to compute the Journeyer level, the underlying trust metric is run on the subset of the trust graph consisting of Journeyer and Master edges. The final certification assigned to the node is the maximum level for which the node is accepted.

Trust "seed" is hard-wired. In practice, hard-wired seed hasn't caused problems, but it is a concern. Providing user-configured alternate seeds is a possible extension, but the need for it hasn't been strongly felt.

4.2 Implementation details

One motivation for the implementation choices was to experiment with the platform provided by the Apache module interface. C is not a popular language for implementing Web sites. Yet, our experience has been quite good.

A core feature of the Apache platform is the use of resource *pools*. In a pool discipline, memory and other resources are allocated from an explicitly identified pool. At some later time, when all memory allocated in the pool is known to be unreachable, the entire pool is freed.

Pools have been used for a long time. Ross[32] presented a storage package with explicitly identified *zones*, roughly equivalent to Apache's pools. There has been a fair amount of recent interest in providing explicit language support for pools, including extensions to the C language[16], theoretical language designs[9], and the ML Kit compiler for Standard ML[35].

In the case of Apache, there is no explicit language support. Rather, allocation functions take an additional explicit pool argument, and are called using the standard C runtime discipline. For example, the low level memory allocation function is:

```
void *ap_palloc (struct pool *p, int nbytes)
```

In the context of mod_virgule, most allocation is done in a *request pool*, which has a lifetime exactly equal to one Web request. When the request is complete, the Apache process frees the request pool, and typically becomes available to serve the next request.

The pool discipline is appealing in many ways. When properly used, memory management is nearly painless. Further, memory leaks are rare. These advantages are similar to those provided by garbage collecting runtimes, but with several advantages:

- Finalization is deterministic.
- There is no need for language or runtime support.
- Time and space overhead is minimal.
- The discipline interoperates easily with external libraries.

The last of these points is demonstrated by mod_virgule's use of libxml[**?**].

4.3 Experimental results

Provide data about graph: number of active nodes in trust graph (¿ 1000), number of Apprentice, Journeyer, and Master nodes. Also present distribution of capacities of nodes. Average capacity of accepted nodes is related to vulnerability of trust metric in the face of random attack.

Maybe visualize Master sub-graph, although it's already too tangled to see much.

One conclusion: manually creating certifications to build a trust graph is realistic, if there is some expected payoff.

Chapter 5

An attack-resistant name service

This chapter presents the design of an attack-resistant name service, using the group trust metric described in Chapter 3 to provide high assurance even under attack.

For the purpose of this chapter, name service is the problem of finding the appropriate public key corresponding to a given name. This problem is quite thorny in practice. Determining who owns the rights to a given name is not well-defined. Rather, it is the subject of social, economic, legal, and political issues. Real world CA's have misbehaved. Cite VeriSign/Microsoft.

We address these issues by separating mechanism from policy. This requires policy to be formally specified. Thus, the policy language must be rich and flexible enough to accommodate the policies desired.

Probably the only workable policy is "first-come, first-served," as it's the only one that's understandable.

Choice of namespaces: flat, hierarchical, hash-based. Also cite SDSI for relative namespaces.

5.1 Related work

There are many designs and implementations of systems to bind keys to names. The most popular is the X.509 family of standards, loosely synonymouse with the Public Key Infrastructure (PKI). In this framework, authority for the namespace is assigned to one or more *certification authorities* (CA's), generally composed of *root CA's* that delegate authority over some or all of the namespace to various subsidiary CA's. A consequence of this design decision is the *root vulnerability*—the compromise of any of the root CA's leads to catastrophic security failure.

Root vulnerability is a serious concern. Even though individual CA's may be well managed (with private keys reasonably well protected), modern applications depending on PKI such as Web browsers ship with dozens of root CA keys enabled. Further, the grouwing popularity of CA software for off-the-shelf platforms (such as Entrust/PKI and Entegrity's Notary) implies that CA's will be deployed in contexts where it is difficult or impossible to protect the CA with a high degree of assurance.

Often, the CA given root authority is implementing a fairly simple and well defined policy. For example, VeriSign's Class 1 certificates are issued on a first-come, first-served basis. Domain name registrars issue second level domains on an effectively first-come, first-served basis, with the possibility of revoking the name (and issuing it to someone else) for non-payment of the bill, or as the result of a dispute. Actual implementation of these policies is at the whim of whoever holds the CA's private key. An attacker who succeeds in compromising the CA key can reassign names completely at will, with no regard to any policy.

It is possible to implement well defined policies and avoid root compromise. The naming service presented in this chapter factors the name service problem into to sub-problems: a *policy language* for expressing policies formally, and a *distributed trusted third party* for implementing the policies. In cases where the complex blend of factors controlling ownership of names can be distilled into a formal policy, our naming service can provide much higher assurance with lower certification cost than existing PKI's.

A design factored in this way can only be successful if two goals are met: the policy language should be rich enough to express a range of policies useful in the real world, and the distributed trusted third party should be trustworthy. Section 5.2 presents such a policy language, and Section **??** presents a distributed network based on an attack-resistant trust metric.

The policy language introduces the concept of *asymmetry* between initial registration of a name and updates to that name. In traditional PKI designs, the authority granted to the CA is total, so the CA has equal power to register and update name/key bindings. However, there are many interesting asymmetrical policies where the power to update is more restricted than the power to register. The first-come, first-served policy is an extreme example. Authority to register new names is trivially granted, but authority to update existing names is never granted.

5.2 Policy language

The goals of the policy language are:

- Flexibility: different policies can be specified for different parts of the namespace.
- Low certification cost if security against unauthorized registrations is not important.
- High security if higher certification cost is tolerated.
- Good security against unauthorized updates.
- Low vulnerability to root compromise.

Meeting all these goals requires a moderately complex language. In practice, a simpler policy, such as first-come first-served, may be a better choice. First-come first-served is poor at assigning names consistent with some other concept of ownership (such as trademarks), but its limitations are easily understood and analyzed.

Before defining the policy language itself, we define the space of *names* and *requests* over which the policies are defined. Names are similar to familiar Internet hostnames and e-mail addresses, such as "raph@cs.berkeley.edu". As in DNS, names are hierarchically structured. Thus, the parent of "raph@cs.berkeley.edu" is "cs.berkeley.edu", its parent in turn is "berkeley.edu", and so on. Names with children are *domains.*

Each request is either a *registration* or an *update* request, and in either case contains a name, a value (such as a public key) to be bound to that name, and policies to be associated with that name. In addition to standard names, there are special namespaces for policy fragments and groups. Each request is also accompanied by a signature from zero or more public keys.

This design does not address read access to the database. Always allowing lookups is reasonable and consistent with existing Internet practice. Read access control is a plausible extension, however.

A request is processed as follows:

- The signatures on the request are verified, resulting in a list of public keys that signed the request.
- The body of the request is used to retrieve an appropriate policy from the database.
- The policy is evaluated over the list of public keys signing the request, resulting in a boolean value.
- If the boolean value is *true,* then the database is updated as per the request.

This workflow is similar to that of PolicyMaker*BFL96* and other trust management systems.

For registration requests, the appropriate policy is the *registration policy* of the parent. For update requests, the appropriate policy is the *update policy* of the name. The update policy for a name is fixed upon registration of the name. It cannot itself be directly updated. The registration policy for a domain can be updated.

The policy language is defined as follows:

$$\begin{array}{rcl} \mathit{policy} & ::= & \mathit{int}\,'\mathit{of}'\ \mathit{group} \\ & | & \mathit{policy}\ '\wedge'\ \mathit{policy} \\ & | & \mathit{policy}\ '\vee'\ \mathit{policy} \\ & | & 0 \\ & | & 1 \\ & | & '*'\ \mathit{policy\text{-}name} \end{array}$$

$$\begin{array}{rcl} \textit{group} & ::= & '\{' \ \textit{key} \ '\}' \\ & | & \textit{group} \ '\cup' \ \textit{group} \\ & | & \textit{group} \ '\cap' \ \textit{group} \\ & | & '*' \ \textit{group-name} \end{array}$$

$$\begin{array}{rcl} \textit{key} & ::= & \textit{key-hash} \\ & | & '*' \ \textit{key-name} \end{array}$$

$$\textit{policy-name} \ ::= \ \textit{id} \ '?' \ \textit{name}$$

$$\textit{group-name} \ ::= \ \textit{id} \ '@@' \ \textit{name}$$

$$\textit{key-name} ::= \textit{name}$$

As stated above, the goal of this policy constraint language is to enable a DNS-like naming hierarchy with a flexible, varying degree of control. A particular goal is to preserve the security of delegated sections of namespace even when the nodes in control of the root, or high-level domains, are compromised. At the same time, if a delegated name turns out to be improperly granted, it should be possible to recall it.

The policy constraint language provides a mechanism to balance these two goals. It easy expresses both the first come, first served policy, by trivially granting registration requests, but never granting update requests. Similarly, the policy of central control is easily represented by symmetrical registration and update policies.

Evaluation of whether a given policy meets a policy constraint expressed in the policy constraint language is a co-NP complete problem. Here we present a brief proof outline in the form of a reduction to 3-SAT, which is a well known NP-complete problem.

Present pseudocode algorithm for implementing policy constraint language.

5.3 Implementation of naming service

In order to avoid a single point of failure, we choose a peer-to-peer network architecture for the implementation of the naming service. For simplicitly, we choose a two-level architecture, with a relatively small number of "server" nodes, and most clients contacting the server nodes to retrieve names and certificates.

All servers have knowledge of all other servers. Thus, notification of server joins and leaves are broadcast operations. When the number of servers is on the order of the square root of the total number of peers, network traffic is optimized. This square-root behavior is neither as bad as flat broadcast networks, such as the original Gnutella protocol, nor as good as the logarithmic performance of systems such as Chord[33].

Not all nodes store records for all names registered in the system. Rather, there is a "responsible server" relation, generally based on a hash function so that the set of responsible servers for a given name is effectively random. An average of 10 to 20 servers should be responsible for each name. This average is a tunable parameter. As it increases, security improves, but overall network traffic also increases.

There are essentially two different authorization decisions in the system. The first, evaluated by nodes when processing registration and update requests, is whether the request is valid. The second, evaluated by clients when performing queries, is whether the node responding to the query is trustworthy to provide the correct answer. Our design mirrors the fundamental asymmetry of these two decisions. Essentially, for *mutation* requests, clients send the request to all responsible nodes for the name, regardless of whether such nodes are to be considered trustworthy. However, for *query* requests, clients only consider responses from nodes which are both responsible for the name and considered trustworthy.

Our design mixes up this clear separation a bit, because nodes processing mutation requests also function in the client role when they retrieve the policy and policy constraint associated with the parent name.

5.3.1 Responsible servers

This subsection presents a simple, effective relation for determining which nodes are responsible for which names.

Each server has an ID, which is the 160-bit SHA-1 hash of its public key. Similarly, the SHA-1 hash of any name gives a value in this space, ie the id of name x is $H(x)$.

We define a "distance metric" $d(n_1, n_2)$ to be the absolute value of $n_1 - n_2$ interpreted using 160-bit two's complement arithemetic. This metric induces a "ring" topology on the space of ID's.

With this definition, a server with id n_s is responsible for name x iff $d(n_s, H(x)) < \Delta d$, where Δd is a tunable parameter of the system. A reasonable target is that 10 to 20 good servers are responsible for any given name, i.e. Δd is equal to 5 to 10 times 2^{160} divided by the number of good servers.

5.3.2 Mutation requests

In this subsection, we describe in detail the algorithm used by nodes to authenticate mutation requests. Overall, this authentication process strongly resembles PolicyMaker[5].

There is no consistency guarantee for mutation requests. In particular, two registrations coming in at roughly the same time may lead to inconsistency, with different nodes giving different answers to the same query. Improving consistency is an interesting research direction. Perhaps the most profitable approach is to adapt the Byzantine fault tolerance research[7].

5.3.3 Query requests

In this subsection, we describe and analyze the algorithm used by clients to validate responses to query requests. This process is strongly dependent on a group trust metric, and we discuss how the security properties of the trust metric affect the security of the system as a whole.

The client's first step is to determine a set of responsible and trustworthy servers. In our design, we simply determine the set of all trustworthy servers first, then apply the "responsibility" relation as a filter. It is tempting to consider only a subset of all trustworthy nodes. Such a design might be considerably more efficient (as is demonstrated by the Distributed Hash Tree work such as Chord[10] and Kademlia[22]), but unfortunately the security of the group trust metric as described in Chapter 3 depends on knowledge of the entire trust graph.

Armed with this list (which is expected to contain on the order of 10 or 20 nodes), the client sends the request separately to each node.

Finally, as the responses come trickling back, the client waits until a majority of nodes queried provide a consistent response. If no single response is in the majority, then the lookup is considered to have failed, with no trustworthy binding to that name.

Note that in the common case, when all nodes provide the same answer, the client accepts the binding as soon as the earliest half of nodes respond. Thus, individual server failures are tolerated, and request latency is equal to the median latency of server nodes.

5.3.4 Security analysis

The voting semantics of the validation of responses to queries is a good match to the guarantees provided by the group trust metric.

Assuming no conflicts during the registration phase, then all good nodes responsible for a name will yield the same ("correct") answer to a query. Thus, if at least one half of the subset of responsible nodes accepted by the trust metric are good, then the correct answer will be obtained.

The group trust metric can guarantee that a fraction of the total nodes accepted is good, but this is not the same as guaranteeing that a fraction of the accepted nodes responsible for a name is good. Indeed, it is likely that an attacker would be able to mount a successful attack against a single name at low cost.

Chapter 6

Attack-Resistant Metadata

Systems should be designed so that metadata is attack-resistant, as well.

One important form of metadata: “this song is beautiful.”

Another example: micropayments in a music distribution system, commonly known as “pay Lars.” But: which Lars? Cite Scott McCloud’s discussion of micropayment systems for comics as well.

Another form of metadata that has received substantial recent attention is relevance of Web search results. To answer this query, an extremely rich trove of raw data is available: the link structure of the several billion documents currently available on the World Wide Web. Two systems have attracted particular attention: IBM’s Clever[18], and the PageRank algorithm used by Google[27]. The latter is claimed to have some attack resistant properties, and the experience from Google’s deployment is encouraging in this regard: the relevance of Google’s search results is widely praised, and it does seem to be rather difficult for spammers or others wishing to unfairly dominate the search results to do so, except at considerable expense.

One interesting recent example is the dominance of pro-Scientology listings in the Google results for a search on the keyword “scientology.” This is discussed in an online essay[29]. To Google’s credit, as of 2 Mar 2002, this search yields one anti-Scientology site, in fact at #4. *(probably cut this—the scieno thing has obviously become far more a political issue than a technical one)*

6.1 Brief analysis of PageRank

A description of the PageRank algorithm was published in 1998[27], but as the work has continued as a business rather than an academic pursuit, there has been no published followup. This section attempts to address this hole in the literature, by providing a brief discussion of the PageRank’s attack-resistance.

6.1.1 Recap of PageRank

For the purposes of analyzing the PageRank algorithm, it is most useful to present it in its random walk formulation. Each walk begins by choosing a node u, using an initial distribution $E(u)$. At each step, with probability $||E||_1$, the walk ends. Otherwise, the next node in the walk is chosen uniformly from the successors. Thus, the probability that a walk is of length k is $||E||_1(1 - ||E||_1)^k$. The rank assigned to each node is the probability that such a walk will end at that node.

Here, $||E||_1$ denotes the L_1 norm of E, informally “Manhattan distance” and formally:

$$||E||_1 = \sum_u |E(u)|$$

Note that when E is nonnegative, $||E||_1 = E \cdot \mathbf{1}$, where $\mathbf{1}$ is the vector of all ones.

The PageRank paper also presents an eigenvector formulation of the algorithm. The adjacency graph A is defined as follows: A_i, j is $1/N_i$ if there is an edge from i to j, otherwise 0, where N_i is the outdegree of node i. Let $R = c(A + E \times \mathbf{1})R$, where $||R||_1 = 1$, and c is maximized. Thus, R is an eigenvector of $(A + E \times \mathbf{1})$. The eigenvector formulation is equivalent to the random walk formulation: $R(u)$ is equal to the probability that a random walk ends in node u.

6.1.2 Attack-resistance of PageRank

What exactly do we mean to say that an algorithm such as PageRank is attack resistant? In fact, the criteria are basically identical to the Advogato group trust metric presented in Chapter 3. The input to the PageRank algorithm is a graph. In the Google application of PageRank, nodes correspond to web pages, and edges correspond to links. The "seed" of the Advogato trust metric corresponds to the initial vector E of PageRank. Finally, the output of PageRank is a real-valued ranking, while Advogato produces a boolean accept/reject value for each node. Note that the Advogato trust metric can be extended to multiple levels of trust by doing multiple runs of the algorithm with different capacities.

There are a number of different ways to quantify the success of an attacker. Here, largely for purposes of simplifying the proof outline, we choose the sum of the rank accorded to all nodes under the attacker's control. This measure is very similar to the one chosen for the analysis of the Advogato trust metric: the number of "bad" nodes chosen.

We state the attack resistance property of PageRank more formally. For any labelling of the nodes in the graph G into "good" and "bad" nodes, the sum of the rank of the "bad" nodes is bounded as follows: ...

The analysis of the attack-resistance of PageRank is dependent on the choice of the initial source of rank, $E(u)$. As in **??**, let the nodes in the "trust graph" (ie web pages) be divided into three classes: good, bad, and confused. To recap, a "confused" node is one that is itself good, but may contain links to bad nodes. Let us then assume that $E(u)$ is zero for all bad nodes u.

Now, it is possible to see that the resulting PageRank meets the "bottleneck" criterion of **??**. A given walk will lead either to a good page or a bad page. In the worst case, if the walk ever touches a bad node, then the resulting page may be bad. Otherwise, it is good.

6.1.3 Significance of initial vector and walk length

Here, we see that the choice of initial vector, as in the choice of "seed" in the Advogato trust metric, has a critical effect on the quality of the results.

If the fraction of good nodes in E is p, then the total rank accorded to good nodes is multiplied by p.

One of the choices of E presented in the PageRank paper is a uniform distribution over the set of all nodes in the graph. This value for E is not attack-resistant; an attacker can simply create a huge number of web pages, and have them link to a "bad" page which will then receive a high rank.

Thus, to achieve attack resistance, E must be chosen with some tendency toward good nodes. Obviously, if it is possible to do this with perfect accuracy, the trust metric is not needed. In general, a good strategy is to hand-pick a small subset of sites that are known to be good, and let the walks cover the rest of the graph. In the context of websites, one good way to choose start nodes is to use a collaboratively edited link directory such as the ODP (dmoz.org).

Given this assumption, the choice of walk length becomes clearer. The shorter the walks, the more attack resistant. However, if walks are too short, then good nodes are inadequately reachable from the start nodes in E. The PageRank paper gives a value of 0.15 that the walk ends at each step. Thus, walk lengths are a geometric distribution with a mean of $(1-||E||_1)/||E||_1$, thus 5.67 for $||E||_1 = 0.15$. This probably represents a good real-world balance between complete coverage and low vulnerability.

The PageRank paper presents another intriguing choice for E: the set of "root" web pages. This set can be determined automatically, reducing the dependence on manual labor. Since setting up a root web site requires purchase of a domain name from a DNS cartel, typically on the order of $10 per year, mounting an attack requires money, generally in an amount proportional to the sum of rank desired. It is unclear whether Google uses this criterion for their own E. If so, this analysis lends credence to the widespread belief that cross-linked web sites with many different domain names are a good way to increase Google rank.

6.2 What makes a trust metric attack resistant?

Now that we know of two different trust metrics that we consider attack resistant, it is worth investigating what the similarities and differences are.

Network flow and principal eigenvectors are two different computations. They are not used interchangeably. However, there is a common thread which allows both algorithms to serve as the basis of attack resistant trust metrics. In this section, we propose the *bottleneck property* as a common feature of both trust metrics. We conjecture that this property is satisfied by all attack resistant trust metrics, although at the moment there is no proof.

The bottleneck property, informally stated, is that the total trust quantity accorded to an $s \to t$ edge is not significantly affected by changes to the successors of t. Informally, when s is a "good" node and t is a "bad" node, this means that manipulation on the part of the "bad" nodes does not affect the trust value.

Scalar trust metrics, no matter how carefully tuned, all fail to satisfy the bottleneck property, and are not attack resistant. It is fairly easy to see why they do not: since the trust computation is performed independently for each final target node, simply adding more target nodes increases the total amount of trust accorded to target nodes. To be attack resistant, a trust metric must dilute the trust accorded to the successors of t as more successors are added, either by decreasing the value of each (as in the case of PageRank), or decreasing the relative probability that any one of these successors is accepted (as in Advogato).

The class of attack resistant trust metrics may have more instances than Advogato and PageRank. Network flow and eigenvectors are both compelling in their simplicity, resulting in relatively easy analysis and implementation. Even so, it may be that some other, yet to be discovered, trust metric is better suited to real applications.

6.3 Generalized metadata

The results of a trust metric or Web page ranking represent a fairly special class of metadata: essentially boolean or real "confidence value" for inclusion in a group. This section describes how an eigenvalue-based trust metric might be extended to more general forms of metadata assertions.

The primary new concept is that of an "assertion," which is simply a statement of some sort. Examples of meaningful assertions include "I think the song hashing to `01..EF` rates 9 out of 10", "Tips to be paid to the artist of song `01..EF` should be paid to Swiss bank account 12345", "The song hashing to `01..EF` is actually *Man of Constant Sorrow*, recorded by the Soggy Bottom Boys".

The ranking of such pieces of metadata is given in terms of random walks. Begin the walk at a start node (generally, this will correspond to the user of the ranking system). At each step, if that node contains a matching metadata record, the walk stops. If not, then the walk with some fixed probability p terminates, or proceeds to a successor chosen at random.

The concept of "matching" metadata record requires a bit of further elaboration. For example, it makes little sense to rate a song as both 4 out of 10 and 9 out of 10. Thus, the template for such a query is "I believe song `01..EF` ranks * out of 10". The random walk will end at any node containing such a matching record.

6.4 Distributed network model

This section presents a distributed network model of the eigenvector approach. The initial presentation is based on PageRank, but it extends straighforwardly to more general metadata.

Each node i maintains a vector $R_i(j)$, assigning a real value to each other node j, with $||R_i||_1 = 1$. Periodically (but asynchronously), each node i performs an *update* operation, as follows. For each predecessor node s in the trust graph, node i requests the current value of R_s from node s. All such vectors must satisfy $||R_s|| = 1$; if not, the vector is clearly bad and should be discarded. A new R_i is recomputed as follows:

$$R_i(j) = \frac{(1-p)}{|\text{pred}(i)|} \sum_{s \in \text{pred}(i)} R_s(j) + (p \text{ if } i = j, 0 \text{ otherwise})$$

It should be clear that such a network will converge to the same value as the random walk and eigenvector formulations as presented above. Specifically, $R_i(j)$ is the probability that a random walk beginning at i will end at j, with p representing the probability that the walk terminates at each step.

Thus, this network computes the "personalized PageRank" for each node in the network. Distributing the computation may ease the computation and communication bandwidth bottlenecks inherent in a centralized approach.

Further, the distributed model eliminates the need for global knowledge of the trust graph. Each node need only query its immediate predecessors.

To adapt this network model to generalized metadata as described in Section 6.3, R_i becomes a vector over assertions, rather than over other nodes. For all metadata statements j local to node i, $R_i(j) = 1$, and $R_i(k) = 0$

for all $k \neq j$ and for which the templates of j and k match. For example, if j is "Song X ranks 7 out of 10", then statements "Song X ranks 4 out of 10" receive a 0 value. For all metadata statements not covered by local assertions, the propagation formula above is used.

In actual implementation, scaling becomes a problem as the number of elements in the vector R grows. For a small network, or even a large network covering a relatively small number of metadata statements, scaling may not be a serious problem. Some techniques worth exploring include:

- Pruning the vector for near-zero values.
- Data compression of the vector.
- Adaptive transmission, so that slow-changing values of $R_i(j)$ are sent less frequently.

Analysis: why is such a network attack resistant? The "bottleneck property" generalizes the capacity constrained flow network of the group trust metric presented in Chapter 3.

Discussion of dynamic behavior: can optimize for "diamond in the rough", by selecting for high recommendations with low confidence value. Then listen to song and inject a new recommendation into the network. If song is excellent, then recommendations will propagate quickly through the network. If song sucks, then low recommendations will remove song from the diamond in the rough category. The goal is to get good coverage of reviews while minimizing the number of people forced to listen to mediocre songs.

Chapter 7

Message Flow Networks

The network flow-based trust metrics presented in previous chapters measure the flow of an abstract quantity through a trust graph, and use the presence of sufficient flow to a target node to decide whether that node should be accepted. In this chapter, we consider flow networks in which messages themselves are the unit of flow.

Such networks are especially useful for spam-resistant messaging. Indeed, using a trust metric for access control is vulnerable to attacks involving huge volumes of spam messages. The goal of a message flow network is to tightly bound the *volume* of spam messages which may be succesfully delivered by bad nodes.

7.1 The basic message flow design

7.1.1 Assumptions

We assume a trust network with similar properties as in previous chapters. Each node in the network represents a participant in the overall system. Each user manually enters *trust edges* into the graph. Each edge is annotated with a message volume associated with the peer.

If a user s wants to receive no more than k messages per time unit from peer t, then s enters a trust edge pointing from t to s, annotated with k units of flow. Note that the direction of this trust edge is reversed with respect to the trust metric presented in Chapter 3.

We also assume a central server which knows the entire trust graph, and also keeps track of dynamic information based on the actual flow of messages sent through the system.

7.1.2 Message flow

In particular, the server maintains a *residual capacity* for each trust edge in the network, named in analogy to the standard algorithm for computing maximum network flows. However, instead of attempting to saturate the graph with the maximal number of augmenting paths, residual capacity is consumed only when messages are actually sent.

In more detail, when a node x wishes to send a message to node s, the server attempts to find a path from x to s through the residual capacity graph. If no such path exists (in other words, if there exists a partition of the graph with x on one side and s on the other in which all trust edges have zero residual capacity), then the message is blocked. Otherwise, the message is sent, and the residual capacity for each edge along the path is decremented by one.

As usual in computing augmenting paths, it's a good idea to use the heuristic of choosing a shortest path through the residual graph, thereby minimizing total capacity consumed for an individual message.

7.1.3 Repleneshing the flow

Edge capacity is measured in units of number of messages per unit of time. In addition to consuming capacity when messages are sent, there must also be a mechanism for replenishing this capacity, so that the network is roughly in equilibrium when there is a steady stream of messages.

There are a number of approaches to this repleneshing. Perhaps the simplest is to replenish *all* residual capacities to match the maximum capacity specified in the trust graph, once every time unit. However, this approach has a number of disadvantages, including an uneven probability of rejecting messages over time, depending on the fractional time since the last increment of the time unit.

It would also be desirable to accommodate bursty traffic patterns. If a user specifies seven messages per week from another peer, it would be unusual to expect messages sent at intervals of exactly 24 hours. Rather, it's more likely that several days would pass without a message, interspersed with bursts of several messages in quick succession.

Thus, we propose a computationally simple and efficient method modelling a process by which flow is replenished with exponential decay. In addition to storing a scalar for each edge representing the residual capacity, the server also stores a timestamp representing the last moment that the capacity was updated.

Then, each time the capacity is queried or modified, it is updated according to the following simple algorithm:

$$\begin{aligned}&cap = cap + (max - cap) \cdot (1 - \exp(timestamp - now))\\&timestamp = now\end{aligned}$$

It should be clear that, over a period of one time unit, the total flow across such an edge will never exceed the maximum specified. At the same time, burstiness is penalized, but only somewhat. For example, if messages are sent in bursts once every time unit, interspersed by inactivity, then the total flow is $1 - e^{-1} \approx .632$ of the maximum specified.

7.2 Security Analysis

Outline: partition network into good & bad.

Total message flow across partition is bounded by total capacity of edges across the partition.

Consider two cases of partition:

1. There is one bad node. Then he can't send more spam than the total cap of edges from good nodes linking in.

2. There is one good node. Then you can't receive more spam than the total cap of edges to you. Have a heuristic that favors short paths when multiple messages are competing for scarce flow. Then, hopefully all the good mail from nearby neighbors gets through, and at worst the spammer blocks only mail from good peers farther away.

7.3 Pragmatics

The design of this chapter is unimplemented as of this writing. However, it should be straightforward to implement as a centralized web service.

Obviously, having a centralized server is limits both scaling and security (all users are vulnerable to an attack on the central server). A completely decentralized peer-to-peer network implementing the core concepts of the message flow network presented above is the topic of the next chapter.

7.4 Application Notes

Outline: email is killer app for this.

A similar app is posting comments and backlinks in blogs. Since # nodes is so much smaller, a centralized server may be appropriate.

Don't need to actually send messages through net. Don't need to replace SMTP. You can just send tokens *authorizing* the sending of a message. Then, the sending email client can query the server for such a token, and insert it into the headers of the mail. The recepient email client extracts the token from the headers, and verifies it.

Chapter 8

Distributed Stamp Trading Networks

This chapter describes how capacity constrained flow networks may be used to build a scalable and robust peer to peer network infrastructure. Peer to peer networks have to deal with the following questions, usually much more straightforward in centralized networks:

- How can a node discover other nodes to communicate with?
- If there is a subset of nodes offering a particular service, how can a node discover nodes in that subset?
- How can scarce resources (storage and bandwidth) be allocated in an open network, where services are provided to all peers?

These questions are so difficult that most implementations of peer-to-peer networks decentralize only part of the total network architecture (for example, movement of bulk data), while retaining centralized servers for this "network metadata." Only very innovative designs such as Freenet[**?**] and Gnutella[**?**] embrace a completely decentralized approach, and such designs tend to suffer from both scalability problems and vulnerability to malicious attacks. For example, there is discussion of "cancerous nodes" in the Freenet community.

This chapter describes a partial design for a peer to peer network in which the discovery of other nodes, as well as the provisioning of services, is governed by a flow of "stamps," or simple authentication tokens. The initial flow of stamps is determined by manually configured trust relationships, similar to those of Advogato and the other trust metrics described in this thesis. The flow of stamps is effectively a capacity-constrained network, thus providing the network as a whole with some attack resistance.

The design presented in this chapter is somewhat speculative. It has not yet been implemented, even as a prototype, and there is no doubt a considerable amount of refinement needed to become truly practical. Even so, it represents a dramatic improvement in robustness over any peer to peer network design currently known, and at a level of efficiency rivalling that of the most advanced designs.

8.1 Stamps

We assume a peer-to-peer network of *nodes* on the Internet. Each node is implemented as a daemon process with access to storage for the node's state. Further, each node has an associated public/private key pair. The hash of the node's public key (typically a 160 bit string, assuming SHA-1 as the hash function) serves as its id. Each node listens for connections on an IP address and port, and can request connections to other IP address/port tuples. Such a model is entirely standard, and is shared by numerous other peer-to-peer networks, including MNet[36], Freenet[**?**], and Freehaven[11].

We use the XOR distance metric between node id's, as proposed in Kademlia[22]. The distance $d(x_0, x_1)$ between two nodes with id's x_0 and x_1 is simply $x_0 \oplus x_1$, the bitwise XOR of the id values.

As with all the trust metrics discussed in this dissertation, we also assume trust links between the nodes. The owner of each node will enter a list of node id's of other trusted nodes. These correspond to successors in a trust graph. With the addition of these trust links, the model is quite similar to the distributed network model for generalized metadata as presented in Section 6.4.

Nodes on a stamp trading network also, naturally, create and circulate *stamps*. A stamp is essentially a five-tuple:

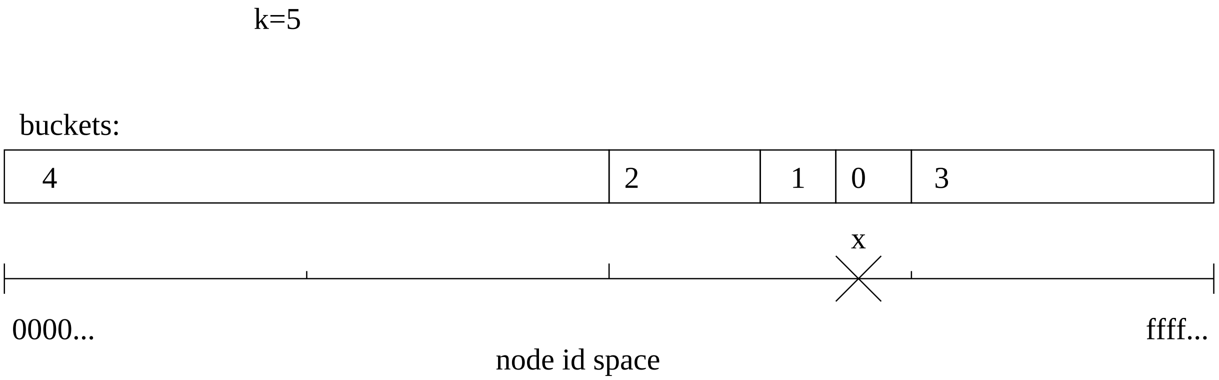


Figure 8.1: Example of XOR-metric buckets.

- The *issuer,* which is the node id of the node that created the stamp.
- A random nonce, used by the issuer to check the validity of the stamp.
- The current IP address and port number for the issuer.
- The expiration date.
- The stamp's *value,* which is a real number.

Each node creates a steady stream of stamps, and distributes them to its successors in the trust graph. After circulating in the network, stamps are *redeemed* by a node with a connection to the issuer, in return for services. The issuer verifies that the nonce belongs to a stamp it issued, the expiration date has not passed, and that the stamp hasn't already been redeemed. If so, it goes on to provide the service requested.

A number of peer-to-peer network designs provide services in exchange for some currency-like token. Mojo Nation[36] is perhaps the best known of these, using a "token server" as the source of currency tokens (known as "mojo") and validation service for all token transactions. Mojo Nation was inspired by the agoric systems literature, including the classic papers by Miller and Drexler[24, 23].

8.1.1 Stamp trading

Each node contains a set of "stamps on hand." These consist of stamps received directly from predecessors in the trust graph, and stamps acquired through trading operations.

Nodes provide the service of offering stamps from its stamps on hand collection. A critical design property is the policy used for setting the *exchange rate* for trading stamps, in other words the ratio of total value of stamps given to the total value of stamps redeemed. One such policy is given below in Section 8.3.

Each node has a background process tasked with maintaining a particular distribution of stamps. The space of node id's is divided into *buckets,* each representing an interval in the space of node id's. These intervals form an exponential series, with the smaller intervals clustered around the node's id. The background process aims to keep the total value of stamps in each bucket roughly the same. In particular, it will attempt to acquire stamps for underfull buckets by redeeming stamps in overfull buckets.

We can define the buckets more precisely. For $0 < i < k$, bucket i for node id x covers all distances from x in the range $[2^{160+i-k}..2^{161+i-k})$. Bucket 0 covers all distances in the range $[0..2^{161-k})$. Typically, k will be on the order of $\lg N$, where N is the number of nodes in the system. An example with 5 buckets ($k = 5$) is shown in Figure 8.1.

The distribution of stamps on hand induces a "stamps on hand" graph. If node s has stamps issued by t on hand, then there is an edge from s to t in this graph. If a node wishes to obtain services from some arbitrary other node, it must find a route to that node in this graph. For each node in this route it redeems stamps issued by that node in exchange for stamps issued by the next node in the route. Note that, unlike the trust graph, which is manually configured and generally fairly stable, the stamps on hand graph may change quite dynamically.

The distribution of buckets is very similar to that of Chord's "finger tables"[10]. Assuming that this background process is successful in maintaining the even distribution of stamp value in the buckets, then the result of Chord that nodes can find short ($O(\log N)$) routes to arbitrary other nodes applies to stamp-trading as well.

TODO: summarize Kademlia argument. Briefly, each hop reduces the distance to the target by 1/2 in the worst case.

8.2 Attacks on the stamp trading network

There are at least two distinct attacks on the stamp trading network worth considering. One is the attempt by bad nodes to consume resources. This is a noted problem in peer-to-peer networks, where the resources in question are typically bandwidth and storage[1]. However, other resources, such as human attention paid to emails, faxes, voicemail messages, instant messages, etc., may be even more worthy of zealous guarding. Spam is one important subclass of this class of attacks[14].

Given minimal assumptions about the setting of exchange rates, movement of stamps through the network is a capacity constrained flow network. This bounds the amount of services an attacker can consume by the number of edges to bad nodes. TODO: state these minimal assumptions on exchange rates.

A dual attack is to block good nodes from acquiring valid stamps. In the system presented above, there is nothing preventing an attacker from flooding the network with *bad stamps.* These stamps may come in many different flavors:

- Entirely valid stamps, but for bad nodes.
- Valid stamps issued by good nodes, but already spent.
- Completely fabricated stamps.

Some of these attacks may be preventable with known techniques, such as using Chaum's digital cash protocols[8] rather than opaque random nonces for the stamp data. However, such techniques are ineffective in the face of valid stamps for bad nodes. Thus, we seek more general techniques that cover all such attacks.

There are two such techniques worth considering. First, we can simply prune all non-reciprocal trust links. Nodes can simply refuse to accept immediate stamps from their trust predecessors unless they are also trust successors. This pruning restricts the inflow of bad stamps to edges from good to bad nodes.

Second, the policy for setting exchange rates will ideally drive the exchange rates for bad stamps to zero, at which point they cannot do any damage.

8.3 Exchange rate policy

In outline:

Note that some bad stamps (double-spent and fabricated) have detectable consequences: the attempt to redeem the stamp will fail. However, it's not always possible to detect which peer was responsible. Intuition: you can keep track of at least some of the possible wrongdoers. When you detect a failure, ding all nodes that may have been responsible. This idea is similar to that of Dingledine's reputation system for MIX cascades[12].

For each stamp on hand, maintain an audit trail of trading partners responsible. For immediate stamps received from trust predecessors, tag each stamp with a singleton list containing the stamp's issuer (ie the predecessor). When a stamp is redeemed in exchange for new stamps, assign the new stamps the audit trail for the old stamp, and append the issuer of the new stamps. Invariant: the last element in the audit trail is the stamps issuer.

Each node maintains a "credibility" factor for (some subset) of other nodes. The final redeem of a chain of stamp trades can either succeed or fail (with a number of subflavors). Apply ++ or – (respectively) to the credibilities of all nodes listed on the stamp's audit trail.

Nodes nearby in the trust graph and/or nearby in Chord distance will participate in many transactions, thus the credibility ratings are likely to be accurate. The credibility rankings will become less useful for nodes farther away.

Each stamp also has a *confidence factor.* For stamps from immediate trust predecessors, confidence factor has a (high) initial value. As part of the stamp trading service, report confidence factors. For every stamp trade, new stamp has confidence factor = old stamp's confidence factor times old stamp's issuer's credibility factor times new stamp's reported confidence factor times a damping factor. All these factors better be damn near 1, as exponent is $O((\log n)^2)$.

Exchange rate is set based on confidence, but will reflect other factors (such as difficulty of getting the stamp, and/or difficulty of keeping bucket full). Also, want to make sure that exchange rate on immediate stamps (ie those obtained from trust predecessors) is inversely proportional to volume of stamps received.

8.4 Applications

Potential application for stamp trading: spam-resistant email. One variation: whether to ring the target's cell phone or go to voicemail.

8.5 Open questions

The design presented in this chapter is quite speculative. There are two large open questions that should probably be answered before the design is implemented.

First, can the system efficiently and reliably find short routes through the network? An analysis similar to Kleinberg's algorithmic analysis of the small-world property[17] would appear to be fruitful, but Kleinberg's analysis can not be applied directly to the stamp trading network. Under the reasonable assumption that the trust graph obeys the small-world principle, it is clear that such short ($O(\log n)$) routes exist, but it is less clear that the system can find them using only locally available information.

Second, is the policy described in Section 8.3 sufficient to prevent against false-negative attacks?

Moreton and Twigg[25] further discuss and analyze stamp trading networks. [TODO: detail]

These questions remain as directions for future research.

Bibliography

[1] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, September 2000.

[2] Albert-Laázló, Réka Albert, and Hawoong Jeong. Scale-free characteristics of random netwoks: The topology of the world wide web. *submitted to Elsevier Preprint*, August 1999.

[3] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. *Lecture Notes in Computer Science*, 875:3–??, 1994.

[4] Matt Blaze. Oblivious key escrow. In *Proc. Workshop on Information Hiding*, number 1174 in LNCS, pages 334–343. Springer-Verlag, 1996.

[5] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proc. 17th Symposium on Security and Privacy*, pages 164–173, Los Alamitos, 1996. IEEE Computer Society Press.

[6] Marc Branchaud. A survey of public key infrastructures. Master's thesis, McGill University, Dept. of Computer Science, March 1997.

[7] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proc. Third Symp. on Operating Systems Design and Implementation*, New Orleans, February 1999.

[8] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash (extended abstract), 1989.

[9] Karl Crary, David Walker, and Greg Morrisett. Typed memory management in a calculus of capabilities. In *Conference Record of POPL 99: The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, Texas*, pages 262–275, New York, NY, 1999.

[10] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with Chord, a distributed lookup service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society.

[11] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, pages 67–95, 2000.

[12] Roger Dingledine and Paul Syverson. Reliable MIX cascade networks through reputation. In *Proc. Financial Cryptography*, March 2002.

[13] John R. Douceur. The sybil attack. In *Proc. 1st International Workshop on Peer-to-Peer Systems*, March 2002.

[14] Cynthia Dwork and Moni Naor. Pricing via processing or combating junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO '92*, number 740 in LNCS, pages 139–147. Springer-Verlag, 1992.

[15] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 35(3):66–71, 2002.

[16] David Gay and Alexander Aiken. Language support for regions. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 70–80, 2001.

[17] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proc. 32nd ACM Symp. on Theory of Computing*, 2000.

[18] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[19] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[20] Raph Levien and Alexander Aiken. Attack resistant trust metrics for public key certification. In *7th USENIX Security Symposium*, San Antonio, Texas, January 1998.

[21] Ueli Maurer. Modelling a public-key infrastructure. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Computer Security – ESORICS '96*, number 1146 in LNCS. Springer Verlag, 1996.

[22] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. 1st International Workshop on Peer-to-Peer Systems*, March 2002.

[23] Mark S. Miller and K. Eric Drexler. Incentive engineering: for computational resource management. In Bernardo Huberman, editor, *The Ecology of Computation*, pages 231–266. Elsevier Science Publishers/North-Holland, 1988.

[24] Mark S. Miller and K. Eric Drexler. Markets and computation: Agoric open systems. In Bernardo Huberman, editor, *The Ecology of Computation*, pages 133–176. Elsevier Science Publishers/North-Holland, 1988.

[25] Tim Moreton and Andrew Twigg. Trading in trust, tokens and stamps. In *Proc. Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[26] Andy Oram, editor. *Peer to Peer*. O'Reilly & Associates, 2001.

[27] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[28] David Post, 1999.

[29] ptsc (pseudonym). The Church of Scientology's supremacy over the search term "Scientology" on Google, February 2002.

[30] Michael Reiter and Stuart Stubblebine. Path independence for authentication in large-scale systems. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997.

[31] Michael Reiter and Stuart Stubblebine. Toward acceptable metrics of authentication. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 1997.

[32] D. T. Ross. The AED free storage package. *Communications of the ACM*, 10(8):481–492, August 1967.

[33] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[34] Anas Tarah and Christian Huitema. Associating metrics to certification paths. In *European Symposium on Research in Computer Security (ESORICS)*, pages 175–192, 1992.

[35] Mads Tofte and Jean-Pierre Talpin. Region-based memory management. *Information and Computation*, 1997.

[36] Bryce Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In *Proc. 1st International Workshop on Peer-to-Peer Systems*, March 2002.