Scalable Methods for Spam Protection in Decentralized Peer-to-Peer Networks

Jonathan Wernberg
ada09jwe@student.lu.se

Department of Electrical and Information Technology
Lund University

Advisor: Paul Stankovski

May 19, 2016

Printed in Sweden
E-huset, Lund, 2016

Abstract

Decentralized peer-to-peer networks offer several benefits over the today more commonly employed centralized client-server networks. Peer-to-peer networks allow for seamless distribution of web content over all participating nodes. This makes the hosting and access of web content much more fault tolerant, secure, faster and cheaper. However, today’s peer-to-peer networks suffer from the inability to effectively protect against spam. This has led to the web becoming increasingly centralized instead.

Methods to protect against spam on small peer-to-peer networks already exist. However, they all fail to scale up to large networks with a large number of participating peers. In this master thesis work, a new spam protection method based on design structures known to scale efficiently has been created. In order to evaluate how well it protects against spam, it is through simulations compared to what currently is the state of the art in that regard.

The proposed method is found to provide noticable spam protection and scale up with ease. Although the method still lags behind the less scalable methods in terms of effectiveness in keeping spam out, in certain cases it is the only realistic option to implement. This master thesis work takes one additional step towards enabling the web to become more decentralized - meeting the requirements of the future.

Foreword

This thesis was carried out as part of fulfilling a Degree of Master of Science in Engineering, Computer Science and Engineering. It was carried out at the faculty of engineering, LTH, at Lund University. I want to thank my supervisor Paul Stankovski, who has assisted me with valuable input throughout the thesis process.

I also want to thank my parents for giving me all the moral and financial support I needed to complete this education.

Table of Contents

List of Figures

List of Tables

Terminology

Here some terminology used throughout the thesis is described, especially focusing on the words there the precise meaning or distinction need to be clear to fully understand the thesis.

User The real user of a network, typically a real person.

Network Any cluster of computers connected to each other on the application level. Figure 1.1 shows an example of a client-server network and a peer-to-peer network. A computer can participate in multiple networks at the same time.

Server A computer that forms a permanent key part of a network, without which some or all functionality of the network would stop working.

Client A computer temporarily participating in a network, typically to access or post resources on the network. A client joins the network by connecting to a server or another client, and can leave the network at any time.

Peer A client on a peer-to-peer network.

Account A virtual user on a client-server network, typically identified by a nickname or e-mail address, and authenticated using a password. A real user can have zero or more accounts on the same server. Accounts do not have to be tied to a user’s real identity, they can be pseudonymous.

Identity A virtual user on a peer-to-peer network, typically identified by a public key and an associated nickname, and authenticated using the private key belonging to the public one. A real user or client can have zero of more identities on the same peer-to-peer network. Identities do not have to be tied to a user’s real identity, they can be pseudonymous.

Web Content Any content served on a network, such as webpages, messages and files.

Spam Any unsolicited web content, such as mislabeled files, misleading information, advertisement, virus and malware, automatically generated messages, messages without actual content, defamatory messages, and so on.

Spammer A user posting spam.

Honest User A user who is not a spammer, and never posts spam.

Chapter 1 Introduction

Peer-to-peer networking has seen some widespread use throughout the years, from messaging platforms to file sharing. But peer-to-peer networking was always meant to be so much more.

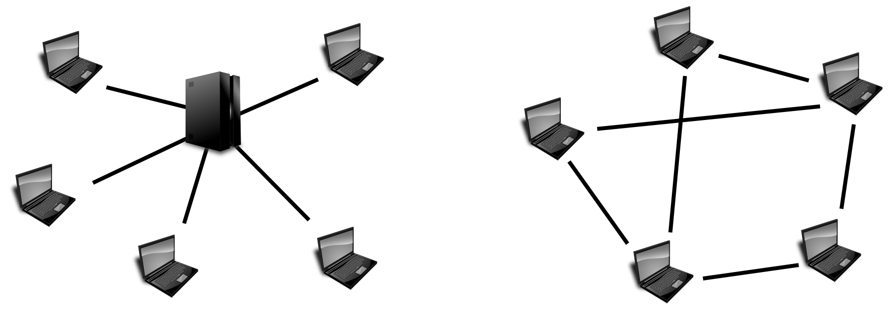

Figure 1.1: Illustrative figure of a client-server network to the left, and a decentralized peer-to-peer network to the right. An edge represents a connection.

A peer-to-peer network is any network where the clients are communicating with each other, instead of all communication going through a central server. This has many benefits. First off, the server does not need to store or transfer large files to a large number of clients. By allowing clients to store and transfer the files among each other, server costs are heavily reduced. Not only that, but transfer speed and response times are also improved since a large file can be downloaded or streamed from many clients at once, including clients that are much closer to you than a central server would be [11].

But it does not stop here. Since the transfers take place between clients, the transfer speed and response times are not affected much if some clients disappear or become unresponsive. In fact, even the server may become inaccessible without any active transfers being interrupted or even affected. This makes peer-to-peer networks very fault tolerant and robust against attacks.

One further improvement on peer-to-peer networks are so-called fully decentralized peer-to-peer networks, which is what is being considered in this thesis. On those networks there is simply no server or central point at all, not even an administrator or moderator. Instead all communication take place between clients, and the users own and maintain the network. Since there is no longer any server, hosting is the cheapest possible, and since there is no single point of failure, it is the most secure against targeted attacks.

Unfortunately, the web today is for the most part built on traditional client-server networks instead of peer-to-peer networks. This results in very visible problems. Both large and small websites are often forced to feature advertisements or beg for donations to cover their running costs, a problem that is only worsened since they often need to have many regional servers thoughout the world to keep access times and speed sufficiently fast. It is also not unusual that large websites and infamous websites have to endure various kind of attacks, or risking both the availability of the service as well as users' private data. Many websites fail to cover the high costs, and are forced to shut down.

Peer-to-peer networks have successfully proved to make distribution of web content much cheaper. BitTorrent is probably the most well-known example, being employed by both open source projects and commercial ones to distribute software and software updates, and employed by websites to distribute large files in general [3][12]. Attempts are also made about every second year to design a new protocol proclaimed to replace or work alongside HTTP as the core protocol for the web, protocols designed entirely with peer-to-peer technology [2].

However, all these peer-to-peer networks have one thing in common. They still depend on a server. The user who wants to access some content must know a trusted handle to this content beforehand, such as a cryptographic hash or public key that signs the content. This trusted handle the user usually finds on a server outside of the network, and thus these peer-to-peer networks have avoided the need to implement any spam protection at all. For BitTorrent, that trusted handle is the torrent file or magnet link.

The problem with making a fully decentralized peer-to-peer network is to find a way to distribute such handles in a searchable way without having to rely on a central server. To make this practical, there absolutely must be a way to filter and block spam. Spam may distract, provoke or even mislead the user. In sufficient large quantities, the user may not even see the legitimate content among all the spam. The inability to block spam on a decentralized network is one of the major things that is preventing peer-to-peer networking from replacing the legacy client-server model, making the web ready for scaling up to future needs.

1.1 Previous Work

The problem with implementing spam protection on peer-to-peer networks is a hard problem and ongoing research. The spam protection methods must both be effective in blocking spam, and scale to a very large number of users, this all while not risk punishing good users. The current state of the art that is actually implemented in real fully decentralized peer-to-peer networks is summarized below.

One file sharing software developed in 2007 uses a gossip protocol where it tries to find peers which are downloading or sharing as many files in common to you as possible, and use what other files they also are sharing as an indication of what you are likely to like. A spammer will have a hard time predicting what files it should claim to have in common with you, and by guessing, different spammers guesses are claimed to zero one another out in the long run. This system is very fast and scales to very many users, but does only give a partial view of the network as it is necessary to keep a size limited list of common files discovered, files which are centered around your current interests. Without this size limit, all spam would be in that list too.

Another file sharing software developed in 2006 also acknowledges the spam problem, but takes the controversial step to solve this problem by releasing its client as closed source, with various obfuscation techniques applied to the binary. By putting various locally enforced limitations and implementing a distributed upvoting/downvoting system, the client software as such is useless to a spammer, any spam will be downvoted quickly. Keeping the system closed source did prove to raise the cost for spammers very much, enough to discourage all spamming. To become useful to a spammer, someone must first reverse engineer the obfuscated protocols or the obfuscated client, or the spammer will be left unable to upload many files or give more than one score per file. But once reverse engineered all spam resistance will be broken.

A more recent file sharing software developed in 2012 is instead using a technique based on so-called trustlists. There are a few so-called pretrusted identities which publish trustlists that are replicated on every node in the network. These trustlists are cryptographically signed, and contains a list of all other identities that are allowed to publish files on the network. Each published file entry is signed by the uploader, and only trusted by others if that uploader's public key is in one of the pretrusted identities' trustlist. You can be added by contacting one of the pretrusted identities through some out-of-band means and ask them to add your key, and they will remove you again if you start spamming. Although very effective in keeping out spam, this method requires every node to know about every publishing identity. It also puts an unreasonable moderation burden on the pretrusted identities if the network grows. The method is decentralized in the sense that any user can substitute the default pretrusted identities in their client to anyone else, if they know the public key of someone they trust does the job well.

A forum system developed before 2008 has implemented a more complicated trustlist scheme often referred to as a "web-of-trust". You assign positive or negative scores to identities, and publish these scores in your own trustlist. In order to find out which identities are trusted, you recursively download signed trustlists from those who you trust, and calculate weighted trust scores for all identities based on transitive trust. Solve a captcha for any already trusted identity to become added in their trustlist with a low score. If anyone trusted gives you negative score for spamming, it outweights your captcha score and you become distrusted. This system does provide very good spam protection without any node having to perform any significant moderation burden. This system has also proved to give an almost uniform view of what constitutes spam even as there are no common pretrusted identities, and it has also proved to be very resistant against any identity getting too much influence on what constitutes spam. But it requires every node not only to know about each publishing identity on the network, but also to keep up-to-date copies of their trustlists and frequently recalculate trust scores, which generates unreasonable amounts of traffic, storage and computational overhead as the network grows.

To clarify, having to know about every publishing identity is a problem, as the list must be kept synchronized at all times, and the number of publishing identities is almost the same as the number of peers on the network. On file-sharing networks for example, measurements show that about 75% of all peers are publishing files [9], and this number should be comparable for other kinds of peer-to-peer networks. This results in way too much storage and bandwidth overhead when the number of total active peers are reaching the millions.

Papers have also been published proposing possible spam protection methods. Some notable examples are mentioned below. Note that some of these papers talk about reputation systems rather than spam protection, but are using the reputation systems to combat spam.

EigenTrust (2003)[5], which is inspired by the famous PageRank algorithm used by search engines to avoid spam, seems to be what spurred most of the research that does exist for spam protection in peer-to-peer networks. It is based on score calculations and transitive trust much like the web-of-trust scheme described above, but the score recalculations are done in a distributed way so that not everyone needs to have synchronized copies of all trustlists. EigenTrust, as described in the paper, does however depend on a central component for Sybil protection.

HonestPeer (2014)[6] is a recent improvement of the original EigenTrust algorithm. This paper also references several other papers analysing the original EigenTrust algorithm, and proposing improvements thereon. However, neither EigenTrust, HonestPeer or any other improvement seems to have solved the scalability problem. They still require that every node knows all identities that are allowed to publish content.

The closest to anything scalable seems to be the system described in another paper (2009)[4]. Instead of keeping a list of all trusted identities locally, a query is made to the network when the trustworthyness of a certain peer must be assessed. Even if this would still be costly if this was applied to the returned search results, as there are many peers that must be assessed, the cost would not increase drastically with the size of the network. The system, as described in the paper, also relies on a central component for Sybil protection.

To summarize all of this, all spam protection methods proposed to this date have serious problems one way or another. It seems the best one can do is to trade one out of content visibility, effectiveness and scalability to get the other two, see Figure 1.2.

1.2 Methodology and Goal

When reading through papers on the subject and looking at the actual implemented spam protection methods, one thing was missing. No one seems to have performed any research on how good spam protection methods based on architectures known to scale seamlessly to a large number of users achieve. One such architecture is the PKI/CA architecture used on the web today to prove associa-

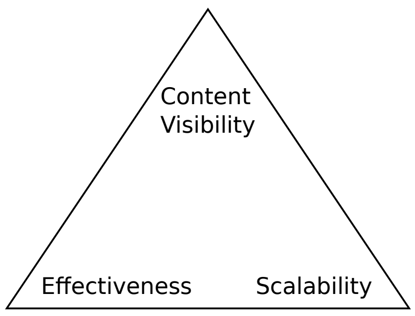

Figure 1.2: The best today's spam protection methods for peer-to-peer networks have achieved is two out of the three properties in the triangle. Content visibility means that peers should be able to see most non-spam content. Effectiveness means most spam content is blocked. Scalability means algorithms must stay fast and moderation burden must stay low for all participating peers even if the network consist of millions of peers.

tion between public keys and domain names. In this architecture the remote party (server) can prove a statement to the first party (client), without the first party having to contact anyone else to verify it. This is done in a way that does not hinder visibility, and it does undoubtedly scale to a very large number of servers. If it turns out that a spam protection method based on it is effective in blocking spam, all properties of the triangle in Figure 1.2 are gained, and the spam problem in peer-to-peer networks would be finally solved.

With this in mind, the goal of this master thesis work was defined. The goal is to design a new spam protection method inspired by the scalability properties of the PKI/CA architecture, and through simulations answer the question of how good spam protection this new method gives compared to a web-of-trust-based trustlist scheme and no protection at all. This should give a hint about whether such new methods can provide good spam protection, and establish a reference that can be improved upon in the future. For the comparison a web-of-trust-based trustlist scheme was selected, as that one seems to currently be the method that provides the best spam protection for decentralized peer-to-peer networks, albeit not algorithmically scalable.

The new spam protection method was carefully designed based on knowledge gained from courses taught at LTH in web security and algorithm complexity, as well as knowledge gained from looking at the current state of the art of spam protection in decentralized peer-to-peer networks.

A simulator was written mimicking a typical modern file-sharing peer-to-peer network, and this spam protection method as well as the two references to compare against were implemented and simulated.

1.3 Delimitations

To limit the scope of this master thesis work, making it more manageable, some delimitations were put in place. First, only a file-sharing network is considered and simulated in this thesis. This choice was made because there exists more information and data about file-sharing networks than other kinds of peer-to-peer networks. This limitation should not pose a problem with regard to generality. Regardless of the kind of peer-to-peer network, the problem with designing spam protection methods is the same, and hence methods designed for one kind of network should be adaptable to other kinds of networks with minimal changes.

Second, only a network based on distributed hash tables (DHTs) is simulated. This since basically all peer-to-peer networks of modern age build on some kind of DHT in order to make the network itself efficient and scalable to a large number of participating peers without sacrifying content visibility. To make an efficient and scalable spam protection method, the spam protection method must be implemented on top of an efficient and scalable network, like this one.

Third, only a peer-to-peer network where anyone can connect to anyone else, and everyone is assumed to be strangers to one another, is considered. There are other kinds of networks such as so-called friend-to-friend networks, where one only connects to those that are already trusted to a certain degree. Those networks typically do not work well unless several of your friends use them too, and stay online all the time, but they do offer many opportunities for spam protection based on this trust, which usually is transitive by nature.

1.4 Thesis Outline

The rest of this thesis is laid out as follows. Chapter 2 presents the theoretical background needed to understand how the spam protection methods work and why they must be designed as they are. In Chapter 3, the simulator is presented. Chapter 4 presents the design of the simulated spam protection methods, and Chapter 5 presents the simulation results. The conclusions and ideas for further improvements are given in Chapter 6.

The reader is assumed to have basic knowledge about networking, public key cryptography, hash tables and time complexity to fully understand the theory presented in this thesis.

Chapter 2

Theoretical Background

Spam protection is a very complex problem, especially for peer-to-peer networks. People capable of spamming usually have strong economical or political reasons to do so, and are willing to break laws. In a sense, spamming is an attack against the network where any weak spot will be exploited by the attacker to get the spam through.

In this chapter, each aspect that has to be considered in order to create a secure and effective spam protection method is explained, to the degree necessary to understand why the spam protection methods in this thesis are designed the way they are.

2.1 The Importance of Captchas

If you ever signed up for an account on a website or another web service, you were probably faced with a so-called *captcha*, often an image of distorted letters that you were asked to read out. This image is constructed in such a way that humans can read it, but even the most advanced character recognition algorithm available today will fail [8].

In order to gain maximum profit from spamming, a spammer typically writes a computer program that automatically signs up for multiple accounts on various web services and starts spamming. Once banned, the program will just sign up for new accounts again, using some other plausible but fradulent account information. But since the computer program cannot read the captchas, it cannot sign up for accounts automatically. Instead the spammer is forced to manually solve captchas, or hire someone that does [8]. Since only spammers face this issue, while legitimate users do not, spamming starts becoming expensive, and the more expensive it gets, the fewer spammers will still stay profitable.

One may think that restricting the number of sign ups per IPv4 address would be a good alternative. IPv4 addresses are becoming increasingly sparse, and therefore increasingly expensive to obtain. But IP addresses will always be much cheaper for a spammer than for legitimate users, as spammers can reuse the same IP address to automatically sign up once to and spam on every single website on the Internet. Futhermore, spammers are ready to deploy malware and infect other computers on the web to get hold of more IP addresses. Captchas do not have these problems, so they are preferable in this context.

Another alternative would be requiring a valid Hashcash [1], in other words requiring a computer to perform a heavy calculation to sign up instead of forcing the user to solve a captcha. However, custom-built compute devices that can calculate Hashcashes more than 10,000 times faster than a powerful desktop computer are selling cheap, yet again putting the spammer in a advantageous position. Requiring human work through a captcha today remains the best countermeasure against spammers.

It is also worth noting that requiring users to sign up for accounts are absolutely necessary for captchas to work effectively. Legitimate users must be able to keep a persistent identity without being forced to solve any more captchas, otherwise legitimate users would not have an advantage over spammers. The account is this persistent identity on client-server networks.

On peer-to-peer networks peers are solving captchas generated by other peers. Therefore the captchas must be of a kind that works even if the spammer has perfect knowledge about the captcha system and can generate captchas of its own, since the whole captcha system is distributed as part of the peer-to-peer software. Captchas with distorted letters are of this kind, but as the day is approaching when character recognising algorithms will become better at recognising characters than humans, new captcha systems must be devised. As for keeping a persistent identity, this must be solved by the actual design of the spam protection method.

Finally, requiring captchas or other proof-of-work is also important to protect against so-called Sybil attacks [13]. Sybil attacks are attacks where a malicious user creates extremely many identities in order to get a large influence on the network. Sybil attacks can also possibly be used to cause a denial-of-service attack. If peers need to store data associated with those identities, they may become overloaded.

2.2 Distributed Hash Tables and Kademlia

Distributed hash tables (DHTs) are the fundamental building block for locating information in a completely decentralized peer-to-peer network. On a decentralized peer-to-peer network there is no server or other central point on which all information is gathered. Because of this, locating information becomes a problem. Distributed hash tables are a way to solve this, while only requiring each peer to know about a tiny fraction of all available information and a tiny fraction of all participating peers.

In a distributed hash table, all information is separated into pairs of a key and a value. The key is something the peer looking for some information already knows, and the value is what is not known. For example, the key may be a keyword to search for, and the value may be a file or some other resource that contains that keyword. Another example is that a key is the ID of a certain file, and the value is the address to a peer that has this file. The key is typically hashed to get a number. This number points out a particular location in the peer-to-peer network. There are typically one or more peers that are responsible for information in that location. The peer that wants to locate some information goes to that location and will find all values that belong to the key. The act of locating such values for a key is called a lookup. A lookup usually returns several values. The act of storing a new value for a key so other clients can find it is called publishing.

The actual process of going to a certain location in the DHT requires contacting multiple peers in several steps. This is required since no peer knows more than a tiny fraction of all participating peers. In each step, the peer performing the lookup discovers the addresses to other participating peers that are more likely to know who is responsible for the final requested location.

The purpose of distributed hash tables is to distribute all information as evenly as possible throughout all participating peers, and to replicate information just enough so that it will not disappear because of leaving peers. By doing this, each peer should only experience a very small fraction of the load a small server otherwise would have seen. Yet, the network can hold an enormous amount of information, and any peer can locate any of this information by performing a lookup for it.

One particular DHT that is commonly employed on real peer-to-peer networks is Kademlia [7]. Kademlia has a proof that a lookup requires contacting at most $O(\log n)$ peers on a network with n participating peers. Kademlia is also robust in situations where many peers continuously join and leave the network, and some peers act maliciously. DHTs are what allowed peer-to-peer networks to scale beyond a few thousand peers to many millions of peers. If any peer would have been required to know about most participating peers or most available information, this would not have been possible. This issue is similar to the issue of spam protection, and that is the reason why spam protection must be implemented on top of a DHT network, or similar, to scale as well.

Finally it can be noted that DHTs are only suited for publishing short messages as information. They are not suited for publishing actual file data, for example. Instead handles to the data are published. They contain the file name, a cryptographic hash over the data, and information about which peers are holding the file. Actual file data can then be transferred using any standard file transfer mechanism. The DHT networks do not provide any spam protection of themselves. The handles published can point to files being spam, or the handles can be spam themselves.

2.3 Diversity of Interests

On a small network, there are probably only a few interests being represented. As long as the network is small, almost only people who have these particular interests are going to use the network. In fact, interests exhibit a clustering effect, where two persons that have a certain interest in common are likely to have more similar or related interests in common. So even if there are several interests represented on the network, the users of the network will access web content related to most of them.

Take for example a smaller Internet forum. There are typically multiple boards, each of which is related to a particular narrow interest. Each board then has threads in which users discuss things related to that interest. The various boards on such a forum are typically covering interests that are similar or at least related to a certain extent. This makes all the boards together cover a wider array of interests, but still clustered. A person frequenting a certain board on that forum likely frequents some of the other boards too, due to sharing those interests too. This makes the work easy for a moderator, the person tasked with detecting and banning spammers and other misbehaving users. The moderator is likely sharing most of those interests, and will thus naturally read most of the boards. Two moderators together will likely cover all boards. A spammer that only targets one particular board will therefore still be detected and banned.

However, when the network increases in size it gets problematic. Now, there are very many interests being represented on the network. So many that most interest are in fact not related at all anymore. Persons with very diverse interests are now using the network. This is the case for large social media sites, and the web in general. But it is also the case for existing large file-sharing networks, to take a peer-to-peer example. Several moderators, let alone a single one, can no longer cover all represented interests. Not only would that require too much work, but it would require the moderators to go through or download content that is not one bit interesting to them. They would not do that naturally, or even voluntarily. In fact, they may not even be able to judge what is spam in the context of interests far from their own.

This problem is important for a spam protection method to solve. A spam protection method where only the judgement of pretrusted identities are used will never work. One example was mentioned in the previous work section. The pretrusted identities would then in a sense be moderators. When only a tiny fraction of all interests are covered by moderators, the field is free for spammers to spam without risk of being banned. A spam protection method must be designed in a way where the judgement of users can be trusted, as opposed to only the highly trusted moderators. There are multiple users for each represented interest after all, so they would know if a spammer starts to spam there. But this issue is tricky, as interests do not appear to form any hierarchy or other easily exploited pattern.

Finally, it can be noted that one pattern has been observed related to interests beside the clustering. Some interests are globally more popular than others [10].

2.4 Spam Protection Using Trustlists

Ignoring the scalability concerns, and just focusing on protecting a small decentralized peer-to-peer network of a few hundred peers, a few spam protection methods have successfully been devised and employed in real networks over the last 10 years. One concept that has proved to give methods blocking spam very well are so-called *trustlists*. Methods based on trustlists may block spam as well as any central web service with appointed moderators, while being entirely decentralized.

A trustlist is basically a signed list produced by a certain identity on the network. The list contains the public keys of other identities that this identity believes should be trusted to not spam. Sometimes the list also contains scores asserting how trusted each of those identities should be. The trustlist is openly published to the network, and peers can download and use it when judging other identities. As trustlists are either propagated to most peers on the network or published on the DHT, they can be downloaded and used even when the identities they are from are offline.

In the simpliest case, the pretrusted identities are the only ones publishing trustlists. Let us say that a peer P wants to perform a search for some web content, but only wants to get web content published by trusted identities. Peer P then downloads, beforehand, the latest version of each pretrusted identity's trustlist. Now, the peer P performs the search. A list of results is returned from the network. Each result entry is a tuple $(data, Q_{pubkey}, \mathrm{SIG}(Q_{pubkey}, data))$, where $data$ is the actual web content or a handle to the web content, Q_{pubkey} is the public key of the publisher, and $\mathrm{SIG}(Q_{pubkey}, data))$ is the signature on the entry made by the publisher. The peer P now checks the results and discards all entries by public keys not mentioned in any of the previously downloaded trustlists. The peer P now verifies all signatures and also discards any entries with invalid signatures. All that is left are the web content published by trusted identities. As seen by this example, trustlists block spam by whitelisting identities that are allowed to publish.

Of course, for good decentralization it is not enough that the pretrusted identities are the only ones publishing trustlists. Every identity, trusted or not, can publish its own trustlist. How it works is described more precisely in a later section, explained using graph theory. But essentially it works by using transitive trust. Let us say that an identity A trusts another identity B to not spam. B in turn trusts another identity C to not spam. If peer P trusts A, that is because A most likely is honest. So P downloads A's trustlist and uses it. But since A in turn trusts B, B is also likely honest. So peer P also downloads B's trustlist and uses it too. Therefore P will also trust content published by C. This is the transitive trust.

Since an identity may go offline for a longer period, each published trustlist also contains a date. If the most recent version of the trustlist from a certain identity is too old, it may be ignored. Using outdated information may cause identities that have begun spamming to still be mentioned as trusted to not spam.

2.5 The PKI and CA Model

On the web today, a *public key infrastructure* (PKI) is employed to allow servers to authenticate themselves to clients. When a client connects to a domain name it has never connected to before, it cannot at first know whether the server that responds is actually authorized to perform communication on behalf of the domain. This since the client does not have any prior information about what servers belong to the domain. The server that responds may be an attacker.

In the PKI model the problem has been solved as follows, omitting the exact implementation details. The client connects to the domain name, and an unknown server answers. The client asks the server what its public key is, and then performs a challenge-response protocol in which the server proves that it possess the matching private key. Now the client knows a public key, and that the public key and server belong to each other, but it still does not know if the public key, and hence the server, belongs to the domain name.

This is the key part of the PKI model. If the server is authentic, it can prove that its public key actually belongs to the domain name. However, if the server never was part of the domain, it will either fail at the challenge-response because it does not know the well-guarded private key, or it will have to present a public key for which such a proof cannot be made.

The proof basically works by having the server provide a valid certificate chain back to a party who the client already trusts. In the PKI model those pretrusted parties are called *root certificate authorities*. Any other intermediate party is just called a *certificate authority* (CA). Let us say that C is the server and D is the client. If C wants to prove to D that (C_{pubkey}, C_{domain}) belong to each other, it may provide that information signed by B, and provide $(B_{pubkey}, B_{is\_authority})$ signed by A, where A is pretrusted and hard-coded in the client's software. D does not need to fetch any additional data, C can provide it with the minimal amount of data needed to prove a claim. Of course, if B did sign the (C_{pubkey}, C_{domain}) information in error, or if the statement is no longer correct, B can not retract that statement unless D queries B too for verification. Same goes for A. In the PKI model this is called *certificate revocation*, and is needed as the statements may become invalid for example due to C's private key leaking and thus no longer being tied to the domain.

Without certificate revocation, the PKI process of proving statements is optimal. A client does not have to contact one single party, even if it needs to verify many statements with completely different certificate chains. Getting a statement signed is also optimal, requiring contacting one single trusted party that performs some checks of whether the statement is true and then signs the statement. However, with certificate revocations, the client must contact multiple parties for each statement it verifies. The client must contact each party in the certificate chain up to the root CA in order to check whether they have revoked their signature or not. This makes it hard to verify more than a few statements in a timely manner. But on the web for usual client-server communication where the number of statements the client must verify is small, the PKI process has proved to scale to a very large number of servers. Yet, the verification times are almost instanteous.

The PKI way of proving statements can be used for other things too, and in other settings. For spam protection in peer-to-peer networks, one can prove statements like $(C_{pubkey}, C_{is\_trusted})$ or $(C_{pubkey}, C_{trustscore})$ instead. A peer that has a valid such statement can then act as a CA itself, signing statements for others. In peer-to-peer networks certificate revocation can not be implemented in a good way, partly because peers are joining and leaving all the time, and partly because each information lookup in the network usually returns many items, each bundled with a statement that must be verified. So the options here are reduced to limiting the validity of each certificate chain to a short amount of time, or making sure the statements are of the kinds that stay valid forever.

2.6 Directed Graph of Trust Relations

One way to visualize what it means to be trusted to not spam in a peer-to-peer network employing a spam protection method is illustrated in Figure 2.1. This

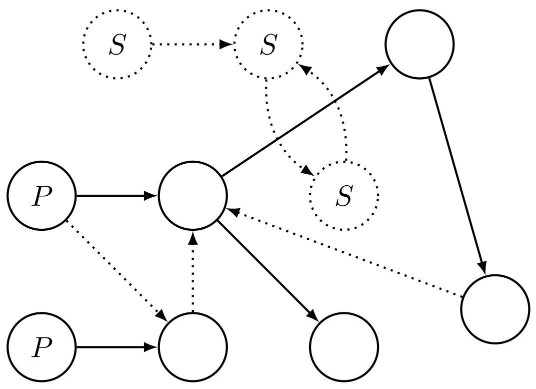

Figure 2.1: A directed graph of trust relations. A node with a P indicates a pretrusted identity. A node with an S indicates a spammer. An edge indicates a trust relation, that an identity trusts another one to not spam. The set of all trusted identities can then be found through a search from the pretrusted identities. One such search result is highlighted above in solid.

is an example where an edge means totally trusted to not spam. There are no edge weights. A new identity that wants to get trusted must contact any already trusted identity, and may for example solve a captcha for that one. The contacted identity will then add an edge, causing the new identity to become trusted by everyone that performs a new search in the graph. Of course, a spammer may also solve a captcha and become trusted. Therefore it is important that an identity can remove an edge it has created, if the one the edge is pointing on starts spamming.

There is a strong similarity between a directed graph of trust relations and how the PKI model handles trust. Pretrusted identities corresponds to root CAs in the PKI model. Spammers correspond to attackers. The main difference is that in the PKI model each edge also contains a statement as described in the PKI section. In the PKI model, an edge is only added after a rigid check of the validity of the statement has been performed by the party adding the edge.

The directed graph of trust relations and the PKI model are actually solving a common problem, how to bootstrap trust. There absolutely must be a pretrusted party. One cannot expect a new user to choose who to initially trust. First off, many users are not knowledgable enough to judge that correctly. Secondly, as a spammer may introduce any number of fake identities, selecting randomly or by vague metrics will risk starting the search from a spammer. By bundling the public keys of the pretrusted parties in the software, searches can always start from known trusted identities. That spammers can introduce a large number of fake identities should then no longer be any problem. Identities that are spamming or that judge spammers as to be trusted will be isolated from the trust graph. Edges to them are removed in response to spam.

Which identities that are pretrusted is usually selected by the one developing the peer-to-peer software. The pretrusted identities must absolutely be highly trusted. They also must be actively using the network, so that they can react if their trustees start spamming. If the pretrusted identities start misbehaving, the software developer must change them in a software update. If there are several softwares for the same network, most pretrusted identities are pretrusted in all of them, to maintain compatibility.

2.7 Shallow Graph and Web-of-Trust

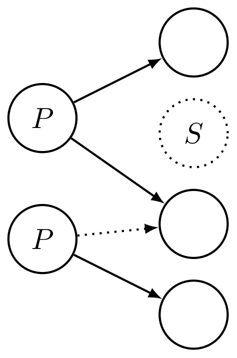

Figure 2.2: A shallow trust graph. Only pretrusted identities may create trust relations, limiting the depth of the graph to 2. An example search is shown in solid.

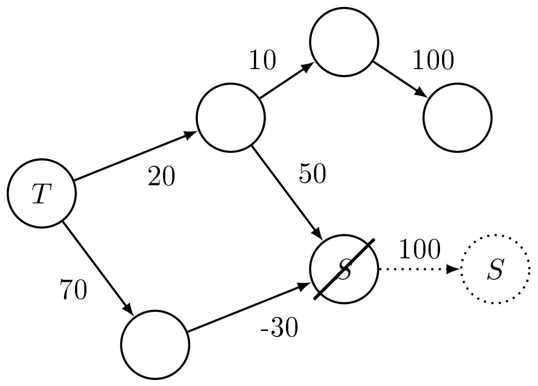

Figure 2.3: A web-of-trust trust graph. Here an identity starts the search in itself T, rather than from any pretrusted identities. Each edge also has a score. An identity may be reachable but considered banned due to the weighted total score on it being negative. An example is shown. The -30 score is weighted by 70, whereas the 50 score is only weighted by 20.

In the previous work section two trustlist-based systems were described, both employed on real peer-to-peer networks today. The first one is a directed graph of trust relations where only the pretrusted identities can add edges. This means the trust graph is shallow, see Figure 2.2. The problem with this is that the pretrusted identities must moderate a very large number of edges each, including generating all captchas. Increasing the allowed depth is crucial for distributing the load, and making the system more decentralized.

The second trustlist-based system allows for a larger depth, but is more complicated. This kind of trustlist-based system is sometimes referred to as a web-of-trust. An example is shown in Figure 2.3. An identity can add edges with positive or negative scores to other identities. Usually these scores are added manually by the user. A positive score means the identity trusts that other identity to not spam and to not add edges to spammers for free. A negative score means the identity thinks that other identity is a spammer or is helping spammers. When an identity is adding an edge to a new identity in response to a solved captcha, that is usually added with a zero score. This is enough to make the new identity discoverable with a non-negative score.

The exact way the web-of-trust system is designed may vary. Some implementations have two separate scores for each edge. One score means trust not to spam. The other score means trust to not add edges to spammers for free. The reason for this separation is that an identity may publish good web content while being terrible at judging others. Another difference between implementations is how deep into the graph scores are used. The whole graph is always followed to discover all identities. However, score calculations are very heavy, especially when the graph contains very many trust relations. If there are spammers on the network that are likely to bother you, those you trust have likely been bothered by them too, and scored them negative. Therefore, the score calculations may be limited to stop already after depth 2.

An identity usually adds edges to the pretrusted identities with a positive score by default. This is needed to bootstrap trust. This way the graph search can be performed even before the identity has added any edges of its own. But the fact that an identity does not have to trust the pretrusted identities is also an important feature with regard to decentralization. An identity may choose to distrust any or all of the pretrusted identities. By doing so, the identity may start to see those that the pretrusted have scored negative, or hide those that the pretrusted have scored positive. Choosing who one trust changes who is visible to be closer to what oneself believe it should be.

2.8 Searching the Trust Graph

The web-of-trust system actually performs a search using knowledge about all identities that are reachable in the search, as well as all edges. This information is what is published in the trustlists described earlier. The trustlists are fetched from the network as the search is performed. In a real network there may be millions of identities, and hundred of millions of trust relations. This is the problem that causes spam protection methods such as web-of-trust to not scale. Even retrieving information about all trust relations becomes a big burden for each peer.

However, as proved by the PKI model, one does not need to perform this search if there are common pretrusted identities. Let us say that the identity furthest to the right in Figure 2.1 wants to prove to someone that it is trusted. Then it just provides a statement that it is trusted, signed by the previous identities hop-by-hop, following the solid edges backwards all the way to the pretrusted identity. As mentioned in the PKI section, for performance reasons this is optimal. But in the case of web-of-trust, this cannot be done. The one that wants the proof may not trust any of the identities in the provided chain directly. And to further complicate things, negative scores must be taken into account. This means either the prover or the one wanting the proof must resort to performing the slow search using full knowledge about all identities.

There is one other thing to note about the performance aspect of searching the trust graph. Any identity can create edges to any other identity as it wants, without requiring captcha solutions. A spammer that has managed to become trusted may create a very large number of fake identities. It can arrange these fake identities in various ways in order to make peers waste a lot of time performing the search. If each fake identity is pointing to the next one in a long chain, a signature chain could easily be made to contain many thousands of signatures, making the verification of the signature chain very heavy. The number of signatures in a signature chain is the depth of the network, so it can easily be limited. A signature chain with more than 10 signatures could be rejected as invalid. In the case of web-of-trust where a full search must be performed, also the number of outgoing edges from each identity must be limited. But doing that in a good way is not trivial.

2.9 Revocation of a Trust Relation

One problem with all methods expressed as a trust graph is that only the identity that has added an edge to another identity may remove that edge again. The identity that has added an edge may fail to realize that the identity it points to has started spamming. Even worse, the identity may even be offline and thus not able to act. And the problem is not made better by the fact that edges usually are added automatically in response to a solved captcha.

For web-of-trust this problem was solved by the scores. Any identity can give a negative score to another one. The sum of weighted positive and negative scores may become negative, thus any trust relation is effectively overridden. In the example shown in Figure 2.3, the identity T trusts the one who gave negative score more than the one who gave positive, such that the weighted sum becomes negative. In most cases a spammer would not have been given any positive scores at all. Even a single lowly trusted negative score would be enough to make the spammer considered banned. This method works because a full search is performed.

However, if fetching trustlists and performing searches are to be avoided by doing it the PKI way, scores cannot be used. Instead, the revocation problem of PKI applies. As mentioned in the PKI section, the only reasonable way to implement revocation is by putting a restrictive time limit during which a signature stays valid. This forces each identity to renew its proof of being trusted often, something that it will fail to do if the truster has removed its trust relation due to spam.

This is enough to ensure that the information of a removed trust relation is taken into account within a reasonable time frame. This is also enough to handle the case where the truster has gone offline, as the identity must renew its proof from someone else instead. This means each identity may have solved captchas for several other identities, renewing its proof from whoever is online at the moment. However, it is not enough to handle the case where the truster fails to realize that the identity has begun spamming. One reason the truster fails to do this is due to the diversity of interests. The spamming identity may spam in interest categories the truster does not know about. In fact, not even the truster's truster, and so on up to the pretrusted identity, may know about these interest categories.

For web-of-trust, the ones a peer starts the graph search from are usually identities that have similar interests as the peer itself. Thus, negative scores can be used and trusted. The diversity of interest is taken into account. But when doing it the PKI way, there absolutely must be a way in which other peers can inform a truster to remove its trust relation to a certain identity. This can be solved either by introducing a shared notion of who are more trusted such that more trusted identities can ban less trusted ones, or by allowing anyone to inform a truster of an identity about mischief. Either way, this must be done in a secure way such that no peer or identity ends up in a situation where it can affect what is to be considered spam, such that non-spam is blocked or some spam is allowed.

Thus far this section has handled the case where the act of distrusting spammers is problematic because of the diversity of interests. But something can also be said about who should be the one deciding what constitutes spam. In the ideal case each user can decide what is to be considered spam itself, and web-of-trust allows for this. But when this is not possible, it can be worth thinking about whether pretrusted identities can be trusted instead to decide what is to be considered spam. Another option would be to design the spam protection system in such a way that what is considered spam is roughly decided by the majority of trusted identities on the network.

2.10 Identifying a Spammer Collective

A spammer may create any number of fake identities. The spammer can create edges between these fake identities in any way it wants, without having to solve any captchas. These fake identities form a collective. In order for the collective to become trusted, it is enough that any of the fake identities are solving a captcha from a trusted identity. After that, they are all reachable in a search. In an attempt to avoid becoming distrusted and having to solve a new captcha, the spammer may behave perfectly honestly with some identities. See Figure 2.4 for an example.

The problem is that a peer that sees spam published by an S identity in Figure 2.4 cannot know who to give a negative score to. Giving a negative score to any of the S identities or to I_2 is pointless. The spammer can just throw them away and create new identities, adding new trust relations from I_1. Likewise, giving a negative score to the identity before I_1 will cause many identities belonging to honest users to become distrusted. Regardless of whether negative scores are

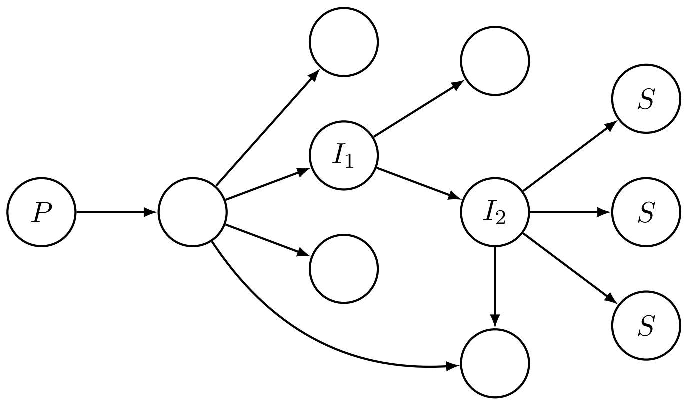

Figure 2.4: A graph of trust relations illustrating the problem of removing the trust to a spammer collective. All I and S identities are owned by the spammer. Only I_1 has actually solved a captcha, the spammer can create edges to others for free. The spammer is acting honestly with the I identities, not publishing any spam. If a peer sees that an S identity is spamming, how will it know who to give a negative score too?

added manually by a user or automatically, correctly identifying I_1 to be the offender is a tricky task.

One way to lessen the problem is to decrease the maximum allowed depth of the graph. In that case, guessing who is the offender becomes easier. The graph in Figure 2.4 has depth 5. With a maximum allowed depth of 10, there could be many more I identities. That would make it much harder to identify who to give a negative score to.

This problem only exists if marginally trusted identities are allowed to add new trust relations. In a shallow graph where only the highly trusted pretrusted identities can add trust relations, this problem would not exist. However, as mentioned earlier, shallow trust graphs are not a scalable solution. Also, on small networks with only about 100 honest identities, the task of manually identifying I_1 is much easier. This since the user may have knowledge about each identity's prior behavior and trustworthyness. This is no longer the case when the network grows in size.

Chapter 3 The Simulator

In order to evaluate the effectiveness of the proposed new spam protection method, it has to be simulated and the results compared to those of an existing spam protection method. In the initial phase of this master thesis work, some research was conducted to see what simulator frameworks already exist and how spam protection has been simulated in the past. The purpose was to see how much can be reused.

Unfortunately, no existing simulation framework suited for simulating spam resistance was found. The simulation frameworks that do exist for peer-to-peer networks are for simulating on a lower level, and it is not apparent whether they can be repurposed for simulation on the level where spam protection is implemented. The best lead on how spam protection can be simulated is what was given in the EigenTrust paper [5]. However, EigenTrust did not consider the existence of captchas and the fact the peers can recreate their identities, so some parts must be done differently. For these reasons it was decided to write the simulator from scratch, mostly based on what was given in the EigenTrust paper, but adapted to consider captchas, DHT networks and other theory outlined in Chapter 2.

In this chapter the design choices for the simulator are presented. This chapter also described what the simulator measures, how and why. The actual spam protection methods are described in the next chapter.

3.1 Peer Behavior and File Model

The simulator is a discrete event simulator. Several peers are simulated, participating in the same simulated network. Upon creation each peer schedules the first action it will perform, such as go online, at a certain point in time. Once all peers are set up, the simulator picks the scheduled event that will occur at the earliest point in time, and the peer who scheduled that event performs its action and then schedules the next action it will perform. This is done until a certain number of virtual days have passed. Spam protection methods are simulated one at a time. Results are gathered throughout the simulation runs or at the end of each simulation run as needed.

The design of a peer's behavior is complicated and is mostly based on the same data and models [9][10] that EigenTrust used. The purpose is to have a model that at least to a certain degree approximates the behavior of real peers, since that can affect how well different spam protection methods work. The exact model used is explained below.

In order to properly model the diversity of interests described earlier, which is important when simulating spam protection, the network is modeled to have several content categories. A content category roughly corresponds to an interest a person can have, such as "cute kitten pictures" or "science documentaries". However, in the simulator it is just a number. When the simulation is first started, each peer selects a few content categories they are interested in. Peers only ever search for or publish files within their selected content categories.

Each file on the network is modeled to be uniquely identified by the content category it belongs to and a rank within that content category. There can thus be many files which belong to the same content category, but each file may only belong to a single content category. No filenames are used. In order to capture the fact that some files may be more popular than other files within the same content category, the rank is used. A file's popularity is modeled using a Zipf-law from the file's rank. This means the file with rank 2 is half as popular as the file with rank 1, the file with rank 3 is one-third as popular as the file with rank 1, and so on. This is argued in one of the aforementioned papers to capture the reality well [10]. When a peer wants to search for a file within one of its selected content categories, it randomly selects which file to search for based on their popularities.

A peer is not equally interested in all content categories it has selected. How interested a peer is in each of its selected content categories is selected from a uniform distribution. This means a peer may perform more searches within one of its selected content categories than within another. However, when a peer is selecting which content categories it is interested in, that is just like files modeled using a Zipf-law from a rank. Each content category has a rank, and that decides how many peers are interested in that content category on average. A peer also decides upon simulation start a set of files it shares, based on its interests and file popularities just like for file searches. A peer never shares duplicates. While online, the peer periodically publishes file handles to its shared files to the network.

The total number of content categories and the total number of files are proportional to the number of peers being simulated. The total number of files within each content category is proportional to the popularity of the content category. How often peers go online, how long they stay online, how many files they share and how many searches they do are based on measurement data from one of the papers [9]. These are measurements from real peer-to-peer file-sharing networks, and should be accurate within that domain. Each peer is selecting unique values here though, using the appropriate random distributions.

All spam on the network is assumed to be of the kind incorrectly labeled files. This means that a peer who has performed a search for a file cannot know whether the returned file handle is authentic or spam. In a real network the peer would have to download the file to know. To model this, each file handle also has a flag indicating whether it is authentic or not. A search for a certain file can then return up to two search results, one handle for an authentic version and one for the spam version. The peer does not look at that flag, because it is not supposed to know, and is thus forced to randomly select one of them. However, when a spam protection method is employed, the spam version may be filtered

Property	Pretrusted Peer	Honest Peer	Spammer
Searches for files	yes	yes	no
Downloads files	yes	yes	no
Marks files good/bad	yes	some do	no
Publishes files	yes	yes	yes
Publishes authentic files	yes	yes	no
Publishes malicious files	no	no	yes
Always online	yes	no	yes
Has a few selected interests	yes	yes	yes
Only searches within interests	yes	yes	-
Only publishes within interests	yes	yes	yes

Table 3.1: Overview of some aspects of the simulated peer behavior.

and not returned. File handles published by honest peers are always authentic. File handles published by spammers are always spam.

When an honest peer has randomly picked one of the up to two returned search results for a certain file, the peer may mark whether that file handle was authentic or not. In a real network the peer would indeed know this after having downloaded the file. Spam protection methods can then use this information. This is modeled such that only a certain fraction of all peers actually mark whether a file is authentic or not once it knows, but those that do always mark all files and always mark them correctly. Spammers never search for or mark files.

Searches, publications and markings are done the same way regardless of spam protection method being simulated. This is important in order to facilitate comparison. Peers must not behave differently just because another spam protection method is being simulated. Thus, all of this results in an abstraction between the simulated spam protection methods and the simulated peers.

Table 3.1 gives an overview of the peer behavior described here and in the following sections.

3.2 Simulating Spam Protection Methods

A total of three spam protection methods are simulated as part of this master thesis work. Each of the spam protection methods is implemented to run on top of a DHT network. Almost all modern fully decentralized peer-to-peer networks use DHT networks of some kind in order to be able to scale efficiently to a very large number of peers. Implementing spam protection methods on top of a DHT is necessary to allow for the spam protection method to also scale to a very large number of peers. The DHT network is also simulated without any spam protection, as a baseline in the comparison. That way one can compare the effectiveness of a spam protection method, not only to other spam protection methods, but also to when using no spam protection at all.

The DHT network is implemented in the simulator simply using a hash table. The searching for and publishing of file handles is realised by lookups and insertions into that hash table. The content category the file belongs to, the rank the file has within that content category and whether the file is authentic or not is hashed together as a key. Any information that a spam protection method needs to store together with a file handle to facilitate verification for those that get the file handle in search results are stored as a value. An expiration date of about half a day into the future from when the file handle was published is also stored as the value. File handles need to be republished frequently, possibly with new verification data, in order to stay accessible.

The lower level details of a DHT network about how each peer is responsible for certain parts of the DHT keyspace is not simulated, as that is not relevant for the higher level spam protection is simulated at. Spammers are also assumed to not be able to subvert the DHT network on a lower level in any way, such as to cause misrouting of information or denial of access to certain parts of the keyspace. Although this also is an interesting topic, it falls outside the scope of this master thesis work.

3.3 Simulating Spammers

Deciding how spammer behavior should be modeled is important when constructing a simulator for spam protection methods. There are many ways in which spammers can attempt to subvert the system, in order to gain an advantage. For example, if a spam protection method has a crucial weakness that allows spammers to circumvent the spam protection altogether, the spam protection method would be worthless. Simulating a behavior of spammers that does not take advantage of that weakness would therefore not give a correct value for how effective the spam protection method is in practice. When evaluating how effective a spam protection method is, it is important that spammers always behave in the way that makes them as powerful as possible, without being unrealistic.

The ways in which spammers can act to undermine the effectiveness of a spam protection method are called *threat models*. Different threat models may be differently effective against different spam protection methods. In the simulator, the spammers always act according to the same threat model. This threat model is believed to be the most effective against all spam protection methods simulated as part of this master thesis work, while being realistic. The threat model was found by reasoning about which behavior would be the most beneficial for spammers, based on the design of the spam protection methods.

Spammers always stay online at all time. This is reasonable, as it is usually cheap for a spammer to leave a computer running all the time. Spammers always only publish spam, but otherwise they are behaving just like honest users when publishing. Different spammers do not cooperate. They are unlikely to cooperate in practice. When at all possible, a spammer forms a spammer collective using fake identities. The spammer then behaves entirely honestly with the identity who solved a captcha, except that it does not publish any files with it. Any spam files are published by fake identities that are as deep as possible with regard to the maximum allowed depth of the trust graph. This is the most advantageous configuration for the spammers. The deeper the spammer collective is, the harder it will be for honest users to guess which identity solved a captcha, and thus can be banned.

At one point towards the end of the simulation implementation phase, it was tested what effect it would have if spammers let some of their published files be authentic. This would possibly give spammers a slight advantage against the web-of-trust trustlist-based method being simulated, as spammers have a chance to obtain good scores too, not just bad scores. However, this was found to not aid the spammers. Thus, this threat model was not considered further, and not simulated. Against the new proposed PKI-based spam protection method, spammers cannot gain anything from behaving partially honest. There are no scores they can hope to increase.

3.4 Metrics

In the EigenTrust paper [5], it was measured how many authentic files that were downloaded by honest users compared to how many bad files that were downloaded. This worked well for measuring the effectiveness of their reputation system used for spam protection. However, the spam protection methods being simulated in this master thesis work also considers captchas among other things.

It quickly turned out that it is hard to design a "hardness" for the captchas in the simulator. To further complicate things, that "hardness" would have to be different for the various spam protection methods being simulated. Due to this, it was decided to measure the effectiveness of the spam protection methods in another way.

In the simulator, both honest peers and spammers are assumed to be able to solve an unlimited number of captchas. They also make sure to stay unbanned at all time. If a peer believes it is not trusted, it will immediately try to become trusted. First it will try doing that without solving any captchas, otherwise it will find someone to solve a captcha for and do that. The fact that both honest users and spammers solve just enough captchas to stay unbanned at all time is important. This will allow to measure the effectiveness of a spam protection method by measuring how many captchas each spammer would have to solve on average compared to the number of captchas each honest user have to solve on average. There is no longer any need to define a "hardness" of captchas.

This spammer:honest captcha ratio gives a good indication of how aggressively spammers are banned. For a good spam protection method, each spammer should have to solve many more captchas than each honest user to stay unbanned. The extra work spammers must perform to stay unbanned is hopefully enough to make spamming too expensive in practice. So the more captchas spammers must solve than honest users, the better the spam protection method must be. In contrast, if honest users are found to solve the same number of captchas that spammers have to, spammers do not get punished at all for the spam they post. In that case, the spam protection method would be worthless.

The spammer:honest captcha ratio does only give meaningful values if all users, honest users and spammers alike, really do stay unbanned at all time. To ensure this really is the case, the number of good and bad file downloads are still measured as well. They are ensured to be the same regardless of spam protection method being simulated. Anything else would indicate a bug or design flaw. If some good or bad files disappeared, that would mean some users failed to stay unbanned.

The simulator has various parameters that can be adjusted. Besides simulating different spam protection methods, different number of participating peers, different fraction of spammers and so on are simulated. This is presented in more details in Chapter 5. To reduce the variance of the measured results, the average of 100 simulation runs for the same parameters is used.

Chapter 4

Spam Protection Methods

In this chapter the three simulated spam protection methods are presented. Those are the new scalable PKI-based method proposed in this thesis, a shallow but not scalable version of that one, and an existing web-of-trust trustlist-based method to compare against.

4.1 New PKI-Based Method

The new spam protection method created as part of this master thesis work is the most important one. It is the only one out of the three spam protection methods that is scalable. It is based on the PKI/CA architecture to prove statements like $(C_{pubkey}, C_{is\_trusted})$.

At first there are only the pretrusted identities. The pretrusted identities may sign file handles for files they publish themselves, and may also sign the public keys of other identities. The latter means the pretrusted identity believes that the other identities should be trusted to not spam. When a pretrusted identity makes the signature it also sets the time when it should expire. In the simulator that is the same time as the validity time of a published file handle, for example half a day. The signature from a pretrusted identity is a start of a certificate chain. An identity that has a valid certificate chain for its own public key may now extend this certificate chain by signing file handles and public keys on its own. This creates the trust graph, the directed graph of trust relations. The depth is limited to 5, to keep the problem of spammer collectives manageable. If each identity signs 100 other identities, that depth is enough to support a network with 100,000,000 identities.

Since extending the certificate chain requires trusting the new identity to not spam to a certain extent, a solution to a captcha should be required first. This spam protection method works as long as the pretrusted identities and most other honest identities require that. To find a random already trusted identity to solve a captcha for, a peer may just do a lookup in the DHT network. All identities that are online and can extend their certificate chain may publish this fact to a known place in the DHT network. How this is best done in practice to avoid heavy load on a single point of the DHT keyspace is outside the scope of this thesis work. In the simulator everyone publishes to the same place, but only the most recent 100 entries are kept.

A file handle or any other entry published to the DHT network will be rejected and dropped by all peers if it lacks a valid signature from a valid certificate chain. It will also be dropped when the certificate chain expires. The publisher is expected to republish the entry before that with a newer certificate chain. The reason why entries can be dropped altogether is because all peers have the same view of who should be trusted, assuming they trust the same pretrusted identities. An identity that must have its certificate chain renewed should if possible contact any of the identities it has solved a captcha for before. Those identities will if possible just renew the certificate chain, without requiring yet another captcha. This is the persistent identity mentioned earlier in chapter 2. In a sense, the trust relation remains. If all those identities are offline or otherwise unable to renew the certificate chain, one may need to solve a captcha for another trusted identity.

This far everything has been uncontroversial. However, a solution for the revocation of trust relations problem explained in Chapter 2 must also be made. For this it was decided to use captchas as well, as the security properties and consequences of that is easy to analyze. When a peer has downloaded a file and marks it bad, it will look at the certificate chain for the file handle to decide who should be banned. The certificate chain also tells who is trusting the one who should be banned. The peer can then contact that truster and solve a captcha for it. If the peer succeeds, the truster assumes that the ban request is valid and removes its trust relation to the one who should be banned. This means, the next time the now banned identity wants to renew its certificate chain, it cannot do that from this truster without solving a new captcha. The truster will now treat the banned identity as if it was a new unknown identity. Banning is possibly a bad word, as the identity may have several trust relations pointing to it. But if it keeps getting banned, it will eventually not, and must solve new captchas.

The security of this banning system builds on manual proof-of-work. As long as the majority of users solving captchas are honest, spammers will be banned. This is analyzed in more details in the conclusions. An alternative system where the truster must verify the ban request is unreasonable. That would require additional work for the truster, and possibly the downloading of a large and totally uninteresting file. So an automated system like the one used is a must.

This spam protection method does put some additional requirements on the pretrusted identities besides them behaving entirely honestly. At least one of all pretrusted identities must be online at all time. If all pretrusted identities went offline or otherwise became unavailable, all certificate chains would eventually expire without possibility of renewal. For this reason, the pretrusted identities are always online in the simulator. This requirement is not a problem in practice, many users leave their computers running at all time. Would the disaster happen where all pretrusted identities becomes unavailable, peers can sense this. If no one is able to offer extension or renewal of certificate chains, the spam protection method can be switched off. Otherwise all content on the network would be dropped. The DHT network can be used to guard the pretrusted identities from being overloaded, for example guard them from distributed denial-of-service (DDoS) attacks, at the cost of lowered speed.

A final note, although not covered in the simulator, is that a spammer may also supply unsolvable captchas or captcha images containing spam, and similar.

Even identities spamming or behaving in a disruptive manner in this way can be banned in this spam protection method.

4.2 Shallow Version of the PKI-Based Method

A shallow version of the new PKI-based method is also simulated. In it, the depth is limited to 2, but otherwise it is the same as the real one just described. This depth limit means only pretrusted identities can add trust relations to other identities. This shallow version is not scalable. It is simulated only to aid the analysis of the results for the real PKI-based method.

Reducing the depth limit to 2 has the effect that spammer collectives cannot form, as explained in Chapter 2. However, shallow graphs are not scalable. If there are 10 pretrusted identities that should support a network of 100,000,000 trusted identities, each pretrusted identity would need to manage 10,000,000 of those. Since all identities would need to refresh their certificate chains periodically, this means many incoming connections and many asymmetric crypto operations per second. Depending on the public key algorithm used and thus the key sizes, even storage may become problematic. All trusted public keys may not be able to be kept in the RAM, requiring many disk accesses per second to. Storing 10,000,000 public keys of 4096-bit each would require 5 GB of storage. With modern elliptic curve cryptography and 256-bit public keys this is less of a problem though. Lastly, the pretrusted identities must generate and verify all captchas both for adding trust relations and removing them, and handle all communication traffic for those requests.

The pretrusted identities effectively end up having to carry the load that a typical server carries. However, although this violates the definition of truly decentralized, it is not neccessarily that bad. In the web-of-trust trustlist-based method every peer would have to carry a server load, due to not being scalable at all. Here only the pretrusted identities need to. Still, it offers some advantages over a peer-to-peer network with a completely centralized spam protection system. The pretrusted identities do not need to find the spammers themselves, nor do they need to verify ban requests manually. Also, there are multiple pretrusted identities, so the load is distributed to a certain extent, and the network is more resilient against some failure scenarios.

4.3 Web-of-Trust Trustlist-Based Method

To compare the effectiveness of the new PKI-based method against something, an existing trustlist-based method of the web-of-trust kind was chosen. They are known to provide very good spam protection, while being entirely decentralized. The effectiveness is state-of-the-art for decentralized peer-to-peer networks. Including a method of this kind in the comparison is very meaningful to evaluate the effectiveness of the new PKI-based method and its shallow counterpart.

To the best of this thesis author's knowledge, the web-of-trust kind of spam protection methods have never been described in any published paper before.

Therefore, the complete method is described here. There exist several implementations of web-of-trust-based spam protection methods, all slightly different in their design. The one described and simulated here is the one from the decentralized forum system mentioned in the previous work section. The design of the method was reverse engineered by reading the source code for the forum client.

Each identity publishes a trustlist containing scores for zero or more other identities. There are two kinds of scores, called *message trust* (MT) and *trustlist trust* (TLT). Each score can be in the interval 0 to 100, where 50 is considered neutral. A score below 50 is still going to be referred to as a negative score, and positive means above 50, because that is the meaning the scores have. At first, each identity only trusts itself and the pretrusted identities. Each identity has set 100 MT and 100 TLT on itself, and 50 TLT on each pretrusted identity. No message trust is set on the pretrusted identities.

The core of this web-of-trust trustlist-based method is the score update function. Each identity locally keeps track of four scores for each other identity. Let us call them local message trust, local trustlist trust, effective message trust and effective trustlist trust. The local scores are those the identity have set itself. The effective scores are the weighted average of what those identities that have positive local trustlist trust have scored.

The effective scores are calculated as follows. First a set A is formed containing all identities with a local TLT of 50 or above. However, those identities that currently have an effective TLT of below 30 are excluded from A. The new effective MT and TLT for each identity is calculated as the weighted average of how each identity t from A has scored it, weighted with the local TLT for t. As can be seen from this description, the scores from identities further away than depth 2 are not used at all. This is probably this way to keep the score calculations somewhat efficient. An example with calculations is shown in Figure 4.1.

File handles and other content from identities with a local MT below 50, or an effective MT below 30 are ignored. Trustlists are not downloaded or traversed in the graph search for identities with a local TLT below 50, or an effective TLT below 30. Identities that should be trusted are discovered through a graph search from oneself. Identities that can be found through the graph search are trusted by default. It is not until they get a low enough local or effective score that they are distrusted. In the simulator all peers are assumed to always have the latest edition of all trustlists. The delay for how updated score information propagates in the network is thus not simulated.

In the forum system where this web-of-trust spam protection method is found, users are expected to assign scores manually. Users are expected to have good judgement of who to assign good scores to, and who to assign bad scores to. However, in the simulator, scores have to be assign automatically. It was decided to raise the MT and TLT by 5 units for the publisher of each good file downloaded and marked as good. The exception is if a negative local TLT has already been assigned, in which case only the MT is raised. If a score is missing, it is assumed to be 50 currently. If a bad file was downloaded and marked as spam, the scores are lowered instead. Both the MT and TLT is lowered by 30 units in this case. The identity the scores are lowered for is not necessaily the publisher. Spammer collectives are assumed to exist. When guessing who is the spammer, identities

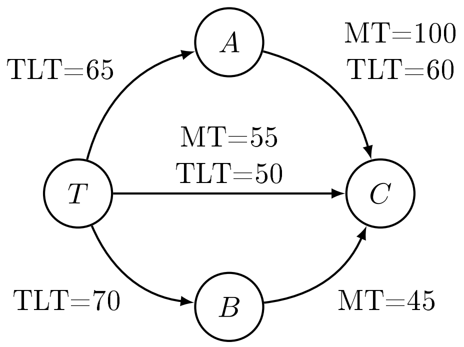

Figure 4.1: An example of how identity T calculates the effective message trust and effective trustlist trust on identity C from the shown local scores. The effective MT is $(55 \cdot 100 + 100 \cdot 65 + 100 \cdot 50 + 45 \cdot 70)/(100 + 65 + 50 + 70) \approx 71$. First is our MT on C, our TLT on ourself is 100. Next is A's MT on C, our TLT on A is 65. Next is C's MT on itself, our TLT on C is 50. And so on. Effective TLT is calculated the same way, but this time B is not used as it does not have a TLT on C.

that have an effective MT of 51 or higher currently are assumed to not be part of the spammer collective. This reduces the options somewhat compared to for the PKI-based method. In the real world where spammer behavior is more complex, the threshold of 51 may need to be replaced with a higher value.

As mentioned before, some users in the simulator do not mark whether a file they downloaded is authentic or not. Due to this, they will never set any local scores on any identities besides the pretrusted identities. Therefore, effective scores are only calculated for those identities that the pretrusted identities have scored. If the pretrusted identities would not set any scores, this would make the aforementioned users much more vulnerable to spam. In the simulator, the pretrusted identities are therefore always marking files they downloaded as authentic or not. This is reasonable, as pretrusted identities tend to care about the network enough to do this in practice. But still, those users may be left vulnerable to spam in interest categories outside what the pretrusted identities have. This is a weakness with the web-of-trust trustlist-based spam protection method. It only works well as long as users are scoring other identities diligently, or as long as the network stays reasonably small.

As mentioned in the web-of-trust section in Chapter 2, the web-of-trust concept is not scalable. For this reason, the web-of-trust concept has only successfully been implemented on small peer-to-peer networks to date.

4.4 Time Complexity

The time complexity of the methods is not simulated. It can be calculated from their design. Let n be the number of peers on the network, and let us further assume that each peer has a single trusted identity. Therefore, n is also the number of trusted identities on the network. Let us also assume the network is a DHT network. In the new PKI-based method, to get a new identity trusted one must perform a lookup on the DHT network to find some identities that may become the new trusters. One must then contact one of them and solve a captcha, after which one receives a valid signature. The most expensive operation here is the DHT lookup of time complexity $O(\log n)$. Offering captchas to others require a DHT publish operation, yet again with $O(\log n)$ time complexity. To verify whether an identity is trusted one must simply verify each signature in a received certificate chain. Since the chain has a limited length, this operation has time complexity $O(1)$. Since the number of identities that must be verified between each renewal of one's own certificate chain can be considered asymptotically constant, the total time complexity for each peer is $O(\log n)$. Each peer is assumed to only keep track of a constant maximum number of identities it trusts, thus the storage complexity per peer is $O(1)$.

For the web-of-trust trustlist-based method it is worse. Here one can see why it is not scalable. To verify whether an identity is trusted one must beforehand have processed trustlists for each trusted identity on the network. On a DHT network, that gives a time complexity of $O(n \log n)$, assuming score calculations are restricted to direct trustees, as done in the simulator. The actual verification can after that be done with a time complexity of $O(1)$, if one uses a hash table. Finding someone who can become a truster for one's new identity can also be done in $O(1)$ when all trustlists have been processed. Offering to become a truster oneself, or publishing one's own trustlist, requires a DHT publish operation, which has time complexity $O(\log n)$. The actual processing of trustlists, or traversal of the trust graph, is therefore the most expensive operation. This spam protection method therefore requires $O(n \log n)$ time complexity per peer. The storage complexity is $O(n)$, as each trustlist or trusted identity must be stored in some manner. The calculated time complexity and storage complexity assumes each identity has a limited and thus constant maximum number of trustees.

Here it is clear that the new PKI-based method is at least as scalable as the underlying DHT network, whereas in the web-of-trust trustlist-based method each peer would see at least quasilinearly increased load as the number of trusted identities increase. Sometimes one may want to look at the network-wide time and storage complexity instead, reflecting the load the network as a whole is facing. Since all n peers are equal, the network-wide time complexity for the PKI-based method becomes $O(n \log n)$. The network-wide storage complexity is $O(n)$. For the web-of-trust trustlist-based method however, the network-wide time complexity is $O(n^2 \log n)$ and the network-wide storage complexity is $O(n^2)$.

Table 6.1 summarizes the time and storage complexities for the various spam protection methods simulated. That table also mentions the shallow version of the PKI-based method, which sees $O(n)$ time complexity as each pretrusted identity gets contacted by $O(n)$ identities.

Chapter 5 Results

In this chapter, simulation results from the simulator outlined in Chapter 3 are presented. All three spam protection methods from Chapter 4 are simulated for various choices of simulation parameters. The three methods are the PKI-based method, a shallow version of that one, and the web-of-trust trustlist-based method. The PKI-based method is the new method proposed in this thesis. It is the only one out of the three that does scale to a large number of participating peers. The other two do not scale at all.

5.1 Primary Results

First, the measured effectiveness in blocking spammers is presented for the three spam protection methods. Figure 5.1 shows the results for the fairly small default network simulated. The parameters used are shown in Table 5.1. The network contains $n_t = n_p + n_h + \gamma n_h$ real users. Each real user has exactly one identity, except for each spammer who may form a spammer collective using many identities of its own. Any user may recreate their identities if they get a permanent bad score, which may happen for the web-of-trust trustlist-based method. As mentioned in the metrics section in Chapter 3, a user that loses all trust edges pointing to its identity will immediately try to get new trust edges by solving more captchas.

As can be seen in Figure 5.1, the PKI-based method proposed in this thesis does only fare slightly better than no spam protection at all. The shallow but not scalable version of the PKI-based method fares better, but the web-of-trust trustlist-based method is still by far the most effective in blocking spammers. The reason why the shallow version of the PKI-based method fares better is solely because it is impossible for spammer collectives to form. The identity which posts spam is also always the identity that should be and can be isolated from the network by having the trust edges to it removed. No possibly incorrect guessing or estimation needs to be done. A more detailed analysis that justifies the actual numbers is presented in the conclusions.

To see why the effectiveness is not better for the new method and its shallow counterpart, more detailed numbers were gathered. These are reproduced in Table 5.2. Here one can see that the primary reason why the proposed PKI-based method is weak is because honest users must solve many captchas. As mentioned earlier, in the PKI-based method it is not enough that users solve a captcha to

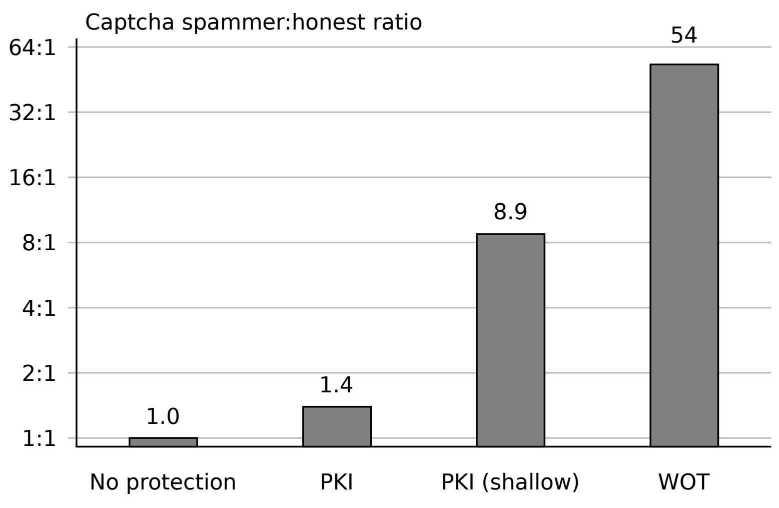

Figure 5.1: The effectiveness of the simulated spam protection methods in a small network. The effectiveness is measured in how many more captchas spammers must solve than honest users to stay unbanned at all time. A higher value means better at blocking spammers. A value of 1.0 means totally ineffective. The scale is logarithmic.

become initially visible, or become visible again after having been banned. Users must also solve a captcha for each identity to ban. Even with the absense of spammer collectives, as in the shallow version, the fact that honest users must solve captchas to ban other identities, spammers, keeps the effectiveness of the spam protection method down.

The web-of-trust trustlist-based method is also susceptible to spammer collectives, yet honest users do not even have to solve two captchas on average there. In the web-of-trust trustlist-based method anyone can give a negative score to another identity, without solving any captcha. The web-of-trust trustlist-based method also has another advantage that may be contributing to why it works so well. Positive scores give useful hints about who is not part of a spammer collective, making banning of spammer collectives more accurate.

Finally it can be noted that the fact that honest users must solve many more captchas in one spam protection method than in another does not mean that the user must do more work. Captchas can be made easier or harder to compensate for that. For example there can be fewer distorted character per captcha to make them easier. The only thing that really matters is how many more times of work spammers must do, and that is the spammer:honest captcha ratio shown in the figures.

Symbol	**Parameter**	**Default Value**
n_p	Number of pretrusted identities	3
n_h	Number of other honest users	100
γ, n_s	There are $n_s = \gamma n_h$ spammers	$\gamma = 10\%$
α	Percentage of honest users using mark features, marking files good or bad	50%
v	How many hours a certificate chain stays valid	12 hours
u	Average number of times each month an honest user goes online and performs searches	15 times/month
D	Number of days simulated	90 days

Table 5.1: Adjustable parameters used in the simulator, and their default values. Each honest user has exactly one identity at any given time. Each spammer may have several identities forming a spammer collective.

	No prot.	PKI	PKI (shallow)	WOT
Solved per honest	0.0	20.1	6.7	1.6
Solved per spammer	0.0	28.6	59.6	84.7
Spammer:honest ratio	1.0	1.4	8.9	54.2

Table 5.2: How many captchas each honest user (including pretrusted ones) must solve on average, how many captchas each spammer must solve on average, and the ratio between those two.

5.2 Larger Network Sizes

The simulation runs are computationally heavy and thus slow. Especially, the web-of-trust trustlist-based method has as implemented in the simulator a total time complexity of $O(n^3)$, where n is the number of simulated peers. Due to this, simulating a network with a large number of identities is impossible. Instead, to examine whether the presented method continues to provide spam protection for larger network sizes, simulations are run for different small choices of n_h. From this it is possible to see how the effectiveness changes with increased network size.

To get the results the simulator had to be modified slightly. Since n_h is small, the average certificate chain length varies quite much as n_h is increased. Eventually, the average certificate chain length would settle on a value, but not for the modest number of peers simulated here. The average certificate chain length affects the results due to how it interacts with spammer collectives. To avoid measuring this, spammers were made to aim for a predefined certificate chain length

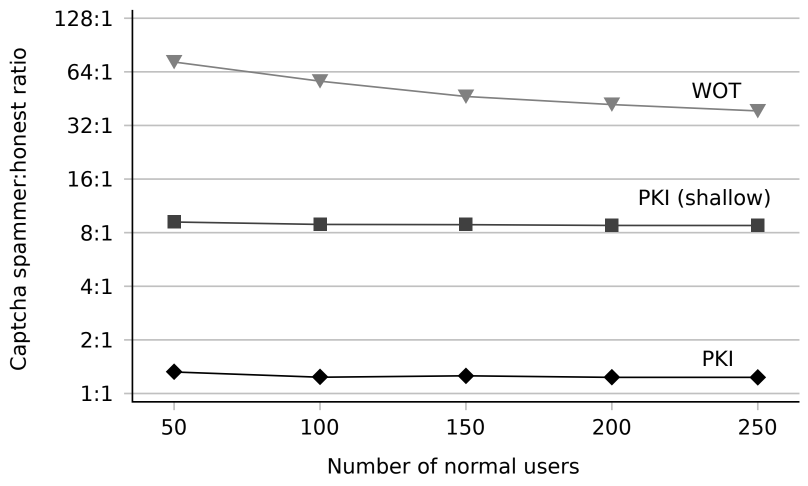

Figure 5.2: How the effectiveness changes when the network size increases. The network is the same as in Table 5.1, except the number of honest users n_h is increased. The number of spammers is also increased proportionally by keeping $\gamma = 10\%$ constant. As can be seen, the effectiveness of the new PKI-based spam protection method does not decrease as the network increases in size. The scale is logarithmic.

when solving captchas to get trusted. The results are shown in Figure 5.2.

A spam protection method must not only run efficiently on large network sizes to be called scalable, it must also maintain its effectiveness of blocking spammers. Due to the design of the PKI-based spam protection method, it should maintain its effectiveness even for very large networks. Figure 5.2 confirms that it does. As is clear from the figure, there is not even a hint of a drop in effectiveness for the PKI-based method and its shallow counterpart. This despite more than doubling the network size. Note that the slightly higher values for the data point $n_h = 50$ do not mean there is a difference in effectiveness. That network size is so small it is approaching the limit of what can give meaningful results from a simulation. There are only 5 spammers in that case, for example. That data point can not be relied on.

One interesting thing to note is that the effectiveness of the web-of-trust trustlist-based method appears to drop quite radically as the network grows. At least according the figure. One possible cause may be the complex interaction between spammer collectives and the successively increased diversity of interests. The increased diversity of interests may benefit the spammers. Another possibility is that there are no problems with the web-of-trust trustlist-based method at all. It may only be the simulator that is overestimating the effectiveness for small network sizes, and that the curve does flatten out eventually. One cannot tell from the figure whether it will flatten out, or just decreases at a successively

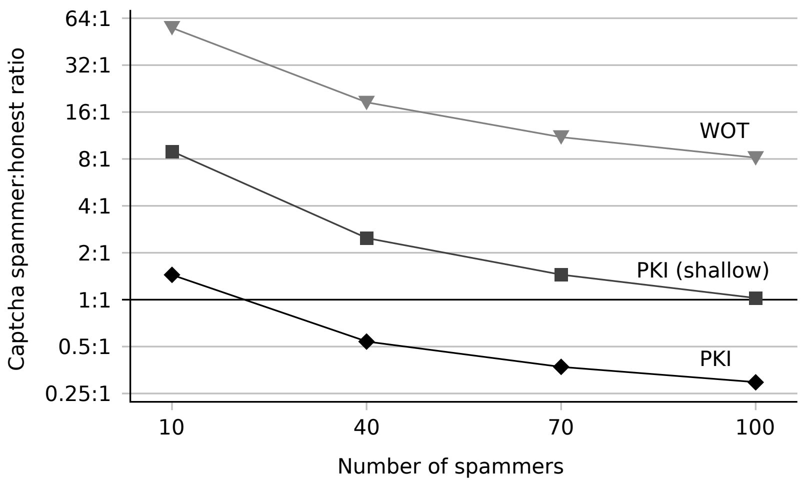

Figure 5.3: How the effectiveness changes when the fraction of spammers increases. The network is the same as in Table 5.1, except the γ is changed as in the x-axis. Since the network has 100 honest users, the x-axis also shows the actual number of spammers. The highlighted 1.0 ratio reflects the effectiveness without spam protection. Anything below that is worse than no spam protection at all. The scale is logarithmic.

lower rate. Either way, this is not investigated further. It was the effectiveness the web-of-trust method has for small networks like the ones simulated that made the method interesting to use as a comparison. The method has never been used on large networks after all, since it does not scale.

5.3 Larger Fraction of Spammers

Another interesting thing is how well the spam protection methods keep protecting against spam as the number of spammers relative to the number of honest identities increases. If a spam protection method can be shown to remain effective when facing a large fraction of spammers, it will handle the fraction of spammers that occur in practice. The result is shown in Figure 5.3.

All three simulated spam protection methods decrease in effectiveness with about the same factor as the number of spammers is increased. The new PKI-based method does not have any disadvantage in that regard when compared to the web-of-trust trustlist-based method. However, due to the modest effectiveness it stops providing spam protection at about $\gamma = 20\%$.

The reason why all spam protection methods decrease in effectiveness is due to there being more spammers to ban. The number of honest users have not increased, therefore roughly the same number of bans are done. Thus, each spammer will

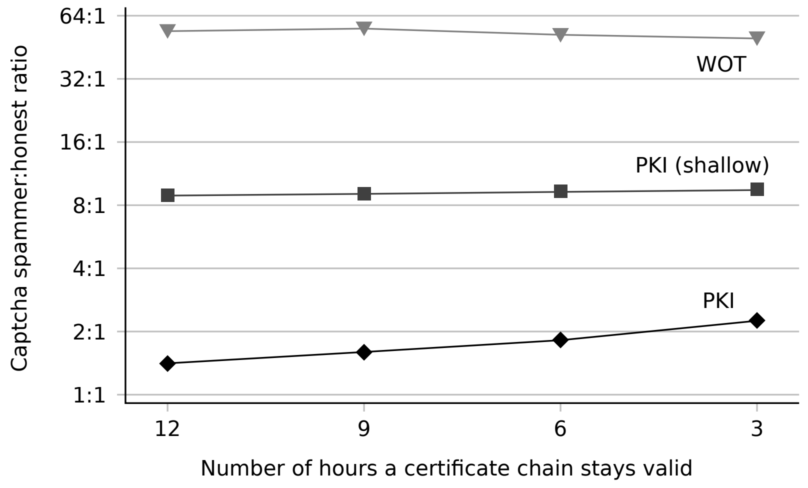

Figure 5.4: How the effectiveness improves when the certificate chains are made to expire faster, having a shorter initial validity time. The network is the same as in Table 5.1, except v is successively lowered as in the x-axis. The web-of-trust trustlist-based method is unaffected by this change, the variations there are noise. The scale is logarithmic.

be banned fewer times. The decrease for the PKI-based method in particular is discussed in more detail in Chapter 6, where the actual numbers are validated.

The figure suggests that the PKI-based method would start having the opposite effect of blocking spammers after about $\gamma = 20\%$. However, this would likely never occur in practice, as users would stop using the spam protection functionality at that point.

5.4 Shorter Validity Times

One problem with the PKI-based method is that certificate chains stay valid until they expire. This because no revocation checks can be done. If the validity time of a certificate chain is as long as 12 hours, a spammer needs to solve a new captcha at most every 12th hour. This even if the spammer gets banned within the first hour. The PKI-based method should perform better when the validity times are shorter. Figure 5.4 shows that it does, and by how much.

One thing that is holding back the improvement despite reduced validity times are the fact that honest users must solve captchas to ban spammers. Spammers do indeed have to solve more captchas to stay unbanned at all time when the validity time is reduced. But each time a spammer solves a captcha to get unbanned, an honest user may counter it to get the spammer banned again. This means spammers do not necessarily get punished more just due to a shorter validity time.

The advantage honest users have may stay about the same. Reduced validity times can however help in the presence of spammer collectives, as shown in the figure. The real PKI-based method does see a more significant improvement than its shallow counterpart.

Note that, although shorter validity times improve the effectiveness of the PKI-based method, it is not a drastic improvement. Shorter validity times also means that all peers must renew their certificate chains and republish their file handles much more often than they otherwise would need to. Depending on the application, it may not be worth the trade off.

5.5 Effect of Approximations

The results produced by the simulator of course suffer from some errors. First off, the variance in the results are quite large from one simulation run to another, using the same parameters. This is due to values for peer behavior and similar being drawn from various random distributions. To reduce this variance and get a better estimation of the real result, 100 simulation runs are performed for the same parameters. The average of the results from those 100 simulation runs are used in what is presented in the figures and tables in this chapter. 100 simulation runs are enough to reduce this error to be much less than the other errors.

Another error is of course that the actual simulator itself just approximates a real DHT network, real peer behavior, real spammer behavior and so on. To get a rough understanding of how big this error is, test simulations were performed with internal model parameters changed slightly. Any such change of unknown direction and amplitude should make the simulator reflect reality more accurately. If the results are affected much by such a change, this also means that those errors that exist as part of the approximation may influence the results a lot.

By this, the errors were found to possibly be up to about a factor 2. This is quite much, but still small enough to be able to reason about the relative effectiveness of the various simulated spam protection methods. However, it may prompt doubt about whether the PKI-based method actually is above the 1:1 line in Figure 5.1 or not. If it were not, the PKI-based spam protection method would not be effective. In the conclusions chapter it is shown that the PKI-based method is effective and the results presented are more accurate than what the errors calculated here suggests, at least for the PKI-based method and its shallow counterpart.

For the web-of-trust trustlist-based method there turned out to be another large error though. That error is due to the fact that so-called network churn is not simulated. Network churn is when new peers are joining and old peers leaving the network over time. This is different from peers merely going offline for a while, just to go online later again. New peers joining the network do not have any identities, and must therefore always solve at least one captcha to get an identity trusted.

In the simulator as designed, initial captchas for all simulated identities are solved during the first few virtual days. After that, captchas are only solved to ban or to get trusted again after being banned. When the number of simulated days

D approaches infinity, those initial captchas get less relevant. As can be seen in Table 5.2, the initial captchas are very relevant for the web-of-trust trustlist-based method, but not for the other two. The value of the parameter D does affect the influence of the initial captchas in the results, much like what would had happened if churn was simulated. But one cannot reason about what value of D would be right. Based on experimentation with various choices of D, it is estimated that there may be up to an additional factor 2 error for the web-of-trust trustlist-based method due to this. This means the total error for the web-of-trust trustlist-based method may be up to a factor 4, in worst-case.

Chapter 6
Conclusions and Discussion

In this master thesis work a new spam protection method based on the scalability of the PKI architecture has been designed. It has through simulations been confirmed to provide spam protection. Although the spam protection it provides is modest compared to the state-of-the-art among non-scalable methods, this new spam protection method is entirely scalable and has a flexible design that allows for many future improvements. For very large fully decentralized peer-to-peer networks seeking a scalable solution to spam, this new spam protection method may be an attractive choice both as it is, and to improve upon. The scalability properties are guaranteed, and it is a complete spam protection method, not just a reputation system.

More importantly, the work in this master thesis contributes to the understanding of how to design scalable spam protection methods for decentralized peer-to-peer networks, and opens up for further research and improvements in this area.

6.1 Why the PKI-Based Method Works

To understand why the proposed PKI-based method works, one can consider the simplified case. The simplified case is that spammers and honest users are assumed to be equal with regard to uptime pattern and activity, and that diversity of interests is not considered. Furthermore, only the steady state is considered. The steady state is when infinitely many days have been simulated, and the initial captchas thus no longer have any relevance.

Let us consider the shallow version of the PKI-based method first. Here spammer collectives cannot form. If an honest user is banning a spammer in response to spam, it must solve a captcha. In response to this, the spammer must also solve a captcha to become trusted again, possibly with a new fresh identity. If there would have been equally many honest users and spammers, this means that they would both solve the same amount of captchas and the spam protection method would thus be ineffective. But this is typically not the case. In the default network simulated, referring back to Table 5.1, there are 10 times as many honest users as spammers. This means that a spammer will get banned 10 times on average for each time an honest user bans someone. Therefore, the spammer would need to solve 10 times as many captchas as each honest user on average. In the simulation results in Figure 5.1 the captcha ratio is 8.9, which is close to the 10.0 from this simplified case.

For the real and scalable version of the PKI-based method however, things are more complicated. Now spammer collectives can form. Since the depth of the trust graph is limited to 5 for this method, there are exactly 4 possible options for who to ban, see Figure 2.4. Only one out of those are correct to ban, namely I_1 in that figure. It is assumed there are always 4 options, meaning the identity found spamming always is as far out in the trust graph as possible. If an honest user that is going to ban a spammer guesses right, both the honest user and the spammer solves one captcha each. If however the honest user guesses wrong, there are two cases. In case one the honest user guesses that another honest user is the spammer. Then both the user who is banning and the banned user must solve one captcha each. In case two the honest user guesses on a spammer-controlled identity in the middle of the spammer collective. In this case the user banning must solve a captcha, and no one else.

In the simulator, honest users trying to ban spammers are doing uniformly distributed guesses. Therefore, things are as follows. If everyone from the identity who was found spamming to the identity trusted by a pretrusted identity are part of a spammer collective, honest users are solving 4 captchas on average for each captcha a spammer solves. This since they are doing three guesses in the middle of a spammer collective for every correct guess on average. If, on the contrary, the identity found spamming is not part of a spammer collective and therefore the real spammer, honest users must solve 7 captchas on average for each captcha a spammer solves. This time the 3 incorrect guesses are on honest users. All this is because the honest users cannot know whether the spamming identities are part of a spammer collective or not, nor how long the spammer collective is. This means that for a network with 10 times as many honest users than spammers, the captcha ratio should end up being between 1.43 and 2.5 in this simplified case. This also matches well the the results from Figure 5.1, where the captcha ratio is 1.4.

From this reasoning, one more thing can be seen. In the extreme case where there are no spammers, honest users do not need to solve any captchas, except for the initial one. The fewer spammers there are, the harder spammers are blocked. Likewise, the PKI-based method also reaches a point where there are so many spammers that the method is no longer effective, or may even be worse than no spam protection at all. From the reasoning above that would for the real PKI-based method occur between $\gamma = 14.3\%$ and $\gamma = 25\%$ in the simplified case. For the shallow version it would occur at $\gamma = 100\%$. Both these match well with the simulation results in Figure 5.3.

6.2 Comparison of the Simulated Methods

The new PKI-based spam protection method proposed in this thesis has been shown to scale just as well as the underlying DHT network with regard to the time complexity. This is a huge improvement over any web-of-trust trustlist-based method. But it is also scalable in all other regards. Both the moderation burden for peers and the effectiveness in blocking spammers remain the same regardless of

	No prot.	PKI	PKI (shallow)	WOT
Time Complexity	$O(\log n)$	$O(\log n)$	$O(n)$	$O(n \log n)$
Storage Complexity	$O(1)$	$O(1)$	$O(n)$	$O(n)$
Moderation Burden	None	$O(1)$	$O(1)$	$O(1)$
Effectiveness	1.0	1.4	8.9	54 *

Table 6.1: Summary and comparison of all simulated spam protection methods. The time complexities and moderation burden assumes each identity only naturally trusts or scores a small and fixed number of other identities. The effectiveness is taken from Figure 5.1, higher means better at blocking spammers.
*) As shown in Figure 5.2, the effectiveness for the web-of-trust trustlist-based method may be decreasing when the network grows.

the number of participating peers in the network. The design of the method should also inspire confidence that the method is secure, and cannot be compromized by clever spammers. It is based on the PKI architecture, and uses a manual proof-of-work scheme to guard against attacks. A summary of all spam protection methods simulated as part of this thesis work is presented in Table 6.1.

Only two downsides with the new PKI-based method compared to a web-of-trust trustlist-based method can be identified, besides the modest effectiveness. The first is that all published handles, such as file handles or forum posts, must be republished frequently. The second is that the pretrusted identities may not all go offline at the same time. Both of these are due to the short validity time of certificate chains. The second downside is rarely a problem in practice, as the users who dedicate themselves to be pretrusted often can spare leaving a computer running all the time. The first downside may be a problem, however, for less popular web content published by users who are offline most of the time. However, it is not a large problem. A published handle must just be signed with any valid certificate chain, it must not necessarily be a signature from the initial publisher. A forum post can, for example, both be signed by the author of the post to prove authorship, and have a certificate chain signing it from anyone trusted to make the post visible. Anyone may republish any forum posts they see and trusts not to be spam, knowing that they will be punished instead if those posts contains spam. This means most web content will remain available even if the initial publisher or author goes offline permanently.

6.3 Discovered Issues

Since the scalability is good now, it is no longer an area where improvements are needed. The next logical step would be to make such scalable methods more effective, step by step, without losing the scalability properties. The three biggest contributions to the modest effectiveness of the proposed PKI-based method are summarized below.

The biggest issue is that honest users must solve a captcha to ban a spammer. This remains the biggest issue regardless of the presence of spammer collectives or how the actual revocations are made. Let us take a look at Table 5.2. If honest users could ban spammers without solving captchas, the PKI-based method would require honest users to solve about the same number of captchas on average as the web-of-trust trustlist-based method does. That means about 1.6 captchas on average instead of 20.1 on average, which would be a factor 12.5 improvement in effectiveness. What all this means is that, the manual proof-of-work system chosen as part of the design of the PKI-based method is no good. It was chosen because it solves the trust revocation problem mentioned in Chapter 2 in a secure way, which is easy to analyze. A more sophisticated system may work significantly better, up to a factor 12.5 better. See the future work section for a few examples.

The second issue is that guesses must be made when trying to ban a spammer collective. Comparing the PKI-based method with its shallow counterpart, the difference in effectiveness is almost entirely due to spammer collectives. The only other contributing factor is that the pretrusted identities are the trusters in the shallow version, and always online. This means that if it was as easy to ban a spammer collective as a single spammer, the spam protection method may be up to a factor 6 better.

The third issue is that the certificate chains remain valid for a certain time, without possibility of being revoked. According to Figure 5.4, even if the validity time is lowered a lot the effectiveness is improved by less than a factor 2. All this while suffering from the increased load, as discussed in the shorter validity times section in Chapter 5. But as also mentioned in that section, if honest users do not have to solve captchas to ban spammers, things may look different here.

6.4 Future Work

The web-of-trust trustlist-based method simulated in this thesis work only uses scores from the immediate trustees to score other identities, essentially limiting score calculations to depth 2. Yet, it outperformed the new PKI-based method and its shallow counterpart in effectiveness in blocking spammers. It may be an interesting research question to see whether trustlists can be used for score calculations like this, but using the new PKI-based method for discovering identities further away than depth 2. The methods can maybe be combined in this way, eliminating the need to solve captchas for banning. That could result in an effective and scalable solution. The number of trustlists that would need to be fetched would in that case be very low and $O(1)$. However, as Figure 5.2 shows a worrying decrease in effectiveness for the web-of-trust trustlist-based method, it must be researched whether this decrease is real and what the effectiveness would end up being for a large network. That problem, if real, will likely affect a method combined like this too.

Otherwise, a solution for increasing the effectiveness by reducing the number of captchas needed to ban others may be a good direction for future research. Introducing a notion of being more trusted, such that more trusted identities can ban less trusted ones may be one possibility. However, care must be taken so that a more trusted identity that starts banning good identities stops being trusted. Another possibility would be to attempt to identify more exact probabilities of what identities are part of a spammer collective, such that the number of incorrect bans can be lowered.

Finally, it would of course be interesting to see the concept of the new PKI-based method be extended to more interesting and diverse use-cases than file-sharing networks. Decentralizing the web and the social platforms people use today are probably the most constructive, to meet the demands of the future.

References

[1] A. Back, *Hashcash - A Denial of Service Counter-Measure*, 2002, `http://hashcash.org/papers/hashcash.pdf`, last fetched 2016-05-19

[2] J. Benet, *IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)*, 2015, `https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf`, last fetched 2016-05-19

[3] B. Cohen, *The BitTorrent Protocol Specification*, 2008, `http://www.bittorrent.org/beps/bep_0003.html`, last fetched 2016-05-19

[4] P. Dewan, P. Dasgupta, *P2P Reputation Management Using Distributed Identities and Decentralized Recommendation Chains*, 2009, IEEE Transactions on Knowledge and Data Engineering, Volume 22 Issue 7, July 2010, pp. 1000-1013, doi:10.1109/TKDE.2009.45

[5] S. D. Kamvar, M. T. Schlosser, H. Garcia-Molina, *The EigenTrust Algorithm for Reputation Management in P2P Networks*, 2003, `http://ilpubs.stanford.edu:8090/562/1/2002-56.pdf`, last fetched 2016-05-19

[6] H. A. Kurdi, *HonestPeer: An enhanced EigenTrust algorithm for reputation management in P2P systems*, 2014, Journal of King Saud University - Computer and Information Sciences, Volume 27 Issue 3, July 2015, pp. 315-322, doi:10.1016/j.jksuci.2014.10.002

[7] P. Maymounkov, D. Mazières, *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*, 2002, `http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf`, last fetched 2016-05-19

[8] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, S. Savage, *Re: CAPTCHAs - Understanding CAPTCHA-Solving Services in an Economic Context*, 2010, `http://cseweb.ucsd.edu/~savage/papers/UsenixSec10.pdf`, last fetched 2016-05-19

[9] S. Saroiu, P. K. Gummadi, S. D. Gribble, *A Measurement Study of Peer-to-Peer File Sharing Systems*, 2002, `https://homes.cs.washington.edu/~gribble/papers/mmcn.pdf`, last fetched 2016-05-19

[10] M. T. Schlosser, T. E. Condie, S. D. Kamvar, *Simulating a File-Sharing P2P Network*, 2003, `http://www-nlp.stanford.edu/pubs/simulator.pdf`, last fetched 2016-05-19

[11] D. Stutzbach, D. Zappala, R. Rejaie, *The Scalability of Swarming Peer-to-Peer Content Delivery*, 2005, http://ix.cs.uoregon.edu/~reza/PUB/networking05.pdf, last fetched 2016-05-19

[12] Wikipedia, the free encyclopedia, *BitTorrent*, https://en.wikipedia.org/wiki/BitTorrent, last fetched 2016-05-19

[13] Wikipedia, the free encyclopedia, *Sybil attack*, https://en.wikipedia.org/wiki/Sybil_attack, last fetched 2016-05-19