Trust*: Extending the Reach of Trust in Distributed Systems

Stephen William Clarke

School of Computer Science

A thesis submitted to the
University of Hertfordshire
in partial fulfilment of the requirements
of the degree of
Doctor of Philosophy

November 2009

Abstract

Building trust is a common requirement in distributed environments especially since many transactions now occur on a person-to-person basis. Examples range from e-commerce on the Internet to peer-to-peer and grid resource sharing. Many solutions to the problem of requiring trust among unknown entities rely on the use of a reputation metric to assess the risk of a potential transaction. However, such reputation systems require (often implicitly) that trust is transitive which can be a problematic assumption.

This dissertation proposes a novel mechanism which we call trust*. The trust* model uses guarantees to extend local trust between unknown end-points. Trust* can be used as a substitution for end-to-end trust. Principals provide guarantees within existing (local) trust relationships to build a chain of localised agreements between the unknown end-points. The guarantees are backed by local micro-payments to provide deterrents and incentives. Trust* relationships can be composed transitively, and the guarantees reduce the risk for the trusting party when doing so. This is because a guarantee is only ever provided locally by a directly trusted principal. Thus, trust management can be reduced to a locally solved problem.

This work aims to develop a new technique for assessing and reducing the risk involved in trusting others in a distributed environment. The thesis of this dissertation is that an electronic analogue of real-world guarantees, is a useful and interesting way to provide these assurances. We develop an extension of the notion of trust, which we call trust*, which is built upon local guarantees, and which provides a novel conceptual framework for analysing and reasoning about a wide variety of trust-related problems in distributed systems.

We present the concept of trust* and apply it to a number of application sce-

narios where it would be beneficial. We simulate the trust* model in these environments for analysis. Also, we describe the key features and other issues related to the trust* model which became evident during its investigation and which are of wider interest.

Acknowledgements

I would like to show my deepest gratitude and appreciation to my extraordinary supervisory team who were willing to support me throughout my PhD studies. The encouragement and inspiration given to me by my principal supervisor Prof Bruce Christianson is second-to-none. He was able to identify my weaknesses early on and gently encourage and support these areas of my personal development. He has always shown a deep interest in my work and was always willing to talk about it. I will always appreciate what he has done for me. My second supervisor Dr Hannan Xiao has also provided me with exceptional support. She has the ability to make me view ideas from a different perspective which helped the overall development of this work. Last, but by no means least, I would like to thank my third supervisor Mr Bob Dickerson. I could throw anything at him and he would instantly have ideas to discuss. Although not a supervisor of mine, I would also like to thank Mr James Malcolm who also took the time to support me and discuss my work on regular occasions.

I would also like to thank my friends and colleagues in the research institute and school of computer science.

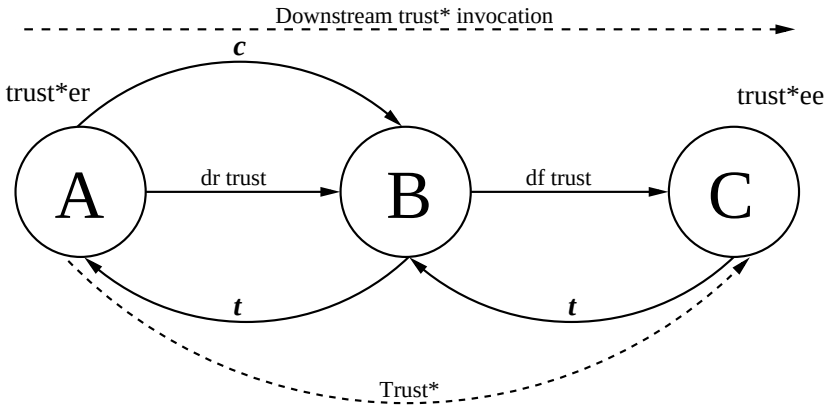
Finally, I would like to show my appreciation to my family and my wife Lisa for their never-ending love and support throughout my studies.

Contents

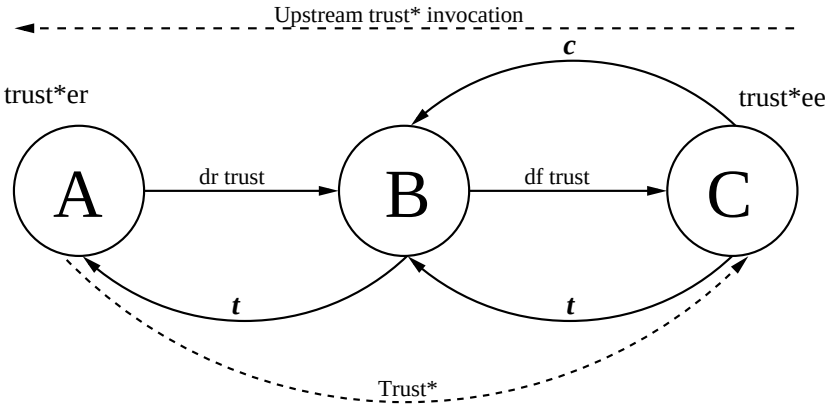
Notation

Symbol	Meaning
df	Direct functional trust.
dr	Direct referral trust.
t	Forfeit payment.
c	Commission payment.

Forward Trust*



Reverse Trust*



This page can be photocopied for easy reference.

Chapter 1

Introduction

This chapter outlines the problem that this work sets out to address and the motivations behind it. This chapter also outlines the thesis advanced by this dissertation and provides a guide to the structure of this dissertation.

1.1 Motivation and Problem Statement

The Internet was once a very static place used primarily for information sharing. Since the revolution of Information Technology [19], services provided over a network have become more diverse and accessible. Many sites and services are now dynamic and allow direct interaction between their users. So, where an e-commerce transaction would once be between a reputable vendor and a customer, over the last decade, person-to-person transactions are becoming more common. This trend is not restricted to services provided via the World Wide Web but includes other distributed environments such as peer-to-peer networks, grids and clouds. Together with the emergence of Web 2.0 and web service architecture such as SOAP and XML, electronic services are now very common. Services provided nowadays are more likely to be a conglomeration of other services from third-party providers including members of the general public. Anyone can now easily set up a shop or service from their home. All of this has led to a problem of trust. It is now extremely likely that when someone transacts with another over a network, they will be completely unknown to each other.

Before the Internet boom when fewer people owned a personal computer and only the larger companies hosted websites, trust was built on the reputation of the company in question. This is still how trust is gained by companies in the real-world. To address the problem of assessing trustworthiness on the Internet today, the most ubiquitous method is to use a reputation system [60, 63, 86, 91]. These are loosely based on the way that trust is built in the real-world by referral from another (trusted) person.

Each of a reputation system's users has a reputation rating which is normally calculated from feedback from previous transactions with others. These ratings can be viewed by prospective users intending on dealing with the principal in question. The outcome of each transaction will affect the reputation score accordingly. Such reputation systems are widely used on the Internet for various purposes and generally work well. However, reputation systems assume (often implicitly) that trust is transitive [61] which can be a false assumption [26, 49]. Assume a user wants to determine the risk involved if they were to trust another (e.g. to provide a described service) by looking at their reputation rating, which contains comments and ratings left from previous transactions. It is unlikely that the user looking knows (or trusts) the other users who have left the comments. But even if they *do* know and trust the people who left the feedback, they will still be transitively trusting the service provider in question.

To give a real-world example, assume that Alice needs to have her car serviced. She trusts Bob's advice who in turn trusts Carol to service his car. Alice is indirectly trusting Carol to be a good mechanic as she trusts Bob's advice. Suppose that Carol isn't a mechanic herself however she trusts David who is. The question is to what purpose is Alice trusting Bob. In the first case, Alice trusts Bob to recommend a mechanic whom he trusts directly. The second case is Alice trusting Bob to trust someone else's (Carol's) recommendation of a good mechanic. This example assumes that Alice trusted Bob in the first place, however in real-world reputation systems, it is unlikely that the person reading reputation ratings or recommendations even knows the person leaving the comments. Also, just because David might do a good job for Carol, it doesn't necessarily mean that David will do a good job for Alice or Bob. It might be that David is really a cowboy mechanic (possibly not even to Carol's knowledge) but will always provide a good service

to Carol because he is sweet on her. A more concrete example is with eBay's reputation system where a particular user might have a very good reputation for selling films. However, this won't necessarily hold when buying books or music from them. This lack of "scope" also makes basing trust decisions on reputation systems risky.

As most existing trust relationships are being transitively derived through intermediaries, a way to reduce the risk and lower the hassle of compliance for all parties involved is required.

1.2 Real-World Guarantees

In real-world protocols, the ability for unknown parties to act as if they trust one another is often facilitated by using an intermediary guarantor as a replacement for transitivity of trust. Guarantees work by shifting the risk to another party and thus lowering the risk for the trusting party. An example of this is letting houses to students, where landlords might require a guarantee against a particular tenant. The guarantor might be a parent who trusts that their son or daughter will pay the rent. The landlord trusts the guarantor so the landlord has shifted the risk of not receiving the rent to the guarantor. The landlord believes that he will always get his rent whether it be from the tenant or the guarantor. The guarantor being a parent is likely to pay the rent as they have a reputation to lose, whereas perhaps the student might not.

This type of agreement is made on a regular basis in the real-world. For example, buying electrical goods such as a kettle. A customer is more likely to buy a kettle if they know that they'll receive a refund or replacement if it breaks within a year. The risk involved for the customer has been removed and shifted towards the guarantee provider (the manufacturer). It is now the incentive of the manufacturer or shop to control the quality of their products in order to avoid paying for replacements.

The electronic equivalents of real-world guarantees are used in the trust* model to lower the perceived risk involved for a trusting principal in the same way as the examples given above. The only difference being that trust* will be used to guarantee electronic services rather than physical products or services.

1.3 Trust*

Trust* builds on pre-existing trust relationships between principals who are known to one another and is based on the electronic equivalent of the real-world guarantee solution. Say that Alice needs to trust Carol about something and doesn't personally know or trust Carol. However, Alice trusts Bob who in turn trusts Carol to do whatever it is Alice needs her to do. In order to change Alice's perception of the risk involved, Bob could guarantee to Alice that Carol will act as intended and offer Alice compensation if Carol doesn't. Assume for now that Bob gets paid a commission by Carol as an incentive to act as a guarantor[1].

The concept of "extending" trust in this way by using localised guarantees is what we call a trust* relationship. Figure 1.1 shows this typical trust* relationship. The trust*er (Alice) can act *as if* she trusts the trust*ee (Carol) directly. In order to shift the risk, forfeit payments are used as a deterrent (to the trust*ee) or compensation (to the trust*er) but assume for now that they are micro-payments. All forfeits are paid locally; if Carol defaults then Bob must pay Alice the agreed forfeit whether or not Carol pays Bob any forfeit she owes him (and the two forfeits may be of different amounts). Also, Carol might not have made a guarantee to Bob that she will reimburse the forfeit. Thus, failure to provide a service — or to pay a forfeit — is likely to result in an update to a *local* trust relationship.

Trust* can be composed to an arbitrary number of hops because all trust is now local and so are the forfeits. It is worth noting that trust isn't the same as trust* even in a one hop scenario. If Bob trust*s Carol to provide a service, it means that Bob trusts Carol to either provide the service or else pay the forfeit[2].

The whole concept of extending trust to trust* makes use of already existing trust relationships rather than creating new ones. It uses guarantees to bridge the gap between unknown principals with a sequence of localised agreements which remove or reduce the perceived risk of the trust*ing principal (when transitively trusting) and shift it towards the principal being trust*ed. Although trust* doesn't assume that referral trust is transitive (so there is no end-to-end trust), Alice can

[1] Incentive and deterrent payments are discussed later in Section 3.3.

[2] It may be that Bob would rather have the money, and believes that Carol cannot provide the service, but will always pay the forfeit.

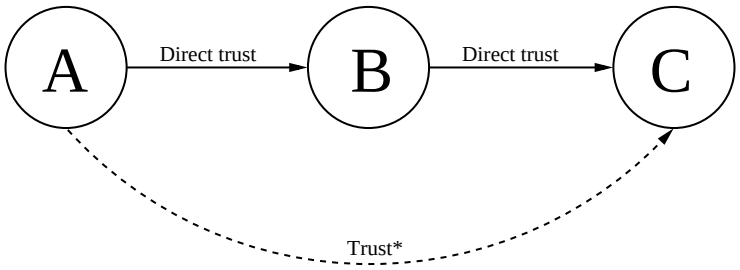


Figure 1.1: A two-hop trust* relationship.

behave *as if* she trusts Carol.

Trust* is flexible in that it can be used in many different applications, however because it builds upon already existing trust, it won't need to replace any existing trust infrastructures. It will integrate with them to manage direct trust relationships and can utilise existing commodities such as reputation or resources. More refinements of the trust* model are introduced throughout the rest of this dissertation by applying it to various application scenarios.

1.4 Structure of this Dissertation

This section provides an overview of the subsequent chapters of this dissertation.

Chapter 2 gives an overview of the related work which provides a background to this work. This primarily reviews research in the field of trust. More specifically, perceptions of trust, types of trust and how it can be modelled and established. A review of how trust can be built on the Internet and for e-commerce is given followed by examples of certification mechanisms. Background work specific to a particular application is reviewed at the start of the relevant application chapter.

Chapter 3 gives an overview of the problem of trust in peer-to-peer (P2P) networks and demonstrates how trust* could be applied to such a scenario. The example is based on an existing P2P client called Turtle, which we extend by adding trust* capabilities. This chapter introduces the idea of payments by resource.

Chapter 4 describes in detail how the trust* model has been simulated. The focus in this chapter is on the design considerations of simulating the application scenario described in Chapter 3. This chapter also describes the simulation testing process and summarises the results for the P2P simulation. However, the simulation model has been designed with the flexibility to allow changes to easily adapt the simulation to the other applications explored in this dissertation and these simulations are reported in later chapters.

Chapter 5 gives an overview of trust in computational grids and explains how trust* could be used to extend trust between unknown users in a distributed system where resources are normally shared. This chapter also discusses how mechanisms originally designed for the Globe distributed middleware could be adapted to work with the trust* model. Implementation of trust* routing is addressed by constructing various analogues to network routing algorithms. This chapter also describes how resource brokering could occur when using trust* in a grid setting. Finally, the changes made to adapt the simulation to a grid application scenario are described.

Chapter 6 describes how trust* can be used to minimise the hassle of compliance in a licensing agreement. We show how trust* could be applied to a software End User License Agreement in order to ensure the legitimacy of the software before installation. Also, the application of trust* to digital music downloads is described. Finally, trust* is applied to situations where a sponsorship or donation undertaking has been made (e.g. by clicking on a link) to ensure that the intended recipient will get their payment. This chapter extends the commission and forfeit models from previous chapters and gives an overview of the types of micro-payment that could be used in these applications. Also, the simulation implications for these applications are presented.

Chapter 7 is the final application chapter in this dissertation. This chapter applies trust* to the sending of emails to guarantee that they are not junk (or "spam") email. This application demonstrates a scenario where trust* paths are built in the opposite direction (i.e. from the trust*ee end) to that of pre-

vious chapters. This chapter also discusses pricing strategies and how to exploit analogues between network congestion control and the commission model of trust*. Finally, the implications of simulating the spam-proof application are presented.

Chapter 8 provides a full description of the trust* model and reviews and integrates the concepts that have been introduced so far. We introduce a notation to formalise the main components of the model. Finally we recapitulate the key features and other issues of wider interest raised by the model with further discussion and examples.

Chapter 9 concludes this dissertation and reviews the contributions to knowledge that this work has made. We discuss some possible modifications that could be made to the trust* model and the corresponding future directions of research. Also, further applications are suggested to which applying trust* could be beneficial.

Chapter 2

Background and Related Work

2.1 Introduction

This chapter provides a background to the work presented in this dissertation by discussing some related work in the field of trust. More precisely, how trust is perceived in cyberspace and applications for which it is required. This chapter gives an overview of these factors including a review of some of the work that addresses trust related issues. It also reviews some of the current solutions for building trust in cyberspace which this work aims to improve, and highlights the difficulties.

Throughout the rest of this dissertation, a number of different applications of our approach are described. Background work which is specific to a particular application is reviewed at the start of the appropriate chapter. In each of these chapters, a short background is given to set the scene for the specific application of trust*.

2.2 Perceptions of Trust

The theory of trust was first viewed as a computational concept by Marsh in his thesis [73] and later by Harbison [49]. Trust is very complex and dynamic [20] and is typically a subjective measure of someone's belief that another will act as intended which is also dependent on the task at hand. Trust is generally used as a

substitute for knowledge. Jøsang [59] shows how complex trust is and focuses on understanding trust in the real-world. Also, a differentiation is made in his work between trusting human agents and trusting systems and he shows that the need to trust is only required if malicious behaviour exists in the first place.

Camp *et al* [18] identify the variations in how trust is perceived from a technological, philosophical and social theory perspective. The authors develop hypotheses as to why these views conflict with each other; technologists often assume that humans are attentive, discerning, and ever-learning; philosophers argue that humans are simplifiers and that they will often trust machines to aid this; social theorists argue that humans slowly lower barriers against trust, rather than refine them. A conclusion drawn by the authors from these hypotheses is that designing security mechanisms for trust should be based on the philosophical and social theories of trust. This is because it is not possible to design a computer security system without making assumptions about human behaviour.

A study by Kindberg *et al* [67] shows how trust perceptions react when using different methods of paying for a meal in a restaurant. These methods all require payment from an electronic wallet (a device which can interact with another device to make a payment) but range from docked to wireless connections. Also, whether a waiter plays a part (i.e. holds the device) or whether any bar-code scanning has taken place is considered. The study shows that people reason about trust judgements in many different ways. For example, where security issues may play a role, other issues such as convenience may be of more importance. For others, it might be a social issue, however, many were aware of the potential security issues when prompted. A trade-off between these issues needs to be identified when designing a system that requires trust.

2.3 Analysis and Modelling

Much of the work related to the area of trust has been to do with developing ways to analyse and model trust. One way of modelling trust relationships is to use the Trust Network Analysis with Subjective Logic (TNA-SL) notation proposed by Jøsang *et al* [62, 65]. TNA-SL requires trust *relationships* to be expressed as a series of beliefs. An example belief opinion $\omega_x^A = (b, d, u, a)$ expresses the relying

party A's belief in the truth of statement x. This statement might be "Party X is reliable regarding σ" where σ is the scope of the trust statement. The values of b, d and u represent belief, disbelief and uncertainty respectively where $b, d, u \in [0, 1]$ and $b + d + u = 1$. The parameter $a \in [0, 1]$ is the base rate and determines the *a priori* trust that is existent in the principal in question. Various operations allow trust and risk to be reasoned with using these measures. TNA-SL also requires that trust *networks* are expressed as directed graphs in order to represent trust relationships. Jøsang doesn't assume that trust is transitive but transitive trust relationships can be expressed in this notation when certain conditions are present. Table 2.1 gives an overview of this notation.

Symbol	Meaning
$A, B, C, \ldots$	Alice, Bob, Carol, etc.
:	Connection of trust arcs.
σ	Trust scope.
f	Functional trust variant.
r	Referral trust variant.
d	Direct trust.
i	Indirect trust.
$\diamond$	Alternative trust path.

Table 2.1: Transitive trust notation.

An example of a transitive trust relationship between Alice and David can be expressed as:

$$([A, D, if\sigma]) = ([A, B, dr\sigma] : [B, C, dr\sigma] : [C, D, df\sigma]) \tag{2.1}$$

This notation gives us details of not only the trust path between Alice and David but also details such as the type of trust and its scope. The types of trust are distinguished into *functional* and *referral* trust variants. Functional trust is used where a principal trusts that another principal is capable of performing the task in question. Referral trust is used where a principal trusts the recommendation of another principal (perhaps a recommendation of a principal whom *they* have functional trust in). Variations of these types of trust can be separated into *direct* trust where the trust relationship is local and *indirect* trust where the trust relationship is derived transitively. For example, Equation 2.1 shows that Alice has *indirect*

functional trust in David. This is due to Alice having *direct referral* trust in Bob, who also has *direct referral* trust in Carol. Finally, Carol has *direct functional* trust in David.

According to Jøsang, a transitive trust relationship will only be valid if the combination of referral and functional trust reflects that of the example given above. This is that a chain can consist of any number of direct referral trust links followed by a single direct functional trust link. It is only then that Alice will have indirect functional trust in David.

The scope to which the trust relationship applies also plays an important role as it is likely to be different between each pair of principals in a chain. The scope of the transitive trust relationship ($[A, D, if\sigma]$) is the common subset of all scopes in the chain ($[A, B, dr\sigma] : [B, C, dr\sigma] : [C, D, df\sigma]$). Say for example that Alice needs new tyres fitted on her car and trusts Bob to refer her. Bob knows that Carol knows someone who can do this. Bob's σ in Carol might just be "I trust Carol to refer me to someone who can fit tyres". However, Carol's σ in David might be "I trust that David can fit *anything* to a car". Even though Carol personally knows that David is a good mechanic who can fit or fix anything regarding a vehicle, the transitive trust scope between Alice and David should only be that he can fit tyres as this is the largest common scope along the chain.

We do believe that trust could be transitive in this way but the trusting principals are still making fragile assumptions and taking large risks[1]. It is desirable to find a way of allowing smaller and more viable assumptions to be made by the trusting principal. Trust* would seem to be a natural extension, where we provide guarantees on top of referral trust (or guaranteed referrals).

Other ways of structuring and defining trust relationships formally include the work by Zhao *et al* [111, 112]. Their definitions allow a number of operations to be performed on modelled trust relationships. For example, how two trust relationships can be combined to build new relationships. Their notation is interesting as it also plays close attention to modelling the "scope" of a trust relationship. An example of a trust relationship in their notation can be expressed as a four-tuple $T =< R, E, C, P >$ where R is a set of *trusters*, E is a set of *trustees*, C is a set

[1]For example, Bob may not know that Carol doesn't fit the tyres herself, and so he may mistakenly assert direct trust in her instead of indirect.

of conditions, and P is a set of properties. The properties include a set of actions that the trustees are trusted to perform and a set of attributes that the trustees are trusted to have. Using this notation, the authors propose many operations that can be performed on existing relationships to form new relationships. For example, Let $T_1 = (R_1, E_1, C_1, P_1)$ and $T_2 = (R_2, E_2, C_2, P_2)$. Then a new relationship $T_3 = (R_1 \cap R_2, E_1 \cap E_2, C_1 \cup C_2, P_1 \cup P_2)$ can be formed.

This section has not attempted to survey every notation used to formalise trust but has summarised the two that are most fitting to the modelling of trust* relationships. Towards the end of this dissertation, chapter 8 uses an extension of the notation of Jøsang *et al* as a basis on which to formally describe the features of the trust* model that are developed in subsequent chapters.

2.4 Transitive Trust and Reputation Systems

Trust can propagate in different ways with the most common way being through transitivity [54, 61, 64, 65, 72]. When a user needs to trust another online, reputation systems are a way to assess the possible risks of trusting that person. A description of the most well known reputation systems (and their models) is given in [60] and [63]. They show the variety of implemented reputation systems that are currently used by websites and although reputation systems work much of the time, they are prone to many problems including unfair ratings, discrimination and ballot stuffing. These problems occur mainly due to the fact that trust is transitively derived in others by using public knowledge (e.g. reputation systems) as opposed to private knowledge (e.g. previous real-world interactions). In a sense, if we had universal access to private knowledge, we wouldn't need trust at all. Also, users might not have enough incentive to leave ratings for others especially if the transaction has already completed.

Reputation systems allow users to rate other users regarding the outcomes of previous transactions or encounters. Others can later view these ratings in order to help them decide whether to trust a principal in future transactions. Many sites employ this method (eBay's "feedback forum" is among the most famous) but such methods are known to have problems such as those mentioned above causing them to give inaccurate and misleading information. This could potentially lead a

principal into a falsely heightened sense of trust with another or to incur a penalty by not undertaking a profitable transaction that was in fact perfectly safe. Other work related to reputation systems and their contribution to building online trust include [36, 91, 106].

The Pretty Good Privacy (PGP) toolkit has its own type of reputation system called the "web of trust" to solve the problem of uniquely identifying public key certificates and who they actually belong to. This was an attempt to decentralise PKI where users give each of the keys in their key-store a rating (or reputation score) depending on their surety that the key actually belongs to the person claiming to own it. If a received key certificate is signed by someone they have already rated (directly or through a chain), they will have an indication of whether the key is likely to belong to that person and whether they can trust the origin of the certificate. Again, a PGP user will be transitively trusting others on the authenticity of a key.

Much research has been conducted to address some of the issues with reputation systems. Examples include TRAVOS [104] which uses probability theory and accounting for previous transactions in order to calculate trust in an agent-based system. It also draws reputation data from third parties and therefore has mechanisms to handle cases where information may be inaccurate or where users might be self-interested.

Dellarocas [34] proposes mechanisms to help reduce discriminatory behaviour and unfair ratings in reputation systems. Discriminatory behaviour could involve a seller (in an e-commerce setting) providing a good service to everyone except for a select few people. As long as this proportion is small, the seller's reputation rating won't be damaged too much. Examples of unfair ratings include ballot stuffing where principals collude to inflate each others reputation. This might involve staging fake transactions to do so. Conversely, principals might collude in bad-mouthing a competitor in order to damage their reputation and effectively drive them out of the market. The proposed solution to these types of behaviour is to use controlled anonymity to hide the identities of the buyers and sellers from each other. In addition to this, a clustering algorithm is used to identify and separate fair ratings and unfair ratings of a principal. The principal's overall reputation will only be calculated from the set of what are considered fair ratings. Also,

reputation systems are prone to threats such as Sybil attacks [38] where the same user can operate under multiple pseudonyms.

An important factor that needs considering when making a trust decision is the amount of risk that is involved (and the losses that might be incurred). In [66], Jøsang and Lo Presti analyse the relationship between risk and trust and demonstrate a way of modelling a principal's risk attitudes when making trust decisions. This shows that people tend to be willing to put different amounts of money at risk depending on the potential gain from a transaction and the probability of its success.

Cvrček and Moody in their work [31] focus on how risk can be assessed by analysing patterns in previous transactions. They argue that risk and trust are orthogonal qualities and show that attacks such as Sybil attacks can be greatly reduced. This works on the assumption that attacking identities can be profiled from their behaviour. By profiling behaviour, threats can be identified and attacks blocked on the basis of similar behaviour traits.

2.5 Local Trust Management

Ways in which trust can be negotiated and established between principals is a well researched area. The term "trust management" was first coined by Blaze *et al* at AT&T Labs [13, 15]. Trust management uses policies and credentials to provide a way of making access control decisions in situations where trust is required. More specifically, implementations of trust management systems aid applications in deciding whether particular operations are allowed or not. The decision will need to take into account what the operation is, who requested it, what the local policy allows, the requester's credentials, and other application specific factors. Trust management systems provide applications with an interface to help them with such decisions, and provide a standard language for writing the policies and credentials. Also, decisions are made in a decentralised manner. For example, rather than making trust related queries to a centralised service, each principal has their own "trusted" system locally[2] to them. It is difficult to find an organisation

[2]"local trust" in this dissertation doesn't mean geographically local. It means that the trust is direct and locally managed based on direct experience involving the principal.

that is universally trusted across the world to provide a centralised service. Decentralising trust management means a company or even an individual can have their own local (and therefore locally trusted) trust management system. Research into trust management techniques by Blaze *et al* led to the implementation of the PolicyMaker and KeyNote trust management systems [14] which are now used in various applications [17]. For example, a module has been written for the Apache web server to provide access control mechanisms for web resources.

KeyNote has its own syntax to allow assertions to be written. These assertions take the form of policies or credentials. Each principal who is planning to have a trust relationship with another will write a policy assertion which states who they are willing to trust and under what conditions. This is usually a list of trusted public keys and a set of condition values that need to be met depending on the situation and its requirements. An example could be for making a file access control decision, where more privileged users may be allowed to write to as well as read certain files. Credential assertions are created and distributed to trusted principals with the allowed conditions encoded within them. These assertions can be digitally signed by the creator to ensure their integrity and essentially serve as permission (or certificates) to perform the specified tasks.

KeyNote also provides tools for the creation of keys and verification of signatures but its primary tool is the compliance checker. When a trust decision needs to be made, the required credentials are passed to the checker along with the relevant policy. It will verify the signatures on the credentials and calculate whether they comply with the conditions set out in the policy. KeyNote will return a value such as true or false, however the result can be more fine grained if desired.

Galice *et al* [44] describe a protocol called Common History Extraction (CHE) to build trust where there is no centralised infrastructure. CHE bases its trust management decisions on previous transactions with other nodes where each device records a history of past transactions. Nodes can then search for previously met nodes and can mutually authenticate and cryptographically prove that they have really met before.

Other work includes applying trust management to web services [92], web applications [27], and to maximising privacy [110]. Details of other trust management systems and applications are given in [48, 93].

2.6 Trust in E-commerce

Since the beginning of the World Wide Web, sites appeared that would allow visitors to search their catalogue, purchase goods and have them delivered to their door-step. Initially, this was a risky transaction over a very new and immature medium. Also, the number of people who owned a Personal Computer (PC) with an Internet connection was low and mainly consisted of people in the IT or computing industry or academia. Eventually, more of the general public owned a PC and every well known brand or shop would have its own web-store. The web consumer data analysed in 1999 [52] shows that the majority of consumers would not shop online due to the fact that they did not have trust in the security of doing so. For example, they feel it isn't safe to input credit card details over the web, or feel that privacy is at stake. As the Internet matured, more mechanisms were implemented in an attempt to secure transactions and hence increase consumer trust when shopping in cyberspace [43, 99, 100]. For example, securing the transfer of payment details (such as SSL) and measures taken by banks to deter or protect against credit card fraud. Also, web-stores began to provide their own guarantees such as policies regarding the privacy of customer data.

So far, trust in a business-to-consumer setting has been described. However, trust in e-commerce is more of a problem nowadays since the growth in the number of online marketplaces and communities. Such websites now make it possible to interact on a person-to-person basis and participate in transactions with completely unknown principals [32, 103]. Even before the advent of sites such as eBay and Amazon, people were trading on a person-to-person basis on Usenet newsgroups. In [68], Kollock discusses the need for trust in online markets and states that trusting online is similar to the structure of the Prisoner's Dilemma. For example, because a person-to-person transaction is likely to involve a bilateral exchange, it is tempting for one to receive a service and not reciprocate. If both parties hold back on their part of the exchange, then both will be worse off. Kollock was one of the first to consider the use of reputation systems as a way to gauge risk when no face-to-face contact will take place during a transaction.

In [10], the notion of using reputation to establish trust is applied to online communities such as eBay. However, individual agents don't have their own rep-

utations but instead they fall under the reputation of the community to which they belong to as a whole. When agents wish to transact with others, the reputations of their corresponding affiliations will be taken into account. The outcome will effect the reputations (positively or negatively) of the affiliated communities. This gives a community the incentive to punish or remove agents which are damaging the overall reputation of the community. The authors use game theory to prove the concepts of a community responsibility system and that it will be effective in building trust in impersonal transactions. To act incorrectly in a transaction will not damage an individual's reputation but will damage the reputation of the community to which they belong. This now becomes a local problem which can be solved internally to the community. It is likely that both the community owners and their well behaved members will want to investigate and possibly purge anyone who might be a liability to their overall group reputation. An example of community reputation is evident in computational grids (as we'll explore later in this dissertation) whereby resources are shared between large organisations or universities. Suppose that principal x of organisation A was to abuse a service provided by organisation B. This might cause problems when another principal of organisation A later wishes to use B's service as x has lowered the reputation of A (in B's eyes). It is now a local problem whereby A can identify and discipline x or any other member who may be damaging their reputation.

2.7 Trust Certification

Traditional certification authorities (CA) issue certificates to websites so that the public key of a server can be verified by a browser for SSL purposes. This allows users to trust that their credit card numbers are being encrypted only for the intended recipient. Now CAs exist that will certify other aspects of a site. For example, that they have a privacy policy that conforms to the CA's regulations, or that a site isn't malicious or fraudulent.

McAfee's SiteAdvisor [3] is a free browser toolbar. It claims to keep you safe from online fraud, spam, ad-ware and other malicious content on the web. Each time someone visits a website, the name of the site is sent to McAfee and a reputation score is sent back to the users browser which then displays it. This

could be seen as an invasion of privacy as McAfee will have access to the full history of sites visited but most people are happy to sacrifice some privacy in order to increase their online security or they won't even consider this factor at all. McAfee regularly run tests on websites and assign safety ratings to them. Ordinary users can comment on whether they agree with these ratings which could eventually alter the rating after further analysis by McAfee. A similar tool to SiteAdvisor is called WOT [6].

Richard Clayton at the Cambridge Security Group has written about how SiteAdvisor works well in most cases but can give inaccurate and misleading advice [29]. He shows some example sites which have slipped through the net and have been given a "green" rating. The example given is an e-commerce site which doesn't accept credit cards but only Western Union money transfers for products that are obviously under-priced compared to their real market value. Customers of this site have complained of not receiving goods after transferring the money. Of course, it is a complex task to rate every site on the Internet but this shows that reputation systems can be hard to get right, especially on this scale. Again, rather than users trusting a site directly, they are transitively trusting McAfee to provide accurate ratings which it cannot necessarily give.

Although these are not strictly reputation systems, they still rely on people trusting certifications from organisations which do have good reputations.
TRUSTe [5] and BBBOnLine [1] are among the most popular. These organisations certify websites after assessing that they have specific policies in place that satisfy the privacy requirements of the organisation. A subscription fee also needs to be paid for the right to carry the certification logo on the website thereafter.

Edelman identifies problems of adverse selection with such certifications in his paper [39] where his results show that certified sites are more than twice as likely to be untrustworthy than uncertified sites. He also shows that sponsored adverts on search engines are also more likely to lead to malicious websites than the organic search results. To the naïve user, these adverts appear to be recommended by the search engine with the user being unaware that the advertiser is paying to have his link displayed above organic search results for certain keywords. This is another example of the problem with a naïve approach to transitivity of trust. Similar problems occur with social networking as we shall see in the next chapter.

Chapter 3

A Peer-to-peer Application

3.1 Introduction

Peer-to-peer (P2P) based networks are widely used on the Internet to enable file sharing, streamed media and other services. With a traditional client-server based network, many clients connect to a fixed server. In contrast, P2P clients are all considered equal and connect directly to each other. Because of this topology, tasks such as sharing files and other resources can be more efficient as a client can connect to many other clients and download content simultaneously.

Much of the content currently distributed via P2P networks is either illegal or violates copyright laws in some way. However, there are also many legitimate reasons why content might be distributed in this way, and there is also copyright free content available such as open source software. P2P protocols such as BitTorrent enable sharing of very large files such as operating systems, and many Linux based distributions are available in this way in order to lower the load on an individual server.

P2P networks have many advantages such as scalability, and due to there being no centralised server, network loads can be easily balanced. However, for the same reasons, a problem with P2P networks is that all peers are regarded as equal and there is no real way to moderate content. Anyone can use a P2P client and share any files they wish. Bad users can easily insert corrupted files into a net-

work[1] which are searchable by other clients and will therefore propagate further. Even good users might be unaware that they are serving incorrect files from their computer. To counter this, hosts might publish an MD5 check-sum on their web-site. However, this is unlikely and it is the user's decision whether and how they actually verify this, and getting hold of the correct checksum leads us back to the initial problem. Also, this approach assumes that the trustee is the original source and not just a middle-man provider.

This chapter explains how trust* could be applied to P2P networks to guarantee the integrity of files being shared. This chapter uses the Turtle P2P client [88] as a basis on which to discuss the approach, although trust* can easily be applied to various other P2P clients in the same manner. Turtle enables files to be shared among friends (people whom you know in the real-world) in the hope to improve safety and overall integrity of the shared content. However, trust isn't transitive in social networks. Applying trust* to Turtle will additionally allow files to be safely shared with unknown principals without the need for transitive trust. Trust* achieves this by providing incentives to act correctly and deterrents for acting carelessly.

3.2 Trust in P2P Networks

Due to the nature of P2P networks and the likelihood that end-to-end interactions will be between completely unknown and untrusted principals, peers in a network need a way to mitigate the risks they might incur if they temporarily trust others. The risks involved are likely to vary depending on what is actually being shared. For example, software should not be corrupted in any way, documents should be authentic and music should be licensed.

There are many security and trust issues related to P2P networks [9, 57, 78, 107] and the trustworthiness of others is normally gauged using some kind of reputation system [58, 69, 98]. However, as mentioned previously, reputation systems have a vital flaw; they require that trust is always transitive [61] which can be a dangerous assumption [26].

[1] Indeed, this may be done by the music industry to discourage people from using P2P.

According to Jøsang *et al* [62], transitivity is possible with the correct combination of the referral and functional variants of trust (see Section 2.3). Trust* instead allows the risk involved when having indirect or transitive trust in another to be underwritten. For example, Bob is not only making a recommendation to Alice, but also offering compensation if something goes wrong. The trust scope is decided locally between Alice and Bob when the guarantee is created. It is assumed that the final guarantor in a trust* chain will have direct functional trust in the end-point (or trust*ee).

Turtle [88] is a P2P client with the intention of providing privacy and safety by sharing only between direct "friends". The client requires you to list your friends whom you trust to share their files with you. The Turtle protocol works by only sending search queries for files to these friends, who pass on the query to their friends as their own query and so on. Such queries and their results are only ever swapped within these local trust relationships. The second stage is for the original requester to choose the file to be downloaded from the list of returned results. They request the file locally from the directly trusted guarantor who in turn requests the file locally from the next principal in the chain. This continues until the end-point is reached (in a similar fashion to how the search query is made). The file itself is then repeatedly downloaded within these individual trust relationships until the request originator is reached.

3.3 Incentive and Deterrent Payments

Two types of payments are used in the trust* model; these are forfeit and commission payments. A forfeit is used to either deter a principal from defaulting on what they have guaranteed or to provide compensation to the other party if they do. The commission payment was introduced in order to provide an incentive for a principal to act as a guarantor and can be seen as a spot price for a guarantee. For example, a principal needing to trust* another would pay this commission to a guarantor whom they trust directly. Or a principal needing to be trust*ed by another would pay the commission to a guarantor who trusts them.

Forfeit and commission payments serve different purposes and don't need to be of the same type (or paid by the same means). Also, these payments and the

actual service being provided need not be like-for-like.

Both the cost of a guarantee and the forfeit that should be paid if it is broken are variable and can be set by a guarantor to reflect their perception of the risk involved in providing a guarantee. For example, as a risky guarantee is more likely to be broken, a higher forfeit might be required by the guarantor to cover his losses (e.g. from the serving peer). A low risk guarantee is unlikely to be broken and so the guarantor will get his incentive through the commission because a forfeit payment is less likely to occur. Another incentive to provide a guarantee is to make a profit from a forfeit. Assume that Carol is trust*ed by Alice with Bob providing the guarantee to Alice (refer back to Figure 1.1). If Carol defaults, the forfeit from Carol to Bob might be more than what Bob has to pay Alice. Note that this gives Bob an incentive to hope that Carol defaults. Alternatively, Carol may pay Bob a commission instead of a forfeit, in which case Bob hopes that Carol doesn't default. The second case is like buying insurance. Commission c has the same expectation (but lower variance) for Bob as $p \cdot q \cdot t$, where p is Bob's estimate of the chance of Carol defaulting, and q is his assessment of the chance of Carol paying the forfeit t (we'll come back to this in Chapter 8).

These considerations lead to some interesting effects regarding the commission and forfeit rates along a chain of guarantees. In this scenario, if Carol were to default the guarantee, only Carol will be out of pocket as the forfeit rate is higher at her end of the chain (and decreases towards the trust*ing end). Every guarantor will make a profit in this case but if we consider a longer chain where risk perceptions fluctuate, guarantors might lose out. For this reason, it is likely that guarantors will only provide guarantees where they believe the rates involved will make them better off with high probability in the long run. This flexibility of perception is vital in ensuring that guarantors get their incentive and principals who might default are sufficiently deterred. The fact that perceptions of risk differ is after-all why we needed trust to begin with.

3.4 Applying Trust* to Turtle

Turtle's localised trust setting is perfect for also finding routes of trust* guarantees, as the query and result route could also be used to make up a chain of guarantees[2]. Extending the example to a longer chain, Alice wants to download file X and sends a query to Bob whom she trusts. Bob forwards this query to Carol whom he trusts. Carol continues to forward this to her friends. David receives the query, he has file X and sends back a positive response to Carol which is forwarded back to Bob and then Alice. Assuming now Alice chooses David's file via Bob from the list of search results and requests that it comes with a guarantee from Bob, a guarantee chain could be negotiated at the same time as retrieving the file. The scope of the trust* guarantee is also negotiated between each pair which states the terms of the guarantee and what constitutes a breach. For example, Carol might guarantee only certain types of files from David. She might be happy to guarantee against any of David's music files but considers the software that he shares as risky so Carol will not guarantee these files. Trust* can be parameterised so as to enable these fine-grained decisions to be made. Even when Carol trusts David directly, she can still be selective over what she'll actually guarantee (and define different conditions to which a guarantee applies).

Suppose that the file X is corrupt in some way. Alice may have inspected the file herself either manually or by calculating a checksum. Alice can claim the forfeit from Bob. Bob may also claim from Carol. Suppose David does not care if his files are correct. So rather than Carol claiming from David, she is likely to stop trusting him altogether, or not guarantee against him again, or charge a higher commission from Bob in future for providing the guarantee in order to reflect what she perceives as the increased risk.

Eventually, say that David is habitually sharing corrupt content and refuses to compensate for losses, all principals who once trusted him are likely to never guarantee his files again. In a commercial context, where David is paid to provide a service, it is David's incentive to reimburse Carol in order to maintain her trust in him. Moreover, in a fair P2P system where credit is gained depending on

[2]Other possible ways in which trust paths can be found using P2P search algorithms are explored in [33].

the quantity of uploaded content, and used to download files from others, David will also have trouble buying guarantees from others in future (or they will be very expensive for him). In this example, the commission can be thought of as a payment for insurance.

The Turtle client was originally developed with an emphasis on privacy and safety of sharing files that might be of a controversial or provocative nature. Due to the localised trust in a trust* chain, such privacy can be easily maintained[3]. However, privacy is not so much of an issue when sharing open content, or in other applications where the integrity of the content is more important.

3.4.1 Good Case

1.	$A \longrightarrow B$:	Can Alice have a guarantee of David, forfeit=t, commission=c
2.	$B \longrightarrow A$:	Negotiation, new forfeit=t, new commission=c
3.	$B \longrightarrow C$:	Can Bob have a guarantee of David, forfeit=t', commission=c'
4.	$C \longrightarrow B$:	Guarantee of David to Bob, id=x etc.
5.	$B \longrightarrow A$:	Guarantee of David to Alice, id=x etc.
6.	$A \longrightarrow B$:	Guarantee x is OK
7.	$B \longrightarrow C$:	Guarantee x is OK

Table 3.1: P2P good case protocol example.

Table 3.1 shows a typical good case protocol run. Supposing Alice has searched for a particular file and finds that David has a copy of it. Alice doesn't trust David and wants to be guaranteed that the file is the original version. Bob is on Alice's list of friends and so receives a request for a guarantee (step 1). Included in the request is a commission offer to Bob and a forfeit requirement. Carol is in Bob's list of friends and so also receives a request for a guarantee. Carol trusts that David can provide correct files. However, Bob might negotiate the commission and forfeit values depending on the perceived risk of Carol's guarantee (step 2 and repeating step 1 again). The same might happen between Carol and Bob (although not shown in the table above) in which case the new values will reflect Carol's risk perception of guaranteeing David. After negotiation, Carol will generate a guarantee for Bob and Bob will generate a guarantee for Alice. Finally, assuming

[3]We return to this issue in Section 8.4.2.

that the file was as expected, Alice will notify Bob of this who will in-turn notify Carol.

3.4.2 Bad Case

Table 3.2 shows an example of a bad case protocol run. This protocol follows the first 5 steps from the good case protocol above. However, in this case, the file downloaded is incorrect in some way and Alice makes a claim from Bob (step 6). Bob will need to pay this forfeit if he wants continued trust from Alice (step 7). Bob will then claim the forfeit from Carol (step 8) who is obliged to reimburse the forfeit to Bob (step 9). There are a number of factors that might affect Bob and Carol's actions thereafter. For example, whether this is David's first offence, or whether he later reimburses Carol. However, if David does become a liability to Carol (or Carol to Bob), she will simply stop guaranteeing him. Commission rates are likely to increase along the chain making the prospect of buying a future guarantee of David along this route unfeasible.

6.	$A \longrightarrow B$:	Make a claim on guarantee x
7.	$B \longrightarrow A$:	Pay forfeit t
8.	$B \longrightarrow C$:	Make a claim on guarantee x
9.	$C \longrightarrow B$:	Pay forfeit t'

Table 3.2: P2P bad case protocol example.

Other possible problems could occur if Alice or Bob lie and make unwarranted claims. Other possible scenarios such as these are discussed and simulated in the following chapter.

3.4.3 Required Changes to Turtle

Turtle provides the functionality for sharing among locally trusted friends. It deals with the routing of search queries and file transfers within these local trust relationships as explained above. This section describes the modifications that would need to be made to Turtle in order to apply the trust* mechanism.

Small changes would need to be made if a principal requires a guarantee for a file that they plan to download. For example, a flag could be set as part of the

retrieval process stating this requirement (which will include a commission offer and forfeit requirement). This will initiate the trust* protocol and guarantee negotiation process. The requesting principal will receive the file with an accompanying guarantee. Alternatively, if multiple routes are available, a list of options will be presented to the requester. Here, a trade-off can be made between the cost and the level of compensation before selecting a route. Turtle would need to be changed to allow such guarantee negotiation, generation and verification to occur during this stage. A friend-list of a principal could also hold details of the maximum c values that they are willing to pay and the minimum forfeit t values they are willing to receive for a guarantee from each friend. It is worth noting that this all assumes that each principal is using the Turtle client to handle trust management (and maybe even payments), however, later Section 5.4 discusses the possible heterogeneity of a trust* chain.

3.5 Service Contracts

Most services provided by servers over a distributed system or network have (like those in the real-world) an underlying contract or agreement. This could simply be that service X will be provided for a fee P and that the service will conform to the terms and conditions of X. In P2P networks, such guidelines cease to exist and clients connect to other clients to become an equal part of the network. Peers are usually free to download anything they wish from other peers and vice versa. Alternatively, there may be situations where content could be charged for or for which a particular service level agreement is in place. However, it is more likely that peers in a P2P network hold a "download at your own risk" policy regarding the files that they are sharing. This is where using trust* could be helpful for providing assurance. Following from the example above, David doesn't care if someone wants to download file X and doesn't care if they aren't happy with it. However, Carol has previously downloaded files from David, and hence trusts that his files are of a high standard. Bob trusts Carol and Alice trusts Bob in the same way so Bob's guarantee reduces the risk for Alice and Carol's guarantee reduces the risk for Bob respectively. If Carol was wrong, she will pay the agreed forfeit to Bob who will compensate Alice with their agreed forfeit.

However, David hasn't necessarily done anything wrong and isn't obliged to reimburse Carol. Carol however is likely to lower her high perception of the quality of David's files and perhaps never guarantee him again, or offer a lower forfeit, or require a higher commission.

Bob's motivation to provide the guarantee could be a commission payment from Alice[4]. Bob will set the level of this commission depending on his perception of the probability of David defaulting (or in relation to the how much he trusts Carol's referral)[5].

3.6 Payment by Resource

The forfeit and commission payments in the trust* model aren't restricted to purely monetary payments. In P2P networks, these payments could be made by using the resource itself as currency. Due to the heterogeneity of the local trust relationships (discussed later in Section 5.4), the payment medium could vary along a trust* chain. Also, the type of payment that might provide an incentive or deterrent could vary from user to user.

Assume that a P2P system were to provide an incentive to share files by awarding download credit to peers. This credit could reflect the amount that has been downloaded from an individual peer or the amount of content they are currently sharing. This credit could be used to reward the peer by increasing download bandwidth or to allow them to download more files. A simple rule could be that for every file uploaded by a peer, a file can be downloaded from another by that peer. This credit might be a global currency but could equally well be a token provided by one peer to another only for use by that individual.

Following the examples given in this chapter, assume that Alice claimed a forfeit from Bob. The forfeit might be in the form of tokens that allow Alice to download files via Bob. After all, Alice already trusts Bob. Alternatively, Bob could issue tokens which act as a commission payment for future guarantees that

[4]In a commercial case, where David provides a service for payment, David may pay Bob a commission for acting as an intermediary (and maybe a forfeit later if Alice claims).

[5]Provided Bob's estimate of the probability of David defaulting is lower than Alice's estimate, both Alice and Bob will be happy with the guarantee. See Section 8.4.4.

Alice might need from Bob. The point is that the payment commodity could be the resource itself whether it be the actual content (e.g. a file) or a means of getting it (e.g. a free guarantee from Bob).

3.7 Conclusion

This chapter has shown how trust* can be used as a mechanism for guaranteeing the integrity of content or services provided over a P2P network. Trust* builds on the idea of sharing with friends in the Turtle client but also guarantees the integrity of downloaded content from non-friends or unknown peers, thus removing the need for and the risk involved when friendship is assumed to be transitive.

Using trust* in this way reduces the risk involved for the downloader as they will be compensated in the worst case scenario. It does so without the need for requiring transitivity of trust, and privacy is still maintained. This is because the guarantees and payments are confined within the same localised pre-existing trust relationships that are already used to communicate the actual search queries and their corresponding results. This approach therefore allows complete localisation of trust management.

We have argued that applying trust* to P2P file sharing will also be beneficial in guaranteeing the integrity of free content such as open source software or copyright-free movies etc. Indeed, trust* will potentially help P2P sharing networks to become “respectable” (instead of the Wild West).

Chapter 4

Simulating Trust*

4.1 Introduction

In order to test the trust* model and its application to various situations, parts of it were simulated using the Repast Simphony agent based modelling toolkit [80, 81, 82] available at [4]. This chapter demonstrates a simulation of the trust* peer-to-peer application as described in the previous chapter. Summarised results of simulations are also presented in this chapter, with more detailed results given in Appendix A. Subsequent application chapters in this dissertation describe the variation in simulation details in relation to the other applications being explained.

4.2 The Repast Modelling Toolkit

The Repast Simphony toolkit provides tools for modelling entities called agents. These maintain a set of properties and behaviours which can exhibit learning behaviour. The toolkit also provides an environment where such agents can interact with each other to form a simulation. In this work, the Repast Simphony framework is used to model the actors in a trust* protocol.

The simulation environment allows agents to be added and networked to each other. It also provides tools for data logging for later analysis. Properties and other simulation attributes can be manually altered during a simulation if necessary.

Repast Simphony integrates with the Eclipse IDE and enables agents to be

modelled in a Java-like language called Groovy. The following section outlines the types of agent, their properties and their behaviours that are used in a typical trust* simulation.

4.3 Modelling Trust*

4.3.1 Agents

To simulate trust*, three types of agent are used to distinguish specific functions in the protocol. Note that in reality, a principal is likely to have multiple "hats" and perform the tasks of all three agents simultaneously in respect of different instances of the relationship. The three types of agent are outlined below.

Trust*er A principal who is the trusting end-point of a trust* relationship. In the simulation, all trust*er agents are called Alice and they are responsible for initiating a trust* relationship[1].

Guarantor A principal who is providing a guarantee to another about someone they trust directly (or indirectly). In the simulation, all guarantor agents are called Bob (uniquely numbered). Chains of Bobs can also be simulated (see Section 4.5).

Trust*ee A principal who is the trusted end-point of a trust* relationship and is being trust*ed by the trust*er. In the simulation, these agents are called Carol.

4.3.2 Agent Properties

Each agent in a simulation has properties which they can control and that other agents can see. Below are the important properties that are used when following the trust* protocol. Other properties include claim counters, credit transfer logs and references to other agents in a protocol run.

[1]This is usually the case, however, later in this dissertation we'll describe an application where Alice is still the trust*er but doesn't invoke the protocol.

credit The credit property is a floating point decimal with the default value of 0.0. It represents the wealth of an agent. During a simulation run, an agent's changes in wealth can be easily identified and whether they have become better or worse off after using trust*.

cOffer and fOffer The cOffer and fOffer properties represent the current commission (offered to a guarantor) and forfeit (required from a guarantor) offers respectively when making a request for a guarantee. The fOffer can been seen as more of a forfeit requirement than an offer in the P2P application scenario. A guarantor is likely to refuse a request if the fOffer is too high, however will want the cOffer to be as high as possible.

cMin and fMin The cMin and fMin properties are the lowest commission and forfeit rates that will be offered or accepted. A guarantor will want the cMin to be as high as possible but will not care about the fMin. However, a claimant will set his fMin property to at least a satisfactory level that will be enough to compensate him.

cMax and fMax The cMax and fMax properties are the highest commission and forfeit rates that will be offered or accepted. The values are the opposite way around to those of cMin and fMin. For example, a guarantor will want to set the maximum forfeit they are willing to pay. Conversely, a guarantee buyer will have a maximum threshold to how much commission they are willing to pay for a guarantee.

active The active property is a boolean value stating whether or not a particular Bob agent is currently available to act as a guarantor for Alice.

4.3.3 Initial Values in a P2P Simulation

Table 4.1 shows the initial values of Alice and Bob. In the P2P simulation, Carol's property values aren't applicable as she doesn't explicitly take part in the protocol unless Bob attempts to claim from her. For the purpose of this simulation, Bob will never ask Carol for a reimbursement of a forfeit[2] and will simply increase the

[2] Although this restriction is relaxed later in Section 4.5.

cost for Alice (effectively making it unaffordable for her to continue).

	Alice	Bob
cOffer	1	n/a
fOffer	10	n/a
cMin	0	1
cMax	5–10	∞
fMin	6	0.1
fMax	∞	25–30

Table 4.1: Initial values for Alice and Bob.

In the simulation, these values are initialised in the same manner for each agent in order to allow results to be easily comparable. However, in a real scenario, these levels would be set individually in relation to some real-world trust or reputation metric. For example, Alice will reflect her personal trust in Bob lowering her initial fOffer and fMin values. These values are variable to the level of current trust in another principal which is assumed to be reassessed before each protocol run.

Alice's cOffer and fOffer values start relatively low. Bob never needs to forward a request in this example so his cOffer and fOffer values aren't applicable. However, where multiple guarantors are needed between Alice and Carol, Bob would need appropriate cOffer and fOffer values in order to forward a request. Also, in this case, Bob would need separate cOffer and fOffer values depending on whether he is providing a guarantee (to Alice) or forwarding the request (to another guarantor). Section 4.5 describes a more complicated multiple guarantor simulation.

Alice's cMin and fMin values are the opposite way to Bob's cMin and fMin values. For example, Alice will want the commission to be low and the forfeit to be high whereas Bob will want the commission to be high and the forfeit to be low. This is also evident with their cMax and fMax values. Note that in the simulation, Alice's cMax is set to a randomly chosen value within a specified range. The same applies to Bob's fMax value. This is to add a small degree of realism to the simulation.

4.3.4 Model Attributes

A simulation can have multiple global attributes which are accessible by all agents. These will normally stay constant unless manually changed during a simulation run. Both attributes in the P2P simulation make use of a Repast library method where a threshold can be set to affect the probability of a random number being returned as `true` or `false`.

malwarechance Is the probability that Carol's shared files will be incorrect, illegal or corrupt. The value can range from 0 to 1 where 0 defines no malware and 1 defines 100% malware.

truthchance Is the probability that Alice will be truthful when claiming. This enables the simulation of Alice making false claims where 0 defines Alice to never be truthful and 1 is 100% truthful.

Another attribute could be used to define how often Bob might refuse to pay a forfeit to Alice. This attribute wasn't used in this simulation as it is assumed that guarantors will always behave correctly[3]. This is so that analysis of the direct effects to the end-points (Alice and Carol) is not complicated by interference from bad guarantors. For example, if a particular guarantor refuses to pay Alice, Alice will simply stop trusting him to provide guarantees. This will limit the routing possibilities between Alice and Carol and hence affect the simulation run time regardless of how well behaved Alice or Carol were. It is assumed that guarantors want to maintain their trust from Alice and will honour any forfeit requests.

4.3.5 Agent Behaviours

These methods provide the main functionality for following the trust* protocol in a simulation.

initiate() This is the first method that is invoked in a simulation run and is called once every tick[4]. It makes Alice initiate the protocol by searching for a

[3]This is also re-considered later in Section 4.5.

[4]A tick is a single unit of time in a simulation which can be used to schedule events.

guarantor (a Bob) between herself and the download source (Carol). Assuming that a guarantor is found, the starting values are set and Alice starts the protocol by calling `requestGuarantee()` on the guarantor agent. This sends details of the end-points and the current fOffer and cOffer values. Otherwise, if no guarantor can be found, the simulation is stopped.

requestGuarantee() Called by Alice to request a guarantee from Bob. Once invoked, Bob will check if the commission is high enough. If so, he will make sure the forfeit isn't too high. If either checks fail, `reject()` will be called on Alice with the reason why. Otherwise, Bob generates a unique id number and invokes `sendGuarantee()`.

reject() Called by Bob to reject a guarantee request. Depending on whether it was rejected because the commission offer was too low or because the forfeit requirement was too high, Alice will increase the commission or lower the forfeit respectively. Alice will check that the new offer values are still within her minimum and maximum bounds and resend the request with these new values. If not, the current guarantor will become inactive and Alice will search for another Bob.

sendGuarantee() Called by Bob to send a guarantee to Alice. For the sake of the simulation, it is decided here whether the file is going to be incorrect by generating a random number and checking it against the attribute `malwarechance`. The `download` method is then invoked on the Alice agent and Bob is paid his commission.

download() This method represents Alice downloading the file from Carol. The file is checked (even though the type of file has already been decided) by Alice and makes a claim to Bob if it is incorrect. If the file is correct, Alice will decide whether to make a false claim by generating a random number and checking it against the `truthchance` attribute. Whatever the outcome, Alice invokes `response()` on Bob stating whether or not a claim is being made. Also, the number of claims and false claims are logged at this point.

response() Allows Alice to respond to Bob regarding a guarantee. It simply checks whether or not a claim has been made. If so, the forfeit is paid by Bob to Alice by invoking her `payment()` method.

payment() The payment method allows agents to make payments to other agents. This method handles the exchange of credit and logs all transfers of commission and forfeit payment for each agent. If this method is invoked on an agent, and a forfeit has been paid, they will increase their required forfeit from the payee in future. This is where reassessment of these values takes place.

4.4 Simulation Test Scenarios

The primary reason for simulating a trust* protocol is to analyse the outcomes of the participating principals in various situations to ensure that they get the right incentives to act correctly and deterrents for defaulting. These situations can be simulated by varying the `malwarechance` and `truthchance` attributes and recording the resulting values of the agent properties. Because the simulation involves invoking a protocol run every tick, the protocol can be repeated continuously and the long term effects for each principal will become evident. The effects and outcomes that are observed are as follows:

- The simulation is programmed to stop when all possible trust* routes have been exhausted, so the total tick count at the end of a simulation gives a good indication of how long trust* could be used between Alice and Carol before all guarantors become inactive. This will vary depending on the values of the simulation attributes and the tolerance of an individual agent, but will be comparable.

- The credit levels of each principal gives a good indication of who made a gain or a loss after a series of protocol runs. This can also be linked to the tick count to show how long a guarantor might have held out until they became inactive. Or similarly, how long Alice could continue to make false claims before losing all possible routes.

- Once results have been generated for a particular simulation run, the steepness of the increases in values such as fOffer and cMin become easily visible. This should be representative of the number of claims that Alice might make. Increases in cOffer and decreases in fOffer will also be caused by request rejections.

- Results also show logs of the total expenditure and gain of credit whether through commission or forfeit payments. Also, the number of claims and false claims is logged.

Table 4.2 shows the simulations that were run and the `truthchance` and `malwarechance` values that were set for each. Full results are given in Appendix A, however a summary of results is analysed in the Section 4.6.

The tests have been split into two sections to test different variations of principals being "good" and "bad". In tests 1 to 6, Alice and Carol begin by both being bad (i.e. Alice never tells the truth and Carol always shares bad files) and gradually become good (i.e. Alice always tells the truth and Carol never shares bad files). These tests show how effective it is to use trust* starting with no principals behaving ranging up until all principals are behaving well. In tests 7 to 12, Alice starts by being bad and Carol starts by being good. This is gradually reversed and eventually, Alice will be good and Carol will be bad. These tests show the effectiveness of trust* when one principal is behaving when the other might not be and vice versa.

To keep tests simple, one Alice agent, one Carol agent and five Bob agents are simulated. Trust paths have been defined so that five possible trust* routes can be found between Alice and Carol. The simulation topology is shown in Figure 4.1 where the arrows indicate the direction of direct trust. The simulation could be of a larger scale with many more agents and possible trust paths. However, the topology in Figure 4.1 is adequate to see the effects of trust*. Changes in an agent's credit might be influenced by other trust* relationships they might belong to. For example, say that Alice is being paid high forfeits by Bob. In this scenario, she would appear to be quite wealthy. However, suppose in reality Alice is also a guarantor in another trust* relationship, she might have to pay forfeits to other principals. Of course, she is likely to ensure that she'll make a profit,

Test	Truthchance	Malwarechance
1	0	1
2	0.2	0.8
3	0.4	0.6
4	0.6	0.4
5	0.8	0.2
6	1	0
7	0	0
8	0.2	0.2
9	0.4	0.4
10	0.6	0.6
11	0.8	0.8
12	1	1

Table 4.2: Simulation test setup.

however she won't appear to be as wealthy as in the first case. Simulating trust* by only allowing certain agents to perform one particular task removes these outside interruptions and allows the true effects to each type of principal to be clearly seen.

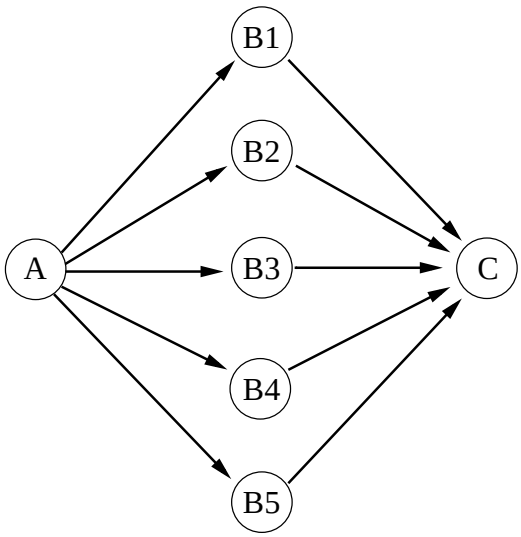


Figure 4.1: Trust topology between Alice and Carol.

The model also allows longer chains to be simulated. This is achieved by forcing the guarantors to record guarantee requests and decide whether they need to

forward the request to another guarantor. A chain of five principals is the shortest for which at least one node is not directly related to either of the end-points. For this particular simulation however, the chains were intentionally kept short to analyse the direct effects on each type of agent. The implications and effects of simulating a five principal chain are explained in the following section.

There are many other circumstances that could have been simulated, but the simulations were deliberately limited to reduce the number of experiments reported in this dissertation to a manageable level. The simulations are just meant to be illustrations of how the trust* concept could work in some practical applications. The real strength of this dissertation is in the number and depth of the different scenarios where trust* could be used.

4.5 Multiple Guarantors

This section describes the design decisions made when simulating the P2P scenario where a trust* chain consists of five principals. Figure 4.2 shows the trust topology between A and C for this simulation.

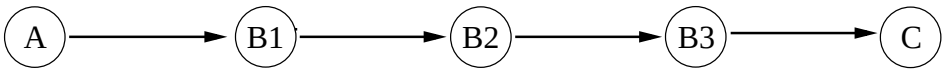


Figure 4.2: Trust topology between Alice and Carol with multiple guarantors in a single chain.

The simulation described previously in this chapter offers a choice of five routes to Alice each through a different guarantor. It was assumed for simplicity that each guarantor was directly trusted by Carol. It is more likely that a trust* route will be longer where certain nodes might have no relation to either of the end-points. For this simulation, some major design changes were made to enable multiple guarantors. These are:

- Now there are three guarantors (we'll call them B_1, B_2 and B_3) which means that changes have been made to how requests are dealt with. In the original simulation, a guarantor only needed to worry about how likely it is that Carol will default (and therefore how much his expected loss will

be to Alice). In the multiple guarantor scenario, a guarantor will not only be providing a guarantee to another principal, but will also be receiving one. Therefore, a guarantor needs to decide on different values for the `cMin`, `cMax`, `fMin` and `fMax` properties depending on whether they are receiving or forwarding a guarantee request. For example, suppose that B_2 has received a request from B_1, he will want a high commission value (as this is what he'll be paid by B_1) and a low forfeit value (as this is what he'll need to pay B_1). If he was to forward the request to B_3, he'll want to pay the lowest possible commission to B_3 but receive the highest possible forfeit from him. The initial values for requesting a guarantee are shown in Table 4.3 and the initial values for providing a guarantee are shown in Table 4.4.

Property	A	B_1	B_2	B_3
cOffer	1.5	1.4	1.3	n/a
fOffer	15	16	17	18
cMin	0	0	0	n/a
cMax	5–10	5–10	5–10	n/a
fMin	6	6	6	n/a
fMax	∞	∞	∞	n/a

Table 4.3: Initial values for requesting a guarantee where a principal wants a low c but high f. Note that for B_i, the `cOffer` is decremented by 0.1 and the `fOffer` is incremented by 1 by each guarantor. In the simulation, these values are calculated from the `cOffer` and `fOffer` values received from the previous principal depending on a guarantor's greed (see below).

Property	A	B_1	B_2	B_3
cOffer	n/a	n/a	n/a	n/a
fOffer	n/a	n/a	n/a	n/a
cMin	n/a	1	1	1
cMax	n/a	∞	∞	∞
fMin	n/a	0.1	0.1	0.1
fMax	n/a	25–30	25–30	25–30

Table 4.4: Initial values for providing a guarantee where a principal wants a high c but low f.

- In this simulation, Alice only has one possible route to Carol in comparison to the five routes in the previous simulation. If any of the guarantors in the chain become inactive (no longer willing to provide guarantees), this will force the simulation to end. In reality, the route could be diverted around the inactive links such as in the previous simulation where Alice would search for a different guarantor. Therefore, the run times for this simulation are on average expected to be five times quicker.

- When a guarantee is claimed by Alice, a guarantor's properties will be affected in both directions. For example, B_1 will decrease his commission offer and increase his forfeit requirement when requesting a guarantee. He will also increase his commission requirement and lower his forfeit minimum for actually providing a guarantee.

4.5.1 Test Scenarios

Several test scenarios were simulated with the five guarantor chain. These are:

- Test 1 — Following the same test set-ups as those in Table 4.2, the credit changes for each principal is recorded. Note that this test assumes that the guarantors will always be truthful. Also, this simulation includes a forfeit reimbursement request from B_3 to Carol if a claim has been made. In this test, she will always reimburse B_3 with the forfeit.

- Test 2 — Again, following the same test set-ups as the previous simulation, but this time with a new attribute `carolpaychance` which defines whether Carol reimburses the forfeit that B_3 might have to pay. The `carolpaychance` is fixed to 0.5 and B_3 has a tolerance of three non-payments before he becomes inactive.

- Test 3 — In this test, Alice and Carol are always good (i.e. where `truthchance`=1 and `malwarechance`=0). However, a new attribute `guartruthchance` defines how often guarantors might make false claims. Ten tests were completed where the value of `guartruthchance` was incremented by 0.1 ranging 0 to 0.9.

There are many more combinations of tests and attribute that could reflect different scenarios. For example, another could be to set a "greed" level for each guarantor to define how much they will alter a `cOffer` and `fOffer` before forwarding a request.

4.6 Summary of Results

From the results produced by the simulations, it is evident that only principals who are well behaved will reap the benefits of using trust*[5]. This is that they can be assured of the content they might download in a P2P network. For good principals, this is enough incentive to continue acting correctly otherwise risk losing these privileges. Moreover, it has proved that bad players in a trust* protocol might temporarily profit, however they will find it harder (or more expensive) to build future trust* relationships. Their trust* usage will be short-lived and they will be isolated from the good principals.

So, the results show that it's in an agents interest to not share incorrect content or make false claims as their future trust* usage will be restricted. For example, if Carol serves bad files, Bob will no longer guarantee her. If Alice keeps claiming and Bob suspects the claims are false, Bob won't provide guarantees to her or they will be expensive in order to cover forfeit costs making it infeasible for Alice to make a trust* chain to Carol (via Bob at least).

A full description of the results generated during each simulation is presented as a series of graphs and analysis in Appendix A. There are six graph types for each test which present various results from a simulation. The first shows the changes in credit for Alice and all of the Bobs. The second shows the changes in Alice's commission and forfeit offers. The third shows the amount of commission she has paid and the amount of forfeit she has received. The fourth shows the commission received by each Bob. The fifth shows the amount of forfeit each guarantor has paid Alice. Finally, the sixth shows the frequency of Alice's claims.

[5] Although from the results, it appears that Alice still benefits when behaving badly. In reality, this isn't likely to be the case as tolerance levels will differ from those simulated (or that cycles of trust* will be built, see later). Also, this will affect her chances and costs of future trust* relationships.

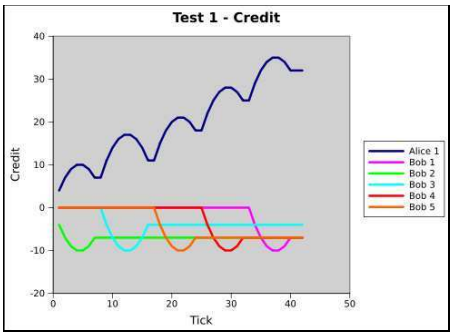


Figure 4.3: Credit values when all principals behave badly.

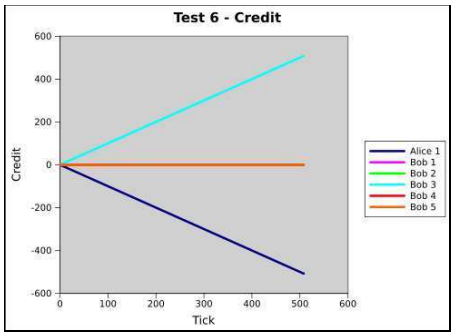


Figure 4.4: Credit values when all principals behave well.

The results of the simulations with a five principal chain are also presented and analysed in Appendix A. Results from Test 1 are given in Section A.3.1, results from Test 2 in Section A.3.2, and results from Test 3 in Section A.3.3.

4.7 Conclusion

This chapter has demonstrated how the trust* model can be simulated to reflect aspects of how it might be used in practice. In particular, the simulation of the P2P application presented in Chapter 3 has been described in this chapter.

The results for this application show that agents in a trust* relationship gain little advantage from acting incorrectly. Moreover, any advantage an agent might gain is short-lived and will be penalised in the future if they wish to build a trust* relationship again. If they find it difficult to find trust* routes, then their future behaviour will have no effect on other parties using trust*.

Simulating trust* will have subtle differences depending on the application that trust* is being applied to. Subsequent application chapters in this dissertation describe the implications and necessary changes that need to be made to the simulation model discussed here in order to apply it to the application in question, and report on the results of further tests.

Chapter 5

A Grid Computing Application

5.1 Introduction

This chapter describes how trust* can be used within a grid network or similar distributed environment. Examples of such environments range from computational grids that might be owned and shared by a company or organisation to volunteer computing projects where anyone can participate. Grids are generally used to solve a computationally intensive problem distributed over multiple machines and can include features such as redundancy, fault tolerance and scalability. Grids also allow organisations to share resources in a cost effective way. For example, a university might share access to their database in return for processing time on another university's cluster. Rather than each university investing time and effort in buying, building, and maintaining their own database or cluster, they can simply share such resources.

Ways in which trust can be built in computational grids (which are likely to span organisational and domain boundaries) is a well researched problem [8, 12, 28, 76, 83, 85, 108, 109]. Popescu [89] outlines some security requirements of the Globe middleware such as needing to cope with a lack of a centralised trust authority and servers which span multiple administrative domains. However, these problems are likely to be evident in any computational grid environment which is required to scale in this manner.

5.2 Globe Distributed Object Middleware

The examples used in this chapter are loosely based on the Globe middleware [11, 53]. Globe's focus is to provide a middleware that is scalable enough to enable worldwide distributed computing.

The Globe infrastructure is built around Globe Object Servers (GOSes) which host Distributed Shared Objects (DSOs) and replicas of other DSOs. Trust management policies can be written by GOS administrators that define the other GOSes which are considered trusted and untrusted. This allows operations to be performed on trusted and untrusted platforms depending on their importance. For example, read only methods can be performed on untrusted servers whereas write (state altering) methods might only be performed on trusted servers. Such trusted servers might be those locally hosted by the organisation and untrusted servers might be those that span other administrative domains.

Trust* can be applied as a solution to building trust in grid middleware and other distributed environments. The trust* model could be used alongside Globe's existing trust management strategies, but could also usefully incorporate some of the ideas introduced by Popescu in his thesis [87]. Popescu's work involved developing the security considerations and functionality of Globe and introduces mechanisms to enable Byzantine fault tolerance through reverse access control and audit in order to maintain the integrity of the DSOs and their replicas in a Globe system, especially in sensitive applications. The following sections explain how these mechanisms relate to and could be used when applying trust* to such an environment.

5.2.1 Byzantine Fault Tolerance

The need for fault tolerance is evident as Globe objects are allowed to be hosted on third-party servers and it is important that Globe objects are behaving correctly. For example, in a critical application such as a stock market system, all data and operations need to be correct (and non-malicious) and so damage prevention would be required. Even one incorrect result or operation could be disastrous (and could replicate, see Section 5.2.4). However, trust* is more likely to be used

for lower value operations whereby new unknown users can be trusted to perform tasks (if the commission rates are high enough and hence the risk perceptions are low enough). If the task is incorrectly performed, the user simply won't be able to continue with that particular project. So a damage control mechanism will be sufficient for maintaining the overall integrity of such a project. For example, Popescu uses two methods for damage control in Globe which are explained in the subsequent sections. These are reverse access control and audit.

5.2.2 Reverse Access Control

Traditional forward access control mechanisms are used to check if an invoking object is allowed to *invoke* methods on another object. Reverse access control is the converse where the invoking object checks whether another object is allowed to *execute* a particular method for it. An object owner can select a trusted group of replicas and write a policy allowing this group to perform particular operations such as those that alter the state of the DSO. These core replicas can then recruit other less trusted replicas to perform the read requests involved. Less trusted replicas serve read requests only and are unable to propagate the request further to other replicas. This relates to the type of policies that are used in trust* although guarantees can be used to allow the less trusted replicas to perform a wider range of tasks. Trust* allows reverse access control to be extended, by permitting a trade-off to occur between the risk involved with a method invocation (or correctness of its results) and the level of compensation required if the results are incorrect.

5.2.3 Audit with Cycles of Trust*

Trust* is intended to be deployed in environments where there is no universally trusted arbiter or referee. If a principal starts claiming a forfeit regularly, the guarantor might either stop providing the guarantees, or may charge more for providing them. Alternatively, the guarantor or trust*ed principal could form a cycle of trust*. Such a cycle consists of a trust* guarantee path in the opposite direction (to an existing trust* relationship) in which the guarantor guarantees compensation if a false claim is made. See Figure 5.1 below.

In Globe, the results of a method invocation on a less trusted replica can be audited by a trusted replica. This is analogous to a cycle of trust*. Rather than the correctness of results being checked as in Globe at present, the correctness of a claim will be checked. For example, in Figure 5.1, assume that Carol is trust*ed by Alice via Bob to execute method x and if Alice considers the results to be correct, she will reward Carol (maybe with a small payment). Otherwise Carol might have to pay Bob if it isn't correct or simply will be removed from the set of replicas hosting a DSO. Suppose that Carol suspects Alice of falsely claiming that the results were incorrect, Carol could make a trust* cycle via David to protect her against this (he might also verify the result and compensate Carol if it was in fact falsely claimed). In a Globe context, Bob could be the auditor (regarding correct execution) for Alice and David could be the auditor (regarding correct claims) for Carol.

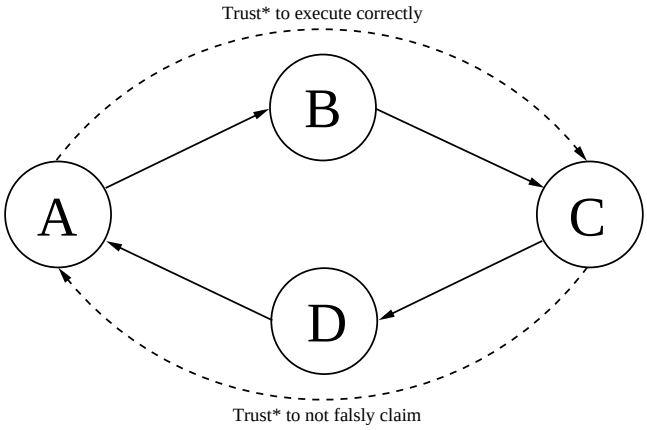


Figure 5.1: A cycle of trust* between Alice and Carol

5.2.4 Damage Prevention

Popescu also suggests two methods for providing damage prevention. These are through state signing and state machine replication. The latter of these might be useful in a more critical or anomaly sensitive application that uses trust*. Replication works by invoking the same method on a number of replicas and choosing the result of the group majority. Although this will be effective in spotting malicious replicas, it is expensive both in the amount of computational resources needed to

handle a request and the latency caused by waiting for multiple replicas to perform the method (although can proceed as soon as a majority of results have been received). Here, there is a trade-off between security and efficiency; the more critical an application is, the more replicas will be needed, although will be more expensive. Less critical operations might only need one or two.

In the event that trust* is being used in a more sensitive application, Globe's replication technique could be used. For example, by building multiple trust* relationships simultaneously via different guarantors between end-points. Suppose that Alice requires two guarantees from two individual guarantors whom she trusts before allowing Carol to execute method x. If Carol fails to execute the method correctly, not only will Alice be entitled to two forfeit payments but Carol will risk losing trust from two principals rather than one. This also makes allowances for the chances that one of the guarantors might be "faulty" and not pay the forfeit.

5.3 Routing

In a P2P network, routing protocols are generally provided within the client software (such as Turtle). Grid middleware (such as Globe) on the other hand are meant to be heterogeneous and don't necessarily need or have a generic routing strategy. A routing algorithm of the user's choice can be used to route trust* protocol messages. This section provides an overview of some of the network routing strategies that would be analogous to trust* routing (or finding an optimal route between two principals).

There are many network routing strategies that could provide the underlying trust path routing for the trust* protocol including those surveyed in [74, 96]. Fixed routing is certainly out of the question as trust relationships are volatile and to configure permanent routes wouldn't work. Flooding, dynamic or even random routing would suit the needs of trust* better.

Finding the best route between two nodes on a network is analogous to finding an optimal route between two principals who wish to form a trust* relationship with one another. The small world phenomenon [77] implies a trust* route can almost always be found. But the "best" route could be the cheapest (according to commission or computational expense) or the most trusted. In this respect,

different levels of trust, forfeit and commission correspond in routing terms to different network Quality of Services.

Most routing decisions are based on some form of least-cost or distance vector criterion and are usually variations of graph search algorithms such as Dijkstra's algorithm [35] or the Bellman-Ford algorithm [42].

Dijkstra's algorithm solves the shortest path problem in weighted graphs between a given source node and all other nodes. It orders paths of increasing length stage by stage. A routing table is initialised by calculating path costs to neighbouring nodes. These are effectively shared with neighbours so paths to all nodes in the network can be made. If a shorter route to a node is encountered, the shortest path is recorded and all nodes update their least cost paths respectively. The algorithm continues until paths have been calculated to all nodes in the network.

The Bellman-Ford algorithm solves the same problem however a node only needs knowledge of its neighbours and their surrounding link costs whereas Dijkstra's algorithm needs complete topological information of the network. This algorithm is more fitting to the requirements of trust* routing.

A popular deployment of a distance vector algorithm is the Routing Information Protocol (RIP). In a network implementing RIP, each router maintains a routing table of information about routes from itself to each destination [102, p86]. A router generally initialises itself by inserting routes to hosts to which it is directly connected. Each entry includes the next-hop address, the cost and an entry age. RIP differs from algorithms such as those above as it only retains the current minimum cost route rather than every possible route. This entry is updated if a cheaper route can be found, however, this allows RIP routers to store little information about its neighbouring hosts. In most RIP implementations, the cost (or distance) might be the number of routers that a packet must pass or could be relative to other computational expense. With trust*, neighbours are directly trusted principals, and the cost would be the commission that needs to be paid to a guarantor in the trust* route.

5.3.1 Routing in Trust*

To find a trust* route, such routing tables could be used although in a slightly different way. For example, if Carol provides a commercial service, then she could maintain a list of her local trust relationships in a similar way to destinations in her physical network.

Assume that Alice needs a guarantee of Carol. Bob is in Alice's routing table and Carol is in Bob's. Alice's entry for Bob will include the commission that Bob will charge for a guarantee. Bob's entry is similar except it states the charge from Carol. There might be situations where multiple routes are possible. Therefore, the cheapest route might be chosen. Routing tables store differences and the trust* tables could do the same. The difference is that guarantees may add to or subtract from the totals before passing the information on. Another difference is that Alice will pay Bob the commission between them only, whereas in networking, the cost between Alice and Bob, and Bob and Carol will be combined to give a total cost between Alice and Carol. A trust* routing table will not require this as Alice will only pay a commission to Bob and it is Bob's obligation to pay Carol. Bob might also have a choice of routes to continue to Carol some of which might be cheaper, but he will surely only provide a guarantee if he probably isn't going to lose out himself. Hence, as a general rule, the longer the chain, the more expensive it will be for Alice.

Another difference with conventional networking is that all our links are one way, because trust isn't generally symmetric, whereas most service contracts are bi-directional. This isn't a problem, because two trust* paths can be found in both directions via a different route of guarantors[1].

After a trust* protocol run, principals may update their commission rates in respect to the outcome of the previous run. In distance vector algorithms, routing tables are normally shared with neighbours so that least-cost routes can be re-calculated. In trust*, the corresponding step would simply be to update the principals who are trusted with their new rate (if applicable). It might be that a deceiving principal will be removed altogether (which corresponds to a link outage) or charged an extortionate rate (which is analogous to network congestion

[1] Review Section 5.2.3 on page 45.

control, see Section 7.9).

In summary, any established network routing protocol will suffice for finding optimal chains of guarantors, although the choice of algorithm may have subtle consequences[2].

5.4 Heterogeneity

In order to implement the trust* relationship mechanism, whether to initiate, provide, or receive a guarantee, a way of making decisions and payments is necessary. One of the advantages of our approach is that both the trust management and payment systems used along a trust* route can be heterogeneous due to the fact that trust (and payments) are confined or localised between directly trusting and trusted principals. If a guarantee has been made from one principal to another, any trust management and payment schemes could be used between them. At the same time, other pairs of principals might use completely different schemes. As long as an agreement has been made in advance on how the protocol will be followed between a specific truster and trustee, then it doesn't matter what is being used along other parts of the chain. This heterogeneity allows users to follow the protocol with more flexibility. For example, by paying each other in a commodity that's of the most value to them.

5.5 Payment and Resource Brokering

As most grids are used to share computational resources (rather than content as in the P2P application chapter) across organisations, these resources could be used as the commodity for forfeit and commission payments. Resources might include CPU cycles, storage or bandwidth. These typically vary in perceived value between the provider and receiver, so resources could also be brokered in this way, converting one resource into another.

Due to the heterogeneous nature of the localised trust between individual pairs of principals, the payments along a trust* chain may be of different types and

[2]For example, a routing algorithm which aims to use minimal resources will take the shortest route.

could be something of a more immediately valuable commodity to them (which may include micro-payments). If a guarantor is taking payments of one type (from a principal they trust) and making payments of another type (to a principal who trusts them), the guarantor is effectively acting as a resource broker between these principals. Users can barter within their local trust relationships to agree on resources that will be shared between them as payment commodities.

5.6 Simulation Implications

The grid simulation is almost identical to the P2P simulation in that a service is provided from one party to another. There are two differences that need to be taken into consideration when simulating the trust* protocol in a grid computing scenario. These are:

- A trust* relationship in the P2P scenario is likely to be between two individuals who are totally unknown to each other who are dealing on a first (and probably only) time basis. The purpose of grid computing is really the same as P2P computing except that resources are being shared rather than content. Also, in grid computing, this might be on an institutional or organisational scale. For example, an agreement to pay for or share computational resources between two universities or companies. The difference from the P2P simulation is that rather than being independent, principals are now representative for the reputation of the institute they belong to. For example, habitual misbehaviour from an individual in a university will affect the reputation of the university as a whole (in the eyes of the trusting institute) and will affect how other members of the university might build trust in the future. By applying this to the trust* simulation, the trust*ed agents can be assigned a domain to which they belong. Agents who act incorrectly might cause an overall price increase for themselves and the rest of their domain.

- When grid services are used between organisations, it's likely that a service agreement or contract will have been written. In comparison to the P2P scenario where connections are fairly ad-hoc and short-lived and where there is no incentive for the server to act correctly, computational agreements are

likely to be between already reputable organisations and span for a longer period of time. The primary difference in this simulation is that Carol now needs to care about her actions and be held accountable for them. For example, Carol has agreed to share her cluster (perhaps for a fee or in exchange for other resources) and will need to pay the forfeit if a claim has been made. Whereas, in P2P networks, peers download files at their own risk and the serving peer typically has no contractual agreement with the downloader and isn't obliged to pay any compensation. To model this, the attribute `carolpaychance` (previously used in the P2P multiple guarantor simulation) defines how likely Carol will pay the forfeit when requested. However, the trust*ee and guarantors will be less tolerant to non-payment.

The results for this simulation reflect the same features as those in the multiple guarantor P2P simulation where Carol is offered the chance to pay the forfeit (refer to Section 4.5). Results for this simulation regarding changes in credit are given in Section A.4. They confirm the results of the previous simulations in that bad behaviour won't be tolerated for long when using trust*. Especially when the service provider refuses to reimburse the forfeit when a SLA might be in place. A bad service provider will quickly lose all possible guarantee routes to them if habitual claims are made and where reimbursement doesn't occur. This is especially important in a grid setting where services are subscribed to and usually paid for in some way.

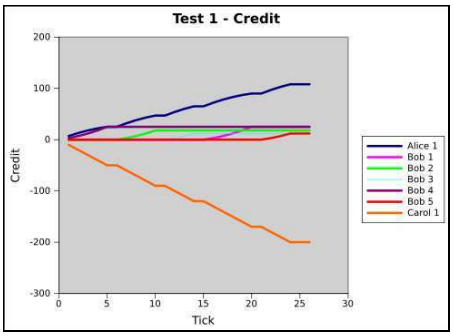


Figure 5.2: Credit values when Carol defaults 100% of the time but always reimburses the forfeit to Bob.

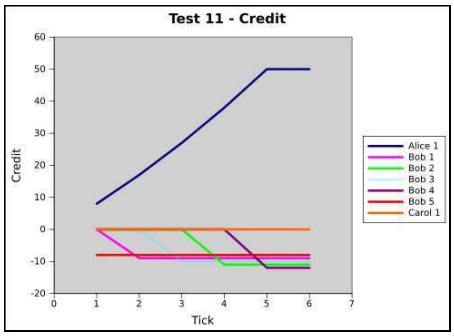


Figure 5.3: Credit values when Carol defaults 100% of the time but never reimburses the forfeit.

5.7 Conclusion

This chapter has described how trust* could be used to extend trust in a grid computing environment. This includes sharing any kind of computing resource across domains, organisations, and countries. In particular, we have discussed the trust mechanisms of a grid middleware (Globe), and how they might integrate with and extend the trust* model. We have also identified ways of providing mechanisms such as fault tolerance and routing.

This chapter has introduced two important features of the trust* model. Firstly, due to all direct trust being local between pairs of principals in a chain, any mechanisms used to follow the trust* protocol are heterogeneous along the chain. For example, the way that trust decisions and payments are made. Secondly, due to this heterogeneous environment, payments can be made by resources that might be shared anyway. However, as particular resources might be of more value to different participants, this allows resource brokering to take place where a resource of one type will be converted into another. Finally, we have discussed the implications of simulating trust* in a grid setting.

Chapter 6

A Click-through Licensing Application

6.1 Introduction

This chapter introduces the idea of applying trust* in order to provide assurances where licence agreements are in place. For example, a “click-through” agreement is commonly found in End-User Licence Agreements (EULAs) during software installation. These types of electronic agreement are increasingly common as more services are being provided digitally. Users can now download software and music without the need to visit a traditional bricks and mortar shop. This chapter discusses the potential benefits of applying trust* to provide assurance that software or music being downloaded has been legally obtained. Other examples of licensing situations where trust* could be beneficial are discussed including online donations and affiliate sponsorship.

6.2 Click-through EULAs

Before software and other digital media such as music was widely distributed over the Internet, vendors would include a licence agreement within the packaged product. The product would be shrink-wrapped before distribution. The agreement needed to be visible through the shrink-wrapping and once an end-user had

bought the product and removed the wrapping, they were deemed to have agreed to the software licence agreement. This arrangement led to many problems concerning the compliance with such an agreement. It could be argued in court that the end-user didn't explicitly agree to the licence. Some software vendors countered this by giving an end-user the option of returning a product within a set period if they didn't assent to the licence agreement. However, due to the increase of software being distributed via electronic mediums such as the Internet, there needed to be a way of allowing end-users to agree to an EULA before actually being able to install and use the software. The problem was solved by showing the EULA as part of the installation process which would only continue with the installation if the user clicked "agree". An electronic EULA or "click-wrap" is now common with most boxed software too. This type of agreement is in place to define how the software can be used and can be legally binding if an end-user breaks the agreement. However, say that an end-user purchases software from a third-party vendor (i.e. not the software producer), they might want assurance that the software has been legally obtained and the licence is legitimate before accepting it. Using trust* with a click-through licence agreement can ease the hassle of compliance by guaranteeing that the software being downloaded has been legally obtained. Here the forfeit would involve the trusted guarantor making the necessary payment to the producer (and claiming the cost of the licence back from the third-party vendor later) and presenting evidence to the end-user that an appropriate licence had been obtained. This way, an end-user can behave as if they know that any of the software they might install is what it is explained to be in its EULA and that it is legitimate. Figure 6.1 shows such a trust* scenario.

Table 6.1 shows a typical protocol for using trust* with click-through EULAs. The first four steps are always performed and continue to steps 5 and 6 in a good case scenario (the licence is legitimate) or skips to the alternate steps 5 and 6 for a bad case scenario. Again, as in Section 3.4.1, steps 1 and 2 could be repeated in order to negotiate a guarantee. Assume that Alice is the downloader (or buyer) of the software, Bob is a guarantor and Carol is the software vendor (who sold Alice the software). Also, David is the software developer or producer.

In a good case, the software that Alice has bought from Carol is a licensed copy and she can safely install the software and accept the accompanying EULA. In

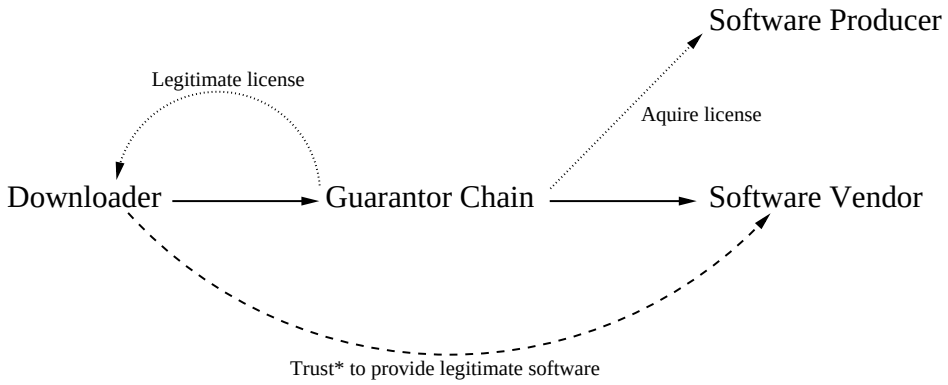


Figure 6.1: An EULA trust* scenario.

	1.	$A \longrightarrow B$:	Can Alice have a guarantee of Carol's software, commission=c
	2.	$B \longrightarrow A$:	Guarantee, id=x
	3.	$A \Longrightarrow C$:	Buy and download software
	4.	$A \longrightarrow D$:	Check the legitimacy of the software
Good	5.	$D \longrightarrow A$:	Software is legitimate
	6.	A:	Install software and accept EULA
Bad	5.	$D \longrightarrow A$:	Software is not legitimate
	6.	$A \longrightarrow B$:	Claim guarantee id=x
	7.	$B \longrightarrow D$:	Buy licence for Alice
	8.	$B \longrightarrow A$:	Legitimate licence for Alice
	9.	A:	Install software and accept EULA
	10.	$B \longrightarrow C$:	Request compensation for licence
	11.	$C \longrightarrow B$:	Payment

Table 6.1: Click-through EULA protocol example.

the bad case, Alice claims on the guarantee from Bob who then buys a legitimate licence directly from David. Alice receives this from Bob and can proceed with the installation. Bob is likely to request compensation from Carol for his losses and whether Bob ever guarantees her again depends on if she pays. Bob would have charged Alice for the guarantee at the cost of c and if he is sure that Carol's software is legally obtained, it is likely that Bob will make a small profit. If Carol begins to supply illegal or unlicensed software, Bob is unlikely to guarantee her for much longer.

A possible problem could be that Alice makes false claims. However, this is

unlikely as she has nothing to gain from doing so as she has already paid for the software and would only receive a second licence. It is more likely that Bob could forge a licence rather than buying one from David and keeping the commission. If Alice is suspicious of this happening, she could simply repeat step 4 with the provided licence. It is assumed that a software producer like David will be happy to perform these checks for those who want to ensure their licence is legitimate before accepting.

6.3 Music Downloads

Trust* could similarly be used with a click-through licence agreement when downloading music to be sure not only that it has been legally obtained, but also that the artist actually receives the royalties they are due. For example, it might be in the interest of an artist's fan-base to ensure that this happens. Trust* could be used to ensure that a music vendor (iTunes for example) will actually pass on the 30 pence (or whatever was agreed) to the artist. If they can't prove that they did, then the guarantor will pay the artist, prove to the end-user that they did, and claim the payment back from the third-party vendor later. This way the artist will always receive their royalties. A possible privacy issue is that by proving the money was paid for a specific individual's purchase, that individual's identity might be divulged to the recording company or artist. Various payment protocols address this, for example, anonymous payments which include a client challenge. Examples of this are discussed later in this chapter.

6.4 Donations and Sponsorship

Suppose that a website is hosting a link claiming that 1p will be donated to a charity for every click made. An individual clicking the link might want some assurance that the intended charity does actually receive this donation. Here the forfeit would be for the guarantor to produce a receipt showing that the donation has been made, possibly by the guarantor. This is an example where using trust* can ensure that someone will always be held liable for making these types of

payment. Figure 6.2 shows this scenario.

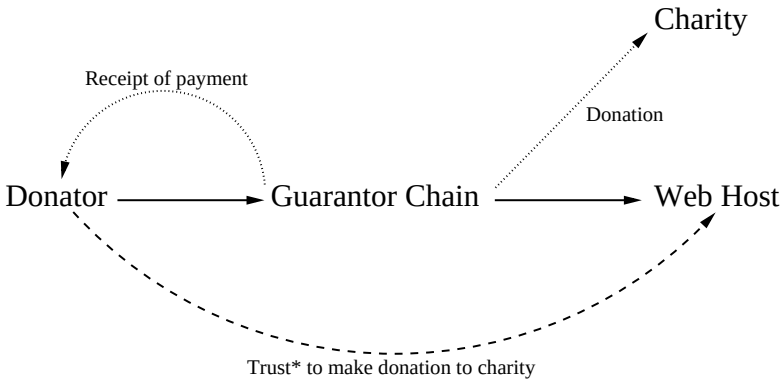


Figure 6.2: A charity donation trust* scenario.

There are other examples to which this scenario could be applied. These include sponsored links such as those provided by services such as Google AdWords. These services allow businesses to bid for particular keywords which relate to the products or services that they would like to advertise. If a user enters any such keywords in their Google search, sponsored links will be displayed above the organic search results. Popular keywords and search terms are more expensive for the advertiser and payment of this type will be made to Google each time an advertiser's sponsored link is clicked. Also, higher bids for keywords will affect the frequency with which a link will be displayed. Trust* could be used to ensure that sponsored link hosts are paid correctly in relation to their cost-per-click agreement and the actual click-through rate that their site encountered. Conversely, the advertiser might suspect that the click-through rate was less than the host is claiming. An example scenario could be two small e-businesses that sell similar products (not the same) and are likely to have the same customer base. It might be beneficial for both businesses to form an affiliation with each other and provide links to each other's site. Without requiring a trusted infrastructure such as Google AdWords, a trust* relationship between the two businesses can be used to ensure that they are honest about the traffic their links have received. Services such as Google Analytics can provide detailed information about a website's traffic which could be used to provide evidence of click-through rates.

6.5 Micro-payments

As described above, trust* can be utilised in situations where an agreement has been made whether it be an EULA, a donation, a sponsorship or another small payment to ensure that all parties involved are compliant with the agreement. Using micro-payments to enforce this lends itself to these types of business model as the payments themselves are going to be very small and will likely be of the same currency. The commission and forfeits (licence fees, royalties, donations, etc) can also be transferred using micro-payments. This section gives a brief overview of the types of micro-payment mechanisms that could be used in a click-through trust* setting.

A survey [105] of all types of electronic payments systems analyses various criteria regarding eleven chosen micro-payment systems. One of the first micro-payment systems was Millicent [45] which allows small asynchronous payments to be made. However, Millicent provides no anonymity to its clients. Many of today's electronic payment systems make use of mechanisms proposed by Chaum *et al*. For example, the electronic "cash" system in [24], which was later improved in [25] allows payments to be made off-line (with no connection to the bank). Other work by Chaum focuses on the anonymity and untraceability of electronic transactions [21, 22, 23]. For example, "blinded" payments in [21] make it hard for a bank to link payments from the same client. This is achieved by multiplying a serial number by a secret "blinding factor" known only to the payer before sending it to the bank for signing. When a signed serial number is returned, the payer can divide the result by the blinding factor to reveal the signed serial number.

Most of the time, trust* payments are confined within a local trust relationship and so issues such as anonymity might not be a problem. However, there are cases where principals who are using trust* need to make payments to principals outside of their local trust relationships. For example, in the applications described in this chapter, where a payment needs to be made to the developer to obtain a licence. Also, to prove to the end-user that a legitimate licence has been purchased, the guarantor would provide an electronic receipt along with the licence (the receipt could be bound to a specific licence). The receipt of payment can be checked by the end-user (e.g. by verifying a digital signature) and the licence can be checked

by contacting the developer directly. The receipt is also cryptographic evidence needed to claim the licence fee back from the original vendor (the trust*ee).

There have also been proposals for micro-payment schemes which use trust management techniques to encode the necessary payment credentials. For example, Blaze *et al* [16] use the KeyNote trust management system to enable an electronic equivalent to bankers cheques. Their solution was successfully implemented in the form of a drinks machine which accepted KeyNote micro-cheques (signed by a trusted bank) from an electronic device such as a PDA.

In [40] and [41], the KeyNote payment method is used to reward clients in a distributed computation platform called WebCom [79] for successfully completed operations. Also, clients can pay servers in return for service usage. An example of KeyNote micro-payments developed in the course of this research for use with trust* is given towards the end of Appendix B.

6.6 Simulation Implications

This section discusses the comparison of simulating the EULA click-through application using Repast Simphony to previous applications described in this dissertation. The model is very similar to the one described in Chapter 4 which forms the basis of the click-through simulation. Applying trust* to music downloads, sponsorship or donations would be the same apart from the roles each agent takes. Below, changes such as these and the implications of simulating a click-through scenario (in this case, EULAs) is described.

- In this simulation, the Alice agent acts as the software buyer, Bob is still a guarantor and Carol is the software vendor. This simulation introduces a new agent called David who is the software producer. Trust still travels in the same direction as it did in the P2P simulation (i.e. from Alice to Bob, and from Bob to Carol) although, in this case, Bob doesn't ever need to pay a forfeit to Alice. In the click-through licensing scenario described above, it is the software producer (David) who receives the payment from Bob and Alice only receives the obtained licence.

- Although the new agent (David) has been introduced to the simulation, he

needn't actually be part of any trust* relationships. However, he is still an actor in the protocol and might even be paid a commission. For example, he will answer queries (from Alice) with regards to the validity of a licence and will receive payments (from Bob) for a genuine licence if required. It is worth noting that David could be greedy and take payments for licences even if Alice's current licence is valid. To simulate this, David has a `truthchance` attribute which is similar to the attribute of the same name that Alice had in the P2P simulation. In David's case, this defines the probability that he will be truthful when replying to Alice's queries.

- Alice could also make false claims, however, she has nothing to gain from doing so as she has already purchased the software and won't be the receiver of any monetary payments. When Bob comes to claim compensation from Carol, she is likely to dispute this and prove that the licence was in fact legitimate. Bob could stop providing guarantees to Alice or charge enough to cover any potential losses. Bob has this option in the simulation to increase his charge if he finds (through an investigation possibly with Carol and David's input) that Alice has falsely claimed. Otherwise, there is no reason for Bob to increase prices for Alice but will more likely re-consider the status of his trust in Carol.

Results for this simulation regarding changes in credit are given in Section A.5. The results back-up the previous simulation findings in that trust* usage will be short-lived for non-compliance. This is evident for the current types of agent we have already seen (the trust*er, the trust*ee, and the guarantors). However, the effects of the software producer's behaviour can be seen. For example, if David habitually lies about the validity of a licence, he is effectively causing the guarantor chain to break (or become too expensive) leading to the vendor. If principals wishing to use trust* to validate their software receive false negatives from David, they will either stop buying software from Carol or won't care about the legitimacy of their software. Also, as it might be too expensive to find a guarantor route to Carol (as she'll effectively be receiving the blame for an illegitimate licence), she'll probably cease selling David's software in the future.

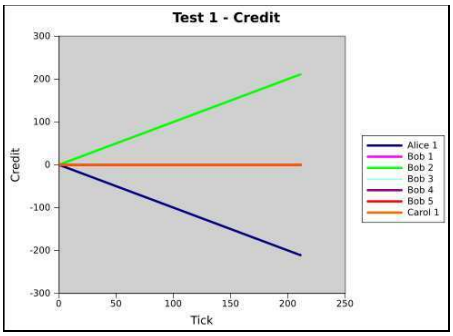


Figure 6.3: Credit values when Carol never defaults and David is always truthful.

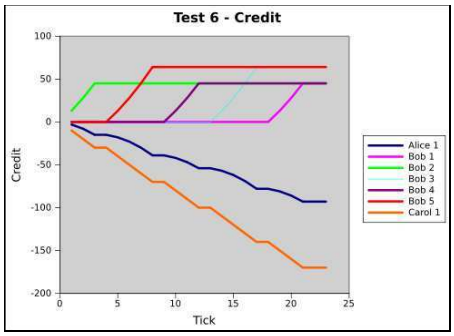


Figure 6.4: Credit values when Carol never defaults, however, David is never truthful.

6.7 Conclusion

In situations where licensing issues exist such as those described in this chapter, mechanisms based on trust* can be used to provide an effective way of minimising the cost and hassle of compliance between the involved parties. Also, applying trust* in this way can be viewed as a self-enforcing protocol in that no advantage can be made by cheating the system. For example, it's in the interest of all agents in this scenario to act correctly, otherwise they will only be burdening themselves. For example, following from the points made in this chapter, Alice has no reason to claim a guarantee if she already has a legitimate licence. Perhaps she could sell the new licence to someone else, however, Bob's premium will eventually out-weigh any profit Alice might make especially if habitual claims are made. Carol has the incentive to provide genuine licences in the first place as she will need to reimburse Bob if they aren't (otherwise risk losing his trust and hence her connections). David also has the incentive to answer queries correctly otherwise it may cause the effects described above (premium increase for Alice or removal of local trust to Carol) due to the effects made to Alice's claim frequency. As a software producer, David will be affected in the long run as Carol might stop selling his software or Alice might stop caring if her licences are legitimate or not.

Chapter 7

A Spam-proof Email Application

7.1 Introduction

This chapter shows how trust* can be used to deter principals from sending unwanted or "spam" email. Spam email is responsible for a high percentage of traffic on the Internet and is an annoyance to end-users. Applying trust* in this way, it is hoped, will lower this traffic by deterring mass spammers by making it an unfeasible business model for them[1]. In order for a principal to send an email to another, a trust* route needs to be found between the sender and the receiver in order to guarantee that the email won't be spam. A counter argument is that email could be charged for in the first place in a similar way to the postal service where "stamps" would need to bought in order to send email. However, the point of trust* is to avoid the need for this type of universal up-front payment. Moreover, trust* reduces the cost for a genuine email user. Although, up-front payments could be used as a way of bootstrapping new users into the system who don't yet have any trust routes.

This chapter begins with an overview of current spam prevention techniques and a discussion of the types of email user and perceptions of spam email. This is followed by the application of trust* and descriptions of the "spam-proof" protocol. Finally, the implications of simulating and implementing such a solution is described.

[1]Or, conversely, allowing the recipient to earn a comfortable living by reading spam email.

7.2 Spam Prevention Techniques

Unsolicited email has been around since the use of the Internet became widespread. Therefore, much work has been done to find ways of filtering and preventing spam email. This section gives an overview of the previous and current solutions to the problem of spam email.

The most common solution is to filter email either at the mail server or at the client. The problem with filtering mail at the server is that spam perceptions of the host might be different to those of the end-user. Therefore, servers tend to flag emails that they think might be spam and allow the user to decide whether to delete it or not. These flags can be used to aid a filter built into an email client such as Mozilla Thunderbird or Microsoft Outlook. Client filters tend to be more configurable, and some can learn about what the user considers to be spam from previous emails and filter incoming mail accordingly.

Mail servers and clients can also maintain whitelists and blacklists of domains or other servers that might be considered senders of spam email. A whitelist is a list of trusted email senders whose email should never be considered spam. The opposite is a blacklist where known spam senders and relays are logged for future reference when filtering spam (and mail from such senders is always considered spam). This is a good solution in most cases where a mail server blocks emails from a known spam relay. However, some institutions and companies have found that their domain has been unduly blocked from certain mail servers. This might be due to an account hijack or spoofed email headers of course, but could be an employee sending copious amounts of unsolicited email from their work account. Hosts of blocked servers often need to prove that their problem has been rectified before being unblocked by the blocking server. Updating this list can be a time consuming and never ending task, so alternatively, mail servers might hold a policy whereby any email from unknown senders will initially be bounced and added to a greylist [50]. Once the sender attempts to send the email again, the message will be delivered. This works on the idea that it will be too costly for mass spammers to resend an email for every bounced attempt (until the spammers catch on at least).

Similarly, a client could hold a queue of messages from unknown senders

(perhaps that aren't in a whitelist or address book) and to send a challenge back to them before fully delivering the mail. The challenge will be something like a Turing test which will ensure that their message is delivered if answered correctly in a reply. For each sender, this should usually only need to be done once, and afterwards they will be added to a whitelist.

Challenge/response protocols such as those above ensure that email is being sent from a real person (and with their consent) and it is unlikely that spammers will resend an email or have time to reply to a challenge. However, challenge/response filters are not widely used by email users. This is due to users ignoring such challenges or mis-identifying them as spam and therefore causing mail to remain undelivered. A similar method proposed in [47] shows that only moderately intrusive techniques are enough to stop outgoing spam from free email providers such as Hotmail. According to [47], the cost of account creation (completing a Turing test) can be amortised by sending 1000 spam emails at the cost of 0.002 cents per message and average earnings of 0.01 cents per message sent. Their solution works by making users pay some cost such as a difficult computation after every 100 messages sent. Legitimate users will only need to perform this 10 times to prove that they are legitimate users. This small cost won't affect legitimate users but will affect spammers as their cost-per-message will become more than their earnings.

Reputation systems can also be used as a way of reporting and filtering spam email [46, 113]. The reputation of a specific user will define the weight that a spam report carries from them. Also, trust perceptions and reputations could be shared between mail servers to aid filtering [75]. However, these all suffer from the same problems regarding reputation systems discussed previously in this dissertation.

Another method for reducing the amount of unnecessary and unsolicited email is to charge for postage "stamps". This works in the same way as traditional mail but users need to buy digital tokens or tickets in order to send an email. Abadi *et al* [7] propose a service which provides tickets that allow sending of email. The service also maintains the number of tickets a particular user has in stock. A receiver of an email can use the service to validate the token which can be refunded if the email wasn't spam. Apart from deployment issues such as gaining mass user acceptance, the trouble with this proposal is that a universally trusted

ticket service will be required for the concept to take-off. A standard mechanism would need to be used by all major email service providers (in a very competitive market, e.g. between GMail, Hotmail and AOL etc), otherwise email users would become distressed with acquiring the correct tickets for the various providers.

Schlegel and Vaudenay designed a system called XToken [95] which allows users to monitor how their email addresses are used. It involves passing tokens with every message which can be validated before reading. There are different types of token, some which don't expire until they are revoked and some which expire after a specified number of uses or date. A system similar to XToken is desirable because no changes need to be made to the email infrastructure for it to work. With XToken, the tokens are distributed to all friends and associates in advance and later included with a message when needed. The policy information for each issued token is stored locally by the receiving user for validation purposes. Ioannidis designed a system which is similar but encodes the policy for each token within the email address itself [55]. This means that no information needs to be stored locally but requires changes to the email infrastructure.

A peer-to-peer payment system called *i*-WAT [94] can be used to counterbalance a loss from receiving a spam email. It works by charging an email sender 1*MU* (Mail Unit) to send a message and works on the assumption that a healthy email relationship between two users will evenly balance the *MU*s between them. As most users are unlikely to reply to unsolicited mail, the sender will never recoup their spent *MU*s.

A corresponding real-world example is a door-bell system that was designed to stop unsolicited callers disturbing a household [97]. The door bell is activated by inserting a low value coin which upon answering is refunded if the caller is welcome, otherwise it is kept. This design has various flaws in the real-world, but the idea might be better suited to deterring spammers in the cyberworld. Although the coin value is low, to call at hundreds of houses would soon add up.

Most spam email is just an inconvenience to the receiver and an added cost to networks such as the Internet. However, email senders exist that have malicious intent. The most obvious attack is to send malware via spurious attachments. More commonly these days, scams such as email phishing [30] can cause problems including monetary loss and identity theft. A study by Jakobsson *et al* [56]

shows that many of the subjects (ages ranging from 18–60 excluding anyone with a computer science background) were fairly informed when it came to identifying indications of a phishing email or website. However, the study had only 17 participants which probably isn't enough to reflect the overall reactions of the general public receiving this type of email. When considering that the scammer needs to send millions of emails to get a handful of responses, there will always be a chance of success as there will almost always be someone who is fooled by a phishing email or website.

7.3 Perception of Spam Email

Email is a widely used medium on the Internet and most users have at least one email account whether it be from an ISP, place of work, or a free account[2]. People use their email accounts for different reasons. For example, a work address should be used for professional reasons and maybe a free Hotmail account could be used for personal and social reasons. Therefore, tolerances to spam email might differ on an account-to-account basis.

Email addresses are shared to other users, companies and organisations which are consequently stored in many places. For example, many websites require you to validate an account via email and later use your address to log on to their service. Also, most sites require you to "opt-out" from subscribing to their mailing list which might be used to advertise new products or services in the future. To the average user, this might not be obvious.

For these reasons (and those in the previous section), the longer an address has been used, the higher the quantity of spam email it is likely to receive.

The problem with email is that every end-user might have a different perception of what constitutes as being "spam". This problem confronts spam filtering software in deciding what might be spam or not. Instead of the software deleting everything that it considers to be spam, it is likely to file them away for further inspection by the user. Spam filters can aid the user as they lower the number of emails that need to be checked manually.

[2]Such as Gmail, Hotmail or Yahoo etc.

Using spam filter software on the client side is a user's choice but many email hosts now do a check on the server side before the emails are actually downloaded. The host will then flag the messages that might be spam to the end-user. An end-user's perception of whether the flagged messages are actually spam might be different to the perception of the company or organisation hosting the mail server.

Some users might consider commercial email to be spam even if they haven't opted out of receiving it (the same applies to receiving a lot of internal memoranda). However, spam is generally considered to be unsolicited adverts for illegal software, counterfeit watches, and drugs etc. Also, other scams such as phishing and email that might contain malware. Using trust* can make the job of deciding whether email is spam easier as a spam filter now can check such emails for a valid guarantee.

7.4 Reverse Routing for Trust*

In previous applications, trust* routes have been built from the source (the trust*er) to the destination (the trust*ee). In the spam-proof application, the trust* route must be built from the destination end. This is done by maintaining different routing tables depending on the direction that a trust* relationship needs to be built. These are:

Forward where a routing table of *trusted* principals is maintained. This table is created and edited by a principal in relation to whom they trust and by how much. In this case, a trust* route goes in the same direction as direct trust does. The previous applications of trust* described in this dissertation are all examples of forward trust* relationships. This is usually invoked by a client in a networked application.

Reverse where a routing table of *trusting* principals is maintained. Principals build this table by receiving information from their direct trusters. In this case, a trust* route goes in the opposite direction to that of direct trust. The spam-proof application is an example of a reverse trust* relationship and is usually invoked by a server[3].

[3]Reverse routing could also be used in other applications such as click-through when commis-

7.5 The Spam-proof Protocol

The protocol in Table 7.1 shows how trust* would work in the spam-proof email application. It involves three principals with one path of delegation. Carol (the trust*ee) wants to email Alice (the trust*er); Bob trusts Carol and Alice trusts Bob. Note that for this example, for the first time, the commission payments and the email itself, go in the opposite direction to the direction of trust[4].

1. $C \longrightarrow B$: Can Carol have a token for Alice, forfeit=t, commission=c
2. $B \longrightarrow A$: Can Carol have a token for Alice, forfeit=t', commission=c'
3. $A \longrightarrow B$: Token for email from Carol to Alice, id=x etc
4. $B \longrightarrow C$: Token for email from Carol to Alice, id=x etc
5. $C \Longrightarrow A$: Email (token x in header)
6. $A \longrightarrow B$: Token x is OK/spam
7. $B \longrightarrow A$: Ack/here is the forfeit
8. $B \longrightarrow C$: Token x is OK/spam
9. $C \longrightarrow B$: Ack/here is the forfeit

Table 7.1: Spam-proof protocol example.

1. Carol sends a request to a principal who trusts her (Bob in this case) for a token to send an email to Alice. A forfeit and commission offer are also sent.

2. Assuming Bob is happy with the t and c values, he forwards this request to a principal who trusts him. Alice will receive this request with Bob's forfeit and commission offer.

3. Alice checks the t' and c' values from Bob and generates a token which is sent back to Bob.

4. Bob forwards the token to Carol. Bob now knows that a chain has been made between Carol and Alice via himself and that his guarantee is active.

5. Carol can now email Alice directly with the token embedded within the email header.

sion is paid by the trust*ee.

[4] i.e. in the opposite direction to the previous applications in this dissertation.

6. Alice (or her email filter) checks that the token is valid before reading the email. She decides whether the email is spam or not and sends a response to Bob either way.

7. Depending on the response, Bob will either send an acknowledgement or pay the forfeit to Alice. Bob now knows that this token has been used by Carol.

8. Bob informs Carol of the decision.

9. Depending on the response, Carol acknowledges this or pays the forfeit to Bob. In a longer chain, this process may continue; as usual, all forfeit and trust updates are local.

7.6 Pricing Strategies

There are likely to be some legitimate senders who can't find a trust route to the intended recipient. This might be because they haven't yet built any trust relationships or simply that a chain of guarantors cannot be found.

A new user can bootstrap a trust relationship by paying a fee directly to the receiver or to a guarantor. For example, we assume that the message; $C \longrightarrow A$: Please may Carol have a token for Alice, forfeit=0, commission=.10 will always work[5]. Now the real spammers need to find a cheaper route based on this.

Eventually, assuming Carol doesn't send spam, Alice might begin to trust her. This will enable the possibility of Carol buying tokens from Alice for routes to recipients that are beyond Alice.

Most payments should be of a very low value, sufficient to deter habitual spammers and not affect ordinary email users. Having to occasionally pay a 0.5 pence forfeit is a very different proposition to paying millions of forfeits.

In the protocol, values for c and t are proposed when requesting a token (steps 1 and 2 in this example). It might be that the receiver of such a request isn't happy with the values offered. In this case, a negotiation of these values might take place.

[5]As the receiver or guarantor is making 10 pence either way. It is the sender's incentive to not send spam to earn trust and enjoy lower premiums in the future.

Bob could simply reject the request and leave it to Carol to re-submit the request with different values. Or Bob could explicitly state the values he requires to provide the guarantee. However, this is most likely to happen after Bob estimates the risk involved and his possible losses so we would expect that the forfeit he might need to pay Alice is always lower than the forfeit Carol would pay him.

7.7 Security Requirements

The tokens that are used to allow a trust*ee to send an email could be cryptographically protected. This might be to prevent forgery of a token or for verification of its origin. Also, a digital signature can be appended to the end of a token. Local verification of this signature could also take place before checking the actual guarantee. However, this isn't strictly necessary as each user will keep a record of tokens they have generated and locally distributed which are still active. Because the protocol requires a response message is to be sent either way (steps 6 and 8), a user always knows whether a token is still active or not and can limit their exposure. Once a token has been received, the id number can be cross-referenced with the generators table to retrieve information such as who the guarantor is and the agreed forfeit etc. Hence, no details of the underlying guarantee need to be encoded within the token[6] and thus forged or already spent tokens won't work. This method also allows the possibility of issuing "bags" of tokens to more highly trusted users. These could be set to expire if they aren't used by a particular time but will lower the overall computational expense of repeating the protocol for each email that needs to be sent.

Also, some privacy is maintained as Carol can't see the value of the forfeit that Bob might need to pay Alice if she defaults[7]. Users could be identified by anonymous keys if they wish to maintain further privacy when using trust*. However, up-front payments need to be protected from forgery (and double spending etc) and immediately claimable by the recipient. Also, this provides Alice with a way for allowing anonymous email without risking spam.

[6] Although the tokens could carry the state cryptographically.

[7] Carol doesn't need to know who the guarantor providing the guarantee to Alice is, especially in a longer chain.

7.8 Bad Scenarios

After the email has been sent, Alice decides whether the email is spam or not. Even though her perception of what is considered spam is likely to be different to Carol's and Bob's perception, Bob is still obliged to honour the guarantee forfeit payment. And the same for Carol if Bob claims the message was spam. There are some issues that may arise because of this.

7.8.1 False Claims

A false claim might be made by the receiver (Alice) where she might claim a forfeit even if she doesn't regard the email to be spam. Again, Bob would need to honour this. Also, even if Alice doesn't claim, Bob could falsely claim the forfeit payment from Carol. Note that Bob trusts Carol but not conversely so if Carol suspects that this might be happening regularly, she could negotiate a cycle of trust* (as described in Section 5.2.3) to Alice or Bob to insure her against this. This attack could also be blocked by requiring Bob to prove to Carol (directly or via the cycle) that a claim has been made by Alice and that he has paid Alice the forfeit. This could be in the form of a cryptographic micro-payment receipt. For example, assume that Carol has found a cycle of trust* to Bob via David whom she trusts. David trusts Alice or Bob to not make false claims. If Carol is suspicious, she can alert David who will investigate whether false claims have been made (by requesting proof of payment from Alice or Bob) and will compensate Carol if necessary. David will also charge a small commission for providing this service however it will prevent Carol suffering greater loss from repeated false forfeit claims.

7.8.2 Non-payments

Of course, Carol could refuse to pay the forfeit. Due to the payments being low in value, this won't affect Bob on a one-off basis. However, if regular non-payments occur, Bob is likely to stop trusting Carol altogether. Hence, Bob will no longer provide email tokens (or guarantees) for Carol or the price might go up (see the following section).

If Bob fails to pay Alice, she will simply stop trusting him and will cease accepting guarantees from him. The problem of non-payment of a forfeit or commission is a local problem which can be solved using local trust management mechanisms.

7.9 Congestion Control

The effects of false claims and non-payments in the trust* model are analogous to some of the techniques used to control congestion of packets in data networks. An example of such a technique is network back-pressure.

Paraphrased from Stallings [101, p384], back-pressure produces an effect similar to that in fluids flowing down a pipe. When the end of the pipe is restricted, the pressure backs up to the point of origin where the flow is stopped or slowed down. This technique can be selectively applied to logical connections in a network, so that the flow from one node to another can be restricted or halted. This restriction propagates back to the source of the connection.

Analogously, if trust* is regularly broken between two principals, the guarantor is likely to either break the local trust completely (never provide guarantees again) with the principal being guaranteed (which corresponds to a link outage) or steadily increase their commission rates (which corresponds to a price increase, or a delay). If a particular link drops between two nodes, a route which previously utilised this link might become more expensive for surrounding nodes. This is likely to cause a bottleneck for other nodes following alternative routes and further increasing their cost. These issues can be explicitly addressed using standard network congestion control techniques such as credit based congestion control [70, 71, 90].

Credit based congestion control is a scheme based on providing an explicit credit to a node. The credit indicates the number of packets that the source node may transmit. Nodes have to wait for additional credit if they run out. This is also analogous to the commission and forfeit rates used in the trust* model except the credit is a node's reputation (or risk) rating (which affect the commission and forfeit rates that will be charged).

If Carol repeatedly defaults the guarantee (by sending spam), Alice is likely

to increase the forfeit she requires from the guarantor (Bob). This will then propagate to Carol as Bob is likely to increase his rates accordingly. Other routes that Carol might take to reach Alice could become congested and therefore might become expensive too. Just as in the networking case, congestion tends to spread and will interact with adaptive routing decisions (and this can be addressed with standard counter-measures e.g. hold-down time [102, p92]).

7.10 Simulation Implications

This section discusses the key enhancements required to simulate the spam-proof application using Repast Simphony in comparison to previous applications in this dissertation. The model described in Chapter 4 forms the basis of the spam-proof simulation. Below, the major changes and their implications are described.

- In the spam-proof simulation, it is important to note that the trust* protocol is invoked in the opposite direction to trust as mentioned earlier in this chapter (also see Section 5.3.1). In the P2P simulation, Alice trusted Bob who trusted Carol not to serve incorrect files. In this case, Carol is trusted by Bob who is trusted by Alice not to send spam email. Due to this, the commission payments are also made in the opposite direction to trust. For the email application, the destination plays the role of the client (the trust*er), and the sender plays the role of the server (the trust*ee). It is now the server who invokes the trust* protocol rather than the client in the P2P application.

- The agent types are the same except that Carol is now the email sender and Alice is the email recipient. The multiple Bob agents still act as guarantors. Also, the properties that each agent holds are the same, however, the default values for the forfeits will be the opposite compared to the default values in the P2P simulation. This is due to the fact that the server is now the invoker of the protocol. Commission payments go in the opposite direction as trust now (see Chapter 8 for more on this).

- In this simulation, there are five global attributes that can alter a simulation run rather than the two in the P2P simulation (`truthchance` and

`malwarechance`). The first is `spamlevel` which is an integer value between 1 and 5 which defines the type of spam that Carol will send (if she does). Level 1 can be seen as very low level spam email such as advertising or chain mail. This ranges up to level 5 which might be a scam, fraudulent or malicious email. Also, advertising of drugs, software or jewellery.

- The second is `spamaccept` which defines Alice's tolerance of spam email. This relates to the `spamlevel` that Carol sends. For example, if `spamaccept` is set to 2, an email will only be considered as spam if its `spamlevel` is 3 or more. This feature reflects real life perceptions of what is considered spam and is intended to add some realism to the simulation.

- The third is `spamchance` which is similar to the `malwarechance` attribute in the P2P simulation except it defines the probability that Carol will send an email at her `spamlevel`.

- The fourth and fifth are `rectruthchance` and `guartruthchance` which define the probability that the receiver or a guarantor (respectively) will be truthful when making a claim. For example if the `rectruthchance` value is low, Alice is likely to claim even if Carol hasn't sent an email that she considers as spam. A low `guartruthchance` value, will cause a guarantor to claim from Carol (or another guarantor in a chain) even if Alice hasn't made a claim.

- Finally, due to the trust* protocol being invoked in the opposite direction (by the trust*ee), the spam-proof protocol requires an acknowledgement message to be sent even if no claim has been made for a particular email (see steps 7 and 9 of the spam-proof protocol). This is to inform a guarantor of the outcome of a guarantee if a claim hasn't been made (and hence, a forfeit doesn't need paying). The `ack()` method provides this behaviour. After a guarantor receives this message about a specific guarantee, they can mark its status as complete.

Results for this simulation regarding changes in credit are given in Section A.6. Again, the results correlate with previous application simulations. Habitually

sending spam or being untruthful about claims will only damage the chances of the perpetrator forming trust* relationships in the future either by losing direct trust locally or by routes becoming too expensive. This is the whole idea of using trust* with email — to remove routes for habitual spammers — or to be paid a decent amount to receive spam. In reality, peoples perceptions of spam vary and claims should be honoured by a guarantor. Investigations could be made regarding false claims if they are suspected but this is again a local problem. Cycles of trust* can be built to mitigate the effect of false claims outside of a local relationship.

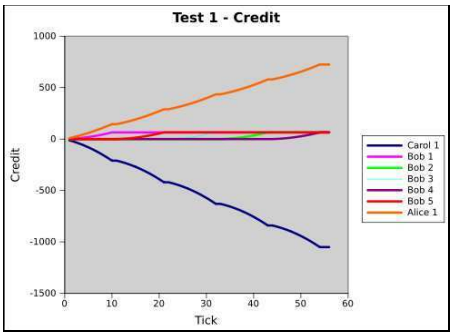


Figure 7.1: Credit values when Carol sends spam email 100% of the time where Bob and Alice are always truthful.

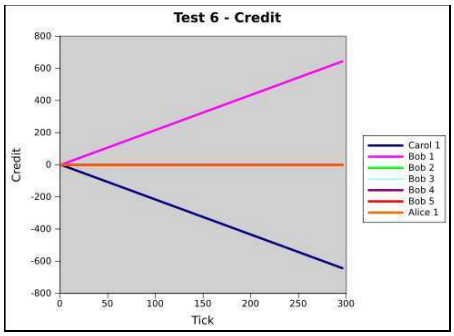


Figure 7.2: Credit values when Carol never sends spam email where Bob and Alice are always truthful.

Appendix B is a walk-through example of a spam-proof protocol run implemented using the KeyNote trust management toolkit. It follows the approach given in this chapter, however, a way of making micro-payments with KeyNote is also included. The spam-proof application including payments is implemented purely using KeyNote in this example and with each pair of principals using the same mechanisms. In reality, this in unlikely to be the case as individual pairs are likely to be using heterogeneous mechanisms.

7.11 Conclusion

Applying trust* to the sending of email differs from applications previously described in this dissertation which are used to promise that a service will be provided. With email, trust* is used to promise that an email isn't spam. The intention

of using trust* to send email is not to penalise users that might send the occasional unsolicited email but the spammers who aim to profit from sending this type of email.

Due to the format of email addresses being simply `user@host`, it is easy to generate a dictionary of usernames for a specific host. Many email servers also use aliases to hide the underlying username of an account and to provide a more human readable and memorable prefix to the address. However, the underlying username is usually a combination of characters generated by the host which can still be used as the prefix to an address. For this reason, it is easy for a mass spammer to generate lists of addresses. Another way to gather addresses would be to buy (or steal) databases of addresses from other organisations.

Using trust* to provide guaranteed emails aims to deter these types of spammers. For example, the "mass spammers" that purposely gather and generate millions of addresses to send email to. Mass spammers rely on sending millions of emails a day to make any respectable profit as only a very small percentage of recipients will actually respond[8]. It would be non-viable for them to do this if even low-value guarantees were required.

Legitimate mailers might unknowingly forward a spam email to another or send an email that might be considered as spam. Trust* deters habitual spammers and won't be costly for the average email user. However, trust* could be extended to unsolicited or unwanted email rather than just typical spam email. Deploying trust* internally for email services in specific companies, universities or organisations could be effective before deploying it gains global acceptance.

For the trust* solution to work, email users participating in the use of guaranteed emails are required to filter emails without guarantees. However, existing spam filter applications can be used for this. The more users who do this, the more effective applying trust* will be in stopping routes for mass spammers. Eventually, habitual spammers won't be able to find any free routes to send emails and email users will only need to read emails with valid guarantee tokens.

People who aren't yet using trust* are likely to still send emails to people who are without realising that they need a guarantee to do so. However, if they adopt trust*, there is a higher chance of these emails being read, which will act as an

[8] Also, addresses might be non-existent if they have been randomly generated.

incentive for uptake to spread. Conversely, people will have an incentive to read trust* certified email, as they will get paid if it's spam.

Chapter 8

Full Description of the Trust* Model

8.1 Introduction

This chapter recapitulates the concepts and features of the trust* model that have been introduced and discussed so far. This dissertation has introduced new parts of the model as required by specific application scenarios, however, this chapter provides a full description of the trust* model and discusses further some of the issues raised.

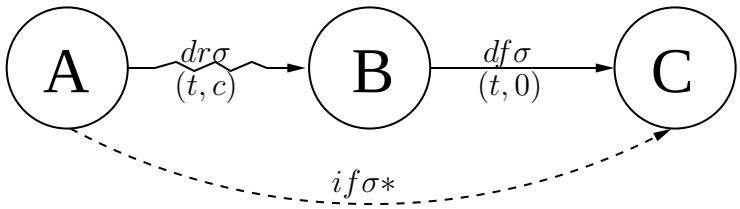
8.2 Trust* Notation

In order to formally describe the trust* model and examples of trust* relationships, a variation of Jøsang's notation in [62] is adopted. This section gives an overview of the notation that is used in this chapter. The top division of table 8.1 is some of the notation from [62] with the lower division being the additional constructs required by the trust* model.

Symbol	Meaning
A, C	End-points of a trust* relationship.
$B^{1..n}$	Guarantors in a trust* chain.
:	Connection of trust arcs.
σ	Trust scope.
f	Functional variant of trust.
r	Referral variant of trust.
d	Direct trust.
i	Indirect trust.
$*$	Represents a trust* relationship.
t	The agreed forfeit.
c	The agreed commission.
$\rightarrow$	Direct functional trust.
$\rightsquigarrow$	Direct referral trust.
$\dashrightarrow$	Indirect functional trust.

Table 8.1: Trust* notation.

Take for example, a trust* relationship between A and C ($[A, C; if\sigma; *]$). This can be expressed diagrammatically as:



The same can be expressed symbolically as:

$$A \underset{(t,c)}{\overset{\sigma}{\rightsquigarrow}} B \underset{(t,0)}{\overset{\sigma}{\rightarrow}} C \Rightarrow A \overset{\sigma*}{\dashrightarrow} C \tag{8.1}$$

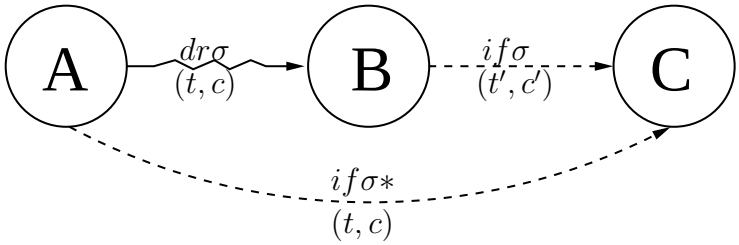
Finally, the same trust* relationship can be expressed in the adapted version of Jøsang's notation:

$$([A, C; if\sigma; *]) = ([A, B; dr\sigma; (t, c)] : [B, C; df\sigma; (t, 0)]) \tag{8.2}$$

The examples above all represent the case where A has indirect functional (if) trust in C because she has direct referral (dr) trust in B and B in turn has direct functional (df) trust in C. The scope (σ) might be defined to mean "trust to provide

legitimate licences with software". The (t, c) represent the locally agreed forfeit and commission rates regarding the guarantees that make up a trust* relationship where t and c can be equal to 0 where they aren't applicable.

Similarly, extending a trust* chain to an arbitrary number of hops can be expressed diagrammatically as:



Again, where the same can be expressed symbolically as:

$$A \underset{(t,c)}{\overset{\sigma}{\rightsquigarrow}} B \underset{(t',c')}{\overset{\sigma*}{\dashrightarrow}} C \Rightarrow A \underset{(t,c)}{\overset{\sigma*}{\dashrightarrow}} C \tag{8.3}$$

And in the adapted version of Jøsang's notation:

$$([A, C; if\sigma; (t, c); *]) = ([A, B; dr\sigma; (t, c)] : [B, C; if\sigma; (t', c'); *]) \tag{8.4}$$

The last notation in Equation 8.4 would be suitable for a KeyNote-like engine.

8.3 Components of the Trust* Model

8.3.1 Guarantees

The most common method of building trust between unknown entities in the real-world is by using guarantees. Although trust* can be propagated transitively, the risk is underwritten by a directly trusted principal. Guarantees only work if the *trusting* principal trusts the guarantor directly and the *trusted* principal is trusted directly by the the guarantor. With multiple guarantors in a chain, a guarantor would trust their neighbouring guarantor, therefore creating a chain of direct trust relationships between the trusting and trusted principals. For example, let A be a trust*er, C be a trust*ee, and B^i a guarantor:

$$A \underset{(t,c)}{\overset{\sigma}{\rightsquigarrow}} B^1 \underset{(t,c)}{\overset{\sigma}{\rightsquigarrow}} B^2 \underset{(t,c)}{\overset{\sigma}{\rightsquigarrow}} B^{\cdots} \underset{(t,c)}{\overset{\sigma}{\rightsquigarrow}} B^n \underset{(t,c)}{\overset{\sigma}{\rightarrow}} C \tag{8.5}$$

So long as A has direct referral trust in B^1, and B^n has direct functional trust in C, and every other B^i has direct referral trust in B^{i+1}, then a trust* chain can be made between A and C. There is no need for end-to-end trust. In real-world scenarios, these local relationships might already be existent and any guarantee agreements between them are likely to be underwritten in some way, for example, by signing a legally binding contract. Trust* is designed to work when there is no umpire.

There is still a transitive combination of direct referral and direct functional trust to form an indirect (functional) trust* relationship between A and C. However, the scope of the trust between each pair of principals has been agreed and underwritten with the forfeit t. Guarantors in a chain can be identified by looking at the type of trust used. For example, any referrals will be guaranteed by the referring trustee making B the guarantor in the examples in Section 8.2. A guarantor will receive c as a payment for their service and be required to pay the forfeit t if the trusting principal requires. The trust*ed end-point (C) won't receive a c payment (as he isn't a guarantor so $c = 0$) but in this example has agreed to pay a forfeit t if required by B.

It is important to note that forfeit payments are *always* paid in the opposite direction to that of the direction of trust regardless of whom invoked the trust* relationship (refer back to Section 7.4). It is only the direction of the commission payments that changes as illustrated in the following examples.

The direction of the commission payment in a forward trust* relationship (e.g. in the P2P application, see Figure 8.1) is made in the same direction as trust itself where the trust* relationship is initiated by the trust*er (a principal needing trust* *to* another). The relationship in Figure 8.1 has previously been expressed in Equation 8.2.

However, the direction of the commission payment in a reverse trust* relationship (e.g. in the spam-proof application, see Figure 8.2) is made in the opposite direction to that of trust (i.e. the same direction as the forfeit payments). In this case, the trust* relationship is usually initiated by the trust*ee (a principal needing

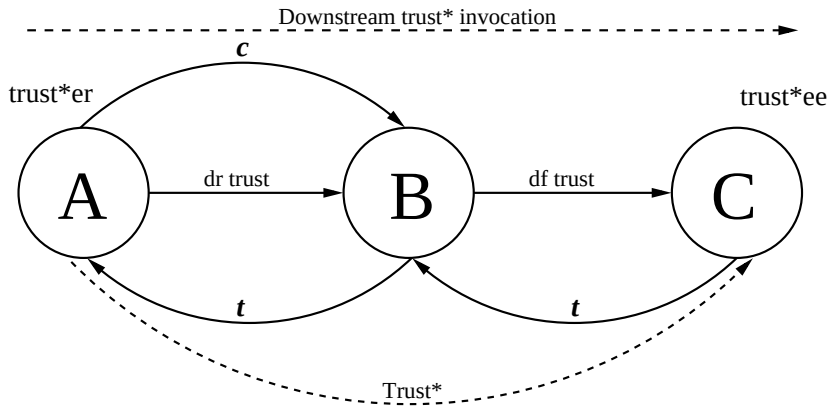


Figure 8.1: Example of a P2P forward trust* relationship where the **trust*er** is the initiator.

trust* *from* another).

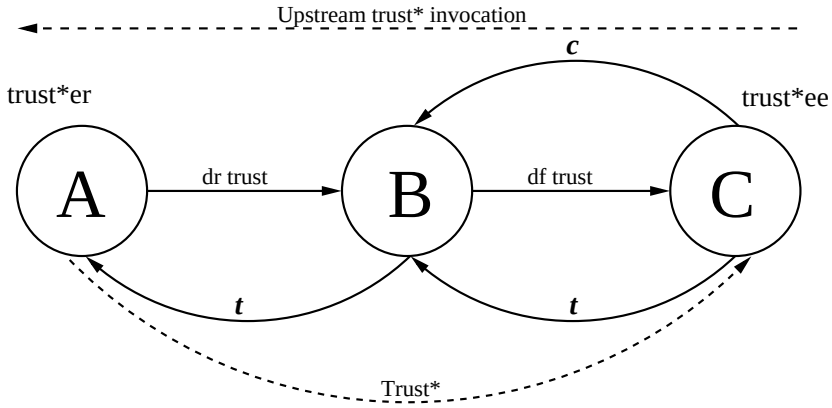


Figure 8.2: Example of a spam-proof reverse trust* relationship where the **trust*ee** is the initiator.

The relationship in Figure 8.2 can be expressed as:

$$([A, C; if\sigma; *]) = ([A, B; dr\sigma; (t, 0)] : [B, C; df\sigma, (t, -c)]) \tag{8.6}$$

Arrows in these diagrams which carry a value of 0 are not shown. However, arrows corresponding to negative values in the formula are paid in the opposite direction.

8.3.2 Payments

The trust* model includes two types of payment which are used for different purposes. The first is a commission (or incentive) payment which is paid to a principal to act as a guarantor. The second is a forfeit payment which serves two different purposes. Firstly, a forfeit can be paid to a principal who has made a claim on a guarantee as a way of compensating them. This is paid by the locally trusted guarantor to the claimant and later claimed from other guarantors until the end-point of the trust* chain is reached. Secondly, the forfeit acts to deter the server from defaulting in the first place as they will eventually be required to pay it. It will be in their interest to pay forfeits if required otherwise they risk losing trust from the local guarantor. Note the difference between the P2P application described in Chapter 3 where the client invokes the trust* protocol compared with the spam-proof application described in Chapter 7, where the server invokes the trust* protocol. The forfeit payments still serve the same purposes in both applications but the trust* protocol is invoked in different trust directions (i.e. upstream and downstream).

The two types of payment need not be like-for-like and may be of different tender between other local trust relationships in a trust* chain. A specific agreement between two principals will involve a negotiation of the commission and forfeit rates that will be a sufficient to give the guaranteeing party enough incentive and the guaranteed party adequate compensation. These rates will reflect the perceived trust between the two principals and the payments will likely be in a commodity that is of most use to them.

After a trust* protocol run, principals (particularly the guarantors and the guarantee receiver) are likely to re-assess the c and t values they are willing to accept from or pay to another. This is necessary to reflect the current level of risk involved with providing a guarantee or receiving a guarantee from another and will affect the subsequent routing costs of trust* relationships.

8.3.3 Protocol

The trust* protocol typically requires three types of principals. The two end-points (the end-point requiring trust* and the trust*ing end-point) and at least one

guarantor. There are cases such as in the P2P example where the principal being trust*ed might not actually take part in a protocol run but in most cases, their consent would also be needed.

The first part of a trust* protocol run consists of a number of guarantee requests being sent to local guarantors and their responses. It is this stage where agreements are made about what is being guaranteed and compensation arrangements. Once a guarantee chain has been created between a trust*er and a trust*ee, the transaction can take place. The second part of the protocol deals with notification of a guarantee and its claim status. This is shortly followed by compensation if a claim has been made.

The main difference in the way that a protocol is followed is defined by the purpose for which trust* is being used in the first place. For example, in the P2P application, Alice (the client) needed to find a way of trust*ing Carol (the server). Hence, this is why Alice invoked the protocol by requesting guarantees from principals whom *she* trusts. Conversely, in the spam-proof application, Carol needed to find a way to gain the trust of (or be trust*ed by) Alice. Carol now invokes the protocol but makes guarantee requests to principals who trust *her*.

8.4 Issues and Features

8.4.1 Heterogeneity

Due to the fact that a trust* relationship consists of a set of locally trusting principals in sequence between a trust*er and a trust*ee, the mechanisms used to implement the trust* protocol needn't be uniform across the chain. For example, Alice might deal with Bob in a different way than Bob deals with Carol. Typically, some kind of trust management system or other decision maker is recommended for building a trust* guarantee chain as risk assessment calculations can be enforced by a policy when following the protocol. However, so long as the two principals in a pairing have agreed on how they will follow the protocol, it doesn't matter what other mechanisms are being used in other parts of the chain. The same applies to the payment protocols that they will follow and the specific commodities they will use for the commission and forfeit payments.

8.4.2 Anonymity

In the trust* protocol, each guarantee is verified by the principal receiving it locally. Once a chain of guarantors has been found (say between C and A via B), how does C prove to A that she is in fact guaranteed to use their database? Some kind of access control credential could be used to encode the guarantee chain details which can be verified by A. However, A doesn't need to know who C is. All A needs to know is that she has received a guarantee from someone whom she trusts (B) and from whom she can claim a forfeit if A mis-uses the service provided. A doesn't care about any other local agreements in the chain, just the one between B and herself. Conversely, if C wants an assurance about the service provided by A (by invoking a cycle of trust*) — A doesn't need to know to whom she is *really* guaranteeing her service. Consequently, the trust* mechanism can be deployed in protocols where anonymity is required[1].

8.4.3 Resource Brokering

Due to the heterogeneous nature of localised trust relationships in the trust* protocol, the way that payments can be made is flexible and can allow resource brokering to take place. For example, a principal could make payments in commodity x and receive payments in commodity y and would be acting as a broker between these two commodities. A more concrete example could follow on from the previous chapter. Assume that for every spam email that Alice receives under guarantee from Bob, Alice is entitled to 1 minute of CPU time on Bob's personal computer or cluster as a means of compensation. Bob happens to have ample CPU cycles available to him and it would not be satisfactory compensation to him if he was to claim the same from Carol. However, Bob is short of hard-disk space and has an agreement with Carol that he is entitled to 10MB of data storage on her FTP server as a means of forfeit payment. Bob is effectively brokering CPU cycles and storage space and can prosper from resources that are more valuable to him and share resources which aren't so valuable to him. It's this difference in perceived value which drives commerce.

[1] Indeed, the guarantee chain can be used to provide anonymity as in Turtle.

The example given above might be considered unrealistic because everyone is assumed to have a commodity that they are willing to share such as CPU time. However, when applying trust* to an application such as grid computing where resources are already being shared, resource brokering is possible. For example, it might be that Alice already subscribes to a set amount of time on Bob's CPU cluster. A forfeit payment could simply be an extension (fee-free) to this allocated time.

8.4.4 Risk Assessment

To estimate the expected cost to a guarantor for providing a guarantee of another, a binomial distribution can be used. Pagano [84] defines binomial distribution as:

> "The binomial distribution is a probability distribution that results from a series of N trials, where on each trial there are only two possible outcomes. The outcomes are mutually exclusive, and there is independence between the outcomes of each trial. When these requirements are met, the binomial distribution tells us each possible outcome of the N trials and the probability of getting each of these outcomes."

We usually assume that the probability p of receiving an amount t is worth $p \cdot t$, but often the variance is even more important than the expectation. However, this depends on who you are (e.g. people buy both insurance and lottery tickets). Assuming that for the trust* model, a fixed amount is preferable to a variable amount with the same expected value, we model the value of a possible forfeit as:

$$\mu - (x \cdot \sigma) \tag{8.7}$$

Where μ is the expected value of $p \cdot t$, x is a co-efficient depending on the user, and σ is the standard deviation of $p \cdot t$. A high volume user such as a guarantee broker will tolerate a lower x value because they deal with a higher volume of transactions[2].

[2]This is because the mean and standard deviation for N independent transactions are $N \cdot p \cdot t$ and $\sqrt{N \cdot (1-p)} \cdot t$.

The forfeit payment is essentially binomial where the expectation is $p \cdot t$ where $p = P(alice\_defaults) \cdot P(alice\_pays\_up)$. So the worth is $t[p - x \cdot \sqrt{p(1-p)}]$. This expands as:

$$t[p - x\sqrt{p} + \frac{x}{2}p\sqrt{p} + \frac{x}{8}p^2\sqrt{p}\ldots] \tag{8.8}$$

This behaves like $p \cdot t$ provided $x << \sqrt{p}$ and like $-x\sqrt{p} \cdot t$ when $x >> \sqrt{p}$. Bob can calculate his possible losses if he provides a guarantee of Carol. During the request stages of the protocol, Bob can ensure that the c and t values that have been agreed between Carol and himself will cover the value of these losses before providing any guarantees to Alice. Consequently, this will affect Alice's route costs via Bob.

We are not economists and are by no means suggesting that this is the best way to calculate expected loss, however, this could provide a simple algorithm for updating costs in trust* routing tables and investigating the features of such algorithms is a possible direction of further work.

8.4.5 Cycles of Trust*

As a prevention mechanism against false guarantee claims, trust* can be deployed in two ways to ensure compliance from both end-points (or even intermediate guarantors). This is something that can be done initially if bi-directional agreements are required. However, it might be more likely that a cycle of trust* will be made if a principal suspects that habitual false claims are being made in an already existing trust* relationship.

It is important to note that even though the two trust* relationships in a cycle of trust* travel in opposite directions between end-points, each individual relationship is invoked in the same manner (i.e. forward or reverse, refer to Section 7.4). Take for example the P2P relationship in Section 8.3.1 which is expressed as:

$$([A, C; if\sigma; *]) = ([A, B; dr\sigma; (t, c)] : [B, C; df\sigma; (t, c)]) \tag{8.9}$$

Suppose that Carol has decided to always reimburse Bob if necessary (it may be that Bob's trust is important to her). Hence, she is careful about the quality of

her shared content. If Carol has good reason to suspect that Alice is making false claims, she could request a guarantee from David (whom she trusts) who in turn trusts that Alice will not make false claims. Carol pays David a commission to investigate a claim, and David will reimburse the forfeit to Carol if he has found Alice's claim to be falsely made in the first place. It is in Alice's best interests not to make false claims as she will risk losing David's trust. In order to prevent this she is likely to provide evidence to David proving that her claim was in-fact legitimate. This trust* relationship can be expressed as:

$$([C, A; if\sigma; *]) = ([C, D; dr\sigma; (t, c)] : [D, A; df\sigma; (t, c)]) \tag{8.10}$$

These are both forward routed trust* relationships as Alice and Carol invoke the protocol in the same direction as direct trust. Conversely, a cycle of trust* in the spam-proof application will consist of two reverse trust* relationships.

8.4.6 Networking Analogues

Many of the problems encountered when deploying trust* have networking analogues and hence have a choice of possible solutions obtained by applying the analogy in reverse to a networking protocol which addresses the corresponding problem. The best solution depends on the specific application in which trust* is being deployed and the infrastructure or services it might also provide. An example of such an analogy is how an optimal trust* route can be found between two principals. This is analogous to routing data packets in computer networks, however the cheapest route would be calculated on the commission cost rather than computational cost or number of hops. It is assumed that the problem of trust* routing is solved using conventional network routing algorithms or by a service provided by the application scenario (for example, using the existing mechanisms in a P2P client such as Turtle).

Another example of a networking analogue is congestion control. Using back-pressure to slow the transmission rate of packets from a source in a network is analogous to many guarantee claims being made and guarantors therefore increasing their commission requirements. If this happens near the destination of a trust* relationship, the effects will propagate back to the source making it harder

for them to continue. Eventually, the source won't be able to request any more guarantees if they habitually default guarantees. Analogously, in networking, the source will no longer be able to transmit packets to the destination until the congestion has been cleared. This process will affect other routes to the destination and hence may become more expensive.

8.5 Conclusion

This chapter has provided a review of the components of the trust* model. Each point reviewed in this chapter has already been introduced in this dissertation as required by a specific application of trust*. This chapter has tied these features together and provided a more formal description and discussion of the features of the trust* model so to provide an abstraction of the model for application to further scenarios requiring trust.

Chapter 9

Conclusions and Further Work

9.1 Introduction

This chapter reviews the contributions to knowledge made in this dissertation. Due to the flexibility of the trust* model and the fact that its protocol is very generic, there are many more applications that it could be applied to. Accordingly, we discuss some further developments of the trust* model and some other applications to which it might be beneficial. Finally, we summarise our conclusions.

9.2 Contributions to Knowledge

The main contribution to knowledge that this work makes is the trust* model. The model is conveyed in this dissertation via a number of different applications and comprises the following significant contributions:

- The use of localised guarantees to reduce the risk of transferring trust to other parties is the main construct of the trust* model. Although the guarantee business model is a real-world model and so is not a new idea, our novel use of trust management techniques to provide and manipulate electronic guarantees over derived trust relationships is a new concept.

- We believe that transitively trusting unknown principals without further pro-

tocol support is a dangerous idea and claim that the addition of guarantees lowers the risk of doing so. This is achieved by requiring a payment model to be followed by principals to provide incentives to act correctly and deterrents for not. A trust* relationship is still transitively derived between end-points, however, local agreements between each pair of direct trustees underwrite the risk involved for each trusting party.

- A commission payment model is introduced to provide incentives to guarantors. Principals can be paid a spot-price to act as a guarantor in a local trust relationship. This price is derived from an assessment of the risk and the likelihood that the principal being guaranteed will default. This offers flexibility in the cost of a guarantee depending on perceived trustworthiness and also allows new principals to bootstrap trust relationships (by initially paying a high premium).

- A forfeit payment model is introduced to provide both a deterrent for acting incorrectly and a compensation payment to affected parties. The forfeit rate also needs to be reconsidered when assessing the risk of other principals. A forfeit will need to be paid by a guarantor if a guarantee is claimed. This will force the guarantor to reconsider providing guarantees again, however, the claimant is compensated for their losses.

- We have applied trust* to a number of application scenarios in order to demonstrate its significance. These include P2P (Turtle), grid computing middleware (Globe), click-through licensing, and spam-proof applications.

Trust* offers a way of extending trust to allow transactions to take place between initially unknown or untrusted principals by using delegation rather than transitivity of trust. Trust is extended by identifying a willing chain of guarantors between the unknown principals. Each guarantor already trusts the next and is provided with an electronic guarantee by them. This guarantee is a local agreement between two principals outlining what is being guaranteed, the cost of the guarantee, and the forfeit that will be paid if a claim is made. This type of agreement localises the risk involved for trust*ing principals. Each principal trusts another to do the right thing or be held accountable (pay the forfeit). When something goes

wrong, the consequences are resolved locally between each pair of principals. If a guarantor refuses to conform with the agreement, it is likely that local trust will be damaged limiting their chances of providing (or receiving) future guarantees.

An advantage of requiring only localised trust is that the constructs used to build the guarantees and means of paying forfeits can be heterogeneous. The incentives to act as guarantor are solved by payments of resources or other tender that is valuable to another principal in local trust relationship. The combination of a commission and forfeit payment allows configurability between pairs of principals. An increasing tariff to aid flow control is analogous to network congestion control. This work provides a mechanism to support multiple policies allowing commission and forfeit rates to increase or decrease depending on the level of trust that is already present.

The whole process of extending trust to trust* makes use of the already existing trust relationships rather than creating new ones. It uses delegation of guarantees to bridge the gap between unknown principals with a sequence of localised agreements which remove or reduce the perceived risk of the trust*ing principal and shift it towards the principal being trust*ed.

Due to this, it is possible to use trust* alongside an existing trust infrastructure such as a reputation system. Also, the flexibility in how payments can be made in localised relationships could allow such payments to be made in a currency related to the trust infrastructure that is being complemented. For example, a principal could model their commission and forfeit rates on another's reputation rating.

There are some orthogonal issues which are either beyond the scope of this work or the trust* model. The main being that the trust* model doesn't solve or constrict the problem of local trust. It is assumed that these problems are solved outside of the trust* model. Other issues could be addressed by future work, which are discussed in the following section.

9.3 Further Work

This section outlines some further possible application scenarios with trust requirements to which applying trust* might be beneficial. Also, we discuss some

possible refinements to the current trust* model and propose some paths for future research.

9.3.1 Volunteer Computing

Also known as CPU scavenging, Volunteer Computing is where people collaborate on a large computational project by donating their spare CPU cycles to form a virtual super-computer. For example, Folding@Home is a project that aims to understand why proteins mis-fold. Another example is Seti@home which is a search for extra-terrestrial intelligence. A final example is the SHA-1 project which searches for collisions in the SHA-1 hash function. There are many more projects which range a variety of disciplines and platforms and which are used for different reasons.

Typically, a project is split into many units which can be distributively processed with their results fed back to the project server. Each user who wishes to participate with a project can register and connect their computer using a client such as BOINC [2]. Participants are normally awarded some kind of rating or credit depending on their contributions and successful computations. Trust* could use this credit as a currency for commission and forfeit payments to provision guarantees among participants. This will help to ensure the integrity of results from an untrusted computing base and help to isolate problem machines. It will therefore lower the overhead of checking and auditing of results.

9.3.2 Second Life

Trust* could be extended to real-world transactions such as e-commerce, but it's easier to keep a transaction purely electronic and use trust* purely in a virtual-world. In Second Life, trust* could be used to facilitate the buying and selling of virtual objects. Second Life has its own currency (Linden Dollars) which could be used for making the required commission or forfeit payments. Also, Linden Labs have recently revised Second Life's scripting language and cryptographic libraries which could make key and guarantee creation and verification even more viable.

9.3.3 P(GP) Web of Trust*

To help solve the problems with key distribution when using services such as a Public Key Infrastructure (PKI), the PGP suite provides a "web of trust" as a way of storing public keys and calculating their trustworthiness and validity based on who has signed them. A key's status can increase if it is signed by multiple trusted principals. The more signatories, the more valid the key is deemed. However, this process is similar to leaving ratings in a reputation system as a principal will be transitively trusting other principals about a particular key. Trust* could be used to provide this assurance by reducing the risk[1]. Rather than just signing a key, a guarantee could be included at a small price to ensure that the key is correct. Otherwise, a forfeit will be payable. So principals will need to think twice before signing a key. Tools such as PGP could have built-in capabilities to enable guarantee verification to take place. Keys that come with valid guarantees can be deemed more trustworthy (and less risky) than those that don't.

9.3.4 Trust* Implementation

The scope of this investigation was to develop the trust* model to enable trust to be extended over existing trust relationships. Also, to investigate whether a guarantee and payment model would be feasible in order to achieve this. The trust* model has thus far been simulated with regards to the applications presented in this dissertation. This was to test whether the trust* model might work theoretically in various scenarios but to also identify and analyse the key issues and features of such a model.

However, when modelling or simulating a trust environment, there is always likely to be some degree of rigidity due to the volatile nature of trust relationships (and how they are built) in the real-world. For example, the rate that a principal might increase their forfeit requirement is hard to predict and might vary drastically between principals. This could only be simulated by using randomly incremented values. Also, in the spam-proof application, a principal's tolerance to spam email is going to vary too and could only be randomised. Therefore, the

[1] A masters student at the University of Hertfordshire has already started working on applying trust* in this way [51].

next logical step should be to actually implement a real application of trust* such as the P2P trust* client discussed in Chapter 3. This could involve implementing a P2P client with built-in trust* capabilities. Perhaps heterogeneity could be sacrificed in this instance to allow a uniform decision maker and payment system to be used throughout a P2P network. Such an application could be tested by a real user-base to identify any issues with the current trust* model so improvements can be made.

By implementing and deploying a trust* solution, new research questions and paths are likely to appear from real results. For example, how heterogeneous can a trust* route be? It would be interesting to see how different schemes might be used in a typical trust* route and what types of brokering might take place. Also, the networking analogues discussed in this dissertation regarding issues such as routing and congestion control could be verified (e.g. by implementing specific routing algorithms or control techniques and analysing their effects). Further research in these areas will be far more fruitful when trust* is deployed in a real application.

9.3.5 Reputation as a Currency

The trust* model has been designed in a way that allows it to be used with any application where trusting unknown principals is necessary. At the same time, it doesn't necessarily need to replace any existing infrastructure, but could complement it. As shown in the spam-proof application, trust* could be used in combination with a traditional spam filter to aid its decisions rather than making it redundant. A possible line of future development could be to apply trust* to an application where reputation systems are already in full use and provide the mechanism to make risk assessments of others. Such services may include sites that act as auction houses such as eBay or other e-commerce companies including Amazon. More recently, the web has seen an increase in person-to-person services such as RentACoder and MyHammer. The first allows individuals or businesses to contact and employ independent software developers to help with projects. Coders have reputation ratings and can quote a price for completing a project. The higher a coder's rating is, the more they'll be able to charge. My-

Hammer is a similar service but allows tradesmen such as carpenters, electricians and plumbers to advertise themselves to people who need work done.

Typically, a reputation rating is altered after a transaction and will either increase, decrease or stay the same. It is proposed that the trust* mechanism could overlay such a system but using this reputation rating itself as a currency for paying forfeits. Habitually defaulting a guarantee will severely damage one's reputation rating. For performing a task or transaction well, an award payment could be agreed in order to increase a user's rating. This will enable users to more accurately gauge another's trustworthiness on sight but will also lower the risk of transacting with that person. They will have more to lose from not complying.

Referring back to Chapter 2 where the idea of community reputation is reviewed, the trust* model could be used alongside a reputation system that takes into consideration the reputation of a group or organisation (or online communities such as those mentioned above). As mentioned previously in the grid computing application (refer to Chapter 5), grids or other applications that cross organisational boundaries could also utilise a community reputation system in conjunction with trust* in this way.

9.3.6 More Anonymity

The fact that principals can be anonymous to other principals further down a trust* chain is a nice side-effect from building a chain of local trust relationships. For example, the end-points don't need to know each other or how many guarantors are between them. Principals in a chain only ever need to know their direct neighbours (i.e. principals whom they trust or who trust them directly) and thus some privacy is maintained.

Although this level of anonymity comes for free, there might be cases where a user wishes to be more anonymous. For example, onion routing could be used when creating a trust* route where longer chains are better than short. An interesting research project could involve applying Tor [37] (or other anonymising network) to the task of finding trust* routes between end-points. Enforcing anonymity in this way is likely to be more expensive for the initiating end-point as the commission premium will increase in relation to the length of the guarantor

chain. The more anonymity that is required, the higher the cost will be.

9.3.7 Auditability

The trust* mechanism could also be extended to situations where a guarantee chain needs to be identified (and verified) during audit. For example, an auditor might want to verify each guarantee which extends trust between A and C in order to prove a forfeit is payable (analogous to a bail bond agent or bounty hunter etc).

9.3.8 An Economic Model

Detailed economic modelling was beyond the scope of this work. Further work could be done to extend and analyse the economics of the trust* model. For example, exploring the effects of forfeit and commission payments in different scenarios and how risk assessments can be made to define these. Or the effects of different x values (refer to Section 8.4.4) on brokering and volume of transactions etc. Also, on a wider scale, do these constructs provide the correct deterrents from being bad and incentives for being good? A proper economic model of trust* and providing guarantees in a large online environment (e.g. a P2P network) is something that can be further explored from both an economist's and security economist's perspective.

9.4 Conclusion

This chapter has reviewed the contributions to knowledge that this dissertation has made. Also, possible extensions to the trust* model itself and the resulting expected research questions have been suggested as paths for future work. This chapter has also suggested some other applications to which applying trust* might improve the way that trust relationships are currently managed. Suggestions are given of further research into features that move beyond the scope of this work such as using trust to complement current trust mechanisms (e.g. reputation systems), and improvements to privacy and the economics of deploying trust* in real-world applications.

We conclude that trust* is an interesting and fruitful concept with which to build upon pre-existing trust relationships.

Bibliography

[1] BBBOnLine. http://www.bbbonline.org.

[2] Berkeley Open Infrastructure for Network Computing. http://boinc.berkeley.edu/.

[3] McAfee SiteAdvisor. http://siteadvisor.com.

[4] Repast Agent Simulation Toolkit. http://repast.sourceforge.net/.

[5] TRUSTe. http://www.truste.org.

[6] WOT. http://www.mywot.com/en/wot/home/.

[7] Martín Abadi, Andrew Birrell, Michael Burrows, Frank Dabek, and Ted Wobber. Bankable Postage for Network Services. In *ASIAN 2003*, pages 72–90. Springer-Verlag, 2003.

[8] Jemal H. Abawajy and Andrzej M. Goscinski. A Reputation-Based Grid Information Service. In *International Conference on Computational Science*. Springer-Verlag, 2006.

[9] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 310–317, 2001.

[10] Sulin Ba. Establishing Online Trust through a Community Responsibility System. *Decision Support Systems*, 31(3):323–336, 2001.

[11] Arno Bakker, Ihor Kuz, Maarten Van Steen, Andrew S. Tanenbaum, and Patrick Verkaik. Design and Implementation of the Globe Middleware. In *Technical Report IR-CS-003*, 2003.

[12] Jim Basney, Wolfgang Nejdl, Daniel Olmedilla, Von Welch, and Marianne Winslett. Negotiating Trust on the Grid. In *Proceedings of the 2nd WWW Workshop on Semantics in P2P and Grid Computing*, 2004.

[13] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos Keromytis. The Keynote Trust-Management System, 1998. `http://www.crypto.com/papers/rfc2704.txt`.

[14] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos Keromytis. The Role of Trust Management in Distributed Systems Security. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*. Springer-Verlag, 1999.

[15] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proceedings of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE, 1996.

[16] Matt Blaze, John Ioannidis, and Angelos Keromytis. Offline Micropayments without Trusted Hardware. In *Proceedings of the Fifth International Conference on Financial Cryptography*. Springer-Verlag, 2001.

[17] Matt Blaze, John Ioannidis, and Angelos Keromytis. Experience with the Keynote Trust Management System: Applications and Future Directions. In *Trust Management 2003, LNCS 2692*, pages 284–300. Springer-Verlag, 2003.

[18] L. Jean Camp, Helen Nissenbaum, and Cathleen McGrath. Trust: A Collision of Paradigms. In *Financial Cryptography*, pages 82–96. Springer-Verlag, 2001.

[19] Manuel Castells. *The Rise of the Network Society*. Blackwell, 2000.

[20] Elizabeth Chang, Patricia Thomson, Tharam Dillon, and Farookh Hussain. The Fuzzy and Dynamic Nature of Trust. In *TrustBus 2005*. Springer-Verlag, 2005.

[21] David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology Proceedings of Crypto 82*, pages 199–203, 1983.

[22] David Chaum. Security Without Identification: Transaction Systems to make Big Brother Obsolete. *Communications of the ACM*, 28(10), 1985.

[23] David Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology*, 1(1), 1988.

[24] David Chaum, Bert den Boer, Eugene van Heyst, Stig Mjølsnes, and Adri Steenbeek. Efficient Offline Electronic Checks. In *Proceedings of Eurocrypt'89*. Springer-Verlag, 1989.

[25] David Chaum, Amos Fiat, and Moni Naor. Untraceable Electronic Cash. In *CRYPTO*, pages 319–327. Springer-Verlag, 1990.

[26] Bruce Christianson and William S. Harbison. Why Isn't Trust Transitive? In *Proceedings of the International Workshop on Security Protocols*, pages 171–176. Springer-Verlag, 1997.

[27] Yang-Hua Chu, Joan Feigenbaum, Brian LaMacchia, Paul Resnick, and Martin Strauss. REFEREE: Trust Management for Web Applications. *The World Wide Web Journal*, 2(3):127–139, 1997.

[28] Brent N. Chun and Andy Bavier. Decentralized Trust Management and Accountability in Federated Systems. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, page 9, 2004.

[29] Richard Clayton. Poor Advice from SiteAdvisor. `http://www.lightbluetouchpaper.org/2007/08/12/poor-advice-from-siteadvisor/`.

[30] Richard Clayton. Insecure Real-World Authentication Protocols (or Why Phishing Is So Profitable). In *Proceedings of 13th International Workshop on Security Protocols*. Springer-Verlag, 2005.

[31] Daniel Cvrcek and Ken Moody. Combining Trust and Risk to Reduce the Cost of Attacks. In *iTrust 2005*. Springer-Verlag, 2005.

[32] Paul B. de Laat. Trusting Virtual Trust. *Ethics and Information Technology*, 7(3):167–180, 2005.

[33] Emerson Ribeiro de Mello, Aad P. A. van Moorsel, and Joni da Silva Fraga. Evaluation of P2P Search Algorithms for Discovering Trust Paths. In *EPEW*, pages 112–124. Springer-Verlag, 2007.

[34] Chrysanthos Dellarocas. Mechanisms for Coping with Unfair Ratings and Discriminatory Behavior in Online Reputation Reporting Systems. In *ICIS '00: Proceedings of the Twenty First International Conference on Information Systems*, pages 520–525. Association for Information Systems, 2000.

[35] Edsger Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerical Mathematics*, 1(1):269–271, 1959.

[36] Roger Dingledine, Michael J. Freedman, David Molnar, David Parkes, and Paul Syverson. Reputation. In *Digital Government Civic Scenario Workshop*, 2003.

[37] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, pages 303–320, 2004.

[38] John R. Douceur. The Sybil Attack. In *Peer-To-Peer Systems: First International Workshop*, page 251. Springer-Verlag, 2002.

[39] Benjamin Edelman. Adverse Selection in Online "Trust" Certifications. In *WEIS06*, 2006.

[40] Simon N. Foley. Using Trust Management to Support Transferable Hash-Based Micropayments. In *Financial Cryptography 2003*, pages 1–14. Springer-Verlag, 2003.

[41] Simon N. Foley and Thomas B. Quillinan. Using Trust Management to Support MicroPayments. In *Proceedings of the Second Information Technology and Telecommunications Conference*, 2002.

[42] L. Ford and D. Fulkerson. *Flows In Networks*. Princeton University Press, Princeton, NJ, 1962.

[43] Batya Friedman, Peter H. Kahn Jr., and Daniel C. Howe. Trust Online. *Communications of the ACM*, 43(12):34–40, December 2000.

[44] Samual Galice, Marine Minier, John Mullins, and Stephane Ubeda. Cryptographic Protocol to Establish Trusted History of Interactions. In *ESAS 2006*, pages 136–149. Springer-Verlag, 2006.

[45] Steve Glassman, Mark Manasse, Martín Abadi, Paul Gauthier, and Patrick Sobalvarro. The Millicent Protocol for Inexpensive Electronic Commerce. In *Fourth International World Wide Web Conference Proceedings*, pages 603–618, 1995.

[46] Jennifer Golbeck and James A. Hendler. Reputation Network Analysis for Email Filtering. In *CEAS*, 2004.

[47] Joshua Goodman and Robert Rounthwaite. Stopping Outgoing Spam. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 30–39. ACM New York, NY, USA, 2004.

[48] Tyrone Grandison and Morris Sloman. A Survey of Trust in Internet Applications. *IEEE Communications Surveys and Tutorials*, 2000.

[49] William Saumual Harbison. *Trusting in Computer Systems*. PhD thesis, Wolfson College, University of Cambridge, 1997.

[50] Evan Harris. The Next Step in the Spam Control War: Greylisting. Technical report, PureMagic Software, 2003. `http://projects.puremagic.com/greylisting/whitepaper.html`.

[51] Sarvjeet Herald. One Way Non-Transitive Trust Model in Public Key Infrastructure. Master's thesis, School of Computer Science, University of Hertforshire, 2009.

[52] Donna L. Hoffman, Thomas P. Novak, and Marcos Peralta. Building Consumer Trust Online. *Communications of the ACM*, 42(4):80–85, 1999.

[53] Philip Homburg. *The Architecture of a Worldwide Distributed System*. PhD thesis, Vrije Universiteit, Amsterdam, 2000.

[54] Jingwei Huang and Mark S. Fox. An Ontology of Trust: Formal Semantics and Transitivity. In *ICEC 2006*, pages 259–270. ACM, 2006.

[55] John Ioannidis. Fighting Spam by Encapsulating Policy in Email Addresses. In *NDSS*, 2003.

[56] Marksu Jakobsson, Alex Tsow, Ankur Shah, Eli Blevis, and Youn-Kyung Lim. What Instills Trust? A Qualitative Study of Phishing. In *FC 2007 and USEC 2007, LNCS 4886*. Springer-Verlag, 2008.

[57] Junjie Jiang, Haihuan Bai, and Weinong Wang. Trust and Cooperation in Peer-to-Peer Systems. In *GCC 2003*, pages 371–378. Springer-Verlag, 2004.

[58] Hai Jin, Xuping Tu, Zongfen Han, and Xiaofei Liao. A Community-Based Trust Model for P2P Networks. In *HPCC*, pages 419–428. Springer-Verlag, 2005.

[59] Audun Jøsang. The Right Type of Trust for Distributed Systems. In *Proceedings of the 1996 Workshop on New Security Paradigms*. ACM New York, NY, USA, 1996.

[60] Audun Jøsang. Trust and reputation systems. In *FOSAD 2006/2007, LNCS 4677*, 2007.

[61] Audun Jøsang, Elizabeth Gray, and Michael Kinateder. Analysing Topologies of Transitive Trust. In *Proceedings of the Workshop of Formal Aspects of Security and Trust*, pages 9–22, 2003.

[62] Audun Jøsang, Ross Hayward, and Simon Pope. Trust Network Analysis with Subjective Logic. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*. Australian Computer Society, Inc., 2006.

[63] Audun Jøsang, Roslan Ismail, and Colin Boyd. Survey of Trust and Reputation Systems for Online Service Provision. In *Decision Support Systems*, 2005.

[64] Audun Josang, Stephen Marsh, and Simon Pope. Exploring Different Types of Trust Propagation. In *iTrust 2006*. Springer-Verlag, 2006.

[65] Audun Jøsang and Simon Pope. Semantic Constraints for Trust Transitivity. In *APCCM 2005*, pages 59–68. Australian Computer Society, Inc., 2005.

[66] Audun Jøsang and Stéphane L. Presti. Analysing the Relationship between Risk and Trust. In *Proceedings of the 2nd International Conference on Trust Management*, pages 135–145. Springer-Verlag, 2004.

[67] Tim Kindberg, Abigail Sellen, and Erik Geelhoed. Security and Trust in Mobile Interactions: A Study of Users' Perceptions and Reasoning. In *UbiComp 2004*. Springer-Verlag, 2004.

[68] Peter Kollock. The Production of Trust in Online Markets. *Advances in Group Processes*, 16, 1999.

[69] Eleni Koutrouli and Aphrodite Tsalgatidou. Reputation-Based Trust Systems for P2P Applications: Design Issues and Comparison Framework. In *Trust and Privacy in Digital Business*. Springer-Verlag, 2006.

[70] H. T. Kung, Trevor Blackwell, and Alan Chapman. Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation and Statistical Multiplexing. In *SIGCOMM*, pages 101–114, 1994.

[71] H. T. Kung and Koling Chang. Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks. In *INFOCOM*, pages 239–252, 1995.

[72] Li Lifen. Trust Derivation and Transitivity in a Recommendation Trust Model. In *CSSE 2008*, pages 770–773. IEEE Computer Society, 2008.

[73] Stephen P. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, 1994.

[74] Nicholas F. Maxemchuk and Magda El Zarki. Routing and Flow Control in High-Speed Wide-Area Networks. *Proceedings of the IEEE*, 78(1):204–221, 1990.

[75] Jimmy McGibney and Dmitri Botvich. Establishing Trust Between Mail Servers to Improve Spam Filtering. In *ATC 2007*. Springer-Verlag, 2007.

[76] Nicola Mezzetti. Towards a Model for Trust Relationships in Virtual Enterprises. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, 2003.

[77] Stanley Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.

[78] Anirban Mondal and Masaru Kitsuregawa. Privacy, Security and Trust in P2P environments: A Perspective. In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 682–686, 2006.

[79] John P. Morrison, James J. Kennedy, and David A. Power. WebCom: A Web Based Volunteer Computer. *Journal of Supercomputing*, 18(1):47–61, 2001.

[80] Michael J. North, T. R. Howe, Nick T. Collier, and J. R. Vos. The Repast Simphony Development Environment. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, 2005.

[81] Michael J. North, T. R. Howe, Nick T. Collier, and J. R. Vos. The Repast Simphony Runtime System. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, 2005.

[82] Michael J. North, Eric Tatara, Nick T. Collier, and J. Ozik. Visual Agent-Based Model Development with Repast Simphony. In *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence, Argonne National Laboratory*, 2007.

[83] Daniel Olmedilla, Omer F. Rana, Brian Matthews, and Wolfgang Nejdl. Security and Trust Issues in Semantic Grids. In *In Proceedings of the Dagsthul Seminar, Semantic Grid: The Convergence of Technologies*, 2005.

[84] Robert R. Pagano. *Understanding Statistics in the Behavioral Sciences*. West Publishing, 1990.

[85] Elvis Papalilo and Bernd Freisleben. Towards a Flexible Trust Model for Grid Environments. In *GSEM 2004*, pages 94–106. Springer-Verlag, 2004.

[86] Filip Perich, Jeffrey Undercoffer, Lalana Kagal, Anupam Joshi, Timothy Finin, and Yelena Yesha. In Reputation We Believe: Query Processing in Mobile Ad-Hoc Networks. In *MOBIQUITOUS 2004*, pages 326–334. IEEE, 2004.

[87] Bogdan C Popescu. *Design and Implementation of a Secure Wide-Area Object Middleware*. PhD thesis, Vrije Universiteit, Amsterdam, 2006.

[88] Bogdan C. Popescu, Bruno Crispo, and Andrew S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-up and Beat the System. In *In Proc. of the 12th Cambridge International Workshop on Security Protocols*, 2004.

[89] Bogdan C. Popescu, Bruno Crispo, Andrew S. Tanenbaum, and Mass Zeeman. Enforcing Security Policies for Distributed Objects Applications. In *Proceedings of the International Workshop on Security Protocols*, volume 3364, page 119. Springer-Verlag, 2005.

[90] Jing-Fei Ren and Jon W. Mark. Design and Analysis of a Credit-Based Controller for Congestion Control in B-ISDN/ATM Networks. In *INFOCOM*, pages 40–48, 1995.

[91] Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kuwabara. Reputation Systems: Facilitating Trust in Internet Interactions. *Communications of the ACM*, 43(12):45–48, 2000.

[92] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust Management for the Semantic Web. In *ISWC 2003*. Springer-Verlag, 2003.

[93] Sini Ruohomaa and Lea Kutvonen. Trust Management Survey. In *iTrust 2005*. Springer-Verlag, 2005.

[94] Kenji Saito. Peer-to-Peer Money: Free Currency over the Internet. In *HSI 2003*, pages 404–414. Springer-Verlag, 2003.

[95] Roman Schlegel and Serge Vaudenay. Enforcing Email Addresses Privacy Using Tokens. In *CISC*, 2005.

[96] Mischa Schwartz and Thomas E. Stern. Routing Techniques Used in Computer Communication Networks. *IEEE Transactions on Communications*, 28(4):539–552, 1980.

[97] Popular Science. Dime put in slot rings doorbell, 1933. `http://blog.modernmechanix.com/2007/05/05/dime-put-in-slot-rings-doorbell/`.

[98] Ali A Selçuk, Ersin Uzun, and Mark R. Pariente. A Reputation-Based Trust Management System for P2P Networks. In *CCGRID*, pages 251–258. IEEE Computer Society, 2004.

[99] Ben Shneiderman. Designing Trust Into Online Experiences. *Communications of the ACM*, 43(12):57–59, 2000.

[100] Craig Van Slyke, France Belanger, and Christie L. Comunale. Factors Influencing the Adoption of Web-Based Shopping: The Impact of Trust. *Database for Advances in Information Systems*, 35(2):32–49, 2004.

[101] William Stallings. *Data and Computer Communications*. Prentice Hall, 2007.

[102] Martha Steenstrup. *Routing in Communication Networks*. Prentice Hall, 1995.

[103] Yao-Hua Tan. A Trust Matrix Model for Electronic Commerce. In *Trust Management 2003*, pages 33–45. Springer-Verlag, 2003.

[104] W. T. Luke Teacy, Jigar Patel, Nicholas R. Jennings, and Michael Luck. TRAVOS: trust and reputation in the context of inaccurate information sources. *Autonomous Agent and Multi-Agent Systems*, 12(2):183–198, 2006.

[105] Octavian Ureche and Rejean Plamondone. Digital Payment Systems for Internet Commerce: The State of the Art. In *World Wide Web 3*, 2000.

[106] Le-Hung Vu and Karl Aberer. A Probabilistic Framework for Decentralized Management of Trust and Quality. In *CIA 2007*. Springer-Verlag, 2007.

[107] Dan S. Wallach. A Survey of Peer-to-Peer Security Issues. In *In International Symposium on Software Security*, pages 42–57, 2002.

[108] Shuo Yang, Ali R. Butt, Xing Fang, Y. Charlie Hu, and Samual P. Midkiff. A Fair, Secure and Trustworthy Peer-to-Peer Based Cycle-Sharing System. *Journal of Grid Computing*, 4(3):265–286, 2006.

[109] Shuo Yang, Ali R. Butt, Y. Charlie Hu, and Samuel P. Midkiff. Trust but Verify: Monitoring Remotely Executing Programs for Progress and Correctness. In *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 196–205. ACM New York, NY, USA, 2005.

[110] Danfeng Yao, Keith B Frikken, Mikhail J Atallah, and Roberto Tamassia. Point Based Trust: Define How Much Privacy Is Worth. In *ICICS 2006*, pages 190–209. Springer-Verlag, 2006.

[111] Weiliang Zhao, Vijay Varadharajan, and George Bryan. Modelling Trust Relationships in Distributed Environments. In *TrustBus*, pages 40–49, 2004.

[112] Weiliang Zhao, Vijay Varadharajan, and George Bryan. Analysis and Modelling of Trust in Distributed Information Systems. In *ICISS05*, pages 106–119, 2005.

[113] Elena Zheleva, Aleksander Kolcz, and Lise Getoor. Trusting Spam Reporters: A Reporter-Based Reputation System for Email Filtering. *ACM Transactions on Information Systems*, 27(1), 2008.

Appendix A

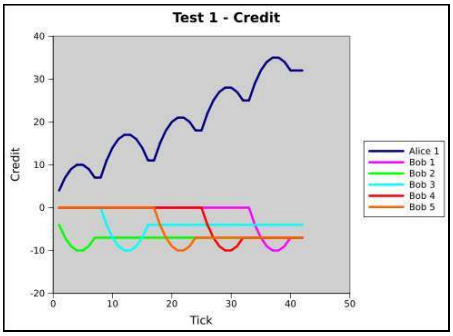
Simulation Results

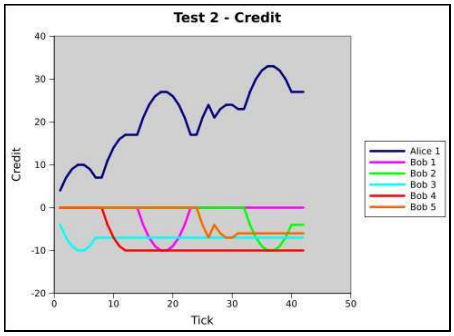
A.1 P2P Tests 1–6

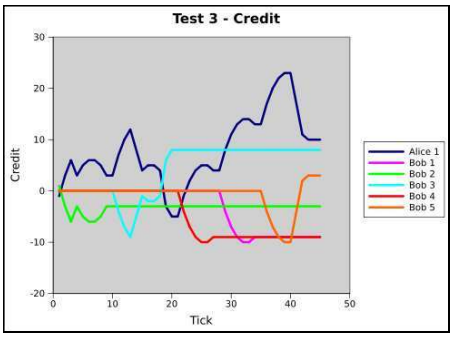
Tick Count These simulations lasted for 42, 42, 45, 48, 99 and ∞ ticks respectively before all possible guarantors became inactive. This would make sense as when both Alice and Carol are bad, the guarantors would become exhausted quickly. This time gradually gets longer as Alice and Carol become good. Eventually in test 6, Only one of the guarantors is ever used as there is no reason for him to stop providing guarantees for Carol and he has no reason to suspect false claims from Alice. Test 6 was stopped at around 500 ticks, however, it would have carried on forever.

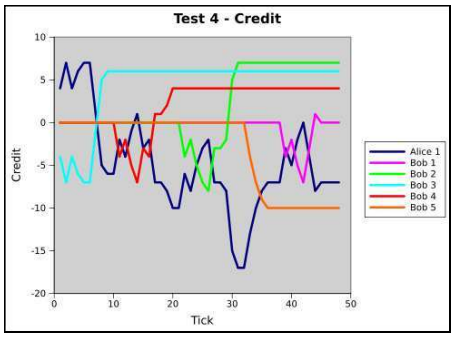
Credit Test 1 and 2 are very similar. The combination of a low `truthchance` and a high `malwarechance` means that Alice will claim a forfeit most of the time. This makes her better off credit-wise, however, each guarantor only allows this to happen for a short time before becoming inactive. By test 3, there is less chance that a claim will be made, and two of the guarantors actually make a profit for their service. Alice is still above zero but not by as much considering that she is still paying a commission every tick. By tests 4 and 5, most of the guarantors end with more credit than Alice who's credit has gone below zero. Test 6 only uses one of the guarantors and shows a steady decrease in Alice's credit and a steady increase for Bob. This reflects the commission that Alice has paid Bob and

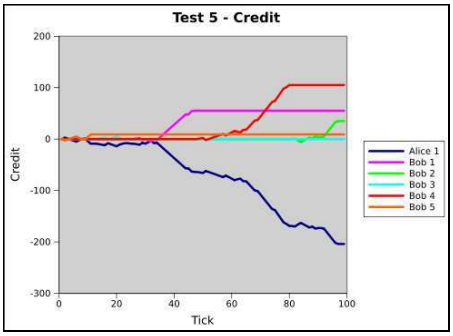
the fact that no claims were ever made.

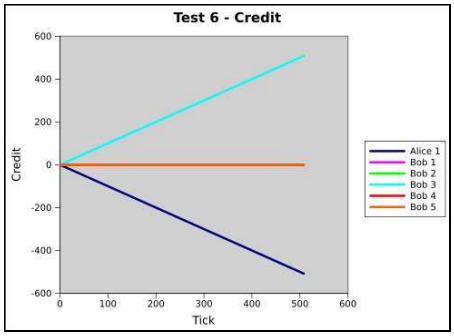






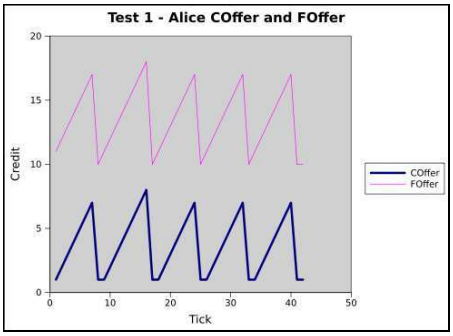


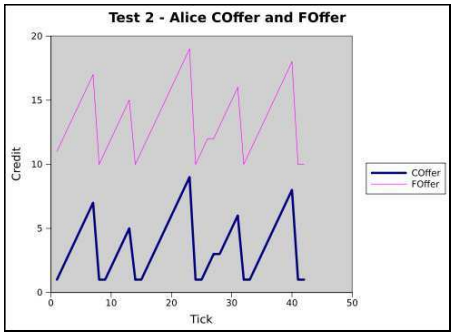


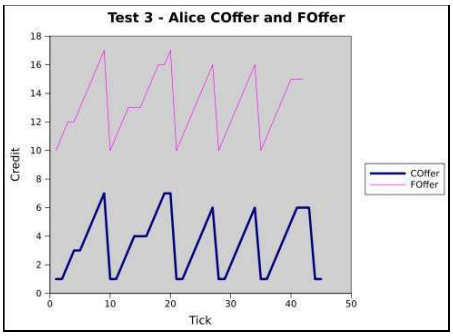


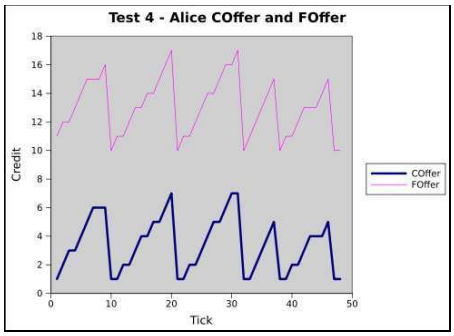
Alice's COffer and FOffer The offers that Alice makes in the first tests increase very steeply. This is due to claims being regularly made and hence Bob requiring a higher commission and Alice requiring a higher forfeit to reflect this. As less claims are made, these increases become more jagged and less steep as the frequency of claims decreases. As Bob has a maximum forfeit that he is willing to pay and Alice has a minimum commission she is willing to pay, these offers

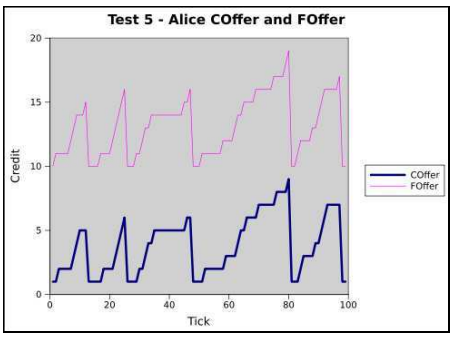
will govern how long a particular guarantor is likely to be used and will effect the simulation length (tick count). As the changes in offers become less steep, the longer a guarantor will be used for. For example, in test 6, the offers never change and hence, the charts have flat-lined.

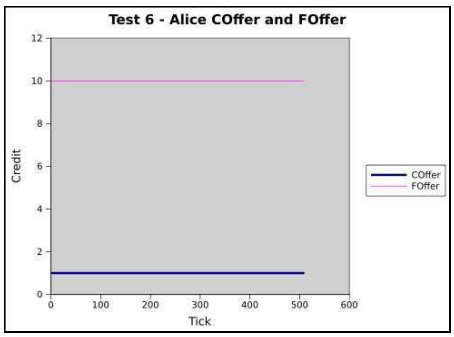






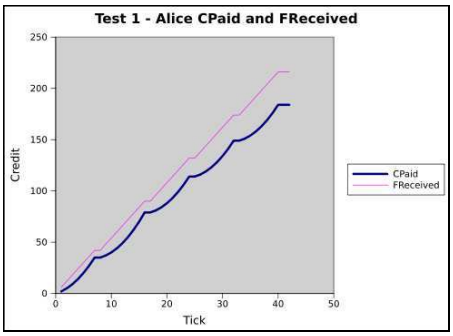


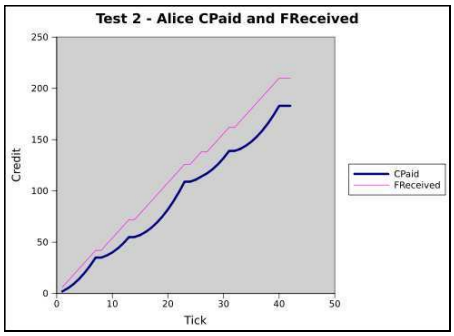


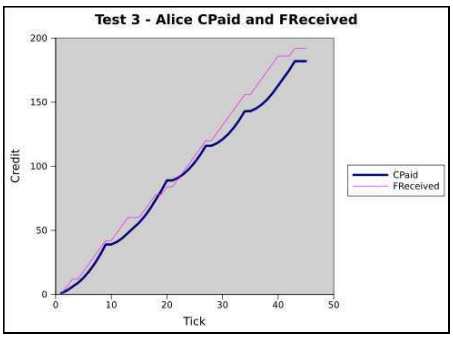


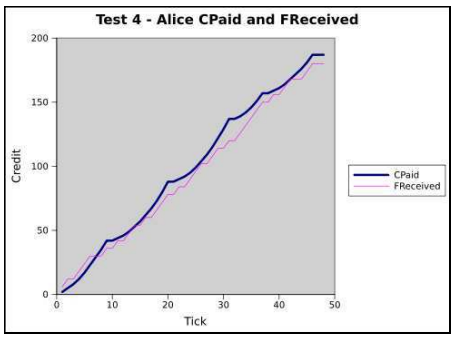
Alice's CPaid and FReceived The overall amount of commission that Alice has paid Bob is initially lower than the forfeit that she has received from Bob. However, the simulation time is at its lowest. As the number of claims decreases,

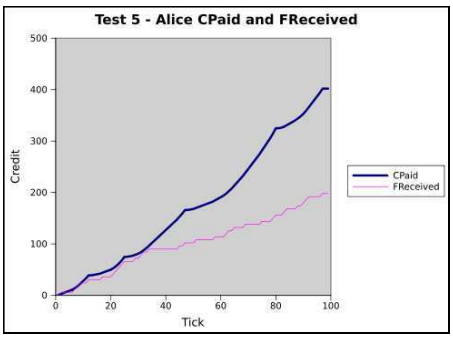
the forfeit received decreases too. Because of this, the simulation lasts longer and by test 6, no forfeit is paid to Alice.

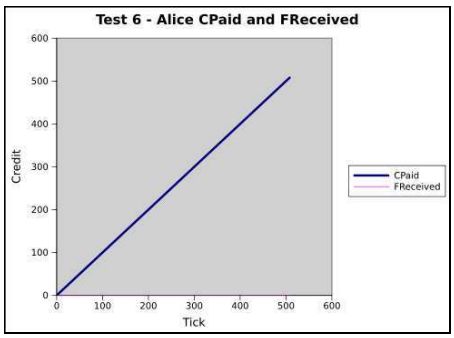




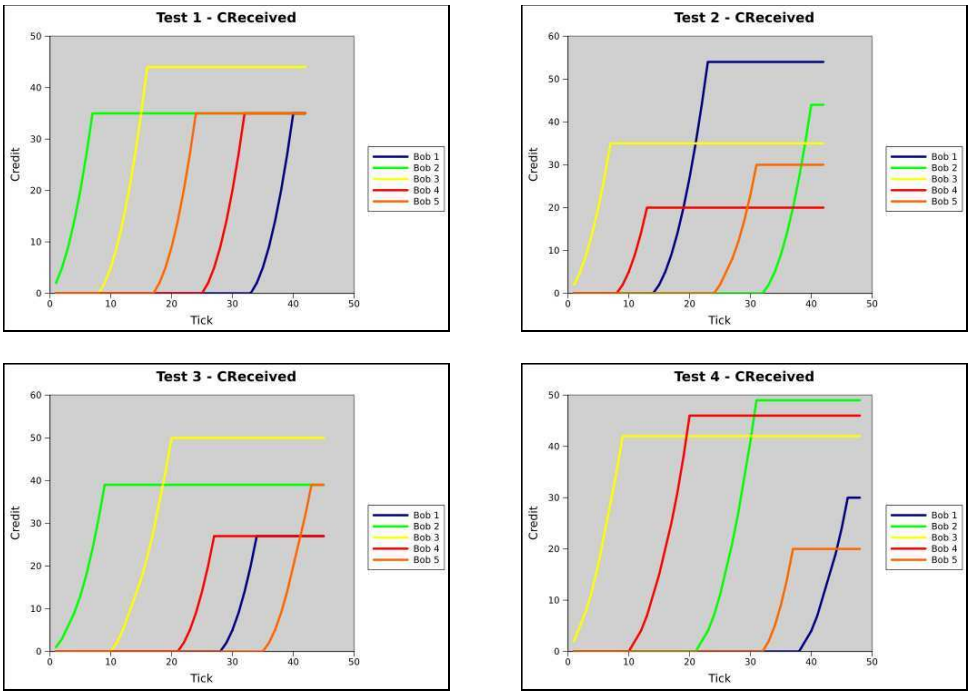






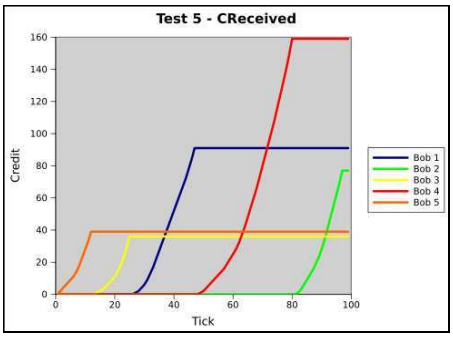


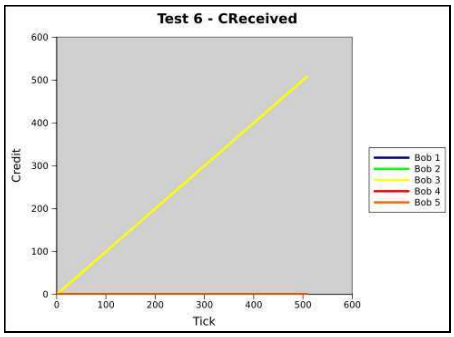
Bob's CReceived Again, due to the the number of claims being high in test 1 and decreasing to none by test 6, the commissions received by all guarantors are fairly uniform between them. By test 5, where claims were less frequent, some guarantors received more commission than others from Alice before becoming inactive. Overall, the average commission received increases as the number of claims decreases.

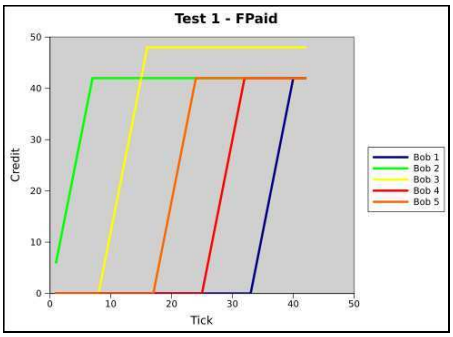


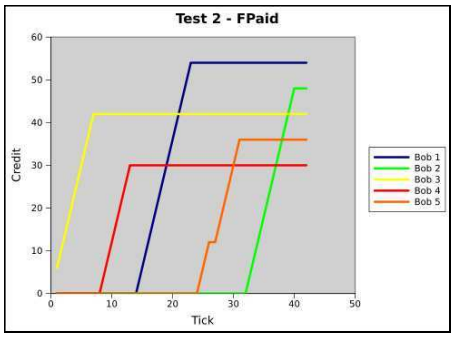
Bob's FPaid These results reflect the commission received in that the guarantor who received the most commission also paid the most forfeit. However, the average forfeit paid decreased as the number of claims decreased.

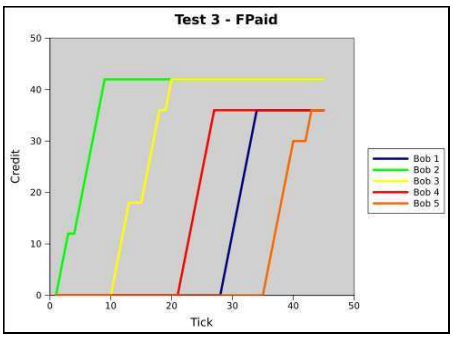
Alice's Claims and False Claims The observations made thus far are mainly related to the number of claims that Alice makes. Test 1 comprises of legitimate claims only. This is due to Carol serving 100% malware, so Alice has no reason to make any false claims. Tests 2, 3, 4 and 5 gradually increases the number of false claims made. The randomness of `truthchance` will effect the frequency of false claims and hence will effect guarantors differently. By test 6, no claims are made. Some interesting effects are caused here by the combination of `truthchance` and `malwarechance` in the number of claims that are made. Even though the total number of claims slowly decreases as both Alice and Carol become good, there is still a high proportion of false claims being made in tests 3–5. For example, in test 5, Alice made 16 legitimate claims and 17 false claims.

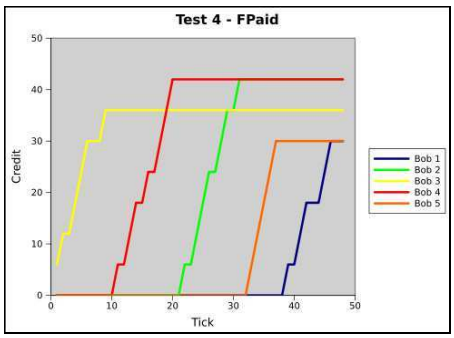

Test 5 - CReceived
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

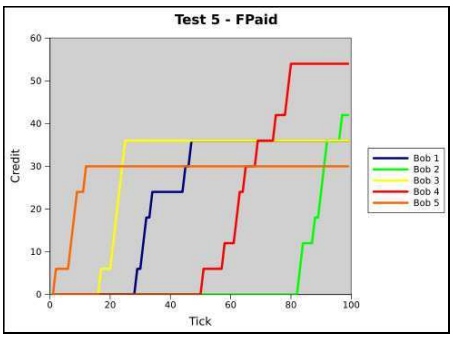

Test 6 - CReceived
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

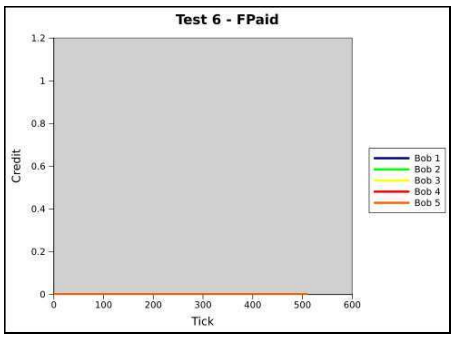

Test 1 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

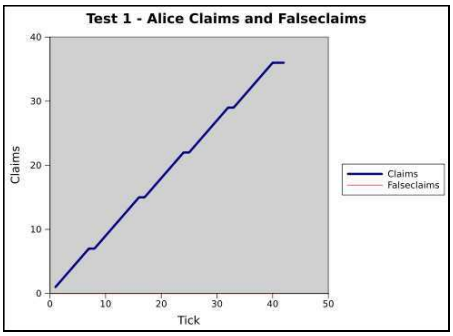

Test 2 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

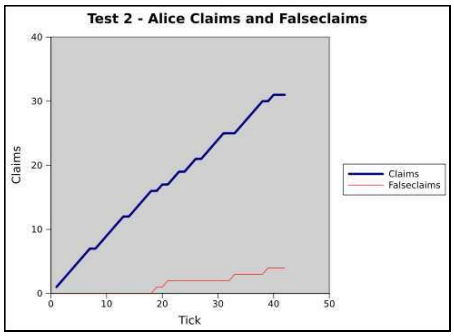

Test 3 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

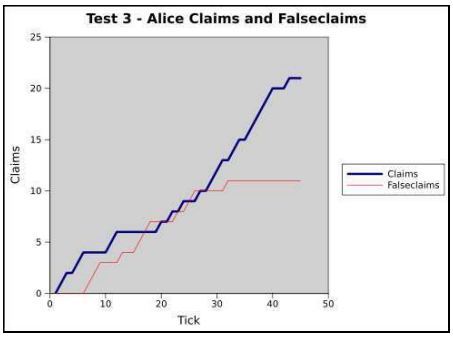

Test 4 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

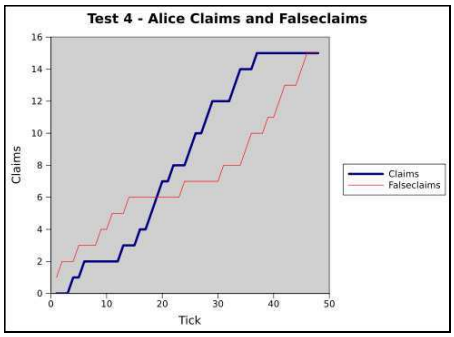

Test 5 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

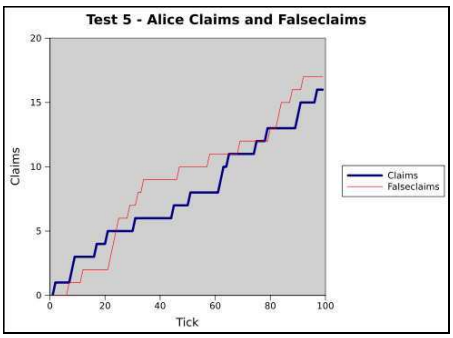

Test 6 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

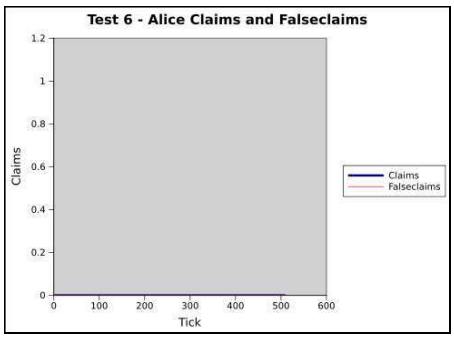

Test 1 - Alice Claims and Falseclaims
Claims
Tick
Claims
Falseclaims


Test 2 - Alice Claims and Falseclaims
Claims
Tick
Claims
Falseclaims


Test 3 - Alice Claims and Falseclaims
Claims
Tick
Claims
Falseclaims


Test 4 - Alice Claims and Falseclaims
Claims
Tick
Claims
Falseclaims

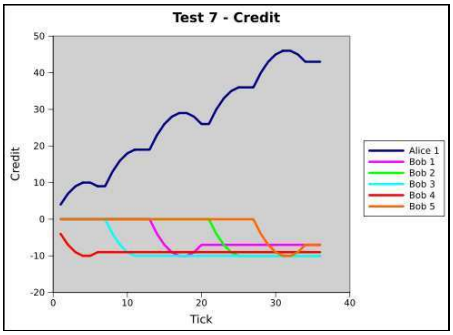

Test 5 - Alice Claims and Falseclaims
Claims
Tick
Claims
Falseclaims

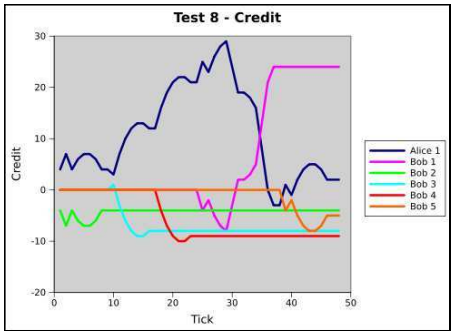

Test 6 - Alice Claims and Falseclaims
Claims
Tick
Claims
Falseclaims

A.2 P2P Tests 7–12

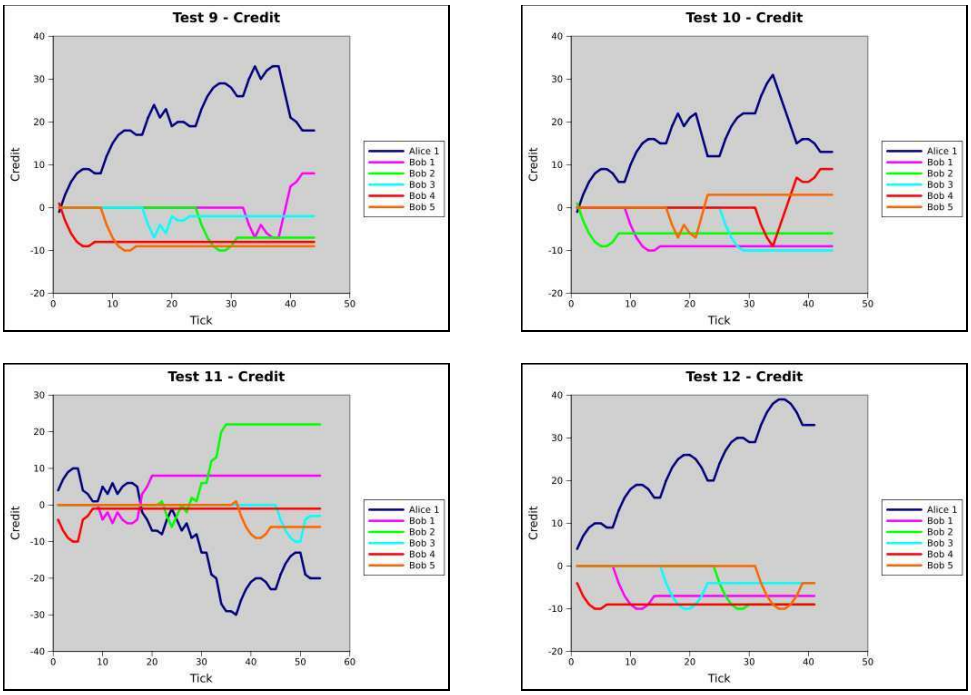
Tick Count These simulations lasted for 36, 48, 44, 44, 54 and 41 ticks respectively before all possible guarantors became inactive. Tests 7 and 12 appear to be the quickest, however tests 9 and 10 also ended quickly. It is evident that when either Alice or Carol act incorrectly (making false claims or serving incorrect content), their choices when using trust* will be hindered. The best trade-off between `truthchance` and `malwarechance` appears to be when they are both 0.2 and 0.8 (in tests 8 and 11 respectively). It is fairly conclusive that when either of the end-to-end principals behaves incorrectly, that the trust* protocol won't tolerate them and their use of trust* will be short-lived.

Credit The changes in credit take a similar pattern to that of the tick count above. Test 7 and 12 are virtually identical except Alice finishes with slightly less credit in test 12. Tests 8 and 11 are similar but appear inverted. For example, test 8 shows Alice making a small profit and only one guarantor making quite a high profit. Whereas test 11 shows the opposite with Alice making a loss and multiple guarantors making a gain. It appears from this that Alice can still make a profit from being untruthful (however short-lived it is). Tests 9 and 10 are again very alike with Alice making a small profit in both cases. Some guarantors made a profit however the average loss by a guarantor seems to be around 10 before they refuse to continue providing guarantees to Alice about Carol's files.

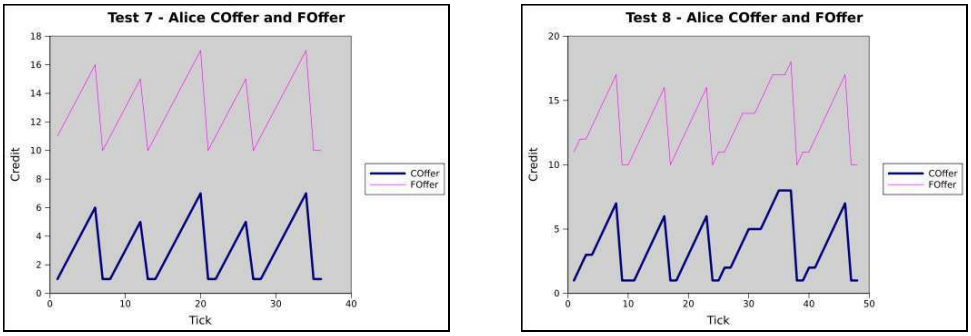




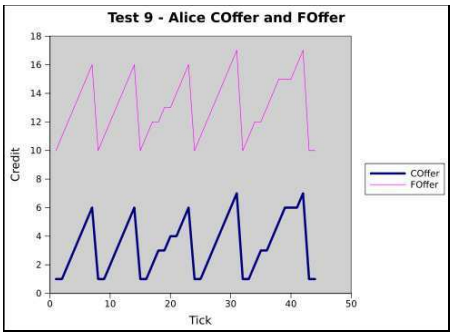
Alice's COffer and FOffer Alice's commission and forfeit offer values increase very steeply with the steepest being in tests 7 and 12 where either Alice

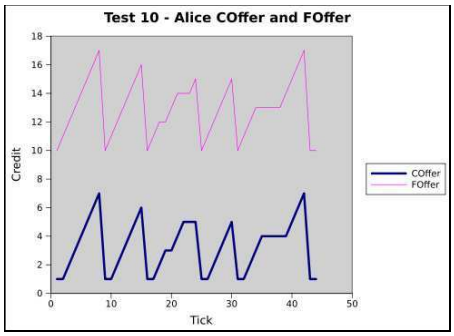


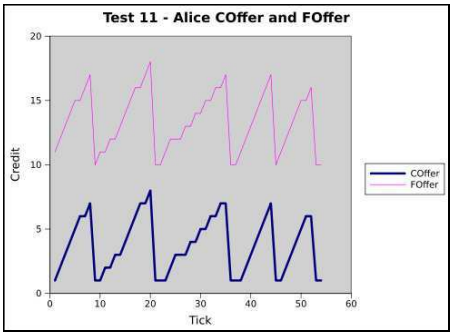
or Carol are acting incorrectly 100% of the time. Some jaggedness is evident throughout tests 8 to 11 and there is no flat-lining as there was for test 6. This explains why the average simulation tick count was very low in tests 7 to 12.

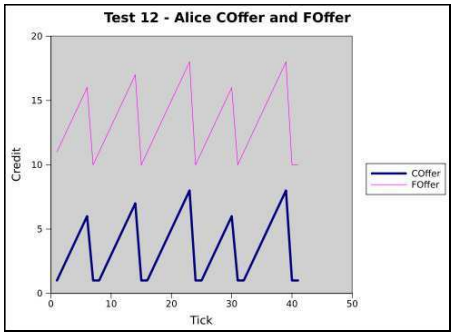


Alice's CPaid and FReceived In most of these tests, Alice received more forfeit payments than she had paid in commission. Even in the worst case (test 11), the difference is very small. It would appear that it might be in Alice's best interests to habitually make false claims as she is likely to make a profit. However, this will not last long before a guarantor increases his `cMin` beyond excess. Similarly

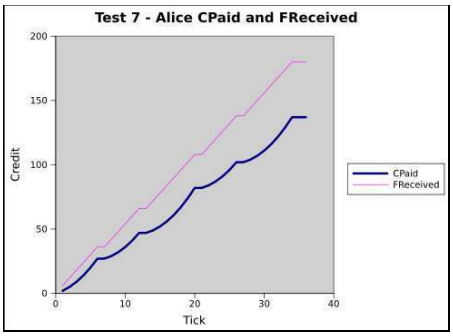


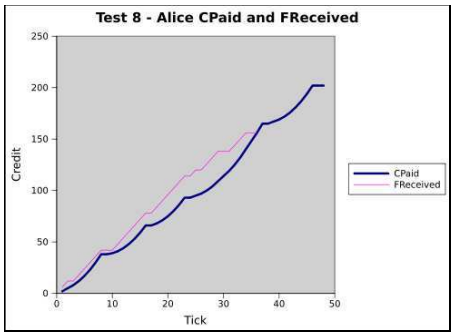






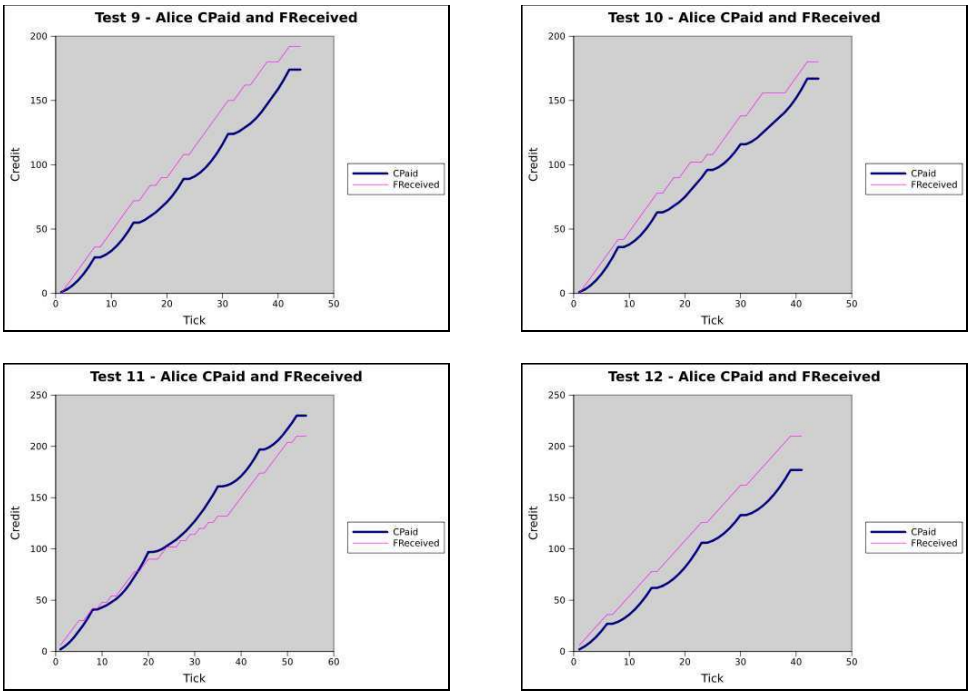
when Carol is at her worst in test 12, a guarantor will only tolerate so many claims whether they be legitimate or not.





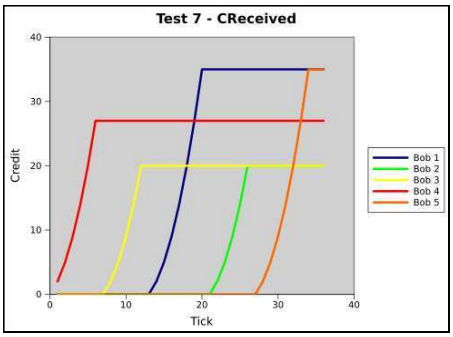
Bob's CReceived As these tests involved someone always acting badly, there are always claims being made. This forces the guarantors to increase their commission requirements. Again, due to the randomness of the claims, guarantors are affected differently with some fairing better than others.

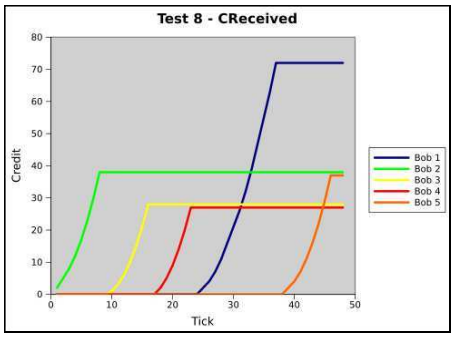
Bob's FPaid The guarantors pay a fair amount of forfeit before they become inactive due to the number of claims being made. However, due to their increas-

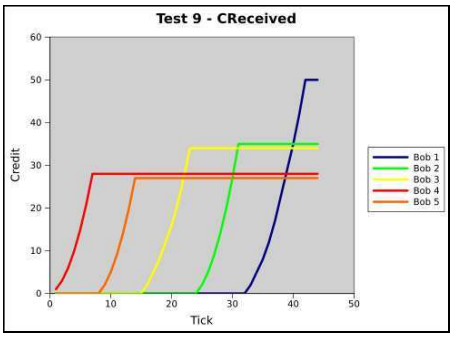


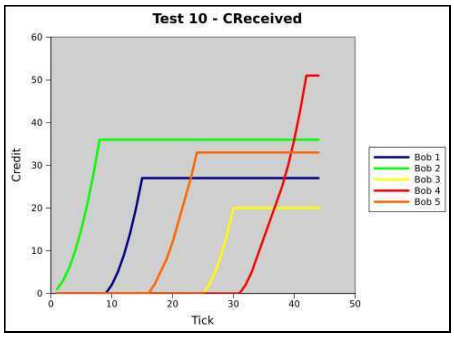
ing commission requirement mentioned above, their loss is minimised which is reflected in their final credit rating.

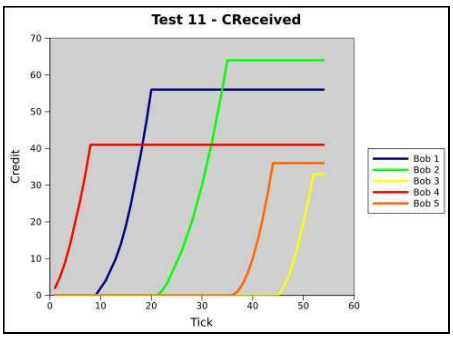
Alice's Claims and False Claims The results range from having 100% false claims in test 7 which gradually decrease to no false claims by tests 11 and 12. Again, the combination of `truthchance` and `malwarechance` restrict agents from making certain decisions. For example, in test 10, 60% of Carol's files are likely to be incorrect. Alice will make false claims 60% of the time, however, this only gives Alice the chance to make a false claim on the other 40% of Carol's files. Even though Alice and Carol are both abusing the trust* model in this scenario, the more malware that Carol serves, the less chance that Alice will falsely claim and vice versa. This is why the guarantors need to make new risk assessments between each protocol run to prevent personal loss for them.

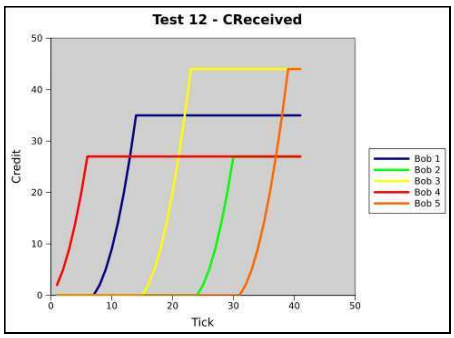

Test 7 - CReceived
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

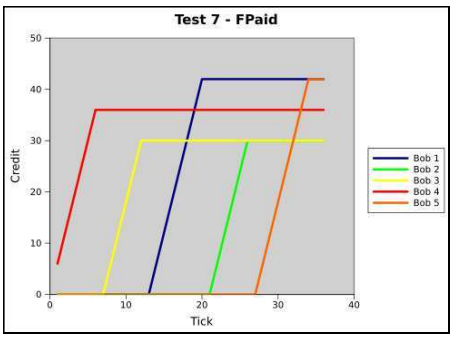

Test 8 - CReceived
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

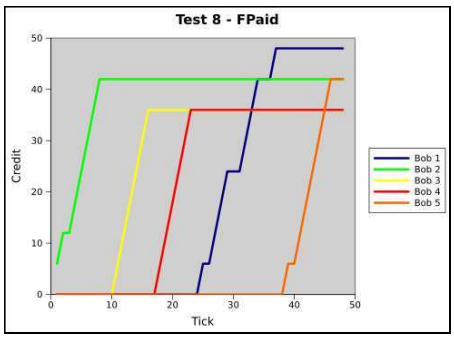

Test 9 - CReceived
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

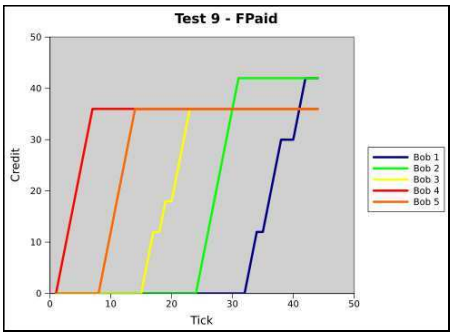

Test 10 - CReceived
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

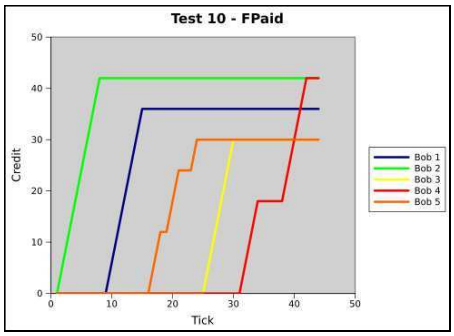

Test 11 - CReceived
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

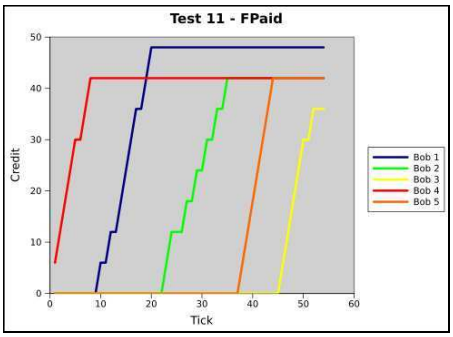

Test 12 - CReceived
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

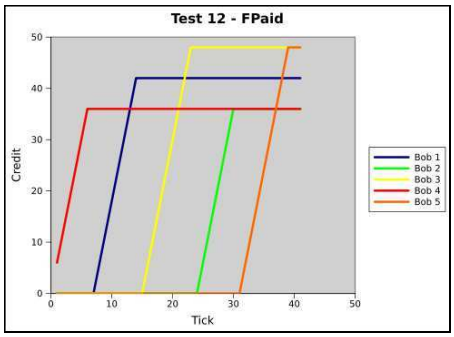

Test 7 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

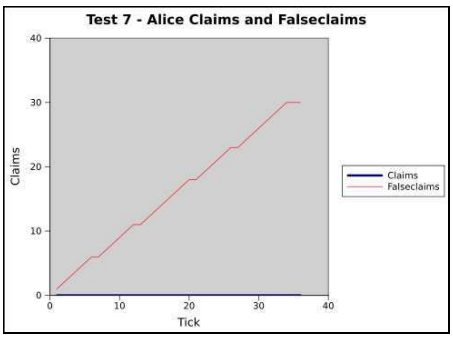

Test 8 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

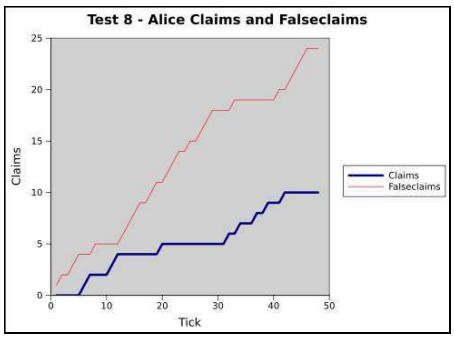

Test 9 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

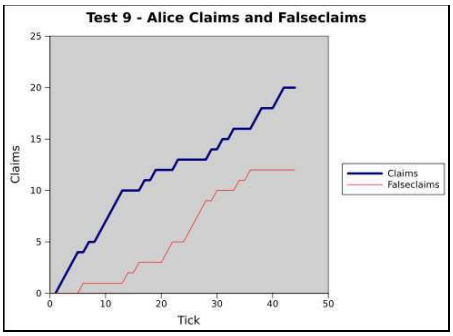

Test 10 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

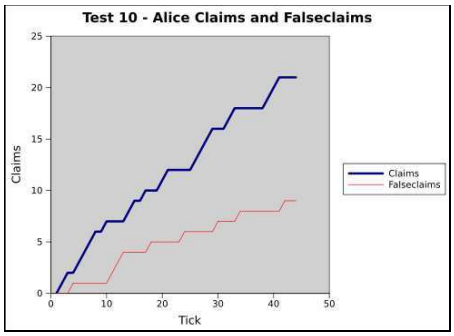

Test 11 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

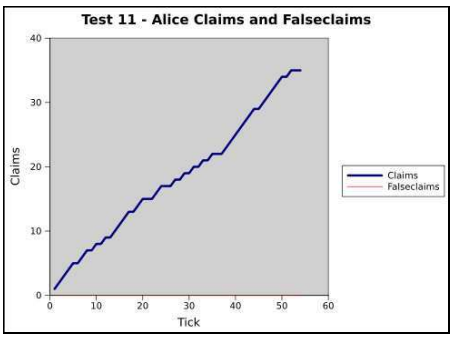

Test 12 - FPaid
Credit
Tick
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5

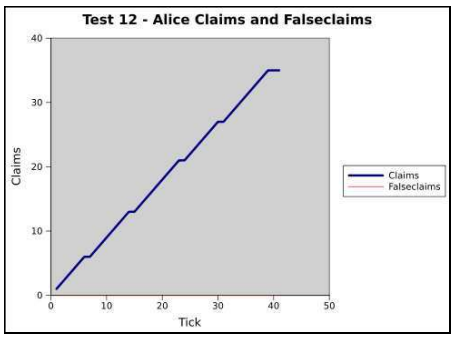

Test 7 - Alice Claims and Falseclaims
Claims
Tick
Claims
Falseclaims


Test 8 - Alice Claims and Falseclaims
Claims
Tick
Claims
Falseclaims







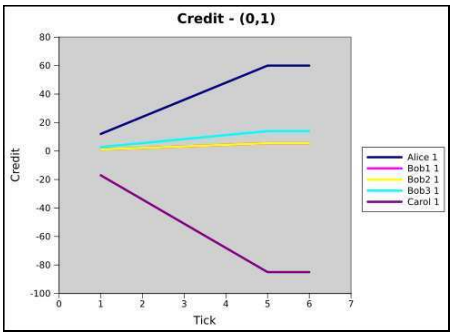


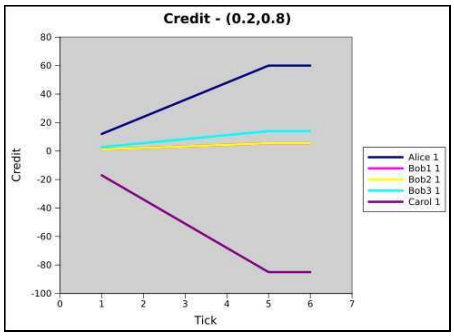
A.3 P2P Multiple Guarantor Tests

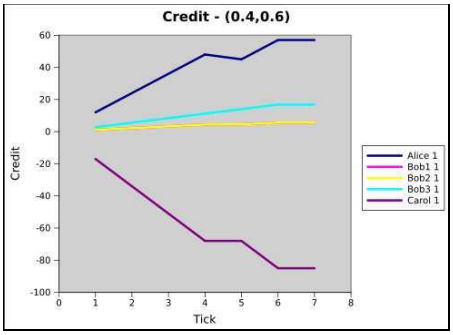
A.3.1 Test 1

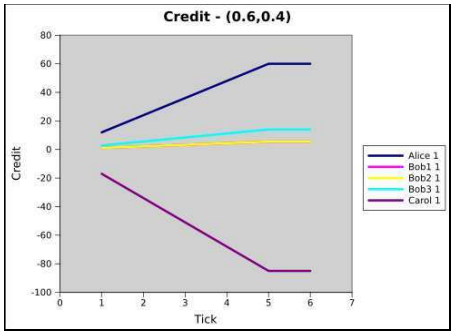
The results show that in Test 1, the trust* chain is much more volatile than a shorter chain as it has more possible points of failure. Having a longer chain also amplifies the effects such as price increases. For example, a price increase towards the end of the chain will cause all previous guarantors to increase their prices. As each guarantor is likely to increase their price slightly to cover their losses, it will lead to a much higher premium for Alice. Thus quickly making the cost of buying a guarantee unfeasible for Alice. The effects of this are more prominent in this simulation as guarantors have different rates depending on whether they are buying or providing a guarantee. For each guarantee that is claimed, the price charged by a guarantor will increase but he is also likely to decrease the price he is willing to pay for a guarantee. This will lead to more links being broken in a chain as prices gradually reach the limits of other principals.

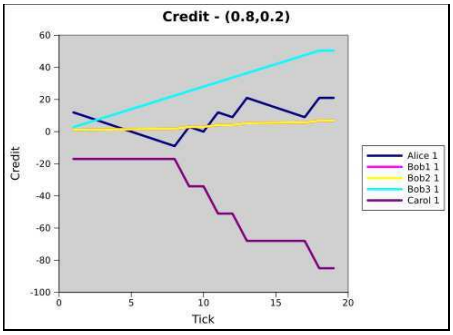
The graphs in this section reflect similar results to the corresponding tests in the short chain simulation above. However, in this scenario, Alice only has one route to Carol rather than five. It only takes a single guarantor to become inactive for the chain to be open. Of course, in reality, a diversion can be taken around the inactive link. This explains why the simulation tick counts here are roughly one fifth of that of the previous simulation.

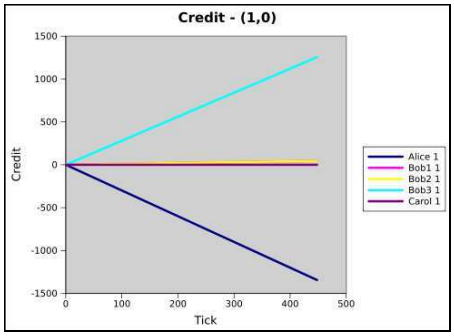


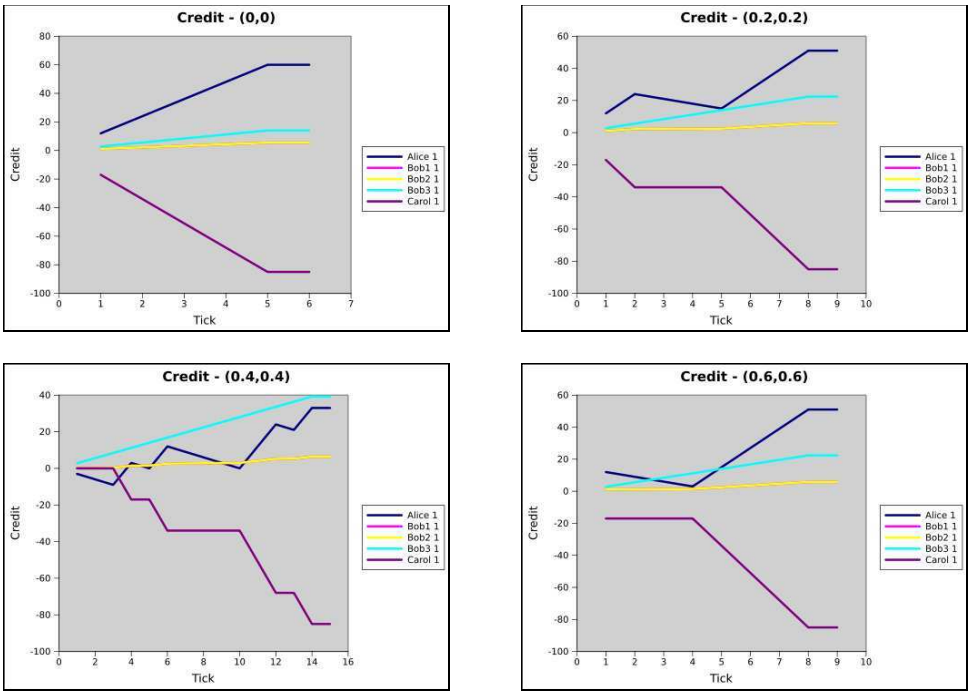










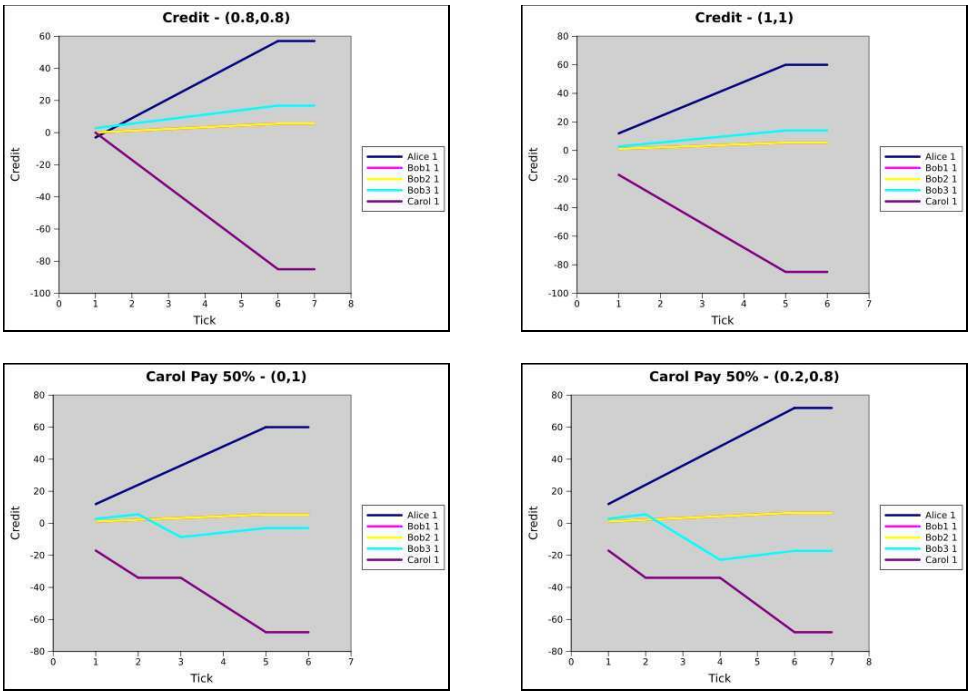


A.3.2 Test 2

As mentioned in Chapter 4, these tests follow the same combinations of the `truthchance` and `malwarechance` attributes but assume that Carol will only reimburse B_3 with the forfeit 50% of the time. B_3 counts the number of times Carol refuses to pay the forfeit and becomes inactive after three non-payments.

The results exhibit the same features as before, however, it is only B_3 who suffers greatly. However, this is assuming that he always pays the forfeit to B_2 who always pays B_1 etc. In reality, there might be non-payments between any local relationship in a chain. Principals will reconsider how many times they will tolerate non-payment from another and how much they will charge for future guarantees of them.

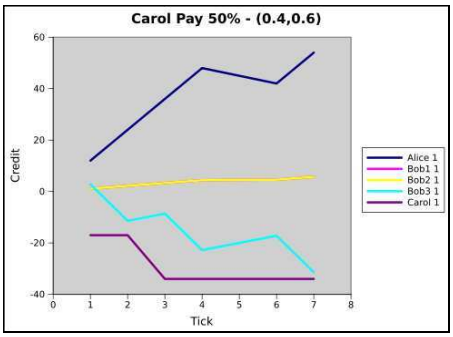
This test could have been applied between other locally trusted principals in a chain. However, the same effects would have been seen on the principal expecting to be reimbursed. This test shows that non-payments only effect locally trusting principals and that it is a locally solved problem. For example, if a principal whom you trust never reimburses a forfeit payment, you simply stop trusting them.

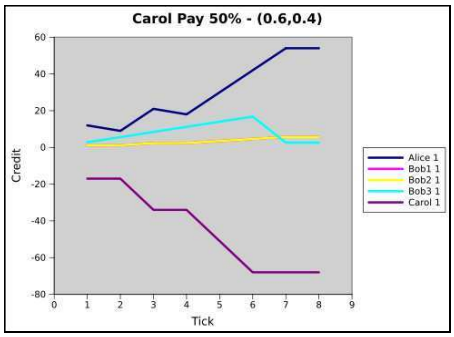


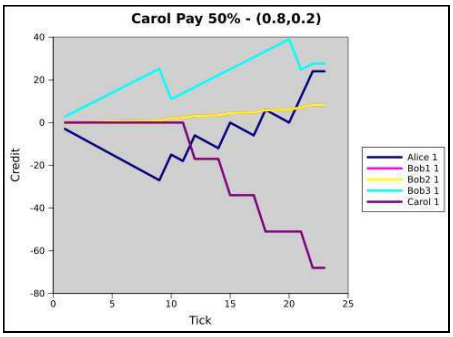
A.3.3 Test 3

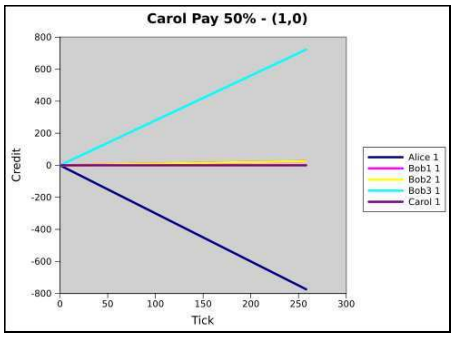
In these tests, both end-points (Alice and Carol) are fixed to being "good". In other words, Alice never has reason to claim a guarantee and never makes false claims. The guarantors however will falsely claim the agreed forfeit from their neighbour at the probability defined by `guartruthchance`.

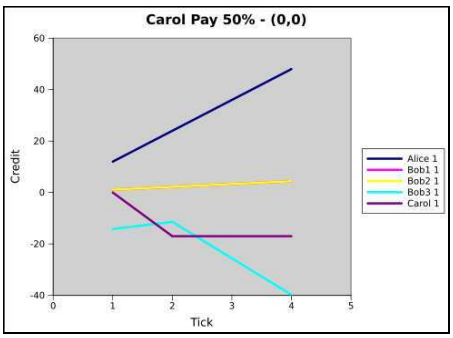
The results show that the guarantors (regardless of which ones made the false claim) always end up better off that Alice and Carol. This backs-up the case that Alice should choose whom she trusts carefully. Carol (or other guarantors) could also investigate a suspected false claim by initiating a cycle of trust* towards another (B_2 for example).

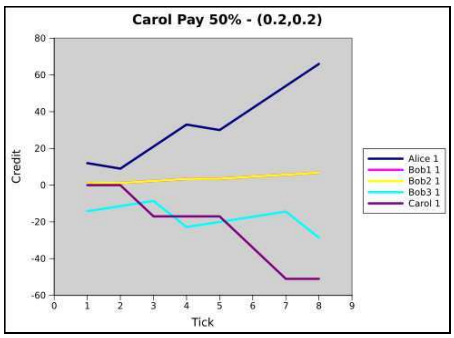

Carol Pay 50% - (0.4,0.6)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

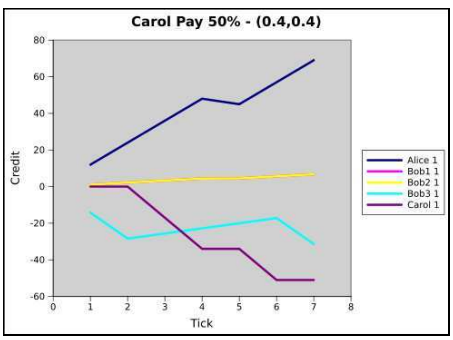

Carol Pay 50% - (0.6,0.4)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

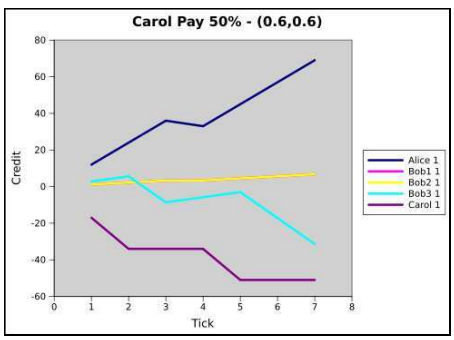

Carol Pay 50% - (0.8,0.2)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

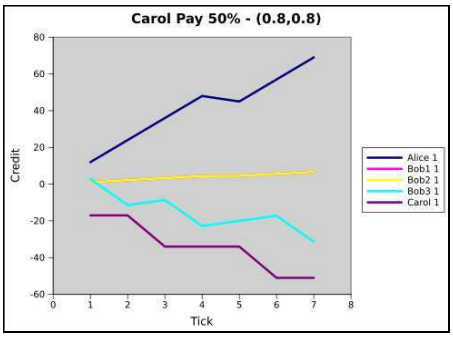

Carol Pay 50% - (1,0)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

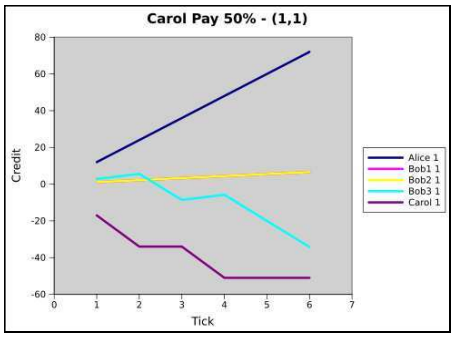

Carol Pay 50% - (0,0)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

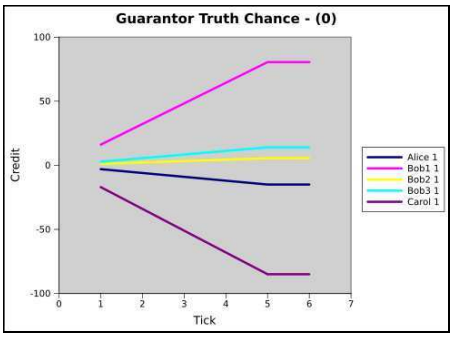

Carol Pay 50% - (0.2,0.2)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

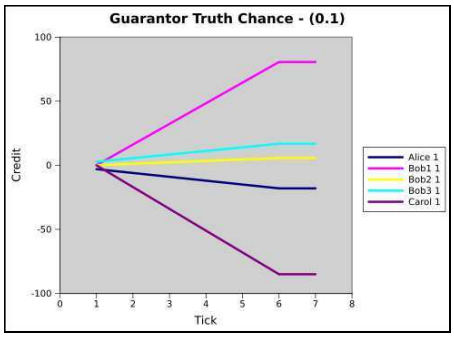

Carol Pay 50% - (0.4,0.4)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

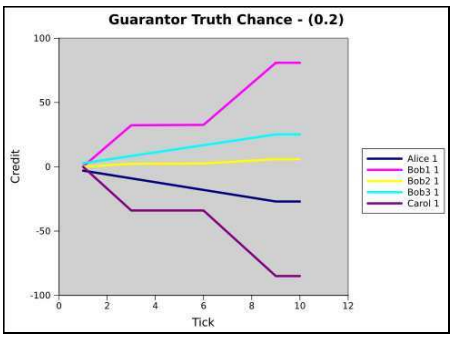

Carol Pay 50% - (0.6,0.6)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

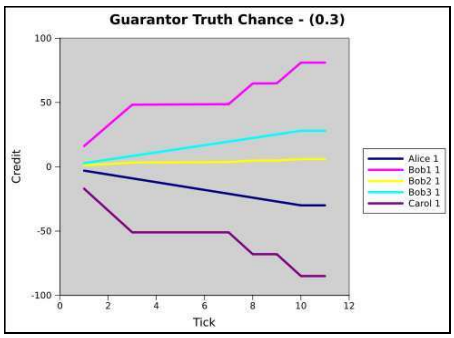

Carol Pay 50% - (0.8,0.8)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

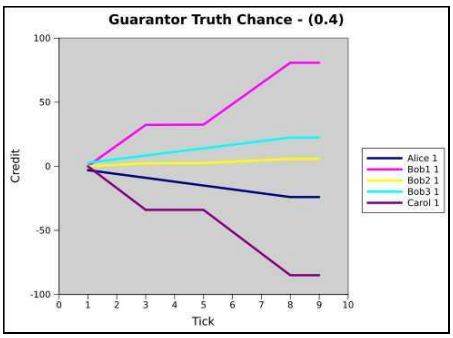

Carol Pay 50% - (1,1)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

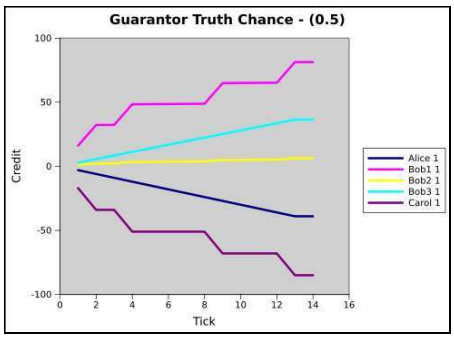

Guarantor Truth Chance - (0)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

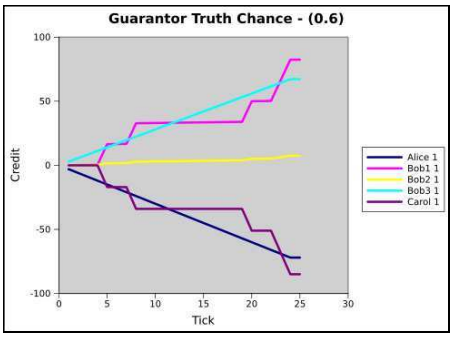

Guarantor Truth Chance - (0.1)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

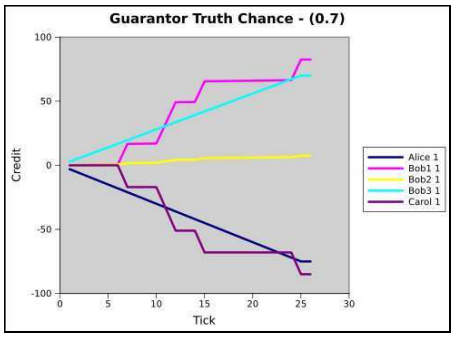

Guarantor Truth Chance - (0.2)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

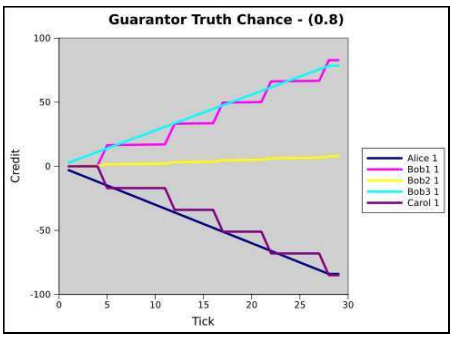

Guarantor Truth Chance - (0.3)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

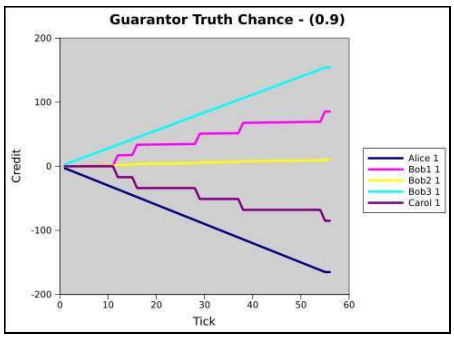

Guarantor Truth Chance - (0.4)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1


Guarantor Truth Chance - (0.5)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1


Guarantor Truth Chance - (0.6)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1


Guarantor Truth Chance - (0.7)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1


Guarantor Truth Chance - (0.8)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1


Guarantor Truth Chance - (0.9)
Credit
Tick
Alice 1
Bob1 1
Bob2 1
Bob3 1
Carol 1

A.4 Grid Computing Simulation

This section presents the results from the grid computing simulation. This simulation is similar to the P2P simulations except that Carol is now obliged to reimburse the forfeit to a guarantor due to a SLA being in place. For this reason, principals in this simulation are far less tolerant to non-payments from Carol than they were in the P2P simulation. The purpose of this simulation was to view the effects of non-payment from Carol.

Table A.1 outlines the combinations of the probability attributes that make up the 22 tests in this simulation. The initial values of agent properties are the same as the P2P simulation. `defaultchance` is the chance that Carol will default, `truthchance` is the chance that Alice will tell the truth about a computation, and `carolpaychance` is the chance that Carol will pay a forfeit when requested to do so.

A.4.1 Tests 1–6

In these tests, Alice is always truthful and Carol always pays the forfeit if required by a guarantor. However, Carol starts by defaulting 100% of the time and gradually defaults less often until she never defaults.

From the results, the length of a simulation can be seen to increase as Carol starts to default less often. When Carol is at her worst (tests 1–3), Alice always makes a profit. As Carol defaults less, Alice receives less forfeit payments, but is still paying for guarantees so makes an overall loss. The guarantors always profit from commission payments as any forfeits they might pay are always reimbursed. Carol always makes a loss except in test 6 when she never defaults. Test 6 was manually stopped as it would continue indefinitely.

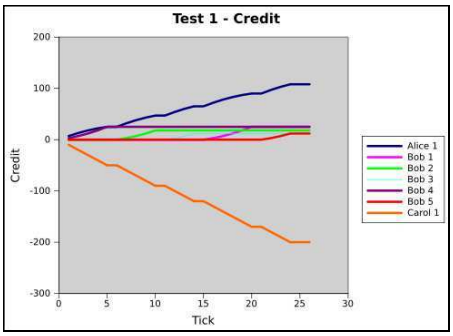
These tests have shown that it is in Carol's best interests to provide a good service as she will suffer the most.

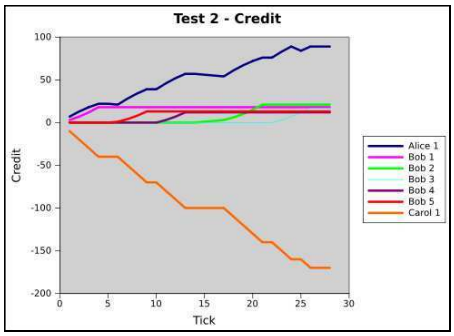
Even though Alice is being paid forfeits for bad service from Carol, she might be less tolerant depending on how critical her application is. In reality, regardless of whether she is compensated, she might stop using Carol's service if the default frequency is too high.

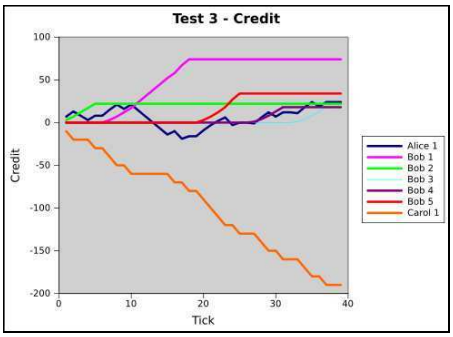
Test	Defaultchance	Truthchance	Carolpaychance
1	1	1	1
2	0.8	1	1
3	0.6	1	1
4	0.4	1	1
5	0.2	1	1
6	0	1	1
7	1	1	0.8
8	1	1	0.6
9	1	1	0.4
10	1	1	0.2
11	1	1	0
12	0	0.8	1
13	0	0.6	1
14	0	0.4	1
15	0	0.2	1
16	0	0	1
17	0	1	0
18	0	0.8	0
19	0	0.6	0
20	0	0.4	0
21	0	0.2	0
22	0	0	0

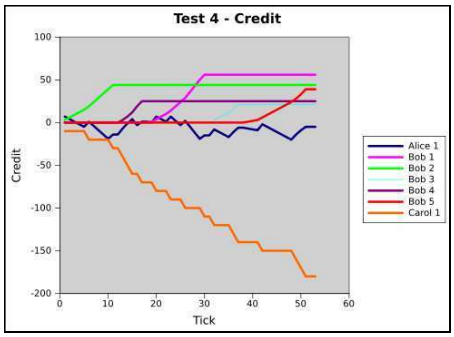
Table A.1: Grid simulation test setup.

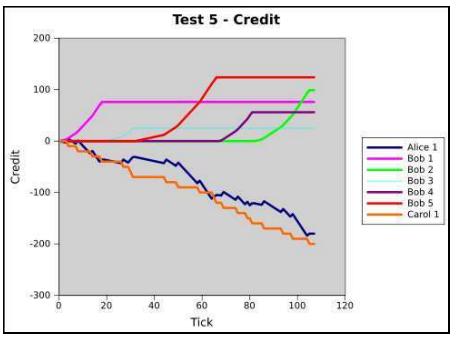
Guarantors will always be happy to provide a guarantee as long as Carol is reimbursing their losses.

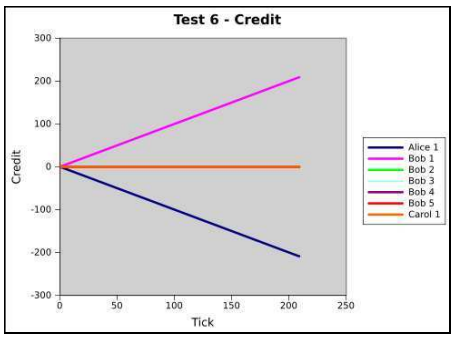

Test 1 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1


Test 2 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1


Test 3 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1


Test 4 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1

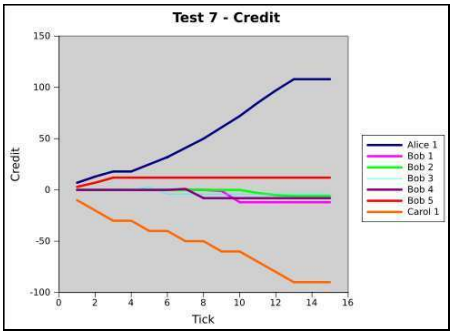

Test 5 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1

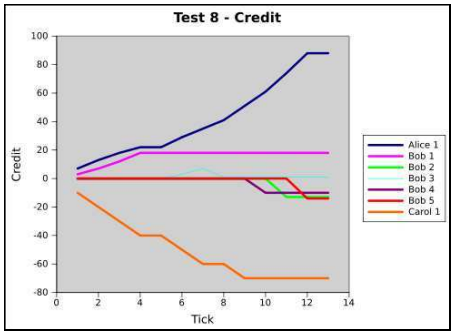

Test 6 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1

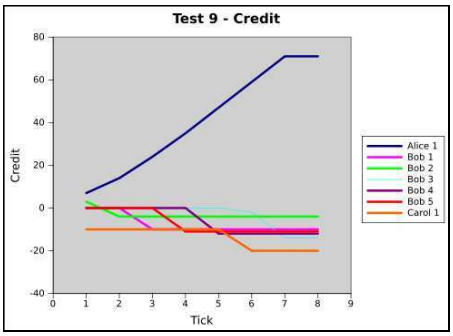
A.4.2 Tests 7–11

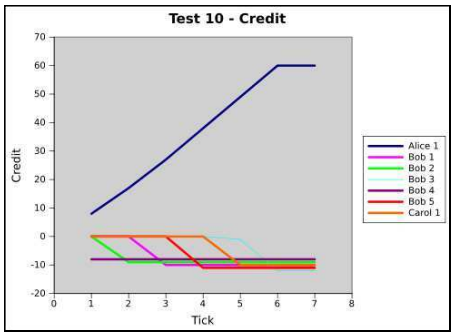
In these tests, Carol now always defaults and Alice is still always truthful (as she'll have no reason to make a false claim). However, the chance that Carol will pay the forfeit to Bob starts at 80% and decrements by 20% in each test until Carol never pays the forfeit.

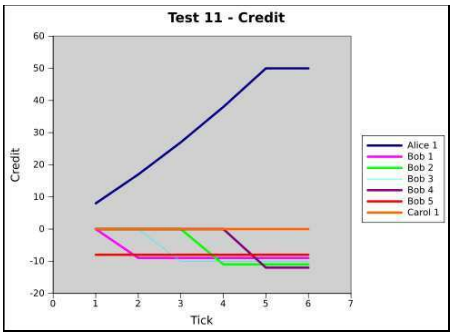
As the guarantors only tolerate one non-payment from Carol before they become inactive, the length of these simulations is reflected accordingly. The guarantors nearly always make a small loss from this single non-payment as they still pay the forfeit to Alice. Due to this, Alice always profits in these tests.

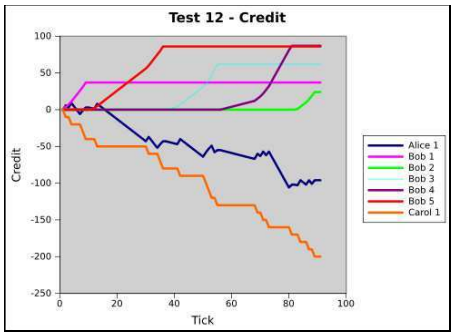










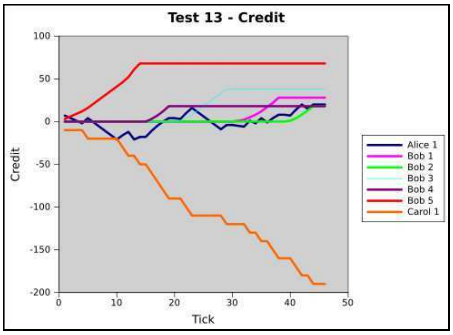


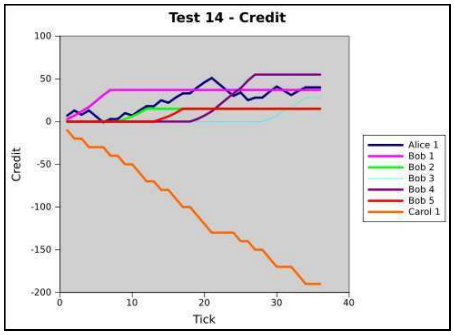
A.4.3 Tests 12–16

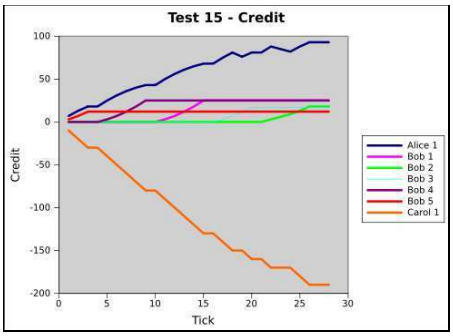
In these tests, Carol never defaults and always pays the forfeit if required. However, Alice starts by being 80% truthful which decrements by 20% in each test until she's never truthful when claiming.

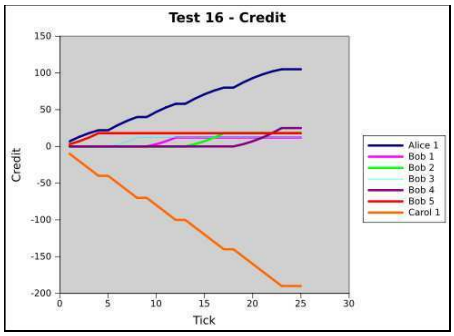
The results show that the more that Alice falsely claims, the shorter the simulation will run for. This is because Carol is effectively taking the blame and hence losing trust from guarantors. Although Carol is reimbursing the forfeit, the guarantors still register a claim and alter their rates accordingly. Eventually, they won't provide a guarantee of Carol.

Carol always makes a loss but the guarantors profit. Alice starts to make a profit when she becomes more untruthful, although it is short-lived. In reality, Carol (or the guarantors) would invoke a cycle of trust* immediately after they suspect that false claims are being made.









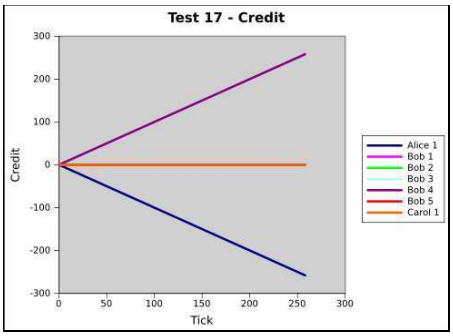
A.4.4 Tests 17–22

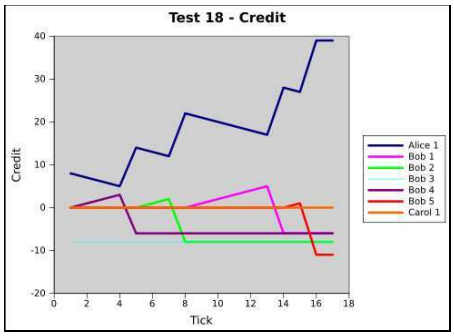
In these tests, Carol never defaults and never pays the forfeit if required by a guarantor. Alice starts by being 100% truthful which decreases to never being truthful.

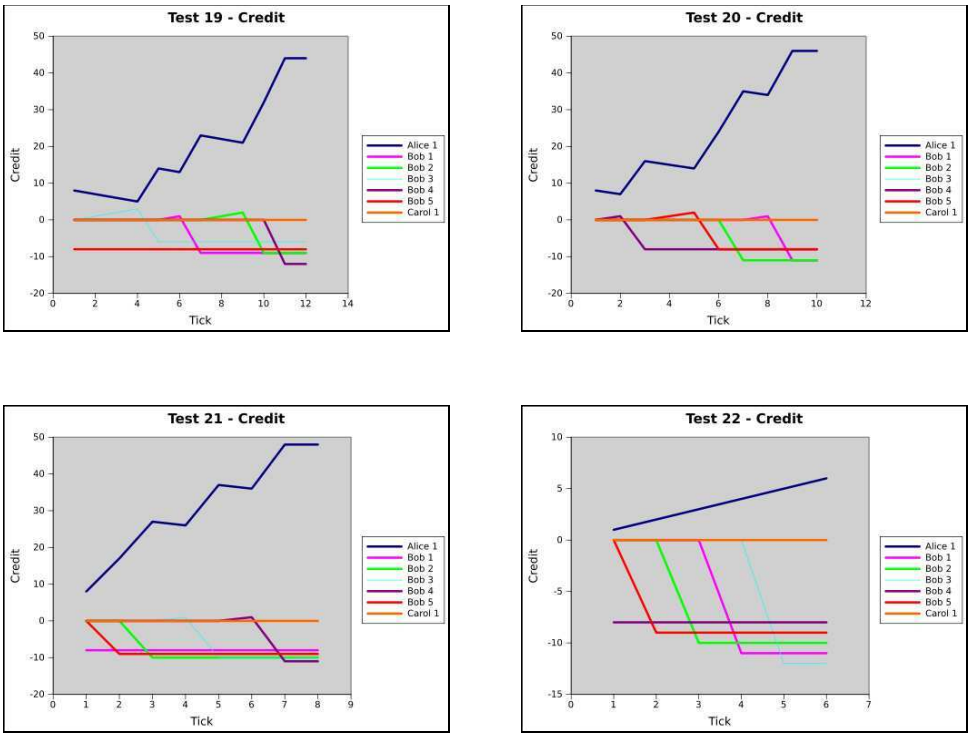
Test 17 was stopped manually as it would continue indefinitely. As Carol never defaults and Alice is always truthful, only one guarantor is used. This guarantor makes a steady profit from commission that Alice has paid and has never needed to pay a forfeit.

From test 18, a dramatic decrease in simulation run time can be seen. By test 22, a simulation only runs for 6 ticks. Here, Alice always profits from the forfeits honoured by guarantors. The guarantors make a small loss from the non-payment from Carol. Carol's credit remains unchanged in these tests as she refuses to pay any forfeits.

These results show that guarantors won't tolerate claims when Carol doesn't reimburse the forfeit. Carol is right to not reimburse the forfeit as she never provides a bad service. Alice is effectively destroying possible trust* routes between herself and Carol.







A.5 Click-through Licensing Simulation

Table A.2 shows the test combinations for the click-through simulation. The `defaultchance` and `carolpaychance` attributes have the same meaning as the previous simulation. However, now we use the `davidtruthchance` attribute to define how truthful David will be when answering queries. We assume that Alice is always truthful when claiming as this has already been tested in previous simulations. Also, she can make no immediate monetary gain from doing so. Again, the initial values of agents are the same as the P2P simulation.

A.5.1 Tests 1–6

In these tests, Carol never defaults but always pays. David starts by being 100% truthful which gradually reduces to 0%.

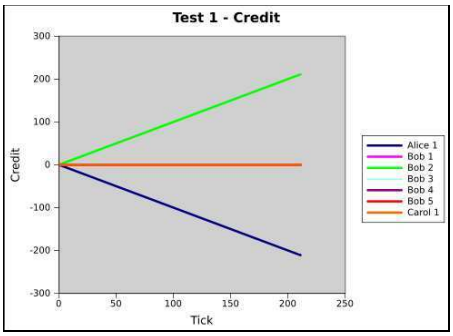
Test 1 needed to be manually stopped but would have continued as no claims were made because David was always truthful.

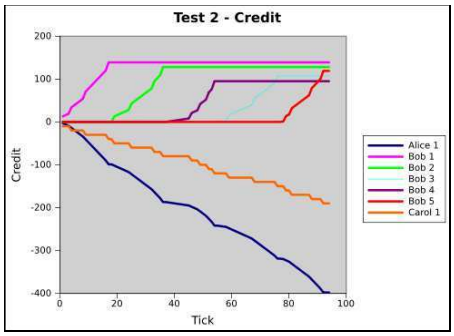
Test	Defaultchance	Davidtruthchance	Carolpaychance
1	0	1	1
2	0	0.8	1
3	0	0.6	1
4	0	0.4	1
5	0	0.2	1
6	0	0	1
7	1	1	1
8	1	1	0.8
9	1	1	0.6
10	1	1	0.4
11	1	1	0.2
12	1	1	0
13	0	1	0
14	0	0.8	0
15	0	0.6	0
16	0	0.4	0
17	0	0.2	0
18	0	0	0

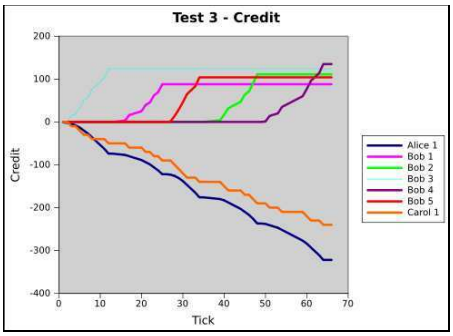
Table A.2: Click-through simulation test setup.

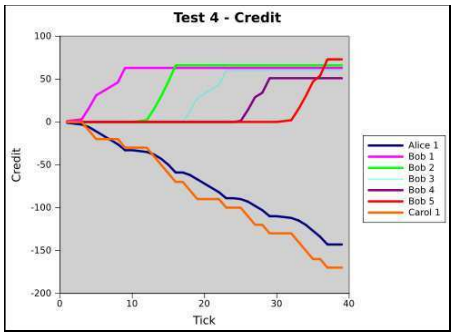
As David's truthfulness deteriorates, the guarantors become inactive quicker due to the increase in claims. However, the guarantors still profit from the commission from Alice as Carol always reimburses the cost of a legitimate licence.

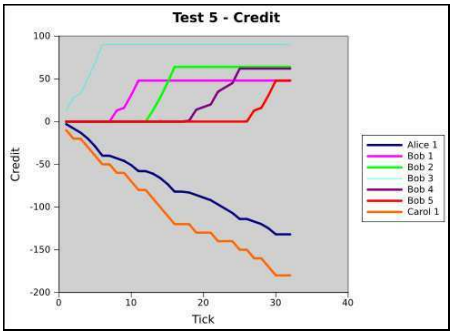
By being untruthful, these tests have shown that David will not gain in the long run as eventually all routes to Carol will be broken. Or she'll stop selling his software altogether.

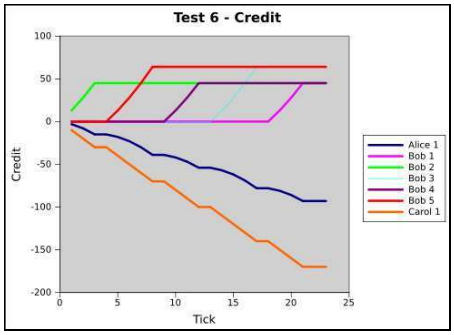










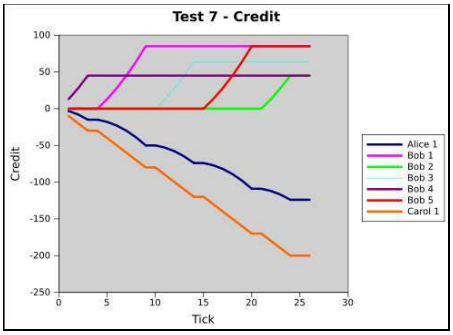


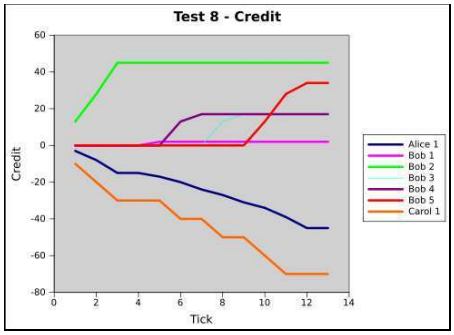
A.5.2 Tests 7–12

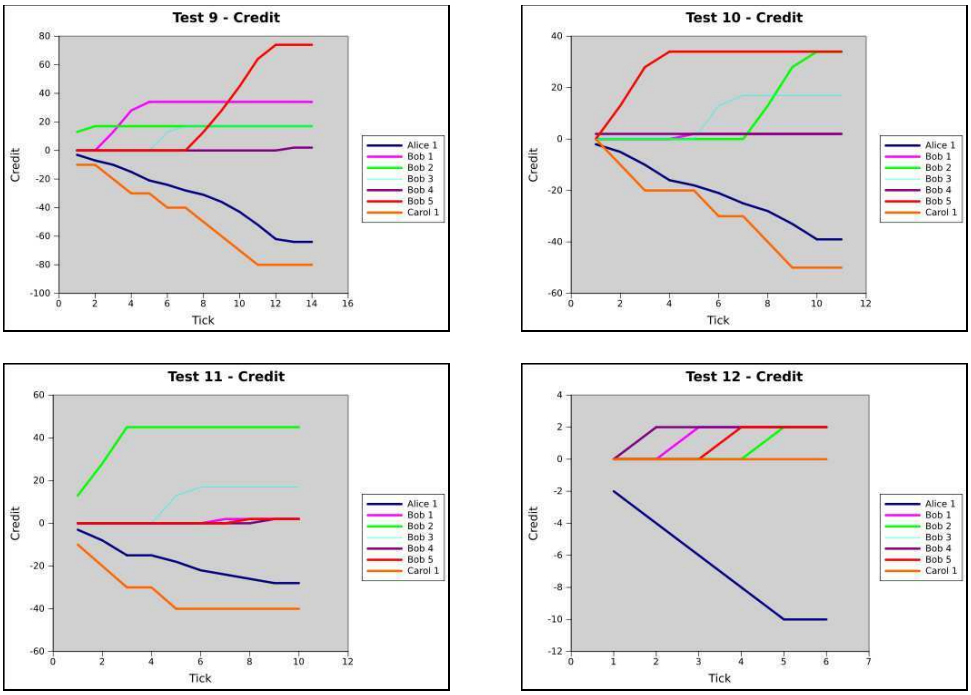
In these tests, Carol always defaults and David is now always truthful. The chances that Carol will reimburse a guarantor ranges from always to never.

The length of the simulation decreases from 26 to 6 in relation to the chance of Carol reimbursing Bob decreasing.

Alice suffers losses in these test whereas the guarantors make some profit. However, even in test 7, the simulation doesn't run for very long. Again, regardless of whether Carol reimburses the forfeit, the volume and frequency of claims if often more important causing routes to be broken between Alice and Carol.





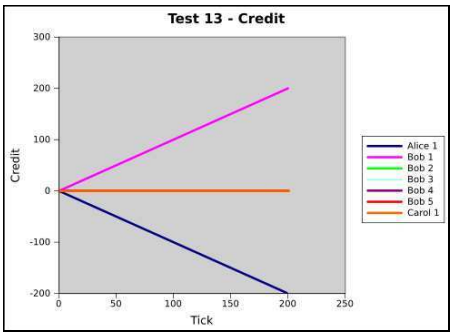


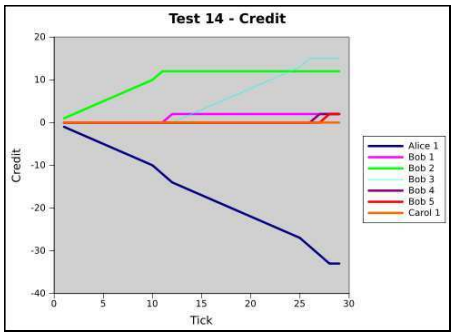
A.5.3 Tests 13–18

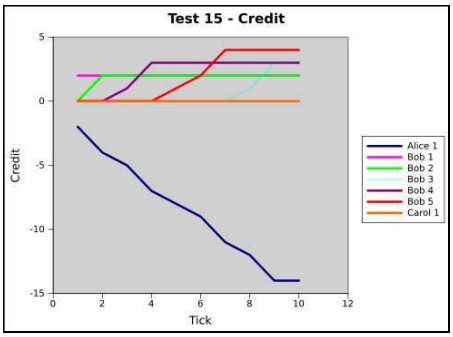
In these tests, Carol never defaults but also never pays any forfeits to the guarantors. David starts by always being truthful in test 13 until he is never truthful by test 18.

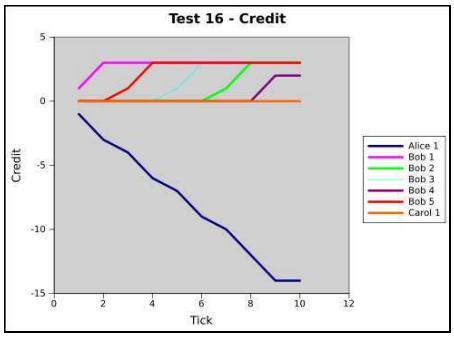
Test 13 needed to be manually stopped but the following tests decreased in run time as the routes became exhausted faster. In these tests, the guarantors make very small profits and Alice makes small losses.

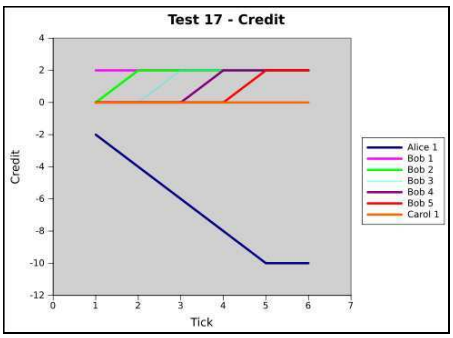
Again, the results show that the truthfulness of David affects trust* routes to vendors of his software. This is not in his best interests as he will lose sales of his software in the long run.

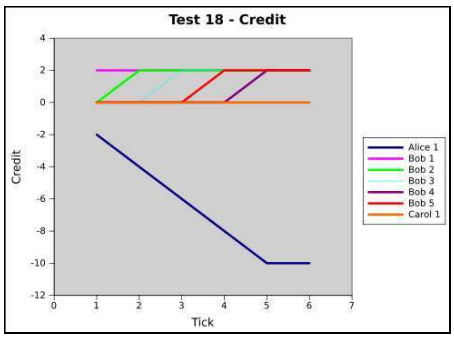

Test 13 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1


Test 14 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1


Test 15 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1


Test 16 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1


Test 17 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1


Test 18 - Credit
Credit
Tick
Alice 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Carol 1

A.6 Spam-proof Simulation

This section presents the results from the spam-proof simulation. Note that the `spamlevel` attribute is fixed to 1 and the `spamaccept` attribute is fixed to 0. These attributes are for modelling spam perceptions and tolerance and have been fixed to reduce the number of required tests. Table A.3 outlines the attribute combinations for the spam-proof tests. The attributes used are `spamchance` which defines the chance that Carol will send spam, `guartruthchance` defines the truthfulness of a guarantor, and `rectruthchance` defines how truthful Alice is when deciding whether email is spam or not. Initial values for agents are given in Table A.4. The change in initial values (and their possible ranges when randomly generated) has increased the tolerance of bad behaviour for all principals. This is to allow the simulations to run for a longer period of time.

Test	Spamchance	Guartruthchance	Rectruthchance
1	1	1	1
2	0.8	1	1
3	0.6	1	1
4	0.4	1	1
5	0.2	1	1
6	0	1	1
7	0.8	0.8	1
8	0.6	0.6	1
9	0.4	0.4	1
10	0.2	0.2	1
11	0	0	1
12	0.8	1	0.8
13	0.6	1	0.6
14	0.4	1	0.4
15	0.2	1	0.2
16	0	1	0

Table A.3: Spam-proof simulation test setup.

	Carol	Bob	Alice
cOffer	3	2.9	n/a
fOffer	50	49	n/a
cMin	n/a	1–5	n/a
cMax	5–10	∞	n/a
fMin	n/a	25–75	25–75
fMax	50–100	∞	∞

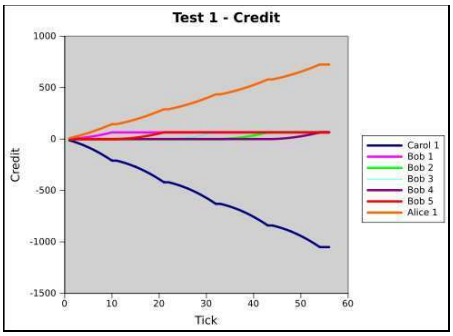
Table A.4: Initial values for the spam-proof simulation.

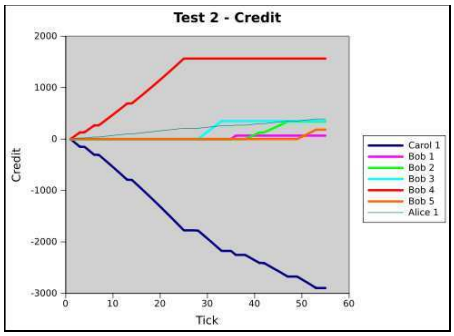
A.6.1 Tests 1–6

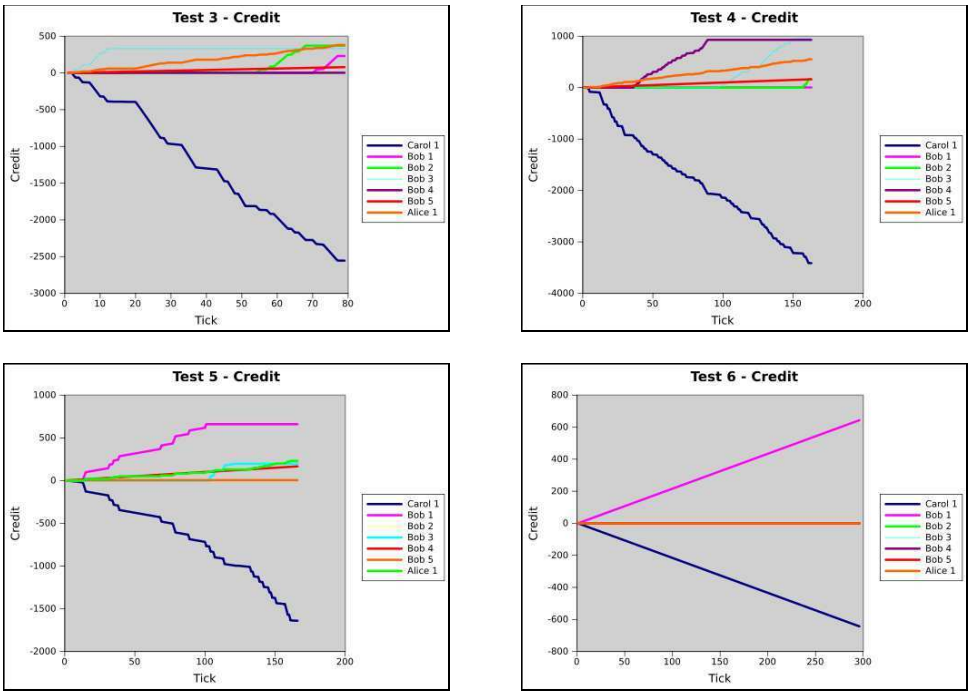
In these tests, the guarantors and the email receiver (Alice) are always truthful about making claims. However, the probability that Carol will send spam email starts at 100% and gradually decreases until she never sends spam.

The length of a simulation increases in relation to the decrease in spam. By test 6, no spam has been sent and as Alice and Bob are both 100% truthful, the simulation continues via a single guarantor until manually stopped.

When Carol is sending lots of spam, she makes quite considerable losses while the guarantors and Alice make profits. This shows that over time, only spammers will be penalised.





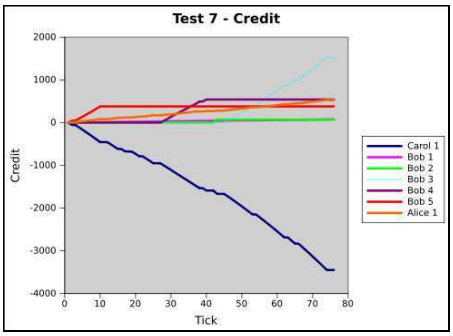


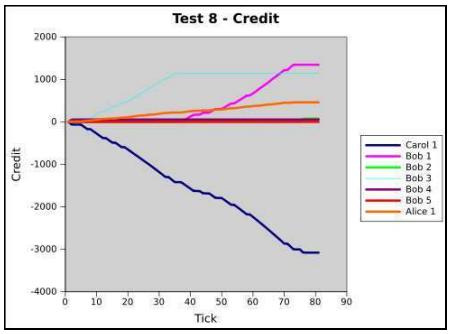
A.6.2 Tests 7–11

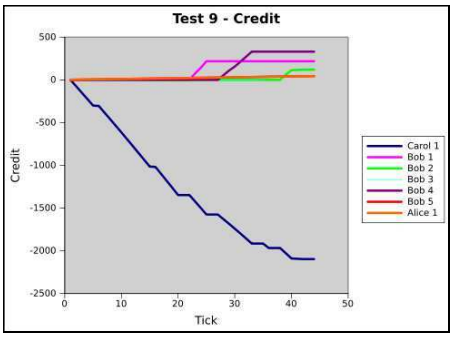
In these tests, the chance that Carol will send spam starts at 80% and decreases to 0% of the time. The receiver is always truthful, however, the guarantors start by being truthful 80% of the time which gradually decreases to 0% of the time.

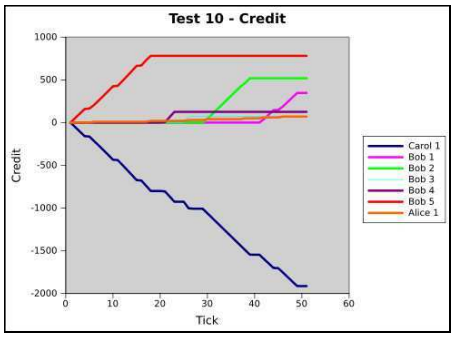
It is only Carol who suffers a loss in these tests even when she never sends spam. This is because a guarantor's chance of being untruthful increases as Carol's spam chance decreases.

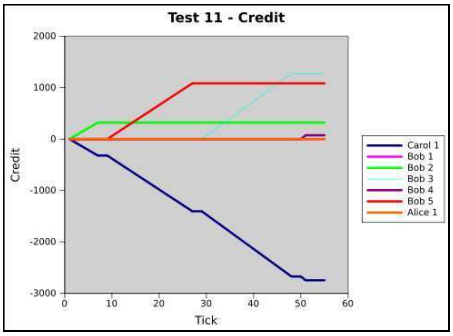
The results show that the guarantors can make some profit from making false claims before Carol refuses to reimburse the forfeit. This is a short-lived gain and using trust* truthfully will be more beneficial in the long run. In reality, Carol would've investigated such false claims by invoking cycles of trust*. Or simply try other routes to Alice to see if claims are still being made.

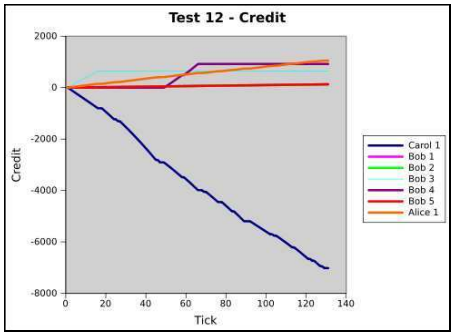

Test 7 - Credit
Credit
Tick
Carol 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Alice 1


Test 8 - Credit
Credit
Tick
Carol 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Alice 1


Test 9 - Credit
Credit
Tick
Carol 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Alice 1


Test 10 - Credit
Credit
Tick
Carol 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Alice 1

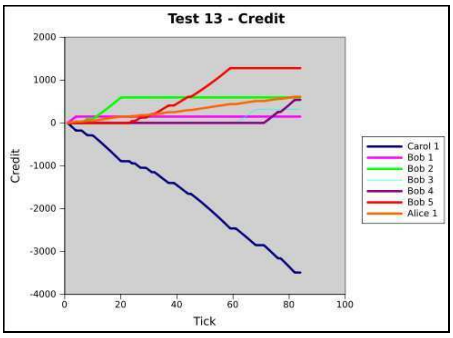

Test 11 - Credit
Credit
Tick
Carol 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Alice 1

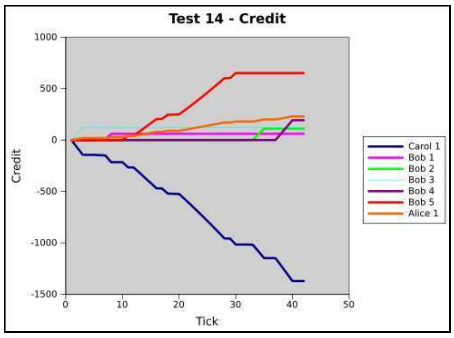

Test 12 - Credit
Credit
Tick
Carol 1
Bob 1
Bob 2
Bob 3
Bob 4
Bob 5
Alice 1

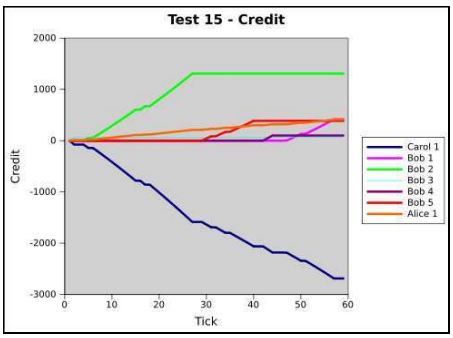
A.6.3 Tests 12–16

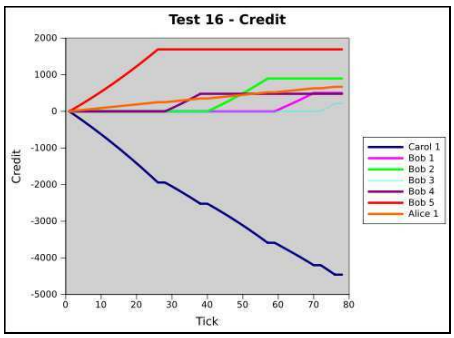
In these tests, the chance of spam ranges from 80% to 0%. However, this time the guarantors are always truthful but now the truthfulness of the receiver ranges from 80% to 0%.

These results reflect a similar pattern to those in tests 7–11. They show that it wouldn't be sensible for Alice to make false claims as she will only be damaging future possible trust* routes to herself.









Appendix B

Trust* KeyNote Implementation

B.1 Introduction

This appendix provides an example of how the trust* protocol could be followed by using the KeyNote trust management toolkit as the core decision maker and to also provide the micro-payment mechanism. The example will follow the protocol of the spam-proof email application previously described in Chapter 7 and will use monetary micro-payments for the commission and forfeits.

B.2 A Spam-proof KeyNote Implementation

This section shows how KeyNote might be used to negotiate a trust* relationship between Bob and Carol. Bob wishes to send an email to Carol which is guaranteed that it isn't spam by someone whom she trusts directly. This example is a good case scenario where there is at least one possible route. Figure B.1 shows a route between Bob and Carol via Gordon and Frank. Principals are identified by RSA public keys which have been replaced by names for the purposes of this example.

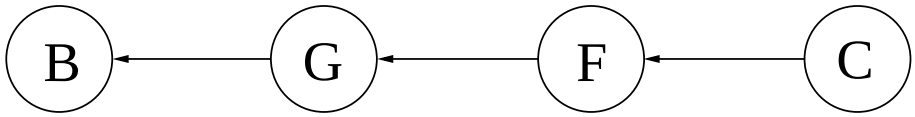


Figure B.1: A trust* route between Bob and Carol.

B.2.1 Multiple Guarantors

Some previous examples in this dissertation have assumed that only one guarantor is needed to extend trust between principals. This example uses two guarantors (Gordon and Frank) to extend trust between Bob and Carol. Carol knows that she is receiving a guarantee from Frank and Bob knows he is being guaranteed by Gordon. However, Bob and Carol don't necessarily know whether these are the same people or how many other guarantors are in the chain. Similarly, Gordon and Frank will transact with each other but are unaware of each others neighbours (whether they be Bob, Carol or another guarantor). This example could be extended to an infinite number of guarantors of which only need to know whom they are receiving a guarantee from or whom they are providing a guarantee to (or both).

B.2.2 Policies

Every principal will have one or more policy files which defines whom they trust and the forfeit and commission rates they are willing to accept from them. A principal might have more policies depending on whom they might be dealing with. For example, a policy might allow discounts for a set of close friends. In this vanilla example, each principal only has one policy which governs who they are willing to receive email guarantees from. Different policies might also be used depending on whether a guarantee is being provided or being received by a principal.

Figure B.2 is an example of Carol's policy (who will be the receiver of the guarantee). It allows her to be the recipient of a spam-proof guarantee from Frank with a minimum forfeit value. If these conditions are all met, KeyNote will return true about a particular guarantee. The forfeit rate is fictitious and might be pence or credit but this depends on what Frank and Carol have decided to deal in. Assume for this example the currency is pence.

A guarantor will be interested in a commission as well as a forfeit value. Figure B.3 is an example of Gordon's policy (who will be a guarantor).

```
Authorizer: "POLICY"
Licensees:  kFrank
Conditions: type == "spamproof" &&
            @forfeit >= 8 -> "true";
```

Figure B.2: Carol's policy.

```
Authorizer: "POLICY"
Licensees:  kBob
Conditions: type == "spamproof" &&
            @commission >= 1 &&
            @forfeit >= 8 -> "true";
```

Figure B.3: Gordon's policy.

B.2.3 Requests and Guarantee Credentials

Figure B.4 shows the direction of the requests and guarantees among the four participating principals and their respective `id` numbers.

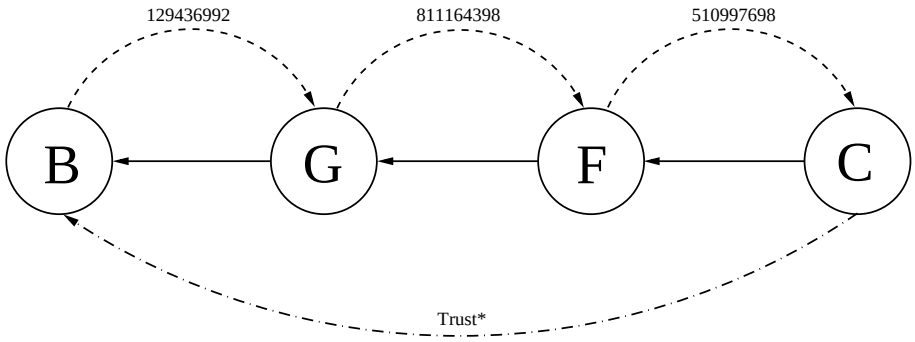


Figure B.4: Request and guarantee paths and id numbers.

Figure B.5 is an example of Bob's request to Gordon for him to act as a guarantor between himself and Carol.

Bob also sends a guarantee (Figure B.6) to Gordon stating the commission rate that he is willing to pay Gordon to forward the request and the forfeit he will pay if the email is considered spam by Carol. The `id` field is simply to link requests to their corresponding guarantees for future reference.

Gordon will forward the request to principals who trust him. Frank receives the request and the guarantee proposal from Gordon (Figure B.7). It is likely the

```
id = "129436992"
type = "spamproof"
from = "bob"
to = "carol"
forfeit = "8"
commission = "1"
```

Figure B.5: Bob's request.

```
KeyNote-Version: 2
Authorizer: kBob
Licensees:  kGordon
Conditions: id == "129436992" &&
            type == "spamproof" &&
            commission == "1" &&
            forfeit == "8" -> "true";
Signature: "sig-rsa-sha1-hex:7fbcde43..."
```

Figure B.6: Bob's guarantee.

commission and forfeit rates will differ from other guarantees if the chain. This way, Gordon can cover his loses if Bob was to default. However for this example, they'll stay the same.

```
KeyNote-Version: 2
Authorizer: kGordon
Licensees:  kFrank
Conditions: id == "811164398" &&
            type == "spamproof" &&
            commission == "1" &&
            forfeit == "8" -> "true";
Signature: "sig-rsa-sha1-hex:08d5406c..."
```

Figure B.7: Gordon's guarantee.

In the same manner, Carol will receive the request and guarantee from Frank minus the commission value and will decide whether to generate a token from the forfeit value and who the request was from. Carol trusts Frank and a minimum forfeit of 8 pence is allowed. Frank's guarantee is shown in Figure B.8

```
KeyNote-Version: 2
Authorizer: kFrank
Licensees:  kCarol
Conditions: id == "510997698" &&
            type == "spamproof" &&
            forfeit == "8" -> "true";
Signature: "sig-rsa-sha1-hex:0d7bc346..."
```

Figure B.8: Frank's guarantee.

B.2.4 Compliance Checking

All recipients of a guarantee will be able to pass the guarantee credential, the request, the relevant public keys, and the policy to their KeyNote compliance checker which will return an indication of whether to proceed or not. As Carol is the intended recipient of the email from Bob, she will now generate an email token for him and send it back to Frank. Frank will pass it back to Gordon and so on until it reaches Bob. It is this process that notifies each guarantor that their guarantee is now active.

It is the responsibility of each principal to verify any received guarantees before proceeding with the protocol. In a guarantor's case, to proceed would be to forward the request and guarantee. For Carol, this would be to generate the token. Once Bob has the token, he can send the email directly to Carol with the token embedded in the header of the email. Figure B.9 shows the direction of the token and the email.

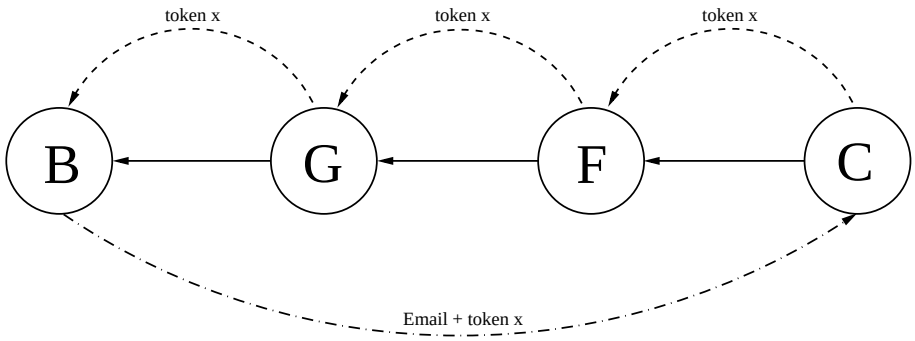


Figure B.9: Direction of the token and email.

B.2.5 After Sending the Email

The next part of the protocol involves Carol responding to Frank with either a notification that the email was not spam or a guarantee claim. Frank will reply with an acknowledgement or the forfeit payment respectively. Figure B.10 shows how this process continues until it reaches Bob.

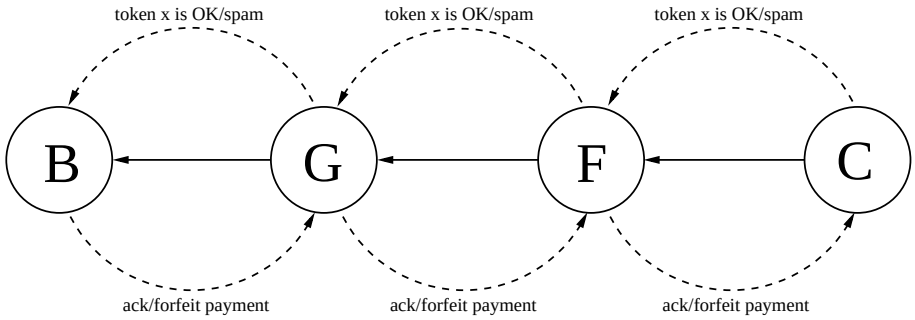


Figure B.10: Direction of responses and payments.

B.3 KeyNote Micro-payments

This section will give an example of how the payment mechanism might be implemented using KeyNote. It is based on a micro-payment scheme by Blaze *et al* [16] which allows off-line payments of very low values to take place. The example will show Bob's commission payment to Gordon.

A principal needing to receive micro-payments will need a policy that identifies the public keys of the Provisioning Agents (PAs) that are trusted to issue payer credentials. A PA can be seen as a bank who issue cheque books to their account holders. People are generally happy to receive a cheque from another if a well known bank is at the top of the cheque. When a cheque is presented to the bank, they will be obliged to honour the payment and debit the money from the payer's account. An example of Gordon's policy is given in Figure B.11.

This policy authorises any payments in the spam-proof application which are signed by any one of the three PAs[1] that Gordon trusts.

[1] || denotes OR.

```
Authorizer: "POLICY"
Licensees:  kPA1 || kPA2 || kPA3
Conditions: type == "spamproof" -> "true";
```

Figure B.11: Gordon's payment policy.

Principals needing to make payments are issued credentials periodically by their PA which specify the conditions under which their payment will be authorised. These credentials are signed by the issuing PA and a principal receiving a payment will be able to verify this against their policy. An example of Bob's payer credential is given in Figure B.12.

```
Authorizer: kPA1
Licensees:  kBob
Conditions: type == "spamproof" &&
            amount < "11" &&
            date < "20091031" -> "true";
Signature: "sig-rsa-sha1-hex:3eadb5e1..."
```

Figure B.12: Bob's payer credential.

This simplified credential will allow Bob to make payments related to the spam-proof application of up to 10 pence (for example) at a time until the expiry date. The number of possible transactions could also be limited but assume these credentials are issued daily and expire daily. This way the PA can do a regular risk assessment of Bob and alter the conditions accordingly.

When Gordon wants to take payment (this might be before or after a guarantee is forwarded depending on how much he trusts Bob) he will send Bob an invoice which is shown in Figure B.13.

```
merchant = "kGordon"
type = "spamproof"
date = "20091030"
amount = "1"
nonce = "e7bdf5dbee3b"
```

Figure B.13: Payment invoice.

Bob now generates a signed micro-payment credential and sends it along with his payer credential to Gordon. This credential is shown in Figure B.14.

```
Authorizer: kBob
Licensees:  kGordon
Conditions: type == "spamproof" &&
            amount == "1" &&
            date == "20091030" &&
            nonce == "e7bdf5dbee3b" -> "true";
Signature: "sig-rsa-sha1-hex:f7e5e22d..."
```

Figure B.14: Bob's micro-cheque.

Gordon sends these credentials, Bob's key, the invoice and his policy to his KeyNote compliance checker to determine whether he is likely to be paid (and whether to proceed). The compliance checker will verify the signatures, and check that the credentials link. It will also check that the conditions are met between the credentials and Gordon's policy. In the example given here, Gordon will have confirmation from KeyNote that he will be able to receive his 1 penny from PA1.

B.4 Conclusion

The point of using a trust management system like KeyNote is so that negotiations of trust* routes can be made fairly autonomously. Policy files state the principals you trust and the minimum rates you will accept from them. The compliance checker could automatically be invoked when a guarantee or request is received (say by building trust* functionality into an email client).

This example has assumed that the trust and payment mechanisms used within the local trust relationships are homogeneous. KeyNote is used between Bob and Gordon, Gordon and Frank, and Frank and Carol as the mechanism to follow the protocol. In reality, as discussed in Chapter 5, the specific mechanism used is more likely to be heterogeneous between principals. Maybe Bob and Gordon use KeyNote but Frank and Carol might use another mechanism. The same applies to the payments involved in a trust* protocol run.

Appendix C

Publications

This appendix includes two refereed publications that are a product of this work. The first introduces the trust* model and formed the basis of the spam-proof application described in Chapter 7 and was presented at the Security Protocols Workshop in Cambridge, UK. The second formed the basis of the P2P application described in Chapter 3 which was presented at ADBIS in Riga, Latvia. Below are their respective bibliography entries:

Stephen Clarke, Bruce Christianson, and Hannan Xiao. Trust*: Using Local Guarantees to Extend the Reach of Trust. In *Proceedings of the Seventeenth International Workshop on Security Protocols*, Cambridge, UK, April 2009.

Stephen Clarke, Bruce Christianson, and Hannan Xiao. Extending Trust in Peer-to-peer Networks. In *Proceedings of the Thirteenth East-European Conference on Advances in Databases and Information Systems*, Riga, Latvia, September 2009.

Trust*: Using Local Guarantees to Extend the Reach of Trust

Stephen Clarke Bruce Christianson Hannan Xiao

Abstract

We propose a new concept called trust* as a way of avoiding the necessity to transitively trust others in a range of distributed environments. The trust* approach uses guarantees based upon already established trust relationships. These localised guarantees are then used to extend trust to a new relationship (which we call trust*) which can hold between principals which are unknown to and do not trust one another. Such chains of guarantees enable the risk involved to be shifted to another party (in a similar way to real world guarantees). If a guarantee is broken, some kind of 'forfeit' is imposed, either to compensate the client or to deter the server from doing it habitually. Due to trust (and hence also forfeits) being localised, the specific micro-payment and trust management mechanisms that are used to implement the protocol can be heterogeneous. This paper describes the concept of trust* and some possible applications within a domain where the service being provided is also electronic.

1 Building on Trust

Building trust on the Internet is a well researched area. Many solutions assume (often implicitly) that trust is transitive. Commonly used examples are reputation systems where each of its users has a reputation rating. These ratings can be viewed by other users and later increased or decreased depending on the outcome of a transaction. Such reputation systems are commonly used on the Internet for various purposes and generally work well. However, as mentioned, reputation systems have a vital flaw; they imply that trust is transitive [8, 7]. Assume a user wants to determine the risk involved if they were to trust another (eg. to provide a described service) by looking at their reputation rating. This might contain comments and ratings left from previous transactions. It is unlikely that the user looking knows (or trusts) the other users who have left the comments. But even

if they *do* know and trust the people who left the comments, they will still be transitively trusting the service provider in question.

The motivation behind this work is to find a new way of building on trust which avoids this need for transitivity. The ability to build trust in the real world is also a common necessity. In real world protocols, this ability is often facilitated by using a guarantor as a replacement for transitivity of trust. Guarantees work by shifting the risk to another party thus lowering the risk for the trusting party.

Trust* is based on the electronic equivalent of the real world guarantee solution. Say that Carol needs to trust Alice about something and doesn't personally know or trust Alice. However, Carol trusts Bob who in turn trusts Alice to do whatever it is Carol needs her to do. In order to change Carol's perception of the risk involved, Bob could guarantee to Carol that Alice will act as intended and offer Carol compensation if Alice doesn't. So, what's Bob's incentive to act as a broker between Alice and Carol? We'll come back to this later, but for now assume that Alice pays Bob a commission.

This concept of 'extending' trust in this way by using localised guarantees is what we call a trust* relationship. The trust*er (Carol) can then act *as if* they trust the trust*ee (Alice) directly. In order to shift the risk, forfeit payments are used. These will be discussed later, but assume for now that they are micro-payments. All forfeits are paid locally; if Alice defaults then Bob must pay Carol the agreed forfeit whether or not Alice pays Bob the forfeit she owes him (and the two forfeits may be of different amounts). Failure to provide a service - or to pay a forfeit - may result in an update to a *local* trust relationship.

Trust* can be composed to an arbitrary number of hops because all trust is now local and so are the forfeits. It is worth noting that trust isn't the same as trust* even in a one hop scenario. If Bob trust*s Alice to provide a service, it means that Bob trusts Alice to either provide the service or else pay the forfeit[1].

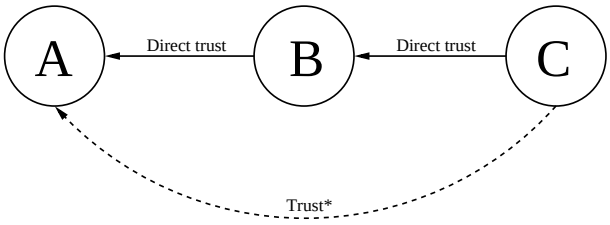


Figure 1: A trust* relationship.

[1]It may be that Bob would rather have the money, and believes that Alice cannot provide the service, but will always pay the forfeit.

2 Applications

There are several promising application areas to which trust* might be beneficially applied.

Spam-proof Email Trust* could be used to implement an email system where messages can be forwarded with an accompanying guarantee claiming that the email is not 'spam'[2]. Spammers rely on sending millions of emails a day to make any respectable profit so it would be unviable for them if even low-value guarantees were required. I'll happily read any email for 10p cash up-front. Now the spammers need to find a cheaper route based on this. This requires email users to filter out emails without guarantees, but existing spam filter applications can be used for this.

Grid Computing How trust can be built in computational grids (which are likely to span organisational and domain boundaries) is a well researched problem [4, 9, 11]. Trust* could easily be applied as a solution and as most grids are used to share resource's across organisations, these resources could be used as the currency for forfeit and commission payments. Resources might include CPU cycles, storage or bandwidth. These typically vary in perceived value between the provider and receiver, so resources could also be brokered in this way, converting one resource into another.

Peer-to-peer Computing When sharing files, most users feel more comfortable knowing that what they might download is licensed, or at least untampered with. Research into building trust in P2p environments has suggested ways of providing this comfort [10, 14]. For example, the Turtle [12] P2p client allows a user to share data with 'friends' or those you already trust directly. In their paper, they suggest that Turtle can be enhanced with an economic model to encourage cooperation and sharing. Applying trust* would not only provide a mechanism to enable this but also allow new principals to join the sharing of files under guaranteed conditions. This application is similar to grid computing except the content itself is now the resource.

[2]The idea was inspired by a 1930's door bell system that was designed to stop unsolicited callers disturbing a household [13]. The door bell is activated by inserting a low value coin which upon answering is refunded if the caller is welcome, otherwise it is kept. This analogy has various flaws but the idea might be better suited to deterring spammers in the cyberworld. Although the coin value is low, to call at hundreds/thousands of houses would soon add up.

Volunteer Computing Many volunteer computing projects require spare CPU cycles to be donated (during screensavers etc) by millions of users worldwide in order to solve a computationally difficult problem. Examples include SETI@Home, a SHA-1 collision search and many others. Multiple projects can be registered and administered using a client called BOINC [1]. There are many security issues [2] related to volunteer computing which could benefit from applying trust*. Volunteer computing differs from grid computing in that anyone can volunteer, whereas grids usually cross organisations which already have a reputation.

Second Life Trust* could be extended to real world transactions such as e-commerce, but it's easier to keep a transaction purely electronic and use trust* in a virtual world. In Second Life, trust* could be used to facilitate the buying and selling of virtual objects. Second Life has its own currency (Linden Dollars) which could be used for making the required commission or forfeit payments. Also, Linden Labs have recently revamped their scripting language and cryptographic libraries within the virtual world which could make guarantee creation and verification possible.

Music Downloads Many people now buy music online rather than buying a physical copy. Services such as iTunes offer single tracks for less than a pound. However, it is unknown to the downloader how much of this money is actually going to the artist or group who produced the music.

Trust* could be used to ensure that a music vendor (iTunes for example) will actually pass on the 30 pence (or whatever was agreed) to the artist. If they don't prove that they did, then the guarantor will pay the artist, prove that they did, and claim it back from the vendor later. This way the artist will always receive their royalties. A possible privacy issue is that proving the money was paid for a specific individual's purchase might divulge their identity to the recording company or artist. Various payment protocols address this, for example, anonymous payments which include a client challenge.

Charity Donations Similarly, a website might include a sponsored link with a promise that 1p will go to charity for every click made. The individual clicking the link might want assurance that the intended charity will actually receive this donation. Here the forfeit would be to produce a receipt showing that the donation has been made, possibly by the guarantor.

3 Discussion

3.1 Networking Analogies

By now you will have noticed that many of the problems with deploying trust* are analogous to well known networking problems. Fortunately, the corresponding network protocol solutions also have trust* analogues. For example, finding the best route between two nodes on a network is analogous to finding an optimal route between two principals who wish to form a trust* relationship with one another. The six degrees of separation argument implies a trust* route can always be found but the best route could be the cheapest (according to commission or computational expense) or the most trusted[3]. It is assumed that any established network routing protocol will suffice for finding optimal chains of guarantors, although the choice of algorithm will have subtle consequences.

Another example is network back pressure. Analogously, if trust* is repeatedly broken between two principals, the guarantor is likely to either break the local trust completely (never provide guarantees again) with the principal being guaranteed (which corresponds to a link outage) or dramatically increase their commission or forfeit rates (which corresponds to a price increase, or a delay). If a particular link drops between two nodes, a route which previously utilised this link might become more expensive for surrounding nodes. This is likely to cause a bottleneck for other nodes following alternative routes and further increasing their cost. These issues can be addressed using network congestion control techniques, and so on.

One difference with conventional networking is that all our links are one way, because trust isn't generally symmetric, whereas most service contracts are bi-directional. This isn't a problem, because two trust* paths can be found in opposite directions via a different route of guarantors[4].

3.2 Commission and Forfeits

The most obvious use of a forfeit is either to deter a principal from defaulting on what they have guaranteed or to provide a way of compensating the other party if they do. The commission payment was introduced in order to provide an incentive for a principal to act as a guarantor and can be seen as a spot price for a guarantee. A principal needing to be trust*ed could pay this commission to a guarantor who trusts them directly.

[3]Different levels of trust, forfeit and commission etc correspond to different network Quality of Services.

[4]The analogy is thus with a network of links which are uni-directional for data flow, although bi-directional for control flow.

Forfeit and commission payments serve different purposes and don't need to be of the same type (or paid by the same means). Also, these payments and the actual service being provided need not be like-for-like.

The price of a guarantee or the forfeit that should be paid if it is broken are variable and could be set by a guarantor to reflect their perception of the risk involved in providing a guarantee. For example; as a risky guarantee is more likely to be broken, a higher forfeit might be required by the guarantor. A low risk guarantee is unlikely to be broken so the guarantor will get his incentive through the commission as a forfeit payment is less likely to happen. Another incentive to provide a guarantee is to make a profit from a forfeit. Assume that Alice is trust*ed by Carol with Bob providing the guarantee to Carol. If Alice defaults, the forfeit from Alice to Bob might be more than Bob has to pay Carol[5].

These considerations lead to some interesting effects regarding the commission and forfeit rates along a chain of guarantees. In this scenario, if Alice was to default the guarantee, only Alice will be out of pocket as the forfeit rate is higher at her end of the chain (and decreases towards the trust*ing end). Every guarantor will make a profit in this case but if we consider a longer chain where risk perceptions fluctuate, guarantors might lose out. For this reason, it is likely that guarantors will only provide guarantees where they believe the rates involved will make them better off in most cases. This flexibility of perception is vital in ensuring that guarantors get their incentive and principals who might default are sufficiently deterred.

3.3 Heterogeneity

In order to implement the trust* relationship mechanism, whether to initiate, provide, or receive a guarantee, a way of making decisions and payments is necessary. In our initial implementation, we used the Keynote trust management system [5] to act as the core decision maker and also to provide the syntax and semantics of the guarantee credentials and policies. To make payments, a micro-payment system [6] (also implemented in Keynote) provided a way for principals to pay commission and forfeits to each other. However, one of the advantages of our approach is that both the trust management and payment systems can be heterogeneous due to the fact that trust (and payments) are confined or localised. If a guarantee has been made from one principal to another, any trust management and micro-payment schemes could be used between them. At the same time, other

[5]Note that this gives Bob an incentive to hope that Alice defaults. Alternatively, Alice may pay Bob a commission instead of a forfeit, in which case Bob hopes that she doesn't default. The second case is like buying insurance. Commission c has the same expectation (but lower variance) for Bob as pqf, where p is Bob's estimate of the chance of Alice defaulting, and q is his assessment of the chance of Alice paying the forfeit f.

pairs of principals might use completely different schemes. As long as an agreement has been made in advance on how the protocol will be followed between a specific truster and trustee, then it doesn't matter what is being used in other parts of the chain.

3.4 Anonymity

Do guarantees ever need to be verified outside of the localised trust relationship? In our protocols, each guarantee is verified by the principal receiving it locally. Once a chain of guarantors has been found (say between Alice and Carol via Bob), how does Alice prove to Carol that she is in fact guaranteed to use Carol's database? Some kind of access control credential could be used to encode the guarantee chain details which can be verified by Carol. However, Carol doesn't need to know who Alice is. All Carol needs to know is that she has received a guarantee from someone whom she trusts (Bob) and from whom she can claim a forfeit if Alice misuses the service provided. Carol doesn't care about any other local agreement in the chain, just the one between Bob and herself. Consequently, the trust* mechanism can be deployed in protocols where anonymity is required[6].

Trust* is intended to be deployed in environments where there is no universally trusted arbiter or referee. If Carol starts claiming that every email she receives is spam, Bob will either stop providing the guarantees, or will charge more for providing them. Alternatively, Alice may form a cycle of trust; Alice might trust Dave (who trusts Carol) to refund her forfeit if it is unfairly claimed.

3.5 Payment by Resource

Micro-payments are generally considered to be small electronic monetary transfers. Due to the heterogeneous nature of the localised trust between individual pairs of principals, the payment could be something of a more immediately valuable commodity to them (in comparison to using purely monetary payments). As mentioned, payment could be by a resource such as CPU time, database access or bandwidth.

If a guarantor is taking payments of one type (from a principal they trust) and making payments of another type (to a principal who trusts them), the guarantor is effectively acting as a resource broker between these principals. Also, trust* could be used alongside an existing trust infrastructure and use payments of an existing

[6]Indeed, the guarantee chain can be used to provide anonymity. Of course, the trust* mechanism could also be extended to situations where a guarantee chain needs to be identified (and verified) during audit. For example, an auditor might want to verify each guarantee which extends trust between Alice and Carol in order to prove a forfeit is payable (analogous to a bail bond agent or bounty hunter etc).

commodity such as reputation ratings or credit (maybe when a forfeit hasn't been paid). Indeed, existing trust or reputation could also be used as a commodity of payment. The point is that this flexibility should make it possible to use trust* to complement existing infrastructures rather than replace them.

4 Conclusion

The whole concept of extending trust to trust* makes use of already existing trust relationships rather than creating new ones. It uses guarantees to bridge the gap between unknown principals with a sequence of localised agreements which remove or reduce the perceived risk of the trust*ing principal and shift it towards the principal being trust*ed.

The next stage of this work will involve applying the idea of trust* to some of the various applications outlined in this paper. The chosen applications will be modelled using a discrete event simulator such as Repast [3] upon which trust* will be applied. This will be a means to defining the boundaries of the existing model. For example, problems might become evident when applying trust* to grid computing that weren't in the spam-proof application.

Trust* is flexible in that it can be used in many different applications, however because it builds upon already existing trust, it won't need to replace any existing trust infrastructures. It will integrate with them and can utilise existing commodities such as reputation.

A The Anti-spam Protocol

This protocol shows how trust* might work in the spam-proof email application. It involves three principals with one path of delegation, as in Fig 1. Alice wants to email Carol; Carol trusts Bob and Bob trusts Alice. Note that the forfeit and commission payments, as well as the email itself, go in the opposite direction to the arrows of trust.

1. $A \longrightarrow B$: Please may Alice have a token for Carol, forfeit=f, commission=c
2. $B \longrightarrow C$: Please may Alice have a token for Carol, forfeit=f', commission=c'
3. $C \longrightarrow B$: Token for email from Alice to Carol, id=x etc
4. $B \longrightarrow A$: Token for email from Alice to Carol, id=x etc
5. $A \Longrightarrow C$: Email (token x in header)
6. $C \longrightarrow B$: Token x is OK/spam
7. $B \longrightarrow C$: Cheers/here is the forfeit
8. $B \longrightarrow A$: Token x is OK/spam
9. $A \longrightarrow B$: Cheers/here is the forfeit

The tokens need to be crypto-protected but Alice, Bob and Carol can be identified by anonymous keys. We assume that the message; $A \longrightarrow C$: Please may Alice have a token for Carol, forfeit=0, commission=.10 will always work.

References

[1] BOINC. `http://boinc.berkeley.edu/`.

[2] BOINC. `http://boinc.berkeley.edu/trac/wiki/SecurityIssues`.

[3] Repast Agent Simulation Toolkit. `http://repast.sourceforge.net/`.

[4] J. Basney, W. Nejdl, D. Olmedilla, V. Welch, and M. Winslett. Negotiating Trust on the Grid. In *In 2nd WWW Workshop on Semantics in P2P and Grid Computing*, 2004.

[5] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos Keromytis. The Keynote Trust-Management System, 1998. `http://www.crypto.com/papers/rfc2704.txt`.

[6] Matt Blaze, John Ioannidis, and Angelos Keromytis. Offline Micropayments without Trusted Hardware. In *Proceedings of the Fifth International Conference on Financial Cryptography*, 2001.

[7] Bruce Christianson and William S. Harbison. Why Isn't Trust Transitive? In *Proceedings of the International Workshop on Security Protocols*, pages 171–176, London, UK, 1997. Springer-Verlag.

[8] Audun Jøsang, Elizabeth Gray, and Michael Kinateder. Analysing Topologies of Transitive Trust. In *Proceedings of the Workshop of Formal Aspects of Security and Trust (FAST)*, pages 9–22, 2003.

[9] Nicola Mezzetti. Towards a Model for Trust Relationships in Virtual Enterprises. *Database and Expert Systems Applications, International Workshop on*, 2003.

[10] Anirban Mondal and Masaru Kitsuregawa. Privacy, Security and Trust in P2P environments: A Perspective. In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 682–686, 2006.

[11] Daniel Olmedilla, Omer F. Rana, Brian Matthews, and Wolfgang Nejdl. Security and Trust Issues in Semantic Grids. In *In Proceedings of the Dagsthul Seminar, Semantic Grid: The Convergence of Technologies*, 2005.

[12] Bogdan C. Popescu, Bruno Crispo, and Andrew S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-up and Beat the System. In *In Proc. of the 12th Cambridge Intl. Workshop on Security Protocols*, 2004.

[13] Popular Science. Dime put in slot rings doorbell, 1933. `http://blog.modernmechanix.com/2007/05/05/dime-put-in-slot-rings-doorbell/`.

[14] Dan S. Wallach. A Survey of Peer-to-Peer Security Issues. In *In International Symposium on Software Security*, pages 42–57, 2002.

Extending Trust in Peer-to-Peer Networks

Stephen Clarke Bruce Christianson Hannan Xiao

Abstract

This paper presents a way of reducing the risk involved with downloading corrupt content from unknown (and hence untrusted) principals in a P2P network. This paper gives a brief overview of the need for trust in P2P networks, introduces a new notion called trust*, and shows how this may be used in place of the conventional notion of trust. Finally, we apply trust* to the Turtle P2P client and show how the integrity of downloaded content can be guaranteed without assuming that trust is transitive.

1 Introduction

Peer-to-peer (P2P) based networks are widely used on the Internet to enable file sharing, streamed media and other services. With a traditional client-server based network, many clients connect to a fixed server. Whereas P2P clients are all considered equal and connect directly to each other. Because of this topology, tasks such as sharing files and other resources can be more efficient as a client can connect to many other clients and download content simultaneously.

Much of the content currently distributed via P2P networks is either illegal or violates copyright laws in some way. However, there are also many legitimate reasons why content might be distributed using P2P, and there is copyright-free content also available such as open source software. P2P protocols such as BitTorrent enable sharing of very large files such as operating systems, and many Linux based distributions are downloadable in this way in order to lower the load on an individual server.

P2P networks have many advantages such as scalability, and due to there being no centralised server, network loads can be easily balanced. However, for the same reasons, a problem with P2P networks is that all peers are regarded as equal and there is no real way to moderate content. Anyone can use a P2P client and share any files they wish. Malicious users can easily insert incorrect files into a network which are searchable by other clients and will therefore propagate further. Even non-malicious users might be unaware that they are serving incorrect files from

their computer. To counter this, hosts might publish an MD5 check-sum on their website. However, this is unlikely and it is the user's decision whether and how they actually verify this, and getting hold of the correct checksum leads us back to the initial problem. Also, this approach assumes that the trustee is the original source and not just a middleman provider.

This paper describes how a new concept called trust* [3] can be applied to P2P networks to guarantee the integrity of files being shared. This paper uses the Turtle P2P client [11] as a basis on which to discuss the concept, although trust* can be applied to various other P2P clients. Turtle enables files to be shared among friends (people whom you know in the real world) in the hope to improve safety and overall integrity of the shared content. However, friendship isn't transitive. Trust* aims to reduce the perceived risk involved when sharing files over multiple hops with unknown principals. Trust* achieves this by providing incentives to act correctly and deterrents for acting maliciously or incompetently.

2 Extending Trust

This section briefly describes the concept of trust* [3]. The main purpose of trust* is to allow unknown principals to interact whilst at the same time lowering the perceived risk incurred by transitively trusting or relying on reputation (particularly when the intention is to use a client *once* and never again).

In the real world, this is often achieved by using an intermediary as a guarantor. An example of this is letting houses to students, where landlords require a guarantee against a particular tenant. The guarantor trusts the tenant and the landlord trusts the guarantor so the landlord has shifted the risk of not receiving the rent to the guarantor. The landlord believes that he will always get his rent whether it be from the tenant or the guarantor.

Trust* is based on the electronic equivalent of the real world guarantee solution. Say that Alice needs to trust Carol about something and doesn't personally know or trust Carol. However, Alice trusts Bob who in turn trusts Carol to do whatever it is Alice needs her to do. In order to change Alice's perception of the risks involved, Bob could guarantee to Alice that Carol will act as intended and offer Alice compensation if Carol doesn't. The concept of "extending" trust in this way by using localised guarantees is what we call a trust* relationship.

The trust*er (Alice) can then act as if they trust the trust*ee (Carol) directly. In order to shift the risk, forfeit payments are used. All forfeits are paid locally; if Carol defaults then Bob must pay Alice the agreed forfeit whether or not Carol pays Bob the forfeit she owes him (and the two forfeits may be of different amounts). Failure to provide a service – or to pay a forfeit – may result in an update to a *local* trust relationship; for example, between Bob and Alice, or

between Carol and Bob. Figure 1 illustrates a typical trust* relationship.

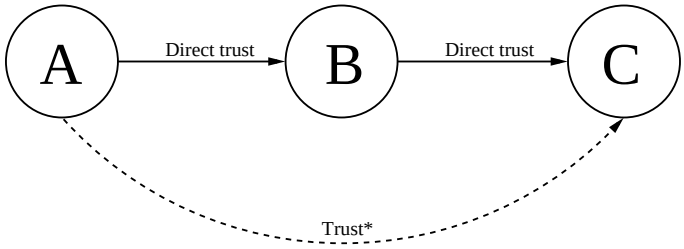


Figure 1: A trust* relationship.

Trust* can be composed to an arbitrary number of hops because all trust is now local and so are the forfeits. It is worth noting that trust isn't the same as trust* even in a one hop scenario; in this case, if Bob trust*s Carol to provide a service, it means that Bob trusts Carol to either provide the service or else pay the forfeit[1].

3 Trust in P2P Networks

Due to the nature of P2P networks and the likelihood that interactions are between completely unknown and untrusted principals, peers in a network need a way to mitigate the risks they would incur if they temporarily trust others. The risks involved are likely to vary depending on what is actually being shared. For example, software should be the correct version and should not be corrupted in any way, documents should be authentic and music should be licensed.

There are many security and trust issues related to P2P networks [1, 5, 9, 13] and the trustworthiness of others is normally gauged using some kind of reputation system [8, 12]. However, reputation systems have a vital flaw; they imply that trust is always transitive [6] which can be a bad assumption [2]. Assume a user wants to determine the risk involved if they were to trust another (eg. to provide a described service) by looking at their reputation rating. This might contain comments and ratings left from previous transactions. It is unlikely that the user looking knows (or trusts) the other users who have left the comments. Also, reputation systems are prone to threats such as Sybil attacks [4] where the same user can operate under many pseudonyms. But even if a user does know and trust the people who left the comments, they will still be transitively trusting the service provider in question.

According to Jøsang *et al* [7], transitivity is possible with the correct combination of the referral and functional variants of trust. However, trust* allows the risk

[1]Bob may believe that Carol cannot provide the service, but will always pay the forfeit.

involved to be underwritten, even when these delicate conditions for transitivity are not satisfied. With trust*, Bob is not only making a recommendation to Alice, but also offering compensation if something goes wrong. The trust scope is decided locally between Alice and Bob when the guarantee is created. It is assumed that the final guarantor in a trust* chain will have functional trust in the end-point (or trust*ee).

Most services provided over a distributed system or network have (as in the real world) an underlying contract or agreement. In most cases, this could simply be that service X will be provided for a fee P and that the service will conform to the terms and conditions of X. In P2P networks, such guidelines do not at present generally exist and clients connect to other clients to become an equal part of the network. Peers are usually free to download anything they wish from other peers and vice versa. There may be situations where content could be charged for or where a particular service level agreement is in place, however, it is more likely that peers in a P2P network hold a "download at your own risk" policy regarding the files that they are sharing.

Trust* can be deployed to provide the missing assurance when indirectly trusting others. For example, Carol doesn't care if someone wants to download file X and doesn't care if they are unhappy with it. However, Bob has previously downloaded files from Carol, and hence trusts that her files are of a high standard. Alice trusts Bob so Bob's guarantee reduces the risk for Alice. If Bob was wrong, he will compensate Alice with the agreed forfeit. However, in this example, Carol hasn't necessarily done anything wrong and isn't obliged to reimburse Bob. Bob however is likely to lower his high perception of the quality of Carol's files and perhaps never guarantee her again. Bob's motivation to provide the guarantee is a commission payment from Alice[2]. Bob will set the level of this commission depending on his perception of the probability of Carol defaulting[3].

4 Applying Trust* to Turtle

The Turtle client requires you to list your friends whom you trust to share files with. The Turtle protocol works by only sending queries for files to these friends, who pass on the query to their friends as their own query and so on[4]. Such queries

[2]In a commercial case, where Carol provides a service for payment, Carol may pay Bob a commission for acting as an intermediary.

[3]Provided Bob's estimate of the probability of Carol defaulting is lower than Alice's *a priori* estimate, then both Alice and Bob will be happy with the guarantee.

[4]If you have read the spam-proof application in [3], please note that the direction of trust in that case goes in the opposite direction to that described here for Turtle. Trust* works perfectly in either direction.

and their results are only ever swapped within these local trust relationships. The second stage is for the original requester to choose the file to be downloaded from the list of results. The file is then downloaded locally by each peer in the chain in the same manner as the search query.

This localised trust setting is perfect for also finding routes of trust* guarantees, as the query and result route used could also make up a chain of guarantees. Extending the example to a longer chain, Alice wants to download file X and sends a query to Bob whom she trusts. Bob forwards this query to Carol whom he trusts. Carol continues to forward this to her friends. Dave receives the query, he has file X and sends back a positive response to Carol which is forwarded back to Bob and then Alice. Assuming now Alice chooses Dave's file via Bob from the list of search results and requests that it comes with a guarantee from Bob, a guarantee chain could be negotiated at the same time as retrieving the file. The scope of the trust* guarantee is also negotiated between each pair which states the terms of the guarantee and what constitutes a breach.

Suppose Alice discovers that the file X is corrupt in some way. Alice can claim the forfeit from Bob. Bob may also claim from Carol. Suppose Dave does not care if his files are correct. So rather than Carol claiming from Dave, she is likely to stop trusting him altogether, or not guarantee against him again, or charge a higher commission from Bob in future for providing the guarantee.

Eventually, say that Dave is habitually sharing corrupt content, all principals who once trusted him are likely to never guarantee his files again. In a fair P2P system where credit or reputation is gained depending on the quantity of uploaded content, and is used to download files from others, Dave will also have trouble buying guarantees from others (or they will be very expensive for him). In this example, the commission can be thought of as an insurance payment.

Alternatively, someone might guarantee only certain types of files from another peer. For example, Carol might be happy to guarantee any of Dave's music files but considers the software that he shares as risky so Carol will not guarantee these files. Trust* can enable these fine-grained decisions to be made. Even when Carol trusts Dave directly, she can still be selective over what she'll actually guarantee.

4.1 Simulation of Trust*

In order to analyse the effectiveness of applying trust* to a P2P scenario, the model was simulated with the Repast Simphony modelling toolkit [10]. A scenario where Alice wishes to download a file from Carol was simulated. There are five possible trust* routes (via the guarantors numbered 1 to 5) and each principal

holds many properties including a credit rating[5]. Two global attributes t and m define the probability of Alice being truthful and Carol sharing incorrect files respectively. The trust* protocol is invoked once every "tick" of the simulation and stops when all available routes have been exhausted. The graphs in figures 2 and 3 show the resulting credit ratings for each principal and how long each simulation ran for.

In graphs (a) and (b), where Carol has a high chance of sharing corrupt files, the simulations stop after 42 ticks. By graphs (c) and (d), the probability of files being corrupt decreases, and hence, the simulation runs for longer. By graph (d) where the corruption chance is 0%, only one guarantor is ever used and the simulation would run forever. Many other graphs show fluctuations in forfeit rates and claims etc, however results presented here are limited for space reasons. The results show that long term trust* usage implies good behaviour from all involved principals. The guarantors will only tolerate misbehaviour for so long before refusing to provide further guarantees of the offending principal.

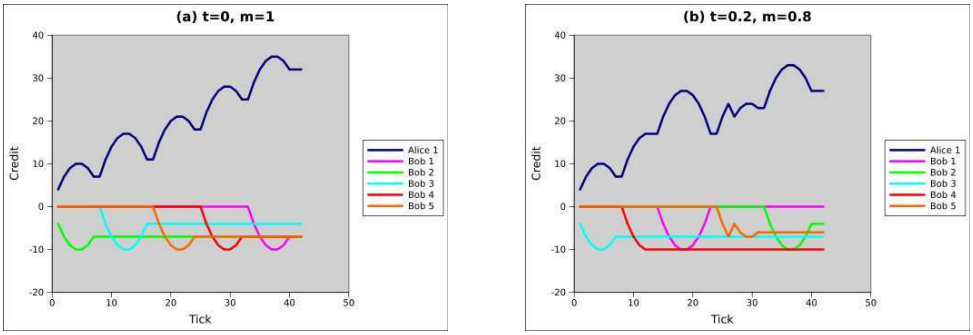


Figure 2: Principals exhibiting bad behaviour.

5 Discussion

5.1 Heterogeneity and Anonymity

In order to implement the trust* relationship mechanism, whether to initiate, provide, or receive a guarantee, a way of making decisions and payments is necessary. This functionality could easily be incorporated within P2P client software. One of the advantages of our approach is that the trust management and payment systems can both be heterogeneous, due to the fact that trust (and payments) are confined or localised. If a guarantee has been made from one principal to another, any trust

[5]This is purely to gauge the total gains and losses of a principal.

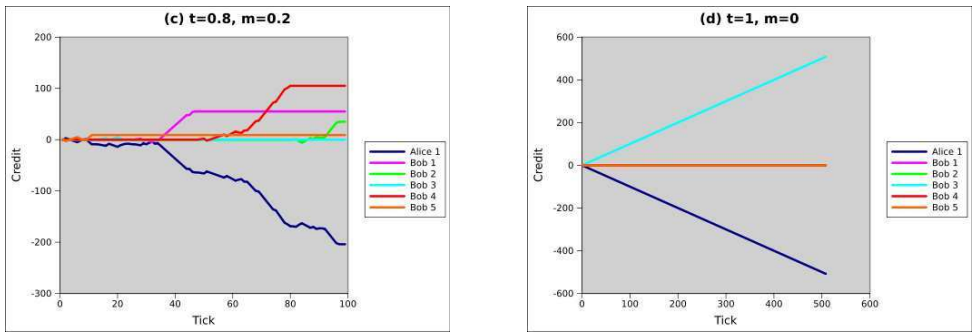


Figure 3: Principals exhibiting good behaviour.

management and payment schemes could be used between them. At the same time, other pairs of principals might use completely different schemes.

Because of this localisation of trust, end-point anonymity can also be maintained as principals only speak to their direct neighbours. No knowledge need be gained about other principals or the schemes they might be using. Also, participants need not know whom they are downloading from.

5.2 Payment by Resource

The most obvious use of a forfeit is either to deter a principal from defaulting on what they have guaranteed or to provide a way of compensating the other party if they do[6]. The commission payment was introduced in order to provide an incentive for a principal to act as a guarantor and can be seen as a spot price for a guarantee. A principal needing to trust* could pay this commission to a guarantor whom they trust directly. Forfeit and commission payments serve different purposes and don't need to be of the same type (or paid by the same means), although in the case of P2P networks, they could easily be.

Due to the heterogeneous nature of the localised trust between individual pairs of principals, the payments could take the form of a more immediately valuable commodity to them than a conventional micro-payment. In P2P file sharing applications, this could be the content itself. For example, credit to download further files or to buy licenses or guarantees.

[6]Note that these are slightly different requirements; a lower forfeit will often suffice for the first.

6 Conclusion

This paper has presented the concept of trust* as a mechanism for guaranteeing the integrity of content or services provided over a P2P network. Trust* builds on the idea of sharing with friends in the Turtle P2P client but also guarantees the integrity of downloaded content from unknown peers derived through transitivity.

Using trust* in this way also reduces the risk involved for the downloader as they will be compensated in the worst case scenario. It therefore lowers the risk of transitively trusting others, and privacy is still maintained. This is because the guarantees and payments are confined within the same localised trust relationships as the ones that are used to communicate the actual search queries and their corresponding results. This approach therefore allows complete localisation of trust management, and the risk of trusting by referral is underwritten by the guarantees. We regard local trust management as a significantly easier problem than global reputation management, particularly in a P2P system where the majority of participants wish to be anonymous (except to their friends). As mentioned earlier, the use of trust* does not constrain the way in which local trust is managed.

Simulation of the trust* protocol shows that misbehaving principals quickly become isolated before major damage can be made. This means that threats such as a Sybil attack can be identified and the perpetrator will eventually be removed from local trust relationships. This will make it harder for them to share files in a P2P community that employs the trust* model as eventually all routes will be removed (or become too expensive).

The Turtle client was developed with an emphasis on privacy and safety of sharing files that might be of a controversial or provocative nature. Due to the localised direct trust in a trust* chain, such privacy can be easily maintained[7]. We have argued that applying trust* to P2P file sharing will also be beneficial in guaranteeing the integrity of free content such as open source software or copyright-free movies.

[7]However, privacy is not so much of an issue when sharing open content, and in other applications where the integrity of the content is more important.

References

[1] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 310–317, 2001.

[2] Bruce Christianson and William S. Harbison. Why Isn't Trust Transitive? In *Proceedings of the International Workshop on Security Protocols*, pages 171–176. Springer-Verlag, 1997.

[3] Stephen Clarke, Bruce Christianson, and Hannan Xiao. Trust*: Using Local Guarantees to Extend the Reach of Trust. In *Proceedings of the Seventeenth International Workshop on Security Protocols*, April 2009. To appear.

[4] John R. Douceur. The Sybil Attack. In *Peer-To-Peer Systems: First International Workshop*, page 251. Springer-Verlag, 2002.

[5] Junjie Jiang, Haihuan Bai, and Weinong Wang. Trust and Cooperation in Peer-to-Peer Systems. In *GCC 2003*, pages 371–378. Springer-Verlag, 2004.

[6] Audun Jøsang, Elizabeth Gray, and Michael Kinateder. Analysing Topologies of Transitive Trust. In *Proceedings of the Workshop of Formal Aspects of Security and Trust*, pages 9–22, 2003.

[7] Audun Jøsang, Ross Hayward, and Simon Simon Pope. Trust Network Analysis with Subjective Logic. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*. Australian Computer Society, Inc., 2006.

[8] Eleni Koutrouli and Aphrodite Tsalgatidou. Reputation-Based Trust Systems for P2P Applications: Design Issues and Comparison Framework. In *Trust and Privacy in Digital Business*. Springer-Verlag, 2006.

[9] Anirban Mondal and Masaru Kitsuregawa. Privacy, Security and Trust in P2P environments: A Perspective. In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 682–686, 2006.

[10] Michael J. North, T. R. Howe, Nick T. Collier, and J. R. Vos. The Repast Simphony Development Environment. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, 2005.

[11] Bogdan C. Popescu, Bruno Crispo, and Andrew S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-up and Beat the System. In *In Proc. of the 12th Cambridge International Workshop on Security Protocols*, 2004.

[12] Ali A Selçuk, Ersin Uzun, and Mark R. Pariente. A Reputation-Based Trust Management System for P2P Networks. In *CCGRID*, pages 251–258. IEEE Computer Society, 2004.

[13] Dan S. Wallach. A Survey of Peer-to-Peer Security Issues. In *In International Symposium on Software Security*, pages 42–57, 2002.