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

Stephen Clarke, Bruce Christianson, and Hannan Xiao

School of Computer Science, University of Hertfordshire, UK
{s.w.1.clarke,b.christianson,h.xiao}@herts.ac.uk

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].

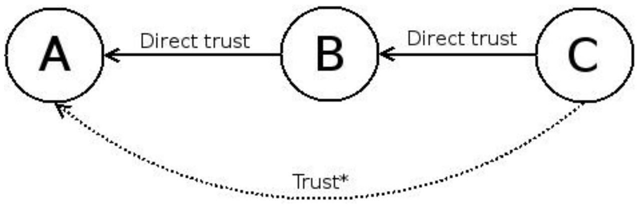

Fig. 1. A Trust* Relationship.

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

[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] 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

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.

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.

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.

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.

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

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.

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.

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

[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.

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 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

[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).

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 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.