Enforcing Collaboration in Peer-to-Peer Routing Services

Tim Moreton and Andrew Twigg

Computer Laboratory, Cambridge University, UK
{Tim.Moreton,Andrew.Twigg}@cl.cam.ac.uk

Abstract. Many peer-to-peer services rely on a cooperative model of interaction among nodes, yet actually provide little incentive for nodes to collaborate. In this paper, we develop a trust and security architecture for a routing and node location service based on Kademlia [1], a distributed hash table. Crucially, rather than 'routing round' defective or malicious nodes, we discourage free-riding by requiring a node to contribute honestly in order to obtain routing service in return. We claim that our trust protocol enforces collaboration and show how our modified version of Kademlia resists a wide variety of attacks.

1 Introduction

Peer-to-peer systems are global-scale distributed applications that consist of nodes or peers typically run by individuals and organisations without an out-of-band trust relationship. Despite this, most existing such systems rely on a cooperative model of node interaction. Participants join the network and use the service: other peers route their queries, serve their requests or store their data. In return, they are expected to contribute their own resources to provide the same functionality to other peers, although doing so yields no direct benefit to them.

Game-theoretic and empirical analysis of such applications [2,3] has shown that users will often defect from providing resources if it is in their personal interest to do so, while still using the service, regardless of the degradation that results. Some adversaries may even actively consume resources in an attempt to deny service to legitimate users.

The decentralized nature of a peer-to-peer system means that it is impossible to account for the validity of peer software implementing the system's protocol: a user may develop or download an alternative to any 'bona fide' implementation. In such an environment, we must assume that each peer acts as a rational, self-interested agent.

Routing substrates such as Pastry [4], Tapestry [5] and Kademlia [1] are an important class of middleware for peer-to-peer applications, and are the focus of much recent research. Being services in the same way as higher-level peer-to-peer applications, each node contributes by maintaining routing table information and carrying out either message-passing or responding to routing table requests.

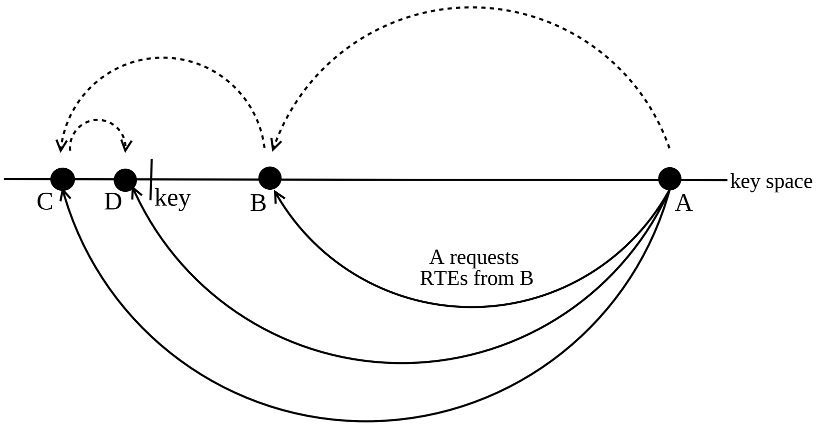

Fig. 1. Locating a node in Kademlia is reminiscent of a source-routing protocol. Node A wishes to route to the closest node to a key (which happens to be D), by learning of and querying successively closer nodes, represented by the solid lines. The dashed lines represent the conceptual routing that takes place

There is no reason to believe that peers will behave any differently in a routing service; as yet though there are no real-world deployments from which to gather evidence.

This work focuses on developing a new model for participation in routing services that are based on distributed hash tables (DHTs), and in particular Kademlia. We describe a trust protocol which forms the basis for enforcing collaboration by providing incentives for nodes to participate both in the routing service and in the trust framework.

The remainder of this paper is organised as follows. Section 2 outlines Kademlia, the routing service to which we apply our model. Section 3 describes the trust model, how to extend Kademlia to support it, and discusses attacks. Finally, we compare our approach with related work in Section 4.

2 Background

2.1 Kademlia

Kademlia [1] is a distributed hash table in which peers are assigned a unique identifier that determines the position they take in a global key space. Each node maintains a set of *routing table entries* (RTEs) organised by the distance between itself and each remote node. Kademlia offers the property common to DHTs that a node's knowledge of the key space is greater for values closer to their own identifier.

Observations made of node uptime data in traces of Gnutella networks [3] show that nodes are more likely to stay connected, the longer that they have remained connected already. Kademlia applies a least-recently-seen eviction policy to nodes in its tables, and never removes entries for nodes that respond to PING messages. This means that long-term pairwise relationships build up between peers close to each other in the key space. In our framework these relationships allow nodes to determine accurate trust information about likely first and second hops.

Unlike other distributed hash tables, Kademlia does not perform message-passing in a hop-by-hop fashion, where each node on the route forwards the request towards its destination key.

Instead, at each step, the source node picks a node B and requests B's k closest nodes' RTEs[1] . Each successive step uses routing information obtained in the previous step, until the source node can determine the identity of the destination peer; at this point they may communicate directly. Figure 1 demonstrates this process.

2.2 Service and Participation in Routing Substrates

In considering Kademlia's routing service, we distinguish three categories of participation: honest participation, and two categories of active, dishonest behaviour: free-riding and deliberate subversion:

– *Free-riders* wish to use the global service, but aim to minimise their participation costs by not contributing some or all of the resources expected of them. By threatening to exclude them from the service, we make it in free-riders' interests to participate honestly.
– *Deliberately subversive* nodes, on the other hand, actively choose to expend their own resources to deny service to other users. In this case, external motivations determine a node's self-interest. Peers acting in this way have no desire to utilise the global service except to gain a position whereby they might deny service.

Our trust model allows nodes to 'route around' both categories of dishonest peers. However, using trust to avoid poorly-contributing nodes is not sufficient. In a routing context, free-riding nodes are less likely to return responses, so fewer nodes will pass requests for routing information to them; the amount of resources they need to consume to contribute is reduced. It is then in the interest of all nodes to defect and free-ride; a global equilibrium may emerge in which no node replies to queries and the service collapses. To avoid this, we need additionally to align the incentives of free-riding nodes with participation in the routing service.

So, we aim to enforce two properties in a collaborative service:

1. Dishonest nodes do not *provide* the routing service for other valid participants (the *avoidance* property).
2. Dishonest nodes may not *use* the routing service (the *exclusion* property).

[1] In practice, at each step the node makes requests for RTEs from α different nodes, to mitigate the effect on the latency of the lookup from nodes that do not respond.

3 Enforcing Collaboration

3.1 Securing Messages and Routing Tables

As it stands, the Kademlia protocol is susceptible to nodes returning false information in replying to requests. Here we present techniques that restrict the class of attacks that a trust model must consider.

Pairwise authentication. Each node has an associated asymmetric keypair, which can be used to sign and verify requests and replies between nodes. As nodes in each other's routing tables tend to interact over an extended period, we are investigating a lower-overhead means of authenticating messages by hashing chains of codewords.

Secure RTEs and identifier assignment. Recall that node identifiers are assigned pseudo-randomly. It is not sufficient to allow each node to determine its own identifier, since this may allow an adversary to install a higher fraction of subversive nodes in one region of the keyspace where it wants to censor a node or a data item.

Rather, we calculate it as the hash of the concatenation of the node's public key and IP address, included to prevent identifiers being swapped between real nodes. A trusted third party Certification Agency (CA) or a similar distributed scheme signs these two unhashed values, so that it is difficult for adversaries to obtain many virtual identities. The operation of the CA and the requirements it makes on nodes (financial, proof of identity, etc.) are beyond the scope of this paper.

Testing for malicious tampering of replies. The above techniques ensure that a set of k RTEs can contain no non-existent nodes. A malicious adversary may, however, return nodes which are not among the k closest to the lookup key, by excluding legitimate entries. A *density test* proposed by Castro et al. [6] may be used to compare the spacing of identifiers in the local routing tables with the spacing of the returned entries. Given the distance between the remote node and the destination key, we can estimate the average expected spacing and range of nodes in the appropriate level of the remote routing table, and compare it to observations.

3.2 The Structure of Trust Values

Before developing the trust model, we present a few brief definitions. *Trust values* are elements of a complete lattice $(T, \leq)$, P is the set of *principals* and the *trust space* t is a partial function $t : P \rightharpoonup (P \rightharpoonup T)$. Initially, let P be the set of nodes in the network, hence we write t^A_B to denote A's trust in node B.

We separate the notions of trust into two categories: trust as a participator in the service and trust as a recommender of other principals. This avoids

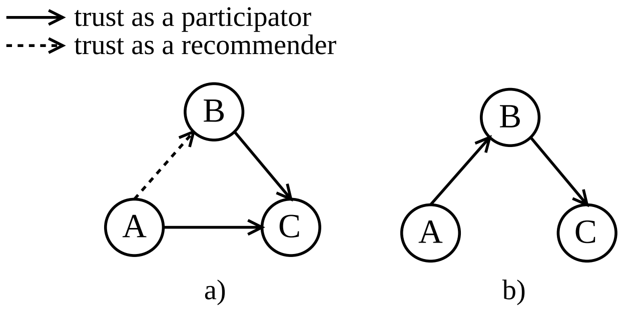

Fig. 2. How separate trust spaces permit selective transitivity in trust relationships. Principal A only trusts C as a participant if A trusts B as a recommender and B trusts C as a participant. Interestingly, A need not trust B as a participant for it to have the transitive relationship

the inherent difficulty associated with having to make an assumption about a principal's ability to *recommend*, based on their ability to *participate*. Without this independence, s service is open to the *colluding nodes* attack. Essentially a node builds up the trust of another node A (by participating), then makes false recommendations to 'transfer' A's trust into a set of malicious nodes, which A now trusts transitively via B. By separating trust in participation and recommendation, we avoid this attack, as described in Section 3.7. We now define more precisely these two notions of trust.

Let $t^A_{P,B}$ be principal A's trust in principal B as a participant of the service, *i.e.* returning valid RTEs. This is computed based on principals' (including A's) interactions with B. Let $t^A_{R,B}$ be principal A's trust in principal B as a recommender. More accurately, it is the trust A has in the proposition 'B returns accurate recommendations about other principals'.

Transitivity. Trust is not, in general, transitive [7] yet it should be transitive for small groups of principals who trust each other *in a certain way.* By separating trust in participation and recommendation, we show how trust can be transferred *selectively transitively.*

Consider the arrangement of nodes in Figure 2. If B trusts C to participate ($t^B_{P,C}$ is high) then should A trust C to participate, i.e. should the trust be transitively transferred? The answer depends on A's trust in B as a recommender (*i.e.* $t^A_{R,B}$), though *not* in A's trust in B as a participator. If $t^A_{R,B} = \top \in T$ then the trust is completely transitive, and if it equals $\bot \in T$ then A completely discounts B's recommendation and no trust in C as a participator is inferred. A value in-between describes a partial transitivity and furthermore, each trust relationship has its own degree of transitivity based on these two trust relationships.

3.3 Making Recommendations

Recommendations are made by a principal B by augmenting the RTEs returned with B's trust in those principals. A typical set of routing table entries returned from B would be as in Figure 3.

$$\left\{ \left\langle \mathcal{C}, t^B_{P,C} \right\rangle , \left\langle \mathcal{D}, t^B_{P,D} \right\rangle , \left\langle \mathcal{E}, t^B_{P,E} \right\rangle \right\}$$

Fig. 3. A typical set of RTEs from a principal B. $\mathcal{C}$ represents all the lower-level data about node C, returned by B as per the standard Kademlia protocol

3.4 Observations and Interactions

Let $\varphi^A_B = (x, y)$ represent the direct observations principal A has made on principal B where x, y are the numbers of successful and unsuccessful interactions, respectively. 'Success' is defined to exclude as many attacks as possible on the routing service. We consider a *successful* interaction to be one where B, when passed a valid request for a given key, returns k *valid* RTEs, where a valid RTE is one which:

- **refers to a node that exists:** the identifier associated with the node is valid; it may be signed by an out-of-band Certification Authority [1];
- **is plausibly in the set of the k-closest nodes to the key that B knows about:** to prevent B inserting valid but colluding nodes into the k entries that it returns; i.e. that it passes the *density test* [6].

No response after a certain timeout, whether because B dropped the request, or because it was never delivered, is an unsuccessful interaction. Note that our definition of a routing service specifies only an ability to find the node whose identifier is closest to a given key, not an ability to interact with it.

The exact nature of φ depends on the trust space used – it could be the proportion of successful interactions (and hence in $[0, 1]$) as in [8,9,10] or taken to be *opinions* in subjective logic [11], as in [12]. The advantage of a subjective logic is that it permits a notion of uncertainty in probabilities, allowing nodes to reason subjectively about the trust in the network and the decisions they make. The trust model we develop in this paper makes no assumptions about the trust values used, only that they form a complete lattice $(T, \leq)$ and that a number of operators which obey certain properties, are well-defined (and closed) over T. These are:

- *discounting* $\otimes$: This reduces the contribution of a node's opinion B by our trust in B as a recommender, and is written $t^{A:B} = t^A \otimes t^B$. We require that $\otimes$ be associative but not necessarily commutative. An example $\otimes$ over opinions in subjective logic is given in [11].

- *consensus* $\oplus$: This takes two trust values and combines them, as if two agents held opinions on different events, and is written $t^{A,B} = t^A \oplus t^B$. We require that $\oplus$ be associative and commutative. As an example, if trust values are pairs of (successful, unsuccessful) interactions then $(a, b) \oplus (c, d) = (a + c, b + d)$.
- *agreement* $\ominus$: This returns a measure of the agreement in opinions expressed in two trust values, and is written $t^{A-B} = t^A \ominus t^B$. We require that $\ominus$ both associates and commutes. Trust values that are very similar will have high agreement, and those which differ wildly will have low agreement.

3.5 Trust Protocol

How nodes interact. When A attempts to interact with B (i.e. by requesting RTEs), the protocol is that B satisfies the request (i.e. responds) in proportion to some combination of its trust in A as a participant *and* a recommender. Considering both trust values satisfies the *exclusion* property of the service. An initial thought is to consider just A's trust as a participant (i.e. to reciprocate A's behaviour in replying), but this fails to uphold the exclusion property since lazy nodes (see the *laziness attack*) are not penalised. Hence our protocol is for B to reply to A with probability proportional to $t^B_{P,A} \sqcap t^B_{R,A}$, i.e. the greatest lower bound of the two values.

The bandwidth cost associated with a poorly-trusted node retrying failed requests may not be sufficient on its own to cause the node to improve its participation. As such, we may also employ a *proof-of-work* scheme where a token that requires a certain amount of computation to generate, but is trivial to verify, is associated with each request; this will serve to limit the rate at which A can make requests, and so increase the relative cost of each failed one.

How nodes route. In Kademlia, A picks α nodes at random from its k closest nodes to perform the next 'hop' on the route. Our trust protocol extends this by choosing α nodes which A trusts to return accurate and valid entries. The expected proportion of accurate and valid RTEs returned from B in response to a request from A is given by:

$$t^A_{P,B} \otimes t^A_{R,B} \tag{1}$$

since $t^A_{P,B}$ is A's trust in B to reply with *valid* entries and $t^A_{R,B}$ is A's trust that the valid entries B returns are accurate.

Hence A chooses α nodes from the k about which it knows that are closest to the desired key, with probabilities proportional to their expected values as in Equation (1). This provides a form of load balancing, and offers nodes with low trust values an opportunity to increase them through successful interactions while adhering overall to the desired avoidance property.

3.6 Trust Computation

In this section, we describe how a node's trust values are computed for both participants and recommenders, and discuss their derivations and solutions.

Computing the trust in participators. We now look at how A computes its trust in B as a participant. Without the use of recommendations, the trust computation for $t^A_{P,B}$ (A's trust in B as a participant) is given by:

$$t^A_{P,B} = t^A_{R,A} \otimes \varphi^A_B \tag{2}$$

where $t^A_{R,A}$ is A's trust in itself as a recommender (which may be $\top$). Including this may seem surprising, but it is not uncommon to not have full trust in one's observations in reality.

Now we consider how to incorporate recommendations from other principals. Let $C : B$ represent a recommendation from principal C about principal B, obtained from an RTE returned by C. Using the *discounting* operator $\otimes$, the recommendation $C : B$ with trust value $t^C_{P,B}$ (C's trust in B as a participant) is discounted (its contribution reduced) by A's trust in C as a *recommender* ($t^A_{R,C}$). This is done for each recommendation *about* B, and the discounted values are combined using the *consensus* operator $\otimes$.

This leads to an iterative fixpoint computation whose solution is the fixpoint assignment of all participating trust values. The final step is to introduce the direct observations. For $t^A_{P,B}$ all the recommendations are combined with A's direct observations on B, φ^A_B. The final trust computation for participation is given below:

$$t^A_{P,B} = \bigoplus_{\forall C \text{ s.t. } C:B} \left\{ t^A_{R,C} \otimes t^C_{P,B} \right\} \oplus \left(t^A_{R,A} \otimes \varphi^A_B \right) \tag{3}$$

Computing the trust in recommenders. Now we consider how A computes its trust in B as a recommender. Essentially what we want to do is *compare* the recommendations it makes with our view of the same, using the *agreement operator* $\ominus$ which represents the agreement between two trust values. Hence $(t_C \ominus t_B)$ is a high trust value iff t_C and t_B concur.

We can combine the results of all these 'comparisons' by using the consensus operator $\oplus$, over all the recommendations made *by* (as opposed to *about* as in Equation (3)) B, about some principal C. The computation is as below:

$$t^A_{R,B} = \bigoplus_{\forall C \text{ s.t. } B:C} \left\{ t^B_{P,C} \ominus t^A_{P,C} \right\} \tag{4}$$

It may be that $\oplus$ is not the best way to combine the comparisons in all applications. Essentially, $\oplus$ performs an 'averaging' over all the comparisons, which allows incorrect recommendations to become 'diluted' or lost if a principal makes enough good ones. The computation could be made 'more strict' by taking the

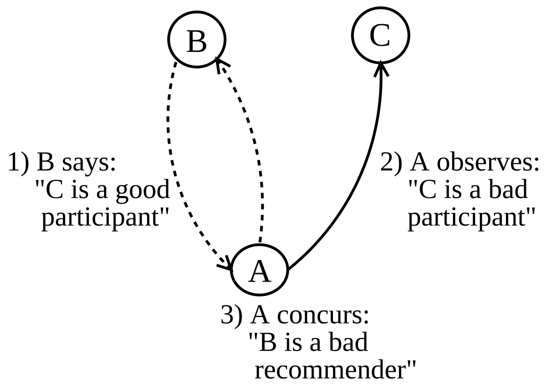
B
C
A
1) B says: "C is a good participant"
2) A observes: "C is a bad participant"
3) A concurs: "B is a bad recommender"

Fig. 4. Updating A's opinion about B as a recommender. In addition to A's direct observations on C, it can use other recommendations from principals it already trusts as recommenders (of course, they can be wrong but the process converges to the correct situation rapidly)

worst comparison, i.e. replacing $\oplus$ with the greatest lower bound of the observations, $\sqcap$ (which exists since our trust values form a complete lattice). It should now be clear that $\oplus$ and $\sqcap$ are just two operators in a partial order which describes how 'strict' A is about computing its trust in B as a recommender. The order includes other operators and a subset of it is given below:

$$\perp \sqsubseteq \sqcap \sqsubseteq \oplus \sqsubseteq \sqcup \sqsubseteq \top \tag{5}$$

where $\perp$ means 'ignore B's recommendations and assign it lowest trust anyway' (and the opposite for $\top$), and $\sqcup$ takes the least upper bound of the comparisons.

This is interesting since it allows each principal control over how it rates other principals, as well as how it computes the trust values. As a node, we do not care about *how* A computes its trust values, only how they compare to our and others' findings. Thus, it is in a principal's interest to accurately compute the values (to avoid being marked as lazy) whilst doing the minimum amount of computation it can 'get away with'.

Meta-recommendations. Consider the situation where A has requested RTEs from B and chooses C as the next step on the route. C returns k valid RTEs as per the protocol (and hence *participates*), but since A has no evidence of C as a recommender, it cannot discount C's recommendations. This situation will not be uncommon in large principal spaces where nodes will have only interacted and received recommendations for, a very small proportion of the principals in the space.

In the same way that principals can recommend principals' abilities to participate, we can increase the propagation of trust by considering the ability to recommend other principals' abilities to recommend. Recall that $t^A_{R,B}$ is A's

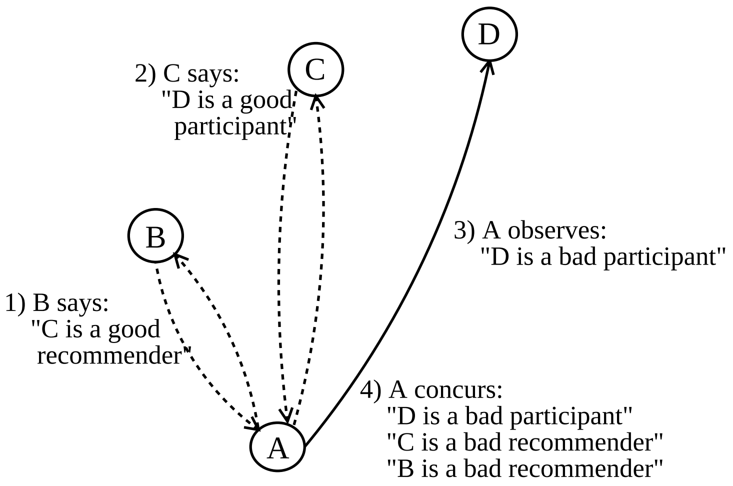
2) C says: "D is a good participant"
1) B says: "C is a good recommender"
3) A observes: "D is a bad participant"
4) A concurs: "D is a bad participant" "C is a bad recommender" "B is a bad recommender"
A
B
C
D

Fig. 5. How meta-recommendations are updated and propagated. A's trust in B's ability as a recommender decreases due to one of two things: C turns out to be a bad recommender, or (not shown) C could be a bad participant. Hence meta-recommendations consider the ability to *recommend in general*

recommendation about B's ability to return correct observations about other principals. A *meta-recommendation* is a recommendation about a principal's ability to recommend principals who make accurate recommendations about other principals. We make the assumption that a principal makes accurate meta-recommendations (i.e. about principals' abilities to recommend) if they make accurate recommendations, since it is the ability to *recommend* in general, not their ability to recommend about participation, that we are interested in modelling.

To handle meta-recommendations, a node returns in its RTEs its trust values of other nodes as recommenders, in addition to their trust values as participants, as shown in Figure 6.

$$\left\{\left\langle \mathcal{C}, t^B_{P,C}, t^B_{R,C}\right\rangle, \left\langle \mathcal{D}, t^B_{P,D}, t^B_{R,D}\right\rangle, \left\langle \mathcal{E}, t^B_{P,E}, t^B_{R,E}\right\rangle\right\}$$

Fig. 6. A typical set of RTEs from a principal B with meta-recommendations. $\mathcal{C}$ represents all the lower-level data about node C, returned by B as per the standard Kademlia protocol

Fixed-point solutions to trust equations. Equations (3) and (4) are mutually-recursive - A's trust in B as a participant affects A's (and others') trust in B as a recommender, which in turn affects the weightings of B's rec-

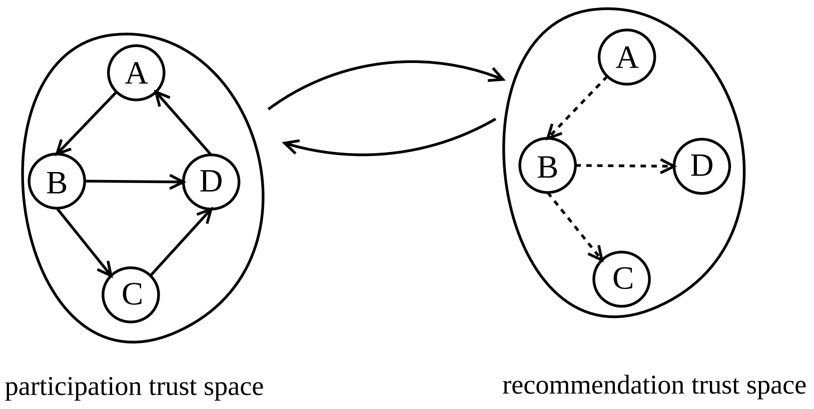

Fig. 7. How the trust spaces affect each other by means of the trust Equations (3) and (4). The fixed-point solution to both sets of trust values represents the 'equilibrium position' between the spaces, i.e. when the values in one do not change the values in the other

ommendations, and so on. The solution is analogous to eigen problems in sparse graphs (such as the WWW), so techniques similar to PageRank [13] can be employed to solve it. A similar computation and its approximate solution is described in [12,9].

Conceptually, the two trust spaces can be thought of as products in a two-way reaction, as in Figure 7. Essentially, the fixed-point solution to the trust equations is the equilibrium position between these spaces.

3.7 Attacks

This subsection outlines a series of possible attacks on the service, and informally describes how they are resisted.

Recall that we so far only model the aggregate property of B as a recommender of principals' participation in the service. One may consider stronger adversarial network models where nodes make false recommendations depending on the subject of the recommendation, and this behaviour can be modelled by transforming the principal space as in [12]. This type of adversarial model is particularly appropriate when one considers the colluding nodes attack.

Attack 1 (Colluding Nodes) *A node B participates in the service and makes true recommendations, except for other nodes in its* collusion set*, about which it falsely reports excellent observations, as shown in Figure 8.*

By treating principals as pairs of nodes, trust values resemble $t^A_{R,B:C}$ meaning A's trust in B as a recommender of C's participation. Then, even if B participates and *makes good recommendations on other nodes outside its set, B cannot 'transfer' A's trust in it into the set via C, since B's correctness in*

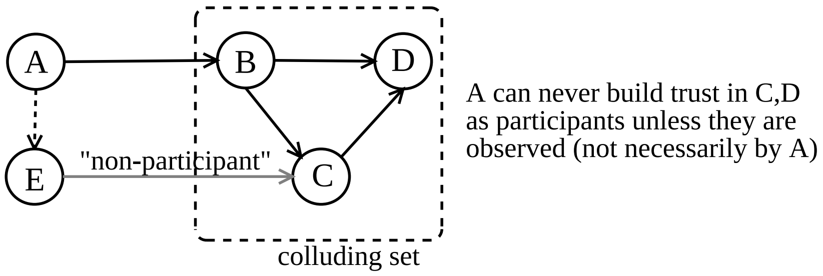
A
B
D
E
C
"non-participant"
A can never build trust in C,D as participants unless they are observed (not necessarily by A)
colluding set

Fig. 8. The colluding nodes attack. Nodes B, C, D form a set where B participates and aims to use this to transfer A's trust in it into the set, where C, D are free-riding. A has high trust in E as a recommender, who has observed C to be a free-rider. Separating trust in participation and recommendation prevents C, D from free-riding at the expense of A

recommending other nodes has no influence on A's trust in it recommending C. Only by observing (or gaining recommendations from other nodes whom A trusts to recommend C) C's participation (and therefore validating or refuting B's recommendation) can A transfer trust into the set. But if C is malicious and does not participate, B's claim is impossible to validate.

Resisting the next attack essentially relies on the system to uphold both the avoidance and exclusion properties of the service, to both avoid and exclude 'free-riders'.

Attack 2 (Free-riding) *A free-riding node B avoids participation in the service by not returning RTEs when requested, or returning invalid RTEs.*

Nodes that request RTEs from B will rapidly concur that $t_{P,B}$ is low, both by their direct observations and from recommendations of principals they trust. The avoidance *property of the service is not compromised since nodes will route around B by avoiding requesting RTEs from it, and the* exclusion *property is upheld since nodes will rapidly deny B their RTEs, ending its reign as a free-rider.*

A variation on free-riding is laziness. Although this may not be an attack in the malicious sense, we still consider it a threat to the service.

Attack 3 (Laziness) *A lazy node B participates in the service by returning* valid *RTEs but with random trust values since it does not wish to (or cannot) expend the resources involved in computing trust values. This is distinct from bad-mouthing, in that expected correlation between B's trust values and the actual values is zero, yet the values are all strongly-negative under bad-mouthing.*

On average the t_P^B values in RTEs returned by B are significantly different from those computed by non-lazy nodes (not just other honest nodes). By Equation (3), the trust values $t_{R,B}$ will be low. Hence B can be identified as a lazy node quite easily and hence avoided. The exclusion property of our service is also upheld, since other nodes will not return RTEs for B very often.

However, we certainly consider the next attack to be of malicious intent since it involves nodes spreading malicious recommendations. Unfortunately, it may be quite difficult to distinguish between bad-mouthing and lazy nodes in reality unless the attack is sufficiently long-lived.

Attack 4 (Bad-mouthing) *A node B attempts to* bad-mouth *other nodes by returning* valid *RTEs but with malicious trust values, often $\perp$.*

This attack is similar to the laziness attack except that B may do one of two things:

1. *Return malicious (i.e. low) trust values for all nodes;*
2. *Return lazy (i.e. random) trust values for nodes it does not bad-mouth;*
3. *Return correct trust values for nodes it does not bad-mouth.*

The first mode of attack can be detected (and treated) in the same way as for laziness. The final two modes require a transformation of the principal space so that principals represent pairs of nodes, as per Attack 1.

A possible motivation for this attack is an attempt to justify later free-riding, by claiming that nodes B bad-mouthed were poor participators, and hence B is justified in reciprocating their behaviour towards them, as if it were following the protocol honestly.

The phrase 'screw-each-server-once' has been used as an attack against systems which do not consider recommendations (where the trust is built 'locally') and therefore is fairly weak in the current context.

Attack 5 (Screw-each-server-once) *A malicious node B which does not participate nor make accurate recommendations (due to laziness or bad-mouthing) attempts to gain maximum use of the service by interacting with as many principals as possible in the hope that it stays 'ahead of its reputation'.*

The use of almost any recommendation and reputation system will counteract this attack eventually. However, the attacker may use knowledge about how the recommendations are distributed in order to maximize its benefit from the attack. Since nodes in Kademlia are distributed randomly over the key space, the graph which describes how recommendations are distributed (by other nodes performing routing) is essentially a random graph. Assuming routing requests are randomly distributed and each route discovers reputations about $O(k \log n)$ other nodes, we conjecture that a node can expect to 'screw' $O((1/k) \log n)$ servers before its reputation catches up with it.

The final attack we consider relies on subverting a particular property of Kademlia's routing protocol. That is, a node returns k valid RTEs but they do not represent the k closest nodes in the key space.

Attack 6 (Not returning the actual k closest nodes) *This can be detected using the* density test *described in Section 3.1, and the node's trust value as a participant will be reduced, excluding it from the service.*

4 Related Work

4.1 Economic and Game-Theoretic Approaches

Work on using economic models to realign nodes' incentives to participate have presented schemes that assume variable demand for services. Geels and Kubiatowicz argue in [14] that replica management in global-scale storage systems should be conducted as an economy. Nodes trade off the cost of storing data blocks with the value they are able to charge for access to them – in this context, a variable demand for blocks is essential.

However, variable demand properties may hold for human-valued commodities, such as the information stored or shared in a peer-to-peer system, but not for routing table entries. Since DHTs typically determine the allocation of items to nodes pseudo-randomly, requests for keys will also be distributed evenly, so no particular value can be conferred on any particular destination.

Currently, the lack of a scalable, low-overhead digital cash system that may be fully decentralized hampers uptake of economic models. Mojo Nation [15], a distributed file storage and trading system, used a currency 'mojo' to discourage free-riding and to obtain load balancing at servers by congestion charging, but relied on a centralized trusted third party to prevent double-spending.

Acqusti et al. [16] develop an incentive model for a distributed application that offers anonymity to its participants. They take a game-theoretic approach to analysing node behaviour and attacks in various system models. Trust is only considered as a means to ameliorate pseudo-spoofing [17] attacks, rather than as a means to provide incentives to peers.

4.2 Trust and Reputation Frameworks

Aberer et al. [10] present a system for 'managing trust' in a peer-to-peer system using a complaint-based metric. However, recommendations from other nodes are not discounted and they present a threshold technique for checking whether a node is 'untrustworthy', based on the difference between its recommendations and the 'average view'. This presents a brittle view of trust which is likely to be difficult to use to effectively reason about decisions on whether to interact.

The NICE system [18] aims to identify rather than enforce the existence of cooperative peers. It claims to "efficiently locate the generous minority [of cooperating users], and form a clique of users all of whom offer local services to the community." We take the view that such systems should work to exclude dishonest users rather than avoid them.

4.3 Levien's Stamp Trading Network

Levien proposes a stamp-trading network [19] applied to Kademlia that offers incentives to enforce end-to-end service, and is underpinned by a trust model based on constrained flow in networks. A node's owner explicitly selects other nodes whom they trust, so a trusted set of live nodes must be obtained or known before each join. *Stamps* are continually created and circulated to these nodes, which may them trade them with other nodes, and which can later be redeemed at the issuing node for service.

Trading is directed by Kademlia's key location service, but the mechanism suffers from practical difficulties. Currently, at each step in the key lookup, a node obtains a stamp to facilitate its next hop. However, stamp *exchange rates*, set to reflect the value of the service offered by nodes, require advertising before an exchange can be made. The interaction is further complicated by the Kademlia protocol usually returning k RTEs at once to reduce latency.

The system obtains feedback when a destination node 'refuses' to redeem a stamp that it issued. An audit trail is maintained in the stamp to detect double-spending; each node trading the stamp appends a signed statement to the trail. Unfortunately, this scheme suffers high overhead given the number of trades required for a stamp, and no scheme is proposed to exclude double-spending nodes.

5 Conclusion

We have presented a trust model which aims to enforce collaboration in a peer-to-peer routing service, based on Kademlia, a distributed hash table with symmetric routing properties. Our work is related to the research goals of the SECURE project [20], which aims to develop a formal foundation for trust and risk in global computing systems. We have presented an *existential* view of trust which separates how trust values are computed from what the computation represents. This methodology allows each node to compute trust approximations of differing quality, yet still be able to exchange recommendations – particularly important in mobile and pervasive computing applications.

Acknowledgements. The authors wish to thank members of the Cambridge Computer Laboratory for reading draft versions of this paper, often at short notice. Thanks also go to the reviewers for helpful comments about the general ideas within the paper. The authors are supported by EPSRC and Andrew Twigg is additionally supported by BTexact technologies.

References

1. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Proceedings of IPTPS02, Cambridge, MA. (2002)
2. Adar, E., Huberman, B.: Free riding on Gnutella. In: Technical report, Xerox PARC, 10 Aug. 2000. (2000)
3. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking 2002 (MMCN '02), San Jose, CA, USA (2002)
4. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (2001) 329–350

5. Zhao, B.Y., Kubiatowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley (2001)
6. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.: Security for structured peer-to-peer overlay networks. In: 5th Symposium on Operating Systems Design and Implementaion (OSDI'02). (2002)
7. Christianson, B., Harbison, W.: Why isn't trust transitive? In: Security Protocols Workshop, 1996, pp. 171–176. (1996)
8. Keane, J.: Trust based dynamic source routing in mobile ad hoc networks. MSc. Thesis, Trinity College Dublin) (2002)
9. Xiong, L., Liu, L.: Building trust in decentralized peer-to-peer electronic communities. In: Fifth International Conference on Electronic Commerce Research (ICECR-5), Canada. (2002)
10. Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: CIKM. (2001) 310–317
11. Jøsang, A.: A logic for uncertain probabilities. Available at citeseer.nj.nec.com/392196.html (2001)
12. Twigg, A.: A subjective approach to routing in P2P and ad hoc networks. In: 1st International Conference on Trust Management, Crete. (2003)
13. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. available as tech. Rep., computer science department, stanford university (1998)
14. Geels, D., Kubiatowicz, J.: Replica management should be a game. In: Proceedings of the SIGOPS European Workshop. (2002)
15. Wilcox-O'Hearn, B.: Experiences Deploying a Large-Scale Emergent Network. In: 1st International Peer To Peer Systems Workshop. (2002)
16. Acquisti, A., Dingledine, R., Syverson, P.: On the economics of anonymity. Available at www.freehaven.net/doc/fc03/econymics.pdf (2003)
17. Douceur, J.: The Sybil Attack. In: 1st International Peer To Peer Systems Workshop. (2002)
18. Lee, S., Sherwood, R., Bhattacharjee, B.: Cooperative Peer Groups in NICE. In: IEEE Infocom. (2003)
19. Levien, R.: Stamp trading networks. Available at www.levien.com/thesis (2001)
20. SECURE: Secure: Secure environments for collaboration among ubiquitous roaming entities. EU IST-2001-32486 (2001)