Sources and Monitors: A Trust Model for Peer-to-Peer Networks

Yan Li and Mohamed G. Gouda
Department of Computer Sciences, The University of Texas at Austin
Email: {yanli, gouda}@cs.utexas.edu

***Abstract*—In this paper, we introduce an objective model of trust in peer-to-peer networks. Based on this model, we develop protocols that can be used by the peers in a peer-to-peer network to compute the trust values of other peers in these networks. According to our model, the trust value of a peer is the probability that this peer sends correct messages to other peers, provided that this probability is at least 0.6. (A peer whose probability of sending correct messages is less than 0.6 is regarded as a bad peer that cannot be trusted by other peers in the network.) Each peer actively monitors several good peers in the network and accurately estimates the trust values of each of them. The peers then exchange messages about the trust values of the good peers that they have monitored, and each of them ends up accurately computing the trust values of many good peers in the network, even though many of the exchanged messages are arbitrarily wrong. Through analysis and simulation, we show that a peer in a network can compute the trust values of about 100 good peers in the network, while keeping the error in computing these trust values below 10^{-4}.**

I. Introduction

In a peer-to-peer network, a peer p[i] may need to obtain a copy of a particular file, but then it may become aware that copies of this file are stored in several other peers in the network. In this case, if peer p[i] can compute a trust value for each of these other peers, then p[i] can identify the peer p[j] that has a copy of the needed file and whose trust value is maximum, and then p[i] can download a copy of the file from p[j] [1]. This simple example demonstrates the importance of enabling the peers in a peer-to-peer network to compute the trust values of other peers in the network.

Towards this goal, we address in this paper the following two questions. First, how to define the trust value of a peer in a peer-to-peer network. Second, what protocols can be used by the peers in a peer-to-peer network so that they can compute the trust values of other peers in the network.

To answer the first question, we postulate that each peer p[i] has a fixed probability *tr* of telling the truth to any other peer in its network. We call *tr* the *trust value* of peer p[i]. If *tr* is at least 0.6, then we call p[i] a good peer. Otherwise, we call p[i] a bad peer. As explained below, no peer should request information or accept any message or a file from a bad peer.

To answer the second question, we postulate that each peer p[i] keeps track of a small number of good peers and accurately estimate the trust value of each of them. Peer p[i] can perform this task by periodically requesting information, that p[i] already has, from each of these peers and checking which of them sends back correct information and which of them sends back wrong information. Over time, p[i] can accurately estimate the trust value of each of these peers. (If at the end p[i] detects that any of these peers is a bad peer, then p[i] drops this peer from the list of good peers that p[i] keeps track of and may add other peers to the list.) Clearly, this is an expensive task to perform, and so each peer cannot afford to perform this task except for a small number of good peers. Now each peer p[i] can send messages, naming the good peers that p[i] has kept track of and announcing the trust value that p[i] has estimated for each of them, to other peers in its network. The net result is that each peer in the network can end up with a large list of good peers in the network and with an accurate estimate of the trust value of each of them.

Several earlier papers have addressed the same two questions [1]–[6], and we give an overview of their contributions in the related work section, Section VI. Here, however, we highlight the main differences between how we chose to answer these two questions and how these earlier papers chose to answer the same questions.

Regarding the first question, most earlier papers postulate that each peer p[i] in a network can assign a primitive trust value, in the period [0, 1], to any other peer p[j] in the network, based on the past experience of p[i] in receiving information from p[j]. These primitive trust values are subjective values that depend only on the peer which assigned them, but otherwise they have no objective significance. For example, a peer p[i] may choose to assign a primitive trust value of 0.9 to another peer p[j] because p[i] has received three correct messages and one wrong message from p[j], whereas under the same situation,a third peer p[k] may choose to assign a primitive trust value of 0.6 to p[j]. By contrast, we adopt an objective measure, namely the probability of sending correct messages, to be the trust value of a peer. One implication of adopting an objective measure for the trust values of peers is that we can now objectively distinguish between good peers (whose trust values by our objective measure is at least 0.6) and bad peers (whose trust values by our objective measure is below 0.6).

Regarding the second question, most earlier papers describe protocols that collect the primitive trust values that have been cast in the peer-to-peer network and use these primitive values to compute the trust value of any peer in the network. The problem is that these protocols assume that the collected primitive trust values are correct, even though they have been sent by mostly imperfect peers, and so using the collected

values directly, without checking whether they are correct or not, to compute the trust values can lead to wrong trust values. For example, assume that the primitive trust value estimated by a peer p[i] for a peer p[j] is 0.6, and the primitive trust value estimated by peer p[j] for a third peer p[k] is 0. Assume also that p[j] reports to p[i] that its estimate of the trust value of p[k] is 0.6 instead of 0. (This is possible since p[j] is not a perfect peer.) Now p[i] can use the two primitive trust values of 0.6 each to compute the trust value of p[k] as 0.6 or 0.36 (depending on whether p[i] computes the trust value of p[k] as the minimum or the product of these two primitive trust values). Had p[i] received the correct primitive trust value from p[j], it would have computed the trust value of p[k] to be 0. By contrast, we develop in this paper protocols that can be used to check whether the received primitive trust value are correct or wrong before they use these values in computing the trust values.

II. A Trust Model

In this section, we present our trust model in peer-to-peer networks. We use this model in later sections to develop several protocols that allow peers to compute the trust values of other peers in their peer-to-peer networks.

Our trust model is based on the following three assumptions.

- ***Trust Values***: Each peer in a peer-to-peer network has a fixed probability of "telling" the truth. We refer to this probability as the *trust value* of the peer. For example, when a peer p[i], whose trust value is *tr*, is about to send a message or file to another peer p[j] in its network, p[i] either sends the correct message or file with probability *tr* to p[j], or sends any wrong message or any wrong file with probability *(1-tr)* to p[j]. A peer whose trust value is at least 0.6 is called a *good peer*; otherwise it is called a *bad peer*.
- ***Sources and Monitors***: A good peer uses a *source discovery protocol* to actively monitor several good peers in its network, and accurately estimate the trust values of each of them. If p[i] monitors a good peer p[j] and accurately estimates its trust value (to ensure that it is indeed a good peer), then p[j] is called a *source* of p[i] and p[i] is called a *monitor* of p[j]. Note that a good peer can have several sources and several monitors, and that its sources and monitors may overlap.
- ***The Good Subnetwork***: A good peer knows all its sources, all its monitors and the trust value of everyone of its sources. A good peer can send messages only to its monitors and can receive messages only from its sources. Therefore, the good peers form a subnetwork, called the *good subnetwork*, within the peer-to-peer network. Over the good subnetwork, the good peers execute a *source propagation protocol* so that each good peer ends up with the identities and the trust values of all other good peers in the (good) subnetwork. Our main focus in this paper is to develop the source propagation protocol over the good subnetwork.

Some explanations of the three assumptions are as follow. First, if the trust value *tr* of a peer p[i] is at most 0.5, then it is impossible for peer p[i] to effectively communicate even one bit of information to another peer p[j] in the same network. For example, assume that *tr* = 0.4 and that p[i] attempts to communicate a bit *b* to another peer p[j] by sending *b* many times to p[j]. In this case, p[j] receives bit b only 40% of the times, and receives arbitrary bits in the remaining 60% of the times. In particular, p[j] can end up receiving bit *0* half the times and receiving bit *1* half the times, and so p[j] can never determine the value of the sent bit b.

On the other hand, if the trust value of a peer p[i] is larger than 0.5 but less than 0.6, then p[i] can communicate information to other peers in the network but it will take p[i] a long time and a very large number of messages to do so. Therefore, peers in a network should avoid receiving information from other peers unless they are certain that the trust values of these other peers are at least 0.6.

Second, each good peer p[i] uses a source discovery protocol to identify and keep track of several good peers and accurately estimate their trust values (to ensure that they are indeed good peers and be able to download files from them). The source discovery protocol that p[i] can employ to accurately estimate the trust values of each of these good peers is to periodically request some files, that p[i] already has, from each of these good peers, then determine whether or not each returned file is correct.

Third, the topology of the good subnetwork can be represented by a directed graph, where each node represents a good peer, and where each directed edge from a node p[i] to a node p[j] indicates that p[i] is a source of p[j] and so p[i] can send messages to p[j]. As mentioned above, we assume that each good peer has already used the source discovery protocol to identify the identities of its sources and accurately estimate the trust values of each of them. Now, the good peers need to use the source propagation protocol (which is presented below) so that each good peer ends up with the identities and trust values of all the good peers in the good subnetwork.

The source propagation protocol is not as straightforward as it may seem at first. This is because each good peer can send wrong messages to its monitor up to 40% of the time. Therefore, whenever a good peer receives a message from a source, it cannot immediately assume that the message is correct. However, if the good peer receives the same message from the same source several times, then it can conclude that the message is correct (with high probability). The communication pattern from a source to a monitor is described in great detail in the next section.

III. The Source-Monitor Network

In this section, we describe the communication pattern from a peer p[0] to a peer p[1], where p[0] is a source of p[1] and p[1] is a monitor of p[0] [7]. This communication pattern from a source to a monitor in this simple network is adopted in our design of the source propagation protocols which we describe in Sections IV and V.

Assume that peer p[0] has an integer constant *vl* that it needs to communicate to peer p[1]. To achieve this goal, p[0] periodically sends an integer value *v* to p[1]. With a probability that equals the trust value of p[0], the sent *v* is the correct value of constant *vl*. And with a probability that equals (1 - the trust value of p[0]), the sent *v* is an arbitrary integer. Peer p[1] receives the sent *v* integers, one by one, and maintains at most one "candidate" for the value of constant *vl*. Eventually, p[1] reaches the conclusion that its maintained candidate for the value of *vl* equals, with high probability, the correct value of constant *vl* in p[0].

The source p[0] in this simple network is specified in Protocol 1. Note that p[0] has two constants *tr* and *vl*. The value of constant *tr*, in the range 60..99, is a measure of the trust value of p[0]. In particular, the trust value of p[0] is *tr/100*. Note also that p[0] has only one action that executes over and over, since the guard of the action is **true**.

Protocol 1 for source peer p[0]

```
const      tr : 60..99     {trust value of p[0]}
           vl : integer
variable   r  : 0..99      {random number}
           v  : integer    {sent value}

begin
    true →
        r := random
        if r ≥ tr then
            v := any
        else
            v := vl
        end if
        send v to p[1]
end
```

The monitor p[1] in this network is specified in Protocol 2. Peer p[1] has one constant, *cmax*, whose value is chosen to be 20, and four variables, *c*, *v*, *cv* and *sv*. Variable *c* is a counter whose value is in the range $0..cmax$. Variable *v* stores the latest received value from p[0]. Variable *cv* stores the latest candidate for the value of *vl* in p[0]. And variable *sv* stores the stable value. The value of counter *c* indicates whether peer p[1] can conclude that the current value of *cv* equals, with high probability, the value of *vl* in p[0]. Peer p[1] reaches this conclusion when, and only when, the value of counter *c* is equal to *cmax*.

Peer p[1] has only one action that is executed each time p[1] receives an integer *v* from p[0]. When an integer *v* is received, peer p[1] checks the value of its counter c. If $c = 0$, then variable *cv* is assigned *v* and counter *c* is assigned 1. If $c > 0$ and *cv* is different from the received *v*, then *c* is decreased by 1. If $c > 0$ and *cv* equals the received *v*, then *c* is increased by 1 (provided that c does not exceed its maximum value *cmax*). Then p[1] compares the values of *c* with *cmax*. If $c = cmax$, then p[1] concludes that the current value of its variable *cv* equals, with high probability, the value of *vl* in p[0], and assigns *cv* to the stable value *sv*.

A *global state* of the source-monitor network is defined by the value of constant *vl* in p[0] and the values of the three variables *c*, *cv*, and *sv* in p[1].

Protocol 2 for monitor peer p[1]

```
const      cmax : integer    {cmax = 20}
variable   c    : 0..cmax    {counter, init. 0}
           v    : integer    {received value}
           cv   : integer    {candidate value}
           sv   : integer    {stable value}

begin
    rcv v from p[0] →
        if c = 0 then
            c := 1
            cv := v
        else if cv ≠ v then
            c := c - 1
        else
            c := min(c+1, cmax)
        end if
        if c = cmax then
            sv := cv
        end if
end
```

An *execution step* of the source-monitor network consists of two parts. First, the source peer p[0] executes its action. Second, the monitor peer p[1] executes its action.

The *probability of error*, denoted PE, of the source-monitor network is defined as the probability that the network reaches a global state where the computed stable value *sv* in the monitor p[1] is different from the value of constant *vl* in the source p[0]. The following theorem, whose proof is in [8], specifies the value of PE.

Theorem 1. *(probability of error)*
The probability of error of the source-monitor network PE is 1.2×10^{-6}. ■

The *convergence span* of the source-monitor network is the average number of steps need to be executed in order to change the global state of the network from one where $c = cmax$ and $sv \neq vl$ to one where $c = cmax$ and $sv = vl$. The following theorem, whose proof is in [8], gives a formula for approximately computing the convergence span of the source-monitor network.

Theorem 2. *(convergence span of the source-monitor network)*

$$convergence\ span \approx \frac{2 \times cmax}{2 \times tr - 1} \quad steps.$$

■

IV. Computing a trust value on a ring

In this section, we describe the source propagation protocol in the special case where the topology of the good subnetwork is a unidirectional ring as shown in Figure 1.

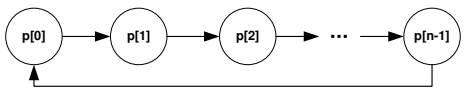

Fig. 1. A ring good subnetwork

This ring subnetwork has n good peers. Each good peer p[i] is a source of peer p[i+1 mod n] and is a monitor of peer p[i-1 mod n]. Thus, each p[i] can send messages to p[i+1 mod n] and can receive messages from p[i-1 mod n].

The peers in this ring exchange messages according to the source propagation protocol, described in this section, so that each peer can compute the trust value of one peer, namely p[0]. Because peer p[1] is a monitor of p[0], p[1] already knows the trust value of p[0] using the source discovery protocol discussed earlier. Thus, p[1] sends messages periodically to p[2] in order to inform p[2] of the trust value of p[0], and p[2] sends messages periodically to p[3] in order to inform p[3] of the trust value of p[0], and so on.

Each peer p[i] has two constants *tr* and *pr* whose values are in the range 60..99. The values of these constants indicate the trust values of peers p[i] and p[i-1 mod n]. In particular the trust value of p[i] itself is *tr/100*, and the trust value of p[i-1 mod n] is *pr/100*. (Recall that p[i] is a monitor of p[i-1 mod n] and so it already knows the trust value of p[i-1 mod n].)

Each peer p[i] stores the latest "stable" estimate of the trust value of p[0] in a variable *sv* whose value is in the range 59..99. The value 59 in this range has a special meaning: when the value of *sv* in p[i] is 59, it indicates that p[i] does not know any estimate of the trust value of p[0]. For example, initially, the value of *sv* in p[1] is *pr*, which is the correct trust value of p[0], and the value of *sv* in each other peer is 59 since none of those peers knows the trust value of p[0] in the beginning.

Peer p[i] in this ring is specified in Protocol 3. Note that this protocol has two actions. In the first action, which is called a source action, peer p[i] acts a source of peer p[i+1 mod n], and so it composes and sends a message to peer p[i+1 mod n]. In the second action, peer p[i] acts as a monitor of peer p[i-1 mod n], and so it receives and processes a message from p[i-1 mod n].

The sent message in the source action of a peer p[i] consists a value *v*, where *v* is the current value of variable *sv* in p[i] with a probability that equals the trust value of p[i], or *v* is any arbitrary value in the range 59..99 with a probability that equals (1 - the trust value of p[i]).

When a peer p[i] receives a message that consists of a value *v*, p[i] recognizes that with a high probability the received *v* is the trust value of p[0]. (But if p[i] is p[1] who already knows the trust value *pr* of p[0], then p[i] uses *pr* instead of *v* in what follows.) Then p[i] checks its counter *c* similarly as that in Protocol 2.

Each execution step of this network consists of two parts. First, every peer in the network executes its source action. Second, every peer in the network executes its monitor action.

Let PR(i) denote the probability that variable *sv* in a peer p[i] in the ring has a value that is different from the correct trust value of peer p[0]. This implies that PR(1) = 0 since the value of variable *sv* in peer p[1] is always the correct trust value of p[0]. Next, we derive a formula to compute PR(i) as a function of PR(i-1). There are two scenarios that can lead p[i] to assign a wrong value to its variable *sv*. In the first scenario, variable *sv* in peer p[i-1 mod n] has a correct value but p[i-1 mod n] sends any wrong value to p[i]. In the second scenario, variable *sv* in peer p[i-1 mod n] has a wrong value and p[i-1 mod n] sends it as is to p[i]. The probability of occurrence of the first scenario is $PE \times (1 - PR(i-1))$, where *PE* is the probability of error of the source-monitor network discussed in Theorem 1. The probability of occurrence of the second scenario is $(1 - PE) \times PR(i-1)$. Therefore,

$$\begin{aligned} PR(i) &= PE \times (1 - PR(i-1)) + (1 - PE) \times PR(i-1) \\ &\approx PE + PR(i-1) \end{aligned}$$

Given that PR(1) = 0 and PE = 1.2 $\times 10^{-6}$, we get

$$\begin{aligned} PR(2) &= PE &&= 1.2 \times 10^{-6} \ , \\ PR(3) &= 2 \times PE &&= 2.4 \times 10^{-6} \ , \quad \ldots \end{aligned}$$

We also simulated a ring that has 100 peers, namely p[0] to p[99], and calculated the probability of error PR(i) for every peer p[i], where i=1, 19, 39, 59, 79, 99. The simulation results, shown in Figure 2, are of the same order of magnitude as those obtained from the above analysis. The simulation results in Figure 2 indicate (as expected) that peer p[99] has the maximum probability of error which is 5.6×10^{-5}.

So far, we discussed the source propagation protocol for computing the trust value of a single peer in a good subnetwork whose topology is a unidirectional ring, where each peer has exactly one source and one monitor. This protocol can be extended so that every peer on the ring computes the trust values of all peers on the ring. Details of this extension are reported in [8]. And through simulation, we showed in [8] that the probability of error in computing the stable trust vector for 100 good peers in the ring is about 10^{-4}.

Protocol 3 for peer p[i] to compute trust of p[0] on ring

```
const     tr   : 60..99     {trust value of p[i]}
          pr   : 60..99     {trust value of p[i-1 mod n]}
          cmax : integer    {cmax = 20}
variable  r    : 0..99      {random number}
          c    : 0..cmax    {counter, init. 0}
          v    : 59..99     {sent or received value}
          cv   : 59..99     {candidate value}
          sv   : 59..99     {stable trust value}
begin
     true →
          r := random
          if r ≥ tr then
             v := any
          else
             v := sv
          end if
          send v to p[i+1 mod n]
||   rcv v from p[i-1 mod n] →
          if i = 1 then
             v := pr
          end if
          if c = 0 then
             c := 1
             cv := v
          else if cv ≠ v then
             c := c - 1
          else
             c := min(c+1, cmax)
          end if
          if c = cmax then
             sv := cv
          end if
end
```

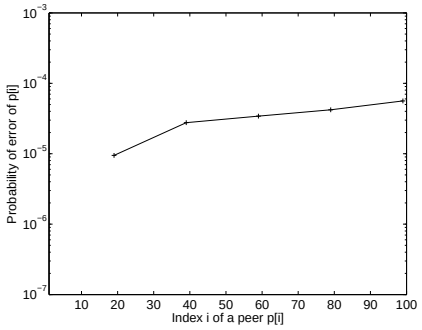

Fig. 2. Probability of error for computing a trust value on a ring subnetwork

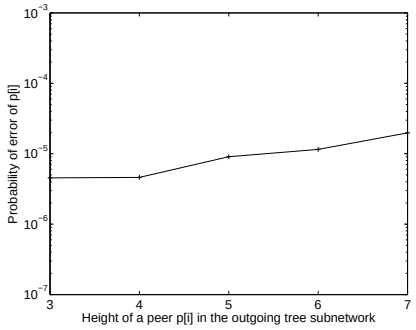

Fig. 4. Probability of error versus height of a peer in an outgoing tree subnetwork

V. Computing the trust vector of a general network

In this section, we briefly describe the results of extending the source propagation protocols for computing the trust vectors in a general good subnetwork, where each peer has any number of sources and any number of monitors. The details of this extension can be found in the technical report [8].

In general, the protocol for a peer p[i] has exactly one source action iff p[i] has one or more monitors, and it has s monitor actions iff p[i] has s sources.

We simulated the source propagation protocol as it executed over different classes of good subnetworks. The objective of this simulation is to measure the probability of error of each good peer executing the protocol and ensure that the measured probability is small enough.

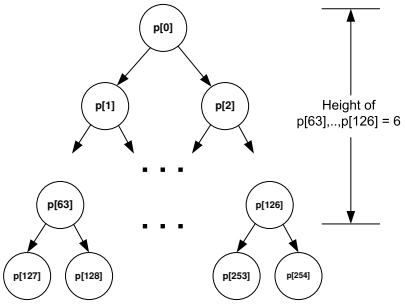

Fig. 3. An outgoing tree good subnetwork

First, we simulated the source propagation protocol as it executed over a good subnetwork, of 255 good peers, whose topology is an outgoing binary tree as shown in Figure 3. In this subnetwork, the *height* of each good peer p[i] is measured from the root peer p[0]. Clearly, all good peers that have the same height in the subnetwork have the same probability of error. The results of this simulation is plotted in Figure 4. Note that the good peers with the maximum height have the maximum probability of error which is on the order of 10^{-5}.

Second, we simulated the source propagation protocol as it executed over a good subnetwork, of 255 good peers, whose topology is an incoming binary tree as shown in Figure 5. In this subnetwork, the *height* of each good peer p[i] is measured from one of the leaf peers p[127], p[128], ..., p[254]. The results of this simulation is plotted in Figure 6. Note that the good peer with the maximum height has the maximum probability of error which is on the order of 10^{-4}.

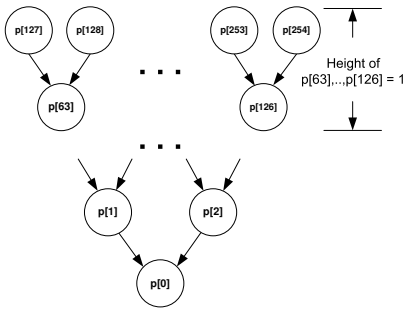

Fig. 5. An incoming tree good subnetwork

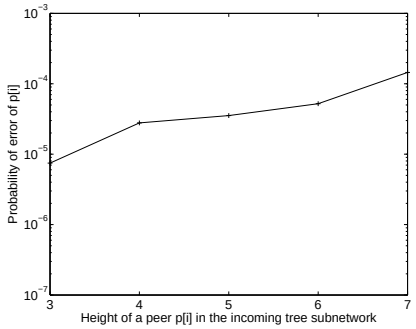

Fig. 6. Probability of error versus height of a peer in an incoming tree subnetwork

VI. Related Work

The distributed nature of peer-to-peer networks makes it a popular architecture for content distribution and file sharing [9], [10]. However, this scalable architecture also brings vulnerabilities to malicious behaviors. As recent studies [11]–[13] show that in peer-to-peer networks, there are quite a lot of inauthentic files injected into the network either deliberately (pollution or poisoning) or unconsciously by non-malicious

users. Another easier way to attack peer-to-peer networks is to add bogus records into the indices to mislead search [14].

EigenTrust [1], based on the transitive trust concept, applies power method to compute a single global trust value for each peer. EigenTrust handles malicious peers lying about their trust values using a Distributed Hash Tables which greatly complicates the system. In our model, we do not assume there is such an underlying structure. Also, the trust value in EigenTrust is relative for each peer. That is, given the computed global trust for a peer, a requesting peer can not tell how much it should trust that peer. In our model, the trust value for a peer does represent how much you can trust that peer.

In NICE [2], trust for each transaction is stored in a signed cookie. Also based on transitive trust concept, NICE searches for a cookie chain to compute the aggregated peer trust. This scheme needs each peer to store tens of both positive and negative cookies and initiate positive cookies and negative cookies search for each request if no cookie is found in its cache.

[3]–[6] introduce personalized credibility of feedbacks when infer trusts. The feedback credibility is computed as the similarity of feedbacks on common interacted peers. However, those common interacted peers sets are hard to find because of the size of the network and the high degree of replication.

XRep [4], X^2Rep [5] and Credence [15] all propose to leverage object trust. However, malicious peers can easily inject inauthentic files or indices into the system as pointed in [11]–[14] which makes the object trust hard to maintain.

In TrustGuard [16], Srivatsa and Liu intend to handle the oscillation behavior of malicious peers using some control theory. But it relies on an existing trust inference algorithm to compute the basic trust value.

VII. Concluding Remarks

In this paper, we proposed an objective measure of trust in peer-to-peer networks. According to our measure, the trust value of a peer is the probability that this peer sends correct messages (or files) to other peers in its network.

Based on this measure, a peer whose trust value is at least 0.6 is considered a good peer and other peers in the network can accept messages from it, or download files from it. On the other hand, a peer whose trust value is less than 0.6 is considered a bad peer that should be avoided by other peers.

Each good peer uses a standard source discovery protocol to actively monitor a small number of good peers in its network and accurately estimate the trust value of each of them. (The source discovery protocol is not the focus of this paper.) Then the good peers use source propagation protocols to exchange messages about the good peers that they have monitored. (These source propagation protocols are the main focus of this paper.) Eventually, each good peer ends up with a large list of good peers and their correct trust values, even though many of the exchanged messages between the good peers are arbitrarily wrong.

Note that the source discovery protocols and the source propagation protocols are always executing. Thus, if an existing good peer leaves the network, then these protocols will eventually drop it from the trust vectors of the other good peers in the network. Also, if a new good peer joins the network, then these protocols will eventually add it to the trust vectors of the other good peers.

Through analysis and simulation, we showed that the probability of error when our source propagation protocols are used in a ring subnetwork of 100 good peers is at most 10^{-4}. This means that the probability, that a computed trust value by a good peer in the ring subnetwork is correct, is .9999.

We also showed through simulation that the probability of error when our protocols are used in an outgoing tree subnetwork of height 7 is at most 2×10^{-5}, and that the probability of error when our protocols are used in an incoming tree subnetwork of height 7 is at most 2×10^{-4}. All these results confirm that our source propagation protocols are reasonably accurate.

References

[1] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *WWW'03*, May 2003.

[2] S. Lee, R. Sherwood, and B. Bhattacharjee, "Cooperative peer groups in nice," in *IEEE INFOCOM*, 2003.

[3] L. Xiong and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, July 2004.

[4] E. Damiani, D. C. di Vimercati, and S. Paraboschi, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *CCS'02*, October 2002.

[5] N. Curtis, R. Safavi-Naini, and W. Susilo, "X^2rep: Enhanced trust semantics for the xrep protocol," in *Applied Cryptography and Network Security*, June 2004.

[6] F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "Choosing reputable servent in a p2p network," in *WWW'02*, 2002.

[7] M. G. Gouda and Y. Li, "The truth system: Can a system of lying processes stabilize?" 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems(SSS'07), November 2007.

[8] Y. Li and M. G. Gouda, "Sources and monitors: A trust model for peer-to-peer networks," The University of Texas at Austin, UTCS Technical Report TR-07-60, 2007.

[9] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM'03*, August 2003.

[10] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling and analysis of a peer-to-peer file-sharing workload," in *ACM SOSP*, October 2003.

[11] J. Liang, R. Kumar, Y. Xi, and K. W. Ross, "Pollution in p2p file sharing systems," in *IEEE INFOCOM*, 2005.

[12] N. Christin, A. S. Weigend, and J. Chuang, "Content availability, pollution and poisoning in file sharing peer-to-peer networks," in *Proceedings of ACM Electronic Commerce*, June 2005.

[13] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel, "Denial-of-service resilience in peer-to-peer file sharing systems," in *SIGMETRICS'05*, June 2005.

[14] J. Liang, N. Naoumov, and K. W. Ross, "The index poisoning attack in p2p file sharing systems," in *IEEE INFOCOM*, 2006.

[15] K. Walsh and E. G. Sirer, "Experience with an object reputation system for peer-to-peer filesharing," in *NSDI'06*, May 2006.

[16] M. Srivatsa, L. Xiong, and L. Liu, "Trustguard: Countering vulnerabilities in reputation management for decentralized overlay networks," in *WWW2005*, May 2005.