Trust Management in Distributed Systems

Huaizhi Li and Mukesh Singhal
University of Kentucky

Distributed systems such as the Internet, peer-to-peer networks, and mobile ad hoc networks involve numerous entities, many of which haven't previously interacted. Trust management can help minimize risk and ensure the network activity of benign entities in distributed systems.

A distributed system is a decentralized network consisting of a collection of autonomous computers that communicate with each other by exchanging messages. These systems are scalable and fault tolerant, and they allow easy resource sharing, concurrent processing, and transparent operation.

As the Internet's popularity grows, distributed applications such as e-commerce are becoming important. In addition, with the rapid development of network and communication technologies, new forms of distributed systems—such as peer-to-peer (P2P) networks and mobile ad hoc networks—are quickly emerging.

Trust is an important issue in distributed systems. Transactions in distributed systems can cross domains and organizations, and not all domains can be trusted to the same level. Even within the same domain, users' trustworthiness can differ. A flexible and general-purpose trust management system can maintain current and consistent trustworthiness information for the different entities in a distributed system.

In e-commerce, for example, a trust-management system lets a buyer and seller become acquainted with each other and estimate the risk of participating in a transaction, thus minimizing the loss. In P2P systems, where each entity acts as both client and server and is expected to contribute to the system, trust management can help reduce free riding, which can seriously degrade P2P system performance.

Finally, in mobile ad hoc networks—a type of distributed system that has no infrastructure and lets nodes move freely—trust management can mitigate nodes' selfish misbehavior, such as dropping or refusing to forward packets for other nodes to save its battery power while still requiring other nodes' services.

Much research exists on trust management[1-3] and reputation management.[4,5] We don't distinguish trust management from reputation management because both can be generalized as dynamic rating systems. Here, we survey the current research on trust management in distributed systems and explore some open research areas.

TRUST MODELS

Trust is a complex subject, and no unanimous definition of trust exists. The *Merriam-Webster's Dictionary* defines trust as "assured reliance on the character, ability, strength, or truth of someone or something." Dictionary.com describes trust as the "firm reliance on the integrity, ability, or character of a person or thing." We define trust as the belief that an entity is capable of acting reliably, dependably, and securely in a particular case.

Trust management entails collecting the information necessary to establish a trust relationship and dynamically monitoring and adjusting the existing trust relationship.

The various models for describing trust and trust establishment in distributed systems include public-key cryptography, the resurrecting duckling model, and the distributed trust model.

0018-9162/07/$25.00 © 2007 IEEE

Public-key cryptography

Many networked services have security mechanisms based on cryptographic techniques such as the Pretty Good Privacy (PGP)[6] or X.509[7] certificate systems, which implicitly use the trust-management concept.

A public-key certificate is a digital certificate issued by a trusted third party to certify a public key's ownership. A certificate contains an entity's identity, public key, and other information, such as the trusted third party's digital signature. Service users are assumed to know the trusted third party's public key so that they can verify the certificate. The trusted third party only vouches for the association between an identity and a public key. It doesn't guarantee the entity's trustworthiness.

In X.509, the trusted third party is a certificate authority (CA), which is usually a trustworthy entity for issuing certificates (Verisign, for example). Another CA might also certify a particular CA.

When a user generates a public/private key pair, it registers its public key with a CA and has the CA certify it. If the same CA certifies two users and they want to communicate securely, they need only exchange their certificates. If different CAs certify two users, they must resort to higher-level CAs, which certify their CAs until they reach a common CA. So, X.509 uses a hierarchical structure, which constructs a tree of trust.

PGP doesn't use a CA. Instead, every entity certifies the binding of IDs and public keys for other entities. For example, an entity A might think it has good knowledge of an entity B and is willing to sign B's certificate. An entity might assign a degree of trust—unknown, untrusted, marginally trusted, or fully trusted—to its certifiers.

The user chooses how to use the certificate. User C might be confident about A's trustworthiness and accept B's certificate, which A has signed. A pessimistic user might only accept certificates certified by fully trusted entities, whereas an optimistic user might trust marginally trusted signers.

Traditional certificate schemes like X.509 and PGP only bind public keys to identities. Because binding an identity to access rights or authorized actions is outside the certificate framework, a certificate framework only provides partial trust management.

Resurrecting duckling model

Frank Stajano and Ross Anderson's[8] *resurrecting duckling* model also has a hierarchical structure. The entities in a network have a master-slave relationship. The master entity is the mother duck and the slave entity is the duckling.

A slave entity recognizes the first entity that sends it a secret key through an out-of-band secret channel (through physical contact, for example) as its master in a process called *imprinting*. The master passes instructions and access control lists to its slaves, and the slaves always abide by their master. The master, a time-out, or a specific event can break the relationship between a master and a slave. After that, other entities can imprint, or resurrect, the slave. A slave entity can also become a master to other entities through the imprinting process.

Thus, the relationship among nodes is a tree-like trust relationship. An entity controls all the entities in its subtree. Breaking the relation between two entities causes the relationships in the entire subtree to break.

This model is appropriate for devices that can't perform public-key cryptography. However, the model requires an out-of-band secret channel to deliver the secret key, which might not be feasible in some networks, such as ad hoc networks.

The distributed trust model assumes asymmetrical trust.

Distributed trust model

Alfarez Abdul-Rahman and Stephen Hailes developed a distributed recommendation-based trust model.[1] They propose *conditional transitivity of trust*, which hypothesizes that trust is transitive under some conditions. For example, if A trusts B, and B trusts C, we can't simply conclude that A trusts C, because trust generally isn't transitive. Abdul-Rahman and Hailes claim that we can conclude that A trusts C if the following conditions are true:

- B recommends its trust in C to A explicitly;
- A trusts B as a recommender; and
- A can judge B's recommendation and decide how much it will trust C, irrespective of B's trust in C.

The model's motivation comes from human society, where human beings get to know each other via direct interaction and through a grapevine of relationships. The same is true in distributed systems. In a large distributed system, every entity can't obtain first-hand information about all other entities. As an option, entities can rely on second-hand information or recommendations. However, because recommendations have uncertainty or risk, entities need to know how to cope with second-hand information.

The distributed trust model assumes asymmetrical trust. It defines two types of trust relationships: *direct trust* and *recommender trust*. It categorizes a trust relationship between two entities in terms of different interactions. Trust in one category is independent of trust in other categories. This model uses continuous trust values for direct trust and recommender trust, as Tables 1 and 2 define.[1] Other researchers fix trust value within the range (0, 1).[2]

The recommendation protocol is straightforward. For example, entity A needs a service from entity D (say car service). A knows nothing about the quality of D's ser-

vice, so A asks B for a recommendation with respect to the car service category, assuming that A trusts B's recommendation within this category. When B receives this request and finds that it doesn't know D either, B forwards A's request to C, which has D's trustworthiness information within the car service category. C sends a reply to A with D's trust value. The path $A \times B \times C \times D$ is the *recommendation path*.

We use the following formula to calculate the trust value from the returned value[1]: $tv\_T = [rtv(1)/4] \times [rtv(2)/4] \times \ldots \times [rtv(i)/4] \times \ldots \times [rtv(n)/4] \times tv(T)$, where $rtv(i)$ is the trust value of the ith recommender in the recommendation path, $tv(T)$ is the trust value of target T returned by the last recommender, and tv_T is the calculated trust value of target T.

When multiple recommendation paths exist between the requester and the target, the target's eventual trust value is the average of the values calculated from different paths.

This model has some weaknesses:

- It doesn't consider false recommendations and assumes that a recommender with a good recommender trust value always makes reliable recommendations, which might not be true.
- It doesn't provide a mechanism for monitoring and reevaluating trust, which is dynamic.

Trust shouldn't be considered a binary concept (that is, either to trust or not to trust). Abdul-Rahman and Hailes quantified trust as a multiple value concept.[1] Many trust-management systems use the same approach.[2,5,9,10] The key challenge then is how to process the trust values to minimize the influence of false recommendations.

We can classify the trust models into two categories:

- *evidence-based model*, in which entities establish a trust relationship based on some evidence, such as keys;[6-8]
- *recommendation-based model*, in which recommendations from intermediaries set up the trust relationship between two strangers.[1]

We can also place the trust-management systems for distributed systems into these two categories. For example, Laurent Eschenauer, Virgil Gligor, and John Baras[11] used an evidence-based approach, while Li Xiong and Ling Liu[2] and Karl Aberer and Zoran Despotovic[3] used a recommendation-based approach.

TRUST MANAGEMENT IN P2P SYSTEMS

P2P systems are distributed systems without centralized control or organization. The peers interact directly and are both consumers and service providers. P2P systems need trust management to ensure cooperation—for example, to reduce free riding.

Table 1. Direct trust value.

Value	Meaning	Explanation
–1	Distrust	Completely untrustworthy
0	Ignorance	Can't decide
1	Minimal	Lowest trust
2	Average	Mean trustworthiness
3	Good	Trusted by major population
4	Complete	Fully trustworthy

Table 2. Recommender trust value.

Value	Meaning	Explanation
–1	Distrust	Completely untrustworthy
0	Ignorance	Can't decide
1	Minimal	The entity itself judges the reliability of recommender's recommendation.
2	Average	
3	Good	
4	Complete	

In P2P systems, peers often must interact with unknown entities whose trustworthiness is also unknown. Centralized schemes or schemes that rely on global knowledge won't work.

Recommendation-based trust management

Xiong and Liu[2] based their distributed trust-management system on feedback or recommendations that help establish trust relationships between unknown or unfamiliar peers. They define a satisfactory interaction as 1 and a complaint as 0. Their trust metric is

$$T(u,t) = \frac{\sum_{v \in P,\, v \neq u} S(u,v,t) \cdot Cr(v,t)}{\sum_{v \in P,\, v \neq u} I(u,v,t)} \quad (1)$$

where

- P is a set of peers in the P2P system;
- u and v are peers in the system, $u, v \in P$;
- $S(u,v,t)$ is the degree of satisfaction that u has with v until the tth transaction;
- $T(u,t)$ is u's trust value evaluated by other peers until the tth transaction;
- $Cr(v,t)$ is the balance factor for filtering feedback from v; and
- $I(u,v,t)$ is the number of interactions that u has with v up to the tth transaction.

So, $T(u,t)$ is the ratio of the cumulative weighted satisfaction that u receives to the total number of interactions that u has within the P2P system.

Xiong and Liu[2] approximated that $S(u,v,t) \cdot Cr(v,t)$ by $I(u,v,t) - C(u,v,t) \cdot T(v,t)$. $C(u,v,t)$ denotes the degree of complaint that v files against u. $C(u,v,t) \cdot T(v,t)$ indicates the filtered complaint filed by v against u. The definition of $T(u,t)$ becomes

$$T(u,t) = 1 - \frac{\sum_{v \in P,\, v \neq u} C(u,v,t) \bullet T(v,t)}{\sum_{v \in P,\, v \neq u} I(u,v,t)} \quad (2)$$

Therefore, $T(u,t)$ is within the range (0, 1). The higher $T(u,t)$ is, the more trustworthy u is. This approach uses v's trust value $T(v,t)$ as a balance factor, similar to Abdul-Rahman and Hailes's approach, which uses recommendation trust value as a balance factor. The higher $T(v,t)$ is, the more reliable v's complaint is. Thus v's complaint has more impact on u's trust value.

The trustworthiness decision criterion is

If $I(u,t) > C1$ and $T(u,t) > C2$, then u is trustworthy.

C1 and C2 are thresholds, with C1 defining the minimum number of interactions required. Obviously, a certain number of interactions are necessary to improve accuracy.

Because Xiong and Liu's approach considers both positive and negative evaluations and interaction history,[2] it is more likely to produce accurate results. However, their trust-management system has several drawbacks:

- The decision criteria in Equation 2 require a minimum number of interactions, which is a disadvantage for newcomers and reentry nodes, which are common in P2P systems.
- Because the balance factor used in Equation 1 is a peer's trust value, the system assumes that a peer with a higher trust value always gives more reliable feedback than a peer with a lower trust value, which might not be true.
- A peer's behavior changes over time. More recent feedback is closer to a peer's current behavior than older feedback. In this model, all previous feedback has the same weight in evaluating a peer's trust value.

Aberer and Despotovic's[3] trust-management system for P2P networks has some similarities to Xiong and Liu's approach. They used other peers' feedback to evaluate a peer's trust value. However, they only considered complaints about a node, which makes the system too sensitive to misbehavior. They used a probabilistic method to analyze the collected complaints and make decisions. The method also uses a binary trust value—that is, a node is either trustworthy or distrusted—which is too coarse.

Anonymity in P2P trust-management systems

In a P2P trust-management system, a peer queries other peers in the network to get another peer's trust value. A malicious peer could discover peers that report a bad trust value for it and attack those peers in revenge or to prevent them from reporting the bad trust value (for example, by using a denial-of-service attack).

Anonymity is one way to cope with this problem. Aameek Singh and Ling Liu's TrustMe is an anonymous trust management system for a P2P system.[12] It uses public-key cryptography-based mechanisms to realize anonymity. For a particular peer, TrustMe randomly selects a certain number of peers in the network to manage that peer's trust value. These peers are the *trust-holding agent* peers for that peer, which doesn't know the identities of its THA peers.

Although TrustMe protects THA peers' identities, a malicious THA peer can disseminate negative trust information for a peer. Remaining problems include preventing malicious peers from becoming THA peers and preventing peers from reporting a wrongful trust value for another peer.

Initial setup. TrustMe requires that a node i has two pairs of private/public keys, denoted by (P_i, B_i) and (P_i', B_i'), in which P denotes a private key and B denotes a public key. A bootstrap server, which is the entry point for peers in a P2P network, also has a pair of private/public keys, (P_{BS}, B_{BS}). It gives the public key B_{BS} to all the nodes joining the network.

The bootstrap server generates a special private/public-key pair (SP_i, SB_i) for node i. However, node i knows only the public key SB_i. The bootstrap server also generates an identifier BID_i for node i: $(BID_i = P_{BS}(\text{"}ValidNode\text{"}|B_i'))$, where $P_{BS}(M)$ means encrypting a message M with key P_{BS} and "|" means concatenation. A node can decrypt BID_i using B_{BS} and knows that BID_i is a valid node's identifier although it doesn't know node i's real identity. It also knows node i's public key.

Node join. When node i joins the network, it sends its B_i and B_i' to the bootstrap server. After generating (SP_i, SB_i) for node i, the bootstrap server sends SB_i to node i. The bootstrap server chooses a set of available peers in the network to serve as the THA peers of node i. The bootstrap server generates a trust-host message for each of node i's THA peers. For example, for peer x, the trust-host message is $TH_{BS}(i) = BID_x|P_{BS}(BID_x(|B_x'|ID_i|B_i|SP_i|SB_i))$, where ID_i is node i's identity.

The bootstrap server sends the trust-host message to node i, which broadcasts the message in the network. When it receives the message, peer x reads it using B_{BS} and P_x' and puts node i in a local database that stores the IDs and trust values of the peers that it serves as a THA peer. Only peer x can interpret the trust-host message. All THA peers of node i know node i's SP_i. Furthermore, node i only knows the BIDs of its THA peers, not their real identities.

Trust value query. When a peer j needs another peer i's trust value, peer j broadcasts a query message, $Query$: $\{ID_i\}$. When a THA peer of peer i, say peer x, receives the query, it generates a reply containing peer i's trust value and sends it to peer j. The reply is $R(x, i) = ID_i \mid B_i \mid SB_i \mid SP_i(TV \mid TS \mid BID_x \mid P_x'(TS))$, where TV is the trust value of peer I, and TS is the message's time stamp. Peer j uses the key B_i for future communication with peer i. The encryption with SP_i guarantees that a THA peer generates the reply message. Peers use BID_x to identify a THA peer, although BID_x isn't peer x's true identity. Peer i can put BID_x on a blacklist if peer x is a malicious peer. The time stamp TS prevents a malicious node from replaying a reply. Peer i can extract B_x' from BID_x and decrypt $P_x'(TS)$, which can verify BID_x's correctness.

Feedback. When peer j interacts with peer i, they exchange a proof-of-interaction message: peer i receives $P_j(TS \mid B_i \mid ID_i)$ from peer j, and peer j receives $P_i(TS \mid B_j \mid ID_j)$ from peer i. No other peers can generate a fake proof-of-interaction message. After the interaction, peer j can report a new trust value for peer i by broadcasting a *report* message: $Report(j, i) = ID_i \mid SB_i(\text{"report"} \mid TV \mid B_j \mid P_j(P_i(TS \mid B_j \mid ID_j)))$, where TV is a new trust value for peer i.

P2P systems assume that the network layer is reliable and that data delivery can be guaranteed.

Only peer i's THA peers know SP_i, so only these peers can read the report message. The $P_i(TS \mid B_j \mid ID_j)$ part of the report message is the proof-of-interaction message exchanged with peer i, which is used to prevent peers from faking a report. From $P_i(TS \mid B_j \mid ID_j)$, the THA peers also get the identifier of the report message's sender, which is peer j.

Node leave. When a peer leaves a P2P network, it notifies the bootstrap server. The bootstrap server selects another peer to assume the leaving peer's responsibilities. The THA peers maintain a time stamp for the information of each peer in their database. When a peer accesses another peer's information, the server updates the time stamp. If the stamp is older than a certain value, a THA peer will delete the information. Therefore, a leaving peer's information will eventually be deleted.

TRUST MANAGEMENT IN MOBILE AD HOC NETWORKS

P2P systems assume that the network layer is reliable and that data delivery, such as request and response, can be guaranteed. This isn't true for ad hoc networks. Therefore, it isn't directly possible to apply the previous approaches to trust management in ad hoc networks.

An ad hoc network relies on all participants actively contributing to network activities such as routing and packet forwarding. An ad hoc network's special characteristics—such as limited memory, battery power, and bandwidth—can cause nodes to act selfishly (refuse to participate in routing and provide services to other nodes, for example). Trust management can help mitigate this selfishness and ensure the efficient utilization of network resources.

Monitoring-based trust-management systems

In ad hoc networks, a node can only sense the packets transmitted within its transmission range. Sonya Buchegger and Jean-Yves Le Boudec's[5] Confidant (Cooperation of Nodes, Fairness in Dynamic Ad Hoc Networks) protocol promotes cooperation in ad hoc networks by detecting and isolating malicious nodes.

Each node in the network runs the Confidant protocol. Confidant's *monitor* component observes the behavior of neighbor nodes to detect misbehavior, such as packet dropping. This requires nodes to run in promiscuous mode.

When the monitor finds a misbehavior, it notifies the *reputation system*, which manages a table containing nodes and their ratings. The rating is a number within a certain range depending on the implementation. If the number of times a node misbehaves exceeds a threshold, the reputation system updates the node's rating. If a node's rating falls below a threshold, the system considers it a malicious node. The reputation system maintains a blacklist containing the malicious nodes. When forwarding packets, nodes avoid next-hop nodes on the blacklist.

When the reputation system detects a malicious node, it notifies the *trust manager* to broadcast an alarm message in the network. Trust managers also receive alarms from other trust managers. A trust manager only distributes and accepts alarms from senders on its friends list. (Establishing friendship is a research topic. One possible method is the resurrecting duckling model.[8]) Each trust manager maintains a table with the trust levels of received alarms.

The *path manager* ranks the path according to the ratings of the nodes on the path. It deletes all paths containing malicious nodes and drops route requests received from malicious nodes.

Buchegger and Boudec didn't discuss how to compute reputation values.[5] In addition, Confidant can't prevent malicious nodes from disseminating false information about other nodes, and trustworthy nodes can lie.

Sergio Marti and his colleagues proposed two methods to improve an ad hoc network's throughput in the presence of misbehaving nodes: a watchdog method and a path rater method.[9] They assumed that a wireless interface supports the promiscuous mode.

The watchdog is a misbehaving node locator running on every node that maintains a buffer of recently sent packets. After overhearing a packet, the watchdog com-

pares it with the packets in the buffer to see if there's a match. If there is, the packet has been forwarded and the watchdog removes the packet from the buffer. If a packet stays in the buffer for longer than a certain period, the watchdog increases a failure count for the node responsible for forwarding the packet. If the count exceeds a threshold value, the watchdog considers that node as misbehaving.

A path rater at a node maintains a rating for every other node that it knows in the network. To pick a route that is most likely to be reliable, it computes a path metric by averaging the rating of the nodes on the paths and chooses the path with the highest metric. It assigns misbehaving nodes a very low rating, and thus excludes them from routing.

Because of ad hoc networks' characteristics, the proposed approaches can't accurately detect misbehaving nodes in situations such as packet collisions and collusion of malicious nodes.[9]

Evidence-based trust management

Eschenauer and his colleagues present a framework for trust management in ad hoc networks based on evidence distribution.[11] They consider trust as a set of relationships established with the support of evidence. In their framework, evidence can be anything a policy requires to establish a trust relationship, such as public key, address, and identity. Any entity can generate evidence for itself and for other entities. Evidence can be obtained either online or offline, such as through physical contact.

One way to generate evidence is through public-key cryptography. An entity can create a piece of evidence, define its valid time, sign it with the entity's private key, and disseminate it to others. To verify this piece of evidence, other entities will need the originator's public key and certificate. In the Internet, entities can use X.509. However, in an ad hoc network, where there is no CA, PGP might be an option. An entity can invalidate its evidence by generating a revocation certificate at any time.

Eschenauer and colleagues' approach also lets an entity revoke other entities' evidence by generating and disseminating contradictory evidence. However, allowing such actions is open to attack. A malicious entity can distribute bogus evidence to invalidate other nodes' legitimate evidence, which can cause chaos in the network. A malicious entity might generate fake evidence for its own purposes—for example, to impersonate other nodes.

To prevent these attacks, Eschenauer and his colleagues proposed using redundant and independent evidence from various sources. However, they didn't discuss how to evaluate evidence, which is important for trust management. Also, because each node's trustworthiness is not dynamically adjusted, the framework is mainly useful for authentication.

One way to generate evidence is through public-key cryptography.

TRUST MANAGEMENT IN E-COMMERCE

Trust or reputation management is an important issue in e-commerce, where traders might have never met and know nothing about each other's trustworthiness. This lack of information about traders' reputations causes uncertainty and mistrust, which influences the e-market's economic efficiency.

Considerable research has explored trust and reputation management in e-commerce. One possibility is to build a centralized system, like a credit history agency, to manage users' reputations. However, this approach neglects personal preferences and standards.

Online auction and shopping sites, such as eBay and Amazon.com, use reputation management. eBay assigns sellers a rating of 1, 0, or –1 for trustworthiness after one interaction, and computes a seller's reputation as the accumulation of all the ratings received within the past 180 days. New eBay users receive a reputation of 0. Amazon.com rates both sellers and buyers after each interaction. It calculates reputation as the average of all the feedback ratings received during the system's use. A new Amazon.com user has no reputation value.

Users can easily misbehave in e-marketing. After cheating and obtaining a bad reputation, a user can simply discard a current identity, obtain a new one, and reenter the market. This kind of misbehavior causes low economic and system utilization efficiency. To solve this problem, Amazon.com and eBay apply pseudonyms. New users must register with some personal information so the system can trace their real identity. At the same time, pseudonyms provide anonymity.

Reputation management for the electronic community

Giorgos Zacharia[4] proposed Sporas, a reputation mechanism for electronic community. Sporas has the following features:

- Reputation value is within the range of (0, 3000). A new user is assigned 0, the minimum value.
- A current user's reputation is always higher than a new user's.
- Two users can only rate each other once. If two users interact multiple times, Sporas only accepts the latest rating. This helps avoid the problem of two users intentionally increasing their reputation value by frequent interactions.
- It changes the reputation value of users with very high reputation values more slightly.

- In evaluating a user's reputation, Sporas assigns more weight to the most recent ratings because they're closer to the user's current behavior.

Sporas uses the following formulas to update a user's reputation after a transaction:[4]

$$R_i = R_{i-1} + \frac{1}{\theta} \cdot \Phi(R_{i-1}) \cdot R_i^{\text{other}} \cdot (W_i - E_i)$$

$$\Phi(R_{i-1}) = 1 - \frac{1}{\left(1 + e^{-(R_{i-1}-D)/\sigma}\right)}$$

$$E_i = \frac{R_{i-1}}{D} \qquad (3)$$

where

- R_i is the user's reputation after the ith transaction;
- R_{i-1} is the user's reputation after the $(i-1)$th transaction;
- R_i^{other} is the reputation of the user with whom the first user had the ith transaction with;
- W_i is the rating that another user gave to user i, ranging from 0.1 (worst) to 1 (best);
- D = 3,000, the largest reputation value:
- θ is a constant larger than 1 that determines how much an entity's reputation value changes after a transaction;
- F is a damping function, which the system uses to decrease the reputation change of very trustworthy users and also reduces the influence of temporary malicious accusations;
- s is a factor for F; and
- E_i is a user's predicted rating (if the feedback is less than the predicted rating, the user's reputation goes down).

Similar to a credit score evaluation system, entities might have a low credit score at the beginning of their use of the system, but, if they perform well for a period of time, their credit score will increase, and the initial low score won't have much influence.

A distributed trust-management broker framework

Kwei-Jay Lin and his colleagues proposed a distributed trust management broker framework for e-services, such as e-commerce.[10] In their framework, each user (a client or a trader) is associated with a broker, which collects trust ratings of any service providers for its users. Figure 1 shows the framework's architecture.

The trust-management framework consists of users, brokers, and reputation authorities. A broker maintains a database, which collects and stores trust information for the users that it's associated with.

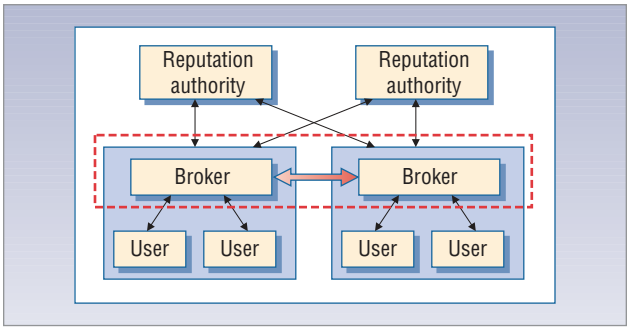

Figure 1. Architecture of a distributed trust-management broker framework. Each user (a client or a trader) is associated with a broker, which collects trust ratings of any service providers for its users. Redrawn with permission from K.-J. Lin.

When a client needs a trader's trust information, it first contacts its broker. The broker's *reputation manager* component processes requests from clients associated with the broker. If it can't handle a request, it passes the request to the broker's *connection manager*. The connection manager sends the request to other brokers and processes their replies. If no broker can provide the required information, it sends the request to the reputation authority.

The reputation authority is designed as a universal database. It collects trust information from the public in a voluntary way and stores trust information for all users. However, the database isn't updated frequently, so the information might be erroneous or obsolete.

After each transaction, a client submits its feedback or rating of the trader to its reputation manager. The reputation manager collects the feedback and computes a trust value for the trader using the following formula:

$$T_{new} = e^{-\beta.\Delta t}\frac{N}{N+1}T_{old} + \left(1 - e^{-\beta.\Delta t}\frac{N}{N+1}\right)r \qquad (4)$$

where

- T_{new} is a trader's new trust value,
- T_{old} is the old trust value,
- N is the current number of transactions,
- r is a trader's feedback,
- Δt is the time difference between feedback and T_{old}, and
- $e^{-\beta.\Delta t}$ is a discount factor of T_{old}.

The formula considers both the feedback and the old trust value. The reputation manager updates the trader's trust value from T_{old} to T_{new} in its local database.

Each broker also has a trust value depending on the accuracy of its recommendations. The connection manager maintains a database that stores other brokers' trust values. After a transaction, if the connection manager asked other brokers for recommendations, the connection manager compares these recommendations with

the transaction's result and updates the brokers' trust values respectively.

When the connection manager receives a trust request from the reputation manager, it forwards the request to the first m brokers whose trust values are larger than a threshold value T. After the connection manager receives recommendation from other brokers, it processes the recommendations using the following formula:

$$R = \sum \frac{X_i \bullet N_i \bullet R_i \bullet F_i}{\sum X_i \bullet N_i \bullet F_i} \quad (5)$$

where X_i is broker i's trust value, N_i is the number of times that the connection manager has asked the broker for recommendations, R_i is broker i's recommendation, and F_i is a time differential factor. If the time difference from the last recommendation is less than a threshold value, F_i is 1; otherwise, F_i is $e^{-\beta.\Delta t}$.

The connection manager sends the calculated recommendation R to the reputation manager, which passes it to the user. The user decides whether or not to proceed with the transaction.

The broker framework[10] is more flexible and scalable than centralized trust management systems, such as Amazon and eBay. Its performance relies on the broker network's trustworthiness and reliability. Recommendations from a broker with low trust value have little value. If some brokers crash, the trust information stored at these brokers isn't available.

Improving the broker framework's robustness is an unsolved problem. Both Sporas and the broker framework use a damping function of the discount factor to reduce the impact of false accusations. Sporas is a distributed system—each entity rates and evaluates other entities. Determining how to effectively search other entities' reputation values is another unanswered problem. In this aspect, the broker framework is a better approach.

Trust management remains an active research area. Many interesting research issues are yet to be fully explored, including trust/reputation value storage. Internet applications, such as e-commerce, can use databases to store users' trust values, updating them regularly to reflect the current trustworthiness of users in the system.

For P2P systems and mobile ad hoc networks, maintaining such data isn't an easy task. Because machines can join or leave a P2P system randomly, a centralized server for storing trust values might not be scalable, and determining where the server should be located would be problematic. For ad hoc networks, a centralized server isn't available. Maintaining data consistency is a challenging issue, because some nodes might crash or leave the network.

Effectively mitigating the influence of false accusations is another open issue. In e-commerce, a user might intentionally give a negative rating to another user. In P2P and mobile ad hoc networks, a node might maliciously assign a low trust value to another node.

Another potential area for research is combining the trust values of different applications. A distributed system, such as an ad hoc network, might involve several different applications, such as packet forwarding, file sharing, and mobile e-commerce. Should we use a different trust value for each application, or use the same trust value for all applications? And, if each application has its own trust value, how can we combine their trust values?

Acknowledgments

We're grateful to the anonymous reviewers whose valuable comments helped us improve this article. This research was partially supported by grant no. T0505060 from the US Treasury Department.

References

1. A. Abdul-Rahman and S. Hailes, "A Distributed Trust Model," *Proc. New Security Paradigms Workshop*, ACM Press, 1997, pp. 48-60.
2. L. Xiong and L. Liu, "Building Trust in Decentralized Peer-to-Peer Electronic Communities," *Proc. 5th Int'l Conf. Electronic Commerce Research* (ICECR-5), 2002; www.mathcs.emory.edu/~lxiong/research/pub/xiong02building.pdf.
3. K. Aberer and Z. Despotovic, "Managing Trust in a Peer-to-Peer Information System," *Proc. 2001 ACM 10th Int'l Conf. Information and Knowledge Management* (CIKM 01), ACM Press, 2001, pp. 310-317.
4. G. Zacharia, "Trust Management through Reputation Mechanisms," *Proc. Workshop in Deception, Fraud, and Trust in Agent Societies, 3rd Int'l Conf. Autonomous Agents* (Agents 99), ACM Press, 1999; www.istc.cnr.it/T3/download/aamas1999/zacharia.pdf.
5. S. Buchegger and J.Y.L. Boudec, "Performance Analysis of the Confidant Protocol: Cooperation of Nodes—Fairness in Dynamic Ad Hoc Networks," *Proc. IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing* (MobiHOC), IEEE Press, 2002, pp. 226-236.
6. M. Elkins et al., *MIME Security with OpenPGP*, IETF RFC 3156, Aug. 2001; www.ietf.org/rfc/rfc3156.txt.
7. R. Housley et al., *Internet X.509 Public Key Infrastructure, Certificate and CRL Profile*, IETF RFC 2459, Jan. 1999; www.ietf.org/rfc/rfc2459.txt.
8. F. Stajano and R.J. Anderson, "The Resurrecting Duckling: Security Issues for Ad Hoc Wireless Networks," *Proc. 7th Security Protocols Workshop*, LNCS 1796, Springer-Verlag, 1999, pp. 172-194.

9. S. Marti et al., "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," *Proc. Int'l Conf. Mobile Computing and Networking* (Mobicom), ACM Press, 2000, pp. 255-265.
10. K.-J. Lin et al., "A Reputation and Trust Management Broker Framework for Web Applications," *Proc. IEEE Int'l Conf. e-Technology, e-Commerce, and e-Services*, IEEE CS Press, 2005, pp. 262-269.
11. L. Eschenauer, V.D. Gligor, and J. Baras, "On Trust Establishment in Mobile Ad-Hoc Networks," *Proc. Security Protocols Workshop*, LNCS 2845, Springer, 2002, pp. 47-66.
12. A. Singh and L. Liu, "TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Networks," *Proc. 3rd IEEE Conf. Peer-to-Peer Computing*, IEEE CS Press, 2003, pp. 142-149.

Huaizhi Li *is a PhD candidate in the Department of Computer Science at the University of Kentucky, Lexington. His research interests include wireless networks, computer networks security, and distributed systems. Li received an MS in computer science from the University of Kentucky. Contact him at hli3@cs.uky.edu.*

Mukesh Singhal *is a professor in the Department of Computer Science at the University of Kentucky, Lexington. His research interests include computer network security, distributed computing systems, wireless networks, and mobile computing. Singhal received a PhD in computer science from the University of Maryland. He is a Fellow of the IEEE. Contact him at singhal@cs.uky.edu.*