Fuzzy Logic Techniques for Reputation Management in Anonymous Peer-to-Peer Systems

Ernesto Damiani[1] **Sabrina De Capitani di Vimercati**[1] **Stefano Paraboschi**[2]
Marco Pesenti **Pierangela Samarati**[1] **Sergio Zara**

(1) Università degli Studi di Milano
Dipartimento di Tecnologie dell'Informazione
Via Bramante 65, Crema CR – Italy
{damiani,decapita,samarati}@dti.unimi.it

(2) Università degli Studi di Bergamo
Ingegneria Gestionale e dell'Informazione
Via Marconi, 5 Dalmine BG – Italy
parabosc@elet.polimi.it

Abstract

Emerging approaches to enforce some form of protection in (pseudo)anonymous peer to peer systems propose the use of reputations as a means to establish the reliability of servents and/or resources. How such reputations can be exchanged, aggregated and used in actually determining the trust one may wish to put in a servent or resource is an issue still to be investigated and for which fuzzy techniques can play an important role. We present an approach for managing and exchanging reputations, based on the use of fuzzy techniques, which we adopted as part of our P2P-XRep system.

Keywords: P2P system, reputation, fuzzy technique.

1 Introduction

Peer-to-Peer (P2P) networks are rapidly achieving an important role in the Internet experience of millions of users [8]. P2P applications allow users to connect directly to other users' machines in order to exchange files and other resources. While a number of different P2P tools have become available in the last couple of years, there are three common characteristics defining a P2P application:

- the ability to discover other peers without the need of a centralized index;
- the ability to query other peers;
- the ability to share content with other peers.

Also, interaction among peers is usually carried out by preserving some degree of anonymity, e.g., by using opaque identifiers rather than names or IP network addresses when communicating with each other. Internet users increasingly find in P2P applications a convenient solution for the anonymous exchange of resources. The success of these applications has produced a significant impact in the research community, which has started to investigate the many open issues arising in this new environment, like performance, usability and robustness. Improvements along these directions can make the P2P networks an interesting alternative to the traditional Web architecture based on the client-server paradigm, replacing it for many situations where users need a simple and efficient solution to share information.

One interesting research field on P2P architectures is the investigation on security mechanisms [1, 2, 6]. In fact, the anonymity of the interaction that characterizes current P2P networks for file sharing, which it is one of the main reasons of their huge success, it also implies that the user has no guarantee on the quality of the resources available on the network. In a traditional Web system (HTTP client/server) the provider of the services vouches for their integrity. In a network with no identities, it becomes relatively easy to spread malicious content, such as viruses, Trojan horses or spam.

Our previous research [3, 4] has proposed a protocol (P2P-XRep) that permits to evaluate the reliability of the shared contents, through a polling on the community of P2P users, who are asked for their opinion on the trustworthiness of a resource obtainable by a specific servent or on the servent itself. The mechanism is based on the exchange of reputations of nodes and resources among the anonymous nodes on the network; the reputation is built on previous users experiences, stored locally on each node in a personal experience-repository. Based upon all the collected opinions, an aggregated reputation value is produced, to synthesize all the received answers to the poll.

1.1 Requirements and basic assumptions

Our reputation system is based on methods that let nodes obtain a reliable reputation by aggregating many votes that taken in isolation may be unreliable. Indeed, the protocol has to pay considerable attention to the fact that, in an anonymous environment with no central control, malicious votes and fake identities can be easily generated.

We consider the different and conflicting requirements that characterize this environment.

1. The system has to maintain user anonymity, as this is a fundamental aspect of current P2P file-sharing networks.
2. Computing reputations must be as transparent as possible to the user and user participation must be minimized.
3. The reputation of a resource/servent should reflect the experience of previous interactions with the network.
4. The reputation must be robust against malicious attacks that might try to attribute a bad reputation to good resources and a good reputation to bad ones.

To satisfy the above requirements, a few assumptions must be made that are essential to the construction of a working system. Our first hypothesis is that the underlying TCP/IP network architecture is trusted and that it is not possible to falsify the IP address of a party involved in a TCP/IP connection. We then assume that malicious users may have considerable resources in terms of computational power and bandwidth, but they can control a limited portion of the global network resources and in particular a small fraction of the network nodes addresses. Moreover, we consider the goal of representing correctly only the reputations of nodes and resources that are known by a significant number of users in the network. The protocol must then be able to correctly reconstruct the reputation in presence of malicious attacks.

1.2 The role of fuzzy techniques

While providing a robust protocol for polling, previous proposals do not fully investigate vote representation and aggregation problem. The protocol is able to return to a node a set of votes that are guaranteed to be a reliable representation of the opinions of the servents community. Reputations are then defined at two levels, both requiring the use of fuzzy techniques: *local* reputations and *network* reputations. Network reputations are built by collecting and combining local ones by way of the P2P-XRep protocol [4]; local reputations in turn derive from a direct experience of a user with the servent or resource inquired about. Determining a network reputation requires to produce an effective synthesis of all the information contained in the votes, because the typical user is not interested in a precise description of all the steps that have been followed by the protocol, but only in a single value that represents the perception of the user community. A way to solve the problem would be the identification of a threshold that would permit to discriminate the outcome of the poll in a good or bad reputation. But, such a crisp value is not adequate and would characterize in the same way a positive reputation produced by the collection of only positive votes by many users and a reputation built with a limited number of heterogeneous votes that produce a value immediately above the threshold; the same reasoning can be applied to negative reputations. An alternative solution to produce an aggregated value reflecting the votes and which looks promising in overcoming the limitations above is represented by the use of fuzzy techniques [7]. We also want to emphasize the role of the synthesis in the construction of the reputation, more than the role of the single vote, which can be considered as unreliable. Fuzzy techniques have the important role to synthesize a possibly huge number of votes into a single value.

Goal of this paper is the investigation of solutions for:

- computing a *local reputation* from direct experience (Section 2);
- expressing votes and controlling votes (Section 3);

- aggregating votes to produce a *network reputation* (Section 4);
- merging local and network reputations (Section 5).

synthesis of all the opinions collected by the poll.

2 Computing a local reputation

When the local reputation derives from a single experience, we may use a crisp boolean value to represent the reputation. When a servent x downloads a resource r from y, x can check that r is what it expects it to be; in this case, the reputation for y and r will be positive (value 1). Conversely, if r deserves a negative vote (because either it contains malicious content or it does not fulfill its description), x will maintain a negative reputation for y and r (value 0). As we said above, different applications may require a finer granularity.

When a servent has had many contrasting experiences with another servent s, a fuzzy value can be used to represent the local reputation on s. Our fuzzy reputation recapitulates the history of past interactions, giving a greater weight to recent experience [9].

Since we desire both to attribute a greater weight to recent experience and to use an efficient evaluation technique, the reputation is computed by a classical exponential average technique. Let $R_l(x)$ be a the value in the interval $[0,1]$ representing the local reputation assigned to x after the $n-1$-th interaction. After the n-th interaction with value v_n (equal to 0 if the interaction is negative, and 1 if it is successful), the new reputation $R_l'(a)$ assigned to servent/resource x will be

$$R_l'(x) = (1-\alpha) \cdot R_l(x) + \alpha \cdot v_n.$$

Coefficient α, a value between 0 and 1, represents the level of responsiveness of the reputation to recent experience (the greater the value of α, the greater the importance of recent values). Simple as it is, this local reputation expresses the user's trust on the node.

3 Voting process

3.1 Expressing votes

If a servent x asks for a poll on a resource r and/or servent y, x will receive a number of votes from other servents that depends on how many interactions node y has had and how many times resource r has been shared. To increase the dynamic nature of reputations, in our approach a servent answering a poll request from another peer does not simply return as a vote its local reputation; instead, it returns the result of its last interaction. The reason for this is twofold: first, the use of a crisp boolean value for the vote (rather that a value within a range) appears adequate. Second, it is important to avoid a situation where a servent acquires a good reputation and then starts to behave maliciously. This may happen either because the servent planned from the start to acquire a good reputation in order to swindle the other peers, or because the servent or its identity has been attacked and is now under the control of a malicious party. If every voter were to return the local reputation, the servent would loose its reputation too slowly. By contrast, when the voters answer with the most recent experience, the network is able to react more quickly to possible changes in behavior.

3.2 Vote verification

Vote verification is a fundamental step of the P2P-XRep protocol, that precedes the computation of the network reputation. Its goal is to protect the protocol against attacks where malicious nodes inject false votes into the network, in order to modify the protocol outcome. Vote verification requires to open a direct connection (on the underlying TCP/IP network) with the voter for a short dialog that permits to verify if the voter really issued the vote that has been received. As vote verification is done by a direct TCP/IP connection, the P2P network is bypassed; an attacker who creates false votes or modifies votes in transit has no way to attack the direct connection.

Since in current P2P networks it would be impractical, for performance reasons, to check all the votes, the P2P-XRep protocol selects randomly a subset of votes to verify. The fraction of verified votes is a critical configuration parameter for the protocol and a trade off has to be considered between bandwidth consumption on one side, that requires to minimize the number of verifications, and protection against attacks on the other, that requires to evaluate a great portion of the votes. Our extension to P2P-XRep assumes that for each received vote, a random value t (uniformly distributed between 0 and 1) is extracted. If t is greater that a given threshold the vote is to be controlled to check its validity, the vote is automatically considered valid otherwise. Of course, it is important for the random function that generates t to be of good quality, otherwise an attack could be mounted that carefully sends false votes in correspondence of positions that the attacker knows are not checked. The threshold becomes the parameter to be configured in order to establish how many verifications will be carried out by the system. The solution that we have implemented uses a dynamic threshold, which follows two principles

1. the larger the number of votes the greater the threshold. Intuitively, this corresponds to requiring a smaller percentage of votes to be controlled when the set of votes is large and aims at avoiding exploding the number of controls to be executed.

2. the threshold should be dynamically changed and adapted to the context: assuming votes are checked in sequence, the threshold can be increased (decreased, respectively) if the checks are successful (not successful, respectively). Intuitively, this corresponds to the fact that unsuccessful checks reflect a critical situation which should be controlled more closely: by lowering the threshold the system therefore increases the probability of the remaining votes to be checked.

A successful and widely deployed system working in a similar way is the mechanism controlling the congestion window in the TCP protocol [5, 10] (the congestion window represents the number of packets of a single connection that can be in the network at the same time). This mechanism shows a great similarity with our problem. In the control of congestion window, the size is dynamically adjusted based on the receipt of notification packets (`ACK`). For each `ACK` received, the congestion window (`CWnd`) is increased with a constant value up to a particular threshold (to avoid a sudden burst of traffic on a network unable to manage it), after which the growth follows a proportional increase. When the timeout for a reception of an `ACK` expires with no received `ACK`, the `CWnd` is reduced. The mechanism can be synthetically represented by the following rules (*SSThresh* represents the slow-start threshold, separating the slow from the fast update):

$$\mathtt{CWnd} = \begin{cases} \mathtt{CWnd} + 1 & \text{if } \mathtt{CWnd} < \mathit{SSThresh} \\ \mathtt{CWnd} + \frac{1}{\mathtt{CWnd}} & \text{otherwise} \end{cases}$$

If a verification is not received before the timeout expires the slow start threshold is updated to keep trace of the size that was reached before an unreceived `ACK`, $\mathit{SSThresh} = \mathit{max}(2, \mathtt{CWnd}/2)$, and the `CWnd` is set to 1. In order to adapt to our problem the above solution, we use a parameter called *Threshold*, analogous to `CWnd`, to represent the current threshold value, and *Delimiter*, analogous to *SSThresh*, to represent the stable threshold that has been reached by the previous verifications. We illustrate the algorithm updating the threshold after a positive and a negative verification.

Negative verification After a negative verification the threshold and delimiters are updated as follows:

$$\mathit{Threshold} = \max(\mathit{Threshold} - \Delta_T, \mathit{Threshold}_{min})$$

$$\mathit{Delimiter} = \max(\mathit{Delimiter} - \Delta_D, \mathit{Threshold}_{min})$$

where Δ_T^- and Δ_D are two updates: smaller ones for the delimiter value and bigger ones for the moving threshold. Possible values, which we used in our implementation, are, $\Delta_D = 0.075$ and $\Delta_T^- = 0.1$.

Positive verification After a positive verification the threshold and delimiters are updated as follows:

$$\mathit{Threshold} = \min(\mathit{Threshold} + \Delta_T^+, \mathit{Threshold}_{max})$$

$$\mathit{Delimiter} = \min(\mathit{Delimiter} + \Delta_D, \mathit{Threshold}_{max})$$

where Δ_D is the same as above, and Δ_T^+ is an update which we kept smaller than Δ_T^- to express the higher weight of negative verifications. Also, Δ_T^+ is in sharply increase when $\mathit{Threshold} < \mathit{Delimiter}$ for fast adjustment of the threshold value. For instance, in our implementation $\Delta_T^+ = 0.05$, if $\mathit{Threshold} < \mathit{Delimiter}$; $\Delta_T^+ = 0.005$ otherwise.

The $\mathit{Threshold}_{min}$ and $\mathit{Threshold}_{max}$ values reflect the impact of the increasing number of votes and identify the range of values that are accepted for the threshold, depending on the number of votes V received. They are dynamically computed by the following rules:

- $\mathit{Threshold}_{min} = 1 - \frac{1}{ln(|V|+e)}$

- $Threshold_{max} = 1 - \frac{1}{ln^2(|V|+e)}$

where e is the basis of the natural logarithm.

In this way we have reached the desired behavior, summed up in the following points:

1. The *Threshold* reaches the value $Threshold_{min}$ after the receipt of several consecutive negative verifications; this is the situation that requires to monitor more closely the received votes, keeping high the probability of vote verification. The *Threshold* should not increase too quickly after having received a single positive verification.

2. The *Threshold* reaches the value $Threshold_{max}$ after the receipt of several consecutive positive verifications; in this situation it is acceptable to decrease the probability of verification. The *Threshold* should not decrease too quickly after having received a single negative verification.

An analysis of the behavior of the algorithm shows that when the number of votes is large, the percentage of the verified votes is reduced to a relatively small value (e.g., it can reach 2% after 1000 votes); at first sight, this could appear a low value, but it is important to observe that the absolute quantity of verified votes is significant, enough to draw a conclusion on vote reliability.

We also observe that when a vote is unsuccessfully verified, we assume that it is a false vote and we subtract from the number of the votes that had been expressed that vote multiplied by the inverse of the difference between 1 and the threshold value used for the random choice. For instance, let us suppose we received the 100-th vote, which is positive; the *Threshold* is equal to 0.95; the random value is greater than 0.95 and the vote is verified. If the vote is not successfully verified, we mark the vote as invalid and remove 20 positive votes from the total ($20 = \frac{1}{(1-0.95)}$). The goal is to make vote falsification a zero-sum game, that statistically neither improves nor decreases the outcome of the poll.

4 Computing the network reputation

Upon reception of the different votes (properly checked and cleaned up of possible fakes), the poller will need to produce an aggregate reputation, which we call network reputation as it reflects the opinion of the network on the servent/resource being inquired about. In our solution such network reputation is computed as follows.

Let V be the set of all votes considered valid for the poll inquiring about servent/resource x and let V^+ and V^- be the set of positive (i.e., with value 1) and negative (i.e., with value 0) votes in V. The network reputation $R_n(x)$ is then computed as:

$$R_n(x) = w^+ \cdot \left(\frac{|V^+|}{|V|}\right)^{|V^-|} + w^- \cdot \left(1 - \left(\frac{|V^-|}{|V|}\right)^{|V^+|}\right)$$

where w^+ (w^-, respectively) is a weight assigned to positive (negative, respectively) votes. The weights can be can be set to different values depending on the relative importance one wishes to give to positive and negative votes. For instance, setting $w^+ = w^- = 0.5$ would grant negative and positive votes the same importance, while the setting $w^+ = 0.4, w^- = 0.6$, which we used in our experiments, consider negative votes as more important as they represent an observed dangerous behavior.

The resulting reputation $R_n(x)$ is a value in the interval [0,1]; value 1 will be produced by the receipt of only positive votes, and 0 by only negative votes. Intermediate values depend on the number of positive and negative votes received and on the weights assigned to them.

5 Merging of reputations

The synthesis of a network reputation from the set of votes collected in the network is the first step. Another aspect that requires consideration and the adoption of fuzzy techniques is the combination of fuzzy values that is needed in the merge of the local and network reputation.

When a servent x is interested in a resource offered by another servent y and x has no previous experience with y, x will be forced to run a poll on y if it desires to estimate the risk of a download from it. When x has instead accumulated a sufficient (be it positive or negative) experience with y, he may decide to forfeit the poll and trust the reputation computed locally from the trace of previous interactions with y. We also envision a third situation, when x has a limited experience with

y, possibly with heterogeneous outcome, and desires to verify y's reputation with a poll. Servent x will then have available two fuzzy values: its local reputation $R_l(y)$ and the network reputation $R_n(y)$. Therefore, x will need to aggregate the two values. We prefer this aggregation mechanism to an involvement of the human user (as already noted, one of our goals is to minimize the complexity of the user interface).

Many aggregators have been proposed for merging truth values in fuzzy logic [7]. Here, we adopt a compensative view by using a weighted average of the two values. Weights are computed according to the number of interactions and votes on which the reputation values have been calculated (the intuition is that the greater this number, the greater is the reputation support) and on the currency of the direct experiences (compared with the age of all the experiences in the personal history).

6 Conclusion

We presented the application of fuzzy logic techniques to the management of reputations in an anonymous P2P environment. This analysis has demonstrated the application of several mechanisms and operators, each adapted to the needs of the specific situation.

References

[1] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proc. of the 10th International Conference on Information and Knowledge Management (CIKM 2001)*, Atlanta, Georgia, November 2001.

[2] D. Bügler. An analysis of gnunet and the implications for anonymous, censorship-resistant networks. In *Proc. of the Workshop on Privacy Enhancing Technologies*, Dresden, Germany, March 2003.

[3] F. Cornelli, E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a P2P network. In *Proc. of the 11th International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[4] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proc. of the 9th ACM Conference on Computer and Communications Security*, Washington, DC, USA, November 2002.

[5] V. Jacobson. Congestion avoidance and control. *Computer Communication Review*, 18(4):314–329, August 1988.

[6] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proc. of the 12th International World Wide Web Conference*, Budapest, Hungary, May 2003.

[7] G.J. Klir and T.A. Folger. *Fuzzy Sets, Uncertainty, and Information*. Prentice-Hall, 1988.

[8] A. Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, March 2001.

[9] D.M. Pennock and M.P. Wellman. Graphical representations of consensus belief. In *Proc. of the 15th Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, July 1999.

[10] W. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.