A Protocol For Pervasive Distributed Computing Reliability

Alberto Ferrante, Roberto Pompei, Anastasia Stulova, Antonio Vincenzo Taddeo [1]
ALaRI, Faculty of Informatics, University of Lugano, Lugano, Switzerland
{ferrante, pompeir, stulovaa, taddeo}@alari.ch

***Abstract*—The adoption of new hardware and software architectures will make future generations of pervasive devices more flexible and extensible. Networks of computational nodes will be used to compose such systems. In these networks tasks will be delegated dynamically to different nodes (that may be either general purpose or specialized). Thus, a mechanism to verify the reliability of the nodes is required, especially when nodes are allowed to move in different networks. In this context, the reliability of nodes is determined by their ability to execute the tasks assigned to them with the promised performances.**

This paper proposes a protocol to evaluate the reliability of the different nodes in the network, thus providing a trusting mechanism among nodes which can also manage the soft/hard real-time constrains of task execution. Some simulation results are also shown to help describing the properties of the protocol.

I. Introduction

Future pervasive systems will adopt new architectures to be both power efficient and capable of performing a large variety of operations. Several European projects are focusing on enabling components for the next generation of pervasive systems by addressing different topics ranging from low-power design [1] to self-adaptability and reconfigurability [2].

One of the ideas is to use distributed computational nodes to compose such pervasive systems. These nodes may be highly specialized or general purpose; each of them may be statically configured, reconfigurable, or even self-adaptive. This structure will give the new pervasive systems great flexibility. Nodes (or sets of nodes) will be able to join different networks depending on the place where the pervasive device is moved. Those nodes will enrich the newly joined networks with their functionalities. One key element of this architecture is the possibility to distribute tasks over the nodes dynamically. Each node, once it has been awarded with a task to perform, will be able to delegate parts of the task to other nodes. Tasks may be assigned by using different criteria depending on the kind of network and nodes considered. Task delegation, though, requires some support mechanisms to be effective. One of the main support mechanisms required is the evaluation of reliability of nodes. A node is said to be non-reliable when it accepts a task to be executed and it does not execute it properly. This may happen for different reasons (non correct evaluation of the free resources, faulty hardware in the node, ...) and in different ways (results delivered late, wrong results, ...).

[1] Authors appear in alphabetical order.

More in general, the problem of trust among nodes is quite important for this kind of systems to work in a reliable way. Trust, in this environment, can be seen in two different ways: the possibility to verify the identity of the nodes and the possibility to verify that nodes are reliable before assigning tasks to them, as aforementioned. Both of these mechanisms are necessary to guarantee the reliability of the system: before assigning a task to a node its reliability needs to be verified to avoid bad nodes (i.e., nodes that do not deliver the required results or do not deliver them in time). Identity verification is a necessary step in reliability verification.

In this paper we address the problem of reliability evaluation for nodes. The aim of this work is to provide a lightweight solution that can be implemented both in software and in hardware. The trust value must be kept up-to-date during the lifetime of the system, but the update operations must be simple enough to be executed with constrained resources. However, the trust protocol should provide protection against possible system faults and attacks. The problem of identity verification is outside the scope of this paper. Though, some considerations about this topic are provided in Section II.

In the remaining part of the paper we describe the trusting protocol we have developed. Some simulations results are presented to show the properties of the protocol. Section II presents some related works; the description of the protocol is contained in Section III. Section IV describes the simulation model and shows some simulation results. Section V discusses the behavior of the systems when a number of attacks are performed against it.

II. Related Work

In this section we provide an overview on works addressing the problem of trusting in distributed networks. A general methodology for trust management in distributed computing can be found in [3]. The authors categorize the trust relationships into two layers. In particular, the trust mechanisms can be reputation-based or credential-based. In the first case, a node behavior is evaluated by considering its reputation with the other nodes in the system; in the second case, the trust mechanism restricts the access to resources and services based on credentials of nodes. According to the classification proposed in [3], our trusting protocol can be classified as reputation based and it refers to the delegation trust in a decentralized manner.

978-0-7695-3393-3/08 $25.00 © 2008 IEEE
DOI 10.1109/WiMob.2008.23

Another work on trust management has been presented in [4]. The authors underline the benefits of introducing trust into distributed networks by analyzing the establishment mechanism, its vulnerabilities, and performances in a scenario were a routing protocol is secured.

In [5] a trusting protocol is described: different trust levels are assigned to different network nodes; these values are updated by using the reports of intrusion detection systems located in each node. The trust levels are used to evaluate the security of message routing. In the aforementioned paper, unlike in our work, the trust computation is only based on indirect observations.

The problem of identity verification has also been addressed for peer-to-peer networks. An example of a trusted authentication protocol is shown in [6]: the authors propose a decentralized mechanism to authenticate the nodes based on threshold blind signatures which are used to certify pseudonyms of nodes. Once a pseudonym is assigned and validated by other signatures, its value is saved and used for all successive interactions without any runtime update.

In [7] a reputation mechanism for P2P file-sharing is proposed to assess both the reliability of a resource to download and its provider. Such a mechanism is based on a distributed polling algorithm that should reduce the spreading of bad contents and should recognize malicious nodes. The last two approaches require many public key verifications. Those operations are resource and time consuming and, thus, not suitable for limited-resource devices.

In grid networks some reliability evaluation mechanisms have been already developed; some of these mechanisms are compared one against the other in [8]. All of these systems rely on the high connectivity and on the computational resources available in grid systems; in general, these conditions will not apply to our nodes that will be designed to be used in pervasive systems. Furthermore, the execution of hard or soft real-time tasks may be required on this kind of systems. None of the already developed protocols address these problems and this is the main motivation for developing a new protocol. In this work we have been inspired both by the reliability check protocols used in grid systems, such as the weighted feedback one [8] [9] and by the peer-to-peer trusting protocols, such as PGP [10, pp. 37-63] [11].

III. The Trust Protocol

The reliability of nodes can be evaluated in different ways, but, in general, it can be considered as the capability of nodes to respect a service agreement. This is a particular procedure that lies behind the identity certification or the encryption process. In the remaining part of this section, the word trust is used to identify the reliability of nodes. However, the protocol presented here can be easily extended to incorporate identity checking and trusting in the classic sense.

As shown in Figure 1, task delegation is a process composed by different steps. The delegating node sends a query to nodes nearby with details on the task to perform and the constraints to satisfy for its execution. All nodes in the environment reply

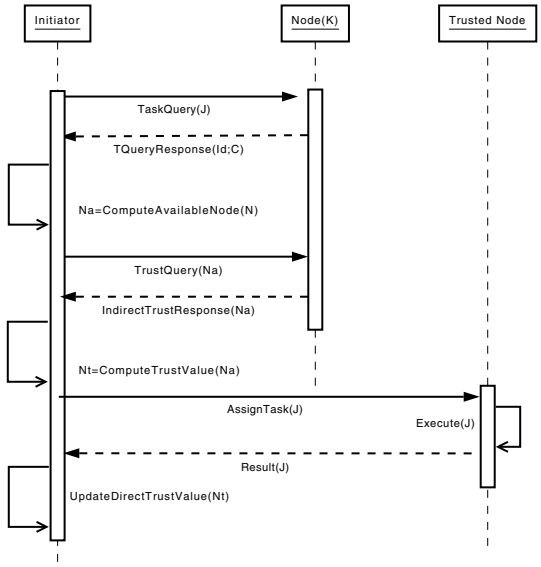

Fig. 1. Task delegation protocol.

by providing the information on the possibility to execute the required task; if any identity check mechanism is required by the security policy enforced in the system, the identity is checked at this step by means of proper certificates. A second query is then performed to collect trusting information on the available peers. All the nodes in the network (or part of them) reply to this query with the personal trust value that each one of them has on the available nodes; this value is used by the delegating node as indirect experience (i.e., non-directly collected reliability information about the other nodes in the network). The algorithm can proceed by calculating the trust values of all the involved available nodes. For this, both personal experience (i.e., the evaluation of previous task executions by the nodes that is being evaluated) and indirect experience are used. Such a trust information determines the delegation of the task to the highest trustworthy node. To avoid a too high overhead for trust computation, the number of nodes considered may be limited to a certain threshold.

Once the execution of a task ends, the results are sent back to the delegating node and they are evaluated on the light of the related service agreement. The personal trust value related to the involved nodes is then updated based on result evaluation; this operation is done by comparing the obtained performance with the parameters specified in the service agreement. Timing constraints on single execution tasks and frequency constraints on continuously executed tasks are both included. For periodic tasks, the real-time concept is strictly related to the different specific applications; a good approach is to reduce it to a constraint on frequency. The strictness applied to real-time constraints can be reduced to two levels: hard and soft.

A. Trust data collection

The trust data collection process is started after a delegation request by the trusting node; more precisely it is started after

the reception of answers from the other nodes in the network. The nodes available for executing the considered task are said to be *active*.

In the general view of the system, each node in the network may have information about any of the other nodes. The knowledge of the network can be used during trust value computation. Thus, for computing the trust value of a node X_j as seen by the node X_d, both the direct personal experience of X_d with X_j and the indirect experience of all the other nodes in their interactions with X_j should be used. The *direct trust* value summarizes the personal experience of the node X_d. The *indirect trust* value, instead, summarizes the experience, regarding node X_j, collected from the other nodes in the network.

The indirect trust value of the node X_j can be computed by the node X_d as follows:

$$IE_j^d = \sum_i V_j^i, \;\; i,j \in [1,N]; \;\; i \neq j; \;\; i \neq d \qquad (1)$$

where N is the number of nodes in the network and i varies in the set of the mediating nodes only. V_j^i is the direct experience of the node X_i with the X_j one.

The trust value of a node X_j as seen by X_d can then be computed as follows:

$$T_j^d = w_p \times V_j^d + w_i \times \frac{\sum_i V_j^i}{m} \;\; i,j \in [1,N] \;\; i \neq j; \;\; i \neq d \qquad (2)$$

also in this equation i varies in the set of the mediating nodes only; w_p and w_i are weights that are used to quantify how much the trust value is affected by the personal experience and by the indirect experience; the sum of w_p and w_i must be 1. $m \geq 1$ is the number of mediating nodes.

The trust levels are numbers contained in predefined ranges. Thus, the ranges must be carefully evaluated to obtain a cheap enough implementation of the protocol. For this work a 16-level, 4-bit trust value has been chosen. The level 0 corresponds to the complete ignorance state; the level 15 identifies a node that is fully trusted.

The procedure explained above to compute the trust value of a node is not suitable for a large number of nodes: in that case it would be required to store a large number of trusting values both direct and indirect. The simplest solution is to limit the evaluation of trusting only to the first k nodes which provide their evaluation on a given active node; k has to be properly selected according to the system properties. Further investigations on this are being performed.

B. Task Delegation

During the delegation process, the trust value must be evaluated in the light of the kind of task that is being delegated. By assuming that a failure on a hard constrained task can cause worse consequences than on a soft constrained one, its delegation could be restricted to all the nodes that have higher trust levels; soft constrained tasks can be executed by all nodes. A set of thresholds for the different categories of tasks is the following: a trust value of 14 for hard real time tasks; 12 for hard deadline tasks; 9 for soft real-time tasks;and 2 for soft deadline tasks. These thresholds are also shown in Figure 2. The threshold definition is system-dependent: it can be defined either at system level or in every single node and may be different from the one used in our case.

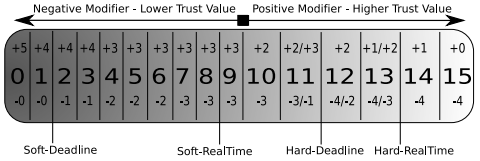

Fig. 2. Modifiers for the trust values.

The delegation of soft constrained tasks to nodes with low levels of trust may become necessary when no enough trusted resources are available. This approach also gives the non-trusted nodes the possibility to gain trust when they behave well.

The dynamic property of the nodes makes the condition of complete ignorance about the environment very common. In this case an initialization procedure for the trust values should be started. The easiest way is to randomly delegate a unconstrained task to different nodes and to update their trust values according to the results obtained. Other approaches that try to reduce the number of iterations necessary assign trust values can also be used. For example, the trusting node can compute it by using the indirect trust values only (i.e., by putting $w_p = 0$). Another way is to initialize the personal experience to a predefined value (e.g., 7). The trust values will anyway converge quickly to the correct values, as shown in Section IV.

C. Trust Value Update

Each node awarded of a task to be executed, is evaluated each time it sends back results to the trusting node. This evaluation consists of a comparison between the negotiated deadline and the time at which the results arrived. Frequency constraints are more complex to be evaluated, but as said before, they can be considered as time constraints applied periodically. Every time a result related to a periodic task is received, its related counter is incremented. This counter is checked at proper intervals of time to compute the average frequency reached in that period; the counter is then reset. If frequency obtained by means of these operations is equal or higher than the required frequency then the constraint is respected, otherwise it is not. No evaluation is performed on the correctness of the results. Though, error detection techniques can be used and the results evaluation can be easily extended to also incorporate this check.

The history of the trust relationships between the trusting node and the trusted nodes is completely kept in the personal trust value. Such a value is kept and updated by the trusting node every time an task delegation ended and results evaluated. Thus, the update process, which is the last of the communication activities, is a fundamental part of the whole algorithm. The update process should be simple, as it needs to be repeated for each received result.

The personal trust value of the node X_j is updated by using the following formula:

$$V_j^d(t') = V_j^d(t) + M \tag{3}$$

where M is one of the modifiers of Figure 2. Positive/Negative modifiers are used for positive/negative result evaluations. A different set of modifiers (shown in Figure 2) can be used for tasks with hard constraints. It has been chosen in such a way that each failure will make the trust level go under the threshold for executing tasks with hard constraints. This choice has been taken because it is supposed that if the node fails to execute a very important task, it needs to regain its trust before having another real-time task assigned.

IV. Simulations

In this section we describe some of the experiments we have performed on a simulated system. Our goal was to test the trusting protocol we have developed and to better study its properties. Some preliminary simulations helped us in improving the protocol and in tuning some of its parameters. Though, some further improvements can be added, as described later in this paper.

In the remaining part of this paper we briefly describe the simulation model we have used and the results we have obtained.

A. Description of the Model

The system has been simulated by means of a SystemC model. The SystemC language [12] was selected to describe our model as it provides the capability to easily describe concurrently running components as the ones we are proposing in our model. In fact, this language was thought to model hardware/software systems.

The model simulates the network message exchanges necessary to compute the trust values only.

The computational network that we have modeled is composed of 35 nodes. The network is composed of general purpose processors. As mentioned before, this may not be true in a real life case as also specialized processors may be present. Though, this does not make any difference in the trust protocol. On the present model the delays related to the different operations (e.g., network communications) are not modeled; for this reason we did not measure any performance figure. Our intent was just to study the evolution of trust values in different conditions and the level of protection provided by the protocol against faulty nodes.

The constraints of the different delegating nodes for task execution are randomly generated; these values are evaluated from the active nodes and compared to their available resources. A different level of faultiness (i.e., the percentage of interactions in which the node behaves in a bad way) can be simulated for every node. Though, the instants of time in which faults occur follow a random distribution.

All the tasks considered in our simulations only had soft deadlines. In all the simulations we assumed w_p and w_i of Equation 2 to be 0.7 and 0.3, respectively.

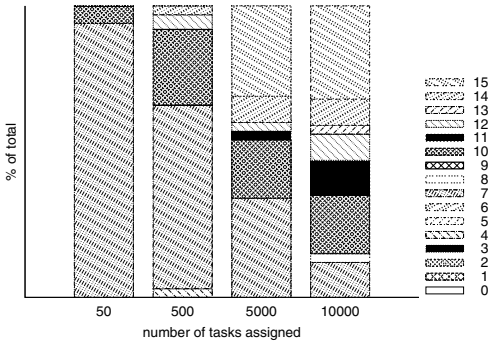

Fig. 3. Trust value evolution of a 0% faulty node.

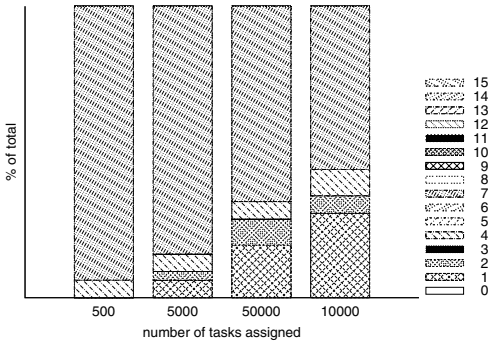

Fig. 4. Trust value evolution of a 100% faulty node.

The trust value for all nodes has been initialized at 7; this value then evolves by following the modifiers shown in Figure 2.

B. Simulation Results

Different experiments have been performed on the system. The goal of these experiments was to explore the properties of our protocol with respect to trusting among nodes.

Figure 3 shows the trust evolution of a 0% faulty node (i.e., all the assigned tasks are completed in time), n_0, as seen by all the other nodes in the network. The evolution of its trust value is studied over the total number of tasks globally assigned on the network. After the assignment of the first 50 tasks (of which 2 awarded to n_0) 6% of the nodes increased the trust value of n_0 to 10. After 500 tasks (of which 17 assigned to n_0) 34% of the nodes increased the trust value of n_0. Though, a small number of nodes (3%) evaluate n_0 in a very bad way (trust value equal to 4). This is not due to a flaw in the protocol, but to the problem of conflicting task assignments: a node declare its availability for two tasks and both of them are awarded to the node, but the node itself does not have enough resources for completing both. After the assignment of 5,000 tasks (of which 172 assigned to n_0) only 34% of the nodes evaluate n_0 with the initial value of 7; all the other nodes rate it at least 10. After 10,000 assigned tasks, 85% of the nodes rate n_0 at least 10. 32% of the nodes rate it 15.

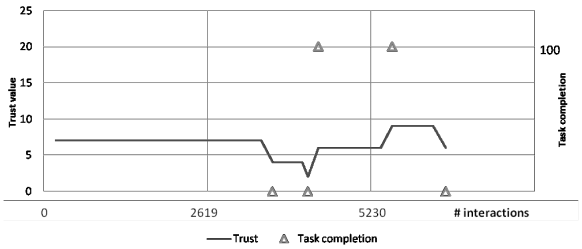

Fig. 5. Trust value of a 50% faulty node as computed by another node in the network. Time measured in number of assigned tasks in the network.

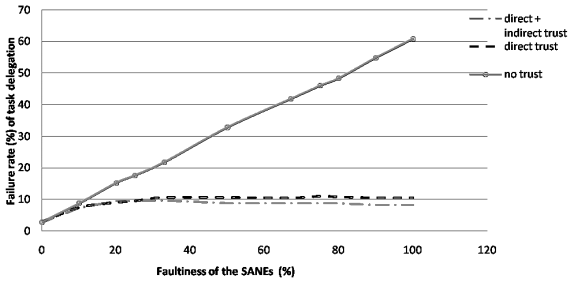

Fig. 6. Failure rate of the network of nodes when 50% of the nodes are 10% faulty and all the others have different levels of faultiness (x axis).

Figure 4 shows the trust value evolution for a node that is 100% faulty (i.e., it does not complete any of the tasks assigned to it). This node is named n_{100}. The time scale is again the total number of tasks globally assigned in the network. After the assignment of 50 tasks (of which 2 awarded to n_{100}) 6% of the nodes decreased the evaluation of n_{100} to 4. After 50 tasks (of which 13 awarded to n_{100}), 15% of the nodes rate n_{100} below 5; 9% of the nodes rate it 1 or 2. After 5,000 tasks (of which only 37 awarded to n_{100}), 33% of the nodes rate n_{100} below 5; 27% of the nodes rate it 1 or 2. After 10,000 tasks (of which only 58 awarded to n_{100}), 44% of the node rate n_0 below 5; 35% of the nodes rate it 1 or 2. As can be noted by the aforementioned data, the convergence of the negative opinion on a node is a slow process. This is due to the fact that very few tasks are assigned to the faulty node due to the recommendation system.

Figure 5 shows the trust value evolution over time for a node that is 50% faulty. This trust value is the one measured by another node in the network. The time considered is again the number of tasks assigned globally in the network. As shown by the graph, the trust value is correctly staying around the average value (7).

Figure 6 show the global task failure rate in the network when 18 of the 35 nodes are not faulty at all and the other nodes have variable levels of faultiness (from 0 to 100%). The graph compares the results obtained when no trust protocol is used with the ones obtained when our trust protocol is adopted. The trust protocol is used in two flavors: with or without considering indirect experience. The use of the trust protocol provides a dramatic reduction of the unsuccessful tasks. In fact, tasks tend to be assigned to non faulty nodes, if any are available. The faultiness of the network increases proportionally to the faultiness considered for the faulty group of nodes when no trust protocol is used; it increases with a lower slope when the trusting protocol is used. The solution in which also indirect trust is used outperforms all the others. Different values for w_i and w_p can be used to give more weight to the indirect trust value. In this case, the number of failing tasks further decreases. Thus, it makes the system more vulnerable to some attacks, as explained later in the following section.

V. Simulated Attacks

In Section IV we have shown how the trust protocol behaves in normal circumstances. In this section we show the behavior of the system when different kinds of attacks are performed on it; furthermore, we propose some solutions to the possible problems caused by some of these attacks. The attacks considered here are the ones called *bad mouthing*, *Sybil*, *on-off*, and *conflicting behavior*. We do not discuss denial of service attacks.

1) Bad Mouthing Attack: The *bad mouthing* attack refers to malicious nodes that provide wrong recommendations. The attack can be performed either by providing negative recommendations of a well behaving node, or by providing positive recommendations of a non well behaving node. The second variant is the most dangerous as it makes the nodes in the system trust a faulty node.

In our simulations we considered a number of attackers providing a trust value of 1 for a target node to evaluate how much a malicious mediating node can decrease the efficiency of the trusting protocol. For each of these simulations we considered 5000 task assignments with 35 nodes in the network. Scenarios with 0, 1, 5, and 10 malicious nodes have been considered. As a consequence of these attacks, the number of tasks assigned to the target node is 347, 307, 289, and 46, respectively. The number of delegations slowly decreases when the number of negative recommendations increases. Thus, the protocol is robust against such an attack. In our scenario the task assignment procedure starts to be significantly influenced by the attack when at least 30% (i.e., 10) of the nodes are malicious. In this case, tasks are assigned to other available nodes that are not subject to the attack.

The protocol can be enhanced in different way to provide protection against this kind of attacks. A history of all direct delegations to a node can be used to perform a more accurate analysis of its behavior. Moreover, we can increase the weight of direct experience, w_p, in Equation 2 to reduce the effects of recommendations.

2) Sybil Attack: If a malicious node can remove its bad trust history by registering as a new node, the system is subject to the so called *Sybil attack*. Our trusting protocol do not consider any specific protection against this attack. Though, as described above, the reliability protocol should be coupled with a proper identity trusting protocol. The authentication of the identity prevents Sybil attacks, as discussed in [6].

3) On-Off Attack: When a malicious node behaves alternatively bad and well, we are in presence of an *on-off attack*. Aim of this attack is to try hiding a malicious node from the detection mechanism. This is done by taking advantage of the dynamic evolution of trust in the time domain: the behavior of a node keeps changing from good to bad. This way the node can compromise the overall performance of the distributed system. Two different aspects determine how much a distributed system is exposed to this kind of attacks: the weight of a good/bad action in the trust update procedure (i.e., how fast a node becomes highly trusted/untrusted) and how much a failure in executing a task influences the performance of the system.

The modifiers used in our protocol for positive and negative evaluations of task executions are shown in Figure 2. By looking at Figure 3 and Figure 4 we can notice that gaining an high trust value takes more steps than losing it. In particular, we can notice that non-well behaving tasks tend to be isolated quickly. Moreover, Figure 5 shows that if a node behaves inconsistently well and bad (50% faulty), its trusting value stabilized around the value of 7.

4) Conflicting Behavior Attack: We have a *conflicting behavior attack* if a node behaves in different ways with different peers; this may affects the relative trust opinion among the peers. In practice, due to this inconsistency in behavior, one node (N_1) can have judge the malicious one (M_x) as trusted; the same malicious node may be evaluated as untrusted by another peer, N_2. If the trusting evaluation of nodes was also influenced by their reliability in providing recommendations, the conflicting behavior of M_x with N_1 and N_2 would negatively affect their relationship.

In our trusting protocol the reliability of nodes in providing recommendations is not evaluated. Thus, this attack does not have any influence on our distributed system. In our protocol, nodes that have an inconsistent behavior will be classified as trusted by some nodes and non-trusted by some others. The recommendation system helps in classifying the malicious nodes as partly bad (the number of nodes providing a negative opinion depends on the behavior of the malicious node). Thus, the number of tasks assigned to them will be highly decreased.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have described a protocol aimed at protecting a network of computational nodes from the faulty and malicious ones by evaluating their trustworthiness. The protocol was designed to be as efficient as possible from the computational resources stand point; more complex and effective security mechanisms can be implemented at the price of added computational resources.

We performed some simulations that allowed us to better study the properties of the protocol and to simulate some well known attacks that can be performed against distributed computational systems. The simulations showed that the protocol provides the capability to isolate faulty and malicious nodes in the network. The protocol is also able to mitigate the effects of most of the attacks. Further protection can be added by slightly modifying the protocol.

Further work will be performed for improving the trusting protocol and for better tuning its parameters; the simulations will be extended to support experiments with mixed kinds of tasks and with different values for the w_p and w_i parameters to better study the influence of indirect trust on the system. Additionally, more effort will be put in improving the protection provided against the different attacks discussed in the paper.

VII. ACKNOWLEDGMENTS

This work was partially supported and funded by the European Commission under the Project AETHER (No. FP6-IST-027611). The paper reflects only the authors' view; the European Commission is not liable for any use that may be made of the information contained herein.

REFERENCES

[1] LoMoSA+ Consortium, "Lomosa+ project website," 2005, http://www.lomosa.org.

[2] Gamrat C. and Philippe J-M., "Self-Adaptive Embedded Technologies for Pervasive Computing Architectures: SANE Concept - Extended Abstracts," in *Directions in FPGAs and Reconfigurable Systems: Design, Programming and Technologies for adaptive heterogeneous Systems-on-Chip and their European Dimensions, DATE*, 2007.

[3] Weiliang Zhao, Vijay Varadharajan, and George Bryan, "General Methodology for Analysis and Modeling of Trust Relationships in Distributed Computing," in *Journal of Computer (JCP)*, ACADEMY PUBLISHER, Ed., vol. 1, no. 2. ACADEMY PUBLISHER, May 2006, pp. 42–53.

[4] Y. Sun, Z. Han, and K. Liu, "Defense of trust management vulnerabilities in distributed networks," *Communications Magazine, IEEE*, vol. 46, no. 2, pp. 112–119, February 2008.

[5] Z. Liu, A. Joy, and R. Thompson, "A dynamic trust model for mobile ad hoc networks," *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, pp. 80–85, 26-28 May 2004.

[6] D. Quercia, S. Hailes, and L. Capra, "Tata: Towards anonymous trusted authentication." in *iTrust*, ser. Lecture Notes in Computer Science, K. Stlen, W. H. Winsborough, F. Martinelli, and F. Massacci, Eds., vol. 3986. Springer, 2006, pp. 313–323. [Online]. Available: http://dblp.uni-trier.de/db/conf/itrust/itrust2006.html#QuerciaHC06a

[7] E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2002, pp. 207–216.

[8] J. D. Sonnek and J. B. Weissman, "A quantitative comparison of reputation systems in the grid," in *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 242–249.

[9] F. Azzedin and M. Maheswaran, "Evolving and managing trust in grid computing systems," *Electrical and Computer Engineering, 2002. IEEE CCECE 2002. Canadian Conference on*, vol. 3, pp. 1424–1429 vol.3, 2002.

[10] P. Zimmermann, *An Introduction to Cryptograpy*. Network Associates, 1999, available at: ftp://ftp.pgpi.org/pub/pgp/6.5/docs/english/IntroToCrypto.pdf.

[11] P. Feisthammel. (2004) Explanation of the web of trust of PGP. http://www.rubin.ch/pgp/weboftrust.en.html.

[12] "SystemC Official Website," http:/www.systemc.org/, 2008.