Journal of Mobile Multimedia, Vol. 5, No.1 (2009) 064-080

DETERMINISTIC TRUST MANAGEMENT IN PERVASIVE COMPUTING

MIESO K. DENKO
Department of Computing and Information Science
University of Guelph, Guelph
denko@cis.uoguelph.ca

TAO SUN
Department of Computing and Information Science
University of Guelph, Guelph
tsun@uoguelph.ca

ISAAC WOUNGANG
Department of Computer Science
Ryerson University, Toronto
iwoungan@scs.ryerson.ca

Received June 2, 2008
Revised September 1, 2008

An effective trust management technique plays a vital role in evaluating relationships among devices in pervasive computing. In this paper, we propose a deterministic trust management scheme that aims at establishing trust relationships among devices using direct and indirect computation methods. Recommendations and trust updating mechanisms are used to increase the reliability of trust computations. We have carried out performance evaluations using simulation experiments. The results show that trust management with recommendation outperforms other schemes.

Key words: Trust, trust management, deterministic trust

1 Introduction

Pervasive computing [1] is an emerging research field that brings revolutionary paradigms for computing models. It integrates the physical world with the information world, and provides ambient services and applications that allow users, devices and applications in different physical locations to communicate seamlessly with one another. In a pervasive computing environment, the devices that are interconnected and embedded in physical objects collect, process and transmit information without human intervention.

Trust takes on a significant role in pervasive computing security and privacy. As a result, the importance of security and privacy has risen to an unprecedented level. In such a decentralized open environment, different devices interact without prior knowledge of each other. These environments must provide effective security and privacy mechanisms to protect data and ensure the quality of

interactions. Without human judgment, devices need to distinguish other peers' identities and behaviours autonomously in the pervasive computing environment. In such an environment, it is not merely sufficient to authenticate devices because most of these devices are unknown and usually, there is no central management mechanism [2].

Trust provides devices with a natural way of judging other devices, similar to how we have been handling security and privacy in human society. With the help of trust, a device can evaluate or predict the security and quality of potential interactions with the other devices before those interactions actually happen. It can draw a conclusion on whether the interaction should take place, and decide what type of data can be transmitted securely. Trust management enhances security and privacy for devices in pervasive computing environments. It can also improve the efficiency and quality of communications among devices.

The main contributions of this paper are: (a) the design of an architecture for distributed trust management (b) the implementation of various modules for trust computation and maintenance, and (c) the performance evaluation of the proposed scheme at various parameter settings.

The rest of this paper is organized as follows: Section 2 presents existing related work. Section 3 presents the proposed trust management scheme. Section 4 provides the performance evaluation. Finally, Section 5 concludes our work and presents some ideas for future works.

2 Related work

Trust is not a concept unique to pervasive computing. It has been widely used in many other disciplines. Paper [3] discusses the differences and similarities between the concepts of trust in social networks and in computer networks. Two models, referred to as entropy-based model and probability-based model, have been proposed to protect the devices in distributed networks from several typical attacks. The authors in [4] designed a trust model to improve mobile ad hoc networks' collaborations based on a Bayesian network. This model uses trust as a reference to choose the identity in the mobile ad hoc network that can provide best quality of service (QoS). The solution proposed in [5] also aims at providing the best possible QoS to devices in a pervasive computing environment. In paper [6], the authors introduced a centralized approach termed as Beta Reputation System, which employ the statistical basis of Beta distributions to build a trust between devices in a network. However, their proposed centralized approach cannot be used for decentralized applications such as pervasive computing.

When designing a trust management solution in pervasive computing, one should determine the specific characteristics of trust in this field. Several important topics related to the role of trust in pervasive computing were discussed in [7], including how trust in pervasive computing is different from that in other fields, and how trust solutions should be evaluated correctly. Trust issues that devices face when communicating with each other in a pervasive computing environment were also discussed in [8]. In paper [9], the authors proposed a user-centric trust network derived from an individual’s social network, and they used it to represent the degree of trust and ensure users' privacy in the pervasive computing environment. However, they do not provide details on how to quantify the trust. In [10], a trust-based security solution designed for pervasive computing environments was proposed. This solution uses a chain to transfer trust when providing services, and users can delegate only to other users that they trust.

As a probability theory, Bayes' theorem can also be used to construct trust models. In [11], a Bayesian trust model is inspired by the probabilistic view of trust. By means of Bayesian learning, the model makes predictions based on both the outcomes of interactions and identities' previous trust information. In [12], a framework called B-trust is proposed for pervasive computing. A multi-level discrete Bayesian theorem-based trust is developed for trust formation and updates. In [13], the authors proposed a trust model using the naive Bayes classifier, a probabilistic classifier based on Bayes' theorem. In their model, recommendations from peers provide information to enhance the decision-making regarding trust. The authors also introduced the time's effect on the recommendations' values, and they proposed a time-based evaluation strategy accordingly.

Pervasive healthcare is one application of pervasive computing. Paper [14] discusses the trust's role in several typical pervasive healthcare scenarios. For instance, wireless sensors can be used to monitor patients and transfer health information to remote healthcare providers. It is important to establish trust among users, providers and medical staff when multiple healthcare providers and security domains are involved in pervasive healthcare. In [15], a trust negotiation protocol is implemented for pervasive healthcare. The directed graph concept is used to construct trust relations in pervasive healthcare. The nodes in the directed graph represent the different types of actions between entities, and the edges connecting nodes represent the routes by which the action can occur based on trust relations.

3 Proposed Trust Management Scheme

In this paper, we extend the trust management scheme for the pervasive computing environment that we proposed in [16] by providing details and improvements to the proposed scheme's principal components. We compare the distinctions between the theory and current applications in pervasive computing. Unlike previous works, we rationalize the trust management by connecting the trust management with the requirements of actual applications in pervasive computing environments.

3.1 Overview of the Proposed Scheme

3.1.1. Definition of Trust

Trust has been defined in several different ways, aimed at different applications [4], [17], [18], [19]. For our trust management scheme, we define trust based on the definitions inherited from papers [4] and [18]: the trust that device A places in device B is the strength of device A's belief that (1) device B will behave without malicious intent, and (2) the service or interaction that device B provides will satisfy device A's request.

This definition shows that, in our proposed trust management scheme, trust is not only a measure of a device's faithfulness, but it is also an indicator of the quality of service that a device provides. The motivation for such a definition of trust is based on the fact that providing adequate quality of service and preventing malicious attacks are important objectives of pervasive computing. The trust value is used to determine the level at which one device trust another device. Trust values range from -1 to 1. $T_A(B) = 0$ indicates that Device A has no trust information on device B, $T_A(B) > 0$ indicates that device A considers device B trustworthy, and $T_A(B) < 0$ indicates that device A considers device B untrustworthy. $T_A(B) = 1$ indicates a complete trust, and $T_A(B) = -1$ indicates a complete mistrust.

3.1.2. Main Components of the Proposed Scheme

Our proposed trust management scheme is shown in Figure 1.

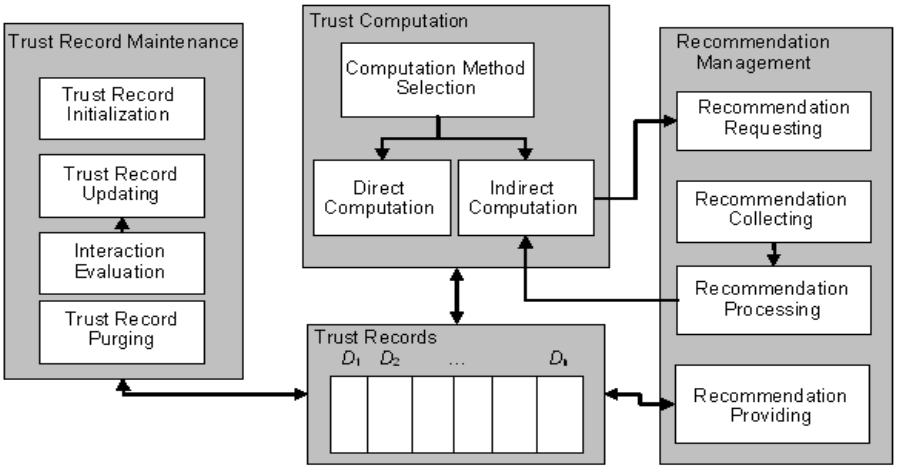

Figure 1. The proposed trust management architecture

This scheme is composed of four main modules, described as follows:

a) ***Trust Computation Module***: This module performs the trust computation, which is the most important function of our trust management scheme. This module includes three sub-modules: (i) the computation method selection, (ii) the direct computation, and (iii) the indirect computation. The computation method selection sub-module chooses a suitable method for computing trust values. The direct trust computation is performed when the direct observation is adequate; otherwise, the indirect trust computation is invoked to obtain the trust value. Performing trust computations is the responsibility of the direct computation and indirect computation sub-modules.

b) ***Trust Records Module***: All trust values are stored in this module. Trust values are provided by the Trust Computation Module, and are maintained by the Trust Record Maintenance Module.

c) ***Trust Record Maintenance Module***: This module takes on the responsibility of maintaining the trust values in the Trust Records Module. In this module, the Trust Record Initialization sub-module sets up a new entry in the Trust Records Module for an unknown device. The interaction evaluation sub-module performs the evaluation after the interaction. Then, the Trust Record Updating sub-module updates the corresponding entry in the Trust Records Module based on the result provided by the Interaction Evaluation sub-module. The Trust Record Purging sub-module deletes the outdated trust values in the Trust Records Module.

d) ***Recommendation Management***: This module has two principal functions: (i) requesting recommendations, and (ii) providing recommendations to other devices. The Recommendation Requesting sub-module is controlled by the Indirect Computation sub-module within the Trust Computation Module. It broadcasts the request for recommendations to other devices in the environment. Then, the Recommendation Collecting sub-module

collects the recommendations. The recommendation value is sent to the indirect computation process after the received recommendations are processed by the Recommendation Processing sub-module. Upon the request of other devices, the Recommendation Providing sub-module picks the desired trust value from the Trust Record Module and sends it out as a recommendation.

3.2 Trust Computation

3.2.1 Direct Trust Computation

A direct trust is computed when two devices that are attempting to interact with each other have no experience with each other. The direct trust computation is based on direct observation, which may come from personal identification or the identity information embedded in devices. A user's identification is embedded in the smart device the user is carrying. It can also be obtained from a specific identification device, such as an electronic badge. If device *A* can collect enough trust information about device *B* through direct observation, it will use its direct observation value as the trust value it places in *B*; i.e., $T_A(B) = O_A(B)$, where $O_A(B)$ is the value obtained through the direct observation.

3.2.2 Indirect Trust Computation

When $O_A(B)$ is smaller than a predefined threshold $Thrd_{DO}$ which is between 0 and 1, the direct observation is not enough to satisfy the direct trust computation. Device *A* needs external assistance to obtain enough trust information to perform a trust computation on the unknown device *B*. Other devices in the environment might have experiencing interactions with device *B*, which is unknown to device *A*, so they have kept trust values on device *B* in their trust management systems. In this situation, device *A* can make use of the trust information stored in other devices to perform its trust computation. In our proposed scheme, this type of external trust information is called a *recommendation*, and this type of trust computation that takes advantage of these so-called recommendations is called an *indirect trust computation*.

An indirect trust computation is performed with inadequate direct observation and recommendations as reinforcement. In the situation where the direct observation cannot be initiated, the indirect trust computation can totally rely on recommendations. Since the trust information contained in recommendations is from the interaction experienced by other devices, it works partially in the case of indirect trust computation. Based upon the above considerations, the trust value that device *A* puts on device *B* can be computed as follows, given that the direct observation and recommendation values are taken into account with different weights.

$$T_A(B) = \begin{cases} \omega_1 * O_A(B) + \omega_2 * R_A(B) \\ \quad if\ O_A(B) \neq 0, 0 < \omega_1 \leq 1, and\ 0 < \omega_2 \leq 1 \\ \\ \omega_3 * R_A(B) \\ \quad if\ O_A(B) = 0, where\ 0 < \omega_3 \leq 1 \end{cases} \tag{1}$$

In Equation (1), $R_A(B)$ is the recommendation value based on recommendations provided by other devices in the environment. The recommendation value is computed as follows:

$$R_A(B) = \frac{1}{n}\sum_{i=1}^{n} T_{D_i}(B) \quad with\ T_A(D_i) > 0 \qquad (2)$$

Other devices D_i provide their trust values on device B as recommendations. The condition $T_A(D_i) > 0$ shows that device A accepts only the recommendations from devices that it has evaluated as somehow trustworthy. By doing this, a device can avoid receiving false recommendations, and hence ensure the veracity of the trust computation.

The coefficients ω_1, ω_2 and ω_3 are the weights based on which a device scales the direct observation and recommendation values. The sum of ω_1 and ω_2 is equal to 1, and ω_1 is greater than ω_2, showing that the direct observation is more important than the recommendations issued from other devices. The indirect trust computation depends on recommendations when no trust information is available through direct observation. The coefficient ω_3 is greater than ω_2 because recommendations are more important in trust computation when the direct observation is not available.

The request for recommendations is broadcast to the whole environment. Considering that a pervasive computing environment may have a large number of devices, requests for recommendations may not be sent to or received by all devices in the environment immediately. If a device always waits for recommendations from all other devices in the environment before computing the trust, it may fall into an infinite waiting time. To avoid this scenario to happen and to ensure the validity of the received recommendations, we set the minimum number of received recommendations as a percentage of the total number of current devices in the environment. When the number of received recommendations reaches this desired percentage of current devices in the environment, the device will stop waiting for recommendations and will indirectly compute the trust. In the sequel, we present an algorithm for trust computation.

3.3 An Algorithm for Trust Computation

The computation mechanism is activated when a new device enters the environment or when a request for communication with an unknown device in the environment is initiated. At this point in time, the device will begin the trust computation process. The pseudo-code of the algorithm is provided in Figure 2, and works as follows. Initially, device A checks local trust records to look for the trust value of the unknown device B. If the trust value is found, then no trust computation is necessary, and device A will use the available stored trust value of B to setup the communication with device B. Otherwise, a trust computation will be performed. Device A will first attempt to perform a direct trust computation through a direct observation. The direct observation usually seeks multiple types of trust information embedded in a device as we have described in Section 3.2. If the desired trust information can be obtained through the direct observation, then a direct trust computation will be performed. If the trust information obtained through the direct observation is inadequate or is not available, then device A will perform an indirect trust computation. In order to run the indirect trust computation, device A will first broadcast a request for recommendations throughout the environment. Then, a recommendation value will be calculated based on received recommendations. Finally, based on this recommendation value, an indirect trust computation will be performed.

```
BEGIN
        A checks T_A(B) in A's trust record;
        IF T_A(B) is found
                Trust computation ends
        ELSE
```

```
                    O_A(B) <- Direct observation of B
                    IF O_A(B) >= Thrd_DO
                           Direct Computation: T_A(B) = O_A(B)
                           Trust computation ends
                    ELSE
                           Broadcast request for recommendation
                           Gather recommendations
                           Compute recommendation value R_A(B)
                           IF R_A(B) = null
                                  T_A(B) = O_A(B)
                                  Trust computation ends
                           ELSE
                                  Indirect Computation T_A(B)
                                  Trust computation ends
                           END IF
                    END IF
             END IF
      END
```

Figure 2 Pseudo-code for trust computation using direct observation and recommendation values.

3.4 Updating the Trust Value

The trust is dynamic since the behaviours of devices are not static. In a pervasive computing environment, most devices are mobile. The network traffic status, the surrounding environment and many other external factors, can influence the behaviours of these devices. Some devices may behave well in a particular time period; however, they may behave poorly in another time period due to changes in circumstances in the environment where they operate. Hence, a fixed trust value is by no means feasible for reflecting the behaviour of a device.

Trust values should be updated after each interaction. When a trust value is updated, two factors should be taken into account: the device's current behaviour and the device's past trust record. Trust should not be evaluated only on a device's current behaviour. A trust updating method should consider both the current and past behaviours of a device.

Based on the above requirements, we have developed a trust value's updating method in our trust management scheme. During each trust update, device A checks device B's behaviour during the previous interaction, and updates the trust value in the following manner:

$$T_A(B) = \begin{cases} 1 & if\ T'_A(B) + C_A(B) \geq 1 \\ T'_A(B) + C_A(B) & \\ -1 & if\ T'_A(B) + C_A(B) \leq -1 \end{cases} \quad (3)$$

In Equation (3), $T_A(B)$ is the new updated trust value, $T'_A(B)$ is the current trust value, and $C_A(B)$ is the adjustment parameter based on the device B's behaviour. The range of $C_A(B)$ is from -1 to 1. In addition, $C_A(B)$ is a combination of two components, $C_A(B) = P_A(B) + N_A(B)$, where

$P_A(B) = WP_A(B) \cdot P$ is the summation of device B's positive actions toward device A, and $N_A(B) = WN_A(B) \cdot N$ is the summation of device B's negative actions toward device A.

The column vector of the positive aspects of device B's behaviour toward device A is denoted by *P*, and the column vector of the negative aspects of device B's behaviour toward device A is denoted by *N*. $WP_A(B)$ and $WN_A(B)$ are two vectors that are used to indicate the observation of negative or positive actions. $WP_A(B)$ and $WN_A(B)$ are obtained as

$$WP_A(B) = \begin{bmatrix} wp_A^1(B) & wp_A^2(B) & \ldots & wp_A^m(B) \end{bmatrix}$$
$$WN_A(B) = \begin{bmatrix} wn_A^1(B) & wn_A^2(B) & \ldots & wn_A^n(B) \end{bmatrix} \quad (4)$$

3.4.1 Positive Action P_i: This is the positive action of device B toward device A, and $0 < P_i < 1$. For $i = 1$ to m, each P_i corresponds to a positive action, and each P_i has a different value according to the corresponding action's importance. The positive actions can be defined according to the actual application environment, such as device B relaying traffic for device A with a high level of throughput and low packet loss ratio, or device B providing the requested service to device A without delay.

- ***Observed Positive Actions*** $wp_A^i(B)$: If a positive action corresponding to P_i is observed, $wp_A^i(B) = 1$. Otherwise, $wp_A^i(B) = 0$.

3.4.2 Negative Action N_j: This is device B's negative action toward device A, and $-1 < N_j < 0$. For $j = 1$ to n, each N_j corresponds to a negative action, and each N_j has a different value according to the corresponding action's importance. The negative actions can also be defined according to the actual application environment, such as device B refusing to provide the service requested by device A, or device B presenting a high packet loss ratio when employed as an intermediate node for device A.

- ***Observed Negative Actions*** $wn_A^j(B)$: If a negative action corresponding to N_j is observed, $wn_A^j(B) = 1$. Otherwise, $wn_A^j(B) = 0$.

Therefore, the adjustment parameter $C_A(B)$ is a summation of observations of both device B's positive and negative actions toward the device A.

3.5 Purging Trust Values

A device may exist in a certain environment for only a short period of time, and then leave it and not return. In this case, if other devices in the environment retain its trust value, this action will result in wasted storage resources and will decrease those devices' efficiency. In addition, outdated trust records may provide false trust information. When device A finds that device B has stopped communicating with him, device A will not update the device B's trust value until device B tries to communicate with him again. After a predefined period, if device A finds that device B's trust value was not updated during this period of time, then device A will delete device B's trust value. Later on, if device B

attempts to establish communication with device A, then device A will treat device B as an unknown device and will compute device B's trust value once again.

3.6 Trust Relationship with a New Device

In a pervasive computing environment, the population is dynamic. New devices constantly enter the environment, and it is necessary to investigate how a new device can build trust relationship with other devices. A new device is unknown to all other existing devices in the environment, so no recommendations are available when an existing device wants to run a trust computation about this new device. In this situation, this new device is given a trust value of 0, as we point out in Section 3.2. The interactions with this new device can still happen because we defined a device as untrustworthy when its trust value is negative.

After the interaction with the new device, an existing device can use the trust value's update mechanism to adjust the trust value on this new device. During the interaction, if it happened that the new device behaved well, its trust value will increase, otherwise its trust value will decrease. Later on, when other existing devices run trust computations about this new device, they might receive recommendations from the devices that have interacted with the new device and use these information to perform an indirect trust computation involving this new device. This way, the devices in the environment can receive a trust information about the new device.

In a decentralized and distributed system like pervasive computing, there is usually no central management mechanism. The trust between devices is accumulated through actual interactions. Many trust management schemes have been proposed for pervasive computing and related fields ([3] and the references therein). But, none of them can allow a device to obtain trust information about other devices without actual interactions. It is true that a device can calculate the trust information through the use of recommendations. However, the trust information contained in recommendations is actually obtained by recommendation providers through interactions.

4 Performance Evaluation

In order to evaluate the performance of the proposed trust management scheme, we conducted a series of simulation experiments on an in-home pervasive healthcare environment. There are three advantages to using simulations to evaluate the performance of a system. First, a simulation environment has the flexibility to allow us to set parameters to different values and observe the results. In a real environment, a phenomena or a consequence might be caused by many factors. However, in a simulation environment, we can avoid the influence of unwanted factors and perform a more effective analysis on the outcome. Second, it is easier to collect the desired data in a simulation environment than in a real environment. In the real environment, data need to be collected by means of special equipment and techniques. In a simulation environment, all kinds of data are produced and transferred in a pure software environment, so we can setup the proper functions to collect the desired data. Third, it is convenient to investigate the approach in a simulation environment. In a real environment, all hardware and software should be complete, and it can take a lot of time to configure each single unit to make them all work together. Some hardware or software might not meet the requirements of the proposed approach. On the contrary, it is easier to build our simulation environment in a virtual space. All necessary software and hardware exist virtually, and they work just as real software and hardware do through appropriate configurations.

The simulation experiments are implemented in Java. Several scenarios are used in the experiments, and multiple types of data are collected and compared to illustrate the effects of the proposed trust management scheme's main components.

4.1 Performance Metrics

We have used three performance metrics: the average throughput, the average packet loss ratio, and the recommendation overhead. Each performance metric is investigated by varying the network size, the traffic load, and the length of the simulation. The three performance metrics are described as follows.

- ***Average Throughput***: We define the throughput as the number of packets that an intermediate node has successfully delivered during a predefined period of time. We evaluate our proposed trust management scheme's performance by comparing the average throughput values in the simulations. The average throughput is defined by the following formula:
-

$$Average\,Throughput = \frac{Pack_{suc}}{Time_{total}} \tag{5}$$

  where $Pack_{suc}$ is the total number of packets successfully transmitted and $Time_{total}$ is the total interaction time.

- ***Average Packet Loss Ratio***: The average packet loss ratio is the ratio of packets that were lost during the transmission to the total packets generated in a certain period of time. The average packet loss ratio is defined by the following formula:

$$Average\,Packet\,Loss\,Ratio = \frac{Pack_{loss}}{Pack_{total}} \tag{6}$$

  where $Packe_{total}$ is the total number of packets generated by devices and $Packe_{loss}$is the number of packets that were lost during the transmission.

- ***Recommendation Overhead***: This is the overhead caused when dealing with received recommendations, the quantity of which is less than the desired number required for indirectly computing the trust. The recommendation overhead is defined by the following formula:

$$Recommendation\,Overhead = \frac{Recomm_{unused}}{Size_{network}} \tag{7}$$

  where $Recomm_{unused}$ is the total number of received recommendations, and these recommendations are not sufficient enough to run indirect trust computations. $Size_{network}$ is the network size (number of devices) of the environment.

4.2 Simulation Parameters

We use three parameters: the network size, the traffic load, and the length of the simulation. We change the values of these parameters in the experiments and observe the performance metrics. The three parameters are described as follows.

- ***Length of the simulation***: This is the duration of the simulation. In the first experiment, we need to investigate the performance of the proposed scheme over varying lengths of time periods. With trust management, a device chooses appropriate devices to communicate with, based on certain trust values. In the simulation, we change the length of the simulation to see how the performance metrics are influenced when applying different trust computations.

- ***Network size***: This is the total number of devices in the environment. We set different values of network size to investigate how performance metrics are influenced by different percentages of recommendations.

- ***Traffic load***: This is the number of packets that a device generates each time. We change this parameter to investigate whether a device can update its trust record according to the actual communication result, and hence, choose appropriate devices to communicate with to get low average packet loss ratios.

4.3 Simulation Experiments

We have designed three experiments to evaluate our proposed model's performance in a pervasive in-home healthcare environment.

4.3.1. The Effects of Simulation Time

In this experiment, we designed three scenarios in order to investigate whether the devices could use our proposed trust management scheme to find the proper device to be used as an intermediate node. This pervasive computing environment has 50 existing devices and 10 new devices. We assume that the traffic load generated by a new device each time is from 1 to 1000 packets, and new devices choose an existing device as an intermediate node to transmit these packets. The simulation experiments are conducted in three scenarios as follows.

- ***Experiment without Trust Management***: In this scenario, new devices use no trust management scheme. New devices have no prior knowledge about the behaviour or performance of other devices in the environment. Since no trust management scheme is applied, new devices do not have any trust values to use when choosing another device to communicate with. When new devices need to transmit packets, they randomly choose a device in the environment as an intermediate node.

- ***Experiment without Recommendations*:** In this scenario, new devices use the trust management method when choosing a node. Initially, in new devices' trust records, all 50 devices' trust values are 0 because new devices have no trust information about each others. After every interaction, any new device will update its trust record based on the throughput value obtained from its previous interaction. In Section 3.3, we introduced a method for updating the trust value. In every interaction, the new device interacts only with devices whose trust values in the new device's trust record is non-negative

- ***Experiment with Recommendations***: In this scenario, we try to solve the problem of choosing the first device to interact with by using the trust management scheme with recommendations. When the new device attempts to interact with an existing device, it requires recommendations from other devices. New devices run the trust computation that we introduced in Section 3.2. After each interaction, new devices update their trust records. New devices' trust records may include multiple devices with non-negative trust values. When a new device chooses a device from its

trusted list to interact with, it picks a device with a non-negative trust value. However, this does not mean that the device with the highest trust value is always chosen, but the higher the trust value a device has, the higher the probability that it will be chosen. When the chosen device is busy interacting with another device, new devices need to choose another device with the next highest non-negative trust value.

Figure 3 depicts the average throughput of the above three scenarios as a function of simulation time. The average throughput without trust management (*no-trust*) remains constant over time, which means that new devices cannot find devices with acceptable throughput. With increasing time, the average throughputs with trust management (*without-recomm* and *trust-recomm*) become stable and are much higher than the average throughput without trust management. This means that the trust management method helps the new device to choose devices with acceptable throughput, and hence gets a good average throughput.

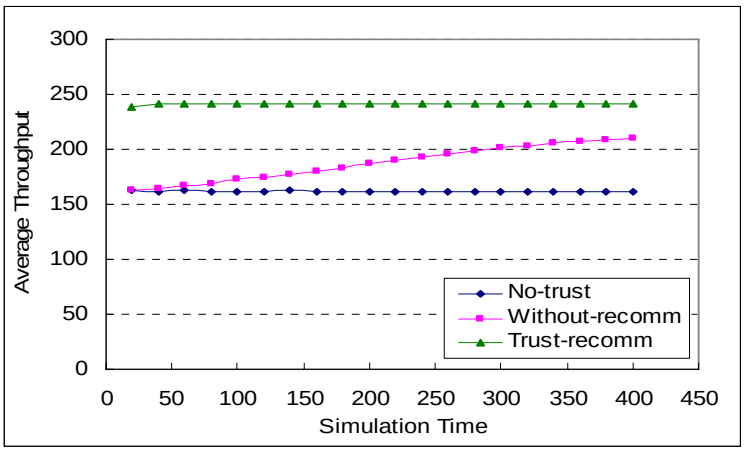

Figure 3 Throughput as a function of simulation time

The initial average throughput with recommendations is higher than the average throughput without recommendations. However, with increasing time, the two average throughputs merge because without recommendations, a device needs to update its trust record from actual interactions. Therefore, new devices may interact at the beginning of the simulation with devices that have low performance, and hence receive a low throughput.

With recommendations, however, new devices can predict a device's performance before the first interaction, and then decide whether or not to interact. In other words, with the help of recommendations, devices with high throughput can be found faster than they can be found without recommendations. In this way, new devices avoid interacting with devices that have low performance, hence, they gain higher throughput. With increasing time, new devices build proper trust records on existing devices under trust management, with or without recommendations. Therefore, the throughputs with or without recommendations tend to be similar to each other.

4.3.2 The Effects of Traffic Load

We used this experiment to show the importance of the trust value's updating mechanism in the proposed trust management scheme. In a pervasive computing environment, devices perform differently from each other when used as intermediate nodes. Among all parameters that are used to evaluate a device's performance, the packet loss ratio is an important one. When the transmitted traffic load is light, devices may provide acceptable packet loss ratios. However, when the traffic load increases, the packet loss ratio also increases. As a result, the packet loss ratio of some devices with low performance may reach an unacceptable high level when the traffic load increases to a certain amount. The trust value updating mechanism is designed to ensure that the trust value accurately reflects the device's performance. The trust management scheme needs to preserve communication quality, no matter how heavy the traffic load is.

In this experiment, we setup 10 new devices and 50 existing devices in a pervasive computing environment. Through trust computations, the new devices are initialized with the trust values of all existing devices. All of these existing trust values were non-negative, which means that any new devices treated all of them as trustworthy when attempting to interact with one of them. The 50 existing devices provided different packet loss ratios when acting as intermediate nodes. We set the maximum acceptable packet loss ratio as 0.2. A device was considered trustworthy when its packet loss ratio was not higher than this value during the interaction; otherwise, it was not considered trustworthy. The devices' packet loss ratios increased with increasing traffic load. Some of their packet loss ratios climbed higher than the maximum acceptable level when the traffic load increased. Others could maintain packet loss ratios below the maximum acceptable level. In this experiment, we increase the number of packets (i.e., traffic loads) generated from new devices, in order to study whether the trust value's updating mechanism could help new devices to find a device with good performance to interact with when the traffic load varied.

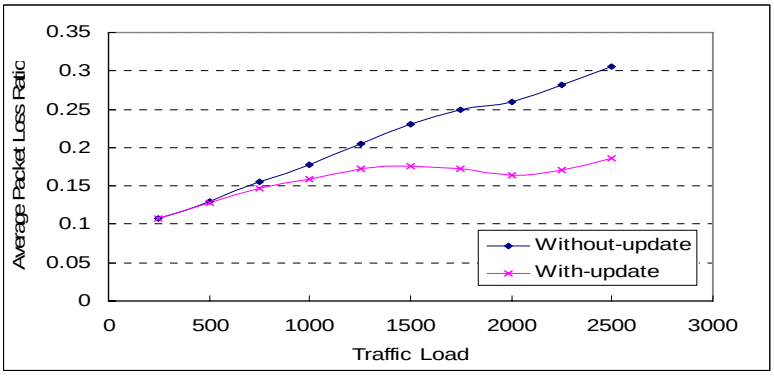

Figure 4 Packet loss tatio as a function of traffic load

Figure 4 shows the average packet loss ratio as a function of traffic load. It illustrates that the average packet loss ratios in both scenarios tend to increase with an increase in the traffic load. However, the average packet loss ratio without trust value update exceeds the threshold value of 0.2 when the traffic load rises to 1200 packets. The average packet loss ratio of the scenario that updates trust values stays under the threshold, although it also increases.

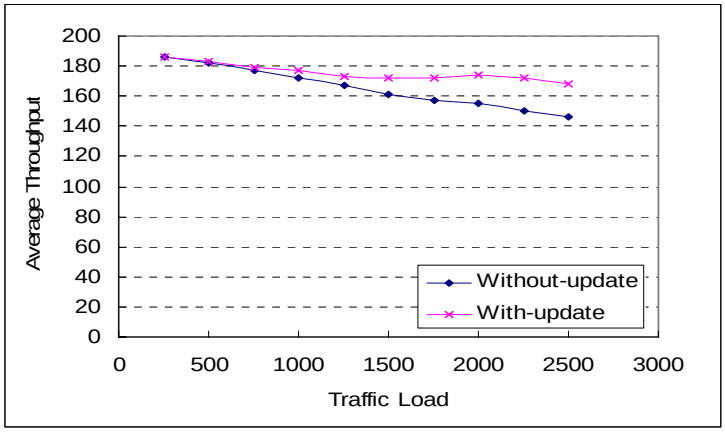

Figure 5 Throughput as a function of traffic load

Figure 5 depicts the average throughput as a function of traffic load. The average throughputs of both scenarios tend to decrease with an increase in the traffic load. However, when the traffic load is higher than 1000, the average throughput without the trust value update mechanism is lower than that obtained using the trust value update mechanism.

Without the trust value updating mechanism, a device checks the trust values in its trust record only when it attempts to interact with another device. After each interaction, the device does not update the trust value according to the device's performance change during the interaction. As a result, devices still interact with a device even when that device's packet loss ratio is above the threshold due to the increased traffic load. Therefore, devices suffer high packet loss ratios when the traffic load is high.

With the trust value updating mechanism, devices update the trust values according to a device's performance change during the interaction. Therefore, when the traffic load increases, devices can find an appropriate device to interact with by checking their updated trust records. Consequently, the trust value updating mechanism helps to keep the average packet loss ratio at an acceptable level when the traffic load increases.

4.3.3 The Effects of Network Size

In this experiment, we investigated how to set the proper percentage of the number of received recommendations and the network size for indirect trust computation. In order to carry out this experiment, we varied the number of existing devices from 5 to 50, and the simulation time was set at 30 seconds. We assumed that the traffic load from a device was from 1 to 1000 packets. An interval of 0 to 1 second separates two communications. If a device always waits for recommendations from all other devices in the environment, the danger of infinite wait time arises. To avoid this, it is necessary to set a minimum percentage, as we have presented in Section 3.2. After a device broadcasts a request for recommendations, it waits until the desired percentage of recommendations has been received, and then uses these recommendations to indirectly compute the trust. Otherwise, it treats recommendations as 0. In the scenario of trust management using recommendations, we varied the percentage values from 20% to 50% to illustrate the scheme's performance with different numbers of devices. We also investigated the recommendation overhead under different percentage values.

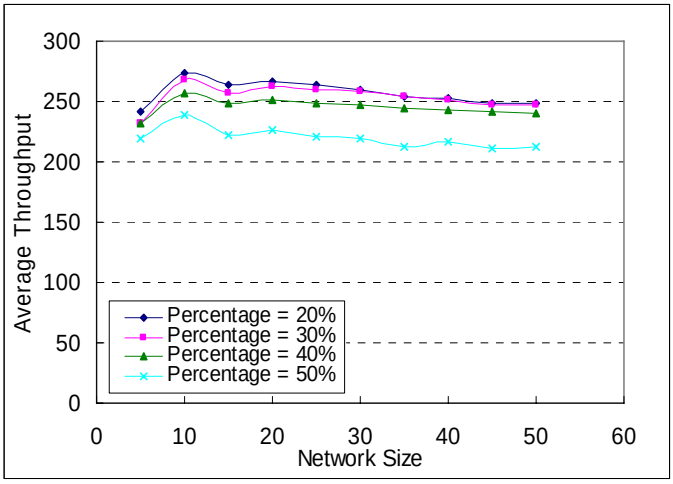

Figure 6 Throughput as a function of network size

Figure 6 illustrates the average throughput as a function of network size, and Figure 7 shows the recommendation overhead as a function of the network size. Figure 6 demonstrates that when the percentage value is 20% or 30% in all environments with network size from 5 to 50, devices can receive a good level of throughput during interactions. Also, graphs with 20% and 30% in Figure 7 show low levels of overhead. This shows that with 20% or 30%, devices can receive the desired number of recommendations needed to indirectly compute trust. In this situation, devices may choose the right device as an intermediate node with the help of properly created trust values.

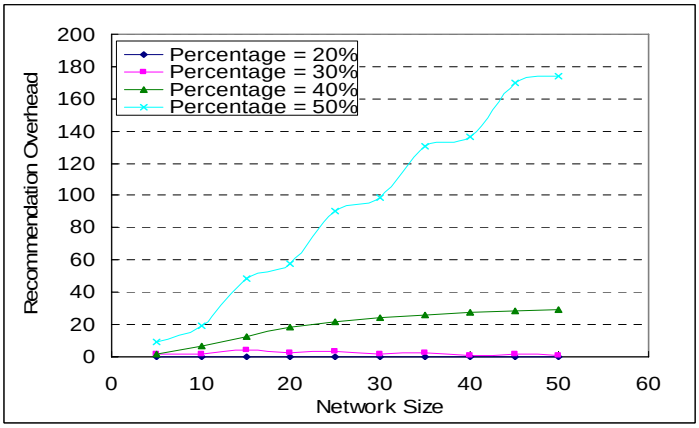

Figure 7 Recommendation overhead as a function of network size

However, when the percentage value exceeds 40%, we observe a decrease in throughput during interactions. Figure 7 shows that the overhead increases with a network size of 10 or more. When the percentage is at 40% or 50%, devices often fail to receive enough recommendations to indirectly compute trust; hence, devices have to manage trust without recommendations, and update their trust

records based on their experience with actual interactions. Devices may interact with devices that have low performance before the proper trust values are computed and updated. This results to a low average throughput for the length of the simulation. Therefore, the experimental results show that 20% and 30% are good percentage values. By considering both the recommendation overhead and the throughput, the experimental results show that receiving recommendations from more than 30% of the devices do not result in performance gains when indirectly computing trust values.

5 Conclusions and Future Work

In this paper, we have proposed a deterministic trust management scheme for pervasive computing. The proposed scheme has two main features. First, it is distributed. The distributed trust management makes the devices independently handle the trust issues under the absence of a central management. Second, the proposed scheme involves two methods for trust computation, direct computation and indirect computation. Recommendations are also used to assist when adequate first hand information is not available. The proposed scheme was evaluated using simulation experiments to investigate its performance in terms of throughput and packet loss at various parameter settings. Future research works will focus on probabilistic trust management using Markov and other mathematical models.

References

1. Weiser, M, The Computer for the 21st Century, Scientific American, vol. 265, 1991, pp. 66-75.
2. Kagal, L., Finin, T. and Joshi, A., Moving from Security to Distributed Trust in Ubiquitous Computing Environments. In LNCS 2867, 2003.
3. Sun, Y. L., Han, Z., Yu, W. and Liu, K. J. R., A Trust Evaluation Framework in Distributed Networks: Vulnerability Analysis and Defence Against Attacks. In 25th IEEE International Conference on Computer Communications (INFOCOM 2006), 2006, pp. 1-13.
4. Nguyen, C. T., Camp, O., and Loiseau, S., A Bayesian network based trust model for improving collaboration in mobile ad hoc networks. In 2007 IEEE International Conference on Research, Innovation and Vision for the Future, 2007, pp. 144-151.
5. McNamara, L., Mascolo, C. and Capra, L., Trust and Mobility Aware Service Provision for Pervasive Computing. In First International Workshop on Requirements and Solutions for Pervasive Software Infrastructures (co-located with Pervasive 2006), Dublin, Ireland, May 2006.
6. Jøsang, A., and Ismail, R., The Beta Reputation System. In Proceedings of the 15th Bled Conference on Electronic Commerce, June 2002, pp. 17-19.
7. Langheinrich, M., When Trust Does Not Compute - The Role of Trust in Ubiquitous Computing. In the Proceedings of Privacy Workshop in Ubicomp'03, 2003.
8. Ranganathan, K., Trustworthy Pervasive Computing: The Hard Security Problem. In Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops, 14-17, March 2004, pp: 117 - 121.
9. Goecks, J., and Mynatt, E., Enabling Privacy Management in Ubiquitous Computing Environments Through Trust and Reputation Systems. In Workshop on Privacy in Digital Environments: Empowering Users. In the Proceedings of CSCW 2002, 2002.
10. Kagal, L., Finin, T., and Joshi, A., Trust-based Security in Pervasive Computing Environments. In Computer, 2001, vol. 34, issue 12, pp. 154-157.
11. Nielsen, M., and Krukow, K., A Bayesian Model for Event-based Trust. In Electronic Notes in Theoretical Computer Science (ENTCS), 2007, vol. 172, pp. 499-521.
12. Quercia, D., Hailes, S., and Capra, L., B-Trust: Bayesian Trust Framework for Pervasive Computing". In Lecture Notes in Computer Science, 2006, vol. 3986, pp. 298- 312.

13. Yuan, W., Guan, D., Lee, S., and Lee, Y.-K. A Dynamic Trust Model Based on Naive Bayes Classifier for Ubiquitous Environments. In the 2006 International Conference on High Performance Computing and Communications (HPCC-06), Munich, Germany, Sept. 13-15, 2006.
14. Yuan, W., Guan, D., Lee, S., and Lee, Y. K., The Role of Trust in Ubiquitous Healthcare. In the Proceedings of the 9th International Conference on e-Health Networking, Application and Services, 2007, 312-315.
15. Dong, C., and Dulay, N., Privacy Preserving Trust Negotiation for Pervasive Healthcare. In the 1st International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth 2006), 2006, 1-9.
16. Sun, T., and Denko, M. K., A Distributed Trust Management Scheme in Pervasive Computing Environment. In the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2007), 2007, 1219-1222.
17. Gambetta, D., Can We Trust? In Trust: Making and Breaking Cooperative Relations, 1988, pp. 213-237.
18. Jøsang, A., The Right Type of Trust for Distributed Systems. In the Proceedings of the 1996 Workshop on new security paradigms, NSPW '96, ACM Press, 1996, pp. 119-131.
19. Varshney, U. Pervasive Healthcare, Computer, vol. 36, Issue 12, 2003, pp. 138-140.