Reliable Delivery of Event Data from Sensors to Actuators in Pervasive Computing Environments*

Sudip Chakraborty, Nayot Poolsappasit, and Indrajit Ray

Computer Science Department
Colorado State University
Fort Collins, CO 80523, USA
{sudip, nayot, indrajit}@cs.colostate.edu

Abstract. The event-condition-action (ECA) paradigm holds enormous potential in pervasive computing environments. However, the problem of reliable delivery of event data, generated by low capability sensor devices, to more capable processing points and vice versa, needs to be addressed for the success of the ECA paradigm in this environment. The problem becomes interesting because strong cryptographic techniques for achieving integrity impose unacceptable overhead in many pervasive computing environments. We address this problem by sending the data over the path from the sensor node to the processing point that provides the best opportunity of reliable delivery among competing paths. This allows using much weaker cryptographic techniques for achieving security. The problem is modeled as a problem of determining the most reliable path – similar to routing problems in networks. We propose a trust-based metric for measuring reliability of paths. The higher the trust value of a path the more reliable it is considered. We propose techniques for estimating the trust levels of paths and propose a new algorithm for identifying the desired path.

1 Introduction

Pervasive computing technology has the potential to impact numerous applications that benefit society. Examples of such applications are emergency response, automated monitoring of health data for assisted living, environmental disaster mitigation and supply chain management. Pervasive computing uses numerous, casually accessible, often invisible, computing and sensor devices. These devices are frequently mobile and/or embedded in an environment that is mobile. Most of the time they are inter-connected with each other, with wireless or wired technology. Being embedded in the environment and interconnected allow pervasive computing devices to exploit knowledge about the operating environment in a net-centric manner. This enables pervasive computing applications to provide a rich new set of services and functionalities that are not otherwise possible through conventional means. Pervasive computing applications frequently rely

* This work was partially supported by the U.S. Air Force Research Laboratory (AFRL) and the Federal Aviation Administration (FAA) under contract F30602-03-1-0101 and by the U.S. Air Force Office of Scientific Research under contract FA9550-07-1-0042. Any opinions, findings, and conclusions expressed in this publication are solely those of authors and do not necessarily represent those of the AFRL, the FAA, or the AFOSR.

S. Barker, G.-J. Ahn (Eds.): Data and Applications Security 2007, LNCS 4602, pp. 77–92, 2007.

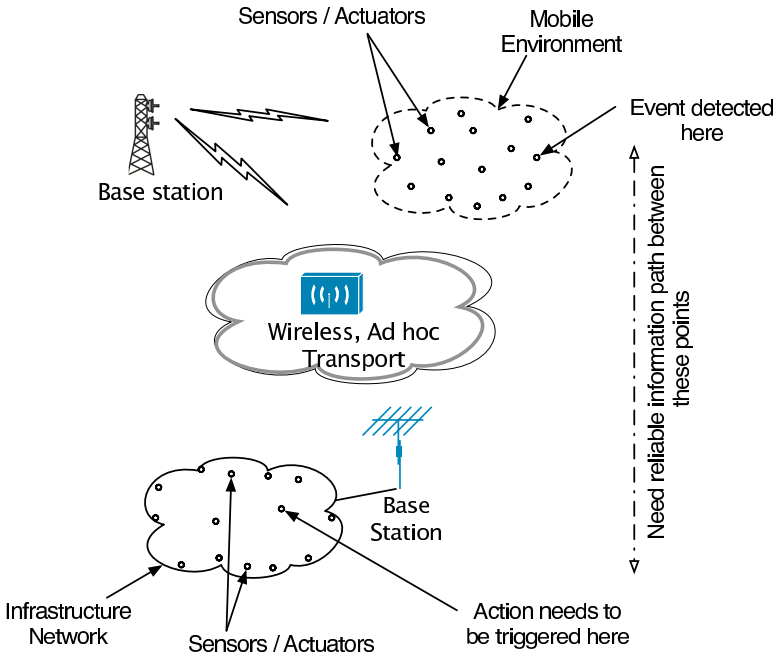


Fig. 1. Pervasive computing environment involving remote event detection and action triggering

on event-triggered obligation policies to operate in a dynamic environment. An obligation policy is associated with events, conditions, subjects, objects and actions. When the event of interest occurs and the associated conditions evaluate to true, the subjects perform the specified actions on the objects. Events are typically identified and captured by embedded sensing devices and actions are actuated by similar embedded devices. Processing of captured event data for evaluation of conditions are, on the other hand, mostly performed at remote processing nodes or base stations. This is because the sensing and actuating devices embedded in the environment are frequently of very low performance capabilities including low computing, low storage and low power. Thus a major challenge in a pervasive computing environment is to provide a path for propagating sensor data to processing nodes and action data to actuating nodes. Reliability of the paths is important. The data should be delivered with the minimum possible error and in as timely a manner as possible.

The reliable transmission path requirement imposes significant challenges in pervasive computing environments. A pervasive computing application can seldom assume a reliable network infrastructure for communication. In a conventional setting, a node that generates a message forwards it to a neighboring reliable node. The receiving node in turn forwards the message to another fixed node that is known a priori. This procedure is followed till the message reaches the destination. Every node in this process knows at least one other reliable node in the path towards the destination to which the message

can be handed over. Frequently a node will know about more than one other node and thus have a choice of a better node. The nodes are static, that is they do not change their location and consequently the links between the nodes are stable. This and the proper use of strong cryptographic techniques, easily facilitate reliable delivery of messages in conventional settings. In a pervasive computing environments, on the other hand, mobility of nodes (sensing, processing or actuating) is frequently considered an asset. Figure 1 depicts the scope of the problem. Nodes are not locationally stable; instead they continuously change their coordinates. Thus, a node that needs a message delivered cannot rely on another fixed node to forward the message but has to make use of one or more nodes that happen to be within reachable distance at that particular moment. In addition, since a majority of these nodes are low capability devices (in the sense of low computational capabilities, low storage and low power provisions), use of strong cryptographic techniques needs to be ruled out. Moreover, in hostile environments these nodes get easily compromised. Under such circumstances it will enormously benefit a pervasive computing application if the path that provides most opportunity of reliable delivery of messages is presented to it. Determining an appropriate path within a network is the problem of routing. In this paper we revisit this problem in the context of pervasive computing environments.

The problem of routing in mobile ad hoc networks have been addressed before [1,2,3,4,5,6,7,8,9,10,11]. Among these [1,6,7,10] study cryptographic techniques for securing the routing protocol. Some use public key cryptography to encrypt the end-to-end transmission of routing messages. Others use digital signature techniques to authenticate routing messages at the peer-to-peer level connection. However, these cryptographic techniques incur high computation and storage overhead which limit their use in sensor devices. Use of secret key techniques instead of public keys alleviate this problem to some extent although at the expense of added complexity. Moreover, key distribution and management is a big problem in secret key based systems. It is difficult to establish a key distribution or certification authority in mobile ad hoc environments. Ensuring the availability of a key distribution center or a certifying authority is almost impossible given the unstable nature of the network.

The Hermes protocol developed by Zouridaki et al. [11] proposes using trustworthiness of its neighbors for routing. The trust values are computed under the assumption that they follow the beta probability distribution. The parameters of the beta distribution come from the empirical observation of the forwarding behavior. Thus, nodes that maintain a good and steady forwarding history have more trust and confidence on them. The route is established for the most trusted path. However, the major problem of this work is its complete reliance on forwarding history for measuring trust. A malicious node can easily fake this history thus presenting itself as a trusted node. Other similar works include [12,13,14]. Among these [13] proposes a signal stability-based adaptive routing (SSR) where the routes are selected based on signal strength. This work looks promising; however it does not discuss how to measure this signal strength quantitatively. In [12] the authors propose an on demand secure routing protocol where the metric is based on past history. Yi et al. [14] present a security-aware ad hoc routing protocol (SAR) in which a route is selected on the basis of degree of ‘security guarantee’ that the route provides. If two routes have same guarantee then the shorter path is chosen.

The security metric can be specified by standard security properties like timeliness, ordering, authenticity etc. However, the paper does not discuss how we can measure these properties quantitatively.

We propose a trust-based routing protocol for pervasive computing environments. Our protocol determines the most reliable path under currently determinable properties of the system to forward a packet from source to destination. A node in the pervasive computing environment is any entity that is able to forward a packet. It can be a sensor node, a mobile device like a PDA or a cellular phone, a powerful computing device or even an actuator like a switch. Reliability of a node is measured in terms of a *trust* value for the node. Each node determines its neighbor's trust based on physical properties of the neighbor that can be directly observed, the neighboring node's behavior history (i.e., results of past interactions) and recommendation (or rating) about the neighbor from other neighbors. The resulting trust value is used to generate the 'cost of forwarding', or simply *cost*. The cost metric is inversely related to the trust metric, that is, higher the trust (reliability) on the node, lower is the cost. This cost is associated with links in the network. We next modify the widely used distance vector routing protocol using these costs between the links to find the path with minimum average cost. The chosen path then becomes the most 'reliable' path.

The rest of the paper is organized as follows. Section 2 gives an overview of our protocol including a discussion on cost function in subsection 2.1. We introduce our trust metric in section 3 where the components and the methods to compute them have been discussed in subsections 3.1, 3.2, and 3.3 followed by the computation of the final trust value in subsection 3.4. Subsequent section 4 presents our trust-based routing protocol. In section 5 we analyze our protocol. We present the security analysis of the protocol in subsection 5.1. In subsection 5.2 we discuss the complexity of our protocol. We start with computation complexity followed by communication complexity and storage complexity respectively. Finally, we conclude our discussion in section 6 with a summary for future work.

2 Overview of Trust-Based Routing Protocol

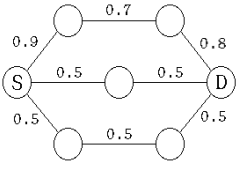
We assume that the pervasive computing environment supporting the application has a very dynamic topology. Nodes join or depart the environment at random. Each node in the network maintains a table RT consisting of tuples of the form $\langle Dest, Win\#, NH, Cost_{avg}, \sum Cost^2, \#Hops \rangle$. In this table the node stores on a per-destination ($Dest$) basis, the identity of the next neighbor (NH), to which the message needs to be forwarded. Together with the next neighbor information, the node also stores the minimum average cost ($Cost_{avg}$) and the number of hops ($\#Hops$) to reach the particular destination. This information is generated periodically. Thus each tuple bears a time stamp in the form of a current time window ($Win\#$). The routing algorithm that we propose is used to update the next neighbor entry in the RT table for a particular destination.

A source node initiates the routing protocol if it has a packet to be sent to a destination for which it does not have any next hop (NH) entry. The source node can also execute the routing protocol when the path to the destination has expired (when the value under $Win\#$ is less than the current window number). The source sends out a route

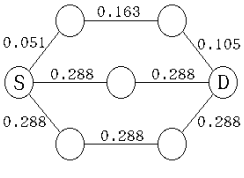
discovery request. We assume that each node that participates in the pervasive computing environment has a *trust relationship* with its neighbor (that is a node at 1 hop distance). A trust-aware node periodically sends a *beacon* message to its neighbors. The beacon message is something like an "I am alive" message and carries information necessary to prove the node's existence. Once in a while a node can also send out a beacon message on demand. In our protocol, a node may request a *recommendation* score from a second node about a neighbor of the second node. In such cases the recommendation score is carried on a beacon message. Beacon messages are broadcast in nature. They carry rudimentary checksums to provide weak protection against integrity violations. The recommendation score is used as one of the parameters for computing *trustworthiness* of a neighbor. Trust relationships are periodically refreshed locally. At some point after system initiation we assume that every node in the system will have a trust relationship with each of its neighboring nodes. We do not assume that trust relationships are symmetric or transitive.

We adapt the trust model proposed in [15]. We express the trust relationships between nodes as $N_r \longrightarrow N_e$ where N_r is the *truster* node and N_e is the *trustee* node. We represent this trust relationship as a tuple $({}_{N_r}P_{N_e},\ {}_{N_r}R_{N_e},\ {}_{N_r}I_{N_e})$. The value ${}_{N_r}P_{N_e}$ represents an evaluation of the *physical properties* of N_e by N_r. The value ${}_{N_r}R_{N_e}$ denotes an evaluation of the *recommendation scores* of N_e from other nodes and ${}_{N_r}I_{N_e}$ evaluates the *interactions* that N_r had with N_e. The exact interpretation of these terms are deferred for the time being till the next section (section 3). We associate a numeric value $v(N_r \longrightarrow N_e)$ (from $[-1,1]$) with the above tuple which we refer to as the trust value for node N_r on node N_e along the edge $(N_r,\ N_e)$. We next convert this trust value to a cost on the link $(N_r,\ N_e)$. The higher the trust value the lesser is the cost to transfer messages on the link. The path having the least average cost from the source node to the destination node is considered the most reliable among the available paths and is chosen by the source node to forward the data.

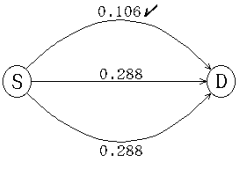
Figure 2(a) describes pictorially the main idea of our protocol. We assume that if a node N_r has a distrust value (that is value less than 0) on another node N_e the cost on that link is infinite and that next hop is discarded. When a node receives a route discovery request from a source, it checks its routing table RT. If a route to the destination is present in RT which has not expired, it sends the '$\#Hops$' and cost related information



(a) The trust relations in trust-aware pervasive computing environment



(b) The forwarding cost in trust-aware pervasive computing environment



(c) Path having least average cost is most reliable

Fig. 2. Trust relation between nodes and the corresponding cost on the link

to the source. The source then evaluates the cost of the link between the neighbor and itself and using the $\#Hops$ it computes the average cost of forwarding the packet to the destination. The source may get multiple such responses. It then chooses the next hop for which the average cost over the path is minimum. If the node that receives a route discovery request from the source, does not itself have the next hop information in its RT for that destination, it initiates a route discovery process as a source. This process can go on till the node which is 1 hop behind the destination initiates a route discovery request.

2.1 Cost Function

Each node tries to find the path to a given destination which has the minimum average forwarding cost. The *cost* of forwarding a packet from the node N_r to N_e is a function of $v(N_r \longrightarrow N_e)$. These two are related as follows: higher the trust, lesser is the cost and the cost increases as the distrust increases (i.e., the trust decreases). Rationale is, the cost (in terms of integrity violation and other malicious activities) of forwarding a packet through a more trustworthy node is less than that through a less trustworthy node. The cost is minimum (not zero though) when N_r has absolute trust ($v(N_r \longrightarrow N_e) = 1$) on N_e. This minimum cost (Min_{cost}) is a small positive cost incurred due to forwarding overhead. It is uniform over the whole pervasive computing environment and set at the bootstrapping of the system. We assume that the decay in cost with increased trustworthiness is logarithmic with the following conditions: at $v(N_r \longrightarrow N_e) = 1$, cost $= Min_{cost}$ and at $v(N_r \longrightarrow N_e) = -1$, cost $= \infty$. The function is defined as,

$$cost(N_r,\ N_e) = Min_{cost} - \ln(\frac{1 + v(N_r \longrightarrow N_e)}{2}) \tag{1}$$

The maximum allowable cost for N_r is incurred when $v(N_r \longrightarrow N_e) = 0$. This corresponds to the situation when N_r is *neutral* about trustworthiness of N_e. This cost, denoted by $MaxAllowed_{cost(N_r,\ N_e)}$, is $Min_{cost} - \ln(\frac{1}{2})$, that is, $MaxAllowed_{cost(N_r,\ N_e)} = Min_{cost} + 0.69$.

Computing this cost value has some overhead but is only linear in the number of nodes in the pervasive computing environment. The cost value is stored in the mobile device for a predetermined time or until a new beacon message has arrived. The absence of a beacon message from a particular node in a particular window of time represents a broken link during that time period. It can happen for various reasons including that the node is compromised. The node in such a case may either discard the broken link from the list of current neighbors or mark it as unused. Routing information is advertised by broadcasting the route setup packets periodically or on demand depending on the protocol used. These packets indicate which mobile nodes are accessible from which others and the average cost associated with the path towards a destination. When a node receives a data packet, it chooses the path which has the lowest average forwarding cost and forwards the packet to the neighbor on this path to be further forwarded towards the destination. During this process, a node also evaluates the packet forwarding performance of the neighbor node. By measuring this the node essentially evaluates an interaction score for the neighbor. The details of this process and other routing processes is explained in section 4 and section 5.2.

3 Trust Metric

As mentioned earlier in section 2 the trust of N_r on N_e depends on three factors – N_e's *properties*, *recommendation* about N_e (alternatively, N_e's rating) from another node N_k, and N_r's *interaction* with N_e. We assume that each of these three factors is expressed in terms of a numeric value in the range $[-1,1]$. A negative value for the component is used to indicate the *trust-negative* type for the component, whereas a positive value for the component is used to indicate the *trust-positive* type of the component. A 0 (zero) value for the component indicates *trust-neutral*. The final trust value, denoted by $v(N_r \longrightarrow N_e)$, is calculated as an average of these component values. Eventually $v(N_r \longrightarrow N_e)$ falls in the range $[-1,1]$. A trustee node is completely trusted (or distrusted) if the value of the trust relationship is 1 (-1). If the value is in the range $(0,1)$ the node is *semi-trustworthy*; if the value is in the range $(-1,0)$ the node is *semi-untrustworthy*. The 0 value represents trust neutrality, that is the trustee is equally trustworthy as untrustworthy.

3.1 Computing *properties*

In our approach trust is used as a reliability metric of a neighbor node for proper handling and forwarding the packet to the destination. A node N_i is a neighbor of node N_j if N_i is within the range of a beacon message from N_j. A node becomes more reliable when it has relatively more resources (in terms of signal strength, signal stability, less propensity to corrupt data etc.). Higher values of these attributes show that it is more capable of handling and forwarding a packet in a reliable manner. This motivates us to measure the node properties quantitatively and include that measure as a factor to evaluate trustworthiness of a node. We focus on two properties of a node – signal strength, and stability factor. A node maintains a property table, $PT = \langle Node\_id, SS_{avg}, SF \rangle$ for each neighbor node where the properties of the neighbor is kept along with the corresponding id. The table is updated after each time-window *win*.

Measuring signal strength. In each time-window *win*, a node periodically sends a link layer beacon message to its neighbors. When the neighbor node receives such a beacon message, the extended device driver interface of the receiving node measures the signal strength at which the beacon was received. In our approach we use the *receive signal strength indicator* (RSSI) unit to measure the signal strength. RSSI is the IEEE 802.11 standard for measuring radio frequency (RF) energy sent by the circuitry on a wireless network interface card (W-NIC). It is a numeric integer value with an allowable range of 0 to 255. However, for the sake of our model we give a transformation to this recorded signal strength value by dividing it by 255. This scales the received signal strength value within the range $[0,1]$. We require this transformation as the final value of the component 'properties' lies within $[-1,1]$. At the end of each time-window, we take the average of these values. This transformed average signal strength value is then stored under the column SS_{avg} in the table PT corresponding to the neighbor. All these signal strength values within the time-window *win* is kept in a separate temporary property table, $PT_{tmp}^{Node\_id} = \langle SS, SB \rangle$ where SB is the *stability bit* explained next.

Measuring stability factor. The stability factor indicates the stability of a node. Higher the stability more reliable is the node to forward a packet. We derive the stability factor using signal strength. The reason is as follows: if a node is locationally unstable, it will have a varying signal strength. Alternatively, if the average signal strength of a node is fairly constant over few time-windows, the link with the node can be considered as stable. Therefore, after storing the strength of received signal, say $SS_{current}$, in $PT_{tmp}^{Node\_id}$, it is compared with the value present in SS_{avg} of PT. If $SS_{current} < SS_{avg}$ then the SB is set to 0, otherwise the default value 1 is kept. At the end of time-window, the stability factor SF is calculated as,

$$SF = \frac{\text{number of bits set to 1 under SB}}{\text{Total number of bits under SB}}$$

At the beginning of each time window the temporary property table $PT_{tmp}^{Node\_id}$ is set to its default values. The default value for SS is 0 and for SB is 1.

Measuring properties. The *properties* component of the node N_e is then computed as

$$_{N_r}P_{N_e} = \alpha * SS_{avg} + (1-\alpha) * SF \tag{2}$$

where $\alpha \in (0,1)$ is a fraction used as the relative importance weight to the signal strength property.

3.2 Computing *recommendation*

Each trust-aware node agrees to provide a 'recommendation' about its neighbors upon receiving a *recommendation request* from a source node. Let N_r request N_k for a recommendation about a node N_e. The source node N_r sends this recommendation request by sending a special message REC_REQ containing the node_id of the target node (in this case N_e). The node N_r can choose this recommender using a *threshold_trust* value T_{thr}. That is, if $v(N_r \longrightarrow N_k) \geq T_{thr}$ then only N_r sends a REC_REQ message to N_k. If N_k has a trust relationship with N_e, then N_k replies by sending a message REC_RESPONSE containing the pair $\langle node\_id, V \rangle$ where $V = v(N_k \longrightarrow N_e)$. The node N_r then scales this recommendation with the trust value that it has on N_k. Averaging all such recommendation received gives the 'recommendation' (or, rating) of N_e. Formally, if N_r receives m recommendations about N_e, then

$$_{N_r}R_{N_e} = \frac{1}{m}\sum_{k=1}^{m}\{v(N_r \longrightarrow N_k) \times v(N_k \longrightarrow N_e)\} \tag{3}$$

The truster N_r maintains a list, called *recommendation list*, $RL = (node\_id, list)$ for each trustee where structure of each item in the list is (node_id, recommendation_value).

3.3 Computing *interaction*

Interaction is modeled as cumulative effect of events encountered by a truster node N_r regarding N_e. We classify interaction in two categories – *packet forwarding interaction* – when the truster considers the behavior of the trustee as a packet forwarder, and *rating*

interaction – when the truster considers the behavior of the trustee as a recommender. Every event in each of these categories has binary outcome; either the truster N_r has trust-positive event or a trust-negative event depending whether the event contributes toward a trust-positive interaction or a trust-negative interaction.

Evaluating packet forwarding interaction. To evaluate packet forwarding interaction, the truster N_r checks the outcome of each packet forwarded to N_e within the specific time window *win*. Each packet forwarded correctly towards the destination is considered as a trust-positive event. Each dropped packet gives rise to a trust-negative event. The node N_r measures the number of forwarded packet by N_e as follows: N_r forwards packets to N_e and with every such packet N_r sends an ECHO message with a *time-to-live* (TTL) $= 2$. Each reply received by N_r denotes correct forwarding of the packet by N_e to the next member N_k in the path. N_r keeps this information in a table $IT = \langle Node\_id, PFC_p, PFC_n, RC_p, RC_n \rangle$ where PFC_p denotes the counter for trust-positive packet forwarding interaction within the window and PFC_n counts the trust-negative packet forwarding interactions. RC_p and RC_n are rating counters used for counting the results of rating interactions. All these fields has default value 0. Whenever a packet is dropped the counter PFC_n is increased by 1 and for each received reply the counter PFC_p is increased. Formally, *packet forwarding interaction*, denoted by I_{pf} of N_r about N_e within the window *win* is defined as the ratio $\frac{PFC_p - PFC_n}{PFC_p + PFC_n}$.

Evaluating rating interaction. The *rating interaction* is evaluated in a similar manner. We assume that each node agrees to provide a 'trust recommendation' about its neighbors upon receiving a *recommendation request* from a source node. We also assume that for each neighbor, the truster keeps a list of node_ids of the nodes who have provided recommendation for that neighbor. Whenever the truster has a 'packet forwarding interaction' with the neighbor node and the result of that interaction matches with the recommendation, that is the truster has positive (negative) experience and the recommendation is also positive (negative), it increases the RC_p in IT of all such recommenders by 1. If there is a mismatch between the outcome and the recommendation, the truster increases the RC_n counter by 1 for those recommenders. For example, let the trustee N_e has provided "positive" recommendations for the nodes N_i, N_j, N_k to the truster N_r. Therefore, in the recommender list RL, N_e appears in the list against each of these nodes. Let N_r have trust-positive packet forwarding interaction with N_i, N_j and trust-negative packet forwarding interaction N_k. Then in the interaction table IT, for the node N_e, the counter RC_p is increased twice and RC_n once. At the end of time window *win* the *rating interaction*, denoted by I_r of N_r about N_e is defined as the ratio $\frac{RC_p - RC_n}{RC_p + RC_n}$.

Evaluating interaction. The *interaction* component of the node N_e is evaluated as

$$_{N_r}I_{N_e} = \beta * I_{pf} + (1 - \beta) * I_r \tag{4}$$

where $\beta \in (0,1)$ is a fraction used as the relative importance weight to the packet forwarding interaction.

3.4 Computation of Final Trust Value

After computing values of the components we evaluate the trust value for the trustee N_e as the average of the components. Formally,

$$v(N_r \longrightarrow N_e) = \frac{{}_{N_r}P_{N_e} +_{N_r} R_{N_e} +_{N_r} I_{N_e}}{3} \tag{5}$$

These information are kept in a trust table, $TT = \langle$node_id, properties, recommendation, interaction, trust_value, cost$\rangle$. After each window *win* this table is updated with new values which are kept and used in the next time window. All other tables are set to their corresponding default values. Next section describes the modified distance vector routing algorithm which finds the path with minimum average cost for forwarding a packet to the destination.

4 Data Path Discovery

To select the most trustworthy path, each node evaluates and dynamically updates the trust components between itself and current neighbors. It then calculates the trust value of the neighbors by the process described in section 3. These values are used to calculate the forwarding cost between two neighbors, using the equation 1 in section 2.1. The path with the minimum average forwarding cost is preferred and the adjacent node on this path is trusted to forward the packets toward the destination.

4.1 Route Discovery

Our algorithm is based on a "rumor" about paths from neighbors. This is incomplete information. We thus choose to use the average and standard deviation of the running sum of cost in our route discovery protocol. This formula does not require the complete path information yet can correctly evaluate the path's reliability like the one with the complete path information. The average and standard deviation of running sum are computed as follows: let, a random variable X take on the values $x_1, \ldots, x_n$ and x be the latest value. We use the following equations:

$$AVG = (x + \sum_{i=1}^{n} x_i)/(n+1) \tag{6}$$

$$SD = \sqrt{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2/n(n-1)} \tag{7}$$

When a node receives a route information message from a neighbor node N_k, it updates the forwarding cost on the path towards node N_j (where node N_k is chosen as the next hop) by adding the current cost between itself and N_k and calculate the updated average cost using equation 6. It then re-evaluates the path to choose the optimal route to the destination. This process compares among all possible candidate routes and chooses the path that has the minimal average cost. If more than one candidate paths have same minimum average cost or have a difference of cost less than a given threshold η, the routing algorithm selects the path that has the least standard

Algorithm 1. Route Discovery in Pervasive Computing

Description: Route Discovery procedure simplifies the modified Distance Vector algorithm.
Input: destination N_j, reachable from node N_k
Output: : The routing table of a given source node S
Initialization:
Initialize cost to all nodes N_j known to S to ∞
Calculate the trust between S and its immediate adjacent node N_k in S's neighbor list
Add all immediate adjacent nodes to the routing table
for all node N_k in the neighbor list **do**
  compute trust between S and N_k (equation 5)
  compute average cost $D^S(N_k, N_k)$ (equation 6)
  compute running sum $DT(S, N_k)$
  compute running sum of squares $DT^2(S, N_k)$
end for
Iteration:
Wait until S detects change from its immediate link N_k or receives a routing packet from its neighbor /* This packet contains the information about the destination node N_j */
if S detects change in its immediate link **then**
  Update the cost and propagate the change to all neighbors
else
  if N_j is a destination that S has never seen before **then**
    Compute routing cost to N_j
    Compute running sum, running sum of squares, and hop count to N_j
    Add N_j and its routing parameters into the routing table
  end if
  if N_j is already in the routing table **then**
    Compute the routing cost to N_j
    Update the routing table if new cost is better than the current cost in the routing table
    Announce the new routing table to neighbors
  end if
end if

deviation as an optimal path. The standard deviation is calculated using the equation 7. Algorithm 1 gives the protocol used in generating the routing table. It consists of two phases: table initialization and iteration. The table initialization phase establishes paths to all immediate neighbors known to the source S. For each neighbor N_k, node S keeps track of hop count, average cost (calculated from equation 5), running sum of cost ($DT(S, N_k)$), and running sum of square of cost ($DT^2(S, N_k)$). These cost parameters are used for calculating the average cost and standard deviation according to the equation 6 and equation 7. The iteration phase is only triggered upon receiving the routing packets or upon changing of an immediate link with its neighbor.

In the first case, if the destination node N_j in the received packet is not known by node S, it will add N_j to the routing table and compute the routing cost to N_j by adding its trust between itself and its neighbor node who has sent the routing information of N_j to S. The routing cost to the destination N_j is computed as:

$$D^S(N_j,\ N_k) = \frac{v(S \longrightarrow N_k) + DT(N_k,\ N_j)}{hop\_cnt(N_k,\ N_j) + 1} \tag{8}$$

where Node N_k is the sender of the routing information and $DT(N_k,N_j)$ is the forwarding cost from N_k to N_j. If S already knows the destination node N_j, it recomputes the routing cost to N_j and compares this value with the existing value. If the new cost is less or more stable[1] than the current cost, the cost to the destination N_j is updated. If the trust value between node S and its immediate neighbor N_k has changed, S has to recompute the routing cost to all destinations N_j where N_k is the next hop. The above equation 8 is used to recomputing the new routing cost. Then S compares the new cost to the current cost that S has in its routing table. If the new cost is less or more stable than the current cost, the cost to the destination N_j is updated.

5 Analysis

5.1 Security Analysis

The trust-based approach to routing is intended to minimize the effect of malicious nodes in the network. We discuss how the proposed scheme can reduce this effect. A malicious node can subvert the network in two ways:

Dropping packets. A malicious node on a path can deliberately drop the legitimate packets. Suppose a node N_i is sending a packet to N_j through the neighbor N_m who is malicious and drops packets arbitrarily. With every drop of packet, N_i increases the 'trust-negative packet forwarding counter' PFC_n corresponding to the node N_m in the interaction table. Note, in our scheme N_i cannot differentiate between a deliberate drop of packet and a packet drop due to valid reasons (like broken link or downtime of a node). However for a malicious node, number of dropped packets will be high compare to number of forwarded packets. This will lower the ratio $\frac{PFC_p - PFC_n}{PFC_p + PFC_n}$ and consequently N_i's trust on N_m will be low. Even if N_m keeps oscillating packet drop behavior, the above ratio will be close to zero and does not help N_m to increase its trust. Also note that N_m cannot fool N_i by dropping the actual data packet but forwarding the ECHO message to N_j. Because N_j will not reply to the ECHO message unless it receives the corresponding data packet.

Providing false recommendation. A malicious node N_m can disrupt the proper functioning of the scheme by providing false recommendation about a node. Suppose the malicious node N_m provides a "positive" rating about nodes N_k, N_l where both of them are malicious nodes. Every time N_i encounters a trust-negative packet forwarding event with any of them, N_i increases the 'trust-negative rating counter' RC_n of N_m which lowers the ratio $\frac{RC_p - RC_n}{RC_p + RC_n}$. Therefore, even if N_m behaves properly in the context of packet forwarding, it cannot subvert the system by falsely 'campaigning' for some other malicious nodes in the network. This also reduces the effect of collusion of malicious nodes

[1] This is used in the case when the new cost is equal to the the current cost or has a slight difference. The path that has less standard deviation is said to be more stable path.

to disrupt reliable routing. Similar action prevents the problem of 'badmouthing' i.e., when N_m provides false 'negative' rating about a 'good' node. This is possible because the trust of the benign node is not dependent just on rating provided by N_m, but involves other parameters on which N_m cannot have any control.

The above discussions show that the proposed trust-based routing scheme can reduce the effect of malicious nodes – working as individual or as a part of collusion, in attacks like arbitrary packet drops, false data injection and badmouthing.

5.2 Complexity Analysis

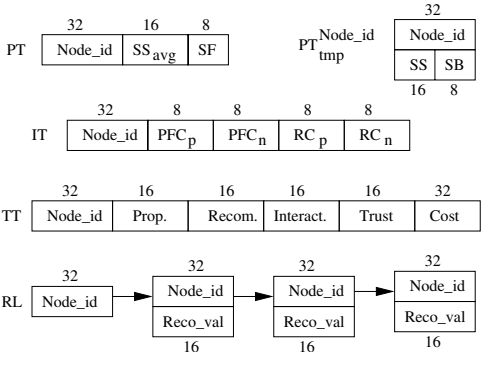
As mentioned earlier, a typical device in a pervasive computing environment has relatively low resources in terms of storage and power. However, it needs to do some computations to evaluate the trust and cost. It also needs to store some values for a specific duration. In this section we analyze the computation, communication, and storage complexity of our protocol.

Computation complexity. In our approach the run-time complexity of the routing algorithm is not affected to a great extent. Our scheme only changes the metric of computing the administrative distance between nodes in the pervasive computing environment. It requires two additional computations for the running sum and the standard deviation. These additional computations do not change the big-O complexity of running time of the routing protocol as they are proportional to the size of the network. However, our approach requires additional methods in order to evaluate the trust between nodes. All these computations are simple arithmetic computations and have linear bounds. Consequently these do not add much to the computation overhead of the proposed protocol.

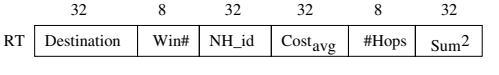
Communication complexity. A node maintains view of connectivity and trust relationship by periodically transmitting a beacon packet. This packet carries the node information including sequence number, hardware address, hardware protocol, and trust recommendation upon request. According to this scheme, there are 3 types of the beacon message – announcement, recommendation request, and recommendation reply. When a node receives a beacon message, it identifies the sender, measures the strength of the beacon signal (section 3.1), collects the trust information and recalculates the trust value of the sender. If the beacon message is a recommendation reply and the trustee is recognized by the receiver, it recomputes the trust value of the trustee with this recommendation information about the trustee. If the beacon message is a recommendation request message, and the trustee is recognized by the receiver, it prepares the trust value of the trustee and sends it to the requester with the next beacon message. We have estimated that a message 256 bits (32 bytes) long is sufficient to carry the beacon message. In the following discussion we show that routing tables in our protocol require only 154 bits for each record. Therefore, for a small network (say, with 50 nodes), the nodes pass a routing table which is of size less than 1KB.

Storage complexity. In our protocol each node has to maintain a certain number of tables. In this section we discuss the storage overhead that is required to store these

tables. For each neighbor, a truster node needs to maintain the following tables and list: PT (properties table), $PT_{tmp}^{node\_id}$ (temporary property table), IT (interaction table), TT (trust table), and the list RL. In any table the node_id field takes 32 bits to store the address. We express the values of SS_{avg} in PT, SS in $PT_{tmp}^{node\_id}$, and *Properties, Recommendation, Interactions, Trust* in TT using 16 bits in which the most significant bit is the sign bit, the next bit expresses the exponent, and the rest 14 bits expresses the fraction. We need only 1 bit to express the exponent as all these numbers are within $[-1,1]$. Also, we get the precision of $1/2^{14}$ for trust related values. In the interaction table IT we use 8 bits to express each counter. The signaling factor is also expressed using 8 bits. However, we use 32 bits to represent the *cost* in TT as it is not bounded. Since the cost is always positive, we use first 16 bits for the exponent and the last 16 bits for the fraction. This gives the precision of $1/2^{16}$ for the cost value which is accurate enough for the environment. The figure 3(a) shows the structure of the tables stored for each neighbor. From the figure we see that the tables PT, IT, TT require $56, 64$ and 128 bits respectively for each record. Each of these tables is maintained for all the neighbors. If a node has m neighbors, then it requires $m \times 228$ bits. To store each signal and stability information we need 24 bits. If we assume that a node receives s signals within a window, then for each neighbor it needs $32 + s \times 24$ bits to store the signal. Hence for all m neighbors the node requires $m.(32 + s.24)$ bits. For the recommendation list RL each record requires 48 bits and we assume at most k $(k \leq m)$ recommenders recommend a neighbor. Therefore for each neighbor it requires $32 + k \times 48$ bits and hence for m neighbors $m.(32 + k.48)$ bits. Note, we have not considered the pointer size here as it depends on the implementation. Hence for each neighbor a node requires $228 + (32 + 24.s) + (32 + k.48) = 292 + 24.s + 48.k$ bits and for all m neighbors



(a) Tables maintained for each neighbor



(b) Routing table of the node

Fig. 3. Storage structure of the tables maintained in each node

total storage required is $m.(292 + 24.s + 48.k)$ bits. This storage is not significant if we assume that each node in the pervasive computing environment interacts with a small number of neighbors. For example, in a pervasive computing environment topology where each node interacts with at most 20 neighbors and at most 100 signals are received within a time window *win*, then maximum number of bits required to store the trust information is $(292 + 24 \times 100 + 48 \times 20) \times 20 = 3652 \times 20 = 73040$ bits $\approx$ 9KB.

Each node also stores a routing table whose structure is shown in the figure 3(b). The *Win*# field keeps the last window number. #*Hops* stores the number of hops to the destination. NH_id field stores the address of the next hop towards destination. Cost_{avg} and $\sum^2$ fields store the cost metrics which are used in the route selection protocol. We also need the metric $\sum$ for route selection. However we do not store this information in the routing table as it can be derived from $Cost_{avg}$ and #*Hops*. Each record of this table requires 154 bits and hence size of the routing table for each node in the topology is $O(154n)$ where n is the number of nodes in the network. In our example if we assume a small network with 50 nodes then each node require maximum 154×50 bits which is $\approx$ 0.9KB. Therefore all together a node requires only 10KB storage space to store the information related to trust-based routing.

The above analyses show that our protocol is light-weight in terms of trust evaluation, feedback management, message passing, and storage, thereby making it suitable for pervasive computing environment.

6 Conclusion and Future Work

In this work, we address the problem of reliable delivery of event data in pervasive computing environments to appropriate action points in order to support obligation policies. The problem is modeled as a routing problem. We present a trust-based approach to routing. Each node measures trustworthiness of its neighbor based on the neighbor's properties like signal strength and signal stability, a neighbor's behavior in forwarding packets from the node as well as in recommending other nodes, and a neighbor's rating by other neighbors. We represent each link in the network as a trust relationship with a numeric value between $[-1, 1]$. This trust metric reflects reliability of a node as a packet forwarder. We have proposed a cost metric, which is inversely related to the trust metric, for each link in the network. We next adapt the distance vector routing algorithm for routing in pervasive computing environments. The modified algorithm uses the cost value assigned to each link to find the minimum average cost path from source to destination. We have discussed how our protocol can reduce the effect of malicious nodes. We have also shown that the scheme does not enhance the computation, communication, and storage overhead to any significant extent.

A lot of work still remains to be done. The proposed scheme is generic in nature. We need to modify specific ad hoc routing protocols using our metrics and compare the results to evaluate the performance of the proposed scheme. We plan to run simulation experiments to compare the routing results with existing ad hoc routing protocols. We are also looking for other node properties like remaining battery power to extend the 'properties' parameter of the trust metric.

References

1. Balfanz, D., Smetters, D., Stewart, P., Wong, H.: Talking to Strangers: Authentication in Adhoc Wireless Networks. In: Symposium on Network and Distributed Systems Security (NDSS '02), San Diego, California, USA (February 2002)
2. Chiang, C.C., Wu, H.K., Liu, W., Gerla, M.: Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel. In: 5th IEEE Singapore International Conference on Networks (SICON'97), Kent Ridge, Singapore pp. 197–211 (April 1997)
3. Corson, M.S., Ephremides, A.: A Distributed Routing Algorithm for Mobile Wireless Networks. Wireless Networks 1(1), 61–82 (1995)
4. Gafni, E., Bertsekas, D.: Distributed Algorithms for Geneerating Loop-free Routes in Network with Frequently Changing Topology. IEEE Transaction and Communication 29(1), 11–15 (1981)
5. Gerla, M., Tsai, J.T.: Multicluster, Mobile, Multimedia Radio Network. Wireless Networks 1(3), 255–265 (1995)
6. Hu, Y.C., Perrig, A., Johnson, D.B.: Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. In: Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MobiCom'02), Atlanta, Georgia, USA (September 2002)
7. Papadimitratos, P., Haas, Z.: Secure Data Transmission in Mobile Ad Hoc Networks. In: ACM Workshop on Wireless Security (WiSe'03), San Diego, California, USA (September 2003)
8. Perkins, C.E., Bhagwat, P.: Highly Dynamic Destination-Sequenced Distance Vector (DSDV) for Mobile Computers. In: Conference on Communication Architectures, Protocols and Applications (SIGCOMM'94), London, UK, pp. 234–244 (August 1994)
9. Toh, C.K.: A Novel Distributed Routing Protocol To Support Ad hoc Mobile Computing. In: IEEE 15th Annual International Phoenix Conference on Computers and Communication (IPCCC'96), Phoenix, AZ, USA, pp. 480–486 (1996)
10. Zhou, L., Haas, Z.J.: Securing Ad Hoc Networks. IEEE Network 13(6), 24–30 (1999)
11. Zouridaki, C., Mark, B.L., Hejmo, M., Thomas, R.K.: A Quantitative Trust Establishment Framework for Reliable Data Packet Delivery in MANETs. In: Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'05), Alexandria, VA, USA, pp. 1–10. ACM Press, NewYork (2005)
12. Awerbuch, B., Holmer, D., Nita-Rotaru, C., Rubens, H.: An On-Demand Secure Routing Protocol Resilient to Byzantine Failures. In: ACM Workshop on Wireless Security (WiSe'02), Atlanta, GA, USA, pp. 21–30 (September 2002)
13. Dube, R., Rais, C.D., Wang, K.Y., Tripathi, S.K.: Signal Stability-Based Adaptive Routing (SSA) for Ad Hoc Mobile Networks. IEEE Personal Communications Magazine 4(1), 36–45 (1997)
14. Yi, S., Naldurg, P., Kravets, R.: Security-Aware Ad Hoc Routing for Wireless Networks. In: Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC 2001), Long Beach, CA, October 2001, pp. 299–302. ACM Press, New York (2001)
15. Ray, I., Chakraborty, S.: A Vector Model of Trust for Developing Trustworthy Systems. In: Samarati, P., Ryan, P.Y. A., Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 260–275. Springer, Heidelberg (2004)