18th International Conference on Information Fusion
Washington, DC - July 6-9, 2015

Trust Metric Integration in Resource Constrained Networks Via Data Fusion

Thomas A. Babbitt
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180
Email: babbit@rpi.edu

Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180
Email: szymab@rpi.edu

***Abstract*—There are numerous environments and situations where network infrastructure is sparse, latent, or partially out of service. There is a growing body of research on protocols, security, information assurance and trust for use in such Resource Constrained Networks. A few examples of a Resource Constrained Network include Delay Tolerant, Wireless Sensor, and many mobile ad-hoc and ad-hoc mesh networks. Data fusion of direct observations and recommendations from other nodes into an aggregate trust value on a given node allows for more efficient routing and enables information assurance security services that support data availability, confidentiality, integrity, authentication, and non-repudiation. Selecting proper data metrics and fusion scheme allows nodes in a network to quickly converge on an accurate trust value for a given node. This minimizes security risks and provides better quality of service to properly functioning nodes as well as quickly isolates malicious ones. This paper discusses our current work on distributed trust management schemes for use in Resource Constrained Networks. Specifically, it introduces fusion methods to integrate diverse clues into a composite trust value.**

I. Introduction

There is a need for reliable network communication in all environments; however, there are situations where this is challenging due to a lack, destruction, cost or overwhelming of network infrastructure. These conditions are likely to be found in battlefield, emergency response or protest/large gathering scenarios. A class of Resource Constrained Networks (RCN) has been defined to help fill the void that exists when infrastructure or node hardware degrade, or are cost prohibitive, to the point where the use of traditional network protocols for routing and security are ineffective.

A RCN is any network that must function with a limited resource or resources that cause significant modification to traditional routing and security protocols. One example is a Delay Tolerant Network (DTN). Due to node mobility, spareness, and hardware constraints (an example is battery), a DTN does not maintain end-to-end routing tables. This constraint led to the introduction of numerous routing protocols to ensure availability of information. These protocols run a spectrum from endemic, sending a message to all nodes, to schemes such as spray-and-wait that send only to a subset of nodes [1]. A second example is Wireless Sensor Networks where nodes must minimize battery power by making the fewest possible broadcasts; an example routing protocol is found in [2].

In order to provide more than availability, nodes must be able to make routing and other decisions based on some form of authentication. Traditionally this is done using shared keys or certificates each of which could be validated or updated efficiently over time across the network. Due to network constraints, this is not efficient in most RCNs and not possible in others. Authentication can be done in a probabilistic and distributed manner using node trust values. There are many definitions of trust and principles proposed for managing trust in a resource constrain network. *Cho et al.* conduct a good review of trust in various academic fields and propose a number of principles for use in a Mobile Ad-Hoc and DTN including that trust be dynamic, subjective, asymmetric, context-dependent and not necessarily transitive [3].

The use of these basic principles necessitate that a distributed trust management system for a RCN include the fusion of both a direct and indirect trust components. Computational trust has been studied in many disciples and an overview is in [4]. A good overview on mobile AD-Hoc network trust computations and dynamics is in [5]. The direct trust component include any metrics or clues that can be directly observed by the node collecting trust to assist in making trust decisions. One example is the observation of a node properly forwarding a message. Direct trust is restricted to what a node can itself observe, dynamic over time, and asymmetric. The indirect trust component consists of another nodes trust information received by the node computing its fused trust value. In some respects this is similar to managing recommendations and reputation. Because trust is asymmetric, and node behavior can change over time, indirect trust in not transitive. There are a number of models and schemes for use in multiagent systems one example is [6], which have many of the same properties as the trust management schemes proposed for use in RCNs.

There are a number of proposed trust management schemes for use in a Resource Constrained Network [7]–[10]. Most of the schemes are focused on Delay Tolerant Networks, but many of the concepts can be generalized to the larger class of RCNs. The basic concept for each scheme is to maintain direct and indirect trust metrics and combine those to create an aggregate trust as a value between $(0.0, 1.0]$ where 0.0 is a complete lack of trust and 1.0 is complete trust. Based on that value, routing and security decisions can be made. Yet, none of the schemes directly dictate to which nodes to forward a message or which to restrict. Based on risk, a minimum value can be set and any node above the threshold is a potential candidate, while nodes

below are "blacklisted." This value can differ depending on the category or type of network traffic and is node dependent.

This paper focuses on the fusion of those trust metrics and proposes a method for use in the trust management scheme introduced in [10]. We will first provide background on the trust management schemes listed in Section II. This is followed by proposed methods to integrate trust clues described in Section III. Section IV provides results justifying our approach. Future work and conclusions are discussed in Section V.

II. Trust Management Scheme Background

As in situations where people interact, trust of person A on person B is based on observed actions of person B and on opinions of mutual "friends" about person B. In a RCN, traditional network trust verification methods do not function efficiently and for some constraints do not work at all. Nodes, in RCNs, like people, make trust decision based on direct and indirect observations. Under this set of conditions, the ability to trust at the proper level leads to better results.

Fusing direct and indirect metrics into an integrated trust within a cohesive management scheme is the source of many new research endeavors. There are multiple schemes proposed in literature that integrate direct and indirect node observation and use the principles listed in [3] to manage trust in DTNs. In [7] the authors use a Bayesian approach to determine the probability that a node is acting good. In [8] the authors creates a bipartite graph and find outliers; the scheme removes nodes with probables outside of a certain value in order to converge on a coherent trust value. A third approach outlined in [9] determines good versus bad encounters over four categories to aggregate a trust value. In [10] path redundancy is used.

A. Bayesian Approach

Denko et al. propose a Bayesian Learning Approach to determine the trust probability of a given node in a network [7]. It takes into account the observed actions, positive and negative, and updates trust levels for each node. Equation 1 [7] is used to calculate the direct trust between node A and node B. Each node maintains two parameters about every other node. The number of good interactions denoted as n_s and the number of negative interactions denoted as n_u. The authors assume that complete information cannot be collected. So they compute a trust value using the beta distribution with parameters $\alpha = n_s + 1$ and $\beta = n_u + 1$ as shown below.

$$T_A(B) = E(f(x;\alpha,\beta)) = \frac{\alpha}{\alpha+\beta} = \frac{n_s+1}{n_s+n_u+2} \quad (1)$$

Modifying for indirect trust computation is done using Equation 2 [7]. The authors calculate indirect trust using a combination of observed interactions and recommendations from other nodes. Assume that a node has i recommendations for another node from k different sources. Using recommendations modifies Equation 1 and makes the updated trust value for $T_A(B)$ defined as follows [7].

$$T_A(B) = \frac{n_s + \sum_{k=1}^{i} n_s^k + 1}{n_s + n_u + \sum_{k=1}^{i} n_s^k + \sum_{j=1}^{i} n_u^j + 2} \quad (2)$$

Because recommendations can be false, the authors add a threshold for excluding or judging recommendations. Since a history of interactions and recommendations is maintained for a time period there is a decay of weight given to each as time progresses. So older recommendations receive a lower weight than newer recommendations. Integrating judging and decaying weight of recommendations and interactions over time create a more robust scheme.

B. Iterative Trust and Reputation Management Mechanism (ITRM)

Ayday et al. propose a Trust and Reputation Management Mechanism (ITRM) in [8]. ITRM is a graph based iterative algorithm with two main goals: computing the reputation of nodes that send message (author designate Service Providers) and determining the trustworthiness of a recommending node.

The first step for ITRM is to complete a bipartite graph between service providers (SP) and nodes that recommend (R). Each rater is a *check vertex* and each SP is a *bit vertex*. The authors in [8] compute an initial value of each bit-vertex j using the following equation.

$$TR_j = \frac{\sum_{i \in A} R_i \times TR_{ij}}{\sum_{i \in A} R_i} \quad (3)$$

An inconsistency factor is computed for every check-vertex i as follows.

$$C_i = \left[\frac{1}{|\Upsilon|}\right] \sum_{j \in \Upsilon} d(TRij, TR_j) \quad (4)$$

If the inconsistency is greater than τ for any check-vertex(es), then the node with the largest discrepancy is "blacklisted" and all its ratings are deleted. Equation 3 [8] is then recalculated minus the "blacklisted" check-vertex while the inconsistency is recalculated using Equation 4 [8]. The inconsistencies are checked again. This process continues until no check-vertex inconsistency is greater then τ.

C. Trust Thresholds - Trust Management Protocol

The authors in [9] use the terms *QoS trust* and *social trust* to represent direct and indirect trust respectively. The former includes two metrics "connectivity" and "energy" and the later "unselfishness" and "healthiness." The trust value node i has for node j at time t is computed as follows.

$$T_{i,j}(t) = \sum_{X}^{all} w^X \times T_{i,j}^X(t) \quad (5)$$

The value X is one of the four aforementioned trust properties (connectivity, energy, unselfishness, and healthiness). The weight given each is w^X and the sum of all weight is 1. The weight values can be application or network configuration dependent.

To compute a trust value over time for one trust property both direct and indirect trust are used. The trust value is calculated using the following [9]:

$$T_{i,j}^X(t+\Delta t) = \left(\beta T_{i,j}^{direct,X} + (1-\beta) T_{i,j}^{indirect,X}\right)(t+\Delta t) \quad (6)$$

β selected from the range [0,1] represents the fraction of the unit significance or weight given to direct versus indirect observations. Each different trust property has a different β to help ensure proper tuning of trust values. Each encounter with another node triggers a trust update. For the encountered node j, node i uses it direct trust and for other nodes it uses trust that node j reports to it. If node i encounters node j and there is not enough time for data to be transmitted there is a small decay of the last direct trust level for j and indirect trust based on node j's recommendations. Each trust property has a different method for updating trust based on the number of encounters, willingness to forward a message for another node, or battery power levels.

D. Path Redundancy

The authors in [10] use path redundancy. The source creates a message M consisting of the message payload with an appended checksum. Using erasure coding, M is broken into s segments. The message is encoded so that k_{EC} segments will allow for the recreation of M and $s > k_{EC}$. Each message segment $m \in M$ flows through the network from source to destination along multiple paths. The destination maintains a set of message segments received for each M designates n_M. Once $|n_M| = k_{EC}$ the destination attempts to recreate M. If successful, trust increases for all good paths and the destination waits for additional segments m for a given time prior to sending an acknowledgement. By checking those with $k_{EC} - 1$ known good segments additional trust information is gathered. If the destination cannot recreate M it waits for more segments or requests M be resent based on the utility functions defined in [10]. The complete state diagram and the value of full path knowledge versus only using directly connected nodes is found in [11]. The authors show that the scheme has merit, it is based on direct observations only and does not fuse indirect and direct observations to make a comprehensive aggregate trust value usable for routing and other decisions.

III. Trust Management Metric Integration Proposal

The trust management schemes listed above use different approaches to integrate trust metrics. We integrate trust using two components. The first is the integration of indirect and direct trust which happens at the end of a time period Δt of collecting indirect trust. Depending on network conditions, indirect trust might receive more or less weight depending on node trust variance, time of trust collection, or last seen time. The second integration is modification of indirect trust which happens each time a node meets another node and receives its trust for other nodes in the network. This trust component is based on recommendations from other nodes and is the most susceptible to malicious activity. Below we propose integration of trust based on the scheme outlined above in Section II-D.

A. Direct and Indirect Trust Integration

All of the proposed trust schemes, listed above, determine direct trust based on observation of different clues or metrics. These are aggregated either as a count or as a modification of trust based on the observed actions. The integration of direct and indirect trust varies as well. The proposal in [9] shown in Equation 5 and Equation 6 most closely resembles the approach here.

There are three vectors of size n that node k maintains. The first is the indirect trust vector CT^k that maintains indirect trust for all other nodes in the network based on trading trust information (see Section III-B below). The second is the direct trust vector DT^k that maintains the trust based on direct observations. The final vector is the aggregate trust vector AT^k that is the fusion of the previous vectors. There are multiple different approaches outlined below to fuse the direct and indirect trust values. The first uses a fixed weight for the indirect observations and one minus that weight for direct ones. The second does so variably based on the current trust of recommending nodes. The third takes into account decay of direct trust over time. The fourth combines approaches two and three.

The use of fixed weights minimizes processing requirements and is based on the following equation for all $j \in N$, where N is the set of all nodes in the network.

$$AT_j^k = (1 - \alpha_a)\, DT_j^k + \alpha_a CT_j^k \tag{7}$$

This is straightforward and depending on the value of α_a can give more or less weight to direct versus indirect observations.

Equation 7 is modified to take into account the current trust of the nodes that give indirect trust information. The tracking and fusion of that is detailed in the next section; however, the average trust of those nodes can easily be determined and is designated as CT_{av}^k. Modifying Equation 7 by substituting $\alpha_a CT_{av}^k$ for α_a the trust of a recommending node is accounted for. If, during a give time period Δt, suspect nodes are met then their recommendations are given less weight than if more trusted nodes are met.

Slightly modifying DT^k and making it an $n \times 2$ matrix where $DT_i^k = DT_{i,1}^k$ is the direct trust value and $DT_{(i,ts)}^k = DT_{i,2}^k$ is the last time node i was met allows for the direct trust value to decay if a given node has not been seen for an extended period of time. The number of time periods Δt is found using

$$t_i^k = \lfloor \frac{CT - DT_{(i,ts)}^k}{\Delta t} \rfloor \tag{8}$$

and the updated value used for trust decisions is found using the following.

$$AT_j^k = \left(1 - \alpha_a^{\frac{1}{\lambda t_i^k}}\right) DT_j^k + \alpha_a^{\frac{1}{\lambda t_i^k}} CT_j^k \tag{9}$$

Equation 9 decays the weight given the direct trust by increasing the weight given the indirect trust; this decay is exponential and based on a set value for λ.

Combining both of the approaches listed above gives the following:

$$AT_j^k = \left(1 - \left(\alpha_a CT_{av}^k\right)^{\frac{1}{\lambda t_i^k}}\right) DT_j^k + \left(\alpha_a CT_{av}^k\right)^{\frac{1}{\lambda t_i^k}} CT_j^k \tag{10}$$

This takes into account the trust of the nodes that give recommendations during Δt and the decay of the direct trust for a node that has not been seen in a number of time intervals.

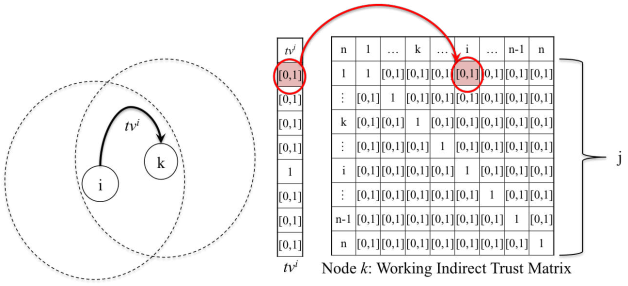


Fig. 1. Meeting Event Between node k and node i

B. Indirect Trust Integration

There are a number of potential methods to integrate inferred trust as shown in Section II. We propose that when node k meets node i the nodes trade trust vectors. These vectors include the current trust values that each node has for all other nodes in the network. There are two approaches explored below. The first is that each node k maintains a more complete history of trust and stores trust values received from all other node i over time. The second approach is to aggregate those values as a running average and update each time node k meets a node i. Our expectation is that although the second approach is an approximation of the first, it limits resource expenditure.

1) Trust Matrix: Node k maintains multiple matrices and vectors to manage and update indirect trust. They include a working and current indirect trust matrix, designated W^k and C^k respectively. Both W^k and C^k are $n \times (n+1)$ matrices. Each column is used to store trust values received from other nodes $i \in N$, where n is the number of nodes in set N. The value $C^k_{(i,j)}$ is the trust recommendation(s) received from node i about node j. The last time that column i was updated in the current trust matrix is designated $C^k_{(i,ts)} = C^k_{(i,n+1)}$ and the number of interactions with node i is $W^k_{(i,count)} = W^k_{(i,n+1)}$. The column for node k in C^k is its inferred trust vector. Matrix C^k holds the results for previous time periods Δt and matrix W^k holds values received during the current time period. Both matrices are initialized with null values equal to -1. The integration of direct trust with indirect trust is discuss above and the method for obtaining the direct trust value is detailed in [10].

There are two types of events for node k. The first occurs when node k meets any node $i \in N$ and it is called a meeting event. Figure 1 show this type of event. The second event occurs when the time Δt expires triggering an update for node k's trust based on trust vectors received during the last time interval Δt. This is called an indirect trust update event.

The first thing that occurs during a meeting event between node k and node i is an exchange of trust vectors. This vector for node i is an aggregate trust based on both direct and indirect metrics and represent node i's current trust for all other nodes in the network; this is represented as tv^i, and tv^i_j is node i trust value for node j. Upon receipt of node i's trust vector, node k will update the column in its working trust matrix corresponding to node i's index for all $j \in N$ as follows:

$$W^k_{(i,j)} = \frac{\left(\left(W^k_{(i,count)} - 1\right) W^k_{(i,j)} + tv^i_j\right)}{W^k_{(i,count)}} \tag{11}$$

This is a simple average. For any given time period Δt, node i's trust vector should not change much and should only be counted once and not multiple times in the update of the indirect trust that occurs when node k conducts an indirect trust update. This helps to eliminate ballot stuffing of recommendations where one person, or in this case node, can have its values count multiple times and overwhelm other nodes.

Once the time Δt expires for node k, it does a trust update. There are four steps to a trust update listed below.

1) Update the current inferred trust matrix for node k
2) Update the inferred trust vector for node k
3) Fuse the inferred and direct trust vectors to create the aggregate trust vector
4) Reset inferred trust vector for node k

Updating the current inferred trust matrix is done by taking the average between the current trust matrix and update trust matrix, see Equation 12. The value $C^k_{(i,j)}$ is column i row j in node k's current trust matrix. This corresponds to the current indirect trust values being used by node k. It represents historical values that node k received from node i about all nodes $j \in N$. The value stored at $C^k_{i,ts} = C^k_{(i,n+1)}$ is the last time that column i was updated in the current inferred trust matrix. This is used to compute $\alpha_i \in [0,1]$, a "freshness factor" for the trust values. There are three cases in Equation 12 listed below.

$$C^k_{(i,j)} = \begin{cases} C^k_{(i,j)} & W^k_{(i,j)} = -1 \\ W^k_{(i,j)} & C^k_{(i,j)} = -1 \\ \alpha_i^{t_i^k} C^k_{(i,j)} + \left(1 - \alpha_i^{t_i^k}\right) W^k_{(i,j)} & \text{Otherwise} \end{cases} \tag{12}$$

1) Case one occurs when node k does not meet node i during the current time period Δt. There is no change and $C^k_{(i,ts)}$ remains the same.
2) Case two occurs when node k meets node i for the first time during the current Δt time period. This sets the values in the current matrix to the update matrix for column i. $C^k_{(i,ts)}$ is updated to the current time.
3) Case three is when node k has previously seen node i and has seen it during the current time period Δt. The average between the values in $C^k_{(i,j)}$ and $W^k_{(i,j)}$ is computed using the "freshness factor" α_i for all $j \in N$. Any value of $\alpha_i \geq 0.5$ will initially give more weight to the values in C^k versus W^k. For them to be equal, assuming consecutive time interval meeting between node k and node i, $\alpha_i = 0.5$. Equation 13 will find the number of time intervals since the last update. $C^k_{(i,ts)}$ is updated to the current time. Figure 2 show and example of this.

$$t^k_i = \lfloor \frac{CT - C^k_{(i,ts)}}{\Delta t} \rfloor \tag{13}$$

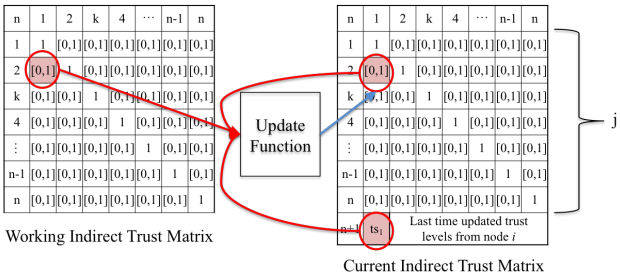


Fig. 2. Node k Updates Current Trust Matrix when Δt Expires

Once C^k is updated based on Equation 12, the indirect trust vector $C^k_{k,j}$ for all $j \in N$ is updated. This is done row by row using Equation 14. All values that are -1 are skipped in the summations in Equation 14. When $CT = C^k_{(i,ts)}$, that occurs when node i was met during the last time period, the value for $t^k_i = 1$. In Equation 12, there is no decay. Decay of values occurs after skipping one or more time periods. This makes older information impact the results less then more current information, which is what we want.

$$C^k_{(k,j)} = \frac{\sum_{i=1}^{n} \left(C^k_{(i,j)} \times \alpha_i^{t_i^k} \right)}{\sum_{i=1}^{n} \alpha_i^{t_i^k}} \tag{14}$$

The thirds step is to fuse direct and indirect values to create an aggregate trust vector (see Section III-A). The final step of the trust update is to reset W^k to all -1 values. This will ensure that a new running average is collected only for nodes seen during the next Δt time window.

2) Rolling Average: The previous section gave an overview using matrices to track recommendations that node k received from a node $i \in N$. That approach works, but in nodes with limited hardware (battery, buffer, and processing) an approximation that maintains much of the same information is merited. This is done using four vectors of size n. The current indirect trust vector is used with the direct trust vector to create an aggregate trust vector (AT^k); see Section III-A. The vectors used for indirect trust include the following:

- Current Trust (CT^k): is the current indirect trust vector for node k
- Working Count (WC^k): This vector tracks the count used to average the received information during the current time period Δt for node k based on interactions with node $i \in N$.
- Working Sum (WS^k): This vector tracks the sum used to average the received information during the current time period Δt for node k based on meetings with node $i \in N$.
- Working Number of Interactions (WN^k): This is the number of times that node k see a node i during the current time period Δt.

At the start all of the indirect trust vectors are initialized to all zeros. Similarly to using matrices there are two types of events. The first is when node k meets node i and the second is when the Δt expires for node k. During a meeting event where node k receives node i's trust vector (tv^i) the following occurs:

1) Update the working count (WC^k): This is done using the following equation.

$$WC^k_j = WC^k_j + \frac{\left(AT^k_i\right)^\beta}{2^{WN^k_i}} \tag{15}$$

This will add in the count for each interaction. This is directly connected to the number of times that a particular node i is seen during Δt. The first time $WN^k_i = 0$ adding one to the count. This is reduced exponentially such that if node i meets with k a large number of times, in a given Δt time, it still counts less than twice. Additionally the current trust value that node k has for node i is taken into account. The β term is used to modify how much to decrease the reported value based on current trust of node i. As β goes to zero the numerator in Equation 15 goes to 1.

2) Update the working sum (WS^k): This is done using the following equation.

$$WS^k_j = WS^k_j + \frac{\left(AT^k_i\right)^\beta}{2^{WN^k_i}} * tv^i_j \tag{16}$$

This is similar to the previous step except that added term is multiplied by the trust values received from node i.

3) Update the working number of interactions: $WN^k_i = WN^k_i + 1$

When Δt expires for node k, it conducts a trust update. This is done by updating the current indirect trust vector CT^k, fusing that with the direct trust vector into the aggregate trust vector and then resetting the three working vectors to all zeros. The following equation is used to update the current indirect trust vector.

$$CT^k_j = (1-\gamma)\, CT^k_j + \gamma \left(\frac{WS^k_j}{WC^k_j} \right) \tag{17}$$

Equation 17 is used to find the exponential moving or running average. The variable γ is used to determine the weight given to previous values; it can be thought of as the decay of the older values. Another way to look at it is how many previous values should be used to determine the current value. To to use the previous 10 values set $\gamma = \frac{1}{10} = 0.1$. This can then be multiplied by any number to include the average trust of the nodes seen during the current time period.

IV. Simulations

To test the integration of direct and indirect trust, we created a discrete event simulator [12]. The input for this simulator is a complete graph of n nodes with edges encoding moving patterns of nodes. Each edge weight is a random number uniformly distributed between [0.0,1.0) that is the inverse of the intermeeting time between two nodes connected by this edge and denoted as $w_{i,j}$ for the edge weight between

nodes i and j. Thus, the weights of node edges represent how often a node is likely to meet with others. If the edge weight is 1.0 then the two connect nodes are in constant contact but if it is 0.0 they never meet. This simulates an arbitrary mobility pattern in the network.

There are three main types of events. The first is a meeting event between nodes. This occurs when two nodes are within broadcast range of each other. During this type of event nodes trade messages and indirect trust information with which nodes will update their message buffers and direct trust vectors. The second type of event is a node trust update event. This occurs when a nodes timer expires after Δt. A node will update first its indirect trust information and then its aggregate trust vector using both the indirect and direct trust vectors. The third type of event is a message event which is a subclass of the node event. This will either put a new message or acknowledgement into the buffer for a given node.

At the initialization of each run, each edge is uniformly randomly assigned a weight between [0.0,1.0). Each node has an initialized working and current indirect trust matrix, direct and aggregate trust vector, and message queue. The matrices are initialized as null values and the vectors are initialized to 0.5. All queues are empty and depending on the specified percentage a number of the nodes are flagged as bad. Those nodes will modify any messages they transfer along a path from source to destination.

The simulation event queue is then populated with initial events. For meeting events, this is done for each node pair $i, j \in N$ by sampling meeting times from Poisson distribution. The variable $mTime_{i,j}$ is set to 0 and then Equation 18 is run and a meeting event is added at that time. For node updates the initial update occurs at $\Delta t * [0.0, 1.0]$. This will stagger node updates. Each node will add an initial message event at a time uniformly randomly selected between $[0.0, 50.0]$ with a randomly selected destination node. Once the initial events are added to the simulation queue it is sorted by event time and the simulation continues until all events that can occur prior to the stop time are executed.

$$mTime_{i,j} = mTime_{i,j} - \left(\frac{1}{w_{i,j}} \times ln([0,1]) \right) \qquad (18)$$

The simulator runs event by event and disregards node handshakes, broadcast collisions, broadcast times, noise, and a number of other network communication details. All of those are important and are implemented in NS3. This simulator is a high level approximation used to initially evaluate the scheme. More detailed simulations used to obtain more precise and robust results are expected to be qualitatively similar to those presented here. As each event in the simulation occurs new events are added. For meeting events that is done using Equation 18. For update events that is done adding Δt to the current simulation time. For message events that is done by randomly selecting a number between $[0.0, 50.0]$ and adding that to current time. This means on average a node will send 4 messages each 200 seconds of simulation. Once a meeting event results in message recreation, based on erasure coding, an acknowledgement event is added after a set time period. This is meant to clear the buffers of all nodes storing the now delivered message.

A. *Effect of α_i, Matrix Version*

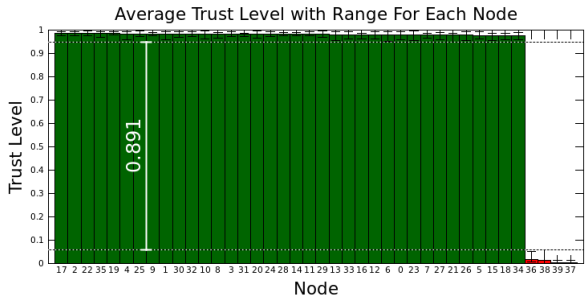
A number of simulations were run to see the effect of α_i. Figure 3 shows the results using the matrix method to track indirect trust and varying the number of malicious nodes. We ran three sets of simulations, each averaging the results over ten runs with network size $n = 40$. The nodes set to act maliciously were nodes 36 to 39 first, then nodes 30 to 39 and finally nodes 24 to 39. Subfigures 3a, 3c and 3e show results where each of the bad nodes always act maliciously. Subfigures 3b, 3d and 3f show results where the adversarial nodes flip a fair coin to decide whether or not to act maliciously.

We set α_i as 0.125, 0.25, 0.5, and 0.75; lower a_i values weaken influence of older indirect trust values on the aggregated trust. All of the results in Figure 3 are for $\alpha_i = 0.5$. The different values of α_i had minimal effect on the outcome of the simulations. When the number of good nodes is high, such as 90%, there is little volatility in trust and recommendations. When the number of good nodes is reduced to 60% or 75%, there is likely to be some change in how fast the trust values converge; however, the construction of the simulations run did not look at that variable. Speed of convergence due to α_i, is an avenue for future work.

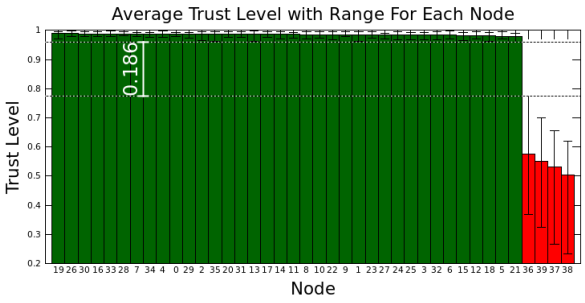
1) Fraction of Good Nodes = 90%: Figure 3a shows the results where 90% of the node are good. There is a clear distinction between the worst simulation run for a good node and the best for a bad; in this case the difference is 0.891. We expected this significant difference because of the limited number of bad nodes. Figure 3b shows the results where each bad node flips a fair coin to decide whether or not to act maliciously. As expected, the bad nodes have higher trust at the end and the range of their intermediate trust values is significantly higher then in the previous scenario, with the difference between worst good node and best bad is 0.186 versus 0.891. All malicious nodes are still clearly identifiable.

2) Fraction of Good Nodes = 75%: Figure 3c shows the results where 75% of the node are good. There is still a clear difference, but smaller than above, between the worst simulation run for a good node and the best for a bad; in this case the difference is 0.525. We expected a significant difference but smaller than the difference for 90%. Figure 3d shows the results where each bad node flips a fair coin to decide whether or not to act maliciously. As expected, the bad nodes have higher trust at the end. What was unexpected is that the range of their intermediate trust values is almost identical to the case when bad nodes always act maliciously. The difference between worst good node and best bad one is 0.525 in the first case and 0.530 in the later. All malicious nodes are still clearly identifiable.

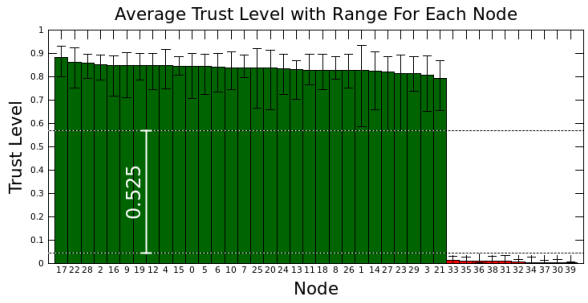
3) Fraction of Good Nodes = 60%: Figure 3e shows the results where 60% of the node are good. Even when 40% of the nodes are bad there is still a clear difference between the worst simulation run for a good node and the best for a bad (0.16). We expected a significant drop from the previous two sections especially since the number of bad nodes is closing in on 50%. Figure 3f shows the results using a fair coin to decide whether or not to act maliciously. As with 75% (section above), the difference between worst good and best bad increases from 0.16 to 0.496 when a node intermittently acts malicious.



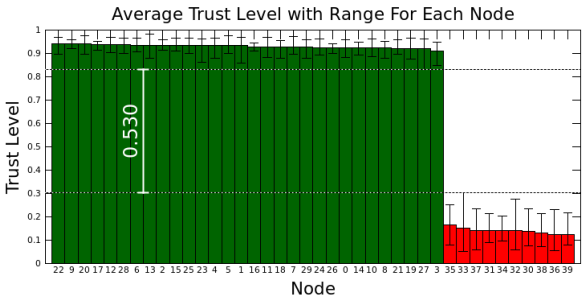
(a) Passive Modification: Percent of Good Nodes = 90%, $\alpha_i = 0.5$



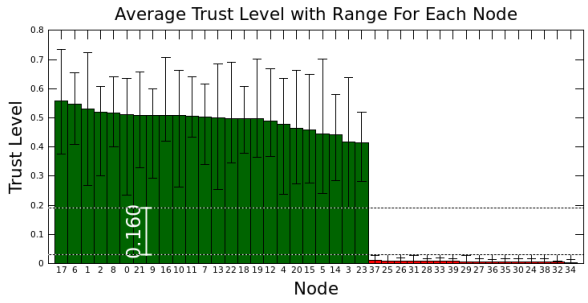
(b) Active Modification: Percent of Good Nodes = 90%, $\alpha_i = 0.5$



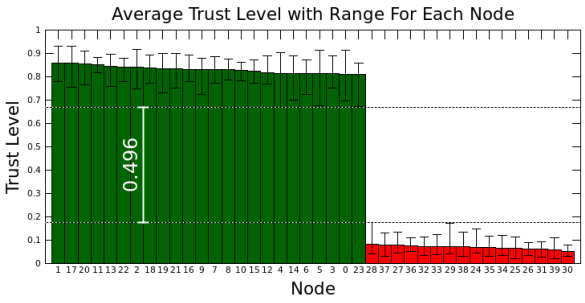
(c) Passive Modification: Percent of Good Nodes = 75%, $\alpha_i = 0.5$



(d) Active Modification: Percent of Good Nodes = 75%, $\alpha_i = 0.5$



(e) Passive Modification: Percent of Good Nodes = 60%, $\alpha_i = 0.5$



(f) Active Modification: Percent of Good Nodes = 60%, $\alpha_i = 0.5$

Fig. 3. Simulation Results: Percent of Good Nodes = {90%, 75%, 60%}, $\alpha_i = 0.5$

4) Conclusions on Variation of Bad Nodes: Our decision to include in the penalty and reward update values the current trust of reporting nodes led to a very interesting and encouraging phenomena. As shown in Figures 3b, 3d and 3f intermittently bad nodes fare better when there are more good nodes (90%) then less (75%, 60%). This is because good nodes have higher trust and their rewards for bad nodes behaving good is therefore higher. Fewer good nodes usually presents a problem; however, in our scheme, this makes the good nodes having slightly lower trust and therefore their rewards for bad nodes acting good are weaker forcing the bad nodes to converge to lower values. Hence, even in the network quite strongly compromised and with bad nodes trying to hide by behaving only intermittently bad, our scheme reliably differentiates between bad and good nodes.

We also checked and confirmed that nodes acting bad 25% of the time are detectable. Yet due to the distributed nature of trust, the difference between worst good and best bad becomes negative and the pronounce drop seen in Figure 3 is less apparent. As nodes reduce how often they cheat, the trust difference between good and bad nodes will eventually converge. A node hides more easily the more often it acts good, but its negative effect on the network diminish.

B. Matrix Versus Vector Comparison

Figure 4 shows an example where a fair coin is flipped to determine if a bad node acts maliciously. The matrix version and the vector approximation for indirect trust converge to similar values. As the percentage of good nodes increases, the difference between the two approaches decreases further. While this does not conclusively show that the vector approximation, with buffer space saving, performs statistically the same as the matrix version, it strongly suggests that it is and merits future evaluation in a broader range of environments.

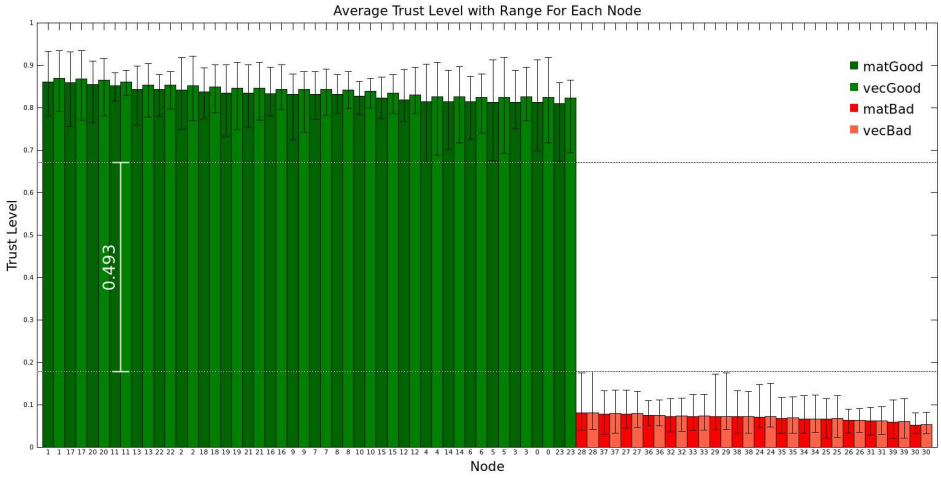


Fig. 4. Comparison Between Vector and Matrix, Percent Good = 60% (Bad Node Flips a Fair Coin to Determine Malicious Action)

V. Future Work and Conclusion

We introduce an integration of direct and indirect trust based on the directly observing information about paths using the trust management scheme proposed in [10]. There are multiple approaches for the fusion described above. The results in Section IV show the effect of α_i on the ability to identify faulty nodes. It appears to have a limited effect on the outcome of trust over a long time period; however, it might be important in more volatile networks or those with more robust threat models.

We also observed that the use of trust in rewards and penalties assigned to nodes via direct observation led to a stable differentiation between bad and good nodes even in cases of significantly compromised networks (in other words in networks with a high percentage of bad nodes even if those nodes try to hide their nature by intermittently behaving good and bad).

Additional research into the the affect of modifying α_a, α_i, β, γ, Δt and λ will help tune the integration for indirect and direct trust for range of attack models and network settings. The intuition is that the size of the network and number of malicious node will dictate their proper values. Expanding the threat model will also effect the tunable parameters and give insight into how the trust management scheme ports to real networks. A comparison to the trust management schemes proposed in [7]–[9] is merited.

The integration of direct and indirect trust is important since it defines how quickly does the trust converges to a state where nodes can identify bad actors helping to reduce their effect on the network. Additional work on integrating this scheme into a more robust routing and security scheme where, based on risk, nodes send messages only to those above a trust threshold is an important direction of our future work.

Acknowledgment

This work was supported in part by the Army Research Laboratory under Cooperative Agreement Numbers W911NF-06-3-0001 and W911NF-09-2-0053. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies either expressed or implied of the Army Research Laboratory or the U.S. Government.

References

[1] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Efficient routing in intermittently connected mobile networks: The multiple-copy case," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 1, pp. 77–90, 2008.

[2] T. A. Babbitt, C. Morrell, B. K. Szymanski, and J. W. Branch, "Self-selecting reliable paths for wireless sensor network routing," *Computer Communications*, vol. 31, no. 16, pp. 3799–3809, Oct. 2008.

[3] J.-H. Cho, A. Swami, and I.-R. Chen, "A Survey on Trust Management for Mobile Ad Hoc Networks," *Communications Surveys & Tutorials, IEEE*, vol. 13, no. 4, pp. 562–583, 2011.

[4] J. Sabater and C. Sierra, "Review on computational trust and reputation models," *Artificial Intelligence Review*, vol. 24, no. 1, pp. 33–60, 2005.

[5] K. Govindan and P. Mohapatra, "Trust Computations and Trust Dynamics in Mobile Adhoc Networks: A Survey," *Communications Surveys & Tutorials, IEEE*, vol. 14, no. 2, pp. 279–298, 2012.

[6] Y. Wang and M. P. Singh, "Formal trust model for multiagent systems," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, pp. 1551–1556.

[7] M. K. Denko, T. Sun, and I. Woungang, "Trust management in ubiquitous computing: A Bayesian approach," *Computer Communications*, vol. 34, no. 3, pp. 398–406, Mar. 2011.

[8] E. Ayday and F. Fekri, "An iterative algorithm for trust management and adversary detection for delay-tolerant networks," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 9, pp. 1514–1531, Sept 2012.

[9] I.-R. Chen, F. Bao, M. Chang, and J.-H. Cho, "Dynamic trust management for delay tolerant networks and its application to secure routing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 5, pp. 1200–1210, May 2014.

[10] T. A. Babbitt and B. K. Szymanski, "Trust management in delay tolerant networks utilizing erasure coding," in *IEEE ICC 2015 - Ad-hoc and Sensor Networking Symposium (ICC'15 (09) AHSN)*, London, United Kingdom, Jun. 2015.

[11] T. A. Babbitt and B. Szymanski, "Trust management in resource constraint networks," in *Proceedings of the 10th Annual Symposium on Information Assurance*, Jun 2015.

[12] K. Wehrle, M. Günes, and J. Gross, *Modeling and Tools for Network Simulation*. Springer Science & Business Media, 2010.