10th Annual Symposium on Iinformation Assurance (ASIA '15), June 2-3, 2015, Albany, NY, pp. 51-56.

Trust Management in Resource Constraint Networks

Thomas A. Babbitt and Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York
Email: {babbit, szymab}@rpi.edu

***Abstract*—There are numerous environments or situations where a computer network has one or more constraints. Impact of these constraints can range from minimal user inconveniences to catastrophic reduction of capabilities. These constraints allow availability of information at the cost of abandoning the safety and reliability of traditional routing and security protocols. A number of environments in which such networks are needed include those present in military battlefields, first responder missions or remote environments. Resource Constraint Networks (RCN) are a class of networks capable of working in such austere environments. Delay Tolerant, Wireless Sensor, and some mobile and mesh ad-hoc networks fall under the broader definition of RCN. In order to provide additional information assurance (IA) security services above information availability such as integrity, confidentiality, and authentication require significant modification to traditional routing and security protocols. One method is to manage trust in a distributed manner in order to make valid trust values available to a node. Using them, a node can then make decisions on how and when to forward a message through the network. While not traditional authentication, distributed trust can be used as a probabilistic proxy, and allow for more secure transmission of information from source to destination in a RCN. The use of path information in a DTN allows for malicious node detection. We show how using erasure coding, complete path information, and inferences assist in better identifying malicious nodes in a DTN.**

Keywords*—*Resource Constraint Networks, Delay Tolerant Networks, Distributed Trust Management, Discrete Event Simulations

I. Introduction

There are a number of scenarios where networks experience one or more significant constraints. This is either by design, due to the environment, or because of conditions changing in an environment. Military and first responders could be in situations where lack or destruction of communication infrastructure makes using traditional networking methods difficult or impossible. A Resource Contained Network (RCN) is any network with a resource constraint(s), such that significant modification of traditional Information Assurance (IA), security, or routing protocols are required to provide the security services of information availability, integrity, authentication, confidentiality, and non-repudiation. One example discussed in literature is Delay Tolerant Networks (DTN) where the nodes are mobile and are spread out in such a manner that static end-to-end routing is not possible and hence constrained. Another example is Wireless Sensor Networks (WSN) where nodes with minimal hardware are used to monitor environmental or other variables, the major constraint is battery power. It is challenging to provide IA security services of integrity, authentication, and confidentiality in a RCN.

While security and trust are not equivalent, in a DTN or WSN the use of centralized servers is not feasible. The ability to trust that a node is not compromised or acting selfish is paramount. Recently a number of studies on trust have been published that outline a framework for trust in a DTN or WSN. The authors in [1] give a good overview of trust in multiple disciplines and propose a number of trust characteristics and properties that can be used as a baseline for determining the clues or metrics used to construct a distributed trust management system. Table I shows the characteristics and properties outlined in [1].

Resource Constrained Networks require accurate and secure message transmission like any other type of network. As previously stated, there are a number of applications and situations where a RCN is necessary. Most of the routing and security protocols in RCNs use redundancy. The authors in [2] use redundancy as a means of determining clues for use in determining path probability. This in conjunction with the use of erasure coding show positive results in determining malicious nodes under a limited threat model, using only direct observations and directly connected nodes.

Recently a few trust management schemes for use in a DTN have been proposed [3]–[5]. Each of them relies on two trust components. The first is a direct trust consisting of clues or observable actions of other nodes and the second is an indirect trust component consisting of referrals, reputation, or recommendations from other nodes. These values are aggregated in a number of differing ways to determine a trust value that is used to identify nodes that act malicious or selfish. In [3] the authors use a Bayesian approach to determine probability that a node is acting good based on good and bad observations. In [4] the authors creates a bipartite graph and find outliers; the scheme removes nodes with probabilities outside of a certain value to converge on a trust value. A third approach outlined in [5] uses both direct and indirect metrics and determines good versus bad encounters over four categories to aggregate a trust value.

This paper advances the idea of using path, temporal, and information redundancy proposed in [2]. It shows the power of utilizing path information and introduces inferred trust properties. Further exploration of direct trust is discussed in Section II and indirect trust is presented in Section III. Conclusions and future work are discussed in Section IV.

TABLE I. TRUST CHARACTERISTICS AND PROPERTIES OF A DTN [1]

Characteristic	Properties
1) established based on potential risks	1) dynamic
2) context-dependent	2) subjective
3) based on nodes interest	3) not necessarily transitive
4) learned	4) asymmetric
5) may represent system reliability	5) context-dependent

II. DIRECT TRUST: USE OF FULL PATH

In a resource constraint network, a node directly observes network traffic within its transmitter/receiver range. Based on the actions of other nodes, direct observations can be used to make trust decisions. The authors in [2] use erasure coding as a routing protocol with an appended checksum to determine clues about a node or nodes that act maliciously by modifying all message segments received through normal network routing in a DTN and then making trust decisions. This approach only takes into account nodes directly connected to the destination. The results in [2] show that using path clues to identified malicious nodes has merit.

Fig. 1 presents a state diagram showing how a node processes each message M it receives. Before sending a message to the destination, the source will append a checksum to it and utilizing erasure coding will break a message into a number of segments s such that any $k < s$ segments will enable the destination to recreate the original message. The exact encoding algorithm is not relevant to this paper; however [6] presents a good overview of multiple encoding options in a DTN.

Assuming node i is the destination for message M, node i starts in state $S1$ and continues to track message segments m as they arrive. If m is unique, the segment is stored in node i's buffer and the message segment ID is saved in set n_M. If $m \in n_M$, then the **SegMatch** function is called. Once k unique segments arrive, node i attempts to recreate the message using the **SegRec** function. Node i then transitions to either state $S2$ if it fails or $S3$ if successful.

Once in state $S2$, node i continues to wait for additional segments m. If $m \in n_M$, then **SegMatch** is called, else **SegRec** is called. If **SegRec** is successful, then destination i transitions to state $S3$, else it determines if it is better to wait or resend the message from the source. The utility functions for waiting are presented in [2]. If it is better to re-transmit the message, then node i sends a message to node j to resend the message.

Most routing protocols in RCNs send an acknowledgement right after a successful delivery to the destination. This clears node buffers and avoids wasting resources to send a message or segment through the network once it is successfully delivered. Taking advantage of path and temporal redundancy node i stays in state $S3$ for a time period and accepts additional segments m. It continues to recreate the message using $k-1$ known good segments and makes trust decisions based on the success of the message recreation.

The **SegRec** function iterates through all permutations of message segments received for M and returns those that successfully recreate the message. When the number of segments received is k that requires one check. Once $|n_M| > k$ then

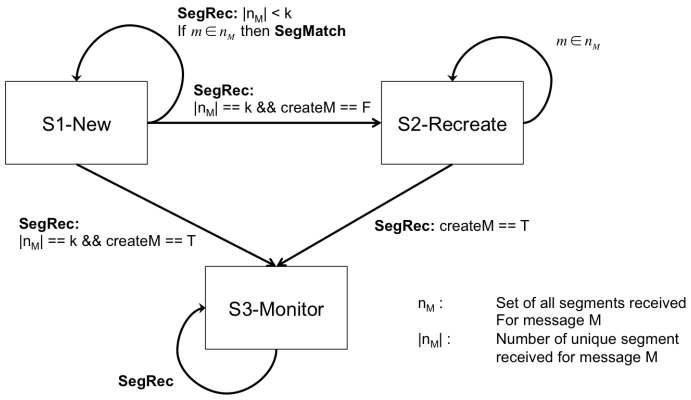


Fig. 1. Node State Diagram

$\binom{|n_M|}{k}$ iterations are required. This is the basis for the utility functions mentioned above for state $S2$.

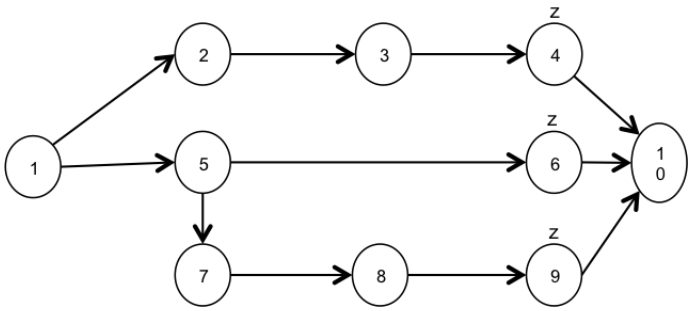
A number of research questions remain. This section presents two additional concepts using direct observations to make trust decisions. The first is to utilize full path knowledge and the second is to observe differences of the same message segments along different paths (**SegMatch**).

A. Expanded Path Information

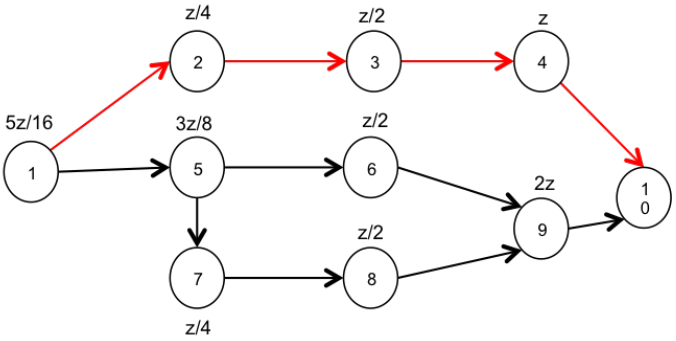
Nodes make trust decisions in [2] based on the directly connected nodes. Only the nodes that send each segment to the destination or an intermediate node with enough segments to recreate M make trust modifications. Fig. 2a shows an example of this. Assume that the source is node 1 and the destination is node 10 and the number of segments required to recreate a message M is $k = 3$. Each of the three required segments take a different path, namely $\{1, 2, 3, 4, 10\}$, $\{1, 5, 6, 10\}$ and $\{1, 5, 7, 8, 9, 10\}$. In this case, node 10 only increases the trust level for nodes 4, 6 and 9 even though six other nodes were along the paths that the segments traveled.

Each node along a path appends its id to each segment as it flows through the network. Assuming nodes act truthfully, the destination will have the path for each segment. Using Fig. 1, upon transitioning to state $S3$ trust is distributed along the paths used to recreate the message. Let's denote the trust change to be calculated as z. Fig. 2b shows an example of this distribution. Each directly connected node receives z increase and then divides that value by 2^h, where h is the number of hops back from the destination along a given path. If there is a situation where a node, directly connected to the destination, is along n_{path} multiple paths, such as node 9, then the trust increase is $z \times n_{path}$. In the example node 3 receives $\frac{z}{2}$ and node 5 receives $\frac{3z}{8}$ because it is along two paths one at hop 3 and the other at hop 4 from the destination.

Additionally negative trust can be distributed back along a path. Assume node 10 is in state $S3$, for message M, and waiting for additional segments to make trust decisions. A segment of message M arrives using the red path in Fig. 2b. Negative trust can be distributed back along the path $\{1, 2, 3, 4, 10\}$. Other than the source, node 1, receiving $\frac{z}{8}$ all other values will be the same.



(a) Path Trust Distribution in [2]



(b) Updated Trust Distribution

Fig. 2. Trust Distribution Changes

B. Trust Updates Using Segment Matching (*SegMatch*)

Fig. 1 shows all the states that a node goes through for each message M. When a node is in state $S1$ or $S2$ and $m \in n_M$, signifying that the node has seen m before, **SegMatch** is called. This is done to compare the paths that the two identical segments took to arrive at a particular node.

In Fig. 3, node 1 is the source and node 7 is the destination that received m along multiple paths. The paths segment m followed are $\{1, 2, 5\}$ and $\{1, 3, 4\}$. Assume that at any given time there is a set of "trusted" nodes consisting of all nodes above a certain threshold and designating this as set A. In Fig. 3 all of the green nodes are above that threshold so $A = \{1, 2, 4, 6\}$. The set of all node along the paths m took is $B = \{1, 2, 3, 4, 5\}$. The set of suspect nodes $C = B - A = \{3, 5\}$.

When two segments m for message M arrive at node i with the same id along different paths the payloads of the m either match or not. If they do not match then some small trust deduction is merited; if they are the same then a small increase is merited. Equation 1 is the trust reduction function for nodes in set C. Because a nodes trust can fluctuate, we consider all nodes suspect and assume $C = B$.

Node i reduces trust for each $j \in C$; the new trust $NT_{i,j}$ is the current trust $CT_{i,j}$ minus a small penalty that consists of three parts. The first part $(1 - CT_{i,j})$ links the penalty to node i's current trust level for node j. If a node is more trustworthy it receives a smaller penalty. The second part $\frac{z}{a}$, where a is the number of elements in C, divides the penalty z by the number of possible culprits. The more there are, the more ambiguity, so the smaller the penalty. The final part $\left(1 + (1 - p_{nx})^{|B|}\right)$ again takes into account the number of nodes. The value for $\left(1 + (1 - p_{nx})^{|B|}\right)$ is the probability that a certain number of nodes are all bad. This takes into account both the size of the network and the current trust average for the network, p_{nx}.

$$NT_{i,j} = CT_{i,j} - \left((1 - CT_{i,j}) \times \frac{z}{a} \times \left(1 + (1 - p_{nx})^{|B|}\right)\right) \quad (1)$$

Equation 2 shows the trust increase when message segments along both paths are the same. Equation 2 is the result of solving Equation 1 for $CT_{i,j}$ and then substituting $NT_{i,j}$ for $CT_{i,j}$ and vice-versa. This makes it so that events that happen in sequence one good and one bad with no other changes result in the trust value of a node remaining the same as it was prior to both events.

$$NT_{i,j} = \frac{CT_{i,j} + \frac{z}{a}\left(1 + (1 - p_{nx})^{|B|}\right)}{1 + \frac{z}{a}\left(1 + (1 - p_{nx})^{|B|}\right)} \quad (2)$$

C. Simulation Results for Direct Trust

A number of simulations were conducted utilizing NS3 and the DTN module proposed by *Lakkokorpi et al.* in [7] and the node trust management object *DtnTrust* proposed in [2]. Modifications to the *DtnTrust* object include tracking the path of each message segment, distributing trust along paths, and making incremental updates when the message segments have the same id but arrived along different paths.

The simulations run in [2] are rerun here with the modifications listed above. A total of 40 nodes are randomly placed on a 2500m x 2500m grid and move utilizing a random way point (RWP) mobility model. Each node sends multiple various size message segments using erasure coding. The average of ten 1000 second simulation runs is used. The same random seeds are used for both set of results.

Fig. 4 and 5 show the power of full path knowledge. Fig. 4 shows results with the fraction of nodes that act truthfully set to 90% and Fig. 5 shows results with this fraction set to 60%. Subfigures b show the results with full path knowledge, while Subfigures a without. There is a pronounced difference

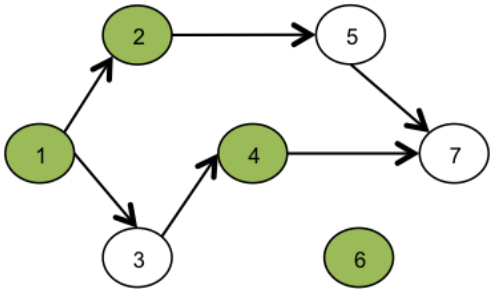
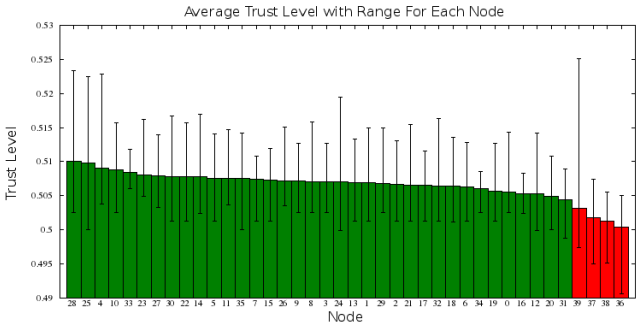
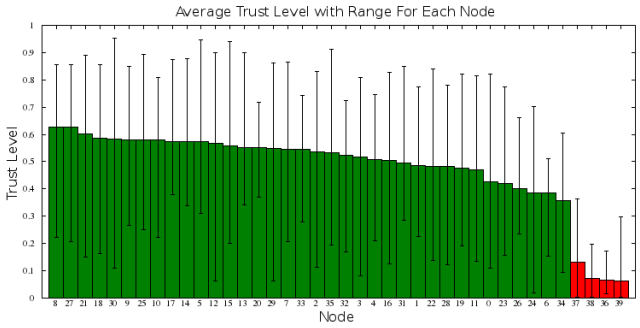


Fig. 3. SegMatch Example

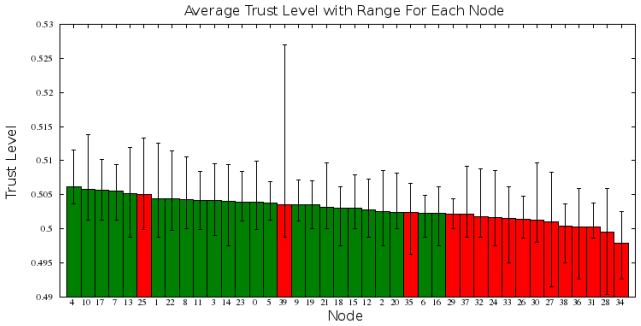


(a) 90% Path Trustworthiness as per [2]

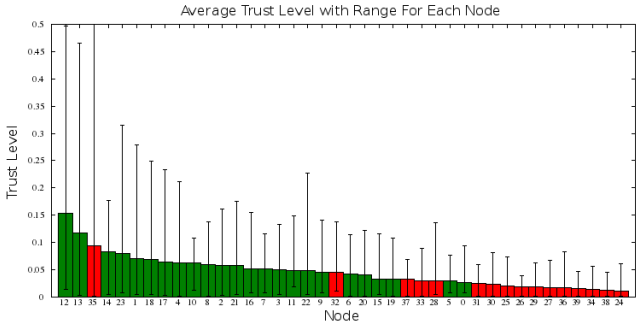


(b) 90% Path Trustworthiness with Modification

Fig. 4. Ranking of All Nodes According to Their Trustworthiness $p_{nx} = 0.9$



(a) 60% Path Trustworthiness as per [2]



(b) 60% Path Trustworthiness with Modification

Fig. 5. Ranking of All Nodes According to Their Trustworthiness $p_{nx} = 0.9$

between trust values for good and bad nodes in Fig. 4b vs. 4a. Higher pollution by bad nodes makes the results less clear when the fraction of good nodes decreases. In future research, we plan to exclude recognized bad nodes from being used in routing which would help to prevent this effect. We will also investigate stronger trust increases for nodes behaving correctly.

III. Inferred Trust

Nodes can infer trust based on interactions. Fig. 6 shows nodes i and j. Assuming that they are within broadcast range and have sufficient time to transmit, they first trade their trust information about other nodes in the network. This is used to update the trust levels for both nodes. In this example, node i sends its trust vector (shaded green), to node j and receives node j's trust vector in return. Node i maintains a $(n+1) \times n$ inferred trust matrix that consists of trust vectors received from other nodes with an appended time stamp; the trust vector received from node i is shaded red. Each entry in node i's inferred trust matrix is a value between 0.0 and 1.0. A node will state that it trusts itself at level 1.0 so the diagonal is all 1's. The column for i in the inferred trust matrix (in yellow) is the inferred trust vector. In addition there are two $(n+1) \times 1$ vectors. The direct trust vector (in blue) is updated by direct trust observations made by node i with a time stamp. The aggregate trust vector (in green) is the combination of the inferred and direct trust vectors. Any trust decision made by node i is done based on the trust values in aggregate trust vector (green).

The inferred trust vector is maintained through two processes. The first is a periodic update based on the inferred trust matrix. After a certain time period or number of encounters the vector is updated. See Section III-A for more details. The second process occurs each time a node receives the trust vector from another node in the network. See Section III-B.

A. Periodic Updates to Inferred Trust Vector

Equation 3 finds the difference between what node i has stored in its trust vector and what node j broadcasts as its trust vector. The difference is determined for each node $w \in N$, where N is the set of all nodes in the network. This value is multiplied by the square of the node i's trust of node j giving more weight to trusted nodes.

$$C_w^i = \left(T_j^i\right)^2 \times \left(T_w^i - T_w^j\right) \tag{3}$$

The difference is not immediately used to update node i's trust vector, but is maintained for a time interval Δt in the inferred trust matrix. Once the time interval is complete, the set of nodes from which node i received trust vectors D, is averaged and node i updates it inferred trust vector using the

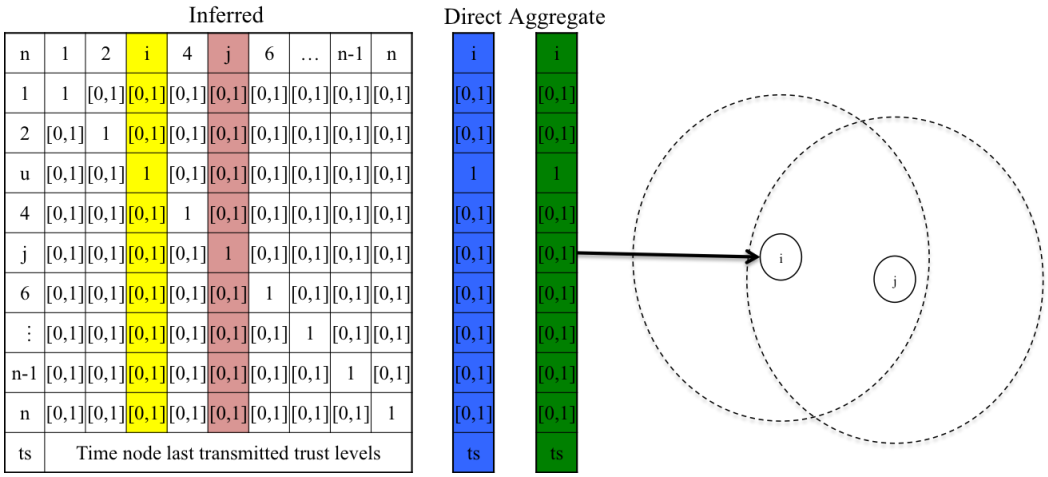


Fig. 6. Storage at node i

following equation.

$$T_w^i = T_w^i + \beta \left(\frac{\sum_{f=1}^{|D|} C_w^f}{|D|} \right) \tag{4}$$

B. Direct Updates to Inferred Trust Vector

Node i updates trust based on any discrepancy between its aggregate trust vector and the trust vector sent by node j. There are two cases when node i receives the trust vector from node j. Case 1 is that all values in the inferred trust vector for node i are within τ of all values in the trust vector from node j ($| \left(T_w^i - T_w^j\right) | < \tau$ for all $w \in N$). Case 2 occurs if one or more of such values are not within τ. Case 1 results is a small increase of trust for node j in node i's inferred trust matrix. For case 2, trust is decreased for all w where $| \left(T_w^i - T_w^j\right) | \geq \tau$. Equation 5 defines the change in trust for node i and Equation 6 prescribes the change in trust for all other nodes that are outside τ.

$$T_j^i = T_j^i \times \left(1 - \frac{\alpha \times d}{2\left(|N| - 2\right)}\right) \tag{5}$$

$$T_w^i = T_w^i \times \left(1 - \frac{\alpha}{2d}\right) \tag{6}$$

For the updated set of equations, there are four tunable parameters, α, β, Δt and τ.

1) α - This is the penalty weight in Equations 5 and 6. The variable d is the number of discrepancies. If $d = 0$ then no nodes get a penalty and the equations are not used.
2) β - This is the weight given to the indirect observation and it is a value between 0.0 and 1.0.
3) Δt - This is the time period the nodes waits between trust updates.
4) τ - This is the risk taken, it represents the acceptable difference between node i and j's declared trust for node w before node i makes trust changes.

C. Simulation Results for Indirect Trust

In order to test some of the tunable parameters, a simulation engine is proposed that acts as a discrete event simulator [8]. A complete graph, is created using the number of nodes n in a given network. Each edge weight is a random number uniformly distributed between [0.0,1.0) and it represents the intermeeting time between nodes connected by this edge. If the edge weight is 1.0 then the nodes are in constant contact and if it is 0.0 they never meet. This simulates an arbitrary mobility pattern in the network.

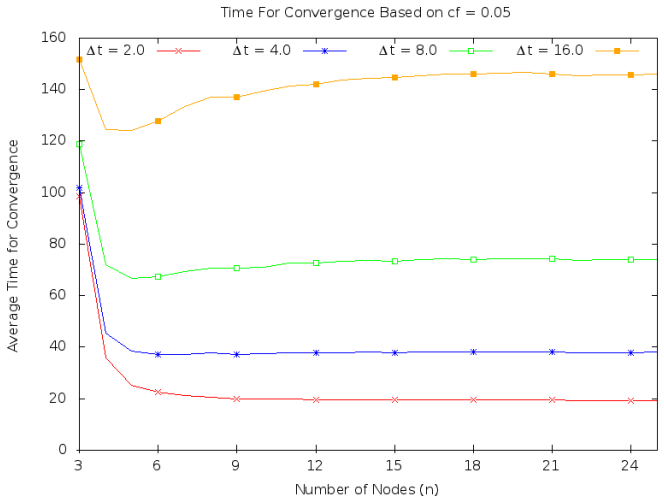
Internal events are those that are node driven and external events are those driven by the interaction between nodes. The former are based primarily on node attributes and the later on edge weights. The only internal event, used for this set of simulations, is a node trust update associated with the Δt value. If node i's timer expires, set to Δt, it triggers a node trust update event using Equation 4.

The external events are the node meeting events. They are derived from the intermeeting time between nodes. The next meeting is based on a Poisson distribution (assumed here and in other publications to be the distribution for intermeeting times between nodes) using the following equation.

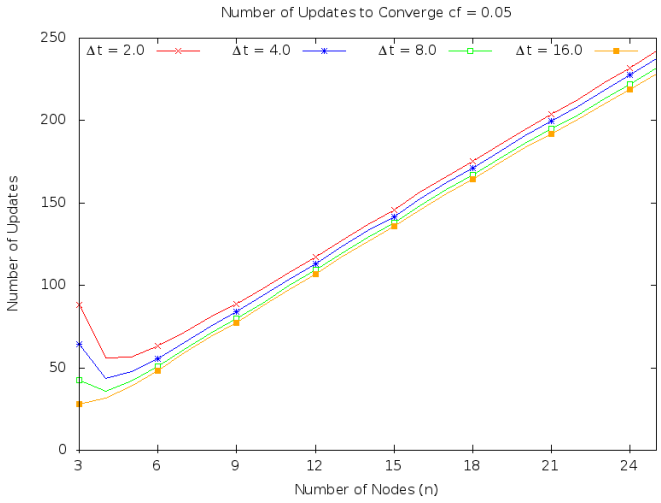
$$mTime_{i,j} += -\left(\frac{1}{w_{i,j}} \times ln([0,1])\right) \tag{7}$$

Where $mTime_{i,j}$ is the current meeting time between node i and node j, initialized prior to each run using the right hand side of Equation 7. The value $w_{i,j}$ is the inverse of the intermeeting time which is also the weight given to each edge in the complete graph.

Each simulation consists of 1000 runs for each $3 \leq n \leq 50$ nodes and returns the average time to converge and the number of node updates that occur. Each node also maintains the last time it updated its trust initialized at the beginning uniformly over the interval [0.0,Δt), its inferred trust vector initialized with values uniformly distributed between [0.0.1.0), and inferred trust matrix with all zeros. For each run of the simulation, each event is taken in order and follows the rules above depending on the event type. The run ends when either



(a) Time to Converge, $cf = 0.05$



(b) Number of Updates, $cf = 0.05$

Fig. 7. Effect of Δt

10,000 time units expire or all nodes have converged to the same trust values plus or minus a convergence factor, cf, from the average.

Fig. 7 shows a couple of key points that justify further research. The first is that the system converges rather quickly. Fig. 7a shows the exact times with $cf = 0.05$ for multiple values of Δt. The second key point is that after the network size reaching $n \approx 20$ the time levels off for all Δt. The third is that the shorter the Δt the faster the network converges, as intuitively expected. The fourth is that the number of updates increases in a linear fashion as n increase for all Δt, as seen in Fig. 7b.

IV. Future Works and Conclusions

The results above show the improvement arising from the use of full path knowledge. The potential for further improvements from using inferred trust is one of the goals of future research. Some additional improvements include integration of the direct and inferred clues, exploration of additional direct and confirmation of inferred clues, threat model expansion and comparisons with other proposed trust management schemes [3]–[5].

The use of complete paths as a clue advances the results in [2]. Research into additional clues and the best methods to identify potential cheaters as well as the effect on node resources should better identify limitations. Indirect or inferred clues add overhead that must be considered.

The fundamental issue for trust management is to converge to a state in which the bad nodes can be identified. In the paper, the assumed threat model was that the bad node would not try to hide, so the longer the scheme runs the more differences between bad and good nodes arise. With sophisticated adversaries, which switch between bad and good behavior, it will be important that a good period does not erase all sins from the past. We believe that our use of the current trust in assigning penalties and rewards is a good way to address this issue, but the right weights need to be established experimentally. This will be 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] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Y. Wang, S. Jain, M. Martonosi, and K. Fall, "Erasure-coding based routing for opportunistic networks," in *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, ser. WDTN '05. New York, NY, USA: ACM, 2005, pp. 229–236.

[7] J. Lakkakorpi and P. Ginzboorg, "ns-3 module for routing and congestion control studies in mobile opportunistic dtns," in *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2013 International Symposium on*, July 2013, pp. 46–50.

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