INFORMATION ASSURANCE IN RESOURCE CONSTRAINT NETWORKS

By

Thomas A. Babbitt

A Dissertation Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

_____
Boleslaw K. Szymanski, Dissertation Adviser

_____
Bülent Yener, Member

_____
Sibel Adalı, Member

_____
Koushik Kar, Member

Rensselaer Polytechnic Institute
Troy, New York

February 2016
(For Graduation May 2016)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

There are more individuals than I can possibly acknowledge and thank that have had a positive effect on my life and, in some small way, contributed to this dissertation. I cannot articulate how much the guidance, teaching, discipline and love shown by my parents Richard and Joanne Babbitt have molded me and given me the basic tools for success. I would like to thank the multiple mentors, too numerous to list, that have guided, taught and shown me an example of success both in the military and academia. I could not have finished this without the help of my committee and specifically my academic adviser Boleslaw Szymanski. He always kept my academic fire within the proper range fan helping to ensure minimal scope changes, innovative ideas and quality work. Most importantly, I would like to express my love and eternal thanks to my exquisite wife Bridget and my wonderful children Grace, Margaret, Eleanor, and Thomas. Your love supported me throughout this and all of my previous jobs and deployments. This is a testament to your love, confidence, support and patience. Thank You.

ABSTRACT

People are connected through a network of friends and acquaintances, most using multiple electronic devices to foster those relationships. First responders and the military work in chaotic environments with the potential for disjointed or destroyed communication infrastructure and must have the capability to establish ad-hoc networks in remote areas. In social, disaster relief, military situations, and sensor networks, there is a growing need for a class of Resource Constraint Networks (RCN). A RCN is a network where some node or network resource constrains the use of traditional Information Assurance (IA) protocols or practices. This can leave security vulnerabilities that are exploitable for nefarious purposes. A Wireless Sensor Network (WSN) is an example of a RCN where the battery power on a node is limited. In many WSNs, nodes communicate using a half-duplex system with only one transmitter/receiver in order to reduce hardware battery consumption. This, along with small buffers, restricts routing information stored locally and affects information availability. Processing is at a premium as each computation draws from the limited battery power, making many traditional encryption schemes costly. A second example of a RCN is a Delay Tolerant Network (DTN) where nodes are connected intermittently and ad-hoc. The connection between nodes is ever shifting as they move. The primary network constraint in a DTN is a lack of end-to-end routing knowledge. In addition to making routing challenging, the use of centralized servers is difficult if not impossible. In both a WSN and a DTN, determining trust between nodes is not trivial.

In all networks, security and privacy of data as it flows from source to destination is paramount. Information Assurance (IA) has not been properly addressed in RCNs. This dissertation will focus on five key IA services: availability, integrity, confidentiality, authentication, and non-repudiation. Due to the nature of a RCN, it is challenging to provide IA services. Most of the research in RCNs has focused on availability by proposing schemes to efficiently move data packets while minimizing delay and device resources such as buffer space, battery power, and processing.

While this research is important, the resulting increase can quickly be lost if the other IA services such as integrity or confidentiality are not maintained. This dissertation explores challenges associated with IA in RCNs and proposes a trust management scheme to defend by exploring clues for use in distributed trust management, integration of direct and indirect clues to create a distributed trust management scheme, and introduces a Trust Based Secure Routing (TBSR) protocol for use in a Delay Tolerant Network.

CHAPTER 1
Introduction

For many people, the idea of not being able to use a cell phone to make calls or interact with social media is a foreign concept. There are many situations where this robust communication is either not present, controlled by an oppressive government, or susceptible to attack or natural disaster. What happens when too many individuals gather to protest or attend events and they either overwhelm, or a hostile government shuts down, cell towers? This occurred in Hong Kong where protesters gathered and overwhelmed cell towers. A number of mobile phone applications that create an ad-hoc mesh network facilitated the ability to chat. One example of such application is discussed in [1], which allowed for text and images but maintained little to no security features. Other than shared usernames, an individual using the application could not tell who a message came from, the validity of the information or who else was able to read it; in fact the organizers of the protest explicitly warned attendees to use aliases since obscuring who was posting to the chat room or sending a message was the only way to provide any security.

There are military, emergency response, natural disaster, and a number of other uses for networks that securely perform with limited resources. Delay Tolerant (DTN), Wireless Sensor (WSN), Mobile Ad-Hoc (MANET), ad-hoc mesh, and Vehicular Ad-Hoc (VANET) networks are all examples of a network with a limiting constraint or constraints. These classes of Resource Constraint Networks (RCN) must provide some level of Information Assurance (IA). Having an adversary modify, block, or even just be able to overhear communications is potentially catastrophic. For the military, this could lead to a failed mission. In an emergency response, this could cost loss of life or expenditure of unnecessary resources that could save someone else. For a protest, this could result in the arrest or mistreatment of key leaders.

There are a number of different information assurance models and terms that frame security problems in a network. The authors in [2] propose a comprehen-

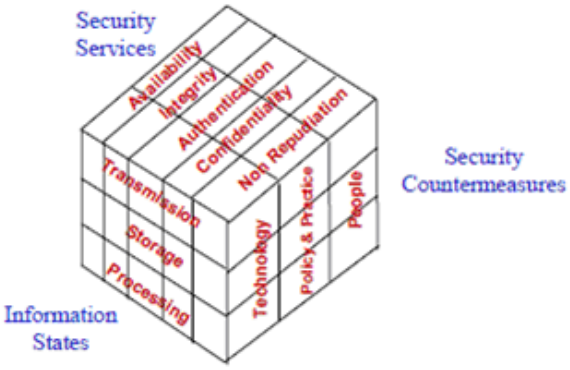


Figure 1.1: Information Assurance Model

sive model with three foundations: information state, security countermeasures and security services. Figure 1.1 [2, Fig. 1] visually depicts this model. For security services, the model explicitly focuses on information availability, integrity, authentication, confidentiality, and non-repudiation. While a valid argument can be made to include other security services or use different terms [3], the five listed above focus on making information available to an application/user in a secure manner.

Providing IA to different classes of RCN is a challenge. To ensure definition continuity a RCN is any network with a resource constraint(s), such that significant modification of traditional IA, security, or routing protocols are required to provide the security services of information availability, integrity, authentication, confidentiality, and non-repudiation (all key terms and variables are defined in Chapter 2). Of course this definition is very broad, it encompasses WSN, MANET, DTN, VANET, and some ad-hoc mesh networks. A DTN, as an example, is broadly defined as a mobile ad-hoc network where nodes are likely to move and have intermittent connectivity with other nodes in the network such that no end-to-end routing tables are maintained and that the applications on the network are able to tolerate some level of delay; essentially trading higher quality of service (delivery rate) for increased end-to-end delay.

To further illustrate the IA challenges in RCNs and using a DTN as an example, there are numerous routing protocol proposals to ensure end-to-end transmission (information availability). Most DTN routing protocols follow a store, carry, and

forward approach. When two nodes meet, each node has one of two choices for each message stored in its buffer, namely, forward or wait. While flooding the network might seem the best approach to ensure delivery, the cost in battery, storage, processing and other node overhead is steep. Most protocols use a quota on the number of replications of a packet such as spray-and-wait [4]. Much of the recent work on DTN network protocols focuses on social-based routing. This concept is valid because the understanding of how individuals interact and move gives insight into when the devices they carry will likely be in range and can transfer data packets [5]. Whatever the approach to determining how many copies of a message or to which node to forward a message, some measure of trust is required by the node forwarding to the node receiving the message that it will follow security and routing protocols. In the social based schemes mentioned above, if a node lies and states it is closer to the destination, it is more likely to receive one of the limited copies of a message.

Continuing with the DTN example, authentication and by extension confidentiality, integrity and non-repudiation are challenging without key management. The ability to provide and check revocation status of signatures is usually managed centrally. Some operations can be done with a shared key, but one compromised node compromises the whole network. Additional work on identity based security [6] might assist in providing confidentiality and integrity. Due to topology changes over time and the constraints in many RCN classes, to include DTN, any use of a client server architecture is limited. The use of a valid trust value can allow for probabilistic authentication.

There are a number of routing and security protocols proposed for use in a Delay Tolerant Network, which are discussed throughout this dissertation. They attempt to limit exposure of a message, and/or make routing decisions, based on some trust component. This dissertation will focus on the use of trust management in Recourse Constraint Networks specifically looking at DTNs, with proposals for modification to other network classes, and secure routing. This will use the IA model in [2].

1.1 Research Problems

The main focus of this dissertation is on utilizing trust to make secure routing decisions in a Delay Tolerant Network. Specifically chosen were three interrelated research problems which explore that focus. The first is the selection of direct observation clues for use in distributed trust management. The second problem is the fusion of direct and indirect observations to best converge on a valid trust value. The third issue is the use of a valid trust value to make routing and security decisions.

Trust management in RCNs and DTNs must take into account multiple trust properties including the fact that trust is dynamic, subjective, context dependent, asymmetric, and not necessarily transitive [7]. Because of the aforementioned trust properties, distributed trust management and secure routing in a RCN is a challenging problem. The three main research topics are listed below.

1.1.1 Use of Redundancy to Determine Direct Trust Clues

A node in a DTN can only observe the actions of another node within the constraints of it's hardware. This limits any selected node in the DTN to broadcasts it can overhear or to which it is a party (sender or receiver of a message). Some of these observations can be used to determine if another node is acting trustworthy. An example could be if node i observes node j resend a message with the same message i.d. but a different data payload. Normally this is done through the use of some form of redundancy, a good overview of which is found in [8]. In the simple example listed above, that would be using redundancy of the same message. This concept directly led to the exploration of path redundancy in a DTN and the idea to use a modified erasure coding routing protocol to send messages and observe any modifications to those messages. This is shown to be an effective clue in determining untrustworthy nodes [9].

1.1.2 Fusion of Direct and Indirect Trust

Not all nodes in a DTN regularly interact. There are isolated nodes and for any given node i there is a set of nodes with which it will rarely ever come in

contact. With a lack of direct observations on a subset of nodes, how can a node maintain a valid trust value? This explores the fusion of direct observations with recommendations received from other nodes. This shows that by properly fusing direct and indirect trust the convergence time decreases and leads to more accurate trust values when nodes are intermittently untrustworthy [10].

1.1.3 Secure Routing Using Trust Management

This researches the use of trust to make secure routing decisions. If each node maintains a valid trust value between 0.0 and 1.0, can it securely route a message from source to destination? Utilizing the distributed trust management results, decisions of what nodes to forward to or "blacklist" are made. There are multiple approaches and those are explored. This shows how making routing decisions based on trust reduces the number of messages sent to untrustworthy nodes and minimizes the increase in delay time as part of the trade off between security and energy use versus message delay and delivery rate [11].

1.2 Dissertation Outline

The remainder of this dissertation follows the chapter outline below.

- **Chapter 2: Key Terms, Definitions and Variables:** This chapter defines all key terms and variable.
- **Chapter 3: Resource Constraint Network Background:** This chapter provides background and some previous work done in Resource Constraint Networks.
- **Chapter 4: Direct Observations through Redundancy:** This chapter discusses in detail direct clue observations for use in determining trust in a Delay Tolerant Network.
- **Chapter 5: Fusion of Direct and Indirect Trust:** This chapter explores the fusion of direct and indirect trust into an aggregate trust value suitable to making secure routing decisions.

- **Chapter 6: Trust Based Secure Routing:** This chapter utilizes trust to make secure routing decisions.
- **Chapter 7: Conclusions and Future Work:** This summarizes the research contributions of this dissertation, proposes future research directions, and outlines any research challenges.

CHAPTER 2
Key Terms, Definitions and Variables

The problems addressed in this dissertation focus on Information Assurance (IA) in Resource Constraint Networks (RCN) and providing secure message transmission from source to destination node. There are many key terms used throughout this text. This chapter outlines those key terms, definitions and variables. The reference from where the definition is taken is cited. In some instances, there is additional discussion about the definition in reference to one or more classes of RCN. If there is no reference next to a particular definition, it is one I formulated to ensure continuity throughout. This dissertation references some papers that use older definitions. An example is in [2] that uses IA terms from the older National Security Agency publications [12], [13] that were superseded by The National Institute of Standards and Technology (NIST) Special Publication 800-53, Revision 4 [14]. Definitions from the new government publication are used when there are differences.

2.1 Key Terms and Defintions

2.1.1 RCN Classes

A Resource Constraint Network is any network with a resource constraint(s), such that significant modification of traditional IA, security, or routing protocols are required to provide the security services of information availability, integrity, authentication, confidentiality, and non-repudiation. There are a number of classes of RCN to include Delay Tolerant, Wireless Sensor, Mobile Ad-Hoc, Ad-Hoc Mesh, and Vehicular networks. Due to the majority of the dissertation's focus on DTN, it has the most rigorous definition. All other network classes are more briefly defined.

2.1.1.1 Delay Tolerant Networks (DTN)

A DTN is broadly defined as a network where nodes are likely to move and have intermittent connectivity with the network, such that no end-to-end routing tables are maintained, and that the applications on the network are able to tolerate

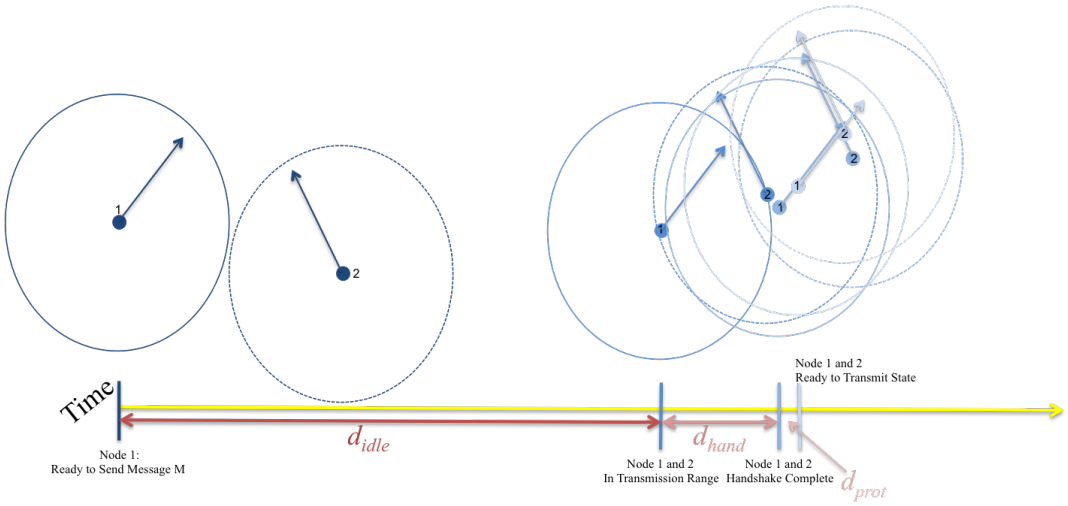


Figure 2.1: Discovery Delay d_{disc}

some level of delay. This trades higher quality of service for increased end-to-end delay. To formalize, a DTN is a wireless computer network in which:

1. Nodes have integrated transmitters and receivers to transfer data. They can use established protocols such at 802.11 or 802.15.

2. Nodes are mobile.

3. No established end-to-end routing table is maintained. This is due to the overhead to maintain forwarding tables or the sparseness of nodes causing significant delay.

4. An application running on node k, requiring network connectivity, is able to withstand some total end-to-end delay $\Delta k > d_{total}$, where Δk is the application delay threshold and d_{total} is the end-to-end delay for a message M. The value for Δk can be days for certain applications that transmit data long distances through space with a limited number of nodes.

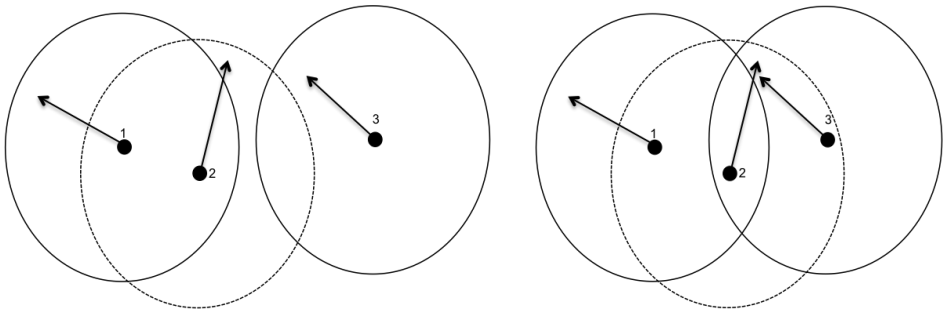
5. The delay at each node is d_{nodal}, which consists of the traditional network node delays of propagation, transmission, queuing, and processing. In addition, a DTN specific discovery delay is necessary. The total $d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop} + d_{disc}$.

6. The discovery delay, d_{disc}, consists of a number of elements which include delays for handshake, idle time, and routing protocol computation. Each delay is described in detail below.

For a particular node, there are a number of types of delay. The processing, queuing, transmission, and propagation are calculated similar to any other computer network [15]. The processing is the time it takes to calculate any checksum and read the header. The transmission is the amount of time it takes to transmit the bits through the transmitter and is L/R, where L is the length of the message and R is the speed at which the transmitter sends one bit. The queuing delay is the amount of time a message M has to wait while other messages are forwarded. The propagation delay is the distance the signal travels divided by the speed of the transportation medium. All four of the delays are hardware dependent, but relatively easy to calculate given hardware specifications.

The additional DTN discovery delay provides more of a challenge and consists of a delay for handshake, idle time, and routing protocol computation $d_{disc} = d_{handshake} + d_{idle} + d_{prot}$. Figure 2.1 illustrates this delay over time. The idle delay, d_{idle}, is the time it takes for two nodes to move into transmission range. The handshake delay, d_{hand}, is the time it takes to create a connection between two nodes. This consists minimally of transmitting hello messages and establishing a connection between the nodes; this is protocol dependent, but if WiFi or bluetooth are being used, the messages sent to establish a connection are standard. The protocol delay, d_{prot}, is the time it takes to determine if a given node is closer to the destination and if there are any packets to forward.

Figure 2.2 shows two separate examples to better explain d_{disc}. The first case shown in figure 2.2a occurs when node 1 and 2 transfer a message M and node 3 is the next hop based on routing protocol. Since node 2 and 3 are not in transmitting range, a delay d_{idle}, occurs until the nodes are within range to communicate. Once the nodes are within range, there is an additional delay, $d_{handshake}$, while the nodes conduct the protocol specific handshake. The final delay, d_{prot}, is the delay while the node determines, based on routing protocol, if the newly connected node is closer to the destination. The second case shown in Figure 2.2b occurs when node 1 and node



(a) With $d_{handshake} + d_{idle} + d_{prot}$ **(b) With only d_{prot}**

Figure 2.2: Nodes Movement

2 are in transmitting range and node 2 and node 3 are in transmitting range. In that case $d_{idle} + d_{handshake} = 0$ and the only additional delay is the processing delay required to check if node 3 is closer to the destination. In this scenario, the delay mimics a traditional network with some minor additional processing delay. In both cases, it is assumed that node 3 is closer to the destination and that node 2 would forward M; however, if that is not the case, then node 2 must wait for another node increasing d_{idle}.

The delay that is likely to have the biggest variance and be the largest is d_{idle}. In certain space based DTNs, it could be days until nodes are within transmitting range. This makes using centralized authentication, trust management, and encryption schemes challenging. Given sufficient battery power, and an assumption that in most cases $d_{idle} > 0$, nodes can use that time to process distributed authentication, trust management and encryption schemes.

2.1.1.2 Wireless Sensor Networks (WSN)

There are multiple definitions for a WSN. The authors in [16] present a survey of WSNs and define one as a network consisting of sensor nodes that are densely deployed. The authors in [17] state that the nodes are limited and that they work together to gather data in the environment in which they are deployed. The difference between an ad-hoc network and sensor network is based on hardware and routing. Sensor nodes are limited in hardware (buffers, processor power, battery, and transmission range), more densely deployed, prone to failure and use broadcast communication protocols such as SHR [18], [19] as opposed to point-to-point such

as TCP/IP [15]. This has led to many applications such as military [20], natural disaster [21] or underwater [22].

2.1.1.3 Mobile Ad-Hoc Networks (MANET)

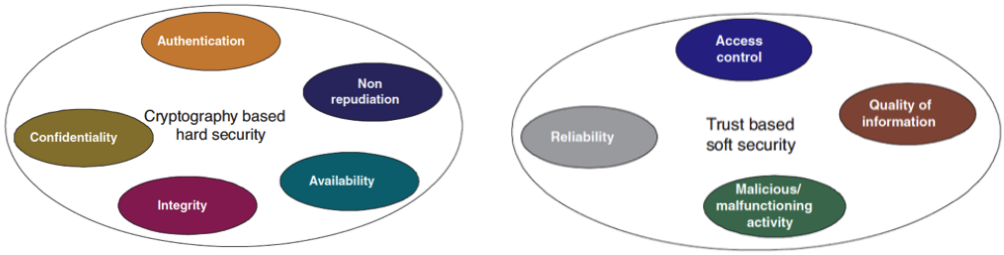
There are two types of wireless mobile networks according to [23]. The first is infrastructure based. An example of which is a mobile cell network. In such networks, there is a large number of cellular devices that are mobile and when carried by individuals interact with a set of deployed cell towers. When a phone moves and loses the signal of one tower and gains signal from another, the node switches communication between towers. The second is an infrastructure-less or ad-hoc network. A Mobile Ad-Hoc Network (MANET) consists of a number of nodes that are mobile and configure themselves to communicate as they interact with other nodes. There are a number of routing protocols such as Ad-Hoc On-Demand Distance Vector (AODV) [24] for use in MANETs.

2.1.1.4 Ad-Hoc Mess Networks

A good overview of wireless mesh networks is in [25], [26]. Similar to the section above, a number of nodes dynamically self-configure as a mesh to create a network. There are typically a number of mesh routers that are relatively static that connect to a gateway/bridge back to conventional routers. The nodes in the network typically use traditional routing protocols such as TCP/IP.

2.1.1.5 Vehicular Networks (VANET)

There is a growing body of research on Vehicular Ad-Hoc Networks (VANET). They consist of a class of networks such as Vehicle-to-Vehicle (V2V), Vehicle-to-Roadside (VRC) and Vehicle-to-Infrastructure (V2I) [27]. The communication is done using Dedicated Short Range Communication (DSRC) and IEEE Wireless Access in Vehicular Environments (WAVE) 802.11p [28]. One issue is that these VANETs consist of very fast moving vehicles that are in broadcast range for a very short period of time.



(a) Hard Security Terms [3] **(b) Soft Security Terms [3]**

Figure 2.3: Information Assurance Terms

2.1.2 Information Assurance

There many Information Assurance terms and concepts utilized to describe key services or methods for providing security in a network. Figure 2.3 shows two different approaches and categorizes them as hard and soft security for use in a mobile ad-hoc network [3]. Both Subfigures 2.3a and 2.3b are key to providing security in any class of RCN. The former set of terms outlines security services used in traditional networks and is the goal for any RCN. The latter set of terms are more generic and can describe many of the processes used for security in an RCN. For example, 100% authentication is not possible in an RCN, nor are end-to-end routing tables available in many classes. This effects information availability. A scheme that provides a high quality of service level, by reducing an adversaries chance of receiving a message, and having a minimal end-to-end delay, could be designated as reliable.

2.1.2.1 Availability

Availability [14] : Availability is ensuring timely and reliable access to, and use of, information.

For a DTN, there are numerous factors that affect availability to include battery limitations, interactions with other nodes, the environment, and failure. The fact that nodes are mobile and that end-to-end routing tables are nonexistent make availability challenging.

A few examples where an advisory can effect availability in a DTN are denial of service and black hole attacks. It is not hard to imagine that an adversary can either insert a malicious node or malicious code that causes that node to advertise

itself as the best node to forward a packet, based on the metrics associated with a given DTN network protocol. All that node has to do is accept the packet and then discard. The effects of one or more nodes doing this could stifle any packet flow. The denial of service attack could be as simple as a continuous broadcast from one or more nodes; while this is not elegant and with time can be quickly found, it is effective. More subtle versions where the battery is drained are just as effective.

2.1.2.2 Integrity

Integrity [14] : Integrity is guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity.

For a DTN, this is ensuring that the data received by the destination is both accurate and complete. This essentially guarantees that the received data is the same data that was sent by the source, no more and no less. In many of the applications, especially military and first responder, modifications to data can be catastrophic such as sending EMS services to the wrong, or a ghost, location.

A few examples where an adversary can affect the integrity of the data are man-in-the-middle attacks and false packets. A man-in-the-middle attack can be conducted through the use of malicious code to hijack a node or by the insertion of a malicious node. That node then acts as any other normal node, potentially increasing its ability to receive a packet through modifications of the metrics used to determine forwarding in a DTN network protocol. Once the malicious node receives the packet, data modification to the packet payload or header can either cause corrupt data to arrive at the destination or the packet to be dropped or improperly routed.

2.1.2.3 Confidentiality

Confidentiality [14] : Confidentiality is preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information.

Most users of applications with military, emergency, business, or social media purposes want their traffic kept private. There are a multitude of attack vectors in a

DTN. The first is that most DTN protocols have no method to ensure confidentiality and any node that simply listens with the proper hardware and software can capture and store all broadcasts within its range. A more sophisticated adversary can either steal a node or gain access to a node and transfer the data. The most sophisticated adversary can conduct a type of man-in-the-middle attack, where the node acts like any other trusted node, but at intervals transfers all the data packets to an out of DTN storage location.

2.1.2.4 Authentication

Authentication [14] : Authentication is verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system.

There are a number of challenges to authenticating each time two nodes are within range and able to transfer data. The first is that time and data transmissions are required to perform the authentication process. There are numerous methods and algorithms for authentication on wired and wireless networks; however, most involve some centralized authentication server or the sharing of predetermined keys. This poses an issue as the network moves and new nodes are added to a DTN. While authentication is a worthwhile endeavor in DTNs, the process of authentication between nodes would likely occur prior to data transmission and would add a constant cost to the throughput on a DTN.

2.1.2.5 Non-Repudiation

Non-Repudiation [14] : Non-repudiation is protection against an individual falsely denying having performed a particular action.

It provides the capability to determine whether a given individual took a particular action such as creating information, sending a message, approving information, and receiving a message. Having processes, procedures and protocols in place that clearly show what person or computer conducted a particular action makes it easier to find, isolate, and quarantine those devices, malicious code, people or, in the case of DTNs, nodes.

(a) Trust Properties [7] **(b) Trust Relationship [30]**

Figure 2.4: Trust Overview

An example in wired networks is the use of digital signatures on an e-mail. If that signature requires a physical object, such as a common access card with a token and pin to unlock it, it is difficult for a person to say they did not send the e-mail. Something similar is possible in DTNs; however, the same issues as authentication come into play. Most methods of non-repudiation require centralized servers and the ability to check tokens.

2.1.3 Trust and Trust Properties

There are many definitions of trust. The *Cambridge Dictionary Online* defines trust as "to have confidence in something, or to believe in someone" [29]. *Adeli* provides a broad definition of trust as, "a relationship between a trustor, who we call Alice, and a trustee, we call Bob" [30]. Different disciples define trust in a multitude of different methods. *Cho et al.* conduct a multi-disciplinary analysis of trust and its definition in [7]. The authors look at how it is defined in sociology, economics, philosophy, psychology, organizational management and automatic computing in industry, and systems engineering. They propose a trust metric consisting of the following taken directly from [7]: "(1) trust should be established based on potential risks; (2) trust should be context-dependent; (3) trust should be based on each party's own interest (e.g., selfishness) (4) trust is learned (i.e., a cognitive process) (5) trust may represent system reliability." Figure 2.4a show the trust properties proposed for use in a MANET and by extension a RCN [7]. Figure 2.4b shows a trust relationship between two entities or, in the case of a network, nodes [30].

2.1.3.1 Trust

Trust : A value between 0.0 and 1.0 that represents the relationship between node i and node j with a value of 0.0 signifying no belief and a value of 1.0 signifying complete faith in node j's ability to successfully follow routing and security protocols.

This definition of trust is used through out this dissertation. Using a value between 0.0 and 1.0 allows for local management at each node and use in make routing and security decisions. This definition is based on the trust properties in [7] and the trust relationships as discussed in [30].

2.1.3.2 Trust Properties

Due to the nature of a Resource Constraint Network, trust is *dynamic*. Since nodes are normally mobile, and in many instances not in constant contact, trust can and does change over time. So the trust values must change over time to match a nodes ability to successfully route and provide security. These changes can be due to malicious activity, selfishness, hardware issues, or isolation.

Trust is *subjective* because nodes base some trust on the actions of other nodes and each node has a different picture of the network. If node i trusts node j at a certain level that does not mean that that opposite is true (*incomplete transitivity*).

As shown in Figure 2.4b, there is a relationship between each node and that relationship is "in a specific context" or *context-dependent*. In most networks, there are different categories of traffic, some is vital and some is best attempt; Chapter 6.2 proposes a message classification. Depending on message classification, forwarding is more dynamic and nodes with lower trust might receive a message. In certain circumstances, emergency response as an example, it might be more important to get the message through and hence you trust everyone. In others, only highly trusted nodes should be used.

Finally trust is not necessarily transitive (*incomplete transitivity*). If node i trusts node j and node j trusts node k, this does not mean that node i must trust node k to the same degree or at all.

2.2 Variable Continuity

Table 2.1 includes all key variables with description and the first location they appear in the dissertation. This is not an exhaustive list; however, it ensures that key variables remain consistent over multiple chapters.

Table 2.1: Variable Crosswalk

Variable	Description	First Referenced
h	Number of hops that a message traveled; if $h = 1$ then the message came directly from the source	Chapter 4.1
i, j, k, w	Used to reference generic nodes	Multiple
p	Probability that the next message segment arrives unchanged	Chapter 4.1
p_{nx}	Percentage of Good Nodes in a network	Chapter 4.1
n	Number of nodes in a network	Chapter 4.1
N	The set of all nodes in a network $\|N\| = n$	Chapter 4.1
h_c	Critical Hop Count Value	Chapter 4.1.3
k_{ec}	Number of Segments for Message Recreation	Chapter 4.2
s	Number of Segments a Message is broken into	Chapter 4.2
M	A Generic Message	Chapter 4.3
m	A message segment of M	Chapter 4.3
IT^k	A vector storing the indirect trust information node k has for all other nodes in the DTN	Chapter 5.1.1
DT^k	A vector storing the direct trust information node k has for all other nodes in the DTN	Chapter 5.1.1
AT^k	A vector storing the aggregate trust information node k has for all other nodes in the DTN	Chapter 5.1.1
α_a	Aggregate trust weight between direct and indirect trust information	Chapter 5.1.1
Δt	Time period to wait between trust aggregations	Chapter 5.1.1
tv^i	Node i's trust vector traded upon handshake	Chapter 5.1.3.2
α^{in}	"Freshness Factor" for Indirect Trust information	Chapter 5.1.3.2
β	Weight given AT^k	Chapter 5.1.3.3
γ	Weight given older indirect trust information	Chapter 5.1.3.3
tm^i	Node i's trust matrix traded upon handshake	Chapter 6.1
$sr^k_{(i,j)}(M_x)$	"Situational Risk" of node k forwarding a segment to node i	Chapter 6.1.1.1
nc	Number of copies of a m forwarded	Chapter 6.1.1.1
w_{nc}	Weight given nc	Chapter 6.1.1.1

CHAPTER 3
Resource Constraint Network Background

This dissertation explores Information Assurance in Resource Constraint Networks. There is a growing body of research about routing, security, and information assurance in multiple classes of Resource Constraint Networks. This chapter gives an overview of current research in a number of key areas: routing in Delay Tolerant Networks (DTNs), trust in a RCN and trust management in DTNs.

Because there is a lack of end-to-end routing tables in most wireless sensor network and DTN routing protocols, redundancy allows for successful message transmission. There are many definitions of redundancy; *Curiac et al* [8] define redundancy in a WSN in 13 different sub-areas. Many of the definitions are sensor network specific such as the ability to sense something from more than one sensor or estimate something based on data from neighboring nodes. The definitions listed below are taken from [8]:

1. Redundancy: the provision of additional or duplicate resources, which can produce similar results.

2. Spatial redundancy: the possibility to obtain information for a specific location from different sources.

3. Temporal redundancy (time redundancy): performing a specific action more than once, skewed in time, followed by checking the results in order to increase reliability.

4. Information redundancy: the use of redundant data, e.g. extra bits, to reconstruct lost information.

While the authors in [8] were specifically focused on WSNs, the idea that using redundancy in networks to maintain information availability is not new. Telecommunication companies, businesses, and governments use redundancy in designing networks to ensure high availability. Usually this is determined as $Ao = (total\ time -$

down time)/*total time*. Many companies sell services that guarantee a certain level of up time such as "five 9s" or 99.999%. The only way to ensure high network availability in a traditional packet-switched network is to have hardware and physical path redundancy. This redundancy is required for both the network hardware (routers, switches, and cabling) as well as the servers that run the required services such as mail, web, or database. There is also a need for redundancy of data locally using a RAID/NAS and globally using some technique for distributed dispersal of information [31]. For military communication planning, the acronym PACE (primary, alternate, contingency, emergency) is used to ensure multiple communication methods are available for each mission. For all examples above, redundancy is used to ensure availability of information.

Resources in a RCN by definition are constrained. For example, in a WSN, the inability to add a second transmitter on a node due to battery constraints effects routing protocols and information availability, because a node can only broadcast or receive at any given time (simplex channel) [18], [19]. This makes the use of redundancy for routing [4], [32]–[35], malicious node detection [36]–[41], and caching in a DTN [42] paramount.

The background information presented here reinforces the need for distributed trust management in a Delay Tolerant Network. It also validates the relevance of the three main research problems explored in subsequent chapters: direct clues (Chapter 4), fusion of direct and indirect trust (Chapter 5) and trust based routing (Chapter 6). Section 3.1 provides background on Information Assurance in a Delay Tolerant Networks. Specifically, it gives an overview of routing and other protocols that provide availability, and attempts to use the best nodes as part of any given path from source to destination. Trust is used as a metric for determining which node is best in a number of algorithms. Section 3.2 provides background on distributed trust management in Delay Tolerant Networks. There are three approaches outlined in [43]–[45] that are further explored below. Each one uses direct and indirect clues and aggregates those values to manage trust. While only the final example discusses specifics on how to route, they all discuss how their respective trust values can be potentially used for security and routing.

3.1 Literature Review: Information Assurance in Delay Tolerant Networks

Redundancy in a DTN is used for routing and malicious node detection in order to increase the availability of information. There are numerous papers that discuss both topics, with two main focus points. The first category uses redundant information to modify a routing protocol to provide additional IA security services or properly replicate the data through trusted nodes [33]–[35]. The second approach is to implement an additional protocol/scheme to assist with IA in order to identify adversarial nodes or defend against a specific attack vector [36]–[41].

3.1.1 DTN Modified Network/Routing Protocols

There are a number of DTN routing protocols found in literature. Each protocol fall in the spectrum between flooding the network and utilizing limited copies of a packet with some replication/forwarding process when nodes meet. There are trade offs between delivery and node resources. Due to the nature of DTNs, it is challenging to ensure delivery, data integrity, and confidentiality.

Three network protocols that attempt to provide information assurance are found in [33]–[35]. Secure Multi-Copy Routing proposes a modification to multi-copy DTN routing protocols using trusted nodes first in an attempt to reduce the probability of data compromise [33]. Erasure Coding is a protocol that splits packets into smaller subpackets through a coding scheme obscuring, and providing a method to ensure the integrity of, the data when it is received by the destination node [34]. CRISP proposes a routing protocol that uses a credit based scheme to create an incentive for a node to act truthfully [35]. All three provide some level of information assurance by increasing the availability of information. More specifics about erasure coding is in Chapter 4.2. Below is a more formal analysis for multi-copy routing. It discusses the adversarial model, algorithm, and IA analysis for multi-copy routing.

Secure Multi-Copy Routing:Adversarial Model: *Bulut et al.* use an unspecified malicious node adversarial model [33]. They explicitly state concern for an approach that attempts to determine, isolate, and bypass a node conducting a specific malicious behavior. Instead, they propose to utilize the trust between nodes

and a collective ability to mistrust an adversarial node to isolate and bypass that node. Additionally, they only consider a message or packet as delivered if, and only if, it does not pass through an adversarial node.

The protocol proposes a routing method using trusts to more efficiently route traffic in a compromised DTN [33]. The proposed scheme is an add-on to multi-copy DTN routing protocols; an example of which is Spray and Wait [4]. The proposal creates a two period routing approach with different configurable forwarding based on trust levels.

Secure Multi-Copy Routing: Protocol Overview: The authors define secure delivery as a message being delivered to its destination if, and only if, the message is received by the destination before the deadline and before any attacker receives it [33]. In multi-copy DTN routing, once the destination receives the message, it sends out an acknowledgement to inform other nodes that might still have the message in a buffer, to delete it. Even if the message is received by the destination, a node in the network can still forward the message to an attacker, if it does so prior to receiving an acknowledgement of successful delivery. Any protocol wants to limit the number of copies while maximizing throughput potential. The theoretical limit, disregarding trust, can be found for Spray and Wait [46] as shown in Equation 3.1 [33].

$$\begin{aligned} L_{min} &= arg\ min\left\{1 - e^{-\lambda L t_d} \geq d_r\right\} \\ &= \left\lceil \frac{ln\,(1 - d_r)}{-\lambda t_d} \right\rceil \end{aligned} \tag{3.1}$$

The goal of secure routing is to deliver a packet to the destination in a given time without it being read by a malicious node. This scheme attempts to accomplish this by sending the message to only trusted nodes in the first attempt. While in the second spraying period, sending to more risky nodes is allowed. Trusted nodes are those with a trust probability greater than p_t. If the sending node meets an attacker, it forwards the message to that adversarial node with a probability of $p = 1 - p_t$.

Secure Multi-Copy Routing:Algorithm: Only the trusted nodes are used in the first attempt to transmit the message from the source to the destination. If

that fails, because the number of trusted nodes is insufficient to transmit the message in a timely manner, then the algorithm in Figure 3.1 [33] is used to determine the optimal number of untrusted nodes to use. The authors prove a number of theorems necessary for the algorithm to function properly. They are taken from [33] and listed below.

1. For a given d_r, t_d, λ (rate of exponentially distributed intermeeting time between nodes), n (number of attackers), and $p = 1 - p_t$, the minimum number of copies that must be distributed to the network is:

$$L_{min} = \left\lceil \frac{ln\left(1 - d_r\left(pn + 1\right)\right)}{-\lambda t_d\left(pn + 1\right)} \right\rceil \tag{3.2}$$

2. When there are L_t trusted nodes carrying the copy of the message in the first period and L_u partially trusted nodes with the probability $p = 1 - p_t$ that start to carry a message copy in the second period (making in total $L_a = L_u + L_t$ nodes with a copy), to achieve a given d_r (with no t_d), the start of the second period, t_2, must be larger than a constant, t_2^{min}, where:

$$t_2^{min} = \frac{-ln\left(\left(1 - d_r\right)\left(\frac{L_a}{npL_u} + 1\right)\right)}{\lambda L_t} \tag{3.3}$$

3. For any given delivery deadline, L_u and L_t, the optimal value of t_2 that gives the maximum delivery rate by t_d is t_2^{opt}, where:

$$t_2^{opt} = t_d + \frac{ln\left(\frac{L_t npL_u}{L_a(L_a + npL_u - L_t)}\right)}{\lambda\left(L_a + npL_u\right)} \tag{3.4}$$

4. For a given delivery deadline, t_d, and desired delivery rate, d_r, the optimal number of untrusted nodes that maximize the overall routing cost which still achieves d_r by t_d can be computed using the algorithm in Figure 3.1.

Secure Multi-Copy Routing: IA Framework *Bulut et al.* proposed scheme has merit, utilizing trusted nodes to transfer messages and only when that fails will it use untrusted nodes. This will isolate and bypass malicious nodes on the

$L_u = 1$
Find $t_2^{opt}(L_u)$ from Eq. 3.4
while $F_{x_2} < d_r$ **do**
    $L_u = L_u + 1$
    Find $t_2^{opt}(L_u)$ from Eq. 3.4
if $L_t + L_u e^{-\lambda L_t t_2^{opt}} > d_r$ **then**
    Find exact $t_2^{opt_e xact}$ by binary search in $\left\lfloor t_2^{opt}(L_u), t_d \right\rfloor$
$opt\_L_u = L_u$
$opt_c ost = L_t + L_u e^{-\lambda L_t t_2^{opt_e xact}}$

Figure 3.1: Find Optimal Routing(L_t, p, n, d_r)

network; however, there are a couple of assumptions made that cause issues. The first is that trust for each node is universally assumed, challenging in a DTN. As in all networks, trust can change over time, especially if a node is compromised. A scheme in which to dynamically update trust values in a distributed manner was identified as a potential future problem. One example of this is presented in [47].

3.1.2 DTN Security Protocols

There are a number of additional add-on security protocols proposed to assist in finding or determining adversarial nodes. They are being classified as security protocols because many work irrespective of the underlying network protocol being used. Some require a class of routing protocol such as single or multi-copy and most were tested given a single layer III protocol. This could cause issues when attempting to port to other protocols but theoretically, with modification, could work with any underlying network protocol.

MUTON [36] uses a number of ferry nodes that move through an area where a DTN is present and take into consideration the transitive property of the network. These nodes calculate the packet delivery probability of each observed node and correlate that to the expected delivery probability to find potential adversarial nodes.

SPoofing by REplica ADjustment (SPREAD) [37] is a scheme to prevent "black hole" attacks. The goal of the scheme is not to stop, detect, nor isolate a node that is spoofing an address, but make the network robust enough to bypass

and still have the data packet arrive. It does so by slowly increasing the replication factor (number of packets) sent through the network. If there are enough malicious nodes, the number of packets slowly increases until it mimics endemic routing.

The authors in [39] propose a method for finding a captured node in a DTN. An analysis modeling the node capture is found in [48], [49]. The authors in [39] specifically focus on real world modeling to determine when a node has likely been captured. These models are largely based on the amount of interaction between nodes. If the node is isolated or only sees a couple of nodes all other nodes decay its connectivity until a threshold is met and the node is "blacklisted" on the network.

The goal of all of the protocols is to guard against a category of attacks such as spoofing, flooding, or non-responsiveness. While there is not a single solution proposed, each one has merit and can be used for further research.

Claim-Carry-and-Check: Adversarial Model: *Li et al.* use an adversarial model where a number of compromised nodes attempt to flood the network [38]. The authors describe two types of flood attacks: the first is a packet flood attack, where an adversary sends out numerous packets to attempt to use the battery and buffer capacity of other nodes. The second is a replica flood attack, where an adversary or selfish node floods the network with replica packets. The authors acknowledge that authentication could help, but it does not help against insider threats.

To counter the threat model, the authors propose a scheme to defend against flood attacks in a DTN [38]. They propose a Claim-Carry-and-Check approach, where each node claims the number of times that they either sent or replicated a packet during a configured time frame. This can be checked in a distributed manner with some probability. Because the network is sparse, a selfish node might attempt to send a packet multiple times to increase its chances for arrival.

Claim-Carry-and-Check: Protocol Overview: The Claim-Carry-and-Check scheme relies on the use of rate limiting proposed in [50]. Each node has a limit on the number of packets that can be generated and the number of packets that it can replicate in a specified time frame. Each node is responsible for counting itself and each packet is appended with the count. The only way for a node to go

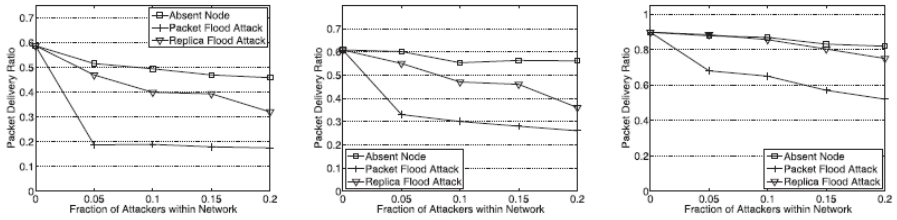


Figure 3.2: Flood Attack Affect on Packet Delivery Ratio

over the count during a time period is to lie. With some probability, a second node will receive two different packets with the same count. This indicates that a node is a potential adversary.

The authors break DTN routing protocols into three categories. Single copy algorithms that forward a copy and then delete it. Multicopy routing where the protocol sprays a certain number of copies. Propagation routing when the node finds the appropriate node to forward to by algorithm and replicates, but keeps its own copy.

Based on the three types of routing protocols, the authors conducted simulations to determine the effect of flooding attacks on DTNs. The specifics can be found in [38]. The overall results in Figure 3.2 [38] show that all three routing protocol types are vulnerable to packet flooding attacks, and single and multicopy protocols are susceptible to replica flood attacks.

Claim-Carry-and-Check: Algorithm The Claim-Carry-and-Check scheme detects an attacker by attempting to determine if a node violates their rate limit L. There are multiple ways to determine L proposed in [38]. Ideally, it would be optimized. If it is too short, then it will not find an adversary and if it is too long, then it could slow legitimate network traffic. Since no node can monitor all traffic on the network, [38] proposes a scheme where the node counts itself and shares that with other nodes by claiming the correct count. Receiving nodes store the claims and can use them to determine if a node is being truthful. In order for an adversarial node to replicate or send more packets than authorized in L, it must lie.

There are two types of attacks that Claim-Carry-and-Check focus on; packet flood and replica flood. Each one requires a different header appended to a network

packet. The packet claim count (P-count) is the number of packets a node i generates in time window T and defends against packet flood. The transmission claim count (T-count) is the number of times node i replicates a packet in time window T and defends against replica attacks.

When a source node S sends a new packet m, it generates a P-claim and appends it to m. If that same node needs to forward a packet, it generates a T-claim and appends that to the packet (see lines 2-5 in Figure 3.3 [38]). After S replicates the packet the number of times authorized by the routing protocol, it should delete the packet.

When a node receives a packet from another node, it conducts a number of checks. The first check is to verify the signature. The second checks the P-claim and T-claim against the count. If either fails, the packet is dropped (Figure 3.3, line 7-8). The node then checks the P-claim and T-claim against previous packets received. If there is a discrepancy, then the sending node is tagged as an attacker and put on a blacklist that is propagated throughout the network (Figure 3.3 lines 9-14). If neither of the first conditions are met then the packet is accepted and stored along with the new P-claim or T-claim.

Claim-Carry-and-Check: IA Framework *Li et al.* solve for the theoretical upper and lower probability of detecting an adversarial node based on the number of times K that a node sends or replicates a message above the threshold. They conduct a number of experiments using multiple routing protocols. Based on protocol and K, the detection rates vary. With a higher K, the detection is higher. This makes sense because the attacker is pushing more packets into the network for a larger effect. When K is lower the detection is lower, but so is the effect by the attacker.

There are a number of benefits to this scheme: the first is that all nodes must identify themselves and sign each packet. This ensures integrity and non-repudiation and allows for blacklisting when a node attempts to lie about the number of replicas or packets sent. There is no method for ensuring confidentiality. All nodes within broadcast range can hear the exchanges.

While the idea of a blacklist is a good one, there is no discussion about using that as a means of denying one, or all nodes, service. Since nodes are in and out

```
1  Metadata (P-claim and T-claim) exchange and attack detection
2  if Have packets to send then
3      For each new packet, generate P-claim
4      For all packets, generate their T-claim and sign them with a hash tree
5      Send every packet with the P-claim and T-claim attached
6  if Receive a packet then
7      if Signature verification fails or the count value in its P-claim or
         T-claim is invalid then
8          Discard this packet
9      Check the P-claim against their locally collected in the same time
         interval to detect inconsistency
10     Check the T-claim against those locally collected for inconsistency
11     if Inconsistency is detected then
12         Tag the signer of the P-claim (T-claim, respectively) as an attacker
             and add it into a blacklist
13         Disseminate am alarm against the attacker to the network
14     else
15         Store the new P-claim (T-claim, respectively)
```

Figure 3.3: Protocol Run By Each Node

of contact, an attacker can claim that all or some of the nodes it encounters are adversaries and have other nodes blacklist them. If that is done to even a small portion of the network, it can stop all network traffic.

3.2 Literature Review: Trust Management in Resource Constraint Networks

There is a number of ongoing research efforts in Resource Constrained Networks to establish useful trust mechanisms. Most of the efforts are in Delay Tolerant and Wireless Sensor Networks. These focus on using directly observable metrics in combination with referrals, references, or reputation to create a distributed trust management scheme where each node maintains trust for all other nodes in the network.

As in situations where people interact, trust of person A in person B is based on observed actions of person B and what mutual "friends" say about person B.

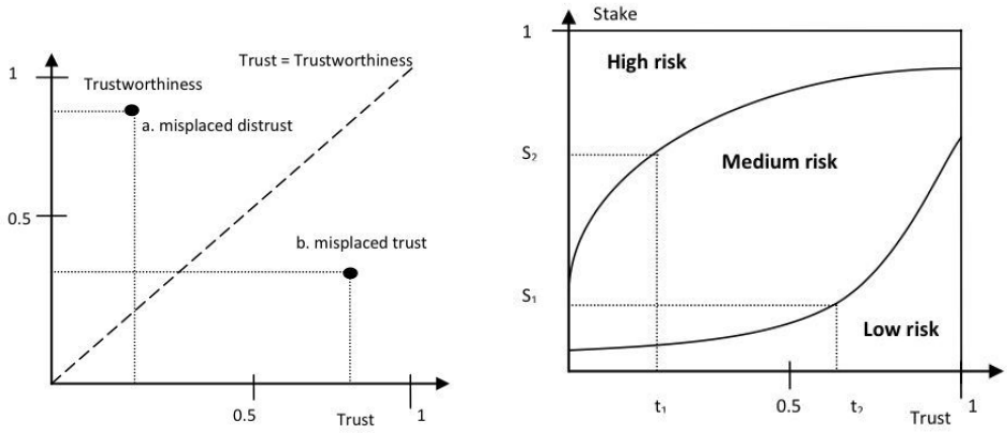
Table 3.1: Trust Characteristics and Properties of a DTN

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

A good overview of trust definitions and metrics is presented in [7] and [30]. The authors outline how nodes interact and propose a number of characteristics and properties (Table 3.1 [7]). In a RCN, typical trust verification methods do not function efficiently and for some constraints do not work at all.

Nodes in RCNs, like people, make trust decisions based on direct and indirect observations as stated above. Under this set of conditions, the ability to trust at the proper level leads to better results. Figure 3.4a displays the level of trust node a has in node b with values ranging from [0.0,1.0). In this figure, the dash line is the trust given by node a when it is equal to the trustworthiness of node b. While in the example they are equal, the graph illustrates two key areas. The areas above and below where they are equal are represented by point a and point b respectively. Point a shows where node a's trust level for node b is lower than its trustworthiness. This can have negative effects on the network leading to node a not forwarding a message to node b (even if b is closer to the destination) because of trust. Point b shows where node a puts too much trust in node b. This can have the obvious effect of passing messages to malicious nodes. Overall Figure 3.4a illustrates the necessity to converge, as quickly as possible, on a trust value node a has for node b that is as close to the actual trustworthiness of node b.

Risk in the U.S. Army is codified in ATP 5-19, Risk Management [53]. The Risk Management (RM) principles consist of: integration of RM into all mission and operation phases; making risk decisions at the appropriate level; accepting no unnecessary risk; and applying RM cyclically and continually. Mission Command, ADP 6-0 [54], explicitly lists "Accept Prudent Risk" as one of the six principles of



(a) Trust Level [7], [51] **(b) Risk and Trust [7], [52]**

Figure 3.4: Trust Figures

mission command. Figure 3.4b shows trust vs. stakes and the risk to the network. If the stakes are low the nodes with lower trust can be utilized without a large increase in risk; however, if the stakes are high, the opposite holds.

Combining the two into an integrated trust management scheme is the source of many new research endeavors. There are three schemes proposed in literature that integrate direct and indirect node observation and use some of the principles listed in Table 3.1 to manage trust in a DTN. In [43], the authors use a Bayesian approach to determine the probability that a node is behaving well. In [44], [55], the authors creates a bipartite graph and find outliers; the scheme removes nodes with probabilities outside of a certain value in order to converge on trust values of other nodes. A third approach, outlined in [45], uses both direct and indirect metrics and determines good versus bad encounters over four categories to aggregate a trust value.

In general, there are a few key points that are consistent through all three of the proposed trust management schemes discussed below. They all become part of the proposed scheme in Chapter 5. The first is that each uses a direct and indirect trust component based on positive and negative interactions. Some do not say how those clues are obtained, but they use them in the data fusion for an aggregate trust. The second is that each attempts to exclude outliers. The third is that they decay older interactions and trust associated with those interactions. Finally, they

produce a trust value. Not all produce a trust value between 0.0 and 1.0, but they can be normalized to such a value. The main difference between each is what clues they use and how the trust values are integrated.

3.2.1 Bayesian Approach

Denko et al. propose a Bayesian learning approach to determine the trust probability of a given node in a network [43], [56]. The probabilistic trust scheme uses both direct and indirect trust computations. It takes into account the observed actions, positive and negative, and updates trust levels for each node.

Equation 3.5 [56] 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 is n_s and the number of negative interactions is n_u. The authors use $\alpha = n_s + 1$ and $\beta = n_u + 1$ for the beta distributions. They also assume that complete information cannot be collected. The authors compute a trust value using the beta distribution as shown below.

$$T_A(B) = E(f(x; \alpha, \beta)) = \frac{\alpha}{\alpha + \beta} = \frac{n_s + 1}{n_s + n_u + 2} \tag{3.5}$$

Modifying for indirect trust computation is done using Equation 3.8 [56]. The authors calculates 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. The total number of satisfactory recommendations is $n_s^r = \sum_{k=1}^{i} n_s^k$ and unsatisfactory recommendations is $n_u^r = \sum_{j=1}^{i} n_u^j$. The authors define α and β as follows.

$$\alpha = n_s + n_s^r + 1 = n_s + \sum_{k=1}^{i} n_s^k + 1 \tag{3.6}$$

$$\beta = n_u + n_u^r + 1 = n_u + \sum_{j=1}^{i} n_u^j + 1 \tag{3.7}$$

Using recommendations modifies Equation 3.5 and makes the updated trust value

for $T_A(B)$ as follows.

$$T_A(B) = \frac{n_s + n_s^r + 1}{(n_s + n_s^r + 1) + (n_u + n_u^r + 1)} = \frac{n_s + \sum_{k=1}^{i} n_s^k + 1}{n_s + n_u + \sum_{k=1}^{i} n_s^k + \sum_{j=1}^{i} n_u^j + 2} \tag{3.8}$$

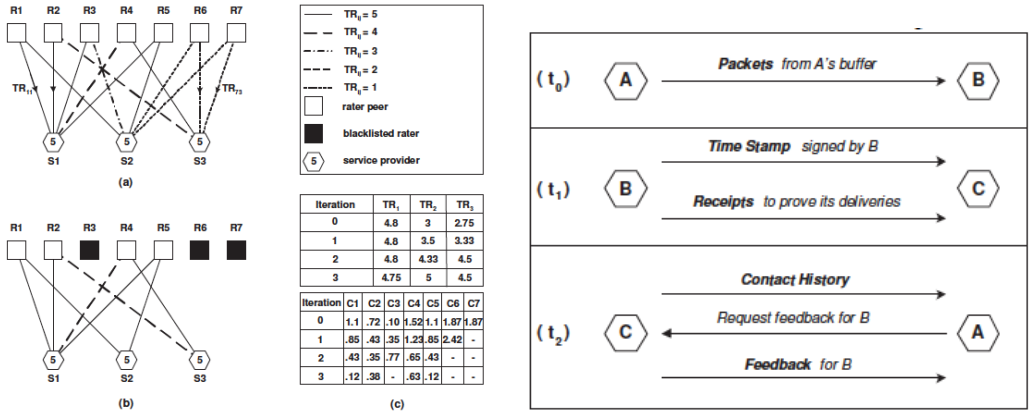
Because recommendations can be false, the authors add a threshold for excluding or judging recommendations. They first exclude recommendations from nodes suspected of bad behavior and then find the average trust using recommendations from valid sources. A threshold is created for trust difference. Each recommendation is used to compute the trust and if $|T_{ave}(B) - T_R(B)| > S$ then the recommendation is thrown out and $T_{ave}(B)$ is regenerated. This is done until no recommendations are outside of the threshold. In theory, this would remove all of the false 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. Older recommendations receive a lower weight than newer recommendations. Integrating judging and decaying weight of recommendations and interactions over time produce a more comprehensive scheme. The authors do simple simulations in [56] and add in a confidence component and further simulations in [43].

3.2.2 Iterative Trust and Reputation Management Mechanism (ITRM)

Ayday et al. propose a Trust and Reputation Management Mechanism (ITRM) in [44] and an iterative algorithm based on ITRM for DTN trust management in [44], [55]. They specifically state their concern for a Byzantine (insider) attack and define malicious nodes as those that drop, modify, or inject packets into the network to deny or disrupt network operations. ITRM is a graph based iterative algorithm with two main goals: computing the reputation of nodes that send a 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



Iteration	TR_1	TR_2	TR_3
0	4.8	3	2.75
1	4.8	3.5	3.33
2	4.8	4.33	4.5
3	4.75	5	4.5

Iteration	C1	C2	C3	C4	C5	C6	C7
0	1.1	.72	.10	1.52	1.1	1.87	1.87
1	.85	.43	.35	1.23	.85	2.42	-
2	.43	.35	.77	.65	.43	-	-
3	.12	.38	-	.63	.12	-	-

(a) ITRM Example [44], [55] **(b) Feedback mechanism [44], [55]**

Figure 3.5: ITRM DTN Trust Figures

each SP is a *bit vertex*. The authors in [44] compute an initial value of each bit-vertex j using the following equation.

$$TR_j = \frac{\sum_{i \in A} R_i \times TR_{ij}}{\sum_{i \in A} R_i} \tag{3.9}$$

The set A, "is the set of check-vertices connected to the bit-vertex j [44]."

An inconsistency factor is computed for every check-vertex i as follows, "where Υ is the set of bit vertices connected to the check-vertex i and $d(\cdot, \cdot)$ is a distance metric used to measure the inconsistency [47]."

$$C_i = \left[\frac{1}{|\Upsilon|} \right] \sum_{j \in \Upsilon} d(TRij, TR_j) \tag{3.10}$$

If the inconsistency is greater than τ for any check-vertex(s), the node with the largest discrepancy is "blacklisted" and all its ratings are deleted. Equation 3.9 is then recalculated minus the "blacklisted" check-vertex and the inconsistency re-calculated using Equation 3.10 [44]. The inconsistencies are checked again. This process continues until no check-vertex inconsistency is greater then τ. Figure 3.5a [44] shows an example of this iterative process.

The trust management scheme for DTNs proposed in [44] transmits packets

using a Bloom filter [57] and some sort of ID-Based Signatures (IBS) [58] to secure direct connections between nodes. Figure 3.5b illustrates the feedback process for determining good versus bad entries in the rating tables used in the ITRM process. The example shows three nodes A, B, and C. In this case, A judges the actions of B. A sends a packet to B which is subsequently sent to C. Later, when A meets C, it checks on the status of the message sent. It then can make a determination on the actions of B.

3.2.3 Trust Thresholds - Trust Management Protocol

Chen et al. in [45] propose a trust management and routing protocol that uses both direct and indirect observations similar to the other two schemes mentioned above. The authors in [45] use the terms *QoS trust* and *social trust*. The former includes two metrics "connectivity" and "energy" and the latter "unselfishness" and "healthiness." A nodes trust level is a real number in the range [0,1], where 1 is complete trust and 0 is no trust. The trust value node i has for node j at time t is computed as follows.

$$T_{i,j}(t) = \sum_{X}^{all} w^X \times T_{i,j}^X(t) \tag{3.11}$$

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. To evaluate $T_{i,j}(t)$, the authors in [45] use the following notation: "node i is the trustor, node j is the trustee, node m is a newly encountered node, and node k is a recommender." The trust value is calculated using the following:

$$T_{i,j}^X(t + \Delta t) = \beta T_{i,j}^{direct,X}(t + \Delta t) + (1 - \beta) T_{i,j}^{indirect,X}(t + \Delta t) \tag{3.12}$$

The value for β is in the range from [0,1) and is the 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. It is either direct, if the encountered node is j in Equation 3.12 [45], or indirect if it is not.

If node i encounters node j and there is enough time for data to be transmitted, and additional encounter(s) to occur, the trust is updated else there is a small decay of the last direct trust level. 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. Since this is a direct connection, the indirect trust from node i to node j is decayed.

If node i meets node m then indirect trust is updated for node j. Again if the time frame is too short to transfer information, the current indirect trust level decays. Assuming that node m gives recommendations that are within a threshold they are used to update the indirect trust. This process takes into account the current trust level that node i has for node j.

Once the direct and indirect trust is updated for each trust property, Equation 3.11 [45] is used to aggregate the trust for each node. The authors in [45] conducted an analysis to determine the best weights for each of their trust properties given the number of malicious nodes in the network.

CHAPTER 4
Direct Observations Through Redundancy

Messages are passed from source to destination in a Delay Tolerant Network (DTN) based on which nodes meet and trade messages in a given time frame. Those meetings, and what a node can observe by monitoring broadcasts of other nodes while not transmitting, limit the type and number of clues for use in detecting malicious nodes. As previously defined, trust is a value between 0.0 and 1.0 that represents the relationship between node i and node j with a value of 0.0 signifying no belief and a value of 1.0 signifying complete faith in node j's ability to successfully follow routing and security protocols. In order to maintain that value, some clue or clues, directly observed by a given node are necessary to modify trust up and down based on other nodes behavior. Ideally, if a node successfully follows the routing protocols and successfully passes a message from source to destination, it should receive an increase in trust and if not, a decrease.

If node i sends a message to node k, and then later observes node k send that same message to node j, and the payload from node k to node i does not match, then trust can be modified. If the source sends a message and then later overhears the same unmodified message, then trust can be modified for both the initial receiving node and the node that forwarded the message. If the source receives a proper acknowledgement from the destination, then it can increase trust to the node that was the first hop on the path. Common in all of these examples is that only those nodes directly observed cause changes in t rust. Each observation occurs over time (time delay) and normally from a different locations/perspectives in the network (nodes moved).

A DTN does not maintain end-to-end routing tables, thus making it challenging to ensure Information Assurance. Each routing protocol attempt to ensure

Portions of this chapter previously appeared as: T. Babbitt and B. K. Szymanski, "Trust management in delay tolerant networks utilizing erasure coding," in *2015 IEEE Int. Conf. on Commun., Ad-hoc and Sensor Networking Symp.*, London, United Kingdom, Jun. 2015, pp. 7959-7965.

availability of information, by prescribing when and how a message is forwarded, when two nodes meet as they move in a DTN. Any message from a source node to a destination node will take redundant paths in space and time. Chapter 3 introduced redundancy in a WSN and provided multiple definitions that apply to a DTN. The two most applicable are spacial and temporal redundancy. These led to the idea of using path redundancy, through erasure coding, as a directly observable clue to be used as a metric to assist in distributed trust management. The scheme takes advantage of the concept of using path (spacial) redundancy and information redundancy associated with erasure coding and a checksum to modify trust levels that are then updated over time (temporal redundancy).

When the source needs to send a message to the destination, it appends a checksum onto the message and then uses erasure coding to break that message into segments. All segments are sent through the network and by design and redundancy take multiple paths to the destination. Based on what arrives at the destination, trust decisions are made based on whether or not the message can be recreated. This chapter explores a number of key concepts in order to use path information as a clue for direct trust. The first is to find the probability that the next segment to arrive at the destination is good (Section 4.1). This is required for the cost and subsequent utility functions that determine if it is better for the destination to wait, or request that a given message is resent, when a bad message segment arrives (Section 4.3.1). The second concept is an overview of Erasure Coding (Section 4.2). The third idea is a complete description of how path information, gleaned by using erasure coding, is utilized as a clue including the aforementioned utility functions proposed for destination wait versus message resend (Section 4.3). Finally, section 4.4.1 provides simulations results for the simple trust management scheme proposed for use with path information as a direct clue.

4.1 Path Probabilities in a Delay Tolerant Network

As discussed in the introduction and in Chapter 3.1, redundancy is used for information availability for routing and for identification of malicious nodes in limited scenarios. Most schemes and protocols for a DTN do this in a distributed

manner. Each node only has access to information or clues that it directly observes. Epidemic modeling is used in biology to show how disease spreads, in computer science to show how viruses spread, and in social settings to show how ideas spread. A good overview is found in [59]. In reviewing DTN literature, there was no clear way of determining, with a definitive probability, that a given packet is intact (not modified, read, or in some other manner tampered with) which, in a fashion, is finding the probability that a disease spreads along the fastest paths from node i to node j. Furthermore, given a value for the percentage of malicious nodes in the network, is there a way to determine the probability that a packet arrives at the destination intact? Doing so is protocol dependent and requires some assumptions about a DTN.

4.1.1 Concept Overview and Assumptions

Assuming that node x is the destination, and at time t has estimated trusts for all other nodes in the network, then the average of those trust values is p_{nx}. Let p_{nx} denote the percentage of good nodes in the network. Let p represent the likelihood that the next packet to arrive at the destination is good, and is equal to the probability along all likely paths from source to destination. In section 4.3.1 below, this model will be used to determine the utility of waiting for another segment.

The rest of this section discusses how to approximate p using the following equation

$$p = \sum_{h=1}^{n-1} f_h^{(n)} \times (p_{nx})^h \tag{4.1}$$

where n is the number of nodes in the given DTN and $f_h^{(n)}$ is the fraction of paths with h hops from the source to destination a segment likely will take. The essential for p value of $f_h^{(n)}$ is routing protocol dependent; however, using a simplified and restrictive routing protocol allows for a reasonable approximation for p. In this simple protocol, each node maintains a value, $t_{i,j}$ from 0.0 to 1.0 that equates to the frequency of inter-meetings between nodes i and j. Assume that $t_{i,j} = t_{j,i}$. Without loss of generality, assume below that the source is node 1 and the destination node n. The node currently holding the message segment, i, will pass it to met node j only if j is the destination ($j = n$) or if $t_{i,n} > t_{j,n}$. Also assume that the network

is fully connected, and if node j does not meet the destination then $t_{j,n} = 0$, so it will never be selected as an intermediate node. This protocol approximates well the number of hops expected in epidemic routing as it trades number of hops for time of delivery. The fastest packet reaching the destination in epidemic routing is likely to traverse the route that segments in this protocol travel.

Let edge (i, n), where $1 \leq i < n$ has weight $0 \leq x_i \leq 1$. According to the routing protocol, an intermediate node $1 < i < n$ is eligible for passing a message segment (in short, eligible), if $x_i > x_j$, where j is the node currently holding the message segment. Below outlines our approximation method for determining the probability of a path having the given hop count and the number of intermediate nodes $n-2$ using the simple routing protocol above. This is used to determine $f_h^{(n)}$ in equation 4.1.

Let $p_h^{(n)}$ denote the probability that a message segment arrives at the destination along a path with h or fewer hops. Of course $0 < h < n$ and $p_{n-1}^{(n)} = 1$ because no message segment is passed to the same node twice. Hence, values of $n > 2$ are considered. It is easy to show, by induction, that the value of interest, $f_h^{(n)}$, is defined by the relevant probabilities as

$$f_h^{(n)} = p_h^{(n)} \prod_{j=1}^{h-1} (1 - p_j^{(n)}) = p_h^{(n)} (1 - \sum_{j=1}^{h-1} f_j^{(n)}) \tag{4.2}$$

Consequently, once $p_h^{(n)} \approx 1$ the fraction of paths longer than h is negligible.

4.1.2 Exact Solution For $h = 1$

There are $n-2$ intermediate nodes, each with a uniformly random edge weight to the destination, distributed with the same probability distribution. Hence, the probability that there are j eligible nodes is $\sum_{j=0}^{n-2} \binom{n-2}{j} (1 - x_1)^j x_1^{n-2-j}$ and the probability that the destination will be chosen with this number of eligible nodes is $\frac{x}{x + \sum_{i=1}^{k} y_i}$ where y_i denotes the weight of the edge from the i-th eligible node to the source. The following $n-1$ integrals is the solution:

$$p_1^{(n)} = \int_0^1 \dots \int_0^1 \sum_{j=0}^{n-2} \binom{n-2}{j} \frac{(1 - x_1)^j x_1^{n-1-j}}{x_1 + \sum_{i=1}^{j} y_i} dx_1 \dots dy_{n-2} \tag{4.3}$$

Algebraic solution for $n = 3, 4$ is simple, and yields:

$$p_1^{(3)} = \int_0^1 \int_0^1 \frac{(1-x_1)x_1}{x_1+y_1} dy_1 + x_1 dx_1 \approx 0.7046 \tag{4.4}$$

$$\begin{aligned} p_1^{(4)} &= \int_0^1 \int_0^1 \int_0^1 \frac{(1-x_1)^2 x_1}{x_1+y_1+y_2} dy_2 + \frac{2(1-x_1)x_1^2}{x_1+y_1} dy_1 \\ &+ x_1^2 dx_1 \approx 0.5763 \end{aligned} \tag{4.5}$$

For larger values of n, the numerical integration can be used to get values for validating simulation results.

4.1.3 Approximation For General Case Of $0 < h < n-1$

After $h-1$ hops, there is $n-h-1$ intermediate nodes left and each has probability $1-x_h$ to be eligible, where x_h denotes the edge weight of the current segment holder to the destination. The expected number of eligible nodes is $(1-x_h)(n-h-1)$ each with the average edge weight to the source being $1/2$. Since each hop, on average, hits the middle of the previous range, the size of the h range, for weights of eligible nodes, is $(1-x_1)/2^{h-1}$ so the value of x_h is $(x_1 + 2^{h-1} - 1)/2^{h-1}$. To get a good approximation, compute the expected value over each half of this range which yields the result:

$$\begin{aligned} p_h^{(n)} &= -\frac{2}{(n-h-3)} + \frac{2^h(n-h-1)}{(n-h-3)^2} \\ &\times \ln\left[\left(1 + \frac{1.5(n-h-3)}{2^h}\right)^{\frac{1}{3}} \left(1 + \frac{0.5(n-h-3)}{2^h}\right)\right] \\ &\approx \frac{2^h * (n-h-1)}{(n-h-3)^2} \ln(1 + \frac{n-h-3}{2^h}) - \frac{2}{(n-h-3)} \end{aligned} \tag{4.6}$$

The critical value for this function is $h_c = log_2(n)$. For $h > h_c$, $\ln(1+(n-h-3)/2^h) \approx (n-h-3)/2^h$ so the result is nearly 1. For example, it is greater than 15/16 for $h > h_c + 1$. For $h = h_c - h' < h_c$ an approximate value is $1/2^{h'} * h' * \ln(2) - 2/n$ close to 0. For example, this value is less than 0.1 for $h' > 5$.

It is easy to check that the peak of fraction of paths happens around $log2(n) - 3$

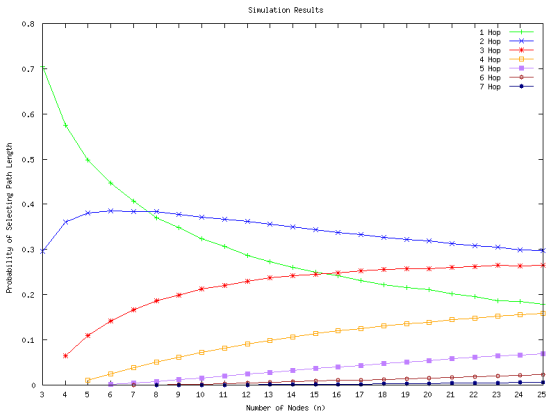


Figure 4.1: Probability of Using a Path with h Hops

and these fractions are significant only for h's from $h_c - 5$ to $h_c + 1$. Approximating again, using $h_c - 3$ as the value for the most common path length and then using Equation 4.6 produces the following.

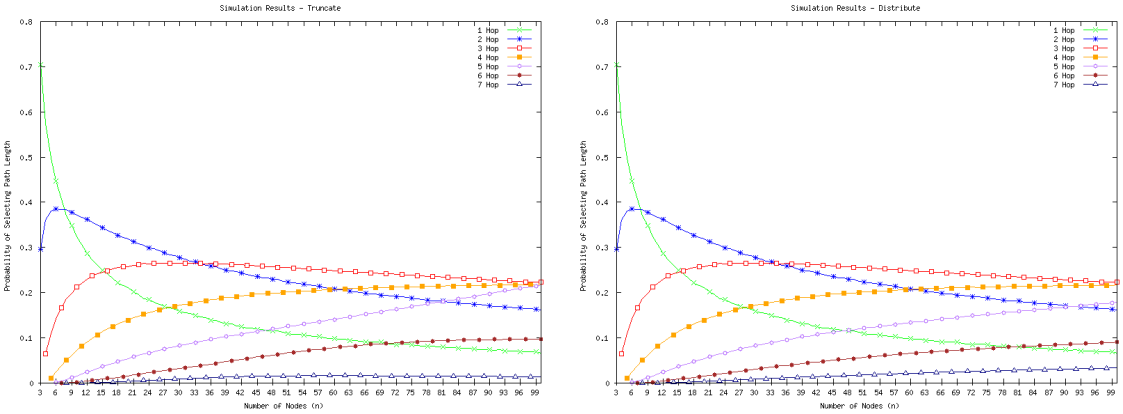
$$p \approx p_{nx}^{\log_2(n)-3} \approx 1 - (\log_2(n) - 3) * (1 - p_{nx}) \tag{4.7}$$

The final approximation holds only for $p_{nx} > 0.8$ and moderate $9 < n < 256$. In summary, the average path length grows and the probability p decays slower than $log_2(n)$.

4.1.4 Simulation Results

In order to confirm the mathematical bounds and behavior discussed above, simulation of the simple routing protocol using a complete graph with n nodes was created (see Appendix A). Each edge has a random weight selected uniformly in the range (0.0,1.0]. In order to confirm the approximations from Equations 4.3, 4.6 and 4.7, the likelihood of using all possible paths is determined in order to ultimately calculate the probability of using a path of h hops. Each simulation was run 50,000 times for each value from $n = 3$ to $n = 100$ and the results were averaged.

For $3 \leq n \leq 27$, the simulation ran through completion with no modifications. Figure 4.1 shows how the probability of using a path with h hops changes as the number of nodes in the network increases. Once $n > 9$, there is a higher probability of using a path with two hops than with one. A path of length three hops occurs



(a) Probability of Using a Path with h Hops - Truncate Approximation

(b) Probability of Using a Path with h Hops - Distribute Approximation

Figure 4.2: Path with h Hops: Simulation with Approximations

more often than one hop when $n > 17$.

Because the number of possible paths is a multiple of $n!$, checking all possible path becomes impractical; however, by slightly modifying the simulations all probable paths are calculated. The program recursively calls possible paths increasing h each time. The residual probability p_r, probability of seeing a node with a shorter path to the destination prior to seeing the destination, of following a given path slowly erodes until it is minuscule. There are two different approaches for dealing with p_r. The first is to assume that it all goes to p_{h+1}. This is a valid approach because it will yield a maximum probability of following a path with a lower value of h. It is more likely that a packet takes a short rather than a longer path. The second is to spread that probability over a number of values for h so $p_{h+1} = p_{h+1} + \frac{p_r}{2}$ and $p_{h+2} = p_{h+2} + \frac{p_r}{4}$. This is continued for a specified number of additional hops. This has merit because over thousands of iterations there is approximately a 0.5 chance that a node with a higher probability of seeing the destination is met prior to meeting the destination.

Figure 4.2 show the results of the simulations for $3 \leq n \leq 100$. The simulations were run given the same specification as listed above except when $p_r < .00001$ for a given path that value of p_r was dealt with in one of the two methods described in the previous paragraph. Figure 4.2a shows the results of truncating ($p_{h+1} = p_{h+1} + p_r$). Figure 4.2b shows the results of distributing the trust; this simulation distributes

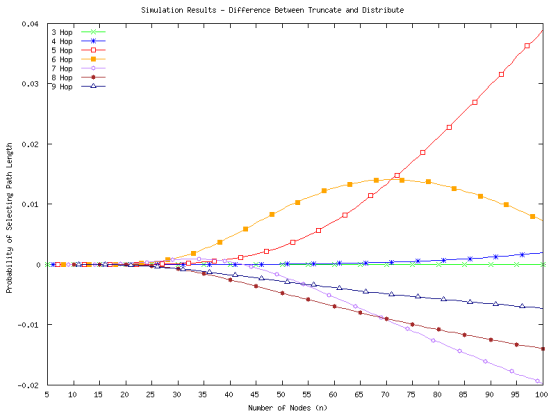


Figure 4.3: Simulation with Approximation Comparison

trust along the next 7 hops.

A comparison of the two approaches is in Figure 4.3. They are identical for $h = 1$ and $h = 2$. As expected, for $3 \leq h \leq 6$ the truncation approach yields a higher value for $f_h^{(n)}$. For $h > 6$ the distribution method yields a greater value for $f_h^{(n)}$. A two-samples t-test with a confidence interval of 0.95 was conducted to compare the results for $h = 5$ and $3 \leq n \leq 100$ between the truncate and distribute method for handling p_r. Using a Null Hypothesis: the difference of means for both methods is equal to 0. Table 4.1 clearly proves the hypothesis demonstrating that the difference between the methods, using the $h = 5$, the value with the most variance (see Figure 4.3), is statistically insignificant.

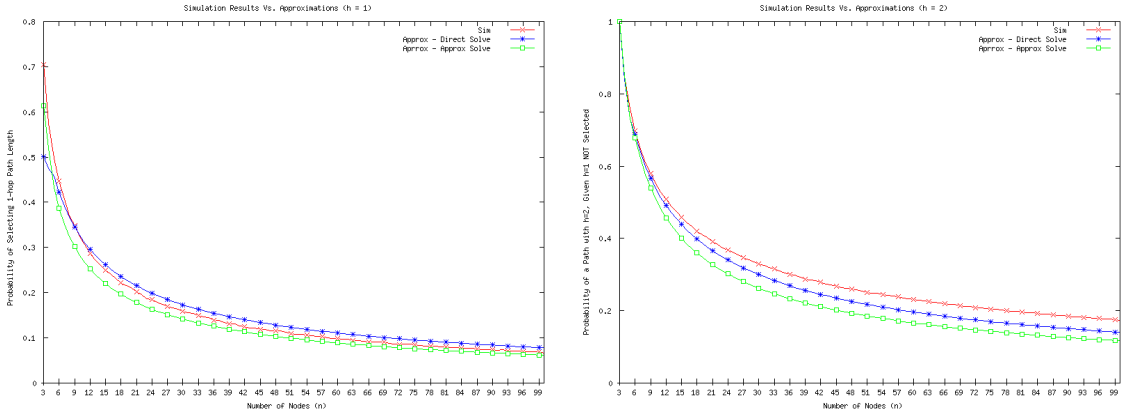
Equation 4.4 gives our expected value for $p_1^{(3)} = 0.7046$ the simulation results are $p_1^{(3)} = 0.7045$. Equation 4.5 gives our expected value for $p_1^{(4)} = 0.5763$ the simulation results are $p_1^{(4)} = 0.5755$. This confirms our intuition about how the protocol reacts but, as stated above, it is not a practical approach for larger n.

Figure 4.4a shows the approximation for $h = 1$; Figure 4.4b shows the approx-

Table 4.1: Comparison of Truncate and Distribute Method for $h = 5$

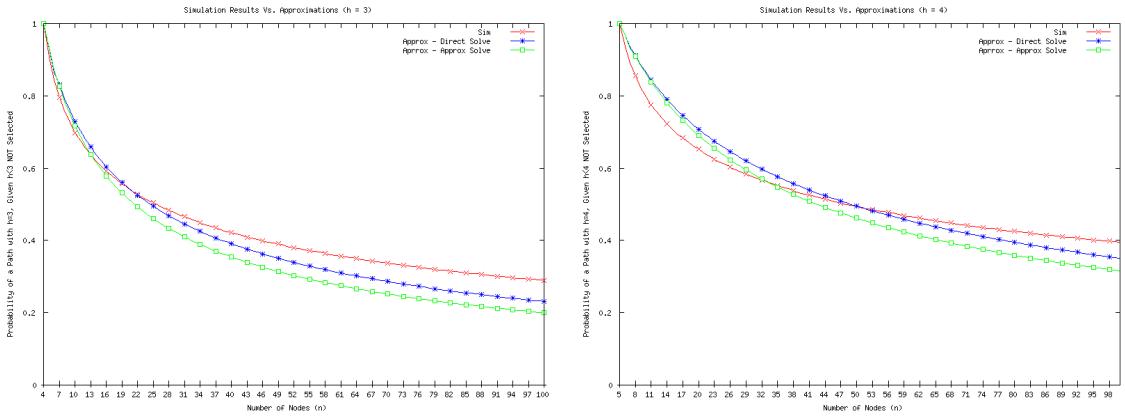
	Approximation Method						95% CI for Mean Difference		
	Truncate			Distribute					
	M	SD	n	M	SD	n		t	df
$p_5^{(n)}$ Sim	0.665	0.137	95	0.645	0.148	95	-.0199, .0617	1.011*	188

*$p = 0.314$; $p \geq 0.05$



(a) Simulation and Approximation Results $p_1^{(n)} = f_1^{(n)}$

(b) Simulation and Approximation Results $p_2^{(n)}$



(c) Simulation and Approximation Results $p_3^{(n)}$

(d) Simulation and Approximation Results $p_4^{(n)}$

Figure 4.4: Approximation Fit for One Through Four Hops

imation for $h = 2$; Figure 4.4c shows the approximation for $h = 3$; and Figure 4.4d shows the approximation for $h = 4$. The "Approx - Direct Solve" is the values using the approximation equation and "Approx - Approx Solve" uses the approximation of the approximation (see Equation 4.6). While the approximation is not exact, it gives a tight lower bound as n increases. Interesting to note is that for $h = 3$ and $h = 4$ the approximation overestimates the probability for lower values of n. Additionally, the probability of stopping at $h = 3$ or $h = 4$ is higher as n increases than for $h = 1$ or $h = 2$. This makes sense because adding more nodes to the network will make it more likely to take a path longer than $h = 1$ or $h = 2$; however, only to a point.

4.2 Erasure Coding

Erasure coding, as a network protocol, has been studied for use in a DTN [60]–[62]. It works by breaking a message into a set of message segments. When a sufficiently large subset of message segments are received, the original message can be reconstructed. Specifically, erasure coding starts with a message of size M and the the total size of information $I = M(1+\epsilon)$ needed for message recreation. ϵ is a small constant that depends on the exact encoding algorithm used. Then, the minimum number of segments, k_{ec} is selected, such that I is divided evenly by k_{ec}. Finally, the total number of segments, $s > k_{ec}$, is chosen and the encoded message is broken into that many segments. The value $r = (1+\epsilon)s/k_{ec} > 1$ is called a replication factor as it defines how much more information is sent to transfer M bytes of a message. For the purposes of this dissertation the exact encoding algorithm is not important.

When using erasure coding, the key aspect is the replication factor r. To recreate a message, only s/r of the message segments must arrive at the destination. In order to transmit the message segments over multiple different paths, an algorithm similar to *srep* [61] can be used. In that variation of erasure coding, the generated message segments are split between $s = k_{ec}r/(1+\epsilon)$ relays. For example if $r = 5$, then in order to send a message of M bytes, $5 \times M$ bytes of data is transmitted, and if $\epsilon = 0.25$ and $k_{ec} = 3$ then the number of nodes that receive segments are $s = k_{ec}r/(1+\epsilon) = 12$. Furthermore, to reconstruct the original message, k_{ec} segments must arrive at the destination. A lower replication factor or multiplier k_{ec} reduce the number of separate message segments that must be transmitted through the network.

Zakhary et al. use the properties of erasure coding and replication and propose "Erasure Coding with Replication" (ECR) [34]. This routing protocol uses the concepts of erasure coding listed above and attempts to increase information availability. The authors use an adversarial model where multiple nodes are compromised followed by a "black hole" attack that drops data packets.

4.3 Erasure Coding Contribution to Trust Management in Delay Tolerant Networks

The proposed trust management scheme uses erasure coding (see Section 4.2) and a checksum to determine the trustworthiness of neighbor nodes. If all message segments successfully arrive to recreate a complete message, then the trustworthiness of those neighbors increases. If one or more of the segments are corrupt, maliciously or not, then by analyzing additionally received segments, bad actors can be determined. This is done by having the source node append a checksum onto every message sent prior to using erasure coding to segment the message and send it through the network. The destination is then in a position to validate each message and make trust decisions. The scheme steps are listed below.

1. Node i needs to send a message M to node j.
2. A Checksum is added to M and s segments using Erasure Coding are generated such that k_{ec} segments are required to recreate M and $s > k_{ec}$.
3. All s segments are sent through the network from i to j.
4. If k_{ec} unique segments arrive at the destination, or some intermediate node, the node recreates M.
5. If M has a valid checksum, the node increases the trust for all nodes that sent a valid segment and skips to 7, otherwise it continues to 6.
6. The receiving node continues to wait for each additional segment m until recreating M produces a valid checksum or based on cost the destination determines it is better to request resending of the message (see section 4.3.1). If an additional segment m allows M to have a valid checksum, the receiving node increases the trust for all nodes sending a valid segment and decreases it for all nodes sending a faulty segment and then continues to 7.
7. The receiving node waits for time T and accepts any addition segments for M, validity of each is checked against $k_{ec} - 1$ known good segments and trust along the relevant path is accordingly changed.

4.3.1 Utility of Waiting For One More Segment

If the first k_{ec} message segments are intact, then all additional segments m can be used to determine if the sending node of the $(k_{ec}+m)^{th}$ segment is truthful, see step 7 in section 4.3 above. This requires one message creation operation using $k_{ec}-1$ known good segments and the unknown $(k_{ec}+m)^{th}$ segment. If the recreated message is good then the sending nodes trust is increased, otherwise it is decreased. The issue is when the first k_{ec} segments do not recreate the original message. In order to conduct the cost benefit analysis below, the probability p that the next segment arrives intact is necessary (see section 4.1 for complete analysis of p).

4.3.1.1 Cost and Benefit

The total probability of being able to assemble the message from $k_{ec}+m$ segments is as follows:

$$P_{k_{ec}+m} = \sum_{i=0}^{m} p_{k_{ec}+i} \tag{4.8}$$

For $m > 0$, the change in probability from $k_{ec}+m-1$ to $k_{ec}+m$ can be expressed as:

$$\Delta P_{k_{ec}+m} = p_{k_{ec}+m} = \frac{(k_{ec}+m-1)!p^{k_{ec}}(1-p)^m}{(k_{ec}-1)!m!} \tag{4.9}$$

The justification is simple. Since $k_{ec}+m-1$ segments were not enough to recreate the message but $k_{ec}+m$ are, the $(k_{ec}+m)^{th}$ segment has to be correct and the $k_{ec}+m-1$ segments received previously must contain exactly $k_{ec}-1$ correct segments. Thus, $p^{k_{ec}}$ defines probability of having k_{ec} correct segments, while $(1-p)^m$ is the probability of having m corrupt segments and the rest of the expression defines the number of ways $k_{ec}-1$ correct segments can be chosen from $k_{ec}+m-1$ segments previously received.

The cost of being able to assemble the message from $k_{ec}+m$ segments is as follows:

$$C_{k_{ec}+m} = \alpha \sum_{i=0}^{m} c_{k_{ec}+i} \tag{4.10}$$

where α defines the ratio of the cost of computation to the value of increased probability of being able to recreate the message from additional segments. When the $(k_{ec}+m)^{th}$ segment arrives it is compared to all $k_{ec}-1$ subsets of $k_{ec}+m-1$

segments already received but not sufficient to recreate the message, hence, for all $m > 0$ the change in cost from $k_{ec} + m - 1$ to $k_{ec} + m$ can be expressed as follows

$$\Delta C_{k_{ec}+m} = \alpha \frac{(k_{ec} + m - 1)!}{(k_{ec} - 1)! m!} \tag{4.11}$$

4.3.1.2 Utility Functions

There are two options after receiving $k_{ec} + m - 1$ segments. The first is to wait for the $k_{ec} + m$ segment. The second is to request sending the message again. To make this decision, we can use one of the two criteria listed below.

1. The first criterion is to compare the expected gain in probability as expressed in Equation 4.9 with the cost of Equation 4.11.

2. In the second criterion, the price of receiving the next segment is weight against the price of resending the message. The expected gain of receiving the message is $P_{k_{ec}} = p_{k_{ec}} = p^{k_{ec}}$ at the cost of $n * r + \alpha$ which is the cost of resending nr plus the cost of unpacking the segments α.

Using the first criterion, wait for the next segment if the gain in probability to recreate the message is greater than the cost. The second is to receive the best price per increase in probability. The former is a simplified function while the later is more comprehensive since it takes into account the cost of resending the message.

The first utility function is $G_{k_{ec}+m} = p_{k_{ec}+m} - \alpha c_{k_{ec}+m}$. It is beneficial to wait for the $k_{ec} + m$ segment if $G_{k_{ec}+m} > 0$. Substituting the values from Equation 4.9 and Equation 4.11, the following holds:

$$p^{k_{ec}}(1 - p)^m > \alpha \tag{4.12}$$

Equation 4.12 has its merits; however, it is based solely on the increase in probability of being able to recreate a message with an additional segment being greater than the cost of this segment processing. The increase in probability shrinks very quickly limiting the number of segments for which the destination waits before requesting resend of the message, which intuitively makes sense.

The second more complicated utility function takes into account the cost of having to resend the message a second time. Ultimately, it compares the price increase between waiting for another segment versus that of resending the message. This gives the inequality of $\alpha\frac{c_{k_{ec}+m}}{p_{k_{ec}+m}} < \frac{(\frac{k_{ec}}{r}+\alpha)}{p^{k_{ec}}}$. Since $\frac{c_{k_{ec}+m}}{p_{k_{ec}+m}} = \frac{1}{p^{k_{ec}}(1-p)^m}$, the inequality can be reduced to $\frac{\alpha}{(1-p)^m} < \frac{k_{ec}}{r} + \alpha$. The cost of $\frac{k_{ec}}{r} \gg \alpha$ so the inequality can be rewritten as $\frac{\alpha r}{k_{ec}} < (1-p)^m$. Taking the natural log of both sides gives the final inequality as follows.

$$m < \frac{ln\left(\frac{\alpha r}{k_{ec}}\right)}{ln(1-p)} \tag{4.13}$$

In this inequality, α represent technology factors, as the node processing becomes faster and faster at the same price due to the chip technology, α becomes smaller and smaller. Yet, the value of m grows only logarithmically with this gain. The erasure coding algorithm parameters r and k_{ec} change relatively little for different applications, so their impact on m can be ignored. Finally, $1-p$, measuring how "polluted" and to a lesser degree how large (see Equation 4.7) the network is, has a large impact; the higher the pollution level the larger m the destination should wait before requesting resend. This again is intuitively clear, as the resend message will face the same treacherous journey to destination as the original message did.

4.4 EC Trust Management Simulations

In order to test the proposed trust management scheme, a number of simulations were executed using NS3. While it is a very powerful simulation tool, there are very few extensions of it written for use with Delay Tolerant Networks. This paper uses the work done by *Lakkokorpi et al.* in [63]. The available DTN module for NS3 implements the DTN bundle layer first proposed by the Delay Tolerant Networking Research Group (DTNRG) and codified in a number of DTNRG request for comments [64], [65]. The bundle layer protocol manages application to application transportation with each bundle usually being larger than a normal IP packet. This facilitates complete application interaction between a source and destination in either one or a small number of bundles. Each bundle has a timer and if it is not delivered within a specified time it is deleted from a nodes buffer. The point

to point connections between nodes can be managed by UDP or TCP. There are numerous routing protocols proposed for use in DTNs. While most were not written to use the bundle layer, almost all can be modified to do so.

The DTN module used in this dissertation is compatible with NS3 version 3.18 and includes the implementation of two DTN routing protocols: endemic [66] and stray and wait [32]. For each protocol a number of duplicate bundles are sent from source to destination to increase the probability that one will arrive. In the former protocol, a bundle is send to any node that does not yet have a copy in its buffer. This has been shown to significantly clog the network and use unnecessary resources. The latter is a more refined approach and sends a smaller number of copies and then waits to see if the bundle arrives. It only increases the number of nodes it forwards to if it does not receive an acknowledgement packet from the destination in a specified time.

The proposed trust management scheme is built using the concept of erasure coding to determine which nodes, if any, modify a segment of the data transmission between the source and destination. The authors in [63] give a good breakdown of the code they created. The code includes the main DTN module *dtn.cc* and the bundle layer encapsulation headers *mypacket.cc* and *mypacket.h*. The necessary modification to simulate EC and the proposed trust management scheme are broken down into two main categories: simulate erasure coding and simulate trust management at the node level. Section 4.4.1 gives an overview of the modifications to allow for erasure coding. Section 4.4.2 give an overview of the changes to replicate node trust management. Section 4.4.3 explains the simulations run and Section 4.4.4 presents results.

4.4.1 Erasure Coding Simulation For DTN Module in NS3

In order to simulate Erasure Coding, some modifications to the existing routing protocols in the DTN module were required. Erasure coding as described above multiplies a message payload by a replication factor r, this modified message is divided and each segment is sent from source to destination. Each message segment is sent as a bundle with its own individual bundle ID. For a given message there

is a single message ID. This update in the DTN module is call a PID and is an addition to the bundle header. When k_{ec} unique unmodified bundles arrive at the destination with the same PID, the message is recreated.

The user provides k_{ec}, ϵ (that for simplicity we assumed to be 0 in the current simulation) and s (which defines r) as initial input to the simulation setup to allow for the segmentation of messages into $k_{ec}r/(1+\epsilon)$ bundles. Based on the values for k_{ec} and r, each bundle is scheduled for transmission. For the first hop the source node will only send one bundle with a particular PID, unless it directly meets the destination. This forces multiple paths from source to destination.

While currently erasure coding is implemented in an austere manner and utilizes the already implemented endemic routing protocol to move the bundles, it is useful for showing preliminary results using EC for trust management. Although currently using epidemic routing, the approximate metrics found in Section 4.3 apply to the simulation results as explained in that section.

4.4.2 Trust Management Object In NS3 For Use In The DTN Module

Like in many distributed trust management schemes, each node maintains a structure that includes the trust level for each node it interacts with. Additionally, many schemes aggregate trust levels with neighbors through periodic comparison of stored trust values. To simulate such a scheme in NS3, a trust management object was added. This was created for use with each NS3 DtnApp outlined in [63]. The DtnTrust object initializes all nodes in the network to a trust level $t_{node} = 0.5$, $0.0 \leq t_{node} \leq 1.0$. This DtnTrust object tracks the bundles that arrive, and once k_{ec} unmodified unique bundles with the same PID arrive, recreates the original message.

After the successful message recreation, each node that sent an unmodified segment has its trust level increased to $t_{node} = t_{node} + .05$. For each node that sent a malicious segment(s), that nodes trust is decremented to $t_{node} = t_{node} - .05$. This can happen at any node that receives enough unique unmodified bundles to recreate a message. Generally this only occurs at the destination; however, certain paths might contain a single node that forwards many bundles that are part of a given message and such nodes may also compute trust for their neighbors. While the current

version uses only a static additive increase or decrease in trust, a modification that mimics the sliding window size change similar to TCP might bear fruit.

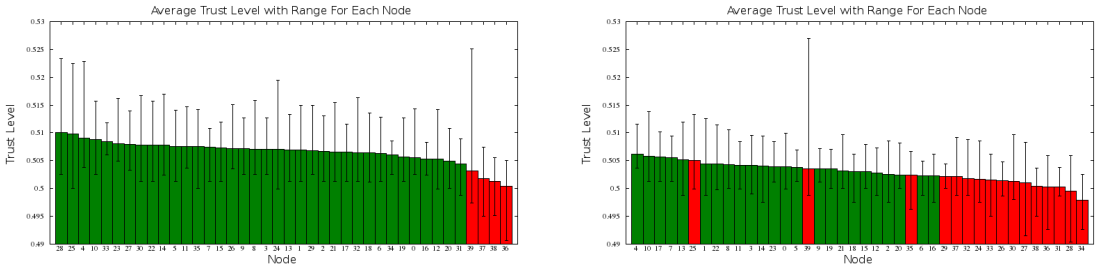
Once a message is recreated at a node, any additional bundles that arrive with that PID are checked. If the segment sent is unmodified the trust for the sender is incremented, else the senders trust is decremented. This occurs for a given time or until all bundles arrive at the destination and anti-packets are sent out that clear all nodes buffers for that PID.

Currently the trust level is not used to deter sending a bundle to a node; however, there are functions to allow a node to check the current trust of all other nodes. An extension is to phase out nodes that do not behave properly. Those nodes would no longer be allowed to send or receive bundles.

4.4.3 Simulation Overview

The NS3 simulator modified with DTN extensions, is used to conduct the simulations presented. The threat model against which this scheme is attempting to protect includes an adversary who either hijacks, reprograms, or adds malicious nodes to the DTN that modify all packets that arrive prior to sending them on as per the routing protocol. There are more sophisticated models where the attacker intermittently modifies packets or where nodes collude. While it is likely this scheme with full use of the Bundle layers security module [65] would provide useful results, such extensions are left for future endeavors. This adversarial model is simulated in NS3 by making x% of the nodes always act maliciously by modifying all bundles they receive prior to forwarding. The malicious node does not modify bundles when it is the source or destination.

The NS3 simulation tool provides the user with a number of traffic and mobility models. Similar to [63], a simple traffic model is used in our simulation. Each node sends a number of variable sized messages, which are broken into a set of bundles defined by k_{ec} and r, at a random time within each 200 second interval of simulation run time to another random node. For node to node transmission, the segments are fragmented into 1500-byte data-grams using UDP, with retransmission, to provide reliability.



(a) 90% Path Trustworthiness **(b) 60% Path Trustworthiness**

Figure 4.5: Ranking of All Nodes According to Their Trustworthiness

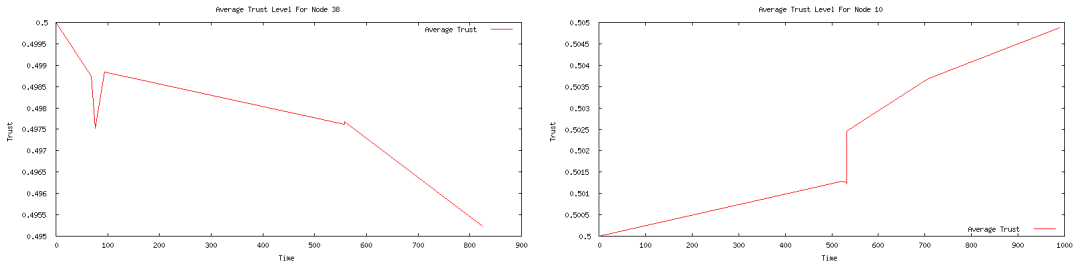
The mobility model used is the Random Way Point (RWP) model, which is conveniently built into the NS3 simulator. For each simulation, a total of 40 nodes move in a 2500m x 2500m grid. These nodes are evenly distributed and select direction and speed at random (uniformly distributed) times. The maximum speed is 20m/s with a 2 second pause. All nodes pause between each movement. The simulation is run 10 times each with a different seed and all simulations are run for 1000 seconds.

4.4.4 Simulation Results

The results show that over time the average probability across all nodes increase for good nodes and decrease for bad ones. There is a large contrast in result when the percentage of good nodes drops from 90% to 60%. This makes sense because with a higher number of untrustworthy nodes, the likelihood that they would corrupt a message from source to destination and negatively impact trust for a good node increases.

4.4.4.1 90% Path Trustworthiness

Figure 4.5a shows the average trust level for each of the 40 nodes sorted in the decreasing order of their trustworthiness. While the decrease is not steep, the four untrustworthy nodes that modified all forwarded bundles have the lowest average trustworthiness among all nodes (those are nodes 35-39 and colored red). The error bars show the largest and smallest individual run. This gives an idea of the range of values.



(a) Trace of Node 38, 90% Path Trustworthiness

(b) Trace of Node 10, 60% Path Trustworthiness

Figure 4.6: Trace of a Given Untrustworthy Node over Time

Figure 4.6a illustrates one malicious node trust evolution over a complete simulation. This figure shows the change in average trust of node 38 from time 0.0 seconds to time 1000 seconds. The trend is down as expected. At approximately 90 seconds there is an increase in trust. It appears when node 38 is the source of a message and sends it directly to the destination. In that case, the trust is increased by the receiving node because in the threat model all messages start without modification, which make sense because even if the message payload contains malicious content, the checksum is being generated at the source and it would be correct.

4.4.4.2 60% Path Trustworthiness

Figure 4.5b shows the average trust level for each of the 40 nodes sorted by their trustworthiness. Due to the number of malicious nodes, 16 in this case, there is no steep slope, however, the majority of the malicious nodes, colored red, have the lowest average trust and only three have trust higher than a good node colored green.

Figure 4.6b illustrates the change of a trustworthy node over time. Its average trust goes generally up, but lowers and then spikes around 550 seconds due to passing on a tainted message segment and then sending a number of trustworthy segments.

4.5 Conclusions

This paragraph proposes a novel trust management scheme that uses spacial, temporal and information redundancy as well as the traits inherent with erasure

coding and checksums to modify trust in a distributed manner. Based on the simulation results, the scheme shows promise and validates the use of redundancy in RCNs and path information as a clue to help with the information assurance properties of authentication. The ability to validate a nodes trust is directly related to the ability to authenticate a node or message in a distributed manner.

Based on the results of the simulations, discussed in the previous section, a number of additional improvements are merited. They include the expansion of the threat model to include a larger array of adversaries, sliding trust windows, implementation and simulation of node exclusion based on changing trust, integration of a trust sharing extension, implementation and experimentation with the bundle security module, ability to add and delete nodes from the network, and analysis of the effects of false positives and false negatives on the network.

A number of potential future research topics are listed above. With modifications, this scheme can assist in filling the gap in information assurance in resource constrained networks. Chapter 5 further explores distributed trust management by expanding the directly observed path clues and exploring the use of sharing trust and the fusion of direct and indirect trust into an aggregate trust that more efficiently converges.

CHAPTER 5
Trust Management in Resource Constrained Networks – Fusion of Direct and Indirect Trust

A number of key observations about a distributed trust management system, for use in a Delay Tolerant Network, were previously explored in Chapter 3.2. One observation is that a distributed trust management system must properly fuse trust gathered from directly observable clues (direct trust) with trust based on recommendations of other nodes (indirect trust). Using direct and indirect trust fusion is necessary, because each node can only observe part of, and interact with only a subset of, the total number of nodes in the network. One of the properties of a Recourse Constrained Network, codified in [7], is that trust is subjective and based on a particular nodes point of view. As nodes interact, they trade messages. Additionally, they must trade trust information in order to converge on the actual trust value for all other nodes in the network. This chapter explores the second research question: What is the proper method of managing indirect and fusing that with direct trust to converge on an accurate trust value in order to create a distributed trust management scheme?

As presented in Chapter 4, path redundancy is a useful clue for observing the direct actions of other nodes in a DTN. It further shows that using information about paths and direct observations will identify malicious nodes over time under a limited threat model. While security and trust are not equivalent, in a DTN or WSN where the use of centralized servers is not feasible, the ability to trust whether or not a node is compromised or acting selfish is paramount.

Recent publications include a number of proposals for managing trust in a

Portions of this chapter previously appeared as: T. A. Babbitt and B. Szymanski, "Trust management in resource constraint networks," in *Proc. 10th Annu. Symp. Inform. Assurance (ASIA '15)*, Albany, NY, Jun. 2015, pp. 51-56.

Portions of this chapter previously appeared as: T. A. Babbitt and B. Szymanski, "Trust metric integration in resource constrained networks via data fusion," in *18th Int. Conf. Inform. Fusion (Fusion 2015)*, Washington, DC, Jul. 2015, pp. 582-589.

DTN (Chapter 3.2). None of the aforementioned DTN trust management schemes take into account directly observable information based on an understanding of redundant paths. The version presented in the previous chapter detects malicious nodes; however, it is limited by threat model and only uses those nodes directly connected to the destination. More details are enumerated below:

1. The scheme only works for a limited threat model (a node that always modifies a message it receives).

2. A node only takes into account direct observations to manage trust (see Chapter 4.3 and Chapter 4.4).

   (a) Using only direct observations can be misleading when only a portion of the network is taken into account. This can skew trust levels for many nodes.

   (b) The complete path a message takes from source to destination is not taken into account.

3. Nodes are not "blacklisted" for having a low trust level and routing decisions are not made based on trust levels.

A more comprehensive distributed trust management scheme is proposed in this chapter and is it's main contribution. This scheme continues to take into account path redundancy but utilizes information about the full path. Additionally, it integrates an indirect trust sharing scheme. These additions directly address the first two items listed above. The final item is addressed in Chapter 6.

There are a number of additional building blocks to a successful distributed trust management scheme. The first additional contribution is the use of path information to expand direct trust (Section 5.1.2). The second is multiple methods for trust aggregation between direct and indirect trust (Section 5.1.1). Finally, multiple means in which to manage indirect trust are presented (Section 5.1.3). Section 5.2 includes comprehensive simulations showing the utility of the scheme.

5.1 Trust Management Scheme Overview

While the schemes listed in Chapter 3.2 use a number of different trust indicators, they all essentially boil down to finding the number of good versus bad transactions for a given indicator. None of them takes into account the complete path followed by a message M. All schemes consist of two parts. The first is a direct trust computation and the second is the use of shared information between nodes as an indirect trust computation. The two are then combined in some fashion to determine the trust node i assigns to node j at a given time t. Other key characteristics include a method to eliminate outliers, decay older interactions/trust associated with those interactions and produce a trust value.

5.1.1 Direct and Indirect Trust Aggregation

In order to manage trust, every node k that is part of the DTN, maintains three vectors of size n. The first is the indirect trust vector IT^k that maintains indirect trust for all other nodes in the network based on trading trust information. 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 two 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. Equation 5.1 updates aggregate trust for all $j \in N$, where N is the set of all nodes in the network. This is straightforward and depending on the value of α_a can give more or less weight to direct versus indirect observations.

$$AT_j^k = (1 - \alpha_a)\, DT_j^k + \alpha_a IT_j^k \tag{5.1}$$

Indirect trust is updated as each node is met or every time period designated as Δt (see Section 5.1.3). If using Δt, modification of Equation 5.1, to take into account

the current trust of the nodes that provide indirect trust information, decays the significance of indirect trust as the average trust of nodes making recommendations decreases. The tracking and fusion of this is detailed in the next section. The average trust of those nodes providing indirect trust information is designated as IT_{av}^k. Equation 5.2 substitutes $\alpha_a IT_{av}^k$ for α_a and makes the aggregate trust update based on the trust of recommending nodes. If during a given time period Δt, only suspect nodes are met, then their recommendations are given less weight.

$$AT_j^k = \left(1 - \alpha_a IT_{av}^k\right) DT_j^k + \alpha_a IT_{av}^k IT_j^k \tag{5.2}$$

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 t_i^k is found using Equation 5.3. The updated value used for trust decisions is found using Equation 5.4 and 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.

$$t_i^k = \lfloor \frac{currentTime - DT_{(i,ts)}^k}{\Delta t} \rfloor \tag{5.3}$$

$$AT_j^k = \left(1 - \alpha_a^{\frac{1}{\lambda t_i^k}}\right) DT_j^k + \alpha_a^{\frac{1}{\lambda t_i^k}} IT_j^k \tag{5.4}$$

Combining approaches two and three above (Equation 5.2 and 5.3) gives the following:

$$AT_j^k = \left(1 - \left(\alpha_a IT_{av}^k\right)^{\frac{1}{\lambda t_i^k}}\right) DT_j^k + \left(\alpha_a IT_{av}^k\right)^{\frac{1}{\lambda t_i^k}} IT_j^k \tag{5.5}$$

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

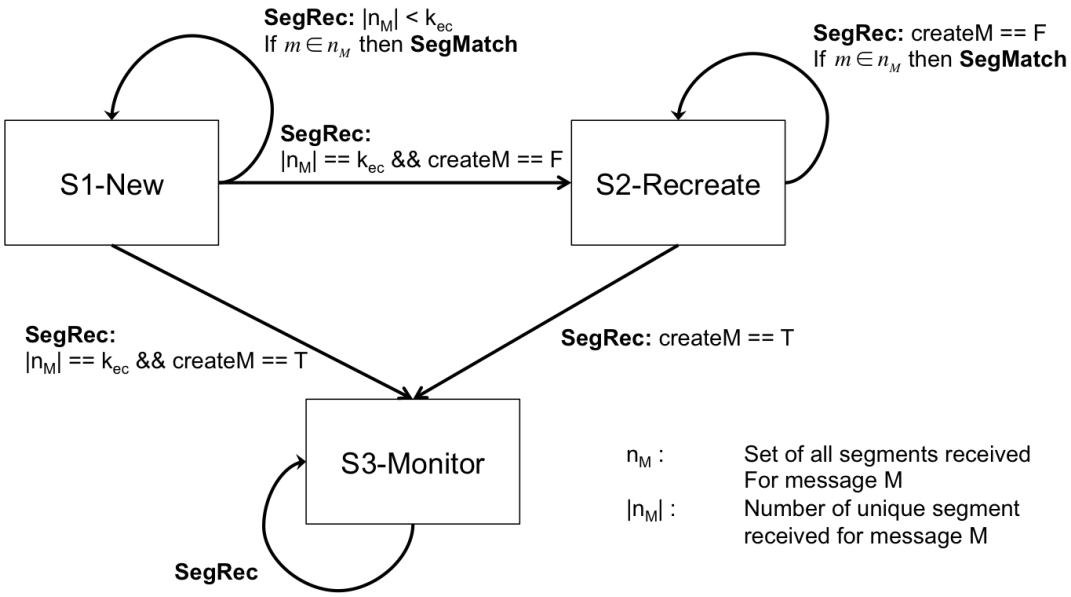


Figure 5.1: Node State Diagram

5.1.2 Trust: Direct Observations/Clues

For the direct component, the primary observed clue is the paths taken to recreate the message. Figure 5.1 presents a state diagram showing how a node processes each message M it receives. Chapter 4 provides a complete analysis and results for using path information as a clue. For clarity, k_{ec} is an erasure coding variable that represents the number of message segments required for a given node i to recreate message M. The additional contributions in this section are the use of full path information (Section 5.1.2.1) and the **SegMatch** function (Section 5.1.2.2).

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_{ec} unique segments arrive, node i attempts to recreate the message using the **SegRec** function. This function checks to see if the checksum matches once the message is recreated and returns a true or false value. 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. The **SegRec** function

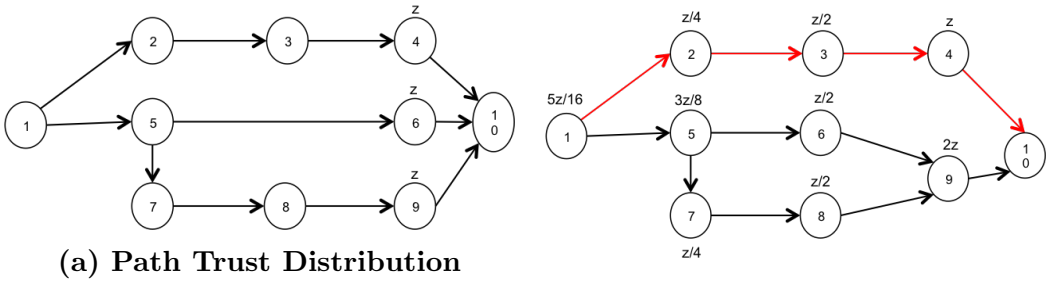
iterates through all permutations of message segments received for M and returns true for those that successfully recreate the message. When the number of segments received is k_{ec}, as occurs when in state $S1$, there is only one function call. Once $|n_M| > k_{ec}$, as in state $S2$, then $\binom{|n_M|}{k_{ec}}$ iterations are required. If **SegRec** is successful for any permutation, then node 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 discussed in Chapter 4.3.1 and published in [9]. If it is better to retransmit the message, then node i sends a message to node j to resend.

Once in state $S3$, node i waits prior to sending an Acknowledgment Message (ACK). This is contrary to most RCN routing protocols that send an ACK right after a successful delivery to the destination. An ACK clears node buffers and avoids wasting resources to send a message or segment through the network once it has been successfully delivered. Taking advantage of path and temporal redundancy node i accepts additional segments m. It continues to recreate the message using $k_{ec} - 1$ known good segments and makes trust decisions based on the success of the message recreation. It will continue to do this for a set time period and then send an ACK.

5.1.2.1 Expanded Path Information

In Chapter 4.3, each node makes trust modifications based solely on which nodes directly forward each segment used to recreate M. Figures 5.2a 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 $k_{ec} = 3$. Each of the three required segments takes a different path. The three paths taken are $\{1, 2, 3, 4, 10\}$, $\{1, 5, 6, 10\}$ and $\{1, 5, 7, 8, 9, 10\}$. In this case, node 10 will only increase the trust level for nodes 4, 6 and 9 even though there were six other nodes involved.

Each node along a path can append its ID to each segment as it flows through the network. Assuming, for now, that a node truthfully adds their node ID, the destination will know the path that all segments travel. Given that z is the trust modifier, either static or sliding, then once a message is recreated trust can be



(a) Path Trust Distribution Chapter 4.3

(b) Updated Trust Distribution

Figure 5.2: Trust Distribution Changes

distributed back along the path that each segments followed. In Figure 5.2b, this is done by giving each directly connected node z increase and then dividing that value in half for each hop back along a path. If there is a situation where a node, directly connected to the destination, is along n_{path} multiple paths, such as node 9 (Figure 5.2b), 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. Let's assume that message M was recreated and node 10 is is in state $S3$. A segment of message M arrives using the red path in Figure 5.2b and **SegRec** returns false. Negative trust can be distributed back along the path $\{1, 2, 3, 4, 10\}$.

5.1.2.2 Trust Updates Using Segment Matching

Figure 5.1 shows all the states that a node goes through for each message M. When a new segment m arrives for M and the message either cannot be recreated because not enough unique segments have arrived (state $S1$) or attempted recreation failed (state $S2$), but the node has seen segment m along a different path, the **SegMatch** function is called. This function slightly modifies trust by comparing the paths that each message segment m took prior to arrival at node i.

In Figure 5.3 node 1 is the source and node 7 is the destination, or some intermediate node. Node 7 receives m twice with the two paths followed as $\{1, 2, 5\}$ and $\{1, 3, 4\}$. Assume node 7, from its perspective, maintains a set of "trusted" nodes consisting of all nodes above a certain threshold and designated as set A. In

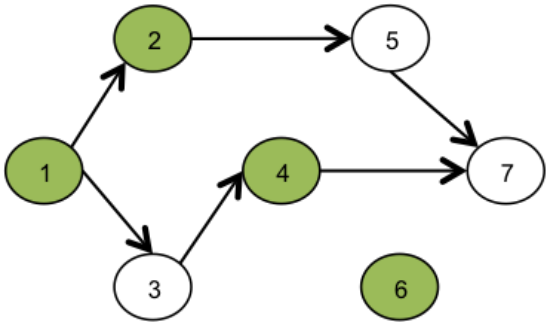


Figure 5.3: SegMatch Example

Figure 5.3, all of the green nodes are above that threshold, $A = \{1, 2, 4, 6\}$. The set of all nodes along the paths m followed is $B = \{1, 2, 3, 4, 5\}$. The set of suspect nodes is $C = B - A = \{3, 5\}$.

When two segments m for message M arrive at node i with the same ID along different paths, the payloads of m either match or are different. If they do not match then some small trust deduction is merited; if they are the same then a small increase is merited.

Equation 5.6 shows how trust is reduced for the nodes in set C. Node i reduces direct trust for each $j \in C$; the updated direct trust is $\widehat{DT}^i_{j\,dec}$ while the value prior to update is DT^i_j. The penalty consists of three parts. The first part $(1 - DT^i_j)$ links the penalty to the direct trust node i has 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 standard penalty z by the number of possible culprits. Again the more there are, the more ambiguity, so the smaller the penalty. The final part $(1 + P_{A\,bad})$ again takes into account the number of nodes. The value for $P_{A\,bad}$ is the probability that a certain number of nodes in set A are bad. This takes into account both the size of the network and the current trust average for the network, p_{nx}. Table 5.1 shows the possible values for $P_{A\,bad}$ given the set B with $|B| = 5$ similar to the example in Figure 5.3. The value of $P_{A\,bad}$ used is based on the number of nodes in set C.

$$\widehat{DT}^i_{j\,dec} = DT^i_j - \left(\left(1 - DT^i_j\right) \times \frac{z}{a} \times (1 + P_{A\,bad})\right) \tag{5.6}$$

Equation 5.7 shows the original formula for trust increase. Similar to the

Table 5.1: Example Table $P_{A\,bad}$ for Given Figure 5.3 with $p_{nx} = 0.9$.

$\|B\|$	$\|A\|$	Probability Occurring	Permutations	Norm. Prob. ($P_{A\,bad}$)
5	1	$(p_{nx})^{(\|B\|-1)}(1-p_{nx}) = 0.06561$	$\binom{5}{1} = 5$	$\frac{0.32805}{0.40951} = 0.801$
5	2	$(p_{nx})^{(\|B\|-2)}(1-p_{nx})^2 = 0.00729$	$\binom{5}{2} = 10$	$\frac{0.0729}{0.40951} = 0.178$
5	3	$(p_{nx})^{(\|B\|-3)}(1-p_{nx})^3 = 0.00081$	$\binom{5}{3} = 10$	$\frac{0.0081}{0.40951} = 0.0198$
5	4	$(p_{nx})^{(\|B\|-4)}(1-p_{nx})^4 = 0.00009$	$\binom{5}{4} = 5$	$\frac{0.00045}{0.40951} = 0.0012$
5	5	$(1-p_{nx})^5 = 0.00001$	$\binom{5}{5} = 1$	$\frac{0.00001}{0.40951} = 0.00002$
$p_{tot} = 5(0.06561) + 10(0.00729) + 10(0.00081) + 5(0.00009) + 0.00001 = 0.40951$				

penalty there are three parts and $\widehat{DT}^i_{j\,inc}$ is the direct trust following the increase while the value prior to update is DT^i_j. The first is that the current direct trust is taken into account. The second, like above, $\frac{z}{a}$ only gives a small amount of trust. The third takes into account the network probability p_{nx}.

$$\widehat{DT}^i_{j\,inc} = DT^i_j + \left(DT^i_j \times \frac{z}{a} \times (1 + P_{A\,good})\right) \tag{5.7}$$

There are a couple of issues with Equations 5.6 and 5.7. The first is that the penalty and the reward result in slightly different values assuming that they occur in succession at the same node and no other events occur in between. Additionally, a good node can change to become malicious and in the current reduction scheme does not get a penalty until its trust is reduced below the given threshold. Since all nodes can be considered suspect, Equation 5.8 is used to decrease trust.

$$\widehat{DT}^i_{j\,dec} = DT^i_j - \left(\left(1 - DT^i_j\right) \times \frac{z}{a} \times \left(1 + (1-p_{nx})^{|B|}\right)\right) \tag{5.8}$$

Solving Equation 5.8 for DT^i_j results in Equation 5.9. Exchanging DT^i_j for $\widehat{DT}^i_{j\,inc}$ and $\widehat{DT}^i_{j\,dec}$ for DT^i_j gives the updated trust increase as Equation 5.10.

$$DT^i_j = \frac{\widehat{DT}^i_{j\,dec} + \frac{z}{a}\left(1 + (1-p_{nx})^{|B|}\right)}{1 + \frac{z}{a}\left(1 + (1-p_{nx})^{|B|}\right)} \tag{5.9}$$

$$\widehat{DT}^i_{j\,inc} = \frac{DT^i_j + \frac{z}{a}\left(1 + (1-p_{nx})^{|B|}\right)}{1 + \frac{z}{a}\left(1 + (1-p_{nx})^{|B|}\right)} \tag{5.10}$$

This change makes the penalty and reward system equal when two sequential events, one good and one bad occur, that include the same nodes. While this case is remote it is best that the process assumes all nodes are suspect and is symmetric.

5.1.2.3 Simulation Results for Direct Trust

In Chapter 4.4, a number of simulations were conducted utilizing NS3. The simulations done in this section again use the work done by *Lakkokorpi et al.* in [63]. Chapter 4.4.2 gives an overview of the NS3 object *DtnTrust* that was added to each *DtnApp* which was part of the original code published in [63]. There are a number of additions added to the *DtnTrust* object to 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 first addition is a method by which to track the path that a particular message segment m for message M traverses. In practice, this could be done with an additional header or footer or by modifying a field in one or the other; however, in NS3 only one header or footer object of the same class can be added to a packet. This left only one of two choices: make the header longer by adding optional header fields or add the path information into the packet payload. The latter was chosen because in the original DTN code, the packet payload is just a string buffer with all zeros. This approach worked more efficiently than adding more length to the header. It also made the messages or "packets" the same length between the simulations reported in this section and in Chapter 4.4. Different lengths likely would make minimal difference but it makes for a more controlled comparison.

The second addition is to update the *DtnTrust* object to distribute trust increases and decreases along the paths used to successfully recreate a message M. Each node stores all message segments m along with the path information. This is then used to update and distribute trust back along all paths as described in Section 5.1.2.1.

The final addition is to update trust when a message segment m for message M previously arrived at a node and that same segment arrives along a different

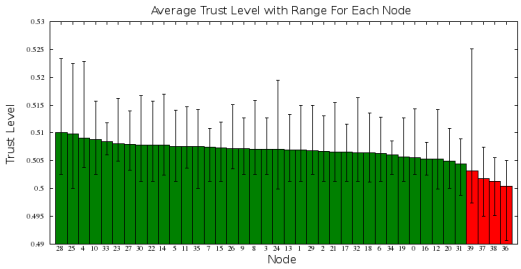
path. Like above, this is updated in the *DtnTrust*. Each node checks to see if it has seen the message segment m before and checks to see if the paths are different. If both conditions, hold then a check is done to see if they are the same. Depending on the outcome of that comparison, the trust is updated for all nodes along the paths as described in Section 5.1.2.2.

The same simulations as described in Chapter 4.4 were conducted. The same number of nodes with the same run times using the same random seeds. The idea is to show that distributing the positive and negative trust change yields better results than those obtained in Chapter 4.4.4.

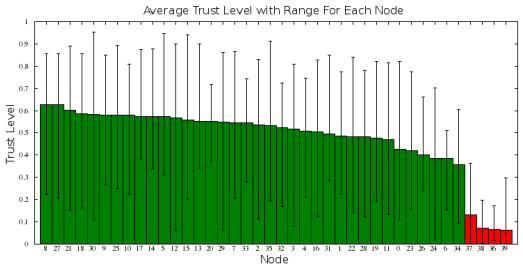
Figure 5.4 shows the power of full path knowledge. Subfigures 5.4a and 5.4b show the results with the fraction of nodes that act truthfully set to 90%, while Subfigures 5.4c and 5.4d show the results with this fraction set to 60%. Subfigures 5.4b and 5.4d show the results with full path knowledge, while Subfigures 5.4a and 5.4c are without.

There is a pronounced difference between trust assigned to good and bad nodes in Figure 5.4b versus those in Figure 5.4a. The drop down of trust from the least trusted good node and most trusted bad node is multiple times bigger than the differences between nodes in either group and the range of trust for the malicious nodes, between simulation runs, is small. Comparing between subfigure sets, the only negative aspect of the changes is that since full path knowledge is used the lowest runs for the good nodes are also lower. The results with 60% of trustworthy nodes are acceptable, but provide smaller differentiation between good and bad nodes due to the higher pollution created by bad nodes.

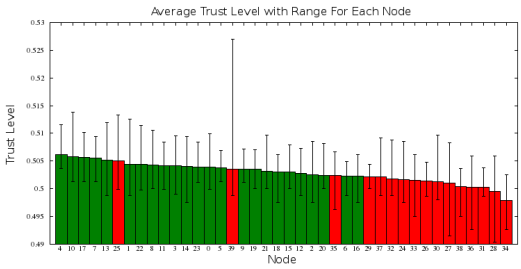
The results in this section only look at direct observations and the two proposed updates using path information discussed above. They represent the expected changes based on using full path information. The integration of indirect trust, discussed in the next section, as well as reducing network traffic to suspected nodes discussed in Chapter 6 refine these results.



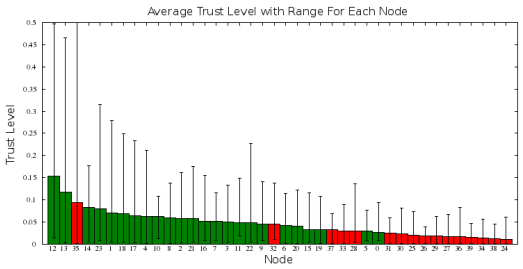
(a) 90% Path Trustworthiness, Chapter 4.4.4



(b) 90% Path Trustworthiness, Full Path



(c) 60% Path Trustworthiness, Chapter 4.4.4



(d) 60% Path Trustworthiness, Full Path

Figure 5.4: Node Trustworthiness Ranking $p_{nx} = 0.6,\ 0.9$

5.1.3 Trust: Indirect Trust Management

Nodes trade trust information as part of the handshake when they first meet and are in broadcast range. Figure 5.5 shows an example of the trust information stored in the buffers of node i. When node i meets node j, assuming that the two nodes are within broadcast range and have sufficient time to transmit, they conduct a handshake prior to forwarding any messages. During this handshake trust information is exchanged. Node i sends its aggregate trust vector (AT^i, Section 5.1.1) that is colored green and labeled "Aggregate" in Figure 5.5 to node j. Node j does the same thing.

Trading trust information allows each node to update trust based on recommendations from the other. For example, node i maintains a $(n+1)\times n$ indirect trust matrix that consists of trust vectors received from other nodes with an appended time stamp; the trust vector received from node j is shaded red in Figure 5.5. Each entry in node i's indirect trust matrix is a value between 0.0 and 1.0. Since all nodes claim to be trustworthy, the diagonal contains all 1's and is ignored for all future

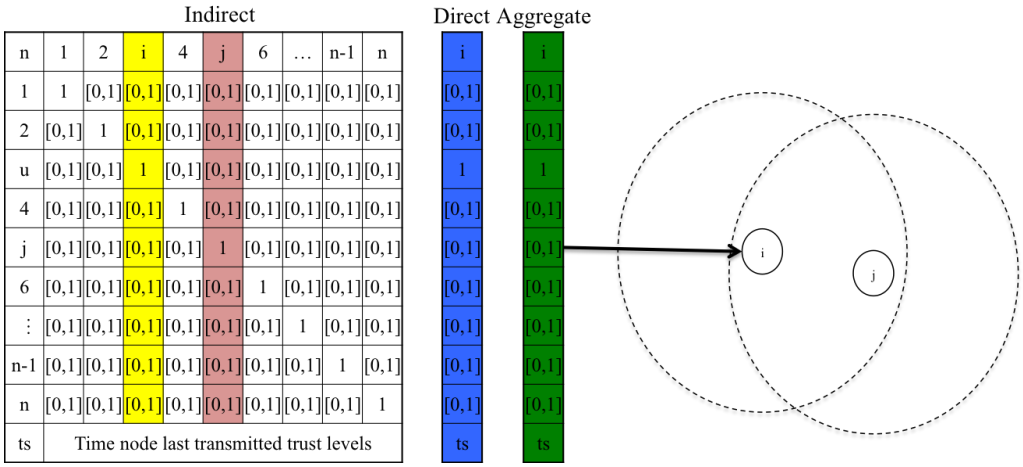


Figure 5.5: Storage at Node i

calculations. The column for i in the indirect trust matrix (shown in yellow) is the indirect trust vector (IT^i). In addition there is a direct trust vector in the buffer of node i. The direct trust vector (DT^i) is updated by direct trust observations made by node i (Chapter 4 and Section 5.1.2). Any trust decision made by node i are done based on the trust values in AT^i.

While indirect trust is utilized by all distributed trust management schemes in Chapter 3.2, the way in which it is handled varies. Since trust information is traded at all node meetings during the handshake, there are two ways in which the receiving node can process that information. The first is instantaneously and the second after a given time period designation Δt. Each is discussed in the subsequent sections. Some of the initial thoughts and concepts appear in [67] and are discussed in Section 5.1.3.1. Those ideas led to a more robust fusion process, appearing in [10], spawning two methods for managing indirect trust. The first uses matrices and is presented in Section 5.1.3.2. Section 5.1.3.3 is an approximation of the matrix process using vectors, exponential moving averages and a smaller buffer size.

5.1.3.1 Founding Concepts for Indirect Trust

The original concept for managing indirect trust stored one $n \times n$ matrix in the buffers of node i. The column for i stored direct trust information and the aggregate trust, designated AT^i. When node i receives a trust vector from node j, node i store that information in column j and immediately update column i using

the following equation for all node $w \in N$:

$$AT_w^i = AT_w^i - \left(AT_j^i\right)^2 \cdot \left(AT_w^i - tv_w^j\right) \tag{5.11}$$

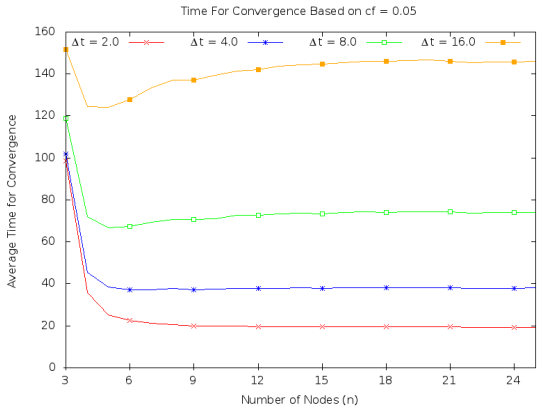
The idea is that the current trust level that node i has for node j is multiplied by the difference between the current trust node i has for each node w and what node j sends as it's trust vector. The current trust value node i has for node j is squared to give more weight to trusted nodes and the results replace the values in the column for i in the indirect trust matrix.

There are a couple of issues with the approach outlined above. The most obvious is that it does not take into account the trust vector of any node other than j. If there is a big discrepancy between node i and node j trust for some other node, this trust can be erroneously modified, especially if node j lies. This led to two key observations. The first was that direct trust and indirect trust needed to be managed separately prior to aggregation. The second was that more than one node should be used to modify indirect trust and by extension aggregate trust, which led to using a time interval for updates designated as Δt. While waiting to update indirect trust seems counter intuitive, it allows for a smoother change in trust values and the ability to identify those values that are outside the norm.

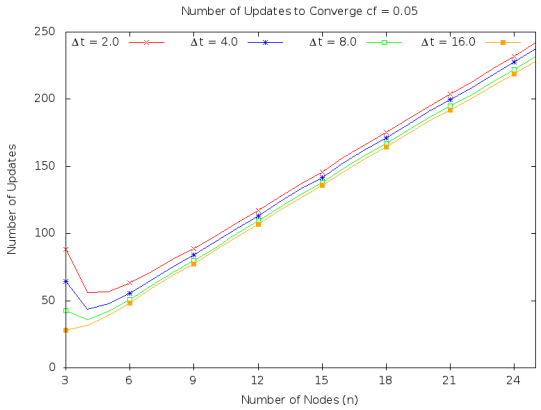
To minimize impact of potential lying and to better take into account discrepancies between node i and the trust vector for node j requires modification to Equation 5.11. To do this the change is calculated for each node $w \in N$, where N is the set of all nodes, as follows:

$$C_w^i = \left(AT_j^i\right)^2 \times \left(IT_w^i - tv_w^j\right) \tag{5.12}$$

This does not immediately update node i's indirect trust vector, but is maintained for a time interval Δt in the indirect trust matrix IT^i. 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 indirect trust vector using the Equation 5.13.



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



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

Figure 5.6: Effect of Δt

$$IT_w^i = IT_w^i + \beta \left(\frac{\sum_{f=1}^{|D|} C_w^f}{|D|} \right) \tag{5.13}$$

In addition, node i updates trust based on any discrepancy between its aggregate trust vector and the trust vector sent by node j directly after concluding the meeting (does not wait Δt). When node i receives the trust vector from node j one of two things occurs. The first happens when all values in the indirect trust vector for node i are within τ of all values in the trust vector from node j ($|\,(T_w^i - T_w^j)\,| < \tau$ for all $w \in N$). The second takes place if one or more of such values are not within τ. For the former case a small increase of trust is given to node j in node i's indirect trust matrix. For the later, trust is decreased for all w where $|\,(T_w^i - T_w^j)\,| \geq \tau$. Equation 5.14 defines the change in trust for node j and Equation 5.15 prescribes the change in trust for all other nodes that are outside τ.

$$IT_j^i = IT_j^i \times \left(1 - \frac{\alpha \times d}{2\,(|N| - 2)}\right) \tag{5.14}$$

$$IT_w^i = IT_w^i \times \left(1 - \frac{\alpha}{2d}\right) \tag{5.15}$$

The analysis done in this section led to a number of major requirements for

indirect trust. The first is that Δt is important and updating periodically instead of instantaneously minimizes the effect of nodes that are seen often and are suspect. The second is that aggregation of indirect trust is not trivial and if done naively like in Equation 5.11 can present a significant vulnerability. Third that decay is necessary and needs to be managed for both direct and indirect trust. Fourth that outliers need to be addressed. Fifth that while Equation 5.14 and 5.15 have merit as future research into trust pattern analysis there effect on indirect trust is minimal. All of these led to the changes proposed in the subsequent sections.

Figure 5.6 illustrates the first point above and are the results of a series of simulations conducted based on the principals in Appendix A. The simulation specifics are in Appendex B.1.1. Subfigure 5.6a is the convergence time of indirect trust for various values of Δt. Subfigure 5.6b shows that the number of updates required as n increases is linear and the selected value of Δt has minimal effect. This suggests that smaller values of Δt are better.

5.1.3.2 Trust Matrix

The first method for addressing the concerns from the previous section is to expand the information stored in the buffer for each node allowing for a more granular approach to managing indirect trust. Each node in the network 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 consisting of all nodes in the network. 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 indirect trust vector ($IT^k = C^k_k$). 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 discussed in Section 5.1.1 and the method for obtaining the direct trust

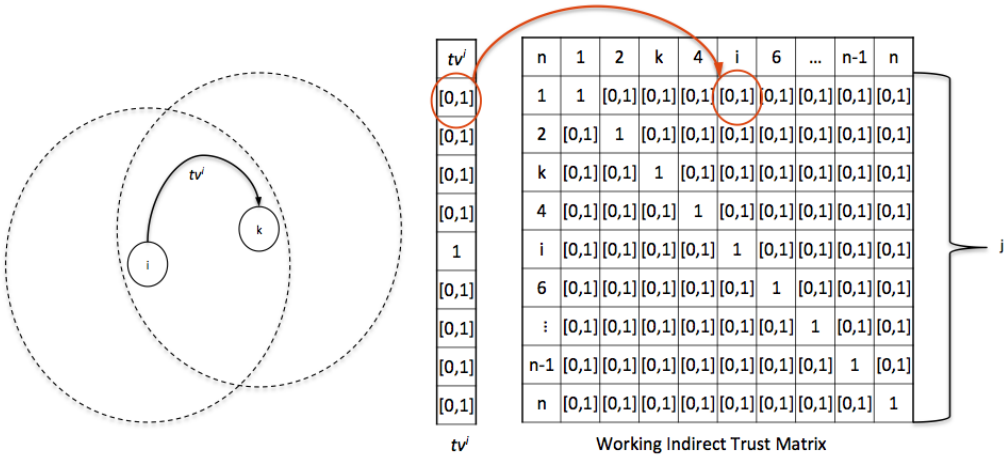


Figure 5.7: Meeting Event Between Nodes k and i

value is described in Chapter 4 and expanded on in Section 5.1.2.

There are two types of events. The first occurs when node k meets any node $i \in N$ and is designated as a meeting event. Figure 5.7 shows this type of event. The second event occurs when the time Δt expires triggering an update for node k's trust based on trust information received during the last time interval Δt. This is called an indirect trust update event.

During the handshake that initiates all meeting events node k and node i exchange 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 denoted as tv^i, where tv^i_j represents node i's 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:

$$W^k_{(i,j)} = \frac{\left(\left(W^k_{(i,count)} - 1\right) W^k_{(i,j)} + tv^i_j\right)}{W^k_{(i,count)}} \tag{5.16}$$

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

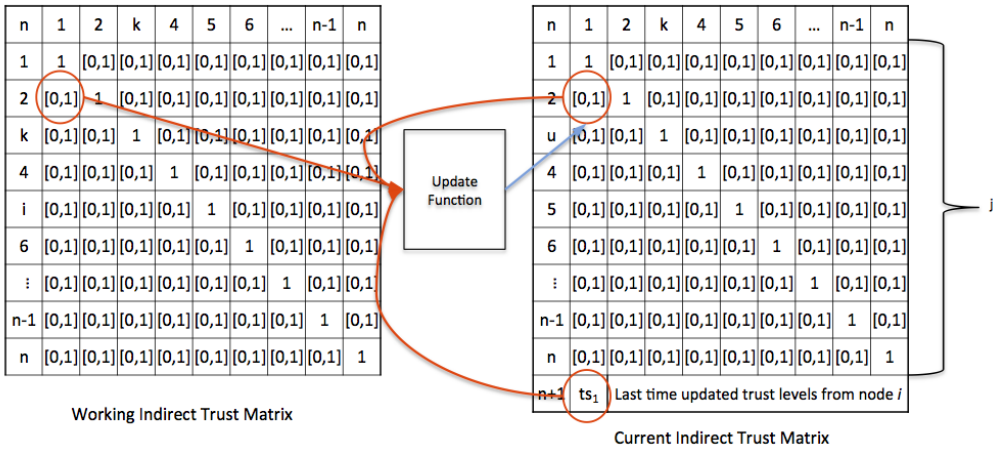


Figure 5.8: Node k Updates Current Trust Matrix when Δt Expires

another node.

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 indirect trust matrix for node k
2. Update the indirect trust vector for node k
3. Fuse the indirect and direct trust vectors to create the aggregate trust vector
4. Reset working indirect trust vector for node k

Updating the current indirect trust matrix is done by taking the average between the current trust matrix and update trust matrix, see Equation 5.17. 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 indirect trust matrix. This is used to compute $\alpha_{in} \in [0, 1]$, a "freshness factor" for the trust values. There are three cases in Equation 5.17 listed below.

$$C^k_{(i,j)} = \begin{cases} C^k_{(i,j)} & W^k_{(i,j)} = -1 \\ W^k_{(i,j)} & C^k_{(i,j)} = -1 \\ \alpha^{t^k_i}_{in} C^k_{(i,j)} + \left(1 - \alpha^{t^k_i}_{in}\right) W^k_{(i,j)} & \text{Otherwise} \end{cases} \tag{5.17}$$

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 working 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_{in} for all $j \in N$. Any value of $\alpha_{in} \geq 0.5$ 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, α_{in} should be set to 0.5. Equation 5.18 will find the number of time intervals since the last update. $C^k_{(i,ts)}$ is updated to the current time.

$$t^k_i = \lfloor \frac{currentTime - C^k_{(i,ts)}}{\Delta t} \rfloor \tag{5.18}$$

Once C^k is updated based on Equation 5.17, the indirect trust vector $C^k_{k,j}$ for all $j \in N$ is updated. This is done row by row using Equation 5.19. All values that are -1 are skipped in the summations in Equation 5.19. When $currentTime = C^k_{(i,ts)}$, that occurs when node i was met during the last time period, the value for $t^k_i = 1$. In Equation 5.17, 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.

$$C^k_{(k,j)} = \frac{\sum_{i=1}^{n} \left(C^k_{(i,j)} \times \alpha^{t^k_i}_{in} \right)}{\sum_{i=1}^{n} \alpha^{t^k_i}_{in}} \tag{5.19}$$

The thirds step is to fuse direct and indirect values to create an aggregate trust vector (see Section 5.1.1). 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.

5.1.3.3 Vector Approximation: Exponential Moving Average

The previous section uses 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 5.1.1. 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 sees 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.

$$WC_j^k = WC_j^k + \frac{\left(AT_i^k\right)^\beta}{2^{WN_i^k}} \tag{5.20}$$

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 $WN_i^k = 0$ 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 5.20 goes to 1.

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

$$WS_j^k = WS_j^k + \frac{\left(AT_i^k\right)^\beta}{2^{WN_i^k}} * tv_j^i \tag{5.21}$$

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

3. Update the working number of interactions: $WN_i^k = WN_i^k + 1$

When Δt expires for node k, it conducts a trust update. This is done by updating the indirect trust vector IT^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.

$$IT_j^k = (1-\gamma)\, IT_j^k + \gamma \left(\frac{WS_j^k}{WC_j^k}\right) \tag{5.22}$$

Equation 5.22 is used to find the Exponential Moving Average (EMA). The variable γ is used to determine the weight given to the new information (designated working here); $1-\gamma$ is the weight given previous EMA values (IT^k) and 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. In economics, that is the number of days to consider for smoothing price averages [68]. The smoothing factor is $\gamma = \frac{2}{previousVals+1}$. Many simulations below use $\gamma = 0.1$ this make $previousVal = 19$. Depending on the number of $previousVal$ taken into

Table 5.2: Simulation and Tuneable Parameter Crosswalk

Aggregate Trust - Direct Indirect Trust Integration (Section 5.1.1)			
P	Description	Sec	Fig
α_a	"Freshness" Factor for Aggregate Vector	5.1.2, 5.2.1	5.9
Δt	Time Period between Trust Updates	5.1.3.1	5.6
λ	Exponential Decay Factor	5.1.2	N/A
Indirect Trust - Matrix (Section 5.1.3.2)			
P	Description	Sec	Fig
α_{in}	"Freshness" Factor for Indirect Vector	B.1.2.1	B.1, 5.10
Indirect Trust - Vector Approximation (Section 5.1.3.3)			
P	Description	Sec	Fig
β	Weight given to current trust	5.2.3.1	5.12, 5.13a
γ	Weight given to previous time periods for a given node	5.2.3.2	5.12, 5.13b

account γ is increased to reduce the effect of older values and decreased to give them more weight.

5.2 Simulation Results

A number of simulations show the power of the scheme described above. This section provides simulation results utilizing a discrete event simulator as described in Appendix A.2 and fully described in Appendix B.1.2. Table 5.2 describes the key tuneable parameters (P) discussed in the previous sections and the sections and figures that show simulation results. Only the figures and their discussion are presented in this chapter. This is to clearly show the contributions and power of the proposed fusion method, much of the details for the simulation set-up and simulator runs are in Appendix B.1.2.

A matrix and a vector approximation discrete event simulator were written to test the tunable parameters. They both use the the trust aggregation methods discussed in Section 5.1.1 and listed in the first block of Table 5.2. The matrix version with specifics and simulation results are in Section 5.2.2 and Appendix B.1.2.1. The vector approximation is in Section 5.2.3 and Appendix B.1.2.2. A brief comparison between indirect trust aggregation methods is done in Section 5.2.4.

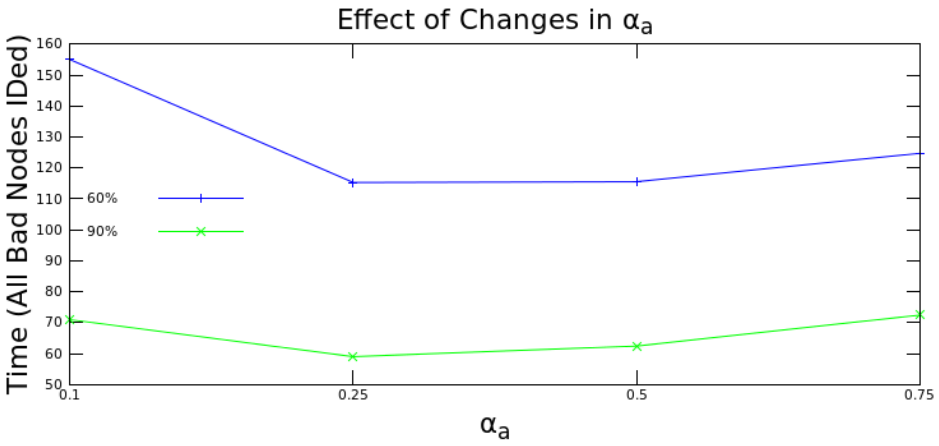


Figure 5.9: The Effect of α_a on Convergence Time

5.2.1 Aggregation Parameters

There are three variables used in Equations 5.1, 5.4, and 5.5. They are α_a, Δt and λ. The first parameter is the weight given to indirect trust in the aggregation process. The second is the time interval each node waits between trust aggregation. The final variable decays direct trust, over time, based on the last direct observation of a given node.

The variable Δt is discussed in Section 5.1.3.1 and Figure 5.6 shows how various values modify convergence time for indirect trust. The effect of λ is left for future work. The goal below is to show that both the matrix and vector approximation version of indirect trust management, fused with direct trust, will identify malicious nodes. To that end, Equation 5.1 is utilized to ensure isolation of the indirect trust variables.

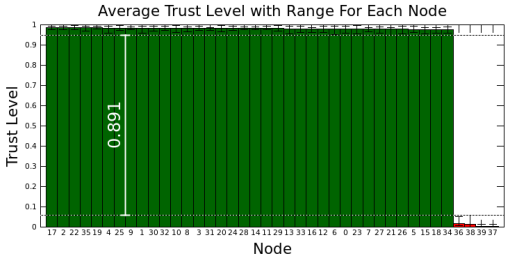
Figure 5.9 shows the results for different values of α_a using the discrete event simulator discussed in the next section. The only modification to the simulator is that, when each node successfully identifies all bad nodes in the network, the time is recorded. For identification, a bad node's trust must be less than 0.25. The results confirm the intuition that lower and higher values for α_a are not ideal. This makes sense because, if the weight given indirect trust is very low, the only way to converge is to make direct observations. Unfortunately, this takes time especially when some nodes are seen irregularly. On the flip side, if to much weight

is given recommendations, then trust will either erratically change or bad nodes can have a more significant impact. The line that represents 60% of the nodes acting good shows this. In the line representing 90%, this is less pronounced. This follows because direct and indirect trust are likely to be similar. The results for the following sections use $\alpha_a = 0.1$. The exact value for observing how trust changes in the long term is not effected by the exact value of α_a. Once this scheme is fully implemented in a more robust tool, the tuning of key aggregation values α_a, Δt and λ become an avenue for future work; especially since, different mobility patterns will effect their values.

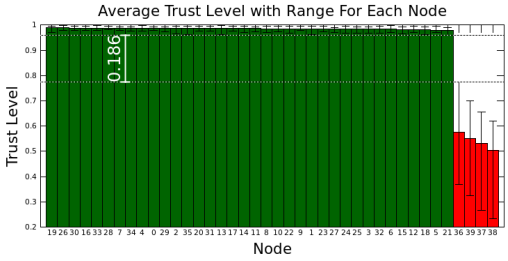
5.2.2 Matrix Version

The matrix version of the discrete event simulator is described in Appendix B.1.2. The network is set up and initialized based on the number of nodes. The values for all tuneable parameters, network settings, and simulation controls are user defined. The simulation and network controls include n number of nodes, number of iterations, simulation execution time, threshold for when a node is no longer trusted, percentage of good nodes, erasure coding variables (k_{ec} and s), time before sending an acknowledgement packet, and percent of time to act bad for a given malicious node. The direct and indirect trust integration variables of α_a, Δt, and λ and the indirect trust variable α_{in} are user defined as well.

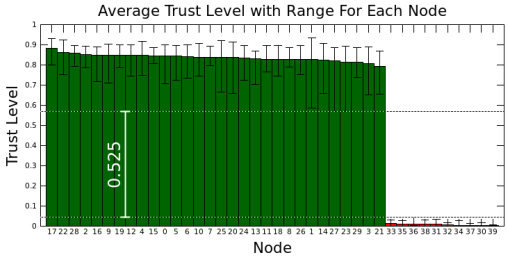
The only indirect trust variable for the matrix version for tracking indirect trust is α_{in}. A number of simulations were run to see the effect of α_{in} and the matrix version as a whole. Appendix B.1.2.1 shows the results of different values of α_{in}. The expectation was that over the long term they would have minimal effect. The convergence time is expected to changes based on α_i, but that is left for future work. As stated in Section 5.1.3.2, to give equal weight to consecutive time periods Δt then $\alpha_{in} = 0.5$. This is the expectation for most situations and the setting for all other results presented in this section. Table B.2 shows the settings for the simulations presented below and specifics about the simulator are in Appendix B.1.2.



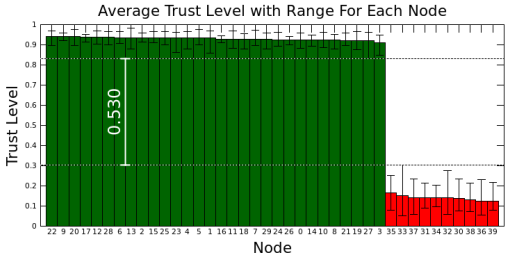
(a) Percent of Good Nodes = 90%; Always Act Bad



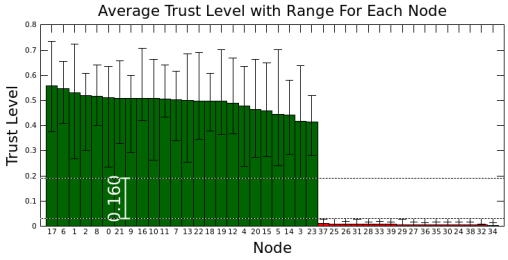
(b) Percent of Good Nodes = 90%; Flip Fair Coin



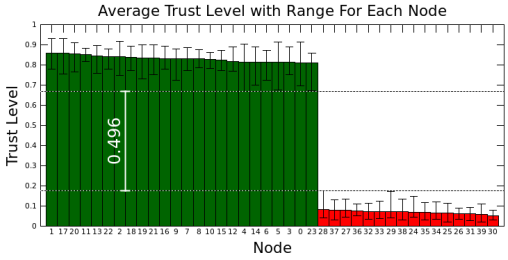
(c) Percent of Good Nodes = 75%; Always Act Bad



(d) Percent of Good Nodes = 75%; Flip Fair Coin



(e) Percent of Good Nodes = 60%; Always Act Bad

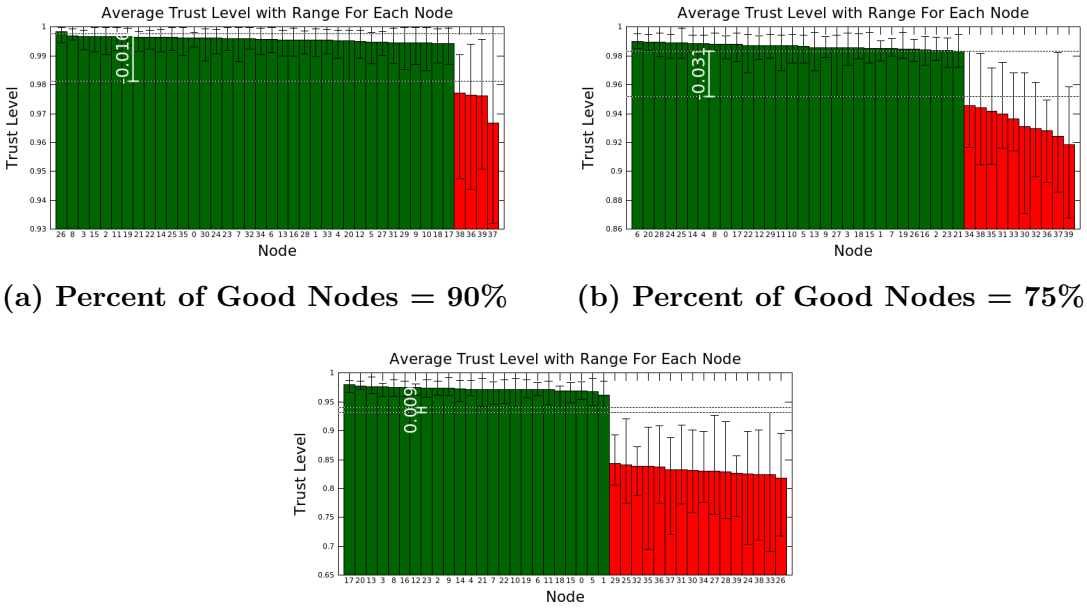


(f) Percent of Good Nodes = 60%; Flip Fair Coin

Figure 5.10: Sim Results: Various Percentage of Good Nodes, $\alpha_i = 0.5$

5.2.2.1 Fraction of Good Nodes = 90%:

Figure 5.10a shows the results where 90% of the nodes 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. This significant difference was expected because of the limited number of bad nodes. Figure 5.10b 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



(a) Percent of Good Nodes = 90% **(b) Percent of Good Nodes = 75%**

(c) Percent of Good Nodes = 60%

Figure 5.11: Sim Results: Bad Node Act Malicious 25% of the Time

the difference between worst good node and best bad is 0.186 versus 0.891. All malicious nodes are still clearly identifiable.

5.2.2.2 Fraction of Good Nodes = 75%:

Figure 5.10c shows the results where 75% of the nodes 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. Again, the expectation was a significant difference but smaller than the difference for 90%. Figure 5.10d 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 latter. All malicious nodes are still clearly identifiable.

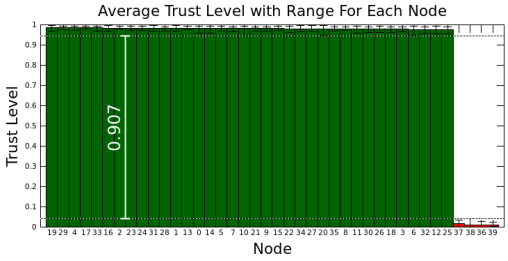
5.2.2.3 Fraction of Good Nodes = 60%:

Figure 5.10e shows the results where 60% of the nodes 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). As expected, there was a significant drop from the previous two sections especially since the number of bad nodes is closing in on 50%. Figure 5.10f 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.

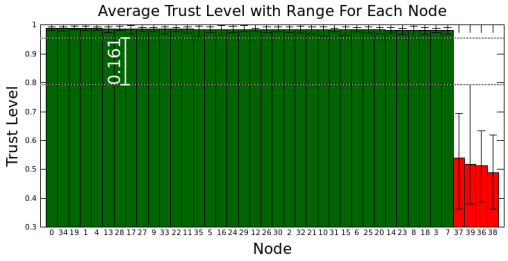
5.2.2.4 Conclusions on Variation of Bad Nodes

The 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 5.10b, 5.10d and 5.10f intermittently bad nodes fare better when there are more good nodes (90%) than 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.

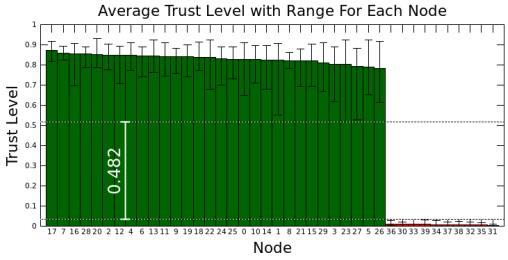
Additional simulations also confirmed that nodes acting bad 25% of the time are detectable (see Figure 5.11). Yet due to the distributed nature of trust, the difference between worst good and best bad becomes negative and the pronounced drop seen in Figure 5.10 is less apparent in Figure 5.11. As nodes 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 as well.



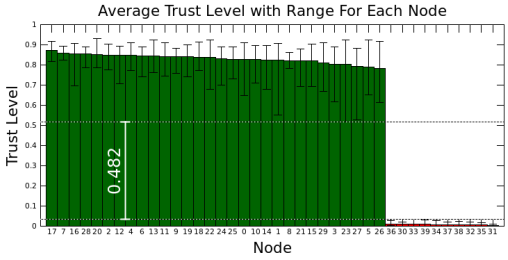
(a) Percent of Good Nodes = 90%; Always Act Bad



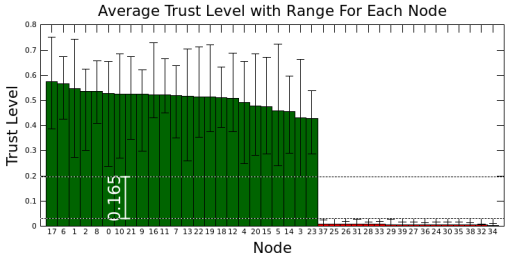
(b) Percent of Good Nodes = 90%; Flip Fair Coin



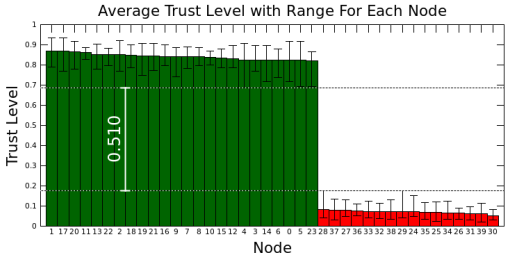
(c) Percent of Good Nodes = 75%; Always Act Bad



(d) Percent of Good Nodes = 75%; Flip Fair Coin



(e) Percent of Good Nodes = 60%; Always Act Bad

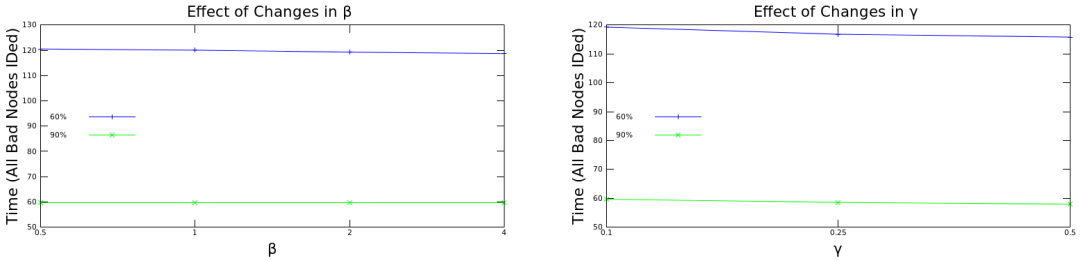


(f) Percent of Good Nodes = 60%; Flip Fair Coin

Figure 5.12: Sim Results: Various Percentage of Good Nodes, $\gamma = 0.1,\ \beta = 2.0$

5.2.3 Vector Approximation

Like in the previous section, the vector approximation version is tested using a discrete event simulator described in Appendix B.1.2. There are a number of user defined variables that set up the simulation network and are evaluated. Those variables are described in Table B.3. The two main variables explored in this section revolve around the effect of β and γ. The former modifies how much the current trust of a recommending node affects indirect trust changes (see Section 5.2.3.1).



(a) Effect of β **(b) Effect of γ**

Figure 5.13: Sim Results: Effects of β and γ

The latter is the weight given to older indirect trust values and is discussed in Section 5.2.3.2.

Figure 5.12 shows the results for various fractions of good nodes (Table B.3 variable *good*). For these simulations $\gamma = 0.1$, which gives exponentially decreasing significance to the last 19 time periods, and $\beta = 2$. The rational for choosing $\beta = 2$ is to compare similar values as to those used in the matrix version.

The results for Subfigures 5.12a, 5.12c and 5.12e are a very good approximation of those in Subfigures 5.10a, 5.10c and 5.10e, in Section 5.2.2, where bad nodes act malicious 100% of the time. Similar result comparisons hold when a bad node flips a fair coin to decide whether or not to act malicious (Subfigures 5.12b, 5.12d and 5.12f versus Subfigures 5.10b, 5.10d and 5.10f).

5.2.3.1 Indirect Variable β

Multiple values for β were used to determine its effect on the discovery of malicious nodes (Equations 5.20 and 5.21). The simulator in Appendix B.1.2 is used with one modification. Once all nodes can identify every bad node in the network, the time stamp is stored. If node k's trust of node i drops below 0.25, node i is considered bad from node k's perspective. Once every node $k \in N$ does this, the time is recorded. Figure 5.13a shows the varying values of β, where $gamma = 0.1$ is static, and the change in time for both 60% and 90% of the nodes in the network being good. There is little change; however, in a more polluted network, where 60% of the nodes are good, the time slightly decreases. This is expected because nodes with lower trust will be given less weight in the indirect calculations. This will cause

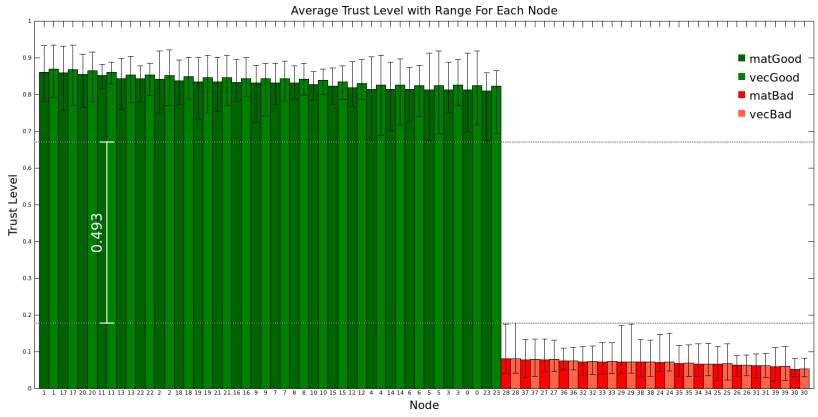


Figure 5.14: Comparison Between Vector and Matrix, Percent Good = 60% (Bad Node Flips a Fair Coin to Determine Malicious Action)

good nodes to erode their overall trust more quickly. This would be less pronounced when fewer bad nodes are in the network.

5.2.3.2 Indirect Variable γ

Similar to the section above, a number of different values for γ are tested. The simulator is initialized and the time is calculated in an identical manner. Figure 5.13b show the results. The key range is $0.0 \leq \gamma \leq 0.667$. When γ is close to zero, then an infinite number of previous values are used. When $\gamma = \frac{2}{3} \approx 0.667$, then the last two are used. When there is a large number of trust modifications in the network, as appears in the simulations, putting more weight on older values slows the convergence process. That is why it is slightly better to use a higher value for γ in the specific scenarios presented. Due to the random mobility, initialization of the network, amount of trust change and the limitations of the simulation tool, fine tuning of γ is left for future work.

5.2.4 Comparison

Figure 5.14 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.

5.3 Conclusions

There are a number of contributions added in this chapter. It proposes a robust distributed trust management scheme that identifies malicious nodes. These nodes are even identifiable when they act truthfully 75% of the time. This is done by expanding the use of path redundancy as a direct clue to include path information and by properly fusing both direct and indirect trust to produce an aggregate trust value.

There are two additional proposals that use path information. The first is to distribute trust increases and decreases along the full path and not just to directly connected nodes. The second is to use the **SegMatch** function that compares segments arriving from different paths and giving a small increase or penalty if a node receives multiple copies. For this dissertation, it is assumed that all nodes properly add path information to all segments that flow through them. Additional direct clues about path information could be identified if nodes are untruthful. This is a possible direction for future work.

The use of indirect trust is shown to converge as discussed in Section 5.1.3. The integration of trust is discussed in Section 5.1.1. Two methods for buffering and integrating indirect trust are proposed. The matrix version keeps more information and the vector approximation requires less buffer space. Each is shown to manage indirect trust well.

The simulations in Section 5.2 show that, by using direct and indirect clues and properly fusing them, malicious nodes are identified. By running the proposed trust management scheme, each node in a DTN maintains trust for all other nodes. This trust is dynamically updated based on direct observations and indirect information through traded trust vectors. This is a value between 0.0 and 1.0 and can be used to make routing and security decisions. The focus of the next chapter is to propose such a Trust Based Secure Routing (TBSR) protocol.

CHAPTER 6
Trust Based Secure Routing Protocol (TBSR)

The previous two chapters explored direct observations and their relation to establishing trust. They also proposed a distributed trust management scheme for use in Resource Constraint Networks. This chapter utilizes the research done to explore and propose a Trust Based Secure Routing (TBSR) protocol for use in a RCN, in general, and DTN in particular. While it uses the trust value obtained through the trust management process previously discussed (Chapter 5), it is a stand alone protocol that can work with any distributed trust management scheme that accurately manages trust as a value between 0.0 and 1.0.

Because there are no end-to-end routing tables in a DTN, all messages are sent from source to destination with the goal of using the best path possible. An omniscient node at every hop would pass the message only once to the next node along the path that ensures the fastest delivery, and not accept a message unless it is on that path. But even in this unrealistic scenario, where all nodes are omniscient, that path the provides the fastest delivery is unlikely to require the minimal number of hops. There is a trade-off between energy use and delivery time. Depending on the node, energy use could be more important than message delivery time. There is also a security trade-off. A message that flows through fewer nodes is less likely to encounter a malicious node. Since global knowledge is unavailable in a distributed network such as a DTN, in order to maximize message delivery time, endemic routing, where all node interactions trade all buffered messages, is required. In order to maximize security and minimize energy use, a routing protocol that only forwards to the destination, source waits for source-destination meeting (WSDM), is required. In practice, neither approach allows flexibility. The former, significantly increasing security and energy use, and the latter reducing delivery time and ratio as some nodes rarely, if ever, meet.

This chapter is to appear in: T. Babbitt and B. K. Szymanski. "Trust based secure routing in delay tolerant networks," in *8th IEEE Int. Workshop on Network Sci. for Commun. Networks (NetSciCom 2016)*, San Francisco, CA, Apr. 2016.

There are a number of approaches to routing in a Delay Tolerant Network aiming to limit the number of copies of a message by selectively sending that message to only those met nodes that are either likely to meet the destination earlier then the sender or are highly trustworthy. Each approach resolves differently the trade-off between energy and security, on one hand, and message delivery delay time on the other. Most first generation routing protocols attempted to minimize energy by limiting the number of messages sent. One such approach is Spray and Wait [46] where a node sends a limited number of copies of a message. It waits a time period and then if the message is still undelivered, additional copies of the message are sent. This approach does limit energy use and has negligible effect on delivery delay under many mobility scenarios.

Other approaches attempt to determine which nodes are "closer" to the destination. Many use community structures to make this determination; others attempt to predict which node will make contact with the destination first. The former has merit, because nodes in the same community as the destination are more likely to meet the destination node than those outside. The latter takes the benefit of the expectation that a node with smaller average meeting time of destination will indeed meet it before the forwarding node. Hence, these determinations are based on social structure and node movement patterns, respectively. A number of social structure based protocols are outlined in [5], [69], and the authors in [70] use friendship relationships. Many are shown to be able to reduce the number of copies of a message and limit resource use while marginally affecting deliver rate and time.

Most of the protocols listed above provide some inherent security through limiting exposure of a message to nodes, but are not based on the use of observations to build and update trust, which can then be used to make routing decisions. In addition to the trust management scheme proposed in Chapter 5, Chapter 3.2 gives an overview of a number of trust schemes proposed for use in a DTN. Any routing protocol in a network where end-to-end routing tables are unfeasible, such as in a DTN, can benefit from some trust metric to assist in making distributed routing decisions. One such approach in [33], [71] uses two time periods per message. Only those nodes above a certain trust threshold are used during the first time period. If,

and only if, a node fails to receive an acknowledgement packet from the destination prior to the expiration of a given time period are less trusted nodes used.

Using trust as a metric to make forwarding decisions also allows for the use of different message classifications. Normally these classifications are used to determine quality of service (QoS). A good discussion on distributed QoS is found in [72], where the main issue is finding a path that allows for the quickest delivery and giving preference to messages with a higher QoS classification. Many businesses and the military use classification of documents to dictate how they are handled (stored, created, destroyed) and who has access [73]. A message classification system for a distributed system must take into account both delivery time to destination, and who potentially should have access to, a given message when determining message classification and selecting risk parameters.

In theory, the used of trusted nodes first, and less trusted nodes only in a panic scenario seems appropriate as outlined above and in [33], [71]. This chapter proposes a protocol that takes a more granular approach to better select forwarding nodes using trust, estimated time to meet the destination and number of copies of a message that a given node forwards. The Trust Based Secure Routing (TBSR) protocol is described in Section 6.1, and a message classification scheme is proposed in Section 6.2. Simulation comparisons are provided in Section 6.3 and conclusion and future work discussed in Section 6.4.

6.1 Trust Based Secure Routing Protocol (TBSR)

The Trust Based Secure Routing Protocol for use in Delay Tolerant Networks is a routing protocol that limits the exposure of a message to nodes by reducing the copies of a message sent, while ensuring timely delivery by using nodes that are "closer" to the destination and avoiding using those nodes with low trust that are therefore more likely to be malicious. TBSR is designed to work with any trust management scheme that converges to, and maintains, a valid distributed trust. The trust value should be normalized to a range between 0.0 and 1.0 where 0.0 is complete lack of trust and 1.0 is complete trust. The lower the value the higher the likelihood of a node being isolated or bypassed when forwarding decisions are made.

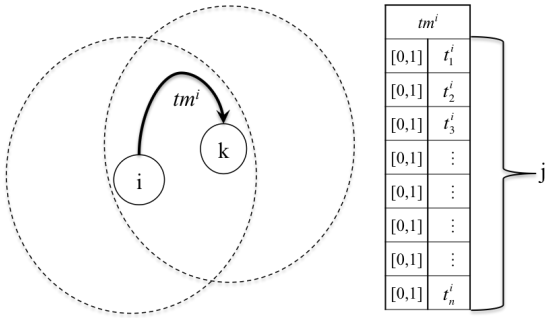


Figure 6.1: Trust Information Sharing as Part of Handshake

In a DTN, when node i and node k meet, they first conduct a handshake to establish a communication link. While TBSR does not put any limitation on protocol used to establish this link, most implementations create a TCP or UDP connection between the nodes. Each node party to the meeting must decide which, if any, messages to forward to the other node. The trust management scheme is often included as the final part of the handshake. Once a connection is established, exchange of trust information, often, like in this system, in the form of a trust vector occurs. This consists of node $i's$ trust value for all nodes in the network and vice-verse.

Figure 6.1 depicts the final step of the handshake between node i and node k. Node i is sending its trust matrix to node k. The trust vector (per Chapter 5.1.3) is extended to a $n \times 2$ matrix, designated tm^i, which includes the estimated times at which node i will meet all other nodes in the network. An individual time estimate given by node i is t_j^i for a given node j. With a valid trust and an estimated next meeting time, node k can make a decision on whether or not to forward a given message segment M_x to node i. If node k has multiple message segments to forward, it might decide to forward all or some of them. Estimated time to the destination is used and defined as the estimation of when a node will directly meet the destination. Another valid approach that merits additional research is the modification of that time based on social interaction or network structure, an example being the use of community affiliation tags.

6.1.1 Determining Situation Risk in Forwarding Decisions

Once node k completes the handshake with node i, it then must decide which message segments it has buffered to forward. This decision is made based on risk assessment (see Section 6.1.2) and on node k's trust of i, as well as both nodes estimated destination meeting time. The current trust node k has for node i is a value between 0.0 and 1.0. The time estimated until node k meets node j, the destination, is denoted t_j^k. The time estimate node i takes until it meets the destination is t_j^i. If node k has low trust for node i, then it should only forward a message if node i will meet the destination significantly faster than node k. Conversely, if node i has a low trust of node k, it should limit the number of messages it accepts from node k.

6.1.1.1 Overview

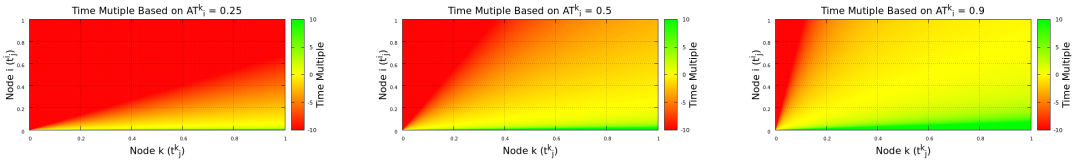
Node k adjusts all time estimates it receives from node i based on its current trust of node i (AT_i^k). Equation 6.1 uses the information that node i provides and then divides that by the current trust node k has for node i. Since $0.0 \leq AT_i^k \leq 1.0$, node k will increase node i's destination meeting time estimates. The more node k trusts node i the smaller the increase.

$$t_{adj(i,j)}^k = \frac{t_j^i}{AT_i^{k\,2}} \tag{6.1}$$

Once node k calculates trust adjusted meeting time between node i and the destination, it determines how many times it is better to either wait or forward. Equation 6.2 returns that value.

$$tm_{(i,j)}^k = \frac{t_j^k - t_{adj(i,j)}^k}{min\left(t_j^k,\ t_{adj(i,j)}^k\right)} \tag{6.2}$$

If Equation 6.2 yields negative values, then it is better for node k to wait and not forward the message. Otherwise, it is generally better to forward the message. This only takes into account the time multiples, assumes that node i is not lying and that the expected time to meet node j is valid. Ideally, node k would only forward a message if it is not likely that the recipient is malicious.



(a) $AT_i^k = 0.25$ **(b)** $AT_i^k = 0.5$ **(c)** $AT_i^k = 0.9$

Figure 6.2: Time Multiple for Node k Varying Trust of Node i (AT_i^k)

$$sr^k_{(i,j)}(M_x) = \begin{cases} {AT_i^k}^{w_{nc} \cdot nc+1} & -1 \leq tm^k_{(i,j)} \leq 1 \\ \left({AT_i^k}^{w_{nc} \cdot nc+1}\right)^{\frac{1}{tm^k_{(i,j)}}} & tm^k_{(i,j)} > 1 \\ \left({AT_i^k}^{w_{nc} \cdot nc+1}\right)^{\left|tm^k_{(i,j)}\right|} & tm^k_{(i,j)} < -1 \end{cases} \tag{6.3}$$

Equation 6.3 takes into account three cases and returns a value between 0.0 and 1.0 that is the "Situational Risk " (SR). The first case is the most likely and occurs when $-1 \leq tm^k_{(i,j)} \leq 1$. This means that based on trust adjusted time (Equation 6.1), there is less than one time interval difference between the trust adjusted and expected time that node i or node k will meet the destination. In this instance, the trust that node k has for node i is raised to the power equal to the number of copies of the message node k previously sent, plus one to account for sending a copy to node i $(nc + 1)$. The number of copies (nc) is multiplied by a weight $0.0 \leq w_{nc} \leq 1.0$ that adjusts the effect of the number of copies on SR. The second case occurs when, based on time difference, it is significantly better to forward, $tm^k_{(i,j)} > 1$. This will cause Equation 6.3 to produce the values higher than the trust, but smaller than 1.0. The final case occurs when $tm^k_{(i,j)} < -1$ and it is significantly better to wait causing $sr^k_{(i,j)}(M_x)$ to be a small fraction of trust and it tends to zero as trust decreases.

6.1.1.2 Analysis

There are three key, and one supplementary, variables that affect whether node k forwards a message to node i using Equation 6.3. These include node k's trust of node i (AT_i^k), the number of times node k has forwarded a message (nc) and the number of time intervals by which the time delay increases or decreases by holding

versus forwarding a message segment (Equation 6.2). The weight of the message count is modified by (w_{nc}). Valid values are $0.0 \leq w_{nc} \leq 1.0$; the smaller the value of w_{nc} the less weight is given to the number of copies. When $w_{nc} = 0.0$ then the number of copies has no effect on SR. Unless otherwise stated, $w_{nc} = 1.0$. The first two key variables are based on information stored in the buffers of node k and while they ultimately effect the "Situational Risk" ($sr^k_{(i,j)}$), they are set values used in every SR calculation. Equation 6.2 yields significantly different values based on the times sent during the handshake (Figure 6.1).

The significant range of values for $tm^k_{(i,j)}$ in Equation 6.2 are $-10 \leq tm^k_{(i,j)} \leq 10$. As an example, if $AT^k_i = 0.1$, a low trust, and $tm^k_{(i,j)} = 10$ then $sr^k_{(i,j)}(M_x) = 0.794$, given node k has never forwarded the segment before ($nc = 0$). This makes it likely that node k will forward the message. If $AT^k_i = 0.9$, a high trust, and $tm^k_{(i,j)} = -10$ then $sr^k_{(i,j)}(M_x) = 0.349$, given node k has never forwarded the segment before ($nc = 0$). Node k will likely not forward the message.

Figure 6.2 shows a comparison of results for Equation 6.2 where node k has various trust levels for node i. The x-axis for each heat map is the node k estimate of the time by which it will next meet the destination node j, and the y-axis is node i's estimate of such time. The color is the z-axis that represents time corresponding to multiple values of ($tm^k_{(i,j)}$) computed from Equation 6.2. Subfigures 6.2a, 6.2b and 6.2c show $tm^k_{(i,j)}$ with $AT^k_i = \{0.25,\ 0.50,\ 0.90\}$ respectively. The area in each subfigure that is shaded red represents values that will make $sr^k_{(i,j)}$ smaller, while the green shaded area contains values that do the opposite. The area shaded yellow covers values close to $tm^k_{(i,j)} = 0$ that have minimal effect on $sr^k_{(i,j)}$.

Any chosen metric that guides forwarding must have the property that the lower a given nodes trust, the bigger advantage it must have in reduction of delivery time to the destination. There are a number of reasons for this, to include that a low trusted node is more likely to lie, or that the link is faulty and there is a higher chance that the message is lost. Equation 6.3 is designed to have these properties that become evident when node k's trust for node i is low ($AT^k_i < 0.5$). Subfigure 6.2a is mostly red and since the trust is low, this ultimately lowers the SR and essentially eliminates nodes with trust values from this area from contention

for forwarding. Even the trust values from the green area are unlikely to result in forwarding, because node i has to expect approximately 1000 times faster delivery than node k for forwarding to happen. The motivation for requiring such vast improvement is that a node with low trust is more likely to lie to get preferential treatment. Subfigures 6.2b and 6.2c have a larger area that is yellow and only when a node is significantly faster or slower in meeting the destination is the forwarding decision affected by the value of $s^k_{(i,j)}$. This is the expectation for efficient routing that should forward data to trustworthy nodes, excluding those that are not likely to meet the destination more quickly than the sender.

6.1.2 Assessing Risk in Forwarding Decision

The previous section analyzed the "Situational Risk" (SR) associated with forwarding a given message from node k to node i (SR varies between 0.0 and 1.0). The higher the value, the lower the risk of forwarding the message. Nodes with low trust are excluded unless they are significantly better. Nodes with high trust are likely to receive messages for forwarding unless their expected time to meeting the destination is larger then that of the sender. This leads to a number of potential algorithms (Figure 6.3) to decide on whether node k forwards a message to node i. In all cases, the node starts with determining $sr^k_{(i,j)}(M_x)$ using Equation 6.3 and then:

1. uniformly randomly picks a number [0.0,1.0]. If that number is less than $sr^k_{(i,j)}(M_x)$ the message is forwarded (Figure 6.3a).

2. if both $sr^k_{(i,j)}(M_x)$ and node k's trust for node i (AT^k_i) is above a threshold the message is forwarded (Figure 6.3b).

3. a combination of 1 and 2 is used (Figure 6.3c).

The first solution (Figure 6.3a) allows all nodes to have a chance to forward a message and does not completely "blacklist" a node. This allows for the distributed trust management scheme to continue to update trust. Thanks to this design, isolated nodes or nodes with intermittent signal can build or update trust. The

Input: $\left(t_j^i,\ t_j^k,\ M_x\right)$, (time i meets dest, time k meet dest, Segment)
Output: Boolean, TRUE to forward and FALSE to wait

Solve $sr_{(i,j)}^k(M_x)$; // Equation 6.3
$r = random(0.0..1.0)$;
if $sr_{(i,j)}^k(M_x) \geq r$ **then**
  return $forward = TRUE$; // Forward
else
  return $forward = FALSE$; // Wait

(a) Random Forwarding Decision Algorithm

Input: $\left(t_j^i,\ t_j^k,\ M_x\right)$, (time i meets dest, time k meet dest, Segment)
Output: Boolean, TRUE to forward and FALSE to wait

Solve $sr_{(i,j)}^k(M_x)$; // Equation 6.3
if $sr_{(i,j)}^k(M_x) \geq riskThresh$ **then**
  return $forward = TRUE$; // Forward
else
  return $forward = FALSE$; // Wait

(b) Risk Threshold Forwarding Decision Algorithm

Input: $\left(t_j^i,\ t_j^k,\ M_x\right)$, (time i meets dest, time k meet dest, Segment)
Output: Boolean, TRUE to forward and FALSE to wait

Solve $sr_{(i,j)}^k(M_x)$; // Equation 6.3
if $sr_{(i,j)}^k(M_x) \geq Tr_r$ *and* $AT_i^k \geq Tr_r$ **then**
  return $forward = TRUE$; // Forward
else
  $r = random(0.0..1.0)$;
  if $sr_{(i,j)}^k(M_x) \geq Tr_r$ *and* $r \leq Tr_a$ **then**
    return $forward = TRUE$; // Forward
  else
    return $forward = FALSE$; // Wait

(c) Random Risk Threshold Forwarding Decision Algorithm [11]

Figure 6.3: Algorithms for Forwarding Decisions in TBSR

Input: $\left(AT_i^k,\ M_x\right)$
Output: Boolean, TRUE to accept and FALSE to clear from buffer

```
1 r = random(0.0..1.0);
2 if AT_i^k ≥ T_{r_r} or r ≤ T_{r_a} then
3 |   return recieve = TRUE ;                    // Store in Buffer
4 else
5 |   return recieve = FALSE ;                  // Delete From Buffer
```

Figure 6.4: Algorithm: Message Reception Decisions in TBSR

downside of this approach is that it may give malicious nodes a greater chance of receiving a message. This disadvantage could easily be mitigated by applying this approach selectively to messages with low importance.

The second solution (Figure 6.3b) forwards messages to all nodes with trust above a threshold that are also expected to meet the destination soon. The downside is that any node below a set threshold is "blacklisted" and does not have an opportunity to modify its trust. This can have a significant effect on nodes that are isolated for a time period. Trust decays over time and that could cause a given node to go below the trust threshold.

Pseudocode for the third solution is shown in Figure 6.3c and allows for both a threshold and nodes with lower trust to forward message segments. If the SR is above a set threshold (Tr_r), and the trust that node k has for node i is above that same threshold, then the message is forwarded. If the SR is above the threshold because node i will meet the destination soon, but node k's trust in node i is low, then depending on the adaptive threshold (Tr_a) the message is forwarded or not. This gives a node with low trust a chance to increase trust, but it does so in a manner that limits the overall network risk.

6.1.3 Assessing Risk in Decision to Recieve a Message

Up to this point, the focus of Trust Based Secure Routing Protocol (TBSR) has been on making the best decision to forward based on "Situational Risk." In addition to forwarding, a given node has to decide whether or not to receive a message. This decision is usually assumed away. If a node is willing to forward

then the receiving node is willing to accept it. If the sending node broadcasts that message then the receiving node will, at a minimum, process it. The decision the receiving node has is whether or not to buffer the message and forward it. If there is a high enough chance that it is corrupt, then the message probably should be ignored.

There are a number of approaches similar to the previous section as follows:

1. Use a threshold and only accept messages from a node with a trust above that threshold.

2. Randomly choose which nodes to accept messages from based on trust.

3. Combination of both

The first approach creates a good base. Any node below a certain trust threshold should, in most instances, be ignored. The downside is that it "blacklists" all nodes with lower trust. These trust values can be from malicious activity and hardware/isolation. In order to maintain dynamic trust, some nodes with lower trust need to interact and be allowed to forward a message.

The purely random approach allows for trusted nodes to be bypassed and, hence, was not seriously considered. This approach will more likely than not accept a message from a good node, but it should in all situations. This also gives a chance that a message from a suspect node is accepted. While this is not an ideal outcome, a suspect node needs a chance to improve it's trust and it can only do so if it can be part of the message flow in the network.

The final approach was implemented and it was a combination of the previous two. Figure 6.4 gives psuedocode for this approach that automatically accepts messages from more trusted nodes and, like in the previous section, allows suspected bad nodes to be utilized. There could be situations where the only path from source to destination is through a node with lower trust.

6.2 Message Classification Scheme

There are two parts to the proposed message classification scheme. The first is a security component and the second is a delivery component. The former is based

Table 6.1: Message Classification

Security \ Delivery	Low	Routine	High
Top Secret	$T_{r_a} = 0.0$, $w_{nc} = 1.0$	$T_{r_a} = 0.0$, $w_{nc} = 0.75$	$T_{r_a} = 0.1$, $w_{nc} = 0.5$
Secret	$T_{r_a} = 0.1$, $w_{nc} = 1.0$	$T_{r_a} = 0.1$, $w_{nc} = 0.75$	$T_{r_a} = 0.1$, $w_{nc} = 0.5$
Unclassified	$T_{r_a} = 0.1$, $w_{nc} = 1.0$	$T_{r_a} = 0.1$, $w_{nc} = 0.5$	$T_{r_a} = 0.25$, $w_{nc} = 0.5$

on document classifications similar to those in [73] and the latter based on delivery [72]. For clarity, there are three levels for each. For security, the classifications are unclassified, secret and top secret. For delivery, they are low, routine, and high.

Table 6.1 presents a proposed method for assigned T_{r_a} and w_{nc} based on message classification for use in tuning TBSR. The values were chosen based on simulation analysis; however, they are tunable based on different network configurations. Additional research could provide a more complete analysis on different scenarios and the proper values to use in those cases. T_{r_a} dictates how often untrusted nodes are used. When that value is zero, no untrusted nodes are utilized. For a message with high security needs, that is paramount, but it comes at a cost in delivery time. The parameter w_{nc} is that weight given the number of copies of a message. When that is zero, then the number of copies has no effect on SR. When more security is required, w_{nc} should be maximized. Again, there is an inverse relationship with delivery time. While specific message tracking simulations that utilize this proposed classification scheme are not implemented, Table 6.1 provide a proposed baseline. Flushing out the interactions of nodes based on message classification is one proposal for future work.

6.3 Simulation Comparisons

In order to illustrate the benefits of TBSR, a number of simulations were conducted to determine improvements in security and the cost of doing so. Using an endemic routing scheme is expected to produce the lowest delay. Since every node met receives a copy of the message, the fastest path is used. Of course, the risk of data compromise is maximized. Any routing protocol that limits the number of copies of a given message will increase message delay, but lower the risk of data

compromise. As long as that delay does not hinder an application, it is better to be selective in sharing a message segment with met nodes.

Risk increases with each message segment received by a bad node since there is a chance that it can be used for malicious purposes. Even in a scheme that uses erasure coding, where multiple segments are required for message recreation, or in situations where the payload is encrypted, once the adversary has the message it can potentially break the encryption or glean useful information just knowing about the source and destination and how often they communicate. Our goal is to limit the delay, while reducing the number of instances where a bad node receives a message segment (selective forwarding also decreases the number of broadcasts caused by message transmission and therefore lowers the energy used for communication, a positive side effect of this solution). Hence, the solution seeks a trade-off between security but large average delays due to limited message sharing, and speed but low security of more promiscuous sharing.

6.3.1 Simulator Overview

A discrete event simulator using the principles outlined in [74] and Appendex A show the value of TBSR. There are three types of events that are managed by the simulation event queue: meeting, update and message initiation. A meeting event occurs when node i and node k are within broadcast range, conduct a handshake and as protocol dictates, trade message segments. A node update event occurs every Δt time period. This is user defined when tuning up performance of trust management. A message initiation event occurs when a node places a message in it's buffer for transmission to another node which is in it's vicinity. The user can designate how often a node sends a message. By default, that message is sent randomly to another node in the network.

6.3.1.1 Simulation and Network Initialization

Each node is initialized with a vector consisting of the Aggregate Trust (AT) for all other nodes in the network with a value between 0.0 and 1.0. The user defines the fraction of good nodes in the network. Since our simulation represents a more mature network, where trust has partially converged due to previous network

activity, during simulation initialization the AT for nodes designated as bad are assigned a trust of 0.25 and those designated as good receive a 0.75 trust.

The event queue is initialized and each node determines when it will meet all other nodes using a Poisson distribution as defined in Equation A.1. The edge weight between nodes i and k is designated $w_{i,k}$ and $mTime_{i,k}$ (from Equation A.1) is the current simulation time; for simulation initialization $mTime_{i,k} = 0$. Once the meeting times are calculated, the meeting events are added to the event queue. For the initial node update event, each node will calculate $\Delta t * U(0,1)$ using the uniform number generator U. The user selected message send time t_{ms} divided in half becomes the average time in which each node will send a message. Each node will send its first message at $t_{ms} * U(0,1)$.

6.3.1.2 Simulation Execution

Once the simulated network is initialized, each event is pulled from the event queue in it's calendar order and executed. Node update events use the vector approximation to track and update trust. Message initialization events add another message to a randomly selected destination at $ct + t_{ms} * U(0,1)$ (ct is current simulation time). Meeting events do the following when nodes i and k meet:

1. *Check to see if either node is busy:* If one or both are currently broadcasting to another node, then the event is skipped and the next meeting time is selected using Equation A.1. This is done to ensure that a single node cannot broadcast to multiple nodes at the same time. We do acknowledge that nodes can step on each others signal, but this is less likely to occur in a sparse network and would only cause slight additional delays.

2. *Trade messages:* This will use a designated routing algorithm to determine which messages to forward. The one utilized here is based on Figure 6.3c and 6.4.

3. *Determine Broadcast Time:* Based on the number of messages sent during a

meeting event, the broadcast completion time is determined using:

$$bEnd_{i,k} = ct + \frac{(4 * n + 100 + sn * 1000) * 8}{100,000,000} \tag{6.4}$$

The numerator is the estimated size of the data to transmit in bits and includes a handshake of 100 Bytes plus $4 * n$ to account for the size of the trust matrix and 1000 Bytes per message segment traded (sn) for the payload. The denominator is based on network speed; we assume a 100 Mbps connection. These values are easily modified to represent different network configurations and message size assumptions.

4. *Process new Messages:* This is done by each node independently. Each node will attempt to recreate messages and update trust accordingly based on the states shown in Figure 5.1. If the node is the destination and this is the first time this node is able to recreate the message, then the delivery delay time is calculated and a message initiation Acknowledgment (ACK) packet is put on the simulation event queue.

5. *Determine next meeting:* This is done identically to initialization using Equation A.1. The only modification occurs when $mTime_{i,k} \leq bEnd_{i,k}$. In this case, subsequent meeting times are calculated until $mTime_{i,k} > bEnd_{i,k}$. This minimizes the number of events that are skipped in step 1.

6.3.2 Simulation Results

Using the simulator discussed above, we ran a number of simulation experiments. The results collected include the average of ten iterations for each configuration. Each simulation runs for 1000 time units with trust initialization based on a node status as described above. Each node sends a message randomly on average every $messageAvg = \frac{t_{ms}}{2}$ time units. Trust is updated using the trust management scheme based on [10]. For each simulation, we observed the following:

- Number of Bad Segments (*NumBadSeg*): This is the number of unmodified message segments that arrive at a bad node.

Table 6.2: Comparison of Routing Protocols

		Fraction of Good Nodes = 60%				Fraction of Good Nodes = 90%			
useTrust	*SendTo*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*
Yes	Closer Nodes	0	0	185,357	0.8462	0	0	493,636	0.6559
No	Closer Nodes	153,629	20,805	1,333,175	1.1388	57,660	10,329	774,435	0.7222
Yes	All Nodes	0	0	344,620	0.5580	0	0	924,919	0.3993
No	All Nodes	226,025	37,610	2,017,515	0.7245	107,374	22,101	1,413,903	0.4715
Wait for Source-Destination Meeting		0	0	69,129	3.4467	0	0	69,129	3.3367
TBSR	$Tr_a = 0.0$	8,194	991	536,299	0.6410	85	15	905,576	0.4770
TBSR	$Tr_a = 0.1$	29,852	1,071	579,155	0.7482	3,515	24	905,639	0.4896
TBSR	$Tr_a = 0.5$	97,521	9,810	743,679	0.7268	15,313	993	945,531	0.4720
TBSR	$Tr_a = 1.0$	134,310	17,830	892,533	0.7674	24,450	2,751	945,531	0.4582
trustFirst	$Mess_d = 2.0$	100,588	19,953	810,689	0.5841	23,842	4,599	1,088,652	0.4102

- Number of Bad Messages (*NumBadMSG*): Since we are using an erasure coding algorithm, this is the number of messages that a bad node would have been able to recreate during the simulation. The case when a bad node receives all segments, enabling it to recreate a given message, even if the message is encrypted, poses a high security vulnerability.

- Energy Used as a Function of Broadcasts (*EnergyUsed*): This is the total number of segments broadcast. While this value does not give a complete account of energy use, it is it's good approximation. The more a node broadcasts and receives, the more energy is drained. Minimizing the number of total segments sent is an important energy saving consideration.

- Average Delivery Delay (*AvgDelay*): This is the average time expired from when a message is created at the source until it is delivered successfully to the destination.

6.3.2.1 Comparison with Boundaries and trustFirst

Table 6.2 presents results with $messageAvg = 50$. It consists of five simulation sets to determine a baseline and enable a better comparison with TBSR. In the first set, the source waits for source-destination meeting (WSDM). In the other four sets, we used two flags to establish routing: *useTrust* and *SendTo*. If the *useTrust* flag is set, only nodes with trust above a threshold Tr_r send or receive messages. If the *SendTo* flag is set, then any node "closer" to the destination will receive a copy of the message irrespective of trust. Equation 6.5 is used to determine which node is closer. Each node determines its estimated time to meeting the destination; the one with the lower time is chosen. If *useTrust* is not set and *SendTo* is set to all, then

routing is endemic. If both flags are set, then only trusted nodes that are "closer" to the destination will receive a copy of the message.

$$tDest_k^i = -\left(\frac{1}{w_{i,k}} \times ln([0,1])\right) \tag{6.5}$$

The top five rows of Table 6.2 give the results for a network, where the fractions of good nodes are 60% and 90% and uses a trust threshold value of $Tr_r = 0.5$ when the *useTrust* flag is set. We made these choices to compare a more polluted network to less polluted one and to get hard lower and upper bounds for all four tracked data points. The source waiting to meet the destination (WSDM) gives the upper bound for *AvgDelay* and the lower bound for *EnergyUsed*. Pure endemic (*useTrust* set to no and *SendTo* set to all nodes) gives the upper bound for *NumBadSeg*, *NumBadMSG* and *EnergyUsed*. Normally, it would also give the lower bound for *AvgDelay*; however, with our network initialization and trust threshold when *SendTo* is set to all nodes and the *useTrust* flag is set, we are running endemic routing using only the good nodes. Our expectation was the use of only trusted nodes would reduce broadcast congestion and would avoid suppressing some potentially useful segment sharing. Therefore, in a highly polluted network, this protocol may make the average delay slightly lower than when using pure endemic routing, thus constituting the lower bound for *AvgDelay*. The results validate this reasoning. The lower bound for *NumBadSeg* and *NumBadMSG* occurs when the *useTrust* flag is set regardless of the value for *SendTo* and when WSDM is used.

The next four rows in Table 6.2 show the results of using our trust based secure routing protocol with different risk levels. This risk is defined as adaptive trust threshold (Tr_a). The lower this value, the lower the risk. If $Tr_a = 0.0$ then all nodes below the set Tr_r are "blacklisted" and the results, as expected, are closest to row 3 (*useTrust* flag is set and *SendTo* is set to all nodes). The value for Tr_r is the average trust that a given node has at the time it sends a message. This changes over time as trust is dynamically modified.

As Tr_a increases, the more nodes with low trust send and receive messages. When $Tr_a = 1.0$, the results are closest to row 2 (*useTrust* flag is not set and *SendTo* is set to "closer" nodes). The motivation for allowing higher risk is to

provide the trust management with observations about all nodes, including nodes currently classified as bad, to maintain high accuracy of trust values. This can be accommodated by allowing high level risk value for messages of low or fleeting value, which when compromised, will do little damage (or even using "information empty" messages to obtain needed observations for trust management).

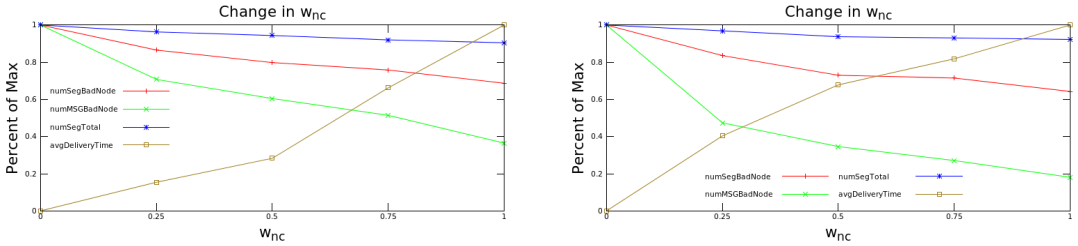
The final row in Table 6.2 shows the results using the protocol proposed in [33], [71] and referred to here as (*trustFirst*). There are two spray periods. The first sends only to nodes above a trust threshold. This threshold is set at $Tr_r = 0.75$. During the second, period any node receives a message. The message delay time $mess_d = 2.0$. The second period starts when $\frac{mess_d}{2.0}$.

Comparing the results for TBSR with TrustFirst in Table 6.2 shows the power of using "Situational Risk." For all values of Tr_a except where $Tr_a = 1.0$, TBSR significantly reduces the number of bad segments, bad messages, total number of messages sent (energy estimate) with only a modest increase in delay. For $Tr_a = 0.1$, there is a 70.32% decrease in *NumBadSeg*, a 94.65% decrease in *NumBadMSG* and 28.56% decrease in *EnergyUsed*. The trade-off is an increase of 28.9% in *AvgDelay*; however, in real time, is less than 0.2 time units while the security and energy decreases are a significant reduction.

6.3.2.2 Congestion Analysis

There were three congestion cases explored to determine how TBSR performed. The first is when the number of messages in the network is increased. This could occur during an emergency when additional message traffic is necessary. The second is when the frequency of nodes meeting increases. This can happen during a protest or class when many nodes gather in one location. The final is when both events occur. In many situations, such as emergency response of protests, a large number of nodes gather and they increase the message traffic.

Appendix B.2 has the results of the simulation (Table B.4). The key take away is that increasing the frequency of messages increases the number of bad segments, messages, and total messages sent (energy use). Increasing the frequency of meetings decreases the delivery time. Real congestion was not observed in any of



(a) 60% of Nodes Good **(b) 90% of the Node Good**

Figure 6.5: Energy, Security and Delivery Time Trade-Off using w_{nc}

the simulation test cases. Since the simulation tool is designed to abstract much of the network process away to be able to determine how tuneable parameters work, many network conflicts such as broadcast collisions are minimized. The simulation does not allow a single node to talk to multiple nodes at the same time; however, that interaction can cause delays to the point where other node interactions might not occur. This has been shown to be a good method of estimation that takes considerably less time to implement. It dictates what might be a good method to implement in NS3 or some other simulation tool.

6.3.2.3 Effect of Weight on The Number of Message Copies

One of the key tunable variables in Equation 6.3 is the weight given to the number of copies of a given message segment that a node forwards (w_{nc}). As stated above, valid values are $0.0 \leq w_{nc} \leq 1.0$. The motivation is to allow for more granularity in how TBSR handles the message classifications discussed in Section 6.2. Giving a weight to the number of message copies limits the number of times each node forwards a given message, while still allowing for the possibility that a message is sent to a node that has a high enough trust and/or has a shorter estimated time to reach the destination. There could be circumstances where the message

Table 6.3: Comparison of w_{nc}

	Fraction of Good Nodes = 60%				Fraction of Good Nodes = 90%			
w_{nc}	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*
0.0	43,466	2,938	640,022	0.6463	5,472	133	981,507	0.4379
0.25	37,610	2,078	616,401	0.6620	4,572	63	950,665	0.4588
0.50	34,687	1,776	604,128	0.6751	3,993	46	919,905	0.4730
0.75	32,949	1,510	589,071	0.7137	3,911	36	913,621	0.4802
1.0	29,852	1,071	579,155	0.7482	3,515	24	905,639	0.4896

classification has a fast or high delivery requirement and the number of copies and subsequent security and energy trade-off is an acceptable risk (secret/high). When this occurs w_{nc} should be a lower value or equal to zero. When $w_{nc} = 0.0$, the number of copies does not effect Equation 6.3, but trust and estimated delivery time do. The expectation is an inverse relationship between the average delivery time and security/energy as w_{nc} increases.

Table 6.3 presents the raw results using TBSR with $Tr_a = 0.1$ and with each node sending a message on average every 50 time units. These were run using 60% and 90% as the percentage of good nodes in the network. There are multiple values for w_{nc}. As expected, as w_{nc} increases the average delivery time also increases; this is due to more restrictions on the number of copies being forwarded. In addition, the number of messages sent to a bad node and the total number of messages sent decrease. Figure 6.5 shows this inverse relationship. Subfigures 6.5a and 6.5b show 60% and 90% good node density respectively. For all cases, modification of w_{nc} allows for the trade-off of energy/security and message delivery time.

6.4 Future Work and Conclusion

The Trust Based Secure Routing Protocol is designed to use trust to determine which nodes to forward or from which to accept messages. Nodes with lower trust values are more suspect and should be avoided; however, in order to dynamically maintain trust, and to account for nodes that are rarely seen or become temporarily isolated, they must not become completely "blacklisted." By smartly choosing which nodes to trust based on message classification (Section 6.2) and the subsequent "Situational Risk" calculations in Equation 6.3, improvements are made over other protocols that only use trust.

There is a trade-off between average delay, security and energy used. One extreme is endemic routing which minimizes the average delay but maximizes the energy use and security risk. The other extreme is to only send the message to the destination (WSDM) which minimizes security and energy use but maximizes delay. Neither approach is appropriate for any of the message classifications. TBSR uses three key values to determine "situational risk" namely trust, the number of copies

of a message and the "closeness" between each node that is the estimated time to meet the destination. The key value in TBSR is the SR, which is likely to be higher when trust is higher. It is shown to give significant increases in security and energy use for a modest increase of delivery time over a two spray period *trustFirst* routing protocol.

Additional research areas for this include full implementation of TBSR in a more robust network simulator such as NS3. Further research into different mobility patterns and adversarial models will help refine the results. Testing and determining message classification based on different operating environments will aid in tuning key parameters. Exploring the use of trust patterns as well as trust values can provide an additional clue for direct and indirect trust.

CHAPTER 7
Conculsions and Future Research

There is a requirement for secure and robust networks in all environments. This dissertation explored a class of Resource Constraint Networks (RCN) in order to improve information assurance. It mainly focused on Delay Tolerant Networks but many of the concepts are portable to the larger class of RCNs. While researching the topic, it became clear that managing trust in a distributed environment and using that trust to make secure routing decisions at each node was an interesting and open topic. This led to three research questions tackled in Chapters 4, 5 and 6.

1. What are the best clues for use in observing and updating direct trust?
2. How to fuse direct and indirect trust to create an aggregate trust value?
3. How to take trust and apply that to routing, in order to as securely as possible forward a message from source to destination?

Based on the questions enumerated above, this dissertation presents a number of contributions to trust, security and routing in a DTN.

7.1 Path Clues

There are multiple possible clues for use in a DTN. This presents the idea of using path information to observe and make direct trust updates. Each node maintains trust, as a value between 0.0 and 1.0, for every other node in the network. Using erasure coding and a check-sum, each message is broken into multiple segments and forwarded through the network. When the segments arrive at the destination the message is recreated. Based on the results, trust is increased or decreased for nodes along the utilized paths. A number of utility functions are proposed to determine when the destination should request a message be resent. For either utility function to work, in a distributed manner, the probability that the next segment arrives and is untainted is necessary. This needs to occur at each node

with only information stored at that particular node. A method for determining this probability is presented.

Future Work :

There are a number of improvements for future work. Testing the results on a more robust set of threat models will identify any potential vulnerabilities. Modifying the trust penalties and rewards to be more adaptive, such as sliding windows similar to TCP, has the potential to better refine the results. Different mobility patterns will show situations where the approach is vulnerable or confirm it's utility. Further analysis on path flow and the probability of segments arriving untarnished can lead to further refinement of directly observable path clues in a distributed environment.

7.2 Managing Indirect Trust

Two methods for managing indirect trust are proposed. Each stores historical information about recommending nodes. That information is then fused to create an indirect trust value. This is done at each node independently and decays older values, takes and gives more weight to the recommendation of more trusted nodes and aggregates all recommendations about a given node to produce an indirect trust value. This is done every time period Δt and is not done in real time, but near real time. Nodes that are often seen can not overwhelm and significantly modify trust (ballot stuff). Additionally, based on method, the recommendations from other nodes decay and time interval to time interval are averaged minus that delay. This again keeps one node from overwhelming and causing drastic changes.

Future Work :

Additional simulations with different mobility patterns will help tune the indirect trust variables to work in different environments.

7.3 Trust Aggregation

Trust is aggregated using direct and indirect trust information based on direct path clues and trading recommendations. The use of weight and decay are not new; however, the specific interplay presented here is. By aggregating trust, every Δt more information is obtained and a more smooth convergence is observed. This again reduces the effect of one bad node and results in quickly identifying malicious nodes. Even when a malicious node only acts bad 25% of the time, it is identified.

Future Work :

The complete distributed trust management system works well in a limited threat model. Additional tests with more robust threat models will help refine the results. The trust management system should be implemented in NS3, OMNet++ or some other more powerful network simulation tool. This will allow for the refinement of tunable parameters as well as the use of more robust mobility models. Changes in trust or trust patterns overtime might help to create a set of clues for use in more quickly determining bad nodes.

7.4 Trust Based Secure Routing

This dissertation presents a Trust Based Secure Routing (TBSR) protocol. TBSR uses trust and the "closeness" that a prospective next hop node is to the destination to make forwarding decisions. Both values are used to determine the risk associated with forwarding a message. That risk is based on the trust the sender has in the receiver. If that trust is low, then that node must be "closer" to the destination to offset that ambiguity. If that trust is higher, then it should receive a copy of the message, unless it is significantly better to wait. The number of copies of a message is limited and forwarding is based on the "Situational Risk" (SR), which directly effects SR. This is shown to reduce the number of copies that are sent to bad nodes with a minor hit in delivery time.

Future Work :

There is a proposed message classification scheme. More analysis will lead to

a better selection of parameters for determining SR. This could be mobility pattern specific.

REFERENCES

[1] C. Baraniuk. (Sep. 2014). Protesters adore firechat but it's still not secure, [Online]. Available: http://www.wired.co.uk/news/archive/2014-09/30/firechat-app-hong-kong-protesters (Date Last Accessed, Sep. 30, 2014).

[2] W. V. Maconachy, C. D. Schou, D. Ragsdale, and D. Welch, "A model for information assurance and integrated approach," in *Proc. IEEE Workshop on Inform. Assurance and Security*, West Point, NY, Jun. 2001, pp. 306–310.

[3] K. Govindan and P. Mohapatra, "Trust computations and trust dynamics in mobile adhoc networks: a survey," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 2, pp. 279–298, Feb. 2012.

[4] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility," in *5th Annu. IEEE Int. Conf. Pervasive Computing and Commun. Workshops*, White Plains, NY, Mar. 2007, pp. 79–85.

[5] Y. Zhu, B. Xu, X. Shi, and Y. Wang, "A survey of social-based routing in delay tolerant networks: Positive and negative social effects," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 387–401, Jan. 2013.

[6] A. Kate, G. Zaverucha, and U. Hengartner, "Anonymity and security in delay tolerant networks," in *3d Int. Conf. Security and Privacy in Commun. Networks*, Nice, France, Sep. 2007, pp. 504–513.

[7] J.-H. Cho, A Swami, and I.-R. Chen, "A survey on trust management for mobile ad hoc networks," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 4, pp. 562–583, Apr. 2011.

[8] D.-I. Curiac, C. Volosencu, D. Pescaru, L. Jurca, and A. Doboli, "Redundancy and its applications in wireless sensor networks: A survey," *W. Trans. on Comp.*, vol. 8, no. 4, pp. 705–714, Apr. 2009.

[9] T. Babbitt and B. K. Szymanski, "Trust management in delay tolerant networks utilizing erasure coding," in *2015 IEEE Int. Conf. on Commun., Ad-hoc and Sensor Networking Symp.*, London, United Kingdom, Jun. 2015, pp. 7959–7965.

[10] T. A. Babbitt and B. Szymanski, "Trust metric integration in resource constrained networks via data fusion," in *Proc. 18th Int. Conf. Inform. Fusion (Fusion '15)*, Washington, DC, Jul. 2015, pp. 582–589.

[11] T. Babbitt and B. K. Szymanski, "Trust based secure routing in delay tolerant networks," in *8th IEEE Int. Workshop Network Sci. for Commum. Networks (NetSciCom '16)*, San Francisco, CA, Apr. 2016.

[12] National Security Agency. (Sep. 2000). National information systems security (infosec) glossary, National Security Agency, [Online]. Available: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA433929&Location=U2 (Date Last Accessed, Oct. 15, 2015).

[13] Committee in National Security Systems. (Apr. 2010). National information assurance (ia) glossary, Committee in National Security Systems, [Online]. Available: http://www.ncsc.gov/publications/policy/docs/CNSSI_4009.pdf (Date Last Accessed, Oct. 15, 2015).

[14] National Institute of Standards and Technology. (Apr. 2013). Security and privacy controls for federal information systems and organizations, National Institute of Standards and Technology, [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf (Date Last Accessed, Oct. 15, 2015).

[15] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 5th ed. Boston, MA: Addition Wesley, 2010, pp. 36–45.

[16] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Comput. Networks*, vol. 38, no. 4, pp. 393 –422, Mar. 2002.

[17] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Comput. Networks*, vol. 52, no. 12, pp. 2292 –2330, Aug. 2008.

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

[19] T. Babbitt, C. Morrell, and B. Szymanski, "Self-selecting reliable path routing in diverse wireless sensor network environments," in *2009 IEEE Symp. on Comput. and Commun. (ISCC '09)*, Sousse, Tunisia, Jul. 2009, pp. 1–7.

[20] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," in *Proc. 2nd Int. Conf. Embedded Networked Sensor Syst. (SenSys '04)*, Baltimore, MD, Nov. 2004, pp. 1–12.

[21] M. Castillo-Effer, D. Quintela, W. Moreno, R. Jordan, and W. Westhoff, "Wireless sensor networks for flash-flood alerting," in *Proc. 5th IEEE Int. Caracas Conf. on Devices, Circuits and Syst.*, vol. 1, Nov. 2004, pp. 142–146.

[22] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater acoustic sensor networks: Research challenges," *Ad Hoc Networks*, vol. 3, no. 3, pp. 257 –279, May 2005.

[23] S. Taneja and A. Kush, "A survey of routing protocols in mobile ad hoc networks," *Int. J. of Innovation, Manage. and Technology*, vol. 1, no. 3, p. 279, Aug. 2010.

[24] B.-R. C. Perkins and S. Das. (Jul. 2003). Ad hoc on-demand distance vector (aodv) routing, [Online]. Available: https://www.rfc-editor.org/rfc/rfc3561.txt (Date Last Accessed, Nov. 9, 2015).

[25] R. Bruno, M. Conti, and E. Gregori, "Mesh networks: Commodity multihop ad hoc networks," *IEEE Commun. Mag.*, vol. 43, no. 3, pp. 123–131, Mar. 2005.

[26] I. Akyildiz and X. Wang, "A survey on wireless mesh networks," *IEEE Commun. Mag.*, vol. 43, no. 9, S23–S30, Sep. 2005.

[27] S. Zeadally, R. Hunt, Y.-S. Chen, A. Irwin, and A. Hassan, "Vehicular ad hoc networks (vanets): Status, results, and challenges," *Telecommum. Syst.*, vol. 50, no. 4, pp. 217–241, Aug. 2012.

[28] IEEE Computer Society. (Jul. 2010). IEEE standard for information technology, telecommunications and information exchange between systems local and metropolitan area networks specific requirements; part 11: Wireless lan me dium access control (mac) and physical layer (phy) specifications; amendment 6: Wireless access in vehicular environments, [Online]. Available: https://www.ietf.org/mail-archive/web/its/current/pdfqf992dHy9x.pdf (Date Last Accessed, Nov. 9, 2015).

[29] Cambridge Dictionary Online. (Oct. 2014). Trust, Cambridge Dictionary Online, [Online]. Available: http://dictionary.cambridge.org/us/dictionary/american-english/trust (Date Last Accessed, Oct. 15, 2014).

[30] S. Adalı, *Modeling Trust Context in Networks*. New York, NY: Springer, 2013, pp. 5–8.

[31] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. of the ACM*, vol. 36, no. 2, pp. 335–348, Feb. 1989.

[32] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: An efficient routing scheme for intermittently connected mobile networks," in *Proc. 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN '05)*, Philadelphia, PA, Aug. 2005, pp. 252–259.

[33] E. Bulut and B. K. Szymanski, "On secure multi-copy based routing in compromised delay tolerant networks," in *20th IEEE Int. Conf. Comput. Communiations and Networks, Workshop on Privacy, Security and Trust in Mobile and Wireless Syst.*, Kyoto, Japan, Jul. 2011, pp. 1–7.

[34] S. Zakhary and M. Radenkovic, "Erasure coding with replication to defend against malicious attacks in dtn," in *7th IEEE Int. Conf. Wireless and Mobile Computing, Networking and Commun.*, Shanghai, China, Oct. 2011, pp. 357–364.

[35] U. Sadiq, M. Kumar, and M. Wright, "Crisp: Collusion-resistant incentive-compatible routing and forwarding in opportunistic networks," in *Proc. 15th ACM Int. Conf. Modeling, Anal. and Aimulation of Wireless and Mobile Syst. (MSWiM '12)*, Paphos, Cyprus, Oct. 2012, pp. 69–78.

[36] Y. Ren, M.-C. Chuah, J. Yang, and Y. Chen, "Muton: Detecting malicious nodes in disruption-tolerant networks," in *2010 IEEE Wireless Commun. and Networking Conf.*, Sydney, Australia, Apr. 2010, pp. 1–6.

[37] M. Uddin, A. Khurshid, H. D. Jung, C. Gunter, M. Caesar, and T. Abdelzaher, "Making dtns robust against spoofing attacks with localized countermeasures," in *8th Annu. IEEE Conf. Sensor, Mesh and Ad Hoc Commun. and Networks*, Salt Lake City, UT, Jun. 2011, pp. 332–340.

[38] Q. Li, W. Gao, S. Zhu, and G. Cao, "To lie or to comply: Defending against flood attacks in disruption tolerant networks," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 3, pp. 168–182, Mar. 2013.

[39] M. Conti, R. Di Pietro, A. Gabrielli, L. V. Mancini, and A. Mei, "The smallville effect: Social ties make mobile networks more secure against node capture attack," in *Proc. 8th ACM Int. Workshop on Mobility Manage. and Wireless Access (MobiWac '10)*, Bodrum, Turkey, Oct. 2010, pp. 99–106.

[40] M. Conti, R. Pietro, A. Gabrielli, L. V. Mancini, and A. Mei, "The quest for mobility models to analyse security in mobile ad hoc networks," in *Proc. 7th Int. Conf. Wired/Wireless Internet Commun.*, Enschede, The Netherlands, May 2009, pp. 85–96.

[41] V. Natarajan, Y. Yang, and S. Zhu, "Resource-misuse attack detection in delay-tolerant networks," in *30th IEEE Int. Conf Performance Computing and Commun.*, Banff, Canada, Sep. 2011, pp. 1–8.

[42] M. J. Pitkänen and J. Ott, "Redundancy and distributed caching in mobile dtns," in *Proc. 2nd ACM/IEEE Int. Workshop on Mobility in the Evolving Internet Architecture (MobiArch '07)*, Kyoto, Japan, Aug. 2007, 8:1–8:7.

[43] M. K. Denko, T. Sun, and I. Woungang, "Trust management in ubiquitous computing: a bayesian approach," *Comput. Commun.*, vol. 34, no. 3, pp. 398–406, Mar. 2011.

[44] E Ayday, H. Lee, and F Fekri, "Trust management and adversary detection for delay tolerant networks," in *2010 IEEE Conf. on Military Commun. (MILCOM '10)*, San Jose, CA, Nov. 2010, pp. 1788–1793.

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

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

[47] S. Adalı, R. Escriva, M. Goldberg, M. Hayvanovych, M. Magdon-Ismail, B. Szymanski, W. Wallace, and G. Williams, "Measuring behavioral trust in social networks," in *2010 IEEE Int. Conf. on Intell. and Security Informatics*, Vancouver, Canada, May 2010, pp. 150–152.

[48] P. Tague and R. Poovendran, "Modeling adaptive node capture attacks in multi-hop wireless networks," *Ad Hoc Networks*, vol. 5, no. 6, pp. 801 –814, Jun. 2007.

[49] P. Tague, D. Slater, J. Rogers, and R. Poovendran, "Vulnerability of network traffic under node capture attacks using circuit theoretic analysis," in *27th IEEE Conf. Comput. Commun.*, Phoenix, AZ, Apr. 2008, pp. 664–672.

[50] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," in *Proc. 2007 Conf. Applications, Technologies, Architectures, and Protocols for Comput. Commun. (SIGCOMM '07)*, Kyoto, Japan, Aug. 2007, pp. 337–348.

[51] B. Solhaug, D. Elgesem, and K. Stølen, "Why trust is not proportional to risk?" In *Proc. 2nd Int. Conf. Availability, Reliability and Security (ARES07)*, Vienna, Austria, Apr. 2007, pp. 11–18.

[52] A. Josang and S. LoPresti, "Analyzing the relationship between risk and trust (itrust'04)," in *2nd Int. Conf. Trust Manage.*, Oxford, United Kingdom, Mar. 2004, pp. 135–145.

[53] Department of the Army. (Sep. 2014). Risk management, Department of the Army, [Online]. Available: http://armypubs.army.mil/doctrine/dr_pubs/dr_a/pdf/atp5_19.pdf (Date Last Accessed, Sep. 30, 2014).

[54] Department of the Army. (Mar. 2012). Mission command, Department of the Army, [Online]. Available: http://armypubs.army.mil/doctrine/DR_pubs/dr_a/pdf/adp6_0_new.pdf (Date Last Accessed, Sep. 30, 2014).

[55] E. Ayday and F. Fekri, "An iterative algorithm for trust management and adversary detection for delay-tolerant networks," *IEEE Trans. Mobile Comput.*, vol. 11, no. 9, pp. 1514–1531, Sep. 2012.

[56] M. K. Denko and T. Sun, "Probabilistic trust management in pervasive computing," in *2008 IEEE/IFIP Int. Conf. Embedded and Ubiquitous Computing*, Shanghai, China, Dec. 2008, pp. 610–615.

[57] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[58] S. Cui, P. Duan, and C. W. Chan, "An efficient identity-based signature scheme with batch verifications," in *Proc. 1st Int. Conf. Scalable Inform. Syst. (InfoScale '06)*, Hong Kong, May 2006.

[59] A. Barabasi. (Aug. 2014). Network science, [Online]. Available: http://barabasilab.neu.edu/networksciencebook/download/network_science_ch10_ver_19.pdf (Date Last Accessed, Aug. 10, 2014).

[60] E. Bulut, Z. Wang, and B. Szymanski, "Cost efficient erasure coding based routing in delay tolerant networks," in *2010 IEEE Int. Conf. on Commun.*, Cape Town, South Africa, May 2010, pp. 1–5.

[61] Y. Wang, S. Jain, M. Martonosi, and K. Fall, "Erasure-coding based routing for opportunistic networks," in *Proc. 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN '05)*, Philadelphia, PA, Aug. 2005, pp. 229–236.

[62] Y. Lin, B. Li, and B. Liang, "Stochastic analysis of network coding in epidemic routing," *IEEE J. Sel. Areas Commun.*, vol. 26, no. 5, pp. 794–808, May 2008.

[63] J. Lakkakorpi and P. Ginzboorg, "Ns-3 module for routing and congestion control studies in mobile opportunistic dtns," in *2013 Int. Symp. Performance Evaluation of Comput. and Telecomm. Syst.*, Toronto, Canada, Jul. 2013, pp. 46–50.

[64] K. Scott and S. Burleigh. (Nov. 2007). Bundle protocol specification, [Online]. Available: http://www.ietf.org/rfc/rfc5050.txt.pdf (Date Last Accessed, Oct. 1, 2014).

[65] S. Symington and S. Farrell. (May 2011). Bundle security protocol specification, [Online]. Available: http://www.ietf.org/rfc/rfc6257.txt.pdf (Date Last Accessed, Oct. 1, 2014).

[66] A. Vahdat and D. Becker. (Apr. 2000). Epidemic routing for partially-connected ad hoc networks, [Online]. Available: http://www.cs.duke.edu/techreports/2000/2000-06.ps (Date Last Accessed, Oct. 15, 2015).

[67] T. A. Babbitt and B. K. Szymanski, "Trust management in resource constraint networks," in *Proc. 10th Annu. Symp. Inform. Assurance (ASIA '15)*, Albany, NY, Jun. 2015, pp. 51–56.

[68] B. Twomey. (Nov. 2015). Simple vs. exponential moving averages, [Online]. Available: http://www.investopedia.com/articles/trading/10/simple-exponential-moving-averages-compare.asp?header_alt=true (Date Last Accessed, Nov. 12, 2015).

[69] J. Burgess, B. Gallagher, D. Jensen, and B. Levine, "Maxprop: Routing for vehicle-based disruption-tolerant networks," in *Proc. 25th IEEE Int. Conf. Comput. Commun.*, Barcelona, Spain, Apr. 2006, pp. 1–11.

[70] E. Bulut and B. Szymanski, "Friendship based routing in delay tolerant mobile social networks," in *2010 IEEE Global Telecommun. Conf. (GLOBECOM 2010)*, Miami, FL, Dec. 2010, pp. 1–5.

[71] E. Bulut and B. Szymanski, "Secure multi-copy routing in compromised delay tolerant networks," English, *Wireless Personal Commun.*, vol. 73, no. 1, pp. 149–168, Jan. 2013.

[72] S. Chen and K. Nahrstedt, "Distributed quality-of-service routing in ad hoc networks," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 8, pp. 1488–1505, Aug. 1999.

[73] Department of the Army. (Sep. 2000). Department of the army information security program, Department of the Army, [Online]. Available: http://www.apd.army.mil/pdffiles/r380_5.pdf (Date Last Accessed, Nov. 4, 2015).

[74] J. Gross and M. Günes, "Introduction," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Günes, and J. Gross, Eds., Berlin, Germany: Springer-Verlag, 2010, pp. 1–11.

[75] G. Chen, J. Branch, M. Pflug, L. Zhu, and B. Szymanski, "Sense: A wireless sensor network simulator," in *Advances in Pervasive Computing and Networking*, Springer US, 2005, pp. 249–267.

[76] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. of the ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.

[77] A. Keränen, J. Ott, and T. Kärkkäinen, "The one simulator for dtn protocol evaluation," in *Proc. 2nd Int. Conf. Simulation Tools and Techniques*, Rome, Italy, Mar. 2009, 55:1–55:10.

APPENDIX A
Discreet-Event Simulator

There are three methods for performance evaluation for network protocols and schemes: mathematical analysis, measurements, and simulations. The first is discussed in detail throughout this dissertation where mathematical bounds are proven. The second, based on implementation of the system and numerous factors including cost, is best done after scheme or protocols show merit through the other two methods. The third uses computer simulations. In an ideal world, success in mathematical analysis would lead to simulation success and finally, success in real world applications. The challenge is what to model and then what tool or tools best evaluate a given scheme or protocol.

A computer simulation imitates a set of real world processes over time. There are various types of computer simulations that include discrete-event, continuous, Monte Carlo, spreadsheet, trace-driven etc. For network protocols and schemes discrete-event simulation is the norm. *Wehrle et al.* [74] discuss a number of simulation models and tools. Those tools include ns-3, OMNeT++, IKR Simulation Library, and openWNS. While these are open source, they are very robust simulation tools with large libraries for simulating many network protocols, schemes, and models. They are useful for simulation of Resource Constraint Networks, but were not made with their use in mind. Numerous other tools for use in Resource Constraint Network simulation have been proposed to fill the gap. One example is SENSE [75] that is very powerful for simulation of WSN routing protocols. Other principles are outlined in [76], [77].

This appendix proposes a simulation tool to model trust schemes in a Resource Constraint Network. This tool approximates or generalizes many network functions and focuses on the key components for trust management, mainly the transfer of message and trust information and node calculations done to update and manage trust. Section A.1 provides discrete-event simulation principals and overview. Section A.2 discuses the simulation tool in detail based on the principals of discrete

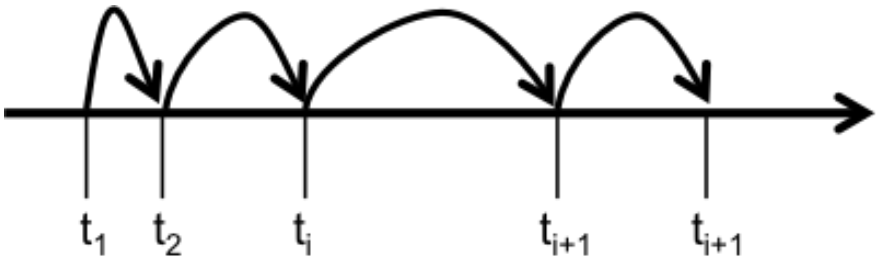


Figure A.1: Event Occurrence at time t_i

event simulation.

A.1 Principals of Discrete-Event Simulation

The basic concept for discrete-event simulation is to jump from key event to key event, each event being instantaneous in time, so each event E_i occurring over time period of t, is represented by two instantaneous events start-E_i at time t_i and end-E_i happening at time $t + t_i$. Looking back at someones life, it normally consists of a sequence of key events such as birth, birthdays, graduations, marriage, birth of children, and major professional accomplishments. While a persons life, like a network, is not a discrete-event system, a model that represents each key event can give insight into how someone lived or how well protocol and schemes work for a network. As each event occurs, the system state changes. Then potential and future events are generated. While in life those might not be preordained, in a network situation, one event might trigger another in the future or even at the current system time. The authors in [74] call these future events "event notices" and the queue of events "future event list (FEL)." To simplify, I will use, *events* on the *event queue*. There are two features defined for an *event*: one is the time it will occur and the other it's type. Figure A.1 [74, Fig. 1.1] depicts how these events occur. Each one event occurs at a discrete time and t_1 is the time when event 1 occurs. For any given event i, the time of it's occurrence is defined and denoted t_i. Two things are clear, looking at the timeline. The first is that the time between events is not constant and that there are periods of time where no events occur.

In general, there are a number of components required for discrete-event simulation. They were summarized in [74] and appear below:

- System/Network State: This describes the possible states and the transition between them. It consists of a series of variables.
- Clock: This tracks the time in the simulation.
- Statistical Counter/Log: This can be a set of variables to count key events, and/or a log file that prints required state variables at key times.
- Event Queue/Timing Routine: This will retrieve the next event.
- Event Routine(s): This is what happens when an event is called and the particular routine(s) called can be dependent on the network state and event type.

Figure A.2 [74, Fig. 1.2] shows the basic flow for a discrete-event simulator. The network state is initialized and then the first event is selected. Based on the type and network state, a series of routines are called that modify the network state and update counters and logs. This can be as simple as updating a trust level at a particular node. This will continue until the simulation terminates. Normally, this is at a given time, but could be based on network state or total number of events.

A.2 Simulation Tool

The simulation tool proposed here is written in Python and is an approximation for more robust tools such as ns-3. It 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 ns-3. This simulator is a high level approximation used to initially evaluate scheme parameters. More detailed simulations used to obtain more precise and robust results are expected to be qualitatively similar to those using this tool. The goal is to confirm or deny intuitions about the effect of malicious nodes on node and network trust.

The simulation tool creates a complete graph with n nodes, where node i is in the set of all nodes $i \in N$. 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

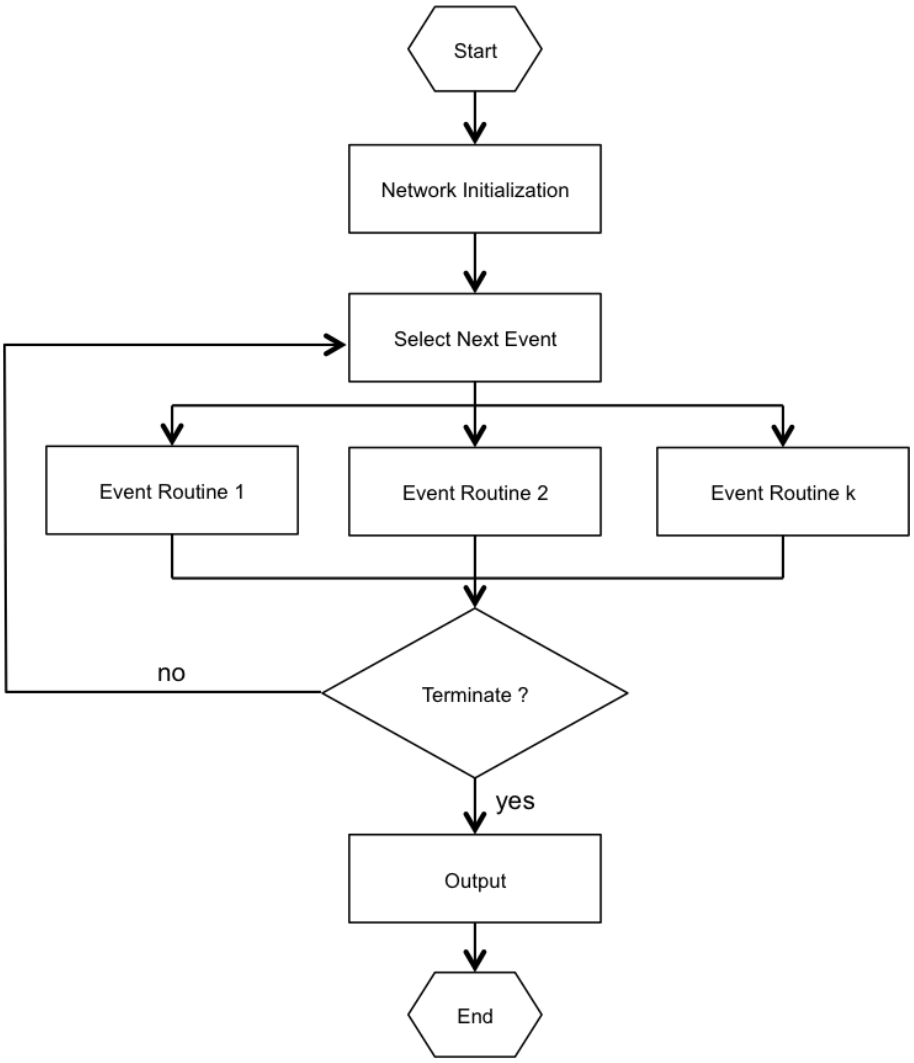


Figure A.2: Flow for Discrete-Event Simulation

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

A.2.1 Network State

The network state is a combination of the location of, and information buffered at, each node at a given time t. The location of nodes dictates which nodes can transmit messages and if any broadcast conflicts occur. The trust levels and messages buffered by nodes in broadcast range will determine if any messages are transmitted.

This lends itself to two categories of events. The first is a meeting event between two nodes. The second is a node update event that occurs based on a timer or conditional.

Figure 2.1 in Chapter 2.1.1.1 depicts the time between meetings in a DTN. It includes all of the normal networking delays of processing, queuing, transmission and propagation. The definition of a DTN adds an idle and handshake delay. Equation A.1 is used to determine when node i and node j meet. It uses a Poisson distribution and the randomly assigned inverse of the intermeeting time, or edge weight, between the nodes. By creating meeting events in this manner, it takes into account the average idle delay between two nodes; however, it does not take into account any of the other delays. This model assumes that if node i and node j meet they transmit all information immediately upon being in transmission range without normal network delays. Additionally, if node i meets node j at t_l and node i meets node k at $t_{(l+1)}$ and $t_{(l+1)} - t_l$ is less than the time it would take for node i to complete it's transmissions with node j, then either node k would have to wait, or by broadcasting to node i, disrupt the communication between i and j. The assumption that there is no network delay beyond that of d_{idle} will simulate optimistic results due to ideally fast transmissions that occur in this tool, that will not occur under normal network operations. This approximation is acceptable. It is likely that some messages sent in this tool will not be sent in a more robust tool; however, over a longer time of system operation, even in reality, similar trust results will occur.

A.2.1.1 Initialization

During the initialization of each run, each edge is uniformly randomly assigned a weight between [0.0,1.0). Each node has an initialized trust object and message queue. At a minimum, the trust object contains a vector with the aggregate trust node i has for all nodes $j \in N$ and a structure to track direct and indirect trust. The values in the aggregate trust vector are initialized as 0.5 and the direct and indirect trust structures as null. 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 with some probability.

A.2.1.2 Event Types

There are two 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 node's timer expires after Δt. A node will update first it's indirect trust information and then it's aggregate trust vector using both the indirect and direct trust vectors. A subclass of the node event is a message event. This will either put a new message or acknowledgement into the buffer for a given node.

A.2.2 Event Queue

The simulation event queue is 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 using

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

to create an initial meeting event 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 $[0.0, x]$ 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.

A.2.3 Event Routines

There are three categories of event routines: network, message, and trust. The first category are routines used to manage the network. This category includes routines that are invoked when nodes meet. The second category includes routines involved in message transmission. They determine what messages to send once

two nodes are in contact, they also include sending acknowledgement packets and clearing buffers of stale messages. The final are trust routines which update and manage the trust object at each node. The type of routines called is based on the type of event executed.

A meeting event between node i and j calls a routing to transmit trust vectors, check for acknowledgments message and trade message segments. This will update indirect and direct trust based on received acknowledgement and message segments. If either node can recreate a message, then a trust update routine is called. A new acknowledgement will clear the buffers of all messages with that message ID. Finally, the next meeting between nodes is determined using Equation A.1.

A node update event will call the trust routines required to update indirect trust and then aggregate that with the direct trust. The details of operation of this event depend on the exact implementation. Some of the possible solutions of which are proposed and discussed in Chapter 5.

APPENDIX B
Supplemental Simulation Results by Chapter

There are a number of simulation experiments that test the effect of multiple scenarios, cases, and adjustments of tunable parameters. There are also key interim proposals and simulations that led to further research and experimentation. The key results and conclusions for each research question are presented in Chapters 4, 5 and 6. This appendix presents supplemental results that either support, or significantly contributed to, obtaining those results. They are included here with a complete description of the simulation set up, as done in either NS3 or a discrete event simulator as described in Appendix A.

Each one of the sections below gives a complete description of the simulation tool used, the initialization values and results. There is some overlap in discussion between this appendix and the corresponding chapter for each simulation experiment described below. This is done on purpose to avoid having to refer back to the chapter for details on simulator initialization or run. This is only done for the first set of multi-case simulations with key changes outlined for all subsequent tests. While this is meant to present supplemental results, the goal is for each simulation test case to be fully discussed below.

B.1 Supplemental Simulations: Chapter 5

This section contains additional details on simulations and setup that supports Chapter 5. There were two different discrete event simulators created. The goal of the first was to explore how indirect trust converges (based on Chapter 5.1.3.1),

Table B.1: Supplemental Simulations

Purpose	Concept Supplemented	Chapter Supported
Significance of Δt	Convergence Time and Number of Updates	Ch. 5.1.3.1
Comparing α_{in}	Comparing α_{in}, Given $n = 40$ and $p_{nx} = 60$	Ch. 5.2.2
Test Congestion	Effect of Message and Meeting Frequency	Ch. 6.3.2.2

and did not take into account direct trust (Chapter 4 and Chapter 5.1.2) or the aggregation of trust (Chapter 5.1.1). The goal of the second is to show the power of the complete trust management system utilizing both direct and indirect trust to fuse into an aggregate trust value.

B.1.1 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 (see Appendix A.2). 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 latter 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 5.13.

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 using Equation A.1, 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 A.1. 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.

The simulation network is initialized with each node's indirect trust vector initialized with values uniformly distributed between [0.0.1.0), and indirect trust matrix with all zeros. 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 prior to convergence. If the next event on the event queue is a node trust update, then the selected node updates it's indirect trust vector using the values in the indirect trust matrix and Equation 5.13 with $\beta = 1.0$. The next update for the

node is changed to be the current time plus Δt. If the next event is a node meeting event, then the nodes involved trade trust vectors; using equations in Chapter 5.1.3.1 to update indirect trust. Only the last trust vector received for a given node is used. The run ends when either 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.

Figure 5.6 illustrates the importance of Δt and are the results of the simulations described above. Subfigure 5.6a is the convergence time of indirect trust for various values of Δt. Subfigure 5.6b shows that the number of updates required as n increases is linear and the selected value of Δt has minimal effect. This suggests that smaller values of Δt are better.

B.1.2 Simulation Results for Distributed Trust Management System

To test the integration of direct and indirect trust, a second discrete event simulator based on Appendix A.2 was created. 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 connected 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 node's timer expires after Δt. A node will update first it's indirect trust information and then it's 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 A.1 is run and a meeting event is added. For node updates, the initial one occurs at $\Delta t * [0.0, 1.0]$, thereby staggering 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.

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. 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 A.1. For update events, that is accomplished adding Δt to the current simulation time. For message events, this 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.

There are two slight variations to this simulator. The first is a matrix version and uses the indirect trust updates as described in Chapter 5.1.3.2. The second is a vector approximation version that uses the indirect trust updates as described in

Table B.2: Simulation Setup Matrix Version Indirect Variables

Simulation and Network Control Variables		
Variable	Description	Value
n	Number of Nodes in the Network	40
num_its	Number of Iterations to Run	10
sim_time	Simulation Run Time	1000
$good$	Percentage of Nodes that are Good	{0.6, 0.75, 0.9}
k_{ec}	Num Segments to Recreate Message	3
s	Num Segments Message Broken Into	9
ack_wait	Time a node Stays in S3 (Figure 5.1)	10.0
act_bad	Bad Node Acts Malicious Percent	{0.5, 1.0}
t_thresh	Value when a node is no longer trusted	0.25
Aggregate Trust Integration Variables		
Variable	Description	Value
α_a	"Freshness" Factor for Aggregate Vector	0.1
Δ_t	Time Period Between Trust Updates	2.5
λ	Exponential Decay Factor	Using Eq 5.1
Indirect Trust Integration Variables (Matrix Version)		
Variable	Description	Value
α_{in}	"Freshness" Factor for Indirect Vector	{0.125, 0.25, 0.5, 0.75}

Chapter 5.1.3.3. Based on simulation version, there are a series of tunable parameters; Table 5.2 outlines them. The two sections below describe the parameters set for specific simulation runs.

B.1.2.1 Matrix Version

The only indirect trust variable, for the matrix version for tracking indirect trust, is α_{in}. A number of simulations were run to see the effect of α_{in} and the matrix version as a whole. Table B.2 shows the simulation configurations for this set of simulations. These simulations set α_{in} as 0.125, 0.25, 0.5, and 0.75; lower α_{in} values weaken influence of older indirect trust values on the aggregated trust. These values of α_{in} were run with various values for the percentage of good nodes $good = \{0.6,\ 0.75,\ 0.9\}$.

There are a number of paragraphs below that show some of the key results based on this set of simulation runs. The first two discuss α_{in} in more detail, and

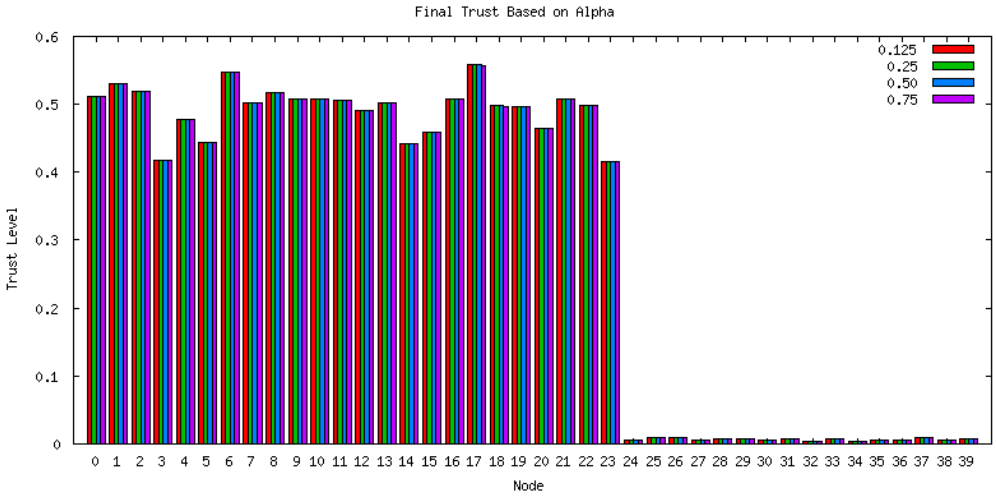


Figure B.1: Comparing α_{in}, Given $n = 40$ and $p_{nx} = 60$

subsequent ones discuss the results for the different fraction of good nodes using various values for how often a bad node acts maliciously.

Effects of α_{in}: For all of the variations of *good*, the different values of α_{in} had minimal effect on the outcome of the simulations. Figure B.1 shows an example of the results for all 40 nodes, given the initial set up and run time, where the fraction of good nodes = 60% and $\alpha_i \in (0.125,\ 0.25,\ 0.50,\ 0.75)$. For each of the nodes, the value for α_{in} makes a negligible difference in the final trust value. While some variation was expected, this makes sense that, over time, the values would all converge. 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.

B.1.2.2 Vector Approximation Version

There are two indirect trust tunable parameters when the vector approximation method of indirect trust management is utilized γ and β. The former is the weight given to older indirect trust values. The latter modifies how much the current trust of a recommending node affects indirect trust changes (see Chapter 5.1.3.3). Table B.3 outlines the values used for these simulations.

Table B.3: Simulation Setup Vector Approximation Version

Simulation and Network Control Variables		
Variable	Description	Value
n	Number of Nodes in the Network	40
num_its	Number of Iterations to Run	10
sim_time	Simulation Run Time	1000
$good$	Percentage of Nodes that are Good	{0.6, 0.75, 0.9}
k_{ec}	Num Segments to Recreate Message	3
s	Num Segments Message Broken Into	9
ack_wait	Time a node Stays in S3 (Figure 5.1)	10.0
act_bad	Bad Node Acts Malicious Percent	{0.5, 1.0}
t_thresh	Value when a node is no longer trusted	0.25
Aggregate Trust Integration Variables		
Variable	Description	Value
α_a	"Freshness" Factor for Aggregate Vector	0.1
Δ_t	Time Period Between Trust Updates	2.5
λ	Exponential Decay Factor	Using Eq 5.1
Indirect Trust Integration Variables (Vec Approx Version)		
Variable	Description	Value
γ	Previous Time Interval Weight	{0.1, 0.125, 0.167, 0.25, 0.5}
β	Current Trust Weight	{0.5, 1.0, 2.0, 4.0}

B.2 Supplemental Simulations: Chapter 6

In order to test the effects of congestion on Trust Based Secure Routing (TBSR), and a two period routing that uses only trusted nodes, in the first time period prior to sending to all [33], [71], a number of test cases were run using a discrete event simulator similar to the one described in Appendix A. There are three types of events that are managed by the simulation event queue: meeting, update and message initiation. A meeting event occurs when node i and node k are within broadcast range, conduct a handshake and, as protocol dictates, trade message segments. A node update event occurs every Δt time period. This is user defined when tuning up performance of trust management. A message initiation event occurs when a node places a message in it's buffer for transmission to another node, which is in it's vicinity. The user can designate how often a node sends a message. By default, that message is sent randomly to another node in the network.

B.2.1 Simulation and Network Initialization

Each node is initialized with a vector consisting of the Aggregate Trust (AT) for all other nodes in the network with a value between 0.0 and 1.0. The user defines the fraction of good nodes in the network. Since our simulations represent a more mature network, where trust has converged due to previous network activity, during simulation initialization the AT for nodes designated as bad are assigned a trust of 0.25 and those designated as good receive a 0.75 trust.

The event queue is initialized and each node determines when it will meet all other nodes using a Poisson distribution as defined in Equation A.1. The edge weight between nodes i and k is designated $w_{i,k}$ and $mTime_{i,k}$ (from Equation A.1) is the current simulation time; for simulation initialization $mTime_{i,k} = 0$. Once the meeting times are calculated, the meeting events are added to the event queue. For the initial node update event, each node will calculate $\Delta t * U(0,1)$ using the uniform number generator U. The user selected message send time t_{ms} divided in half becomes the average time in which each node will send a message. Each node will send it's first message at $t_{ms} * U(0,1)$. In order to increase the frequency of meeting, Equation A.1 is modified to Equation B.1. The extra variable $meetF$ consists of any value $0.0 < meetF \leq 1.0$. The lower the value, the more frequent nodes meet.

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

B.2.2 Simulation Execution

Once the simulated network is initialized, each event is pulled from the event queue in it's calendar order and executed. Node update events use the vector approximation to track and update trust. Message initialization events add another message to a randomly selected destination at $ct + t_{ms} * U(0,1)$ (ct is current simulation time). Meeting events do the following when nodes i and k meet:

1. *Check to see if either node is busy:* If one or both are currently broadcasting to another node, then the event is skipped and the next meeting time is selected using Equation B.1. This is done to ensure that a single node cannot broadcast

to multiple nodes at the same time. We do acknowledge that nodes can step on each other's signal, but this is less likely to occur in a sparse network and would only cause slight additional delays.

2. *Trade messages:* This will use a designated routing algorithm to determine which messages to forward. The one utilized here is based on Figure 6.3c.

3. *Determine Broadcast Time:* Based on the number of messages sent during a meeting event, the broadcast completion time is determined using:

$$bEnd_{i,k} = ct + \frac{(4 * n + 100 + sn * 1000) * 8}{100,000,000} \tag{B.2}$$

The numerator is the estimated size of the data to transmit in bits and includes a handshake of 100 Bytes plus $4 * n$ to account for the size of the trust matrix, and 1000 Bytes per message segment traded (sn) for the payload. The denominator is based on network speed; we assume 100Mbps connection. These values are easily modified to represent different network configurations and message size assumptions.

4. *Process new Messages:* This is done by each node independently. Each node will attempt to recreate messages, and update trust accordingly, based on the states shown in Figure 5.1. If the node is the destination and this is the first time this node is able to recreate the message, then the delivery delay time is calculated and a message initiation Acknowledgment (ACK) packet is put on the simulation event queue.

5. *Determine next meeting:* This is done identically to initialization using Equation B.1. The only modification occurs when $mTime_{i,k} \leq bEnd_{i,k}$. In this case, subsequent meeting times are calculated until $mTime_{i,k} > bEnd_{i,k}$. This minimizes the number of events that are skipped in step 1.

B.2.3 Results

There were three test cases run. The first increases the frequency that messages are sent from each node. The second decreases the time in which nodes interact;

Table B.4: Congestion Results

Test Case 1: Change in Message Density									
		Fraction of Good Nodes = 60%				Fraction of Good Nodes = 90%			
Protocol	*Message Density*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*
TBSR*	$messF = 100$	29,852	1,071	579,155	0.7482	3,515	24	905,639	0.4896
TBSR*	$messF = 50$	136,431	15,450	1,367,496	0.6926	13,103	765	1,871,278	0.4656
TBSR*	$messF = 25$	389,851	51,844	3,007,822	0.6620	27,471	1,527	3,589,403	0.4693
TBSR*	$messF = 12.5$	914,444	127,906	6,317,443	0.6913	43,174	1,228	6,841,708	0.4826
TrustFirst**	$messF = 100$	100,588	19,953	810,689	0.5841	23,842	4,599	1,088,652	0.4102
TrustFirst**	$messF = 50$	307,317	59,383	1,918,753	0.61111	80,347	15,476	2,244,411	0.4108
TrustFirst**	$messF = 25$	697,095	133,479	4,051,075	0.6212	190,836	36,885	4,576,102	0.4117
TrustFirst**	$messF = 12.5$	1,477,877	281,765	8,335,280	0.6254	405,445	77,462	9,202,060	0.4132
Test Case 2: Change in Node Meeting Density									
		Fraction of Good Nodes = 60%				Fraction of Good Nodes = 90%			
Protocol	*Density Value*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*
TBSR*	$meetF = 1.0$	29,852	1,071	579,155	0.7482	3,515	24	905,639	0.4896
TBSR*	$meetF = 0.50$	31,508	1,269	584,142	0.3679	3,751	23	912,057	0.2428
TBSR*	$meetF = 0.25$	31,143	1,313	576,094	0.1851	3,720	32	902,928	0.1215
TBSR*	$meetF = 0.125$	31,759	1,421	589,825	0.0903	3,725	27	915,594	0.0609
TrustFirst**	$meetF = 1.0$	100,588	19,953	810,689	0.5841	23,842	4,599	1,088,652	0.4102
TrustFirst**	$meetF = 0.5$	66,702	12,176	719,933	0.2949	20,159	3,887	1,070,440	0.2024
TrustFirst**	$meetF = 0.25$	67,397	12,379	718,160	0.1501	21,038	4,062	1,075,540	0.1039
TrustFirst**	$meetF = 0.125$	62,413	11,568	703,331	0.07516	17,960	3,426	1,062,751	0.05207
Test Case 3: Change in Node Meeting and Message Density									
		Fraction of Good Nodes = 60%				Fraction of Good Nodes = 90%			
Protocol	*Den/Mess Value*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*	*NumBadSeg*	*NumBadMSG*	*EnergyUsed*	*AvgDelay*
TBSR*	$meetF = messF = 1.0$	29,852	1,071	579,155	0.7482	3,515	24	905,639	0.4896
TBSR*	$meetF = messF = 0.50$	131,711	14,599	1,345,727	0.3449	13,598	898	1,873,028	0.2325
TBSR*	$meetF = messF = 0.25$	383,357	50,741	2,990,939	0.1715	25,842	1,234	3,555,253	0.1190
TBSR*	$meetF = messF = 0.125$	907,156	125,935	6,330,560	0.0864	41,176	1,015	6,773,886	0.0604
TrustFirst**	$meetF = messF = 1.0$	100,588	19,953	810,689	0.5841	23,842	4,599	1,088,652	0.4102
TrustFirst**	$meetF = messF = 0.5$	257,370	47,734	1,720,359	0.3146	77,096	14,841	2,232,769	0.2072
TrustFirst**	$meetF = messF = 0.25$	633,995	117,763	3,677,474	0.1632	185,905	35,915	4,548,539	0.1041
TrustFirst**	$meetF = messF = 0.125$	1,354,045	249,474	7,575,490	0.08193	394,857	75,864	9,200,409	0.0520

*All TBSR use $Tr_a = 0.1$
**All trustFirst use $mess_d = 2.0$

causing more frequent interaction. The final did both. It is likely that during an emergency, protest, or other event message traffic will increase. It is also likely that more nodes will be closer and interact more often. Realistically, both will occur at the same time. For each of the three test cases, TBSR has $Tr_a = 0.1$ (see Chapter 6.1.2). For trustFirst, the message delivery time is $mess_d = 2.0$. The second period starts when $\frac{mess_d}{2.0}$.

B.2.3.1 Test Case 1: Change in Message Density

For this test case, $meetF = 0$ in Equation B.1. All other settings, as described above, remain static except that the values for $messF = \{100,\ 50,\ 25,\ 12.5\}$. The simulations for TBSR and trustFirst consists of the average of 10 runs with 40 nodes and the percentage of good nodes set as 60% and 90%.

The results are in the top section of Table B.4. As expected, the number of bad segments and messages increase for each, with TBSR performing better in all categories than trustFirst. There is not a significant change in delivery time for either. This suggests that the congestion point has not been met or that the simulator does not fully support congestion. Likely, it is a combination of both.

The discrete event simulator was designed to estimate trust management and later message flow. Many network collisions are avoided on purpose.

B.2.3.2 Test Case 2: Change in Node Meeting Density

For this test case, $meetF = \{1.0,\ 0.5,;\ 0.25,\ 0.125\}$ in Equation B.1 and $messF = 100$. All other values remain static as described above. The simulations for TBSR and trustFirst consists of the average of 10 runs with 40 nodes and the percentage of good nodes set as 60% and 90%.

The results are in the middle section of Table B.4. There is a decrease in the delivery time by approximately 0.5 for each reduction in $meetF$ for both protocols. This makes sense because the average meeting time is cut in half. The number of bad messages and segments does not change for TBSR; however, there is a change for trustFirst. All of those values are decreased quite a bit on the first reduction, but then are steady when the percentage of good nodes is lower. When the percentage of bad nodes is higher, there is a slight decrease for all. It makes sense that trustFirst would have an increase in security because if nodes meet more often then they are less likely to panic and send to bad nodes. In a more polluted network, there are less good nodes so there is a higher chance that a message does not arrive prior to the second sending period. This also shows that TBSR finds a more ideal balance between delay and security/energy.

B.2.3.3 Test Case 3: Change in Node Meeting and Message Density

For this test case, $meetF = \{1.0,\ 0.5,;\ 0.25,\ 0.125\}$ in Equation B.1 and $messF = \{100,\ 50,\ 25,\ 12.5\}$. In order to mimic a network where both change at the same rate, the following pairs are used, formatted as ($meetF$, $messF$): $\{(1.0,\ 100);\ (0.5,\ 50);\ (0.25,\ 25);\ (0.125,\ 12.5)\}$. All other values remain static as described above. The results are in the bottom section of Table B.4. For each, it consists of the increase based on sending more messages (Test Case 1) and decrease of delivery time as seen in (Test Case 2).