

Secure Adaptive Routing Protocol for Wireless Sensor Networks*

University of Rhode Island
Department of Computer Science
Technical Report – TR10-329

Lisa Cingiser DiPippo
Yan Sun
Kenneth Rahn Jr.

University of Rhode Island
Kingston, RI 02881
dipippo@cs.uri.edu
yansun@ele.uri.edu
krahn@cs.uri.edu

Rajini Anachi
Onur Savas

mZeal Communications, Inc.
Fitchburg, MA 01420
arajini@mZeal.com
osavas@mzeal.com

Table of Contents

Section	Page
List of Figures	v
List of Tables	v
1.0 SUMMARY	1
2.0 INTRODUCTION	2
3.0 METHODS, ASSUMPTIONS AND PROCEDURES	4
3.1 Phase I Assumptions	4
3.1.1 <i>Problem Space</i>	4
3.1.2 <i>Trusted Deployment</i>	4
3.1.3 <i>Attacks</i>	4
3.2 Trust Framework	5
3.2.1 <i>Trust Management Framework Overview</i>	5
3.2.2 <i>Multi-dimensional Trust Metrics</i>	1
3.2.3 <i>Mathematical Properties of Trust Values</i>	7
3.2.4 <i>Trust Initialization</i>	8
3.2.5 <i>Dynamic Forgetting</i>	8
3.3 Secure Adaptive Routing Protocol	9
3.3.1 <i>Protocol Overview</i>	9
3.3.2 <i>Discovery Phase</i>	9
3.3.3 <i>Node Information</i>	10
3.3.4 <i>Transactions</i>	10
3.3.5 <i>Loop Detection and Handling</i>	11
3.3.6 <i>Trust Reporting</i>	13
3.4 Defense against Attacks	14
3.5 Cost/Benefit Analysis	15
4.0 RESULTS AND DISCUSSION	19
4.1 Simulations	19
4.1.1 <i>Selective Forwarding (SF) Attacks</i>	19
4.1.1.1 Simulation 1: Selective Forwarding Probability	19
4.1.1.2 Simulation 2: The Number of Attackers	20
4.1.1.3 Simulation 3: The Distance of Data Source from the Base Station	21
4.1.1.4 Simulation 4: The Number of Data Sources	22
4.1.1.5 Simulation 5: Network Size	23
4.1.2 <i>Intentional Loop Creation (ILC)</i>	24
4.1.3 <i>Bad Mouthing Attack</i>	25
4.2 Live-mote Experiments	26
5.0 CONCLUSIONS	29
6.0 RECOMMENDATIONS	30
7.0 REFERENCES	31
APPENDIX A	32
LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS	33

List of Figures

Figure	Page
1. (a) Sybil, (b) Wormhole, and (c) Blackhole Attacks	2
2. Trust Management Framework.....	1
3. Example WSN with an Untrusted Node	11
4. Loop Detection Scenario with One Data Source and 3 Attackers	12
5. A Selective Forwarding Scenario	17
6. Throughput Results with Varying Selective Forwarding Probability.....	19
7. Throughput Results with Increasing Number of Attackers.....	20
8. Throughput Results with Increasing Distance of the Data Source from the Root	22
9. Throughput Results with Increasing Number of Data Sources.....	22
10. Throughput Results with Increasing Network Size	23
11. The Demonstration Scenario.....	26
12. Screenshots from the Live-Mote Demonstration	28
A-1. Main Components of Crossbow TelosB.....	32

List of Tables

Table	Page
1. Malicious Attacks on WSNs.....	5
2. Defense against Attacks.....	15
3. The Data Source IDs for Each Number of Hops and the Associated Overhead under SARP	21
A-1. Crossbow TelosB Mote Platform Specifications.....	32

1.0 SUMMARY

This draft final research report covers the first seven months of our progress towards the completion of SARP. In particular, we begin with stating the problem space, assumptions about sensor network deployment, and the attacks we considered. Next we in great detail explain the trust framework including the multi-dimensional trust metrics, mathematical quantification of trust, initialization and dynamic forgetting. We present a distributed, dynamic, efficient tree-based routing algorithm that uses the multi-dimensional trust-metrics in order to find the most trusted paths. The integration of the algorithm and trust framework including how new routes are formed avoiding the suspected nodes (less trusted nodes) are explained in great detail. The implementation of defense against representative attacks including selective forwarding and intentional loop creation is also complete. An 8-mote proof-of-concept system was shown to be operational in a Crossbow TelosB test-bed and autonomous operation of the protocol including forming most trusted routes, and avoiding suspected nodes was demonstrated. Various simulations including throughput results with increasing number of nodes and increasing number of attackers were performed. The cost/benefit analysis shows that with a small overhead (~4% for a 25-mote grid) it is possible to avoid the attacking nodes and sustain a secure, operational sensor network.

The system is modularized for generic sensor networks and the hardware-dependent part has a thin Application Programming Interface (API) to the hardware-independent protocol engine, which can be easily replaced to deploy on other sensor platforms. The system as a whole and many reusable components such as the trust framework, and defense algorithms are expected to provide substantial performance increase for many situational awareness scenarios including persistent and ubiquitous surveillance, target tracking and localization.

2.0 INTRODUCTION

The need for secure routing in Wireless Sensor Networks (WSNs) arises from the recent success of WSNs in order to provide low-cost/high-benefit situational awareness but the lack of a holistic routing solution to integrate WSNs into the military specific security standards. Therefore there is an enormous amount of ongoing research in WSNs including the routing aspects of it. Since the nodes of WSNs are very limited, especially in terms of power, almost all of the routing efforts have focused on providing energy efficient solutions. However, in order to integrate WSNs into military's network centric operations, WSNs have to conform to the military specific security requirements. Therefore any solution for routing in WSNs should consider security as one of the top priorities, to which WSN research community hasn't paid enough attention compared to those of energy-efficiency.

Wireless networks are more prone to security vulnerabilities due to their broadcast nature, and WSNs, which are comprised of tiny limited sensor nodes, are even more prone to security attacks such as physical tampering and eavesdropping (see [9, Chapter 17] for a very detailed treatment of security in WSNs). However an attempt to deploy even more security is impeded by the inherent limitations of WSN nodes (check a current list of WSNs in [10]). For example, it is very hard if not impossible to run a 512 bit public key cryptography algorithm with 64kB instruction space and it is even harder to expect a reasonable operation with the resulting overheads in a 10 Kbits/sec communicational bandwidth. Therefore the security solutions for a Linux workstation or even for a PDA, for example running a firewall program, are not feasible for WSNs: new approaches and new concepts are required.

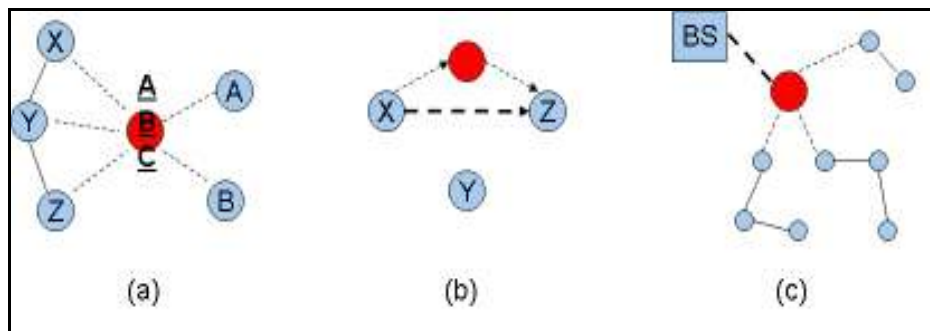


Figure 1 – (a) Sybil, (b) Wormhole, and (c) Blackhole Attacks

In order to fully understand and appreciate the defenses against attacks in WSNs, one should first understand the types of common attacks in WSNs (see [9]). Some of the WSN attacks are common to all kinds of data networks, for example, physical tampering, flooding the communicational links and causing Denial of Service (DoS), or spoofing or altering the information flow. However, some of the attacks are more peculiar to wireless ad-hoc and specifically to WSNs, such as wormhole, blackhole, and Sybil, whose pictorial representations can be found in Figure 1. For example in the blackhole attack, the adversary's goal is to lure almost of all of the traffic by creating a metaphorical sinkhole. The sinkhole is generally located near the data fusion center. In selective forwarding, the attacker may decide to filter some of the packets by discarding the rest. In a wormhole attack, the attacker tunnels the messages by creating artificial communicational links between colluded nodes. If the wormhole is placed very close to the base station, the adversary may be able to completely disrupt routing. In a Sybil attack, a single node presents multiple identities to other nodes in the network. In a hello flood

attack, an adversary might create the illusion that it is a neighbor even though it is far away.

There have been some complete or partial solutions for the considered attacks in WSNs. For example, the general defense for blackholes is using routing algorithms that are resistant to arbitrary configurations. Also sensor nodes may employ end-to-end verification under certain circumstances. For wormholes, a defense is to use geographical forwarding while a defense based on packet leases, where the distance a message travels is also limited, is also considered. In Sybil attacks, proper authentication is the strongest defense. Also location verification can be used as an alternative defense. [3, 7] provides a multi-dimensional trust evaluation, the detection of attacks, and reliable propagation of trustworthiness metrics throughout the network. Even though some of the solutions against attacks coincide with the solutions with those of wireless ad-hoc networks, the solutions for wireless ad-hoc networks do not necessarily solve the security problems of WSNs. For example, public key cryptography is too expensive for sensor networks.

Even though some smart and considerably comprehensive solutions have been proposed, to the best of our knowledge, a unified secure, scalable, energy-efficient routing approach including evaluation of universal trustworthiness metrics and its reliable propagation, efficient attack detection methods, and their integration with distributed energy-efficient WSN routing algorithms is still lacking. Ideally, the optimal solution for a secure routing protocol must be deployable in all kinds of current and foreseeable-future sensor nodes, must be adaptable to node topology changes, must quickly find alternate paths in security failures, and must be resistant to known and even future attacks. Clearly some of the requirements for the optimal solution contradict with each other hence there will be always trade-offs between different requirements. Also, while developing secure protocols, one must always respect the computational, communicational, and memory limitations of the sensor nodes, and the proposed protocols should be energy-efficient, scalable, robust, and adaptive. Survivability and graceful adaptation to ingoing/outgoing sensor resources are also two desirable features for all kinds of WSNs as well.

3.0 METHODS, ASSUMPTIONS AND PROCEDURES

3.1 Phase I Assumptions

The research performed in Phase I has been based upon certain fundamental assumptions about the type of sensor network, the type of data, the deployment of the network, and the types of attacks that might be made on the nodes in the network.

3.1.1 Problem Space

Applications for wireless sensor networks vary widely from broad habitat monitoring, to building monitoring to battlefield target tracking. These varied applications can pose very different requirements on the deployed WSN. Thus, it is necessary to define the focus of this Phase I research on a particular problem space. The solutions we provide in Phase I focus on WSN applications that have the following characteristics:

- **Static nodes** – The nodes in the WSN are static in the sense that they do not have the capacity to move on their own.
- **Large-scale random distributions** – The types of applications that this work is targeting will involve 100's to 1000's of nodes and the deployment of the nodes will not be regular, but rather we assume they will be scattered randomly in the field of interest.
- **Single base station** – We assume that the WSN has a single stationary base station that collects data from the sensor field. This does not imply that all processing is done at that base station, however. It is possible that some in-network processing and/or aggregation of data is done.
- **Upstream data flow** – Most of the data in the network will be flowing from the nodes in the network to the base station. The base station will occasionally send query requests or code updates downstream to the nodes in the network, but the majority of the data flow is upstream.
- **CSMA MAC Layer Protocol** – We assume that the nodes run a CSMA-based MAC layer protocol. In our implementation, the nodes run B-MAC.

3.1.2 Trusted Deployment

The type of application that our protocol is designed for consists of a one-time deployment of sensor nodes. That is, all of the nodes are deployed in a short period of time. We assume that the nodes enter a discovery phase (see Section 3.3.2) soon after they are deployed and that there is not enough time between deployment and the end of discovery for an adversary to compromise any of the nodes in the network. Thus, the information learned and shared in the discovery phase can be assumed to be trusted.

3.1.3 Attacks

In the current literature, many secure routing algorithms only address one or several types of attacks. However, in practice, smart attackers can launch various attacks targeting the routing algorithms as well as the defense mechanisms. In this project, we have conducted an in-depth investigation into possible attacks and synthesized the attacks as shown in Table 1.

Table 1 – Malicious Attacks on WSNs

Attacks	Description
Attacks against Routing Protocols	
<i>Selective forwarding attack</i>	The malicious node on the data transmission path intentionally drops a certain percentage of the packets passing through the malicious node. It is also referred to as a <i>blackhole</i> attack when the malicious node drops 100% of the packets and a <i>grayhole</i> attack when the malicious node drops some but not all packets.
<i>Wormhole attack</i>	The malicious node tunnels packets through a direct low-latency channel to another malicious node. The goal is to misrepresent the distance between the two colluding nodes and mislead route discovery process.
<i>Sybil attack</i>	The malicious node creates multiple identities or takes on identities of multiple nodes.
<i>Lying about hop counts or location</i>	The malicious node lies about its hop-count or location such that it has larger chance to be on the data transmission route and/or disturbs the route discovery process.
<i>Intentional loop creation</i>	The malicious node intentionally creates routing loops to disturb data transmission.
Attacks against Trust Management	
<i>Bad mouthing attack</i>	The malicious node badmouths honest nodes or praises its malicious peers, and aims to mislead trust calculation.
<i>On-off attack</i>	The malicious node behaves well and badly alternatively in the time domain. The goal is to recover its trust value after bad behaviors.
Attacks against Implementation of the Defense Solutions	
<i>No-response attack</i>	When the defense scheme requires the network nodes to process or send messages, the malicious node keeps silent and does not respond.
<i>Faking-report attack</i>	When the defense scheme requires the network nodes to process or send messages, the malicious node sends fake messages. Bad-mouthing attack is a special type of fake-reporting attack.

It is important to point out that the malicious node can launch several attacks simultaneously. For instance, the selective-forwarding attacks can be strengthened if the bad-mouthing attack is launched at the same time.

We have implemented and tested each of the above types of attacks and demonstrated that we can detect each type, and that our defense strategy works to avoid routing through untrusted nodes.

3.2 Trust Framework

3.2.1 Trust Management Framework Overview

Different wireless sensor networks can adopt different routing protocols based on network size, traffic pattern, application scenarios, and many other factors. In this project, the primary goal is not to secure a specific routing protocol. Instead, we aim to solve fundamental research challenges and the solutions should be applicable to various routing schemes. We have developed a theoretical foundation for trust evaluation in wireless sensor networks. This framework addresses fundamental challenges in trust evaluation, and is suitable to many routing protocols.

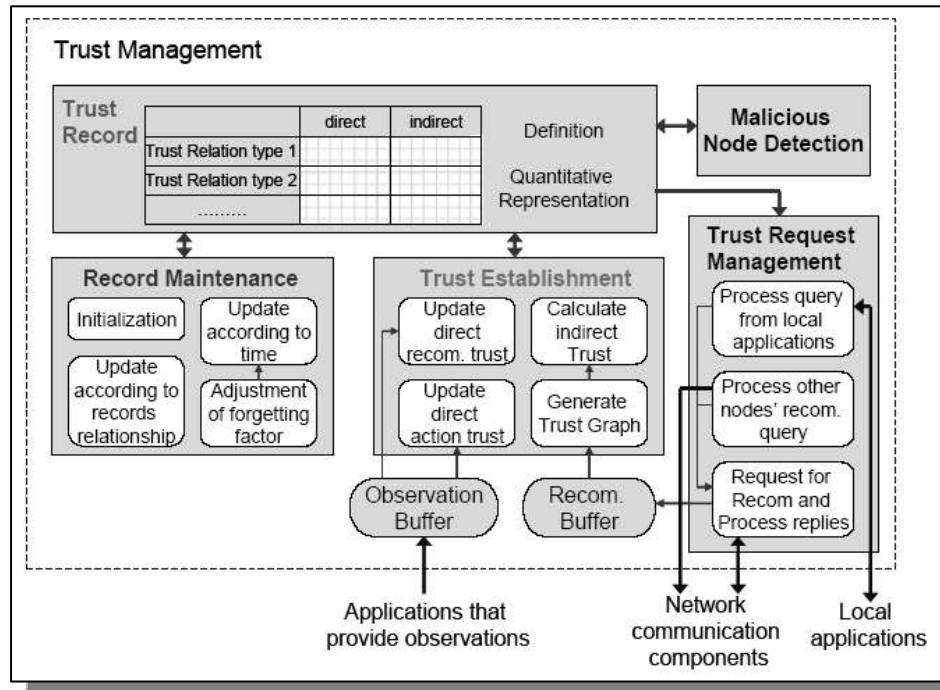


Figure 2 - Trust Management Framework

As shown in Figure 2, this framework contains the following core components:

- **Multi-dimensional trust metrics** - Determining quantitative measure of trust and confidence
- **Mathematical properties of trust** - Addressing how to calculate trust values based on observations and based on recommendations through third party, i.e. direct and indirect trust.
- **Trust initialization** - Setting initial trust values based on network conditions.
- **Dynamic properties of trust** - Addressing how trust values change with time.
- **Abnormal detection** - Detect misbehaving nodes as well as the occurrence of that attack against routing algorithms, trust management, and implementation of defense solutions.

3.2.2 Multi-dimensional Trust Metrics

We reviewed the existing literature on trust in both computer science and social science. We have identified the most appropriate interpretation of trust in WSNs as *belief*. That is, one entity believes that the other entity will act in a certain way, or believes that the network will operate in a certain way. Based on this generic and fundamental understanding of trust, we specify the notation of trust relationship and define the multi-dimensional trust metrics.

A trust relationship is always established between two parties for a specific action. We introduce the notation: $\{subject: agent; action\}$ to represent that the subject trusts the agent to perform the action. Let p denote the probability that the agent will perform the action in the subject's points of view. Based on the observations, we can estimate the value of p . Then, the **trust value** is defined as the mean of p and the **confidence value** is defined as the variance of p .

For each sensor node, we investigate the trust and confidence values for three key trust relationships:

- *Forwarding Trust*: {node A: node B; forward packets}
- *Availability Trust*: {node A: node B; response to request/control messages}
- *Reporting Trust*: {node A: node B; honestly send reporting message upon request}

3.2.3 Mathematical Properties of Trust Values

How to quantitatively evaluate trust? To answer this question, we identify mathematical properties of trust values that address this question from four perspectives: direct trust, indirect trust, trust initialization, and dynamic properties of trust.

Direct Trust. Direct trust can be established when node A directly observes node B's behavior. We have developed a light-weight **monitoring scheme** for evaluating direct *forwarding trust* and *reporting trust* at the node level. We explain the basic idea of this scheme using a simple case: $A \rightarrow B \rightarrow C \rightarrow D$ (i.e. source node A sends data to destination node D, through node B and node C).

- node A sends NS_A packets to node B
- node B collects the number of packets that B receives from A (i.e. NR_B) and the number of packets that B sends to C (i.e. NS_B)
- node C collects the number of packets that C receives from B (i.e. NR_C) and the number of packets that C sends to D (i.e. NS_C)
- node D collects the number of packets that D receives from C (i.e. NR_D).

Upon receiving a request from node A, node B, C and D broadcast reporting messages that contains their NR and NS numbers to their K hop neighbors. The initial choice of K is 2. Then, a node performs a consistency check. For example, if A sees that $NS_A = NR_B$, $NS_B = NR_C$, and $NS_C = NR_D$, node A believes that B and C report honestly. In this case, A can calculate how much A trusts B or C in terms of forwarding packets (i.e. forwarding trust). When the consistency check fails, A will reduce its trust in B or C in terms of honestly reporting (i.e. reporting trust). Furthermore, *availability trust* is calculated. Briefly speaking, if node C expects to receive NF and NR report from node D but does not receive such report after certain time, node C will reduce the availability trust of node D.

We have developed specific **algorithms** to calculate these trust values. These algorithms are originated from beta-function based trust model. Furthermore, we synchronize the collection of NR and NF numbers by introduce a new concept called *Data Transactions*. The basic idea is that transactions are created to delineate groups of packets that can be analyzed for performance in the trust reporting. The cost of this scheme is some extra book keeping, but the advantage of this scheme is big. It not only solves the synchronization problem, but also enables an efficient way to detect routing loops, which will be explained in detail in Section 3.3.5. Without the data transaction concept, routing loop detection would require complicated algorithms and significant overhead.

Indirect Trust. As described earlier, a node (e.g. node C) will evaluate its neighbors' *forwarding trust*, *honest reporting trust*, and *availability trust*. When any of these trust values drop below certain threshold, node C should report it to the BS by sending a warning report. Upon receiving a warning report, the BS cannot 100% trust the report because malicious nodes can send fake reports. Therefore, we have developed a solution for the BS to calculate indirect trust. Briefly speaking,

- BS evaluates how much it can trust node C's report. This trust value, referred to as *recommendation trust*, describes {BS, node C, sending honest warning reports}.
- After obtaining the recommendation trust of node C, the BS calculates indirect trust through trust propagation path $BS \rightarrow C \rightarrow D$. That is, given how much BS trusts node C, and how much node C trusts node D (as indicated in the warning report), the BS calculates how much the BS should trust node D. To calculate the indirect trust value, we adopt the trust models developed in our previous work [3, 7].
- After the BS calculates the indirect trust in D, the BS will broadcast that D is a highly suspicious node if (1) the trust value in D is lower than a certain threshold and (2) the associated confidence value is higher than certain threshold.

3.2.4 Trust Initialization

Trust initialization has been recognized as a tricky problem because it affects how fast trust values converge. In this project, we can greatly simplify this problem because we assume a trusted deployment, as discussed in Section 3.1.2. Due to this assumption, we set initial trust value of all nodes to 1, which means fully trusted.

3.2.5 Dynamic Forgetting

An on-off attack is a strong attack against trust evaluation schemes. In an on-off attack malicious entities behave well and badly alternatively, hoping that they can remain undetected while causing damage. This attack exploits the dynamic properties of trust through time-domain inconsistent behaviors.

Trust is a dynamic characteristic. A good entity may be compromised and turned into a malicious one, while an incompetent entity may become competent due to environmental changes. In wireless sensor networks, a sensor node may experience bad channel conditions sometimes and have low packet forwarding trust. When the channel condition becomes good, some mechanisms should be in place to recover its trust value.

In order to track these dynamics, an observation made a long time ago should not carry the same weight as one made recently. The most commonly used technique that addresses this issue is to introduce a forgetting factor. That is, performing K good actions at time t_1 is equivalent to performing $K\beta^{t_2-t_1}$ good actions at time t_2 , where $0 < \beta < 1$ is often referred to as the *forgetting factor*. In existing schemes, using a fixed forgetting factor has been taken for granted. However, when β is small (i.e. forgets very fast), the malicious nodes can behave badly, and wait for some time, and then recover their trust value. When β is large (i.e. forgets slowly), it takes long time for the trust value to drop when a node turns from good to bad. Both choices have deficiencies when the malicious users launch the on-off attack.

To defend against the on-off attack, we propose a scheme that is inspired by a social phenomenon – whereas it takes long-time interaction and consistent good behaviors to build up a good reputation, only a few bad actions can ruin it. We propose an *adaptive forgetting scheme*. Instead of using a fixed forgetting factor, β is chosen as $\beta=1-T$, where T is the current trust value. By doing so, the system will forget faster when trust value is high, and forget slower when the trust value is low.

3.3 Secure Adaptive Routing Protocol

Based on the assumptions listed in Section 3.1 and using the trust foundations described in Section 3.2, we have developed a secure routing protocol that can adapt to dynamic changes in the trust values of nodes in the network. In this section, we describe the protocol, and provide examples to illustrate the important features of the solution.

3.3.1 Protocol Overview

Given that we assume a single base station, the protocol is based on the construction of a routing tree. Upon deployment, each node in the network discovers its neighbors and learns the distance from the base station of each neighbor, and itself. When a node has data to send to the base station, it creates a transaction made up of a specific number of packets. For the transaction, each node along the path from the data source to the base station must choose from among these neighbors the “best” node to deliver its data to the base station. The determination of “best” parent node is based on a metric in which we combine number of hops to the base station with the sending node’s trust of its neighbors. Each packet in the transaction uses this chosen path. After each transaction, a trust reporting phase begins, in which nodes compute trust values for the nodes in the path of the transaction. In the following sub-sections, we describe the details of the protocol.

3.3.2 Discovery Phase

Once the nodes in the WSN are deployed, they begin a discovery phase in which each node learns which nodes are its neighbors, and how far it is from the base station. The discovery phase begins when the base station sends out a beacon that includes a “0” to indicate that the sender of the beacon is the base station (0 hops). Any node that can hear that beacon assumes that it is 1 hop from the base station, so it sets its hop count value to 1 and rebroadcasts the beacon, but this time with a “1” in the message. Thus, a node that can hear this beacon, and did not hear the first one, will assume that it is 2 hops from the base station. This beaconing spreads through the network so that all nodes can compute their own hop counts. Any time a node hears a beacon from one of its neighbors, it updates its hop count if it is lower than the hop count that is already stored in the node.

In this phase each node also learns and stores information about its one-hop neighbors (those it can communicate with directly) and its two-hop neighbors (the one-hop neighbors of its one-hop neighbors). The discovery phase continues for a specified amount of time, and then it is assumed that all nodes have discovered their neighborhoods and their own hop counts. In our simulations, we have set the discovery phase time limit to 30 seconds. This can certainly be modified for larger networks.

Recall from Section 3.1.2 that we assume that the deployment phase cannot be compromised, and so any information discovered in this phase is considered to be trusted. During this phase,

each node determines its own hop count from the base station. It learns the hop count of its one- and two-hop neighbors as well. We assume that this hop count remains static, and it can be trusted throughout the lifetime of the network. This assumption is valid because no nodes can be added to the network, and nodes will die only seldom in the network. If a node needs to change its hop count because another node has died, it must confirm this with the base station.

3.3.3 Node Information

In order to execute the routing protocol, each node stores certain information. The following are the elements that are stored in each node:

- **Neighbor Table** - Each node keeps its own Neighbor Table, storing information about each of its neighbors. Each entry in the table stores the node identifier, the hop count, the parent and the trust for the neighbor. The trust field in the neighbor table includes the various types of trust that are used to compute the overall trust value (see Section 3.2.1). Each of the trust values is initialized to 1.0 – maximum trust.
- **Transaction Table** - Data is passed through the network as a series of transactions. In each node, a transaction is implemented as circular buffer that holds information about a set of data messages that come from the same source and take the same path through the tree. The Transaction Table contains records that include information about the transaction identifier, the data source that initiated the transaction, the number of packets in the transaction, and other information that is useful for trust reporting.

3.3.4 Transactions

In our work, a transaction is the unit of data transmission and of trust reporting. When a node has a stream of data to report to the base station, it sets up a transaction, and continues transmitting data in that transaction, along the same path, for a specified number of packets.

Initializing Transactions. A node initializes a transaction record if it has its own data to report, or if it is asked to forward a packet that has a *transaction_id* that the node has not seen before. Nodes have a transaction memory only as big as the size of the Transaction Table, before new transaction overwrite older ones. For nodes closer to the fringe of the tree, this buffer size shouldn't present a problem, but for those closer to the root (or higher traffic outer nodes) it may be problematic. We address this concern by keeping a limit on the number of packets per transaction, and thus the nodes along a route do not have to remember it for very long. To ensure that each transaction identifier is unique, it is drawn from the initiating node's own node identifier, which is unique across the network.

Choosing a Transaction Parent. When a node receives a new transaction it must choose a *transaction* parent, to which it will forward all traffic in this transaction. The strategy is to choose one of its neighbors that will move the data closer to the base station without compromising the security of the network. It does this by choosing the parent node with the highest trust value. In certain cases, the trust values of all of the nodes closer to the base station (parents) are too low to use them for forwarding. In this situation, a node may need to forward through a neighbor that is either the same distance from the base station (peer) or one hop further from the base station (child). To choose a transaction parent, the node keeps track of the following in its neighbor table:

- *highest_parent* – the neighbor parent with the highest trust value.
- *highest_parent_or_peer* – the node with the highest trust value among all parent and peer nodes.
- *highest_child* – the child node with the highest trust value.

If the *highest_parent* neighbor's trust value is above a specified threshold, then this parent is chosen as the transaction parent, and the data is transmitted through it. If the *highest_parent* has a trust value that is too low (below the threshold), the node considers the trust values of peers as well as parents. The node compares the trust value of the *highest_parent_or_peer* neighbor to a second threshold. If the *highest_parent_or_peer* node has a high enough trust value, then it is chosen as the transaction parent. Otherwise, the node chooses the higher of the *highest_child* neighbor and the *highest_parent_or_peer* neighbor as the transaction parent. This strategy allows us to attempt to route the data around untrusted neighbors, while directing it towards the base station.

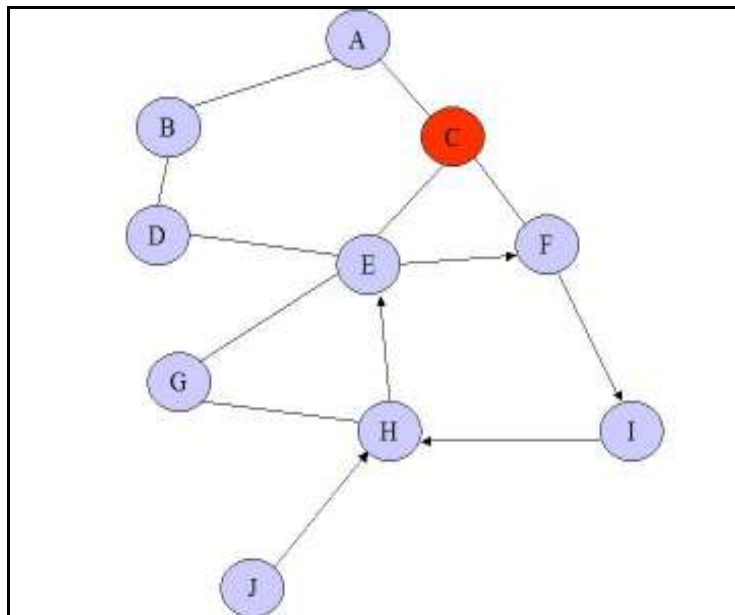


Figure 3 – Example WSN with an Untrusted Node

3.3.5 Loop Detection and Handling

In the absence of any attacks, our routing algorithm will not incur loops because a node always routes data through a neighbor that is closer to the base station than itself. However, as mentioned above, it is sometimes necessary to route data through peer nodes or even child nodes if the trust values of parents are too low. If this is the case, there is the potential for loops to arise in the routing path. We have devised a technique for detecting loops, and for breaking the loop if it is found. Consider the WSN depicted in Figure 3. In this figure, node *E* would normally choose node *C* to route through, but since it has been determined to be untrusted, it chooses node *F* instead. So, if node *H* receives a message that is part of transaction *X* from node *J*, it keeps track of this in its transaction table. If later, node *H* receives a message from a different node, say node *I*, that is also from transaction *X*, this means that transaction *X* has made a loop and has come back to node *H* through a different path. Thus, node *H* detects the loop.

In order to break the loop, the first node that detects it, in this example, node *H*, tries another direction for the transaction. That is, it considers a different neighbor to route the message through, for example, node *G* in the figure. If node *H* had no other options, either because of the trust values of its neighbors, or because it has no other neighbors, then it passes the message along to the next node in the loop and that node tries to break the loop the same way. If this continues and no node in the loop can break the loop, the first node that detected the loop tells one of its other neighbors to try an alternate route.

No matter how the loop is eventually broken, the source node of the transaction is notified that it should start a new transaction since the path to the base station has changed. The node that eventually breaks the loop sends a message back to the source to let it know.

Finding a loop in a path to the base station affects the trust values of the nodes in the loop. Any node that determines that it is in a loop modifies its trust value of the node that it originally routed through, since that choice was a factor in forming the loop. This type of “distrust” is considered less severe than distrust that arises from bad forwarding or bad reporting. But it is considered to some extent when making a decision in the future about routing data. So in the example of Figure 1, node *H* detects the loop, and it updates the trust value of node *E*. This will make it less likely to choose node *E* in a future transaction.

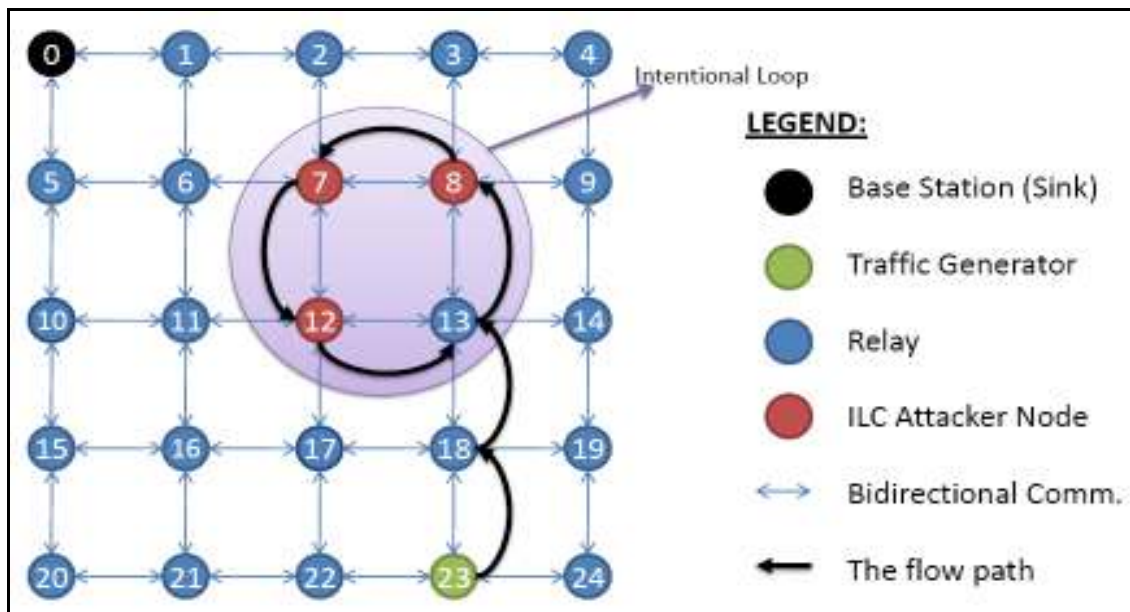


Figure 4 – Loop Detection Scenario with One Data Source and 3 Attackers

In some situations, the loop handling solution that we describe above can cause the message to fall back into the loop. Consider the scenario in Figure 4. In this case, the red nodes (7, 8 and 12) are attackers that intentionally create the loop, and the green node (23) is the data source. Using the loop detection/handling solution described above, node 13 would be the first node to detect the loop because the same message within a transaction from node 23 would come through it twice. When the loop is detected, node 13 would try another node to forward through. Node 13 would not forward the message to node 12 because that is where the message came from. Therefore, node 13 would choose to forward the message to node 14. This by itself is not

a problem, but when node 14 forwards to node 9, and then node 9 forwards to node 8, the message has found its way to the loop again.

To prevent this problem we have developed a way for nodes to communicate if they need another node in the route to try to break the loop. This solution involves a node stamping its *node_id* into a field in the message, and setting a bit in another field in the data message (henceforth called the *otherChoice* bit). This introduction of two 16-bit unsigned shorts into the data message will reduce the data payload size by 4 bytes, which leaves 12 bytes for the data. This is plenty of room for any sensor reading.

When a node detects a loop, it stamps its own *node_id* into the message. If it has another possible route to try, it unsets the *otherChoice* bit and tries the other route. If the message comes back with its own *node_id* stamped on the message (or there is no other possible route), then the node sets the *otherChoice* bit and sends along the "alternate route". Setting the *otherChoice* bit tells the receiving node that the previous node has tried all its possible neighbors, and needs someone else to break the loop. The next node detects the loop by noting the "stamped" id is the same as the sender, and stamps its own *node_id* over the sender's and repeat the process described in this paragraph.

Our mechanism has each node in the alternate route try all of its neighbors until it does not receive the packets back anymore. In the case that the loop cannot be broken, there is a maximum number of packets relayed per transaction before a node considers the packet to be in an infinite loop -- and it should not use any more of its energy on those packets. In Figure 4, if any of nodes 9, 3, or 2 sends the packets back into the loop then it will receive them back again, with their *node_ids* stamped on the transaction, and will try an alternate loop. Ultimately, node 2 will choose node 1.

3.3.6 Trust Reporting

The SARP routing protocol relies on updating trust of nodes based on performance of the nodes in transactions. Once a transaction is complete, the data source begins a trust reporting process so that the nodes along the path can update the trust values of their neighbors.

Requesting Trust Reports. When a transaction finishes, the source node sends a *report request* to its *transaction parent*. When that node receives the report request, it performs two actions: 1) it sends the same request to its *transaction parent*, and 2) it sets a timer that sends a report response back to the requestor. When any request is sent, a timer is set that, upon firing, checks to see if both the one- and two-hop reports have been received. If either of them was not received, then a specific request is sent the appropriate node, and the timer is reset. There is a maximum number of times a node will ask for a specific report before it considers that report to have timed out. If this occurs, it will flag the report and reset the timer.

When a report is received, the node first extracts the pertinent data, and then determines if it should forward the report along. In either case, it resets the timer.

Processing Trust Reports. When the timer fires, and both one- and two-hop reports have either been received or timed out, then the node processes the information from the transaction reports. There are three dimensions implemented in our trust calculations - how much a node trusts another node to 1) Forward Packets (*trustFP*), 2) Report Honestly (*trustRH*), and 3) Send Trust Reports (*trustAV*). These are all adjusted using the trust record's previous values of its Good

Forwarding Behavior (*GFB*), Bad Forwarding Behavior (*BFB*), Good Reporting Behavior (*GRB*), Bad Reporting Behavior (*BRB*), Good Availability Behavior (*GAB*), and Bad Availability Behavior (*BAB*). The previous values are multiplied by a forgetting factor before adding in the corresponding transaction values, and recomputing *trustFP*, *trustRH*, and *trustAV*. If any report was not received, there is incomplete transaction information. The trust will be calculated in the following manner:

- 1) If either neighbor does not report, it shall be penalized with an increased *BAB*. Otherwise, it shall be rewarded with an increased *GAB*.
- 2) If the one-hop neighbor is the only one to not report, the two-hop's *packetsReceived* count will be used as the one-hop's *GFB* for this transaction, and its *BFB* will be the number of packets this node sent minus the *GFB*.
- 3) If the two-hop neighbor is the only one to not report, then the one-hop's report cannot be proven inconsistent and will be used to evaluate its forwarding trust.

3.4 Defense against Attacks

We have discovered that the relationship between attacks and defense techniques is not a one-to-one mapping. One defense solution can help to address several attacks, whereas one attack may only be defeated when multiple defense methods are jointly applied. In this project, the defense solutions are highly integrated and reused.

Here is an example. The assumption of a *trusted deployment* enables us to address several other attacks.

In a **Sybil attack**, a malicious node aims to create multiple fake IDs. Since the attacker is not quick enough to compromise sensor nodes at the setup phase, they cannot conduct the Sybil attack at the setup phase. Therefore, we can defeat the Sybil attack by disallowing any new nodes from joining the sensor network after the setup phase.

In a **Wormhole attack**, two malicious nodes lie about their locations or hop counts, and aim to lure traffic to them. Since we have trusted deployment, malicious users cannot launch the wormhole attack at the beginning. Once the network is running, the hop count for a node remains static and nodes cannot change their static hop count. This prevents the wormhole attack. One may argue that a malicious node can still lie and forward data to its malicious peers. We would like to address this question from two angles. First, since simply forwarding data to its malicious peers will not affect other honest nodes' routing decisions, this type of bad behavior should not be called wormhole anymore. Second, simply forwarding data to its malicious peers does not affect the network performance directly. This type of bad behavior must be combined with selective forwarding attack or dishonest reporting of RN/FN numbers. We already developed defense against selective forwarding and dishonest reporting.

Finally, we summarize the attacks that we have addressed in Phase I in Table 2.

Table 2 – Defense against Attacks

Attacks	Defense Approaches	Reference
Attacks against Routing Protocols		
Selective Forwarding	(a) <i>Reporting mechanism</i> , (b) evaluating <i>forwarding trust</i> , and (c) avoiding untrustworthy nodes in routing algorithm. (Note: Part (c) is used in the defense of most attacks, we will not repeat it in the rest of table.)	Section 3.2.2, 3.2.3, 3.3.3 and 3.3.5
Blackhole Attack	Same defense approach as for selective forwarding.	Section 3.2.2, 3.2.3, and 3.3.3
Wormhole attack	<i>Static hop count</i> scheme and <i>forwarding trust</i> evaluation.	Section 3.3.2, 3.4
Sybil Attack	Assumption of <i>trusted deployment phase</i> .	Section 3.4
Lying about hop counts	<i>Static hop count</i> scheme prevents malicious nodes from changing their hop count.	Section 3.3.2
Intentional loop creation	<i>Loop detection and handling</i> scheme.	Section 3.3.4
Attacks against Trust Mechanisms		
Bad-mouthing attack	Evaluation of <i>reporting trust</i> ; the nodes conducting this attack will have low reporting trust and be avoided.	Section 3.2
On-off attack	With the <i>dynamic forgetting scheme</i> , the malicious nodes cannot recover their trust easily by performing good actions after bad actions	Section 3.2.5
Attacks against Implementation of the Proposed Solutions		
No-response attack	Evaluation of <i>availability trust</i> ; the nodes conducting this attack will have low availability trust and be avoided.	Section 3.2.2, 3.2.3
Fake-warning-report attack	Evaluation of <i>recommendation trust</i> and proper <i>indirect trust calculation</i> method. The nodes conducting this attack will have low recommendation trust. As a consequence, the BS will give low weight to the warning reports from these nodes.	Section 3.2.3

3.5 Cost/Benefit Analysis

Both in simulations and live-mote demonstrations, our goal is to compare all aspects of protocol implementation with and without employing SARP. For brevity, we will adopt the abbreviation of NO-SARP when the experiment is performed without employing SARP. We use the following criteria to evaluate the performance of SARP and to compare it with NO-SARP:

Defense/Security: Our main goal is to detect an attack, avoid the attacked node(s) in routing, and warn other sensor nodes against the attack. Therefore the main comparison between SARP and NO-SARP will be done in terms of their capability of defending against the attack. For example, in the Selective Forwarding case, the defense will be the capability to detect the Selective Forwarding mote and to avoid the mote after the trust reports indicate that it is dropping packets.

Performance Increase: Sometimes, an attack halts the operation of a sensor network completely, such as in the case of an intentional loop. In other cases, the attack degrades the performance of the sensor network. Hence the applications that exploit the sensor data run with low efficiency. We analyze the performance increase of SF in terms of throughput increase. In particular, we seek to measure the throughput gain under SARP and under NO-SARP. In the case of intentional loop creation (ILC), we propose and use a metric that involves the number of times the loop is detected and broken. The details of these calculations are given in their corresponding sections below.

Cost: The cost of using SARP can be measured in terms of communication, computation and storage overhead. However, computational and storage overhead is negligible to the communication overhead because of the simplicity of the operations and the need for small buffer sizes, and because in a WSN, communication is a much more expensive operation due to the energy constrained nature of the nodes. Therefore, we consider the communication overhead, i.e. the cost of transmitting messages, as the cost of using SARP over NO-SARP. This line of thought has been experimentally justified in numerous studies because of the power required to transmit a bit is order-wise larger than to process it. The development and maintenance of SARP can also be considered as one of the costs but we believe the implementation of a secure adaptive routing protocol justifies these costs.

We are mainly concerned with calculating the per-node throughput with and without SARP. Both with and without employing SARP, there is a common initial route setup phase before any data transaction begins, where START and DISCOVERY messages are transmitted. Since they are common to both SARP and NO-SARP and only transmitted before data transactions, we exclude the route setup messages from overhead calculation. However, under SARP, there is also trust messages transmitted, which contribute to the overhead of SARP. Therefore we calculate the per-node SARP throughput with and without overhead, namely raw and effective throughput, which will be explained shortly. Since route messages are the only messages in NO-SARP and excluded from the overhead calculation, raw and effective throughput under NO-SARP are the same. Specifically we measure the following throughput values:

- (i) per-node average *raw* throughput under SARP,
- (ii) per-node average *effective* throughput under SARP, and
- (iii) per-node average throughput under NO-SARP.

The raw throughput is measured as dividing the correctly delivered data packets by the total number of data packets transmitted. It represents the percentage of packets (out of all transmitted data packets) correctly. The effective throughput is obtained by dividing the correctly delivered data packets by the total number of packets including the trust reporting messages under SARP. It represents the percentage of correctly delivered packets out of all transmitted packets. We next discuss the specifics of these calculations:

Overhead Calculation of SARP: During an execution of data transmission under SARP, the transmitted messages can be broadly classified into three categories: (i) The routing setup messages, (ii) the trust reporting messages, and (iii) the data packets themselves. The number of messages transmitted for each category is dependent on how long the route between the data source and the base station is, and how frequently the trust reporting messages are sent.

Specifically, consider a route between the data source and the base station, which is of n hop length. We focus our attention to the nodes that are actively involved in the route setup and data transmissions. Under the current implementation, the routing messages are comprised of:

- (i) $2n$ START_MESSAGES, and
- (ii) $15n$ DISCOVERY_MESSAGES (because each mote transmits a discovery message every 2 seconds during 30 second routing setup phase).

Hence there are $17n$ routing messages transmitted before any data transaction begins. The trust reporting messages can be split into four categories:

- (i) $(n - 1)$ REPORT_REQUESTS,
- (ii) $(n - 1)$ END-TO-END transmission reports,
- (iii) $(n - 2)$ TWO_HOP reports, and
- (iv) 1 broadcast message.

Therefore, there are a total of $(3n - 3)$ trust reporting messages. Also, if any report is not received, e.g. due to a time-out, then the requesting node will send a specific request for a report to that node. Including the response, the overhead is 2 messages for a one-hop neighbor and 4 messages for a two-hop neighbor. We will assume the reports are not lost (at least in the simulations), hence we do not include these messages in the overhead calculations.

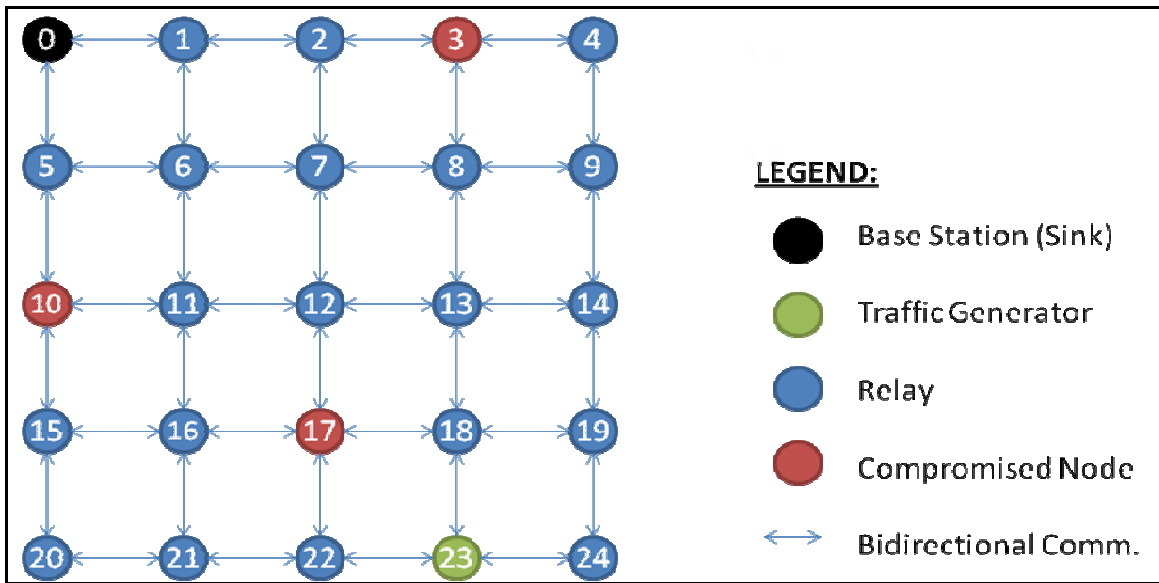


Figure 5 – A Selective Forwarding Scenario

The trust report messages are transmitted after a batch of data packets (a transaction). Assume L transactions of m data packets are transmitted during a complete data transmission cycle. The length of the route between the data source and the sink (the base station) is n therefore a total of nLm messages are transmitted/relayed as data messages. As mentioned earlier, we are excluding the routing setup messages from overhead calculation because whether SARP is employed or not, they are common to both SARP and NO-SARP, and they will be transmitted *before* any transmission begins. Thus, considering only trust report messages, a total of $L * (3 * (n - 1) + 1) = L * (3n - 3)$ trust report messages are transmitted because $(3n - 3)$ trust reports are sent after each data transaction. Therefore the overhead will be equal to

$$(L * (3n - 3)) / (nLm + L * (3n - 3)) = (3n - 3) / (nm + (3n - 3)).$$

Notice that the overhead is independent of L . To clarify these calculations, consider the following example, whose very parameters have been extensively used during the simulations:

Example 1: Consider Figure 5, where mote **23** is the sole data source. Since mote **23** is 7 hops away from the base station, $n = 7$. We assume a data fusion scenario, in which **5** transactions of **100** data packets are transmitted. Hence $L = 5$ and $m = 100$ resulting in transmission of **500** data packets per-node. After each of the **5** transactions, $3n - 3 = 18$ trust reports are sent hence totally $L * (3n - 3) = 5 * 18 = 90$ trust report messages are sent. Therefore the overhead is calculated to be approximately **2.5%** in this scenario.

Remark: It should be noted that the overhead under SARP is explicitly dependent on the relative frequency of the transmission rate of trust reports and data traffic. For example, it is possible to reduce the overhead by transmitting **500** packets contiguously and use the trust reporting at the end of this transmission. However, this kind of approach will reduce the effectiveness of SARP because if the packets are going through an attacker node, it might be too late to wait until the transmission cycle is completed. At the same time, sending the trust reports, for example after **10** data packets are transmitted, would substantially reduce the throughput. Therefore, there is a trade-off between choosing L and m to send a fixed amount of data packets. We believe this depends on the sensor network application and the type of attack. Under the studied scenarios in this report, we were able to keep the overhead to a reasonable value while still detecting and avoiding the attacks quickly. In addition, we develop an idea to further reduce the overhead. In the future, L and m values can be chosen dynamically. When the network is in good condition (i.e. nodes having high trust values), the network updates trust values less frequently. When low trust values or oscillating trust values are observed, the network updates trust values more frequently. By doing so, the overhead can be reduced when there are no malicious attacks.

We also want to emphasize again that the routing is done once (in the initial route formation phase). That is once we determine the neighborhood for each node, we do not have to do this again (unless there are significant topology changes, which is not expected in this problem space).

4.0 RESULTS AND DISCUSSION

4.1 Simulations

4.1.1 Selective Forwarding (SF) Attacks

In the following subsections, five Selective Forwarding attack scenarios are simulated in detail, the throughput results are plotted, and the results of the simulations are discussed.

4.1.1.1 Simulation 1: Selective Forwarding Probability

In the first experiment, we change the selective forwarding probability of attackers by fixing all the other parameters. In particular, we use a 25 mote grid topology with mote 23 being the sole data source and 0 being the base station (root). We include two attackers, each of which is a selective forwarding attacker with the same probability. In the seven trials of the experiment, the probability of attack is increased from 0.1 to 0.7 in steps of 0.1. Per-node throughput between mote 23 and the base station in terms of successfully received packet ratio is measured. Experiments using the same parameters are run under SARP and NO-SARP case. The obtained raw and effective throughputs are presented.

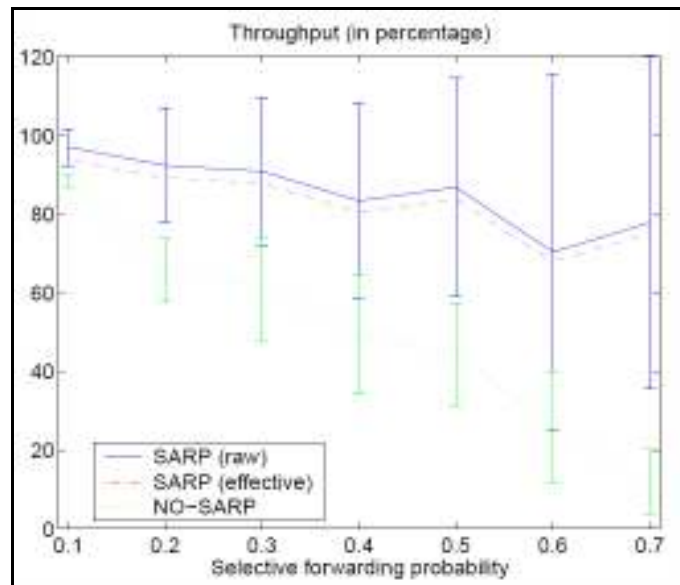


Figure 6 – Throughput Results with Varying Selective Forwarding Probability

During the simulation, mote 23 sends 5 batches of 100 data packets to the base station. As discussed in the previous section, after a transaction is complete, trust reports are exchanged and the route is updated if necessary. Since the setup and parameters used here are the same as Example 1, the overhead under SARP is 2.5%. The throughput values in percentages are plotted in Figure 6. The error bars associated with 90% confidence intervals also accompany the plots. This simulation is repeated 5 times both for SARP and NO-SARP cases.

The first observation about Figure 6 is the almost constant throughput maintained under SARP. However, if SARP is not employed, the throughput constantly decreases as the selective forwarding probability increases. The attack avoidance capabilities especially become significant when the selective forwarding probabilities of attack nodes begin exceeding 0.3. However, it should be noted that the throughput is only a measure of successfully received portion of packets by the base station and if SARP is not employed, the packets can keep being routed by the attacking nodes. The relatively large error intervals associated under SARP results from the high variability of data: For example, if the selective forwarding probability is 0.6, approximately 20% of the packets make their way to the base station. However, once attacking nodes are detected, they are avoided and throughput approaches 100%. Therefore the standard deviation (or variance) of the experimental data is higher under SARP than that of NO-SARP. However, it should be noted that once the attackers are detected, the throughput of the SARP scenario will level out and the standard deviation over a longer run of the system will become smaller. Also, in each simulation, the time of detecting the attacking nodes may change a little bit. This little change can result in large variation in throughput, which might be another reason for this large error interval.

4.1.1.2 Simulation 2: The Number of Attackers

The second simulation is performed by changing the number of attackers while the other parameters, e.g. the selective forwarding probability and number of data sources, are kept constant. We employ a 25-node grid topology with node 23 being the only data source and 0 being the base station. We increase the number of attackers from 1 to 7 between the data source and the base station. Recall that 7 is the number of hops between node 23 and the base station, which is equal to the shortest path between node 23 and node 0. Each selective forwarder drops packets with probability 0.2, and per-node throughput values both under SARP and without SARP are recorded.

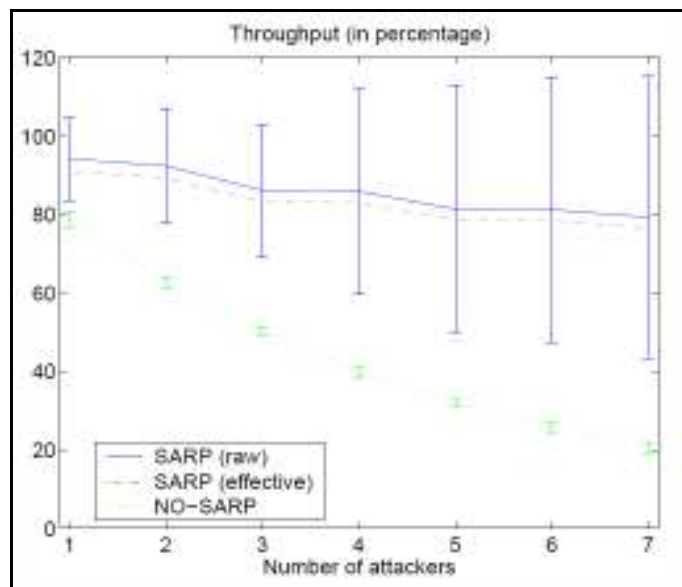


Figure 7 – Throughput Results with Increasing Number of Attackers

During the simulation, node 23 sends 5 batches of 100 data packets to the base station, i.e. the exact setup of Example 1. Therefore overhead under SARP is 2.5%. The throughputs in

percentages are drawn in Figure 7. The error bars accompanying each data point represent 90% confidence intervals. This simulation is repeated 5 times both for SARP and NO-SARP cases.

Consider Figure 7. Even though there is a slight drop in the observed throughput, employing SARP results in higher throughput compared to NO-SARP case. This difference in the throughputs is more easily observed especially when the number of attackers increases. When SARP is not employed, the decrease in the throughput is almost exponential. However, similar to our comments in Simulation 1, the main goal of SARP is not to increase throughput but to detect and avoid attackers, which sometimes comes with a cost of decrease in throughput. The confidence in NO-SARP data is higher than that of SARP because there is a higher variability in SARP simulations. However the 90% confidence intervals do not coincide, which indicates the reliability of simulation data.

4.1.1.3 Simulation 3: The Distance of Data Source from the Base Station

In this simulation, we change the location of the data source from the base station while keeping the number of attackers and selective forwarding probability constant. We use a 25-mote grid and have one data source. There are 2 attackers, each of which is a selective forwarder with probability 0.5. The data source is moved from 3 hops away to 8 hops away. Recall that in a 25-mote grid, 8-hop is the maximum distance possible between any two motes. Specifically, the data sources selected for each distance are indicated in the second row of Table 3 (the mote numbers can be found in Figure 5).

Table 3 – The Data Source IDs for Each Number of Hops and the Associated Overhead under SARP

# of hops	3	4	5	6	7	8
Data source	7	12	13	18	23	24
SARP overhead	0.8%	1.3%	1.7%	2.1%	2.5%	2.9%

The traffic generation is similar to the previous two simulations: 5 batches of 100 data packets are transmitted by the data sources. Since the overhead is dependent on the distance of the data source from the base station, overheads are no longer the same. The associated overhead for each number of hops (and data source) is indicated in the third and fourth rows of Table 3. The resultant per-node throughputs are plotted in Figure 8 along with the 90% confidence intervals. Each simulation is repeated 5 times.

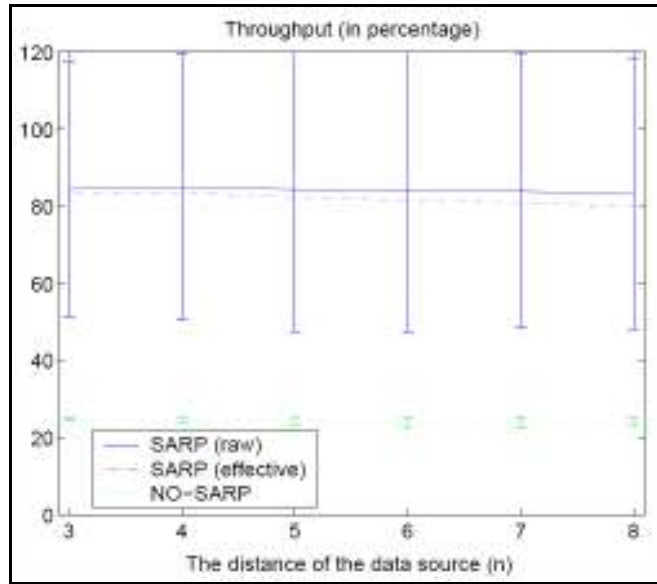


Figure 8 – Throughput Results with Increasing Distance of the Data Source from the Root

Consider Figure 8. The throughput under SARP outperforms that of NO-SARP. This is true for each distance from the base station. Therefore we conclude that the distance from the base station does not really affect the performance of SARP hence SARP is scalable with increasing number of hops (under the selected parameters).

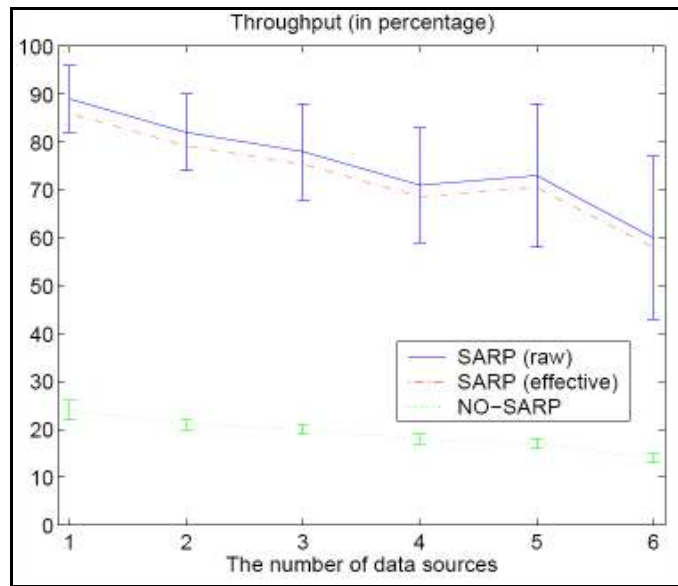


Figure 9 – Throughput Results with Increasing Number of Data Sources

4.1.1.4 Simulation 4: The Number of Data Sources

In Simulation 4, we increase the number of data sources while keeping the communication topology, the number of attackers, and the selective forwarding probability constant. This simulation is also done on a 25-mote grid and in each simulation we have 2 attackers, each of

which is a selective forwarder with probability 0.5. The number of data sources is increased from 1 to 6. The choice of the data sources is done manually. Specifically, nodes 23, 24, 19, 18, 22, and 14 are used as data sources, respectively with increasing number of data sources.

During the simulation, each data source independently injects 5 batches of 100 data packets into the network, all of them are destined for the base station (the root). Since the overhead associated with each source-destination pair is dependent on the distance between the sources and destination, the overhead is a simple average of overheads over all data sources. Similarly, the per-node throughput results are also averages of per-node throughput values of all data sources. The same simulations are repeated for SARP and NO-SARP 5 times, and the raw and effective throughputs are plotted in Figure 9 along with their 90% confidence intervals.

In this simulation, we are using more than one data source. Hence, we observed some events that have not been observed in the previous three simulations. For example, the data packets have been dropped due to buffer overflow/congestion. Since SARP does not differentiate between any packet losses, the trust values of intermediate congested nodes have been dropped. As observed from Figure 9, as the number of data sources increased, the end-to-end throughput slightly decreased. However, for the range of number of data sources (1 to 6), even with the overhead removed, SARP performed better than NO-SARP.

4.1.1.5 Simulation 5: Network Size

In the last simulation, we increase the grid size while keeping the number of attackers, and the selective forwarding probability, and the number of data sources (1) is kept constant. We have 2 attackers, each of which is a selective forwarder with probability 0.5. The network size is increased from 9 (3 by 3) to 64 (8 by 8) with grid size being to equal to a square of an integer each time.

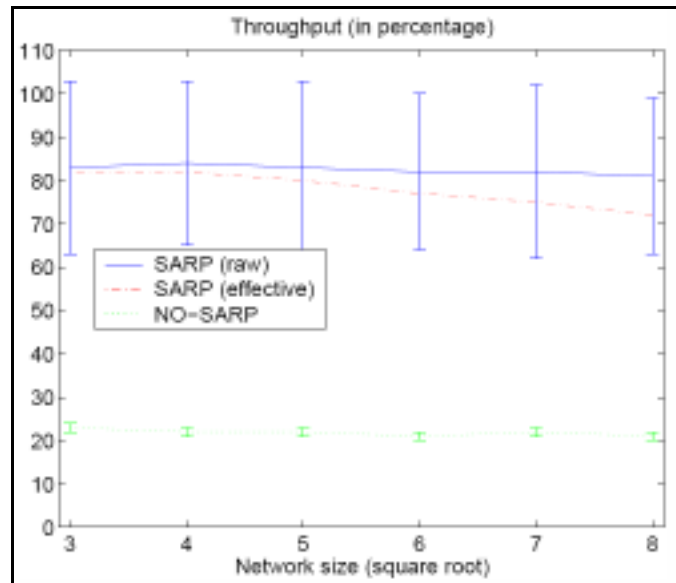


Figure 10 – Throughput Results with Increasing Network Size

During the simulation, each data source independently injects 5 batches of 100 data packets into the network, all of them are destined for the base station (the root). Each data source is placed at a location furthest from the base station. Hence, the data source is 4 hops away in a 9-node grid

while the data source is 14 hops away in a 64-mote grid. Since the overheads are dependent on the hop size between the data source and the base station, the overhead increased as the network size increased. The resultant throughput is plotted in Figure 10.

The almost constant throughput with increasing network size suggests the scalability of SARP with increasing network size. However, as mentioned before, the overhead increases as the number of hops from the base station increases. The overhead can be kept small with careful choice of data packets in each batch. For example, if 2 batches of 250 packets are used in order to transmit 500 packets, the overhead would be smaller. However, this comes with a risk of late-detecting the attack with a relatively larger number of packets passing through the attacking motes.

4.1.2 Intentional Loop Creation (ILC)

In these simulations, we used the topology of 25-mote grid with mote 23 being the only data source. Mote 0 (base station) is again designated as the destination for the data packets. Different from selective forwarding attacks, there are no parameters to vary in intentional loop creation scenario: A node is either an attacker or not. However, when the mote is an attacker, it is always possible to direct the packets into a specific mote. We use the same setup of Figure 4 of Section 3.3.5. Hence motes 7, 8 and 12 work collaboratively to form an intentional loop. Therefore when mote 13 forwards the packets to mote 8, it is first forwarded to mote 7, then mote 12 and mote 13 ends up with its own packet. If no attack detection algorithm is employed, the net end-to-end throughput will be zero. Therefore, end-to-end throughput will not be a comparison factor as it was in the selective forwarding attack.

Our main goal is two-fold: (i) We want mote 13 to detect the loop, and (ii) when mote 13 detects the loop, we want it to *break* the loop. Therefore our performance metrics are based on the detection and breaking of the loop. We run the simulations 20 times and recorded the statistics of how many times the loop is detected and broken. In all of the cases, the loop was detected therefore the success rate is 100%. However, we were only able to break the loop 16 times out of 20. Therefore loop breaking success is 80%. In particular, we use the following metric in order to evaluate the performance of loop detection algorithm:

Let N be the number of tests, k_1 be the number of times we can detect the loop, k_2 be the number of times we can break the loop. Also define Δ as the difference between “time when loop is created” and “time when loop free route is found” if a loop is detected and solved.

We ran the loop detection/handling experiments with $N=20$ tests and we measured the following:

- $k_1/N = 1$ since we detected every loop that was created.
- $k_2/N = 0.8$.
- Average Δ over k_2 tests, which is very close to 0.

The first two results above are as expected. However since the loop detection and breaking happens very quickly, the delay (Δ) could not be accurately measured in this topology. In future experiments, we are planning to use a larger loop and a larger topology where the delay measured scales with the execution time scale of the algorithm.

The more detailed explanation of how the simulations are done is as follows:

- 1) This is the first version of the loop detection/resolution code, and we will try to improve it using the algorithm refinement described in Section 3.3.5 so that the loop breaking success will approach 100%.
- 2) We have configured mote 23 to always forward its packets to mote 18 to mote 13 to mote 8. Therefore we always had the packets entering into the loop.
- 3) The first mote to detect the loop is always mote 13 and it always detects the loop.
- 4) The loop resolution algorithm is designed to have the next mote try some other mote randomly, and if that does not work, then pass it onto the next mote.

4.1.3 Bad Mouthing Attack

Up to the time of writing this draft final report, we have not completed all of the simulations that we plan to have finished by the end of the project (9/11/09). One other suite of simulations that we intend to perform involves demonstrating our defense against the bad mouthing attack. Recall that the bad mouthing attack is an attack on our trust mechanisms. We can think of defending this type of attack as analogous to an arms race in which adversaries build up defenses against each other. Consider an adversary that wants to hamper the flow of data from a data source to the base station. It infiltrates a node and inserts a selective forwarding attack, thus only a small percentage of packets are forwarded through this node. In defense of this attack, we have implemented the trust reporting mechanism, which is demonstrated in simulations described in Section 4.1.1. The adversary may then try to attack this defense by having the malicious node provide erroneous trust reports, thus making the system assume that the bad node is forwarding all of the packets it receives. To defend against this attack, we have implemented reporting trust, which allows a node to determine how much it trusts its neighbors to provide honest reports.

To demonstrate the effectiveness of this defense against bad mouthing, we will perform a suite of tests that show the progressive use of defenses described above. We will run a set of simulations, measuring overall throughput, under each of the following conditions:

1. Selective Forwarding with NO-SARP
2. Selective Forwarding with SARP
3. Selective Forwarding and Bad Mouthing with SARP (forwarding trust only)
4. Selective Forwarding and Bad Mouthing with SARP (forwarding trust and reporting trust)

This suite of tests will demonstrate that our SARP solution can handle the complex attacks involving both selective forwarding and bad mouthing. We expect the results to show that Test 2 will perform better than Test 1, as we have already shown in various simulations in Section 4.1.1. We further expect that Test 3 will perform worse than Test 2 since the bad mouthing attack will counter the benefits of SARP with forwarding only. Finally, we expect Test 4 to perform better than Test 3, and comparable to Test 2 because the bad mouthing attack is being handled.

4.2 Live-mote Experiments

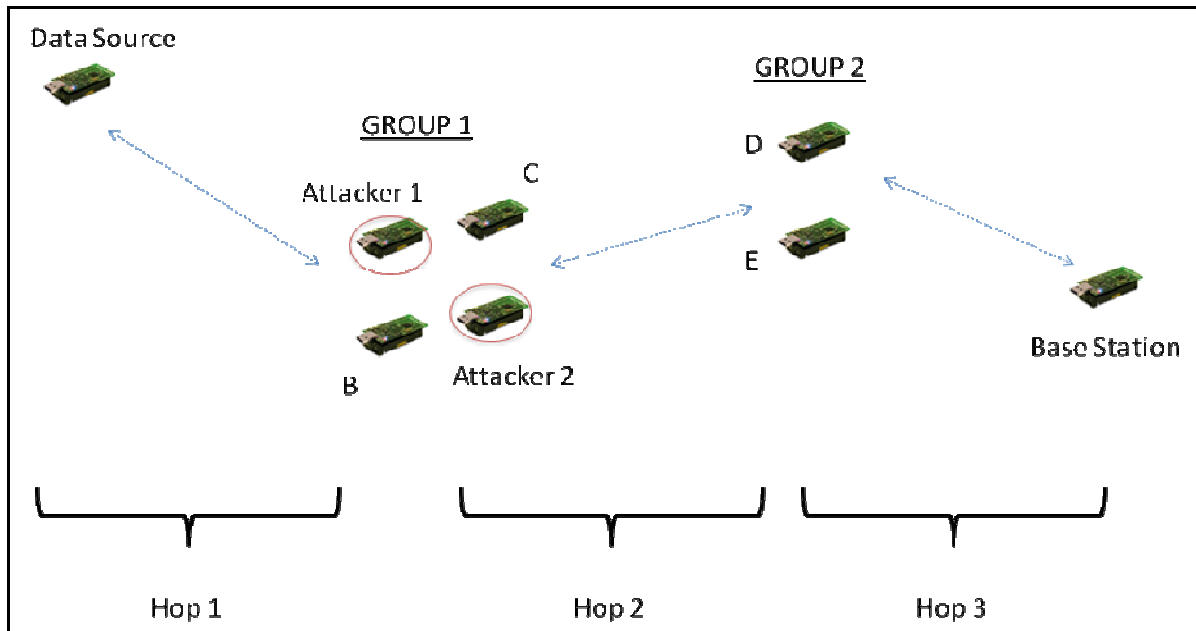


Figure 11 – The Demonstration Scenario

We have demonstrated the defense against selective forwarding attack in an 8-mote scenario involving CrossBow TelosB motes. The specifications of the mote can be found in Appendix A. A rough sketch of the demonstration is depicted in Figure 11. The motes have been placed into groups of four forming a three-hop sensor network:

- 1) *Data source*: This mote sends data packets to the destination (the base station) at a rate of 10 packets every 1 minute. The data packets are separated 3 seconds from each other hence the data source is silent for the final 30 seconds of 1 minute data transaction cycle. It is also connected to a PC via UART in order to read the trust reports and to observe how many packets made their way through.
- 2) *Group 1*: There are four motes in this group: Two of them are selective forwarders with probability 0.5 and two of them are non-attacking relay motes. When the network is formed, one of the motes is randomly chosen to form a communication link to the data source (and to one of the motes in Group 2). Initially all trust values are equal to 100% hence there is equal probability whether an attacker or a non-attacker mote is chosen. If an attacker mote is chosen initially, after the first data transaction, the base station will report to the data source that not all the packets the data source has transmitted have arrived. Therefore, the attacking mote's trust value will drop and another mote will be chosen. If a non-attacking mote is chosen initially, then due to "normal" packet losses, e.g. because of low SINR, the trust value will drop and another mote will be chosen again. In either case, after a couple of data transactions and trust reporting, the attacking motes will have lower trust values than non-attacking motes and will be avoided from the route.
- 3) *Group 2*: This group consists of two non-attacking motes and their purpose is to relay packets back and forth between the base station and Group 1 motes.

- 4) *Base Station*: The base station is the destination for the data packets. It also initiates the route setup by sending discovery messages and when the data transactions are complete, it sends trust reports back to the data source.

During the execution the following events have been executed and observed:

- 1) All motes are reset with the base station being the last mote to be reset. With the reset of the base station ($t = 0:00$), it sends *START_DISCOVERY* messages to the other motes in order to initiate the building of the routing tree.
- 2) The route setup ended at $t = 0:30$ and the route *Data Source -> Attacker 1 -> D -> Base Station* has been formed.
- 3) At the $t = 1:00$, the first data packet injection started, which we were able to observe because blue LEDs were programmed to blink when a data packet passes through them.
- 4) After $t = 1:30$, trust reports were sent to the data source, which indicated that Attacker 1 dropped packets. The trust value of Attacker 1 dropped to 47 (from 100). The network formed another route *Data Source -> B -> D -> Base Station*. Therefore there are no attackers in the route this time.
- 5) At $t = 2:00$ another data injection started and after $t = 2:30$, the trust reports have arrived to the data source and it was observed that B's trust has dropped to 98.
- 6) Therefore we have the following trust values for Group 1: Attacker 1 with 47, B with 98, Attacker 2 with 100 and C with 100.
- 7) Another route was formed and it was observed that Attacker 2 was chosen (C could have chosen as well because they had both equal trust values of 100).
- 8) At $t = 3:00$, another data injection started. This time the data packets went through Attacker 2 and after $t = 3:30$, base station reported to the data source that Attacker 2 was dropping packets.
- 9) Attacker 2's trust value was dropped to 30. Therefore C remained with the highest trust value (100) in Group 1.
- 10) Another route was formed, this time including C: *Data Source -> C -> D -> Base Station*.
- 11) This cycle can continue forever. We were able to demonstrate that each time one of the motes with the highest trust value was included in the route.



Figure 12 – Screenshots from the Live-Mote Demonstration

The video has been mailed to the Review Committee in a CD. Figure 12 has the screenshots from the video.

5.0 CONCLUSIONS

During the first seven months of Phase I, our research progress closely followed what was proposed and we accomplished what was promised. In particular, we were able to architect a trust framework including multi-dimensional trust metrics and mathematical quantification of trust for wireless sensor networks. We also proposed a highly distributed and dynamic routing algorithm and integrated the trust framework with the routing algorithm. The routing algorithm uses the trust metrics as link costs. We fully implemented the trust framework and the routing algorithm in TinyOS including defenses against representative attacks, namely selective forwarding and intentional loop creation. Live-mote demonstrations proved the feasibility of SARP for larger deployment and a more functional routing architecture. Various simulations demonstrated the scalability and energy-efficiency with increasing network size and increasing number of attackers. Overhead computations show that the benefits of SARP come with an affordable overhead. The performance increase under SARP promise a secure low-cost situational awareness for many applications, target tracking, localization to name a few.

6.0 RECOMMENDATIONS

There are many lessons learned from our research; most significantly the importance of sticking to a well-defined research plan and closely following it. The nature of our proposed solution required many tasks to be carried out at the same time. For example, architecting the trust framework, developing the algorithm for the routing protocol, and implementing it had to be carried simultaneously. Moreover, the pieces were dependent on each other. Therefore, we benefited a lot from having a well-defined research schedule and sticking to it.

The test-beds we chose for live-mote demonstrations and simulations were excellent in terms of development and implementation. Crossbow TelosB provides many features including power control, sufficient memory, and sensors (to provide situational awareness data) while being a representative sensor platform for a large class of sensor networks. TinyOS and TOSSIM 1.x also provide many built-in features to test, experiment and demonstrate capabilities of SARP. The ease of integration of the chosen hardware and software platforms allowed us to focus on the algorithmic and development aspects.

7.0 REFERENCES

- [1] "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).
- [2] "TOSSIM: A Simulator for TinyOS Networks," User's manual, in TinyOS documentation.
- [3] "A Trust Evaluation Framework in Distributed Networks: Vulnerability Analysis and Defense against Attacks," in Proc. INFOCOM, 2006. (Y. L. Sun et al.)
- [4] "Defense Trust Management Vulnerabilities in Distributed Networks," IEEE Comm. Magazine, Feature Topic on Security in Mobile Ad Hoc and Sensor Networks, Feb. 2008. (Y. L. Sun et al.)
- [5] "Energy-efficient MAC for Broadcast Problems in Wireless Sensor Networks," in Proc. of the Third Int. Conf. Networked Sensing Systems, June 2006. (D. Tucker, T. Henry, W. Day, K. Bryan, L. DiPippo and V. Fay-Wolfe)
- [6] "Routing-Aware Coloring in TDMA for Time-Critical Data Aggregation in Wireless Sensor Networks," in Proc. of the International Conf. on Wireless Networks, 2008. (T. Ren, T. Henry, L. DiPippo, K. L. Bryan, V. Fay-Wolfe)
- [7] "Information Theoretic Framework of Trust Modeling and Evaluation for Ad-Hoc Networks," IEEE JSAC Special Issue on Security in Wireless Ad Hoc Networks, vol. 2, no. 2, February 2006. (Y. Sun, W. Yu, Z. Han, and K. J. Ray Liu)
- [8] "The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks," in Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Urbana-Champaign, IL, 2005. (W. Xu, W. Trappe, Y. Zhang, and T. Wood)
- [9] "Security in Distributed, Grid and Pervasive Computing", Auerbach Publications, CRC Press, 2007. (Y. Xiao)
- [10] http://en.wikipedia.org/wiki/Sensor_node, list of sensor nodes.

APPENDIX A

CROSSBOW TELOS B SENSOR MOTE PLATFORM

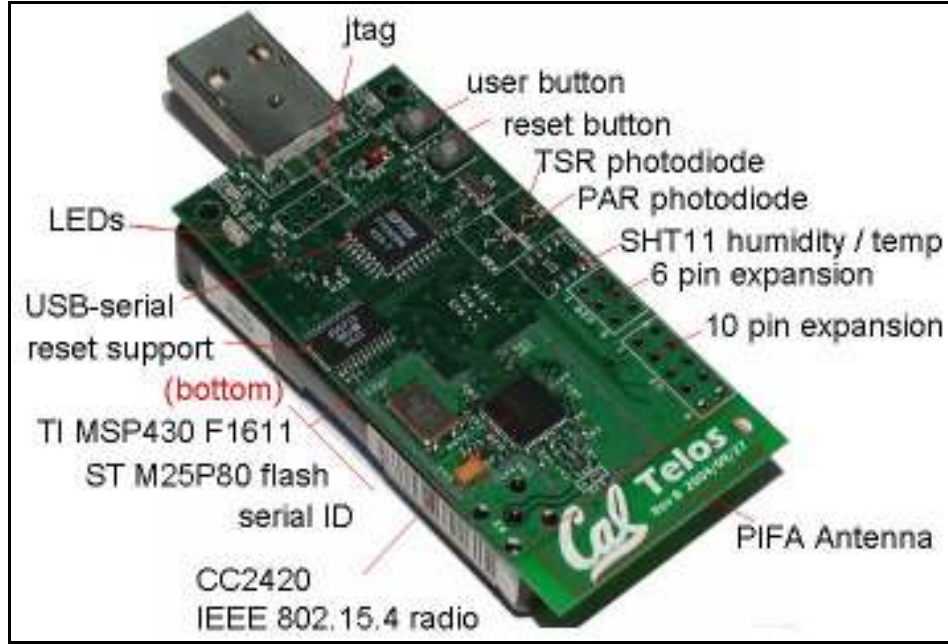


Figure A-1 – Main Components of Crossbow TelosB

Figure A-1 shows the main components of the TelosB sensor platform. TelosB is an open source, low-power wireless sensor module designed to enable experimentation for the research community. The TelosB bundles all the essentials for lab studies into a single platform including: USB programming capability, an IEEE 802.15.4 compliant, high data rate radio with integrated antenna, a low-power MCU with extended memory and an optional sensor suite. A summary of important features of TelosB can be found in Table A-1.

Table A-1 – CrossBow TelosB Mote Platform Specifications

<i>Radio</i>	IEEE 802.15.4 (ZigBee)
<i>Data rate</i>	250 kbps
<i>Microcontroller</i>	TI MSP430
<i>RAM</i>	10 kB
<i>Transceiver chip</i>	CC2420
<i>On-board sensors</i>	Temperature, light, and humidity
<i>Programming</i>	USB

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

ACM	Association for Computing Machinery
AFRL	Air Force Research Laboratory
BS	Base Station
CSMA (w/ CA)	Carrier Sense Multiple Access (w/ Collision Avoidance)
GRG	Geometric Random Graph
IEEE	Institute of Electrical and Electronics Engineers
ILC	Intentional Loop Creation
LED	Light Emitting Diode
MAC	Medium Access Protocol
MCU	Microcontroller Unit
NR_B	Number of Packets Received by Node B
NS_B	Number of Packets Sent by Node B
P2P	Peer-to-peer
RF	Radio Frequency
SA	Situational Awareness
SARP	Secure Adaptive Routing Protocol
SF	Selective Forwarding
SINR	Signal to Interference and Noise Ratio
TDMA	Time Division Multiple Access
TI	Texas Instruments
TOSSIM	TinyOS (Mote) Simulator
UART	Universal Asynchronous Receiver/Transmitter
URI	University of Rhode Island
USB	Universal Serial Bus
WPAN	Wireless Personal Area Networks
WSN	Wireless Sensor Network
Xbow	Crossbow