ENERGY CONSUMPTION DURING ATTACKS AND COUNTERMEASURES IN SENSOR NETWORKS

BY

KARTHIKRAM SHESHACHALAM

Bachelor of Engineering and Computer Science

Annamalai University

Annamalai Nagar, INDIA

2002

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2006

ENERGY CONSUMPTION DURING

ATTACKS AND COUNTERMEASURES

IN SENSOR NETWORKS

Thesis Approved:

Dr. Johnson P Thomas

Thesis Adviser

Dr.Venkatesh Sarangan

Dr.Debao Chen

A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to convey my sincere gratitude to my thesis advisor Dr. Johnson P Thomas for the guidance and encouragement that he gave me throughout my thesis work.

I would also like to extend my appreciation and gratefulness to my committee members Dr. Venkatesh Sarangan and Dr. Debao Chen for their support and advice.

I would like to thank my parents Mr. Sheshachalam,Mrs. Renuka Sheshachalam and my brother Mr. Ajay Ram for their love and support throughout the years.

Finally I would like to thank all my friends who stood beside me with their unfailing and indispensable support.

TABLE OF CONTENTS

Chapter **Page**

LIST OF FIGURES

Chapter **Page**

LIST OF TABLES

Chapter **Page**

1. INTRODUCTION

1.1 Sensor Networks

Sensor Networks are formed with a number of small low cost sensor devices which connect wirelessly to gather data. A sensor can obtain data from it's surrounding and transmit to a central base station where it can be processed into meaningful information. Common applications include defense systems and environment monitoring. These use multi-hop routing similar to Ad-hoc wireless techniques, to communicate with one another which allows a certain degree of mobility and flexibility. They can be deployed in a number of ways with a multitude of configurations in large numbers, depending on the application they are intended for.

Sensors used in the real world have a number of limitations in resources and capabilities. Research conducted in these areas aim at enhancing or improving on these limitations. Most of the work done is in improving the life of the power source available in the sensor which is limited with what's available during deployment and improving aspects of the network and communication layers. The limited resources available to process and store the collected data is also a challenging problem

1.1.1 Energy and Power Consumption

Power consumption is a major focus for many applications as power is a very limited resource. Power supply for the sensors is through a battery source that is not replenished. Sensors rely on periods of inactivity to conserve and lower their power consumption. Data collection and computation is not done constantly but in periods of relevance. They do this in aggregation where only some nodes remain in the active state and some of them go into a passive mode to conserve power. The Tiny OS beaconing protocol is used to operate the sensors and nodes. A breath first spanning tree is constructed to map all the nodes in the vicinity and they are constantly updated with each hop [1].

Security is an important criterion after power. A secure network would work more reliably with minimal threats from intrusions [5]. Tradeoffs between power and security are usually the deciding factor in a successful sensor network deployment. Security may not be an important factor in a building structural change monitoring sensor where there is little or no threat but power consumption is an important factor. However in a battlefield where security plays an important role, the options are usually low computation intensive algorithms to protect inter node communication. Some of the more complex forms of encryption are not feasible because of the limited computation available and the possibility of breaking into a node which can be physically compromised. Some of the security related efforts are aimed at improving, routing protocols to protect sensors from malicious attacks and being energy efficient too. A lot of new research has proposed various security mechanisms for sensor networks. Various countermeasures in response to attacks have been proposed. However, as far as we are aware, no one has investigated the energy consumed in activating and executing these various countermeasures. If a countermeasure requires

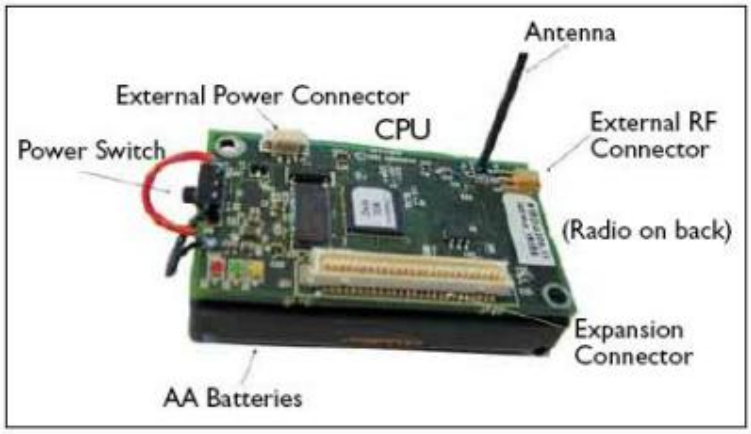
large amounts of energy, it may not be economical. On the other hand, there may be countermeasures for other attacks which may not require a lot of energy. Knowing the energy consumed in defending against attacks will help the sensor network designer in deciding which countermeasures should be encoded into a sensor. As far as we are aware, this is the first work to measure the energy consumed in defending individual sensor nodes against different kinds of attacks. The results from this work can be extrapolated to provide defense mechanisms for a whole sensor network.

The thesis concentrates on documenting power drain in various attacks that occur on sensor networks in a real world scenario and to apply relative countermeasures to recover from these attacks. Power levels of the sensors are constantly monitored to determine the best solution. Some of the work performed on monitoring power levels are on simulators with communication rates and node duty cycles without considering the power consumed by devices on the sensors themselves. Some of the real world scenarios may usually reveal a lot more issues than a typical setup in a simulator.

1.1.2 Sensor Overview

A sensor network is a combination of the central data gathering unit called a base station and several sensors. The number usually ranges from 100~1000 units. Some of these sensors have a multitude of sensing capabilities like temperature, pressure, location (GPS), altitude, movement and light. The configuration and deployment preferences depend on the application at hand. The base station serves as the gateway to collect data from the nodes and channel it to the data processor. The base handled computation has better resources than the sensors themselves.

Nodes communicate in a multi hop approach. During an initial deployment phase the nodes are randomly deployed around a base station and a trust relationship is formed where all the nodes start communicating with the base station. Normal working period of a node is usually from 2-3 months with the battery pack normally equipped with the nodes.



Typical Sensor node manufactured by Crossbow.
Figure 1

Although several sensors are available, in this thesis we work on the Mica Mote sensors. This sensor has good simulation support and is commonly used for research purposes.

The processor used, is an Atmel 8bit CPU with a 4 MHz clock speed, 4 KB or RAM and 512 KB of cache memory[8]. The power consumption of the CPU is 5.5mA (3V) when active and half that amount when it is in the sleeping mode. The node is equipped with a radio beacon that has a range of around 250 – 300 meters [6]. Crossbow manufactures the motes with various sensor setups. Research is ongoing to improve sensor capabilities and communication protocols.

Tiny OS is an operating system developed by Berkeley to operate the sensors. The TinyOS beacon protocol is used for communication for the sensors. A typical setup can have more than one base station and a node can be associated with more than one base station at a time. Power management is a critical issue for sensor networks which depend on very limited power supply (two AA type batteries for the Mica Mote sensors). Prolonging the power is typically achieved by using energy efficient routing mechanisms. Sensors enter constant idle moments or sleep states to conserve power. Some event triggers the sensors out of sleep mode into active mode. In active mode each event or task requires precious power and drains the batteries.

Besides energy conservation, the other major issue facing sensor networks in security from an adversary. Attacks can occur when the sensors least expect them and the applied countermeasure may require substantial power expenditure. In this thesis some of these attacks and countermeasure are studied with a provision to monitor the remaining power levels. Effects of various attacks reveal information about the outcome of these attacks and the success of these countermeasures.

1.1.3 Security Issues

Since the nodes in a sensor network lack capability and power capacity, communication typically takes place through an insecure channel with an emphasis on low power consumption and low radio communication. This makes it very susceptible to attacks. An adversary can take control of the sensors physically and snoop at the information available or modify certain parameters in the sensor to take control of the entire network. Alternately an attacker can override the built in security measures to cause inexplicable damage from which a network cannot recover. An attacker can also obtain information about the details of data capture and its contents, both of

which could prove damaging to the network. Some of the secure routing protocols like distance routing and source routing protocols used in Ad-Hoc networking are not suitable in sensor networks because of their incapability to sustain the overheads involved. Other secure protocols can be deployed but the trade-off might be to sacrifice either lifetime of a normal working sensor or some of its other capabilities.

Some of the best solutions to secure routing are at the application layer where an adversary can be prevented by data checking and correction. Some of the attacks are described here with countermeasures.

In Chapter 2 we review the various attacks on sensor networks and the corresponding countermeasures. Chapters 3 and 4 describe the methodology and implementation details. This includes the tools used to simulate our attacks and countermeasures. Chapter 5 provides the results of the experiments. The conclusions are given in chapter 6

2. LITERATURE REVIEW

2.1 Attacks

Sensor routing protocols are primarily designed with energy efficiency and not security in mind. Consequently, they are susceptible to most of the attacks. The attacks can be classified into the following categories [1].

2.1.1 Spoofed, altered or replayed routing information

An adversary can spoof, alter or replay routing information communicated to a target node to node communication. This form of attack is common as it does not take much to achieve; rather it involves intercepting information being exchanged between nodes and relaying them. A successful attack may involve creating routing loops, generate false error messages or partitioning the network into incompatible nodes or increase the end-to-end latency between nodes.

2.1.2 Selective forwarding

In selective forwarding the attacker needs complete access to the path of the data flow. An adversary can successfully jam or cause collision to the data thereby causing disruption to the data flow. Multi-hop networks are often based on the assumption that these participating nodes will faithfully forward messages received. Malicious nodes may refuse to forward certain messages and simply drop them, ensuring that they are not propagated any farther. A simple form of this attack is when

a malicious node behaves like a black hole and refuses to forward every packet she sees. However, such an attacker runs the risk of neighboring nodes determining that she has failed and deciding to seek another route. A more subtle form of this attack is when an adversary selectively forwards packets. An adversary interested in suppressing or modifying packets originating from a select few nodes can reliably forward the remaining traffic and limit suspicion of any wrongdoing. The other attacks where an attacker can include in the path of the data flow are Sybil attack and sinkhole attack.

2.1.3 Sybil attacks

A Sybil attack is created when a malicious node assumes the identity of another node on the network. A node illegally assumes multiple identities either forging an identity or stealing legal identities. A simple way to detect a Sybil attack is to verify the identity of each node and confirm its integrity [14]. Without a central system verifying the identities it becomes easy for a malicious node to infiltrate a network.

2.1.4 Sinkhole attacks

Sinkhole attacks are similar to selective forwarding, where a rogue node attracts all the traffic from neighboring networks depending on the routing algorithm. A metaphorical sinkhole is created with the adversary in the center luring all the application data towards it. Sinkhole attacks can also cause many other attacks like selective forwarding because of the inherent nature of their presence near the path of the data.

Routing data away from the sink is possible by replaying an advertisement from a previously good quality link. Some protocols might actually contain reliability or latency information. An adversary with a palm pilot or a laptop which is much higher powered than a sensor sink can provide a higher link by transmitting enough power that the nodes might think that the base station can be reached in one hop thereby deceiving the others into thinking that they are nearer to the base station to which they need to report. This information is later propagated to its neighbors and traffic is eventually diverted. Effectively, the adversary creates a large “sphere of influence” [1], attracting all traffic destined for a base station from nodes several (or more) hops away from the compromised node.

One motivation for mounting a sinkhole attack is that it makes selective forwarding trivial. By ensuring that all traffic in the targeted area flows through a compromised node, an adversary can selectively suppress or modify packets originating from any node in the area. It should be noted that the reason sensor networks are particularly susceptible to sinkhole attacks is due to their specialized communication pattern. Since all packets share the same ultimate destination (in networks with only one base station), to influence a potentially large number of nodes a compromised node need only to provide a single high quality route to the base station.

2.1.5 Wormhole Attack

A Wormhole Attack is caused by an attacker who tunnels packets at one point to another point in the network, and then replays them into the network from that point. The wormhole attacks can form a serious threat in wireless networks, especially against many routing protocols. The simplest instance of this attack is a single node

situated between two other nodes forwarding messages between the two of them. Since the tunneled distances are longer than the normal wireless transmission range of a single hop, the source will prefer the path including the attack nodes. Then the attack nodes may perform various attacks, such as the black hole attacks (by dropping all data packets) and grey hole attacks (by selectively dropping data packets). Because the wormhole nodes do not modify or fabricate packet, cryptographic techniques cannot detect this type of attack. It is a severe attack that is particularly challenging to defend against. Methods to defend a wormhole attack are by adding information about geography or time to a packet to restrict the packets maximum allowed transmission distance.

In some cases, an adversary situated close to a base station may be able to disrupt routing completely by creating a well-placed wormhole. An adversary could convince nodes who would normally be multiple hops from a base station that they are only one or two hops away via the wormhole. This may create a sinkhole due to the potential attractiveness of the route created by the wormhole [1]. Those nodes neighboring the adversary on the other side of the wormhole may choose to forward packets destined for a base station through this route. The result would be to propagate knowledge of this route to their neighbors and attract more traffic.

Wormholes become particularly difficult when combined with either a Sybil or a selective forwarding attack as they are similar and detection is extremely difficult.

2.1.6 HELLO flood attack

This is similar to the Sybil attack, except it takes advantage of a basic characteristic in the protocol. Many protocols require nodes to broadcast HELLO

packets to announce themselves to their neighbors, and a node receiving such a packet may assume that it is within range of the sender. Hence a laptop-class attacker with a powerful radio can send out a strong transmission to convince every node on the network that the adversary was the neighbor. In the preceding mayhem, every node will try to use this route, but those nodes sufficiently far away from the adversary would send packets into oblivion. The network is left in a state of confusion. A node realizing the link to the adversary is false could be left with few options: all its neighbors might be attempting to forward packets to the adversary as well [1]. Protocols which depend on localized information exchange between neighboring nodes for topology maintenance or flow control are also subjected to this attack. HELLO floods can also be thought of as one-way, broadcast wormholes.

Even though this is a form of flood attack as seen earlier HELLO attack uses a single hop broadcast to transmit a message to many receivers to cause a state of confusion.

2.1.7 Acknowledgement spoofing

Spoofing is a form of deceiving a node into thinking what's not true. For example a weak link may be portrayed as a strong one or a dead node is shown as a live node. Several sensor network routing algorithms rely on implicit or explicit link layer acknowledgements. Due to the inherent broadcast medium, an adversary can spoof link layer acknowledgments for overheard packets addressed to neighboring nodes. A routing protocol may select the next hop in a path using link reliably. Artificially reinforcing a weak or dead link is a subtle way of manipulating such a scheme.

Packets sent along those paths will not reach its destination and will inherently be lost, thus launching such an attack an adversary can cause enough data loss. Hence besides that selective forwarding attacks can be launched by spoofing.

2.2 Counter-measures

2.2.1 Spoofed and altered attacks.

Since base stations are trustworthy, adversaries must not be able to spoof broadcast or flooded messages from any base station. This requires some level of asymmetry since every node in the network can potentially be compromised, no node should be able to spoof messages from a base station, and every node should be able to verify them [1]. Authenticating broadcast is also useful for localized node interactions. Using a global shared key reduces a number of attacks, such as by a simple link layer encryption and authentication with a global and shared key. Only nodes from the same group would have access to the global key and, hence message authentication can be performed by this method. Spoofing and altering packet information can be avoided.

This prevents replay attacks and changing data packets. A counter is maintained to monitor each link and the next value of the counter is checked to see if messages are being replayed. The counter keeps track of identical messages received from each node. Each node simply remembers the most recently received counter value from each of its neighbors and discards packets containing older values. These mechanisms are enough to counter most of the discussed attacks when mounted by

outsiders. Most of selective forwarding and sinkhole attacks are not possible because the adversary is prevented from joining the topology.

2.2.2 Selective forwarding

Link layer security can provide some degree of security by validating some of the messages using authentication. A selective forwarding attack is followed by a malicious node not forwarding certain messages. The most vulnerable nodes are those that are near the base station. Multi path routing counters these types of selective forwarding attacks [1]. Messages routed over paths whose nodes are completely disjoint are protected against selective forwarding attacks involving at most compromised nodes and still offer some probabilistic protection when other nodes are compromised. However, completely disjoint paths may be difficult to create. Nevertheless link layer security is totally useless with other attacks like wormholes and Sybil attacks.

2.2.3 The Sybil attack

Several solutions have been proposed including symmetric key cryptography. Some of these forms require a lot of computation and processing power. With some authentication in the link layer a Sybil attack can be thwarted. A pair of neighboring nodes can use a shared key to implement an authenticated, encrypted link between them. However an adversary trying to infiltrate the network from inside cannot be stopped. To prevent an insider from wandering around a stationary network and establishing shared keys with every node in the network, the base station can reasonably limit the number of neighbors a node is allowed to have and send an error message when a node exceeds it. If a node is compromised the damage is limited to a

set of known neighbors [2]. In a link layer authentication mechanism the Sybil attack is no longer relevant because nodes are unwilling to accept even a single identity of the adversary. Even with a wormhole an artificial link can only create an unreliable path between two paths but the information cannot be compromised in this method. The damage is limited to low latency between the node and base station and not for the message loss themselves.

When the network size is limited or the topology is well-structured or controlled, global knowledge can be leveraged in security mechanisms. In a relatively small network of around 100 nodes or less, it can be assumed that no nodes are compromised during deployment, then after the initial topology is formed, each node could send information such as neighboring nodes and its geographic location -if known back to a base station. Using this information, the base station can map the topology of the entire network. To account for topology changes due to radio interference or node failure, nodes would periodically update a base station with the appropriate information.

2.2.4 HELLO flood attacks

When compared to linked layer security mechanism, an authenticated code in the packet cannot stop a HELLO attack from distributing false node messages. It would up to a degree stop forwarding and replaying but has no effect on other type of attacks.

Link layer security mechanisms using a globally shared key are completely ineffective in presence of insider attacks or compromised nodes. Insiders can attack the network by spoofing or injecting bogus routing information, creating sinkholes, using the Sybil attack, and broadcasting HELLO floods. An identity verification

protocol to verify the nodes identity to use a shared key would be more than sufficient in preventing a HELLO flood attack. The simplest defense against HELLO flood attacks is to verify the bi-directionality of a link before taking significant action based on a message received over that link. A base station limitation could also be used to limit the number of trusted nodes from exchanging information thereby preventing a large number of HELLO messages from interfering with its performance. This is also done by verifying the bi-directionality of the link between two nodes.

2.2.5 Sinkhole attacks

A significant challenge in securing large sensor networks is their inherent self-organizing, decentralized nature. Sinkholes are particularly vulnerable in protocols that use remaining energy or reliability information between nodes because this information is hard to verify from actual value. In a wormhole attack, the out-of-band channel not being available to the sensor network could compromise a network [18]. Routes that minimize the hop count to a base station are easier to verify, however hop-count can be completely misrepresented through a wormhole. When routes are established simply based on the reception of a packet as in TinyOS beaconing or directed diffusion, sinkholes are easy to create because there is no information to verify the validity of a sinkhole. Probabilistic selection of a next hop from several acceptable destinations or multi path routing to multiple base stations can help with this problem, but it is not perfect. When a node must route around a “hole”, an

adversary can “help” by appearing to be the only reasonable node to forward packets to [1].

The compromised node advertising its location on a path between the targeted node and a base station will guarantee it is the destination for all forwarded packets from that node. Sufficiently restricting the structure of the topology can eliminate the requirement for nodes to advertise their locations if all nodes’ locations are well known. For example, nodes can be arranged in a grid with square, triangular, or hex shaped cells. Every node can easily derive its neighbors’ locations from its own, and nodes can be addressed by location rather than by an identifier

2.2.6 Wormhole countermeasures

The most elaborate form of worm hole checking and countermeasures are to obtain a statistical analysis of routes obtain during route discovery and send an ACK on suspicious paths to check for a wormhole, but this method could be more processor intensive and also not power efficient requiring constant time synchronization.

The other possible method is to use geographic location models to document wormhole attacks. Since we know how data moves we can be sure when a suspicious route is created and data diverted. Artificial links are easily detected in geographic routing protocols because the “neighboring” nodes will notice the distance between them is well beyond normal radio range. Drastic or suspicious changes to the topology might indicate a node compromise, and the appropriate action can be taken .Wormhole is most effective when used to create sinkholes or artificial links that

attract traffic. Geographic routing can be effective in stopping many attacks but that location information is trusted. Apart from Geographic routing methods, directional antennas can also be used to prevent a wormhole attack. Nodes share directional information to avoid being compromised by false neighbors using endpoints [15]. Nodes with one hop accessibility maintain accurate information about their proximity locations and the direction other nodes are located. In this method, zones are created and nodes communicate in separate channels in each zone thus eliminating malicious false neighbors.

3. METHODOLOGY

3.1 Sensor attacks and their effects on power and performance

With the increasing applications of sensors, it becomes more critical to protect sensor networks from malicious attacks. As seen in the previous sections, potential attacks on sensor networks have been identified and furthermore, counter measures to these attacks have been proposed. However, there is no work reported in the literature on the impact of these attacks and counter measures with relation to energy and performance. As energy is severely limited in sensor networks, the impact of these attacks on energy consumption is critical. Furthermore, the energy consumed in protecting or defending these networks is also a critical factor, since high energy consumption approaches to protect sensor networks may not be a practical proposition [3]. The validity of the proposed counter measures is therefore in question until their energy consumption is determined.

In this thesis we propose to measure the change in energy consumption on nodes due to malicious attacks as well as the energy overheads in defending against these attacks. Furthermore, we also propose to investigate the impact caused by these attacks and the subsequent overheads in the counter measures.

In normal conditions the above mentioned sensors operate at full operating capability at 100% power load for a maximum of two weeks [4]. If an attack occurs and sensors are compromised the recovery could be expensive in terms of energy. This is important as the nodes cannot be replaced. Obtaining the energy and performance measures will be essential to determining the best solution for recovery and countermeasure.

Given that counter measures consume energy, it may make sense not to commit a countermeasure and instead quarantine the affected sensors thus saving the current network from unsafe affected nodes and conserve power. Because of an attack some extra precautionary measures could be employed to avoid either spreading the same attack or containing it to certain areas.

Measurement of attack effects and the application of related counter effects can be documented over a period of time frame; this could give us an average working model of the sensors in real time. Sensor networks use Tiny OS which is an event driven operating system. Because of its simplicity the network lacks the security required in an otherwise secure network. They are vulnerable to most of the attacks previously discussed.

In this thesis we will model the following attacks and counter measures and document their power profiles.

3.2 Simulation of Attacks and countermeasures

HELLO flooding attack:

Motes are configured with a program to listen and perform the initial HELLO handshake. A powerful base station with the capability to reach directly out to the nodes is deployed after the network is formed and repeated requests for neighbor information is requested from the motes, causing them to flood the networks with HELLO packets.

Countermeasures are followed up immediately by resetting the motes to load the default program, thus dropping the flood packets. It will restrict the number of connections a base station or a mote can accept. A counter keeps track of the HELLO

packets received from and to the mote. After a few minutes of restricted operation a counter *relax* that allows more connection between motes and return to normal operating conditions. Replay attacks can also be avoided with this mechanism.

A typical Operational Hierarchy:

- Deploy motes with the basic network program loaded in them connected to base station A
- Deploy another adversary base station B which is more powerful than most of the other motes in the network to simulate a HELLO attack.
- Attack becomes evident and data is logged to document effects when the counter shows increased network activity.
- Countermeasure is to load a modified program which would limit incoming and outgoing connections on motes and base station.
- Counter deployed to keep track of messages received

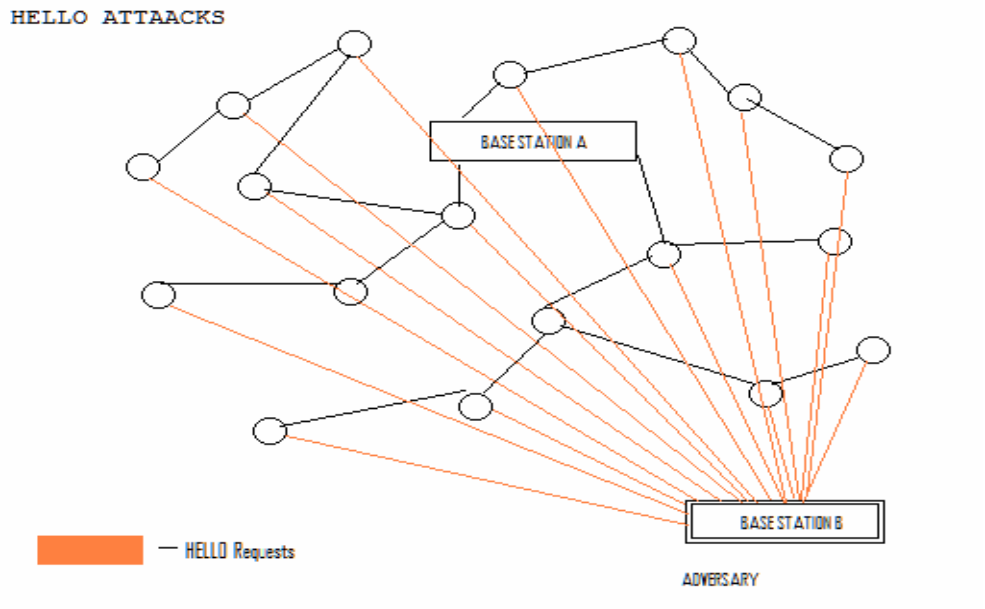


Figure 3.1: HELLO Attacks

Sybil Attack:

Sybil Attacks can be simulated by deploying an *adversary node* on a network which assumes one of an existing mote's identities. Attack detection is assumed when packets are seen dropping at the base station when the adversary node is started [17].

Countermeasure to a Sybil attack is to check the number of connections made by the base station at any given time and compare with the node ID which forms part of the header data transmitted by the nodes. Neighbors ID's can be logged and compared to find cloned node connections. Such comparisons and computations are intensive on energy consumption and can be documented.

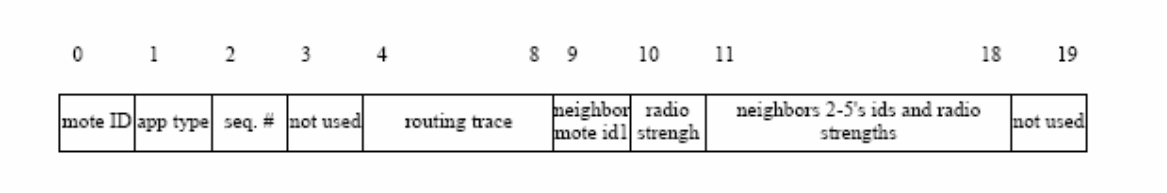


Figure 3.2: Sybil Attacks

Selective forwarding attacks:

In selective forwarding attacks nodes that appear near the base station are targeted which are assumed to have a lot of data due to their proximity to the base station.

Countermeasure: In an attack, nodes which are assumed to be affected are made to use a multi-path routing approach to the base station to deliver packets. In our example nodes A to D represent those that lie close to the base station and vulnerable to a selective forwarding attack. If node D is compromised packets coming to the base station are sent in a multi path route through nodes A through C. Any discrepancy can

be detected by comparing packet count received from A, B and C with that of D. This would confirm a selective forwarding attack.

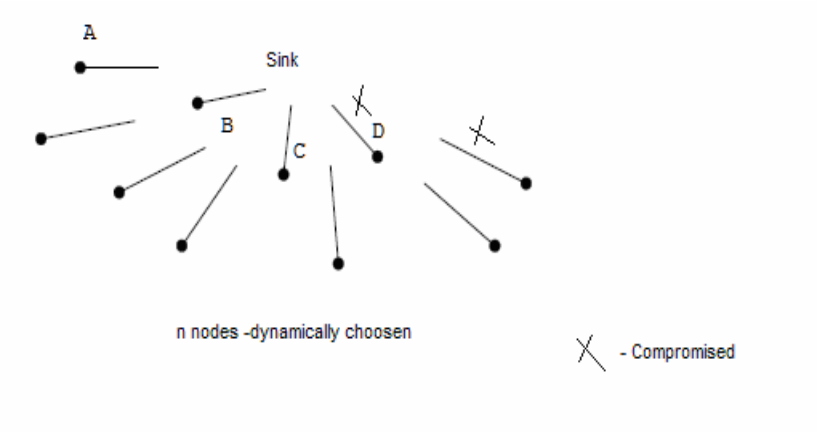


Figure 3.3: Selective Forwarding Attacks

4. IMPLMENTATION

4.1 Simulation and Environment setup

A typical setup for the experiment would involve around 8-9 motes, devices which form the sensor component and a radio component with a battery. They are deployed over a wide area where they form a multi hop connection with one another and eventually connect to a base station. The number of base stations in our experiment is assumed to be one. The motes are separated a distance 10-15 feet and spread wide apart so that they do not connect directly to the base station like an access point in an ad-hoc network. The most commonly used motes for research purposes is called Mica2 and manufactured by a company called Crossbow. A computer loaded with Tiny OS and a suite of test programs to measure throughput and power consumption is configured. The base station is connected to this computer to enable data transfers between the motes and the computer.

Motes have limited memory capacity and computational power, with the capability to execute one application at a time. *Surge-view* is a program written by Crossbow the manufacturer of these motes, to help in forming a very basic topology connecting all motes in the network. Once the network is formed using *Surge-View,* the test program we developed to simulate an attack and counter measure are loaded into the motes. The programs are loaded and run to obtain performance measures.

Each experiment is carried out for 10 minutes, to enable the network to configure properly and give a consistent output. Sometimes readings that are taken

within a short starting time tend to be skewed. On the real motes we need to wire extensive measuring instruments like a multi meter to *each mote,* so that power consumption changes can be gathered in real time. However, before the motes are used for the actual experiment, we develop our programs on a simulator like POWER-TOSSIM or Avrora because of the flexibility achieved in the development process. The programs hence developed using the simulator is compatible with any mote *without* any modification. . Hence these programs can be loaded into the motes without any modifications and executed. A simulator like Avrora helps us in calculating power readings without using multi meters to measure power consumption.

Tools to be used in performing our simulation:

- Tiny OS Operating system developed by UC, Berkeley
- Programs developed in nesC which is an embedded system development environment.
- Data Logging application to gather our data, built on Tiny OS.
- AEON (Accurate Prediction of Power Consumption) Plug-in to Model power consumption. AEON is integrated with AVRORA – A simulation and Analysis Framework tool.

Methods to measure or document power consumption – On a real network the following are needed:

- Multi-meter reading at constant intervals.
- Surge view and Oscilloscope program developed by Crossbow.
- Power-TOSSIM like simulation environment [3].

4.2 Network composition

The composition of network:

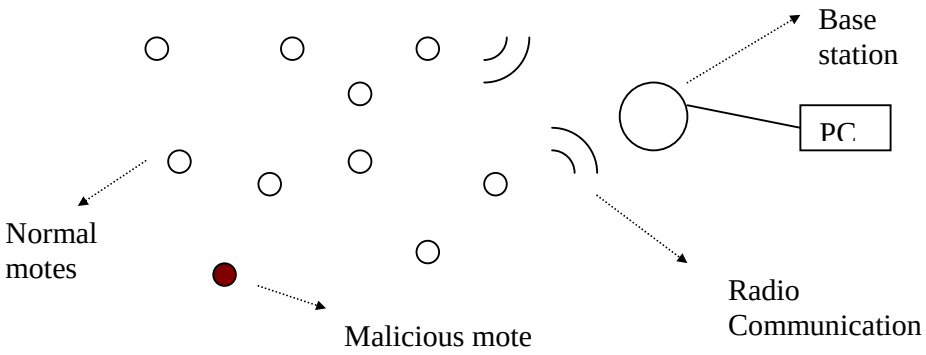


Figure 4.1: Network composition

In all the experiments is with 8-9 Motes running the surge program and one malicious mote representing the attacker.

Normal motes

- All normal motes run a program named “SpoofSurge”. The surge application is a simple multi-hop application. It takes light sensor readings from the motes and sends the data to the base station.

Malicious mote

- For Example: Runs a program named “SpoofReplay”, which is a modified version of *Surge-View* program to simulate a Spoof Attack.

Base station

- Also runs SpoofSurge application. The address of the base station is 0. It receives messages and forwards them through a UART. This node is connected to a PC via a serial link. The PC runs a version of the Tiny OS operating system.

The Attacks are simulated separately with normal versions of the programs running on some motes, and the countermeasure programs are tested with the Attack programs. Several configurations are explored in the experiments. These included choosing parameters such as percentage of attack and the ratio of normal motes to countermeasure and attack motes.

4.3 Data Analysis

Throughput of data packets between the base station and the motes is logged in our data logging application. Other important indicators of network performance like routing overhead and Average End-to-end delay are also calculated.

Measurement conditions and required observations:

- Before an Attack under normal operating conditions.
- During an Attack.(Measure data loss)
- After an Attack (Measure power levels and mote states)
- During counter measure implementations (Power levels)
- After Counter measures are successfully implemented – current state of the network.

Values are charted on a graph where the data could help us determine overheads especially when counter measures are applied. These results would indicate performance degradation in terms of data loss and slow transmission rates. Thus the tradeoffs using a particular approach could be documented.

Power consumption of a sensor node is the important performance characteristics of interest. Considering that the power supply is at a constant voltage throughout the operation (3V), the power consumed is proportional to the leaked

current [7]. We can therefore compare power consumption of the sensors before, and after an attack, and the power consumed when deploying countermeasures. Because power levels in a sensor could indicate the remaining life of a sensor, it should be constantly monitored.

Under normal operating conditions transmitting consumes more power than receiving. A special power down mode is even thriftier in saving power than the idle state of a sensor.

State	Leaked current (mA)
Reception	12 ~ 16
Transmission	14~18
Idle (radio rx mode)	11 ~ 15
Idle (radio tx mode)	13-15
Idle (radio off)	8 – 8
Idle (power down)	0.0 ~ 0.1

Table 3.1: Power consumption in different operating modes (crossbow)

Power supply voltage remains in a constant flux. Voltage has a linear effect on current and a quadratic effect on power and energy. Voltage fluctuations can lead to constant changes in energy use. Hence measuring voltage alone does not give us a clear idea of a sensors power consumption level.

4.4 Network communication on TinyOS

TinyOS (TOS) is the preferred operating system for wireless embedded sensor networks. The Open source community maintained it with some active participation from the University of California at Berkeley it has matured into a very stable

platform.

It supports a wide range of sensors with drivers, data acquisition tools and programs necessary to run them. The time and effort required to develop in this environment is very minimal due to its component-based architecture [7]. TOS uses active message (AM Model) as defined in $TOS/system/types/AM.h to communicate. Message is “active” when it contains destination address, group id, and type.

TinyOS has NesC as the programming language, and most of the programs are developed in NesC.

4.5 Architecture

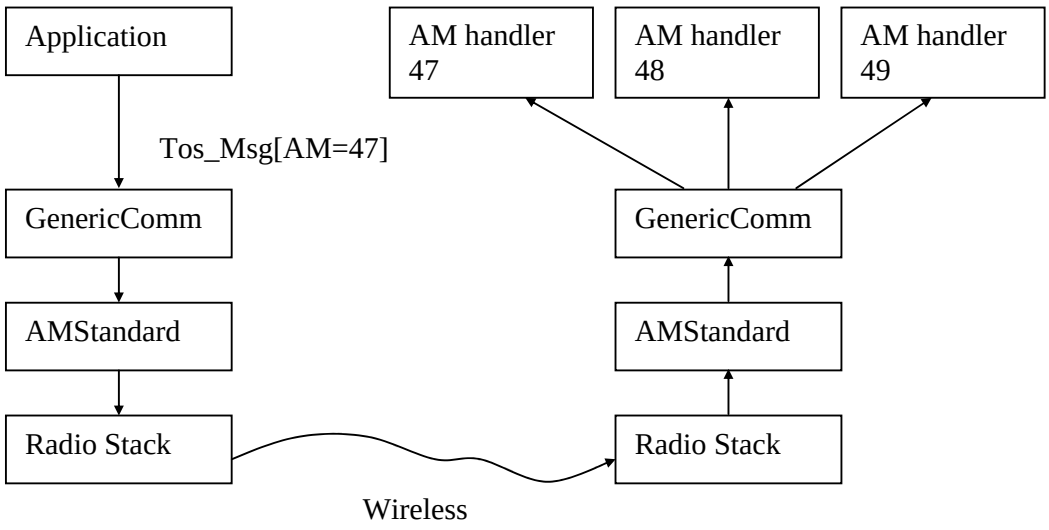


Figure 4.2: TOS communication architecture [16]

We provide an example of the active messaging standard. Components are wired together by connecting users with providers, which forms a hierarchy. In the example Commands flow downward from the application to the radio stack, and events flow

upward to the AM handler from the radio stack. The TOS Thread model is classified into Tasks and Events. Tasks being time flexible and support longer background processing jobs and events are time critical and of shorter durations. As of with Tiny OS version 1.1 the number of concurrent tasks supported are 7.

4.6 TOS communication paradigm

The Network is modeled as a pipeline with minimal buffering for messages to get the maximum efficiency [9].

Send message

To send a message a buffer needs to be filled with data, and the address of the recipients to whom it has to be sent. This Information is passed onto the operating system where it is used by the applications. It is also determined whether this information can be reused again.

Receive message

Incoming message would fill a buffer and the application is notified of the new message's arrival. The application fetches the data from the buffer. The memory management for this operation is dynamic, and the buffer is cleared automatically for the next message to be received.

4.7 Multi-hoping routing

A multi-hoping algorithm is used to route messages in the network. An Active Message is an alternative method to handle network messaging in Tiny OS instead of TCP/IP. The figure below describes a Multi-Hop network.

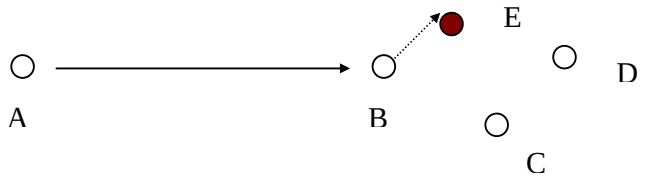


Figure 4.3: Message forwarding multi-hop protocol

Route discovery is done by shortest path from every node to the base station. The base station broadcasts its identity from time to time. A table is maintained with their hop counts from the base station. A parent node is determined by the closest node discovered during this phase. Node A sends an AM (Active Message) packet to node B, B being the parent on the list for node A. When B receives the data packet successfully it will read the destination address to see if this message is for itself. If the destination is not B, then B will forward it to a parent node on the routing table.

Suppose nodes C, D and E are “neighbors” of node B. Now B forwards the packet to its “parent” which is the node most likely on its list to receive data. In this case, E is B’s parent.

B will update its neighbor list periodically by the hop count returned from other nodes, When E stops responding to B, E is determined not to be a reliable node to receive data and act as a “parent”, hence another node is selected to be the new parent of B which can be C or D, depending on their proximity to Node B. Nodes look for parents with minimum hop count from the base station. This process is done periodically to find the best possible route for data.

Data packet format

Header (5bytes)	Payload (max 29bytes)	CRC (2 bytes)

Data structure definition

TOS uses active messages as defined in AM.h header. Message is active because it contains a destination address, groupID and type [16]. A message consists of the Header with handler name and data payload as argument. The following excerpts of a typical program explain the part involved in managing Radio communication. This allows the user to control the radio strength, acknowledgment and security modes.

```
typedef struct TOS_Msg
{
  /* The following fields are transmitted/received on the radio. */
  uint16_t addr;
  uint8_t type;
  uint8_t group;
  uint8_t length;
  int8_t data[TOSH_DATA_LENGTH];
  uint16_t crc;

  /* The following fields are not actually transmitted or received
   * on the radio! They are used for internal accounting only.
   * The reason they are in this structure is that the AM interface
   * requires them to be part of the TOS_Msg that is passed to
   * send/receive operations.
   */
  uint16_t strength;
  uint8_t ack;
  uint16_t time;
  uint8_t sendSecurityMode;
  uint8_t receiveSecurityMode;
} TOS_Msg;
```

4.8 Using Avrora simulator and AEON plugin

Compared to Power-Tossim and the Emstar simulators, Avrora has the ability to measure power consumption as well as simulating a sensor network at the same time. TOSSIM is a single application simulator and cannot simulate a collection of nodes on a network. For example when a single application like “surge” is run, TOSSIM will spawn 10 separate threads to run the program on 10 nodes. On Avrora with AEON, the results returned are accurate to the component level.

4.8.1 Step 0 – Download Avrora

Avrora is available from UCLA’s website. It is a simulation program written mainly for the AVR and mica2 platforms. [12].

4.8.2 Step 1 – Install java

Avrora is written in java (platform independent). Pre-requisite is to install java SDK. (Used Java Version j2sdk1.4.2)

4.8.3 Step 2 – Compile the TinyOS program

The Avrora only supports mica2 platform. Since a binary image composed of ELF and SREC files cannot be directly loaded into Avrora, it is converted into a textual format before it can be used. This is done as follows:

```
$ cd  $TINYOS_ROOT/apps/SpoofSurge

$ make mica2

mkdir -p build/mica2
    compiling SpoofSurge to a mica2 binary
ncc -o build/mica2/main.exe -Os -board=micasb -target=mica2 -I%T/lib/Queue -I%T/
lib/Broadcast -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all -finline-limit=1
00000 -fnesc-cfile=build/mica2/app.c  SpoofSurge.nc -lm
RouterLEPSM.nc:633:2: warning: no newline at end of file
C:/cygwin/opt/tinyos-1.x/tos/system/RealMain.nc: In function `main':
C:/cygwin/opt/tinyos-1.x/tos/interfaces/StdControl.nc:63: warning: `result' migh
t be used uninitialized in this function
    compiled SpoofSurge to build/mica2/main.exe
```

4.8.4 Step 3 – Disassemble the program

After obtaining the main exe file, the “avr-objdump” tool is used to disable textual format file.

```
$  avr-objdump –zhD ./build/mica2/main.exe > SpoofSurge.od
```

Note that the extension of the output file must be “.od”.

We can obtain SpoofReplay.od in the same way. (Before compile, set TIMER_RATE=100)

4.8.5 Step 4 – Run programs

Now we have the programs which can be read by Avrora. The .od extension file can be run using avrora-beta-1.6.0.jar in the following syntax:

```
$  java -jar avrora-beta-1.6.0.jar -platform=mica2 -seconds=600 -colors=false -report-seconds -monitors=energy-log -action=simulate -simulation=sensor-network -nodecount=9,1 SpoofSurge.od SpoofReplay.od
```

Options used with avrora program and their explanations:

-platform=mica2:

Now Avrora only supports mica2 platform.

-seconds=600:

Let all nodes run for 10 minutes.

-colors=false:

Do not display color on terminal.

-report-seconds:

Display event time in second instead of CPU cycle.

-monitors=energy-log:

Monitor power consumption and output log file for each node.

–action=simulate:

Indicate action type.

-simulation=sensor-network:

Simulate multi nodes.

-nodecount=9, 1:

“9, 1” is a node list. That means the first 9 nodes (node 0 to node 8) run the first program, and 1 node (node 9) runs the second program.

SpoofSurge.od SpoofReplay.od

In our example the programs we want to run - 9 nodes run SpoofSurge.od (Normal) and 1 node runs SpoofReplay.od (Attack). SpoofSurge and SpoofReplay are examples of the Spoof Attack.

4.8.6 Step 5 – Collect power consumption data

The final step is to collect the data. The data obtained from a real network is usually light and temperature readings of the sensors.The data from these sensors are

transmitted every 50 Seconds to the base station. On the simulator we get similar readings with LED indicators.

Power consumption contains several aspects: CPU, led, radio, sensor board and flash. Power consumption levels for each node is recorded and calculated in the analysis. Sample output (Avrora with AEON):

```
Energy Consumption Component Breakdown:

Node lifetime: 4423680000 cycles, 600.0 seconds

CPU: 6.519386674160156 Joule
   Active: 0.8983985768261719 Joule, 291792240 cycles
   Idle: 5.620988097333984 Joule, 4131887760 cycles
   RESERVED 1: 0.0 Joule, 0 cycles
   ADC Noise Reduction: 0.0 Joule, 0 cycles
   RESERVED 2: 0.0 Joule, 0 cycles
   Power Down: 0.0 Joule, 0 cycles
   Standby: 0.0 Joule, 0 cycles
   Power Save: 0.0 Joule, 0 cycles
   Extended Standby: 0.0 Joule, 0 cycles
~~~~~~~~~~
Radio: 17.86987400952148 Joule
   Power Off: 0.0 Joule, 3140 cycles
   Crystal: 3.2385253906249998E-6 Joule, 132650 cycles
   Crystal + Bias: 1.7126464843749999E-6 Joule, 3050 cycles
   Receive (Rx): 13.765509851562499 Joule, 3523970522 cycles
   Transmit (Tx):       0: 0.058114546875 Joule, 22670208 cycles
   Transmit (Tx):       15: 4.046244659912109 Joule, 876900430 cycles
~~~~~~~~~~
Sensor Board: 1.26 Joule
   on:  : 1.26 Joule, 4423680000 cycles
~~~~~~~~~~
flash: 0.0036 Joule
   standby: 0.0036 Joule, 4423680000 cycles
   read: 0.0 Joule, 0 cycles
   write: 0.0 Joule, 0 cycles
   load: 0.0 Joule, 0 cycles
```

Power comparisons in the simulator are calculated with two factors.

After an Attack:

- Power consumption 1 = $\frac{\text{After an Attack} - \text{Before an attack}}{\text{Before an Attack}}$ …. (4.1)

After Countermeasures:

- Power consumption 2 = $\frac{\text{After countermeasures} - \text{Before an Attack}}{\text{Before an Attack}}$ …. (4.2)

4.9 Points of Measurement

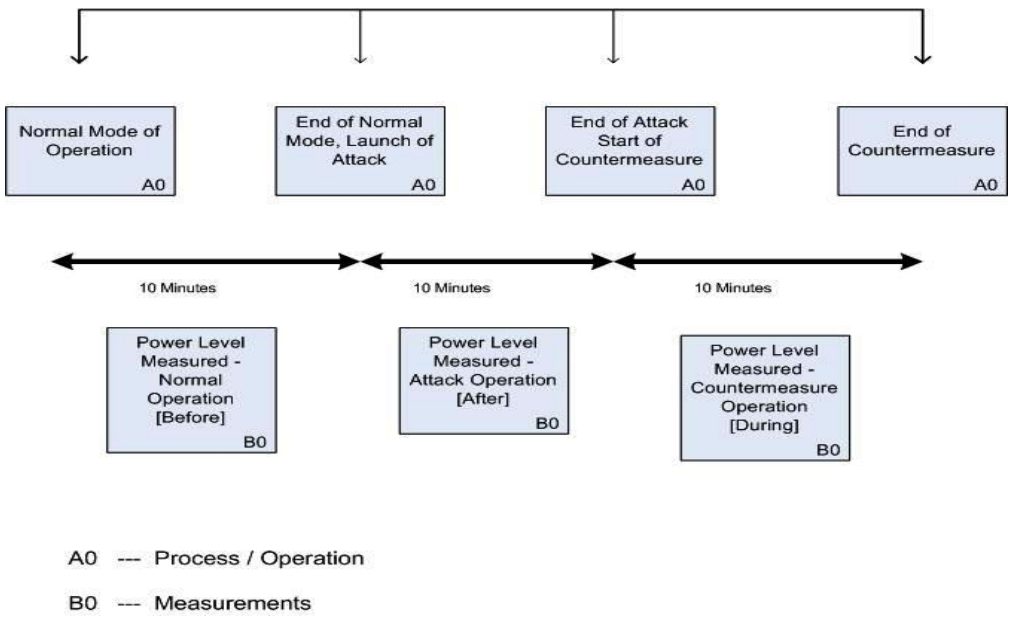


Figure 4.3 Timeline of readings taken

The figure indicates the various points at which we obtain our measurements, with key phase Before, After and During referring to *Before* Attacks, *After* Attacks and *During* Countermeasures. 'After' attack refers to the time period when the node is under attack without any countermeasures.

4.9.1 Types of Simulations

Two sets of simulations were performed. In the first set, 9 nodes were 'good' with 1 malicious node. In the second set, we used a proportion of malicious nodes in the network, which ranged from 10 – 90%. In our second set of simulations we varied the number of malicious nodes. In the graphs below 10% in attacks means that 1 of the

node is malicious from a total of 10 nodes. Therefore 90% in attacks means that 9 of the nodes are malicious and 1 node is good.

In the case of Countermeasures, 90% where nodes that are defending, the ratio was split up to 9 ‘countermeasure’ and 1 malicious node. This was done to get some node to provide attack data on the network.

4.9.2 Normal Operation

“*Surge View”* is the program that executes in the normal operational mode when the node is good; it uses the multi hop network protocol described above. This program is designed and implemented by Crossbow. A node waits for a packet from other nodes on the network and when it receives a packet, it checks the destination address to determine if that packet is addressed to itself. If it is not, then the packet is forwarded to the next node on the network. The next node is determined by the neighbor list for multi-hop routing as described in Section 4.9. The output LED indicates the status of the message. The three states are Green On for Sending a message, Green Off for Send done and Yellow Toggle On and Off for a new message arrival.

High Level Pseudo Code for Surge View (Normal Program):

```
Surge Normal Operation

Initialize: Comm,TimerC,LEdC,RandomLFSR( Randomizer)
Get Communication and timer Parameters

Make a neighbor list from nearby motes

Input: Get message from any mote
Compute: Make copy of the message
Output: Retransmit to the next node on the neighbor list

Show output on Led: Green On -Sending Message; Green off - Send done. ;
                    Yellow Toggle- new message obtained.
Repeat operation.
```

The above Pseudo code describes the normal operation of a Surge program; it consists of input, output and Compute stages. The first step would be to initialize the various service components; Comm for Communication, this module would be to determine which network interface is the standard communication medium. TimerC keeps track of timer functions and provides clock counts, LedC has information about the LEDs used on the device and RandomLFSR provides a random number generation capability. The service modules are ‘wired’ together in a typical program.

4.9.3 Simulation Environment / Details

The simulations using Avrora involved 10 nodes, Avrora has the ability to specify which node runs the ‘Other’ program. The nodes being run on the simulator cannot be placed in specific proximity to the others, i.e. their location cannot be specified. However Avrora assumes a random distance from the base station providing a truly multi-hop capability to the network. Avrora uses a free space radio propagation model, where the default distance is 10 meters between motes and a random distance from the base station. The Free space radio propagation model is used in Avrora to determine the signal strength using the distance between nodes; noise and power transmission strength is also calculated in this way for each node. Although the user can change the distance factor between nodes, the user has no control over the overall area to accommodate the entire node collection.

To simulate the experiment in a real world, would involve deploying 10 motes in a small area and making sure that they are spread out to create a multi-hop network. The programs from these experiments can be directly loaded into the nodes. The data from the base station can be logged in a file and a multi meter can be hooked to each node to gather power consumption readings.

5. RESULTS

5.1 Spoofed attack and countermeasure

Action and result

Attack

- Action: Replay the same data we receive from the other motes and re-transmit it to other neighbors.
- Result: Increases network traffic and power consumption.

Countermeasure

- Action: Keep tracking messages that sent by other motes. Find out if the received message is old or duplicated by comparing the count we maintain in the program.
- Result: Do not forward duplicated messages and discard them.

5.1.1 Network composition

Same as the Network composition in Figure 4.1

5.1.2 SpoofReplay attack details

When SpoofReplay receives data message from a neighbor mote by radio, it will duplicate many copies and send them out to the base station and the other motes in its vicinity. The action is:

Receive message -> Duplicate message -> Send out message

If that malicious mote received a new message, but has not finished sending out copies of the old message, it will do nothing on this new message. Messages are sent out repeatedly until the entire network is flooded. The received message is, AM (active message). Below is the data packet format:

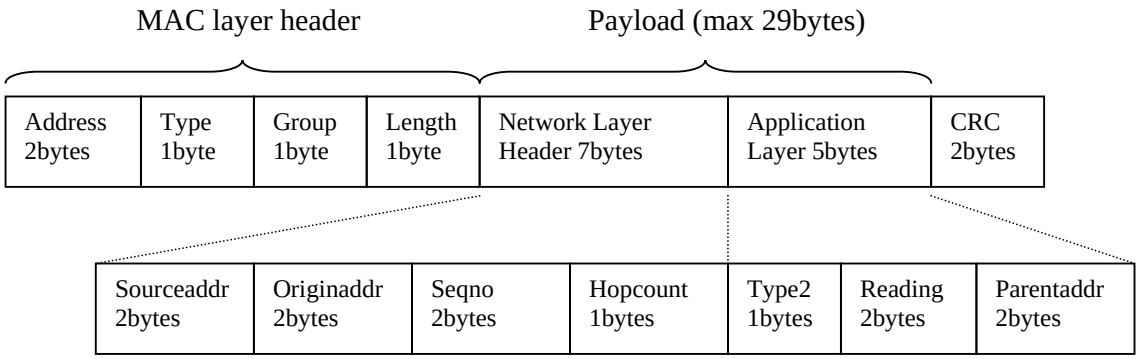


Figure 5.1: Active message packet format of SpoofSurge

High Level Pseudo Code for Spoof Attack:

```
SpoofReplay Attack

Initialize: Comm,TimerC,LEdC,RandomLFSR( Randomizer)
Get Communication and timer Parameters

Make a neighbor list from nearby motes

Input: Get message from any mote
Compute: Make duplicates of the message
Output: Retransmit to random motes for 100 times - Once every second.

Show output on Led: Green On -Sending Message; Green off - Send done. ;
Yellow Toggle- new message obtained.
Repeat operation.
```

Attack Countermeasure details

SpoofSurge is an application that uses multi-hop routing. Each Surge node takes light readings and forwards them to a base station. The node can also respond to broadcast commands from the base. To avoid replayed messages attack, a counting and tracking system is implemented.

High Level Pseudo Code for Spoof Attack Countermeasure:

```
SpoofAttack CounterMeasure

Initialize: Comm,TimerC,LEdC
Get Communication and timer Parameters

Input: when a message is received from a mote
Compute: Check for last sequence Number associated with the packet
        If sequence number is not bigger than the last one received - do not forward message
        If sequence number is bigger - forward to the next node on Neighbor list.

Show output on Led's:
   Red toggle:    send message error;
   Green on:      sending message;
   Green off:     send done;
Repeat operation.
```

Replayed message tracking

Suppose node A is a malicious node. When A receives a message from another node, it duplicates that message and sends as many copies to its parent node B.

Because B holds information of A, it will find out if this message is duplicated according to the last received message number (last_seqno in struct TOS_MhopNeighbor).

```
/* Fields of neighbor table */
typedef struct TOS_MHopNeighbor {
  uint16_t addr;                    // state provided by nbr
  uint16_t recv_count;              // since last goodness update
  uint16_t fail_count;              // since last goodness, adjusted by TOs
  int16_t last_seqno;
  uint8_t goodness;
  uint8_t hopcount;
  uint8_t timeouts;                 // since last recv
} TOS_MHopNeighbor;
```

```
typedef struct MultihopMsg {
  uint16_t sourceaddr;
  uint16_t originaddr;
  int16_t seqno;
  uint8_t hopcount;
  uint8_t data[(TOSH_DATA_LENGTH - 7)];
} __attribute__ ((packed)) TOS_MHopMsg;
```

If seqno in received message is not bigger than last_seqno, then node B believes the message is duplicated and will not pass the message to the next node.

5.1.4 Simulation/Power Analysis - Spoofed attack and countermeasure

5.1.4.1 Before attack

First we just let 9 nodes run *SpoofSurgeNoCounter.od* (Normal program) and do not inject any malicious nodes. Following is the value of power consumption on node 0, 4, 8. (Run time 10 minutes) shown below in Joules. These are default readings for all normal operation on the network.

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.432	0.013	17.283	1.26	0.004	24.992
4	6.416	0.291	17.315	1.26	0.004	25.286
8	6.416	0.286	17.315	1.26	0.004	25.281

Table 5.1: Typical Power consumption with Surge – Normal Operation

Note: Radio power consumption is normally higher than the CPU because of the main communication medium being wireless.

5.1.4.2 After attack

We next inject the malicious node (node 9), which is running the *SpoofReplay* (Attack) program. The power consumed after the attack is over is shown below.

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.482	0.019	17.283	1.26	0.004	25.048
4	6.450	0.324	17.322	1.26	0.004	25.36
8	6.452	0.344	17.331	1.26	0.004	25.391
9	6.514	4.490	17.812	1.26	0.004	30.08

Table 5.1.1: Spoof Attack Power consumption (10 Minutes | Unit in Joules)

5.1.4.3 During counter

Change SpoofSurgeNoCounter.od (Normal) to SpoofSurge.od (Countermeasure), and run again.

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.462	0.016	17.283	1.26	0.004	25.025
4	6.445	0.345	17.316	1.26	0.004	25.37
8	6.444	0.337	17.315	1.26	0.004	25.36
9	6.51	4.408	17.813	1.26	0.004	29.995

Table 5.1.2: Spoof Countermeasure Power consumption (10 Minutes | Unit in Joules)

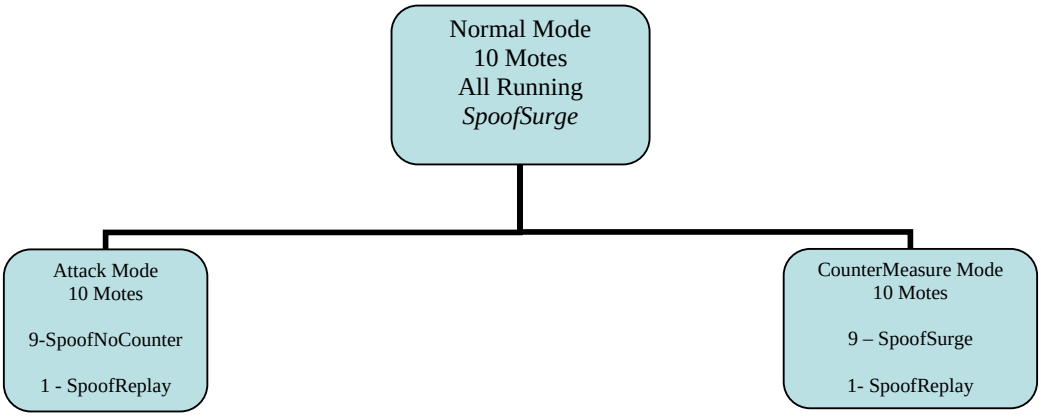


Figure 5.3: Different Setups for Data Measurement. SpoofSurge.od – Normal,

SpoofReplay.od - Attack, SpoofNoCounter.od – No Countermeasure

5.1.4.4 Radio power consumption comparison

Here we can compare radio power consumption before attack, after attack and during countermeasure: (unit is in Joule).

Node	Before	After	During
0	17.28294837495117	17.28294837495117	17.28294837495117
1	17.31497163474121	17.328573483569336	17.32782377536621
2	17.31497163474121	17.33232202458496	17.326003055444335
3	17.31497163474121	17.33939070192871	17.316042646459955
4	17.31497163474121	17.322147413964842	17.31572134223633
5	17.31497163474121	17.32525552224121	17.31497163474121
6	17.31497163474121	17.328573483569336	17.317541992773435
7	17.31668525349121	17.32664566247558	17.320540895678707
8	17.314973809741208	17.331251082958982	17.31497163474121
9		17.811831745922852 (do not include)	17.81333011376953 (do not include)
Total	155.804437246630848	155.917107750244126	155.836565352392562

Table 5.1.3: Spoof Attack Average Radio Power consumption

From section 4.6.8

Power increase rate 1 = (After – Before)/before = 0.000723153368443069894

Power increase rate 2 = (During – Before)/before = 2.06207899656007683e-4

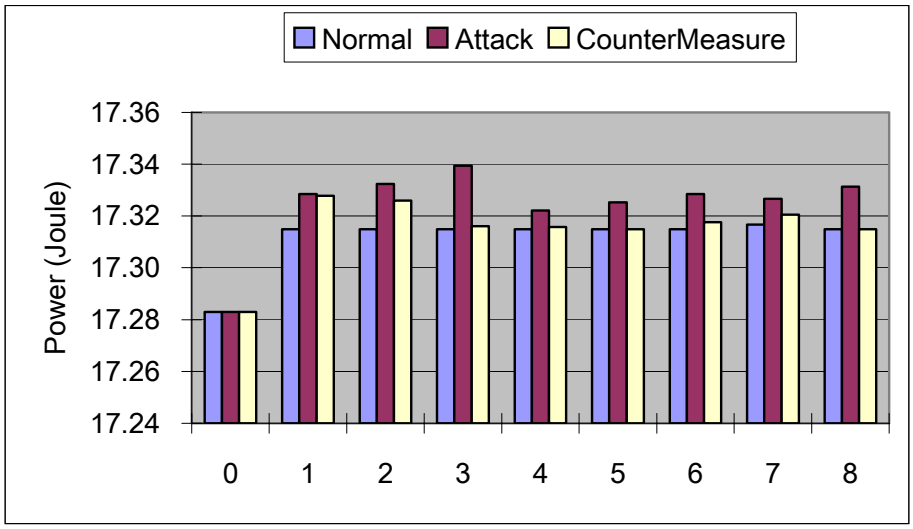


Figure 5.4: Spoof Radio Power Consumption

Spikes are noticed in the Attack data indicated as 'Attack' in the chart because of the increased radio operation to transmit/receive data between the motes during spoof replay. Node 0 with no power level change is the Base Station, which is not affected by spoof attacks. In the graph in fig 5.4 9 nodes are in normal mode with one attacking node. . The attack node is assumed to be the 10^{th} node in the network.

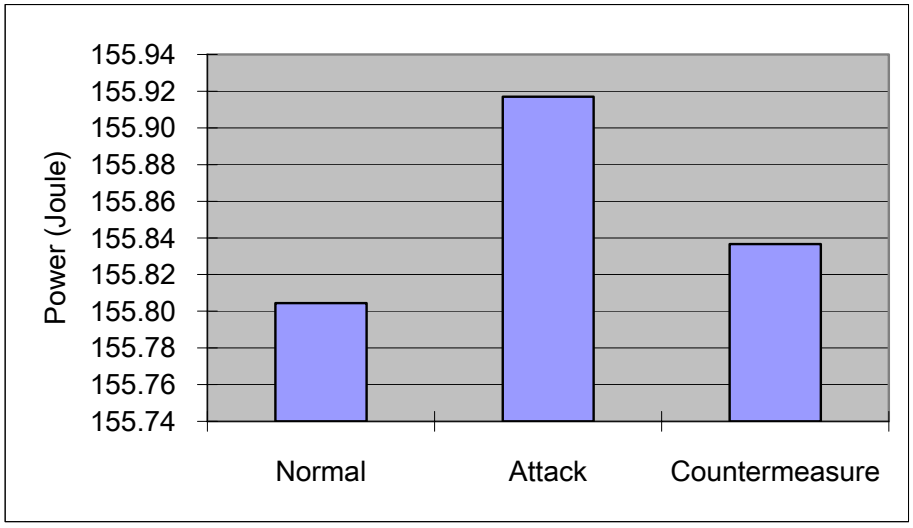


Figure 5.5: Spoof Radio Power Consumption – Overall

The data shows countermeasures consume only marginally more power than during attacks.

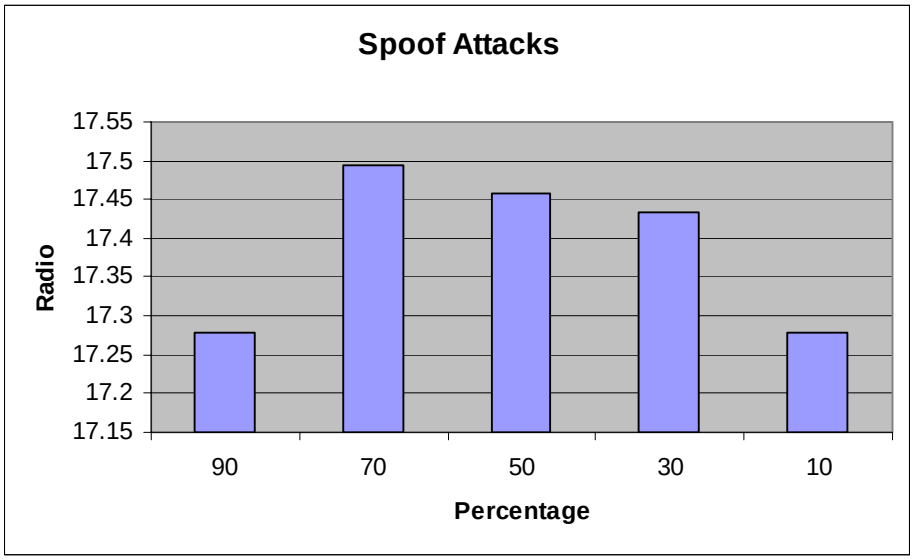


Figure 5.6: Spoof Attack Radio Power Consumption

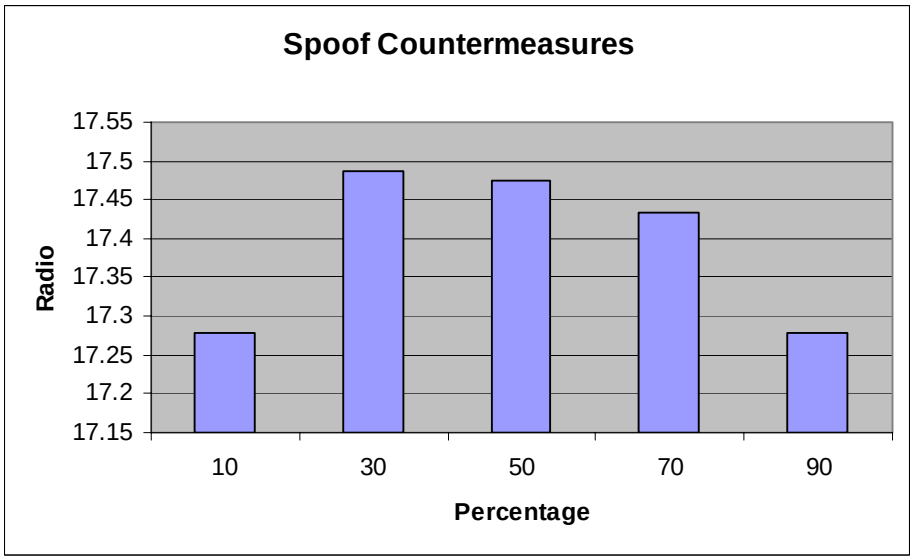


Figure 5.7: Spoof Countermeasure Radio Power Consumption

In the graph in fig 5.6, spoof attack shows 90% (9 bad nodes) of the nodes in the network providing attack data and 10% running the normal version of *Surge* at the first instance. Similarly in fig 5.7 we have 10% (1 counter node) providing

countermeasure with the remainder configured to provide attack data. The maximum impact is noticed at 70:30(Attack: Counter) configurations which are 7 nodes providing attack data and 3 nodes defending by providing countermeasure, this could be because of a large number of normal modes being compromised and influenced by the spoof attack.

Countermeasures are effective as their numbers increase from 70:30 (7 counter and 3 bad nodes) to 10:90 (1 counter and 9 bad node) where power consumption level can be seen going down.

5.1.4.5 CPU power consumption comparison

Node	Before	After	During
0	6.433002153463135	6.481989058326172	6.4620128746928716
1	6.416962906932617	6.451140489597656	6.447122198427491
2	6.416563857842529	6.452043172607422	6.446680593976806
3	6.416787227515625	6.453460800742431	6.44511507122754
4	6.416692427858399	6.449759634051026	6.444794257116699
5	6.416900482207275	6.450354235359863	6.444584724852783
6	6.4168850328396	6.45118967839087	6.445155471581788
7	6.420200987850098	6.450451128157226	6.4454439623159185
8	6.416699559657715	6.451737467600097	6.444475981239014
9		6.513512369165772 (do not include)	6.510362461346924 (do not include)
Total	57.770694636166993	58.09212566483276	58.02538513543091

Table 5.1.4: Spoof Attack Average CPU Power consumption

From section 4.6.8

Power increase rate 1 = (After – Before)/before = 0.005563911437972221541

Power increase rate 2 = (During – Before)/before = 0.00440864526327628156

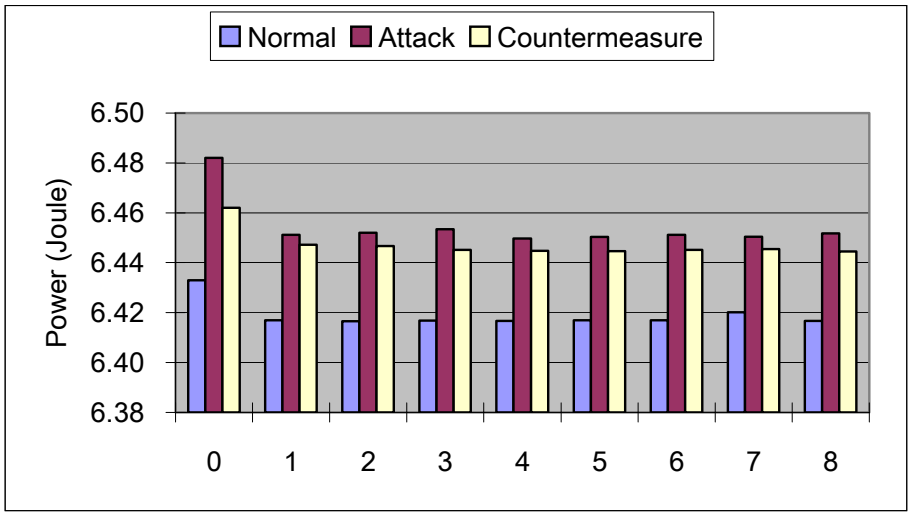


Figure 5.8: Spoof CPU Power Consumption

Node 0 is the base station; increased power consumption is a result of it receiving most of the messages from the malicious node. Most replayed messages are sent to the base station. Notice the increase in power surge after attack. In countermeasure the duplicate messages are eliminated and an attack is thwarted thus attributing to a small reduction in power consumption.

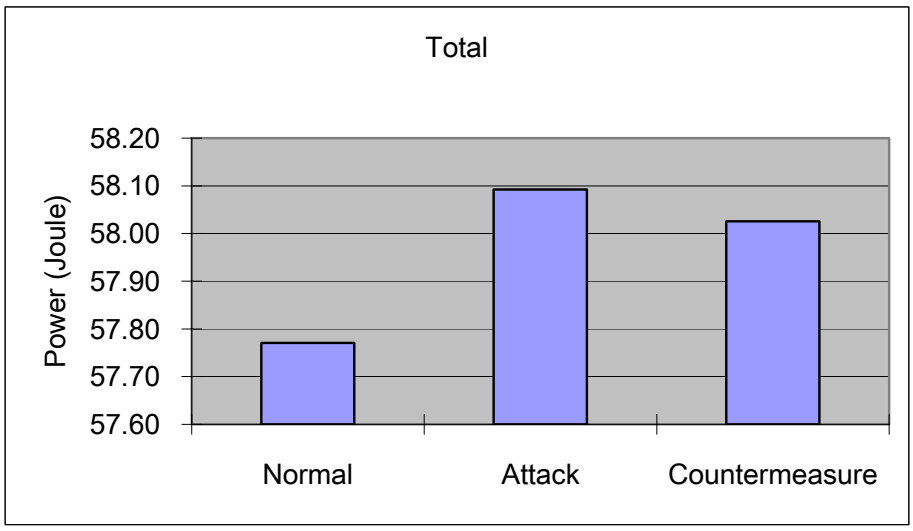


Figure 5.9: Spoof CPU Power Consumption- Overall

The low power consumption shows that these Countermeasures are effective in stopping attacks occurring on the network.

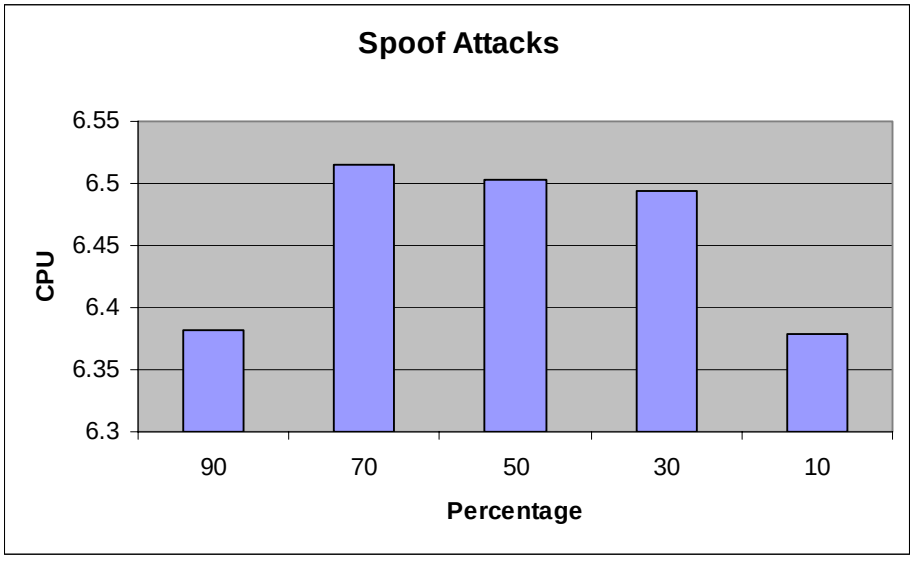


Figure 5.10: Spoof Attack CPU Power Consumption

Similar to Radio power consumption levels in Fig 5.6/5.7 CPU power consumption levels reflect the same behavior with more countermeasure nodes in the network resulting in lower levels of power consumption. This could mean effective countermeasures being utilized.

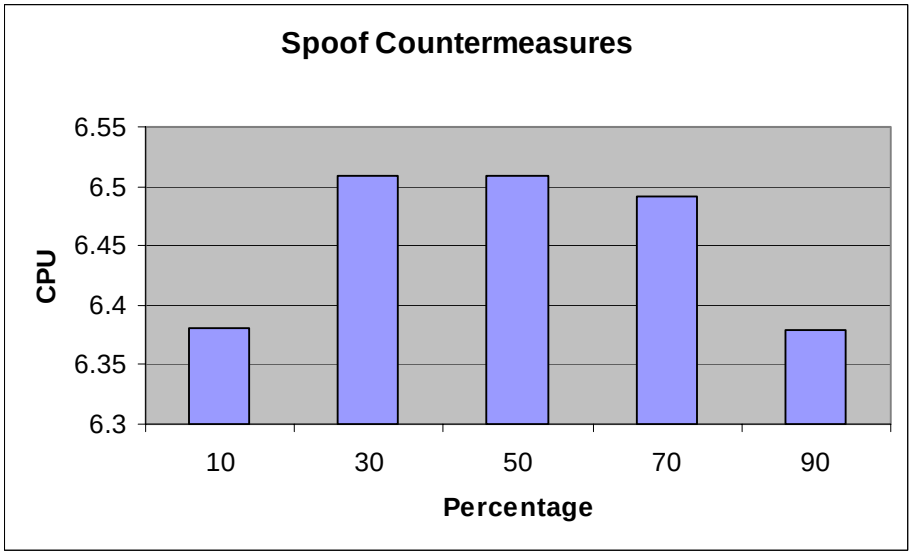


Figure 5.11: Spoof Countermeasure CPU Power Consumption

5.1.4.6 Conclusion

In a spoofed attack, when the malicious node was injected, the power consumption of the entire sensor network increased. During countermeasure, the whole power consumption for the network decreased, though it was higher than before the attack.

Attacks consumed 0.7% more power when compared to normal operation modes, and the power consumed by countermeasures was 0.2% more than normal values. The difference between the two shows the impact of the two operations, clearly showing that these countermeasures are more effective.

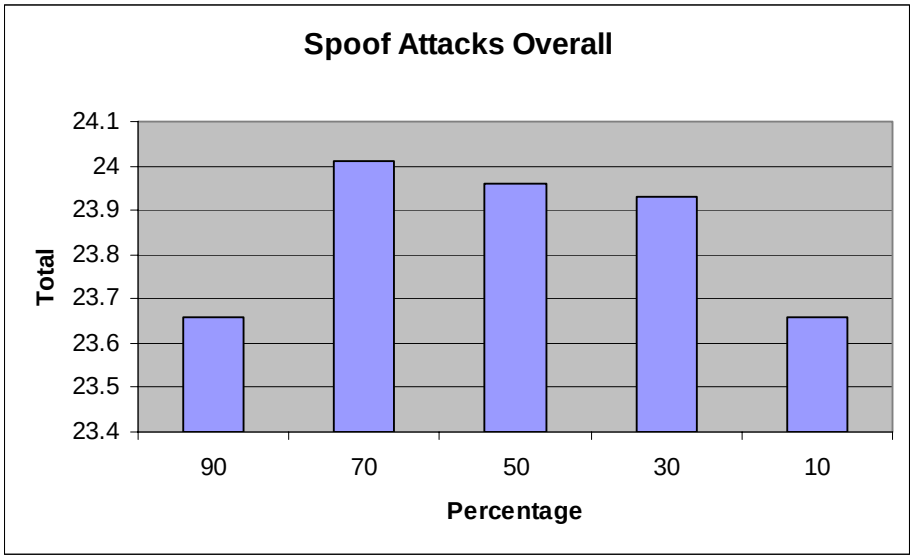


Figure 5.12: Spoof Attack Overall Power Consumption

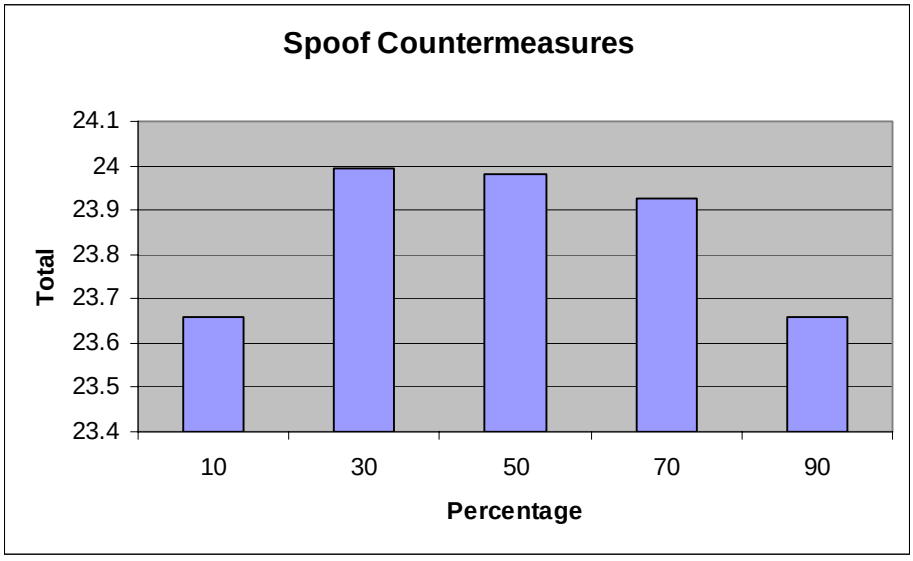


Figure 5.13: Spoof Countermeasure Overall Power Consumption

In the graph in fig 5.12 and 5.13 we have the total power consumption levels of CPU and radio from 10 motes. Maximum power levels are noticed in 70:30 (Attack: Countermeasure) configurations with power levels going down as the proportion of countermeasure nodes are increased from 70:30 (7 Counter nodes and 3 bad nodes) to 10:90 (1 counter node and 9 bad nodes) . Hence countermeasures are effective against the spoof attack.

5.2 Selective forwarding attack and countermeasure

5.2.1 Action and result

Attack

- Action: The malicious mote will only forward received messages from whose address is an even number. For example, messages from motes whose addresses are 2, 4, 6, 8...will be forwarded, and messages from 1, 3, 5, and 7...will be dropped.
- Result: Data loss, longer path to reach destination.

High Level Pseudo Code for Selective Forward Attack:

```
Sforward Attack

Initialize: Comm,TimerC,LEdC
Get Communication and timer Parameters

Make a neighbor list from nearby motes

Input: Ready to receive message from other motes
Compute: Drop messages from motes 1,3,5,7 …….;
Output: Retransmit only messages from motes 2, 4, 6, 8…….

Show output on Led: Green On -Sending Message; Green off - Send done.
Repeat operation.
```

Countermeasure

- Action: Two-path routing.
- Result: Each node will send same message twice.

High Level Pseudo Code for Selective Forward Attack:

```
Sforward Countermeasure

Initialize: Comm,TimerC,LEdC
Get Communication and timer Parameters

Input: when a message is received from any mote
Compute: Choose two parents from the neighbor list
Output: Retransmit to both motes on the network.

Show output on Led:
   Green on:      sending message to main parent;
   Green off:     send done;
   Yellow on:     sending message to another parent;
   Yellow off:    send done;

Repeat operation.
```

5.2.2 Network composition

Same as the Network composition in Figure 4.2

5.2.3 Selective forwarding attack details

This attack application is modified from multi-hop router program [13].

In the RouterM.nc program, when a message is received, if the destination address in message is equal or part of the local address, then it must be forwarded. In the attack mode, only messages from even numbered addresses are forwarded.

```
if (pMsg->addr == TOS_LOCAL_ADDRESS) { // Addressed to local node
  if ((signal Intercept.intercept[id](pMsg,&pMHMsg->data[0],PayloadLen)) == SUCCESS) {
    if (pMHMsg->originaddr % 2 == 0)
      pMsg = mForward(pMsg,id);
    else
      dbg(DBG_ROUTE, "MHop: message from 0x%x dropped\n", pMHMsg->originaddr);
  }
}
```

5.2.4 Countermeasure on selective forwarding

To avoid data loss caused by selective forwarding, when sending a message, we can choose two different paths. In section **Error! Reference source not found.**, we discussed the parent choosing mechanism for routing. For the countermeasure we modify the routing approach by having two different parent nodes. If one route is compromised, the alternative will deliver the message to the destination.

.

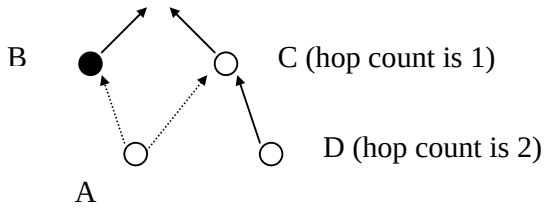


Figure 5.14: Choose another parent – Selective Forwarding.

In figure 5, suppose node “A” wants to send a message to the base station and node B, C, and D are neighbors of A. A has a parent B. We must choose another good parent in C and D. We can take this topology as a tree with the base station as root. The depth (hop count) of C is 1 and D is 2, we think C is better than D because C is nearer to the root than D. Hence we choose node C as A’s other parent and send a message copy to it. Since we transmit the message in two different paths we achieve redundancy.

5.2.5 Simulation / Power Analysis - Selective forward attack

5.2.5.1 Before attack

Let 10 nodes run “Surge.od” for 10 minutes this time. Following is the data on nodes 0, 5, 9. (Unit is Joule)

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.439	0.000	17.283	1.26	0.004	24.986
5	6.419	0.000	17.322	1.26	0.004	25.005
9	6.418	0.000	17.322	1.26	0.004	25.004

Table 5.2: Typical Power consumption with Surge – Normal Operation

5.2.5.2 After attack

A normal node becomes a malicious node (node 9, run *SForwardAttack.od*), and run again. (Run 10 minutes, unit is)

Node	CPU	Led	Radio	Sensor	Flash	Total

				Board		
0	6.437	0.000	17.283	1.26	0.004	24.984
5	6.420	0.000	17.324	1.26	0.004	25.008
9	6.418	0.328	17.315	1.26	0.004	25.325

Table 5.2.1: Selective forward attack Power consumption

5.2.5.3 During counter

Change *Surge* (Normal) to *SForwardCounter.od*, and run again. (Run 10 minutes, unit is Joule)

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.475	0.000	17.283	1.26	0.004	25.022
5	6.440	0.000	17.358	1.26	0.004	25.062
9	6.431	0.408	17.315	1.26	0.004	25.418

Table 5.2.2: Selective forward countermeasure Power consumption

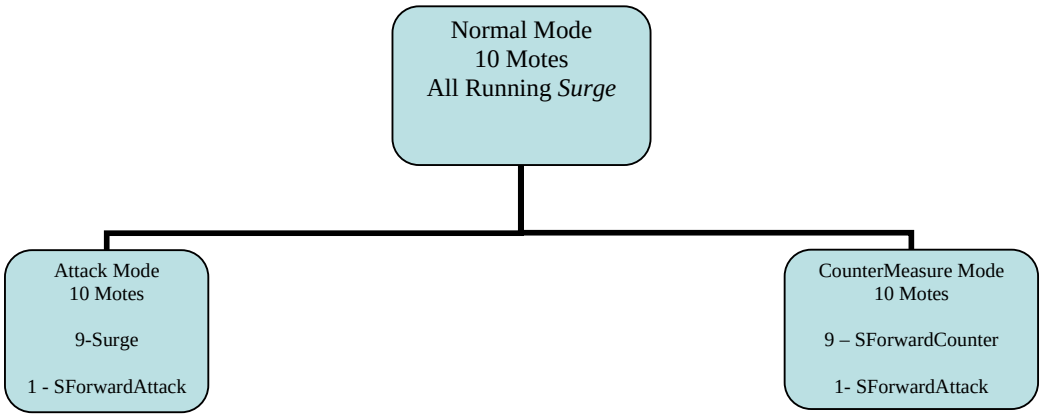


Figure 5.15: Program Execution sequence. *Surge.od* – Normal, *SForwardAttack.od* - Attack, *SForwardCounter.od* – Countermeasures

5.2.5.4 Only radio power consumption comparison

Now we just compare radio power consumption before attack, after attack and during counter: (unit is Joule)

Node	Before	After	During
0	17.28294848432617	17.28294848432617	17.282948317333982
1	17.314971744116207	17.314978269824216	17.346994830200195
2	17.314971744116207	17.314971744116207	17.346994830200195

3	17.314971744116207	17.314971744824216	17.346999180200193
4	17.314971744116207	17.31497826911621	17.375057510815427
5	17.322473176147458	17.32375404020996	17.35727654270019
6	17.314971744116207	17.314971744116207	17.346999180200193
7	17.314971744116207	17.318082028100584	17.34870844895019
8	17.31497826911621	17.315721452319334	17.346994830200195
9	17.31497826911621	17.31497163474121	17.31497163474121
Total	173.12520866340329	173.130349411694314	173.41394530554197

Table 5.2.3: Selective forward Average Radio Power consumption

From section 4.6.8

Power increase rate 1 = (After – Before)/before = 2.96938171552985167 04e-5

Power increase rate 2 = (During – Before)/before = 0.00166779086863111263

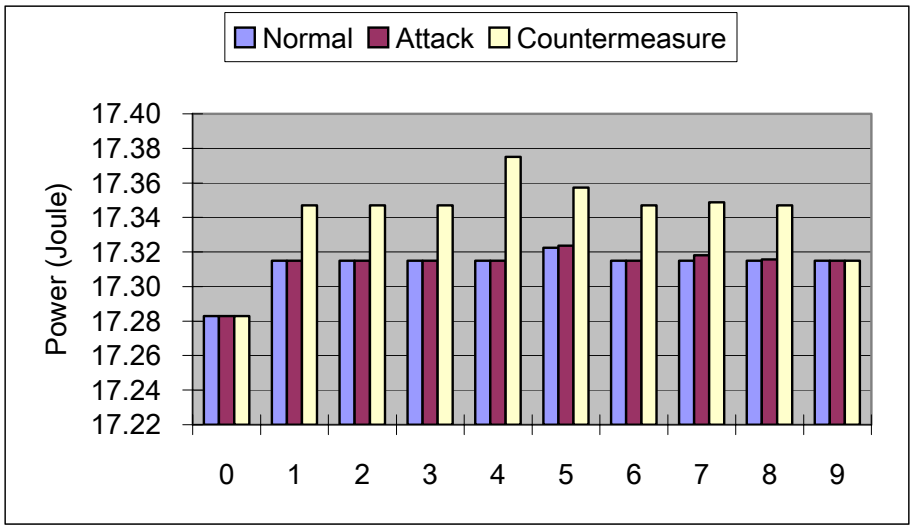


Figure 5.16: Selective forward Radio Power Consumption

The Base station (node 0) does not show any increase in power consumption levels. The base station receives messages only from the border motes, and therefore it does not have a role in the selective forwarding attack or the countermeasure. Increase in countermeasure power consumption is noticed in all the nodes because most of them retransmit the message they receive in two paths to achieve redundancy from selective forwarding attacks. Chart below indicates the same.

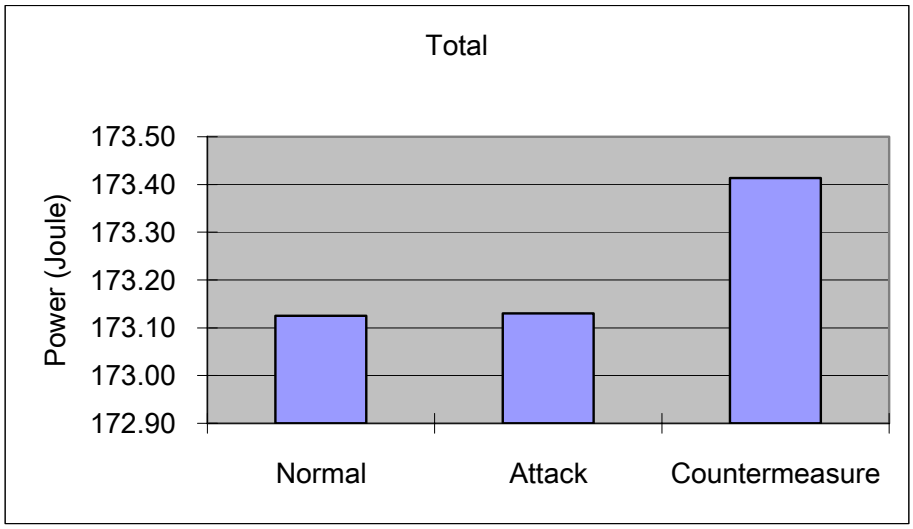


Figure 5.17: Selective forward Radio Power Consumption - Overall

Countermeasure results show more power consumption due to redundant messages being generated in different paths simultaneously resulting in increased traffic.

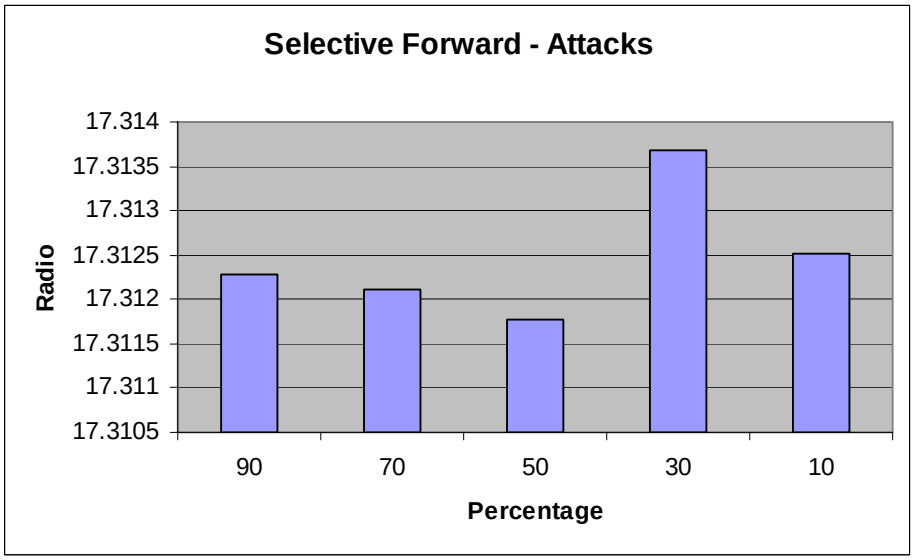


Figure 5.18: Selective forward Attack Radio Power Consumption

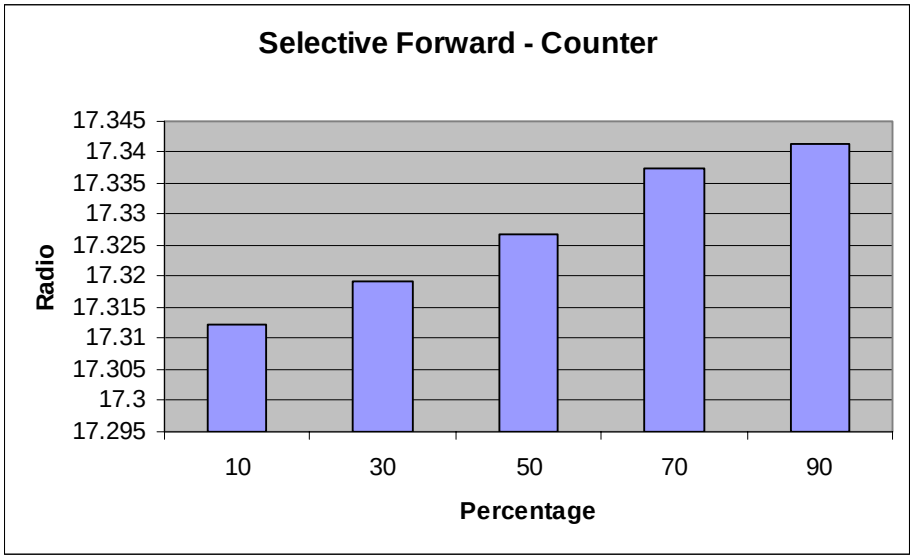


Figure 5.19: Selective forward Countermeasure Radio Power Consumption

5.2.5.5 Only CPU power consumption comparison

Node	Before	After	During
0	6.436892733870361	6.437137944034423	6.4745700777771
1	6.417669621845215	6.417784338985352	6.4373625115061035
2	6.417714777305175	6.417729432723145	6.437382373996827
3	6.4179184683676755	6.417910607921874	6.4374748089899905
4	6.417709932837158	6.417771974335694	6.442518032521241
5	6.419206613960204	6.419599966203369	6.439364050125
6	6.417693816689209	6.417890334708252	6.437257338947509
7	6.41787958201709	6.418542258498535	6.437660701487305
8	6.417750651114991	6.417954426384277	6.437547921103272
9	6.41788059078003	6.417910599329345	6.430664973464844
Total	64.1983167887871085	64.200231883124266	64.411802789919192

Table 5.2.4: Selective forward Attack Average CUP Power consumption

From section 4.6.8

Power increase rate 1 = (After – Before)/before = 2.9830911976371174568e-5

Power increase rate 2 = (During – Before)/before = 0.00332541430695844987

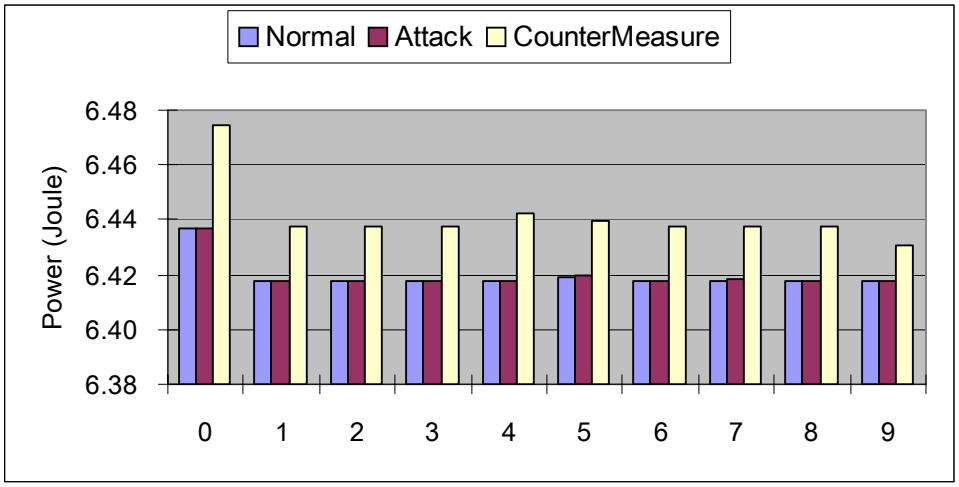


Figure 5.20: Selective forward CPU Power Consumption

Increased countermeasure is a result of transmitting packets in two paths as opposed to sending in one.

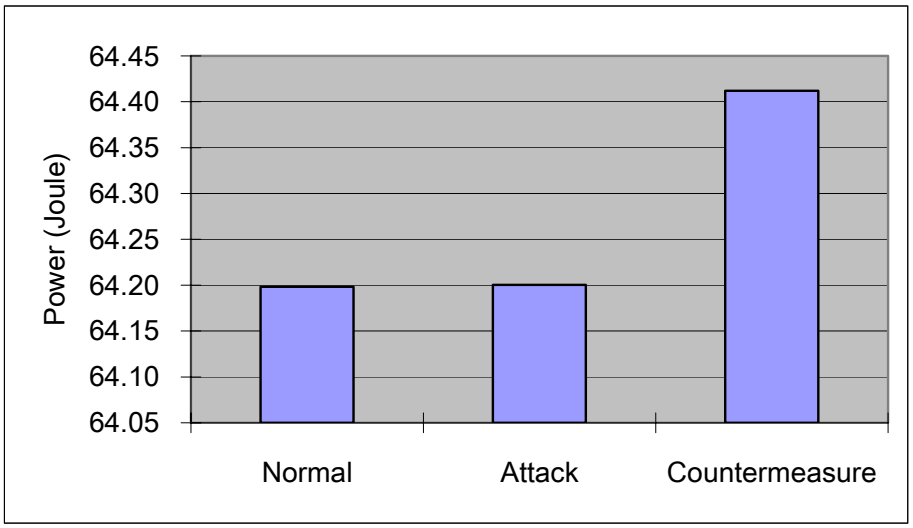


Figure 5.21: Selective forward CPU Power Consumption- Overall

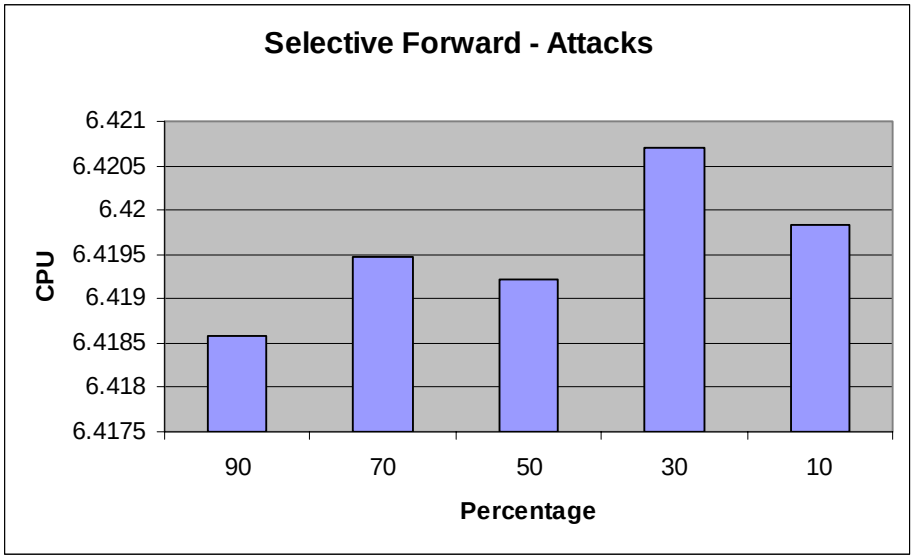


Figure 5.22: Selective forward Attack CPU Power Consumption

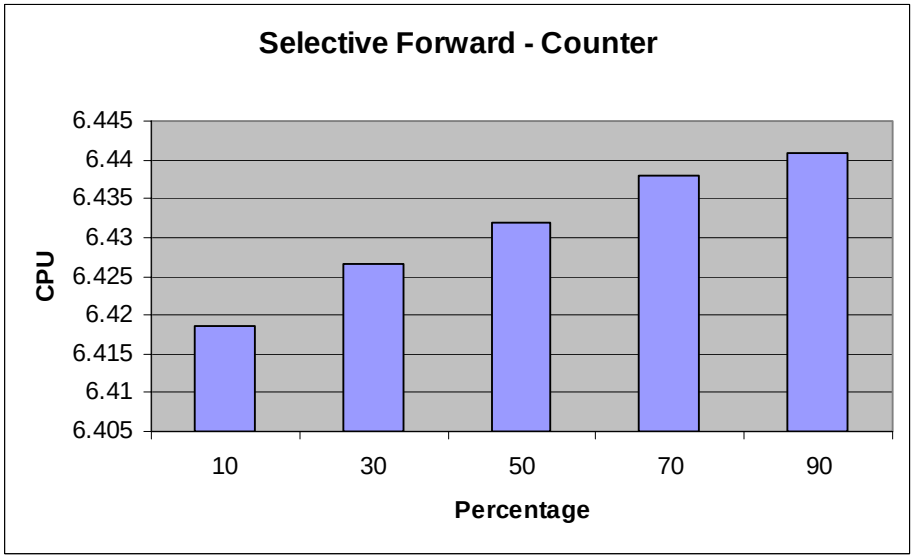


Figure 5.23: Selective forward Countermeasure CPU Power Consumption

5.2.5.6 Conclusion

In the selective forwarding attack, the power consumption loss is small. However the sensors consume more power to counter this attack. This attack results in

loss of data and there is little or no power consumption spike due to lost data. However, power readings are more when countermeasures are being applied to avoid this loss.

This attack did not affect power consumption much; a difference of 0.0002% from normal running mode does not indicate any significant change. However the data loss is damaging for this attack. The countermeasures applied consumed almost 0.3 % more than the normal operating modes; this was significantly higher than the attack power levels. This cost is only because of the attempt to retransmit the same data.

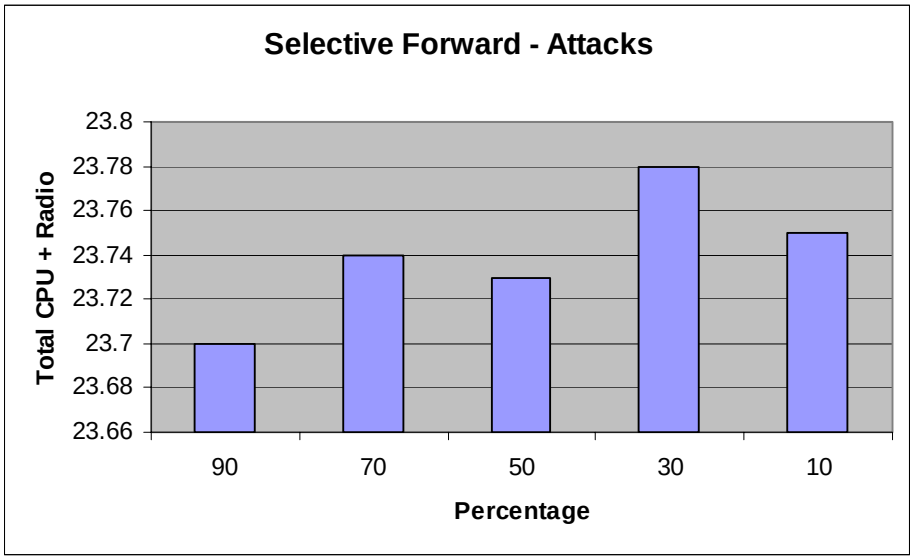


Figure 5.24: Selective forward Attacks Power Consumption – Overall

In the graph in fig.5.24, the maximum power consumption level is noticed at 30% level with 3 of the nodes providing the attack and 70% or 7 nodes consisting of normal motes.

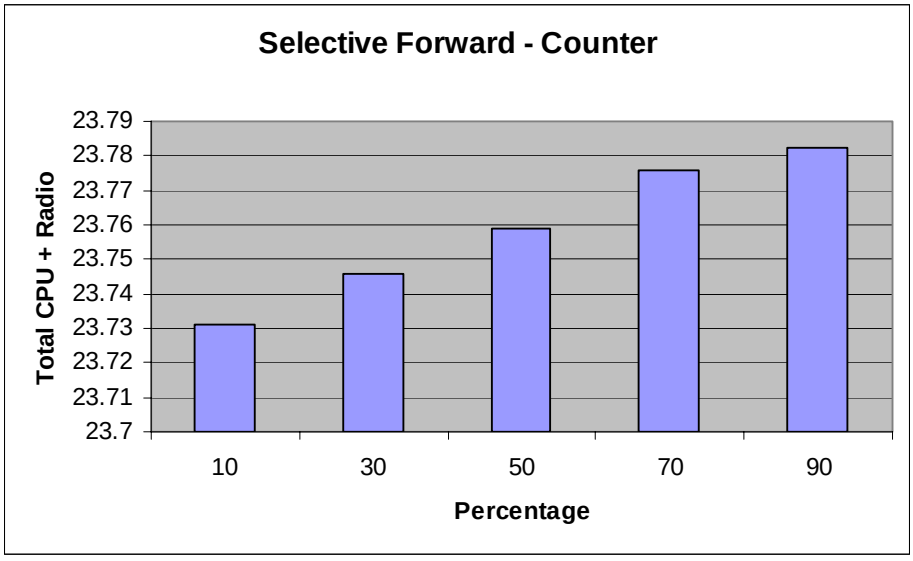


Figure 5.25: Selective forward Countermeasure Power Consumption - Overall

In the above graph fig 5.25 the proportion of nodes providing countermeasures at 90% (9 Counter Nodes) consumes more power than a smaller number. This is due to the fact that the data is actually dropped and not forwarded in the 10% (1 Counter node and 9 Attack nodes) configuration, with the defending nodes having to defend the network and actually relay back messages on two routes. In this type of attack, countermeasures are not very effective and cause more damage when they are deployed in large numbers.

5.3 Sinkhole attack and countermeasure

5.3.1 Action and result

Attack

Sinkhole Attacks aim at causing data loss by dropping packets and not forwarding any message. The also deceive motes in their proximity into thinking that the base station is nearer than the normal path.

High Level Pseudo Code for Sinkhole Attack:

```
Sinkhole Attack

 Initialize: Comm,TimerC,LEdC
 Get Communication and timer Parameters

 Make a neighbor list from nearby motes

 Input: Ready to receive message from other motes
 Compute: Drop all packets
Output: Increase Message send success rate to base station.

 Show output on Led:
   Green on:      sending message;
   Green off:     send done;

 Repeat operation.
```

Countermeasure

The Attacks can be solved by using a two path routing strategy and also transmitting the same message twice.

High Level Pseudo Code for Sinkhole Attack Countermeasure:

```
Sinkhole Countermeasure

 Initialize: Comm,TimerC,LEdC
 Get Communication and timer Parameters

 Input: when a message is received from any mote
 Compute: Choose two parents from the neighbor list
 Output: Retransmit to both motes on the network. Also retransmit same message twice.

 Show output on Led:
   Green on:      sending message to main parent;
   Green off:     send done;
   Yellow on:     sending message to another parent;
   Yellow off:    send done;

Repeat operation.
```

5.3.2 Network composition

Same as the Network composition in Figure 4.2

5.3.3 Sinkhole attack details

Sinkhole attack is similar to selective forwarding attack. When a malicious mote receives a message, it will do nothing and drop packets.

```
    if (pMsg->addr == TOS_LOCAL_ADDRESS) { // Addressed to local node
      if ((signal Intercept.intercept[id](pMsg,&pMHMsg->data[0],PayloadLen)) ==
SUCCESS) {
        // Do nothing
        dbg(DBG_ROUTE, "MHop: message from 0x%x dropped\n", pMHMsg->originaddr);
      }
    `
```

5.3.4 Countermeasure

Same as countermeasure on selective forwarding as outlined in section 5.2.4. Redundancy is achieved by retransmission of data and taking a two path route to reach the destination. The overheads achieved, would be similar to that of Selective forwarding.

5.3.5 Simulation / Power Analysis - Sinkhole attack

5.3.5.1 Before attack

Let 10 nodes run "*Surge.od*" (Normal) for 10 minutes. The same setup as in Selective forwarding attacks as in 5.2.5.

5.3.5.2 After attack

SinkholeAttack.od – program to simulate an Attack.
A normal node is changed to a malicious node (node 9, run SinkholeAttack.od), and run the same setup again. (Run 10 minutes, unit is Joule). We simulate an attack by running the *SinkholeAttack* program simultaneously in 9 motes. The simulations show dropped packets and increase in transmitting traffic from some motes.

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.438	0.000	17.283	1.26	0.004	24.985
5	6.418	0.000	17.315	1.26	0.004	24.997
9	6.418	0.174	17.316	1.26	0.004	25.172

Table 5.3: Sinkhole Attack Power consumption

5.3.5.3 During counter

SinkholeCounter.od – program to simulate a Countermeasure.
Change Surge.od to SinkholeCounter.od, and run again. (Run 10 minutes, unit is Joule)

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.478	0.012	17.283	1.26	0.004	25.037
5	6.436	0.344	17.348	1.26	0.004	25.392
9	6.429	0.196	17.316	1.26	0.004	25.205

Table 5.3.1: Sinkhole countermeasure Power consumption

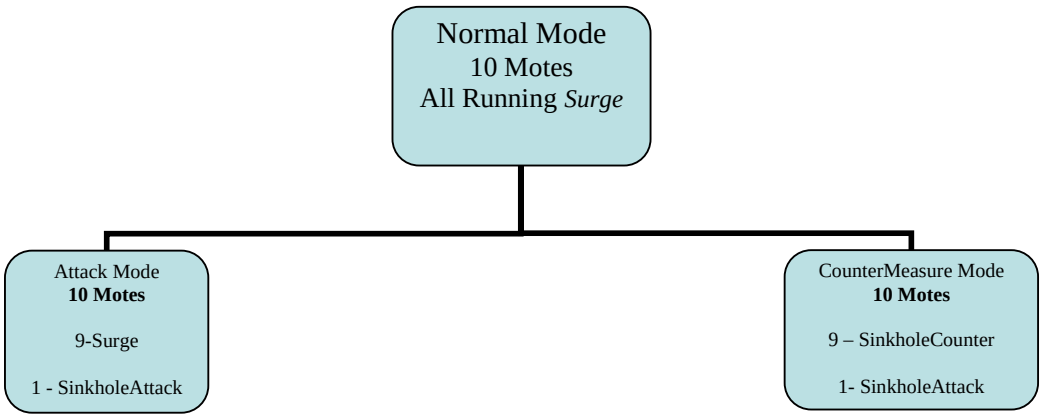


Figure 5.26: Program Execution sequence. *Surge* – Normal, *SinkholeAttack* - Attack, *SinkholeCounter* – Countermeasures

5.3.5.4 Only radio power consumption comparison

Node	Before	After	During
0	17.28294848432617	17.28294848432617	17.282948317333982
1	17.314971744116207	17.31497826911621	17.346994830200195
2	17.314971744116207	17.314971744116207	17.346994830200195
3	17.314971744116207	17.314971744116207	17.346997005200194
4	17.314971744116207	17.314971744116207	17.353635102856444
5	17.322473176147458	17.314978270532226	17.34763743723144
6	17.314971744116207	17.31668536286621	17.347423234887696
7	17.314971744116207	17.314971744116207	17.346994830200195
8	17.31497826911621	17.314971744116207	17.346994830200195
9	17.31497826911621	17.31582844411621	17.31582844411621
Total	173.12520866340329	173.120277551538061	173.382448862426746

Table 5.3.2: Sinkhole Radio Power consumption

From section 4.6.8

Power increase rate 1 = (After – Before)/before = -2.848292229247941548e-5

Power increase rate 2 = (During – Before)/before = 0.00148586217460446398

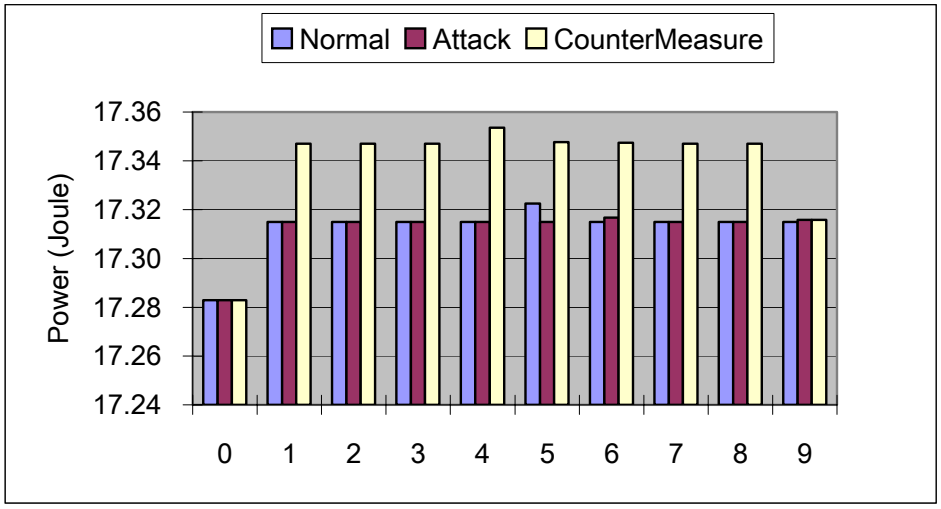


Figure 5.27: Sinkhole Radio Power Consumption

Since packets are dropped no effect is seen due to the attacks power consumption readings; however, a huge spike is noticed in the countermeasure because of retransmission and the two path strategy being followed.

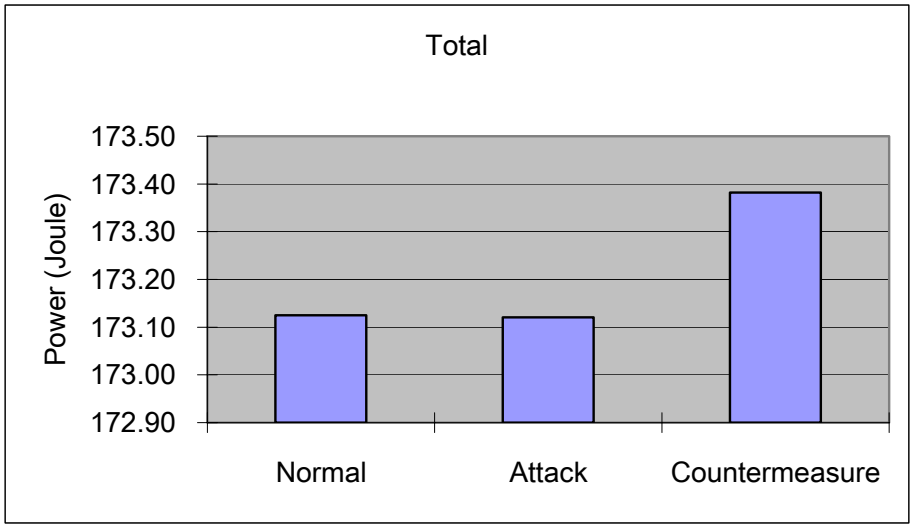


Figure 5.28: Sinkhole Radio Power Consumption – Overall

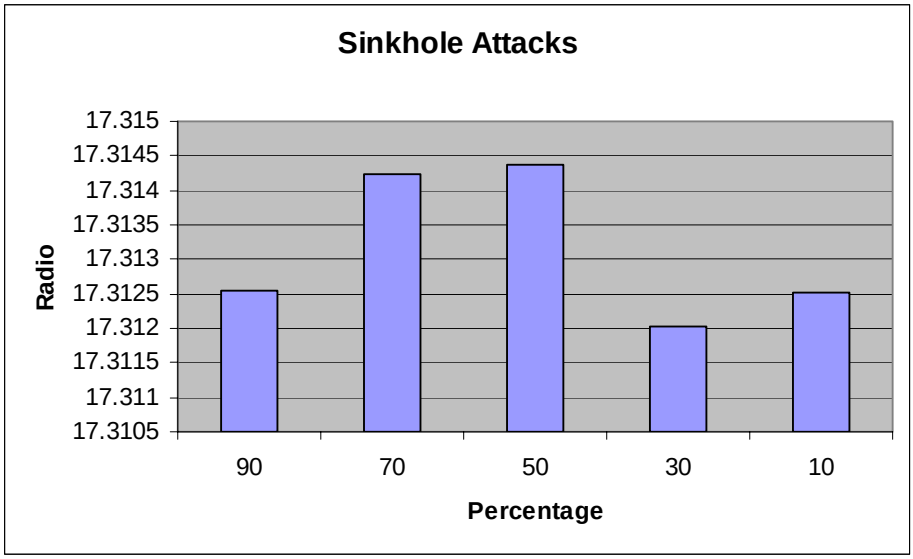


Figure 5.29: Sinkhole Attack Radio Power Consumption

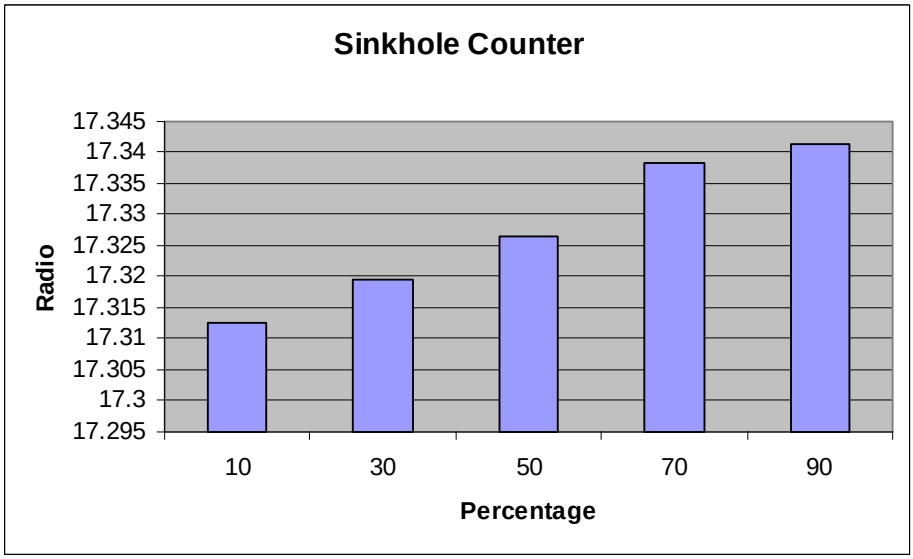


Figure 5.30: Sinkhole Attack Countermeasure Power Consumption

5.3.5.5 Only CPU power consumption comparison

Node	Before	After	During
0	6.436892733870361	6.438104976496094	6.476776941757569
1	6.417669621845215	6.41796414969043	6.436301674402099
2	6.417714777305175	6.417907878934571	6.436459079227783
3	6.4179184683676755	6.418143987891601	6.436361210319092
4	6.417709932837158	6.417974214979249	6.437570359634277
5	6.419206613960204	6.418074903956055	6.43648570060205

6	6.417693816689209	6.4183431231950685	6.436419373149903
7	6.41787958201709	6.418045924791747	6.436295817734131
8	6.417750651114991	6.418063756008545	6.436592150010498
9	6.41788059078003	6.417884335404297	6.429456523581543
Total	64.19831679	64.20050725	64.39871883

Table 5.3.3: Sinkhole CPU Power consumption

From section 4.6.8

Power increase rate 1 = (After – Before)/before = 3.41202092130428268 81e-5

Power increase rate 2 = (During – Before)/before = 0.00312160894584725451

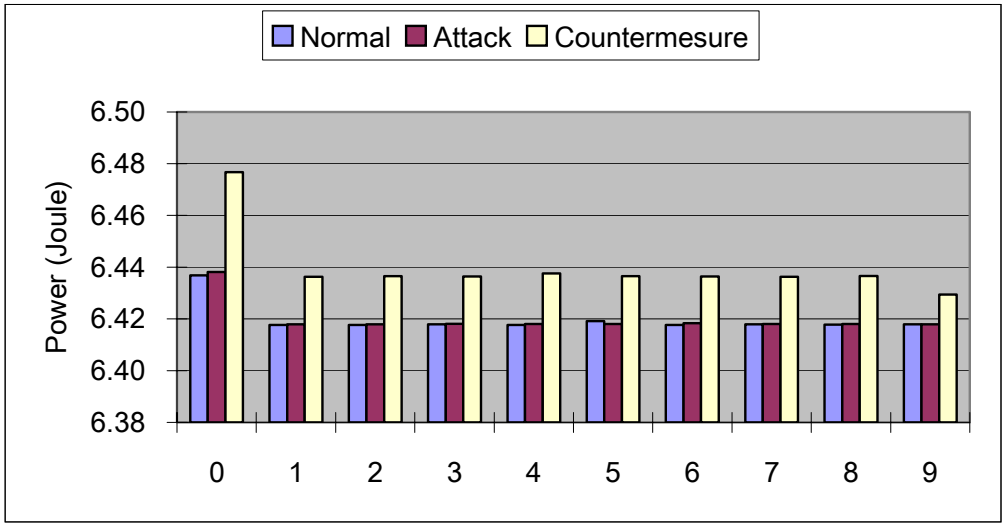


Figure 5.31: Sinkhole CPU Power Consumption

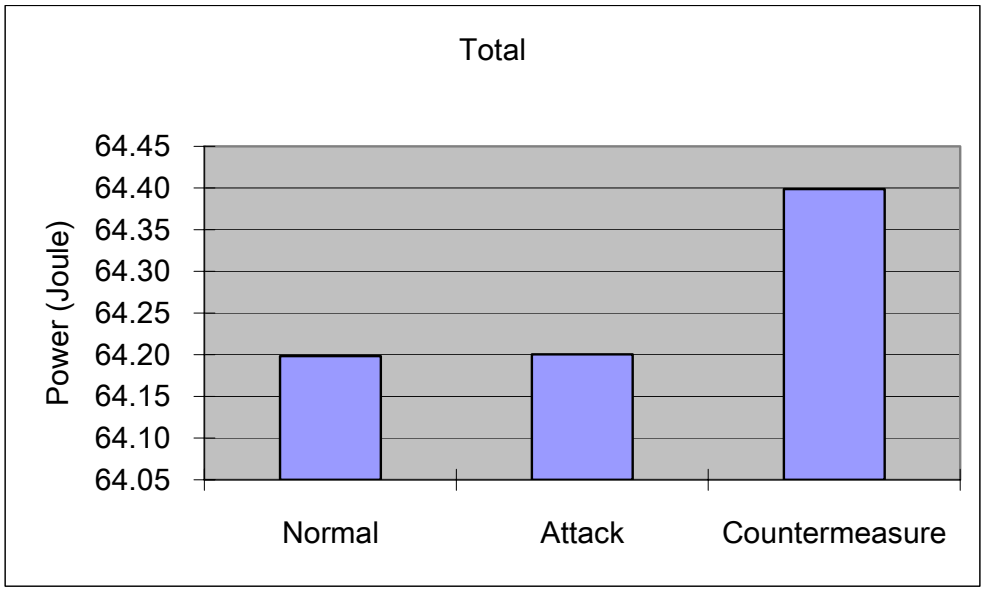


Figure 5.32: Sinkhole CPU Power Consumption – Overall

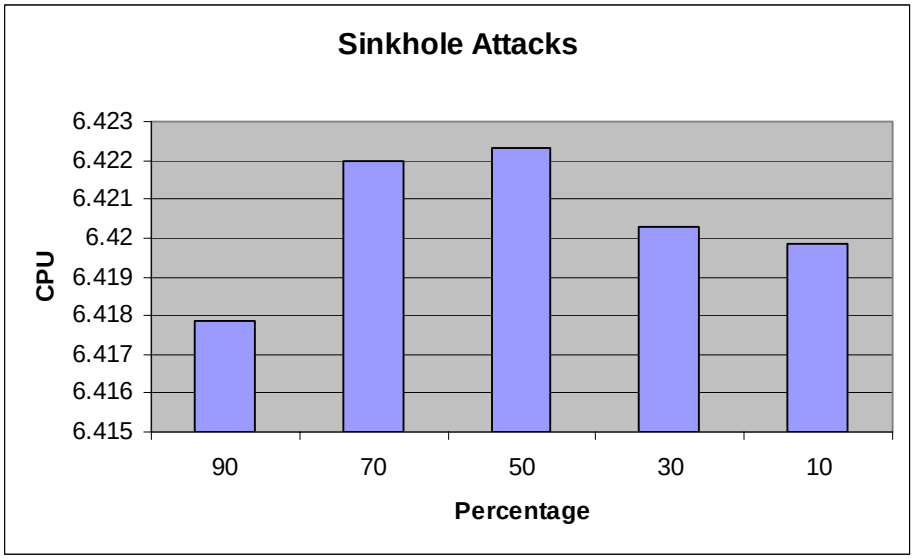


Figure 5.33: Sinkhole Attacks CPU Power Consumption

In the graph in Fig 5.33, 50% (5 Attack and 5 Normal) of the nodes being Attack configuration consume more power than the others. The graph below indicates an increase in power level as the proportion of Counter nodes increase in a network.

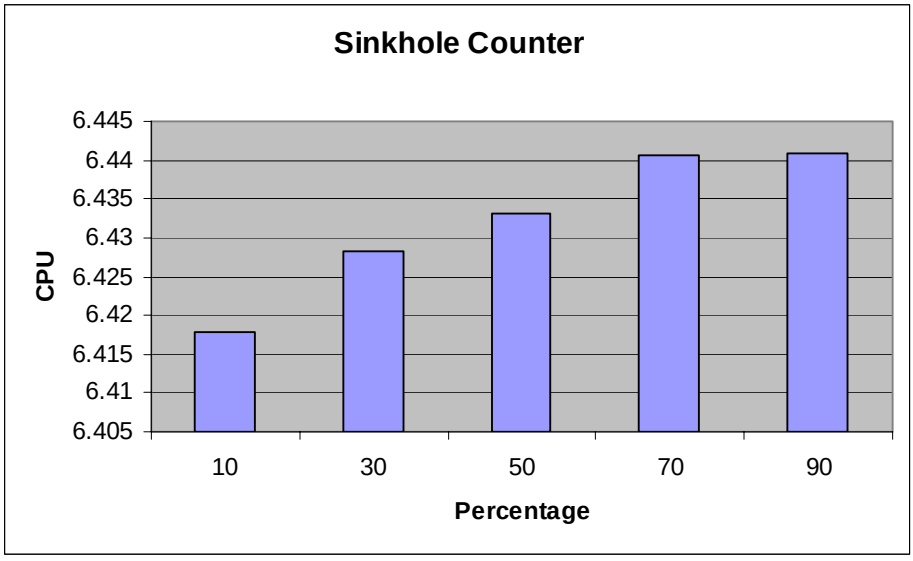


Figure 5.34: Sinkhole Countermeasure CPU Power Consumption

CPU power consumption is expensive during countermeasures. The attacks configuration shows maximum power expenditure during the 70% and 50% mode where 7 and 5 motes are malicious respectively.

5.3.5.6 Conclusion

In sinkhole attack, the power consumption of the sensor network is similar to selective forward attack. We notice a spike in the power consumption readings when the countermeasures are applied. This is because of the increased traffic due to retransmission of the same data twice and redundant packets on the two paths.

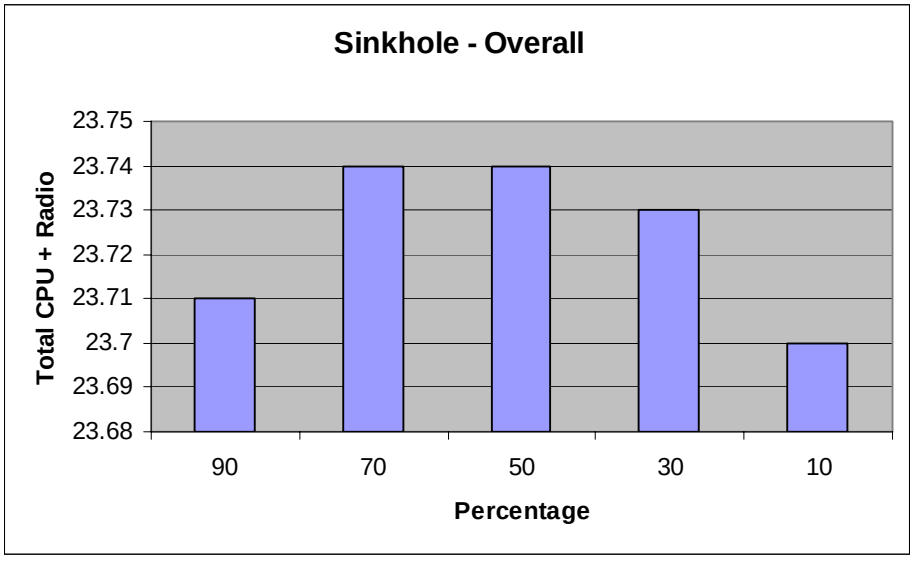


Figure 5.35: Sinkhole Attacks Power Consumption- Overall

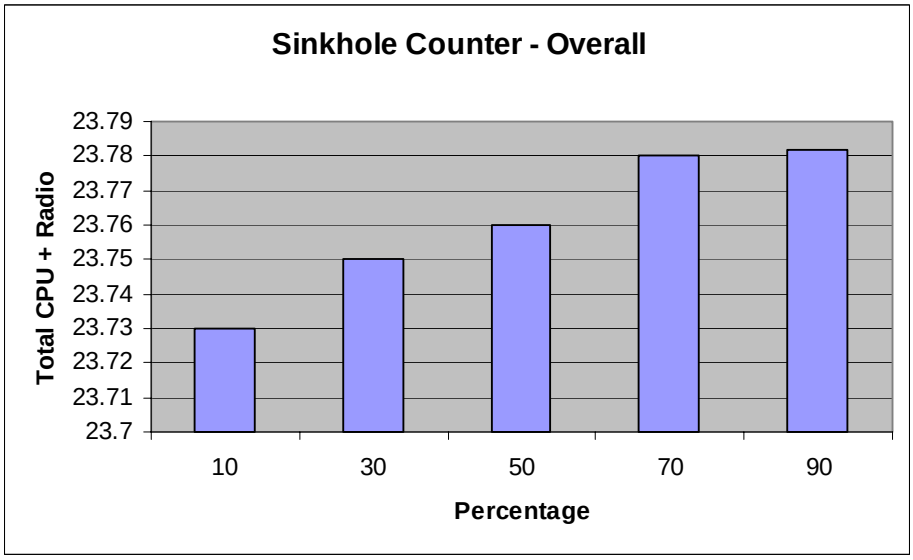


Figure 5.36: Sinkhole Countermeasure Power Consumption- Overall

In the sinkhole attack the countermeasures applied are expensive when compared to the attacks themselves. Fig 5.36 shows that at 10%; Counter motes consumes less power compared to a 90:10 (Counter: Attack) configuration. Graph in

Fig 5.32 also supports this theory as we can see Counter motes consume more power over the full scope of our experiment.

5.4 Sybil attack and countermeasure

5.4.1 Action and result

Attack

A malicious mote assumes the identity of an existing mote on the network. This causes confusion on the network and loss of data.

Countermeasure

This problem could be solved by authenticating the packets in the link layer. Normal motes would not receive messages from a mote whose mote-key is different. Malicious mote does not have the mote-key.

High Level Pseudo Code for Sybil Attack

```
Initialize: Comm,TimerC,LEdC
 Get Communication and timer Parameters
Include TinySec component for Secure transmit ion.
Generate mote key
 Make a neighbor list from nearby motes

 Input: Ready to receive message from other motes
 Compute: Check mote key for each packet received.

Output: Retransmit messages only if they have a valid mote key.
  Show output on Led:
    Green on:      sending message;
    Green off:     send done;
Repeat operation.
```

5.4.2 Network composition

Same as the Network composition in Figure 4.2

5.4.3 Sybil attack and counter details

Both Sybil attack and counter program are secure versions of application *"Surge*" which is used to simulate a normal multi-hop network. Authentication and encryption mechanism have been provided by TinyOS. In the Surge application, we need to change "GenericCommPromiscuous" component to "SecureGenericComm", and when installing the above components on a mote, we need to specify a mote-key (Mote-key is a unique identifier for each mote). [7]. GenericCommPromiscuius and SecureGenericComm are the Interfaces needed for sending and receiving messages. The Generic version does not carry out Encryption/Decryption.

For attacker, when we load the program, we use an address that a normal mote has used. Thus in network, there are two motes whose address is the same.

However we use a different mote-key:

Attacker 6D524D67F24F178B0A69933FDD6C6F7F

For counter, when we load the program, we give each mote a unique address. All the motes have the same mote-key:

Default 6D524D67F24F178B0A69933FDD6C6F7E

5.4.4 Simulation / Power Analysis of Sybil Attack with Encryption / Decryption:-

The simulator cannot model the Sybil attack where a node takes 2 addresses. We instead measure the overhead involved in encrypting/ decryption information on a mote. Encryption/Decryption could avoid Sybil attacks by authenticating packets in the link layer.

By default, TinySec will authenticate all messages, but encryption is turned off. TinySec allows the user dynamically to alter the combination of security mechanisms for the application with the TinySecMode interface. The “AUTH” mode would check for the CRC, and Encrypt/Decrypt would alter the contents of the message body with an Encrypt/Decrypt algorithm.

Let 10 nodes run SybilAttack.od for 10 minutes, and we note the power consumption of CPU and Radio. Next 10 nodes run Surge.od.

Node	CPU	Radio	Total (CPU+Radio)
0	6.596606413420167	17.31140640288086	23.90801282
1	6.577923743991699	17.31336554460449	23.89128929
2	6.576597169171142	17.310418992700193	23.88701616
3	6.576780789803711	17.311841223242187	23.88862201
4	6.5762329404467765	17.310556632202143	23.88678957
5	6.576346703816162	17.310786741162108	23.88713344
6	6.5775006151975095	17.31308867314453	23.89058929
7	6.576381706343506	17.31047837890625	23.88686009
8	6.577122562812011	17.31144219033203	23.88856475

9	6.575540114498535	17.31033415344238	23.88587427
Total	65.78703276	173.1137189	238.90075166

Table 5.4: Sybil Attack Average Radio Power Consumption

Node	CPU	Radio	Total (CPU+Radio)
0	6.436892733870361	17.28294848432617	23.71984122
1	6.417669621845215	17.314971744116207	23.73264137
2	6.417714777305175	17.314971744116207	23.73268652
3	6.4179184683676755	17.314971744116207	23.73289021
4	6.417709932837158	17.314971744116207	23.73268168
5	6.419206613960204	17.322473176147458	23.74167979
6	6.417693816689209	17.314971744116207	23.73266556
7	6.41787958201709	17.314971744116207	23.73285133
8	6.417750651114991	17.31497826911621	23.73272892
9	6.41788059078003	17.31497826911621	23.73285886
Total	64.19831679	173.1252087	237.32352549

Table 5.4.1: Sybil Attack Average Power consumption - Overall

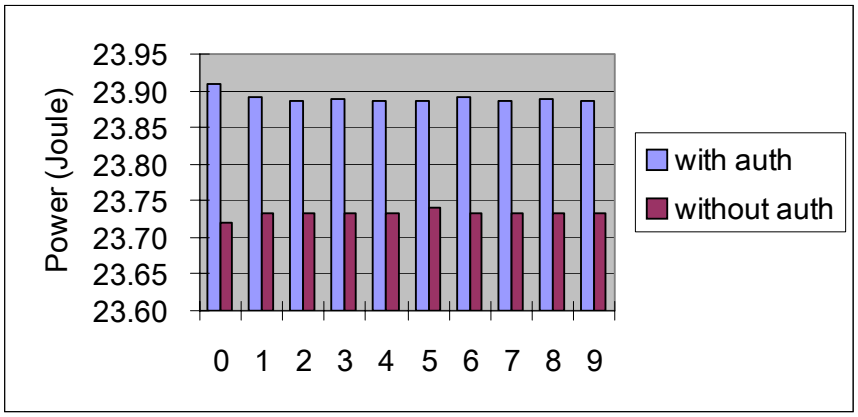


Figure 5.37: Sybil Attack Power Consumption – Overall

With Authentication but without Encryption/Decryption

From the above chart, we can see that here, to authenticate a message (without encryption and decryption); about 0.15 Joule of extra powers will be consumed on each node.

If encryption/decryption is implemented on the data, the data obtained from the corresponding program SybilAuthEncrypt.od(Encryption) for 10 minutes would

be as below.

Node	CPU	Radio	Total (CPU+Radio)
0	6.65703823563501	17.310620923217773	23.96765916
1	6.648699144708251	17.32343519123535	23.97213434
2	6.649376978411377	17.32364078818359	23.97301777
3	6.6491497438186045	17.323546895678707	23.97269664
4	6.650206918786621	17.325626301049805	23.97583322
5	6.648507362891358	17.32324591301269	23.97175328
6	6.648886167982421	17.324282210668944	23.97316838
7	6.6487612824431155	17.323407421313476	23.9721687
8	6.6508710611535635	17.3270611652832	23.97793223
9	6.649017910360107	17.323569971606446	23.97258788
Total	66.50051481	173.2284368	239.72895161

Table 5.4.2: Sybil Attack Average Power consumption - Overall

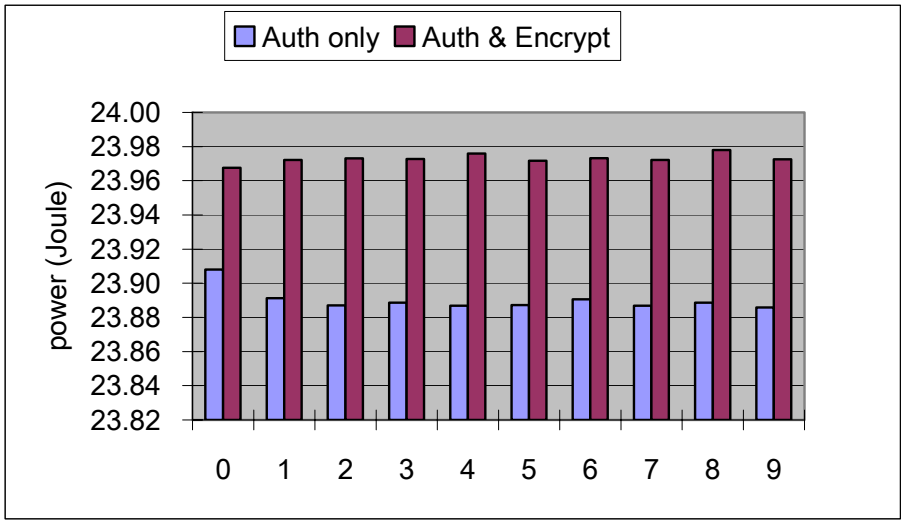


Figure 5.38: Sybil Attack Power Consumption – Overall with Authentication – Encryption / Decryption

5.4.4.1 Conclusion:

About 0.09 Joule of extra powers is being consumed on encryption and decryption on each node with Authentication. Hence we can determine that the power consumption is normally greater in our proposed countermeasure. Real world

experiments may differ, because of the type of authentication used and the level of encryption.

5.5 Hello attack and countermeasure

5.5.1 Action and result

Attack

- Action: The malicious mote sends messages by using stronger radio power, and pretend to be a base station or a node near the base station by modify its hop count value. However, motes far from the malicious node cannot send messages back successfully, because their radio power is low.

- Result: Network confusion and data loss.

High Level Pseudo Code for Hello Attack:

```
Initialize: Comm,TimerC,LEdC
 Get Communication and timer Parameters

 Make a neighbor list from nearby motes
 Set Hop count to 1. To Simulate a Base Station

 Input: Ready to receive message from other motes
 Compute: Send message every two seconds to all motes
Output: Retransmit to all motes on the network.

 Show output on Led:
   Green on:     sending message;
   Green off:    send done;
 Repeat operation
 Repeat operation.
```

Countermeasure

- Action: If a node fails to get through a reply message to a particular node which sends the HELLO, then that node is distrusted.

- Result: Do not send messages to distrust nodes.

High Level Pseudo Code for Hello Attack countermeasure:

```
Hello Countermeasure

Initialize: Comm,TimerC,LEdC
Get Communication and timer Parameters

Input: when a message is received from any mote
Compute: If message is found to be not reaching the base station (malicious)
            Stop sending anymore packets.
Output: Transmit only those packets that can reach a destination.

Show output on Led:
  Green on:     sending message;
  Green off:    send done;

Repeat operation.
```

5.5.2 Network composition

Same as the Network composition in Figure 4.2

5.5.3 Hello attack details

The mote which runs the Hello attack program will pretend to be a mote near the base station (hop count is always 1) and broadcast hello messages in every 2 seconds.

```
task void SendData()
{
  HelloMsg *p = (HelloMsg *)gMsgBuffer.data;

  p->sourceaddr = p->originaddr = TOS_LOCAL_ADDRESS;
  p->hopcount = 1;
  p->seqno = gSendMsgNum;

  dbg(DBG_USR1, "HelloM: Sending message from 0x%x to 0x%x.\n", p->sourceaddr,
gMsgBuffer.addr);
  call Leds.greenOn();
  call SendMsg.send(TOS_BCAST_ADDR, sizeof(HelloMsg), &gMsgBuffer);
}
```

5.5.4 Countermeasure details

The strategy of route selecting was modified a little. If a node fails to reply to repeat HELLO messages from a node, that node will be distrusted.

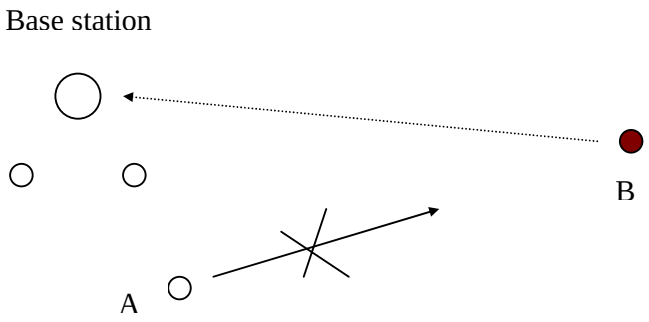


Figure 5.39: Failure to send message back – HELLO countermeasure

In figure 5.35, Node B is the malicious mote that is sending Hello messages. Node A receives the message and takes B as its neighbor. Moreover the hop count of B is 1, and B appears to be a good parent of A. However B is far away from A, when

A sends a message to B, it may be a failure and not reach B. A keeps count of as to how many HELLO replies have been sent out to B. After several failures, A will not trust B and does not send any message to B.

5.5.5 Simulation / Power Analysis - Hello attack

5.5.5.1 Before attack

The Hello attack can be simulated by first running the normal *Surge.od* program in a typical setup of 9 (0 to 8) motes. We measure power consumption for even numbered motes. 0, 4, 8 respectively run for 10 minutes where the unit of measure is in Joules.

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.431	0.000	17.283	1.26	0.004	24.978
4	6.416	0.000	17.315	1.26	0.004	24.995
8	6.416	0.000	17.315	1.26	0.004	24.995

Table 5.5: Typical Power consumption with Surge –
Normal Operation Nodes 0, 4, 8

5.5.5.2 After attack

We inject a malicious node (node 9), into the network and run Program HelloAttack.od (Run 10 minutes, unit is Joule) .This malicious node will flood the network with a HELLO messages thus starting a chain reaction throughout the network.

Node	CPU	Led	Radio	Sensor Board	Flash	Total

0	6.465	0.000	17.283	1.26	0.004	25.012
4	6.447	0.000	17.315	1.26	0.004	25.026
8	6.447	0.000	17.315	1.26	0.004	25.026
9	6.519	4.467	17.870	1.26	0.004	30.12

Table 5.5.1: HELLO Attack Power consumption

5.5.5.3 During counter measures

In the Hello countermeasure, every node will maintain a list called a "neighbor list" to record information of its neighbor nodes. If the node finds that one of its neighbor fails by sending messages above a set limit at one time then this node is eliminated from the list or distrusted, and we do not send any more messages to it.

Change *Surge.od* to *HelloCounter.od*, and run again. (Run 10 minutes, unit is Joule)

Node	CPU	Led	Radio	Sensor Board	Flash	Total
0	6.465	0.000	17.283	1.26	0.004	25.012
4	6.447	0.000	17.315	1.26	0.004	25.026
8	6.447	0.000	17.315	1.26	0.004	25.026
9	6.519	4.705	17.867	1.26	0.004	30.355

Table 5.5.2: HELLO countermeasure Power consumption

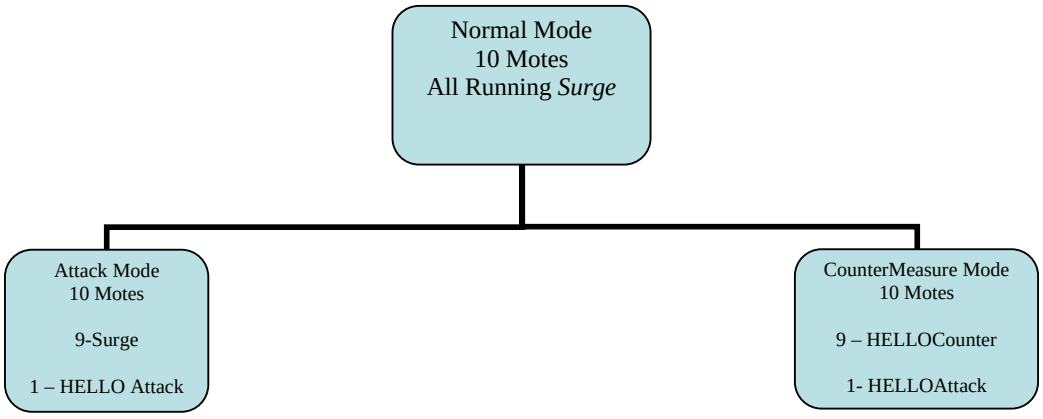


Figure 5.40: Program Execution sequence. *Surge* – Normal, *HELLOAttack* - Attack, *HELLOCounter* – Countermeasures

5.5.5.4 Only radio power consumption comparison

Now we compare radio power consumption before attack, after attack and during counter: (unit is Joule)

Node	Before	After	During
0	17.28294848432617	17.28294848432617	17.28294778046875
1	17.314971744116207	17.31989839802246	17.32471724689941
2	17.31498044411621	17.314971744116207	17.314971040258786
3	17.314971744824216	17.31498044411621	17.314971040258786
4	17.314971744116207	17.31497826911621	17.314971040258786
5	17.314971744116207	17.314971744116207	17.314971040258786
6	17.314978269824216	17.316042755834957	17.32043320002441
7	17.320433903881835	17.329430402319336	17.32300362814941
8	17.31498261911621	17.314971744824216	17.314971040258786
9		17.86987400952148 (do not add)	17.867303853271483 (do not add)
Total	155.8082107	155.823194	155.8259571

Table 5.5.3: HELLO Attack Radio Average Power consumption

From section 4.6.8

Power increase rate 1 = (After – Before)/before = 9.6165021937447870325e-5

Power increase rate 2 = (During – Before)/before = 1.13899003911736725e-4

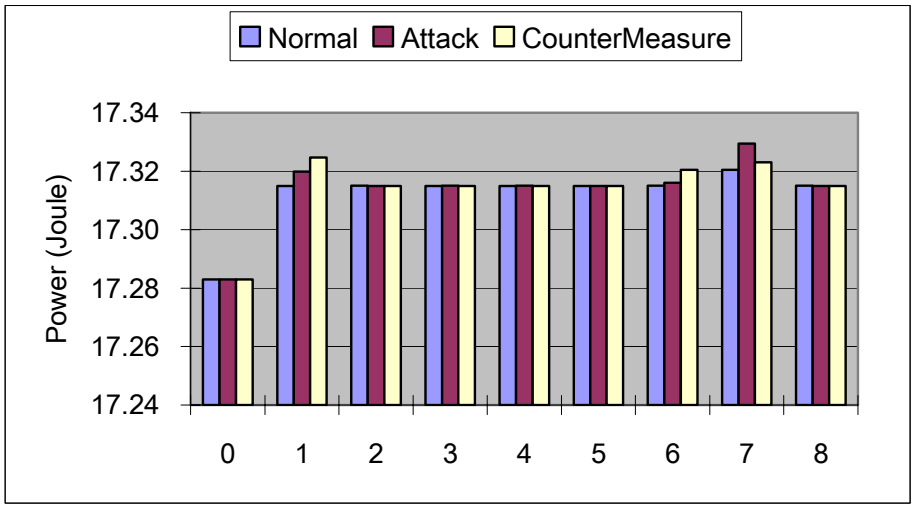


Figure 5.41: HELLO Radio Power Consumption

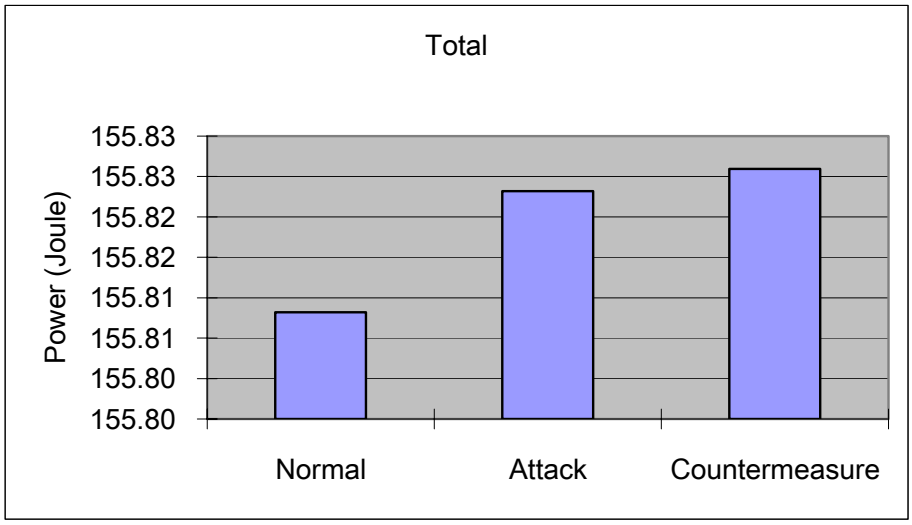


Figure 5.42: HELLO Radio Power Consumption - Overall

In the HELLO attack radio power consumption is higher than the CPU due to the increase in traffic caused by the HELLO message. The base station sees very little traffic and only the nodes seem to be affected. This shows that the nodes are more vulnerable to this type of attack.

The reason countermeasure costs are higher is because the HELLO packet *received* from nodes which have been removed from the neighbor list for which no corresponding reply is sent back.

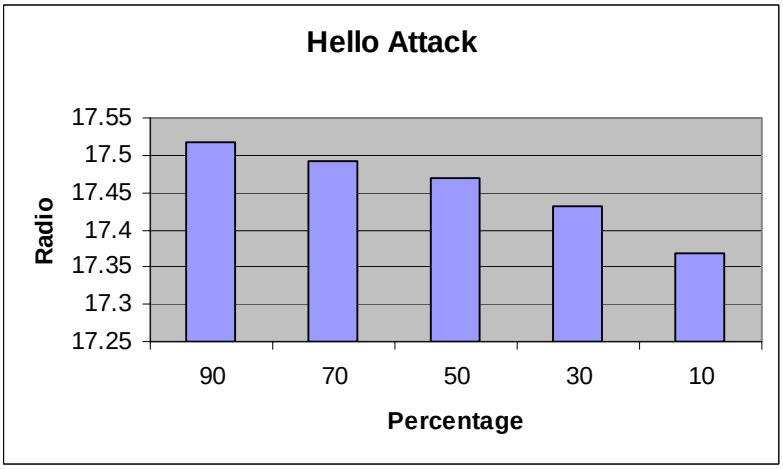


Figure 5.43: HELLO Attack Radio Power Consumption

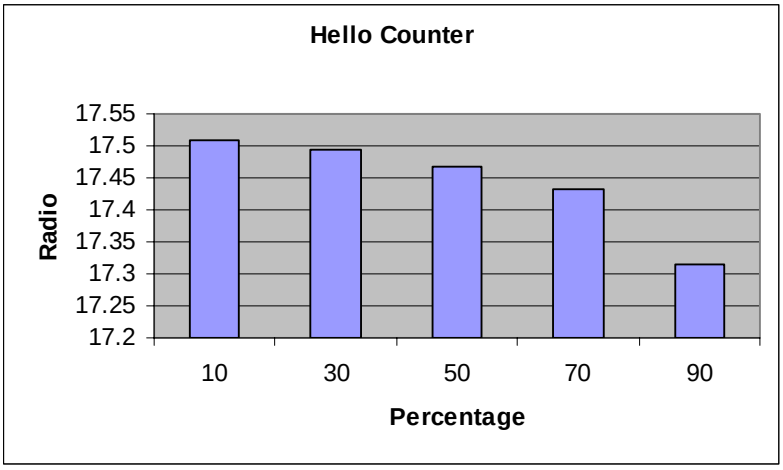


Figure 5.44: HELLO Countermeasure Radio Power Consumption

Fig 5.42 shows that Countermeasures consume a marginally higher power level than the attacks. Similarly from Figs 5.43 and 5.44 higher power levels can be noticed at 90:10 configuration (9 bad nodes and 1 good node) in attacks and 10:90 (1 countermeasure node and 9 bad nodes) in Countermeasures, this is because of countermeasures not being entirely effective. In both cases a higher number of attack nodes caused more spikes in power levels than a smaller proportion. At 10:90 (Counter: Attack) ratio, the counter attack power level was found to be lower when compared to a similar attack configuration of 90:10 (Attack: Normal).

5.5.5.5 Only CPU power consumption comparison

Node	Before	After	During
0	6.43128427829541	6.465208881431397	6.4649492134775395
1	6.416177063417724	6.448134219654053	6.448881996545653
2	6.416095092406739	6.446888423201415	6.446766703149903
3	6.416248120197999	6.447075559896973	6.446752494543458
4	6.416254074820801	6.44725894853125	6.44703907945459
5	6.416176814234375	6.447446990879396	6.447067381527588
6	6.416276387900878	6.447544859788085	6.448334918797851
7	6.4173069449316404	6.450182401958985	6.448545011294677
8	6.416049885391601	6.4471048535478515	6.446822740188965
9		6.519386674160156 (do	6.519110843658691 (do

		not add)	not add)
Total	57.76186866	58.04684514	58.04515954

Table 5.5.4: HELLO countermeasure CPU Average Power consumption

From section 4.6.8

Power increase rate 1 = (After – Before)/before = 0.004933643710134082770

Power increase rate 2 = (During – Before)/before = 0.00490446182874582922

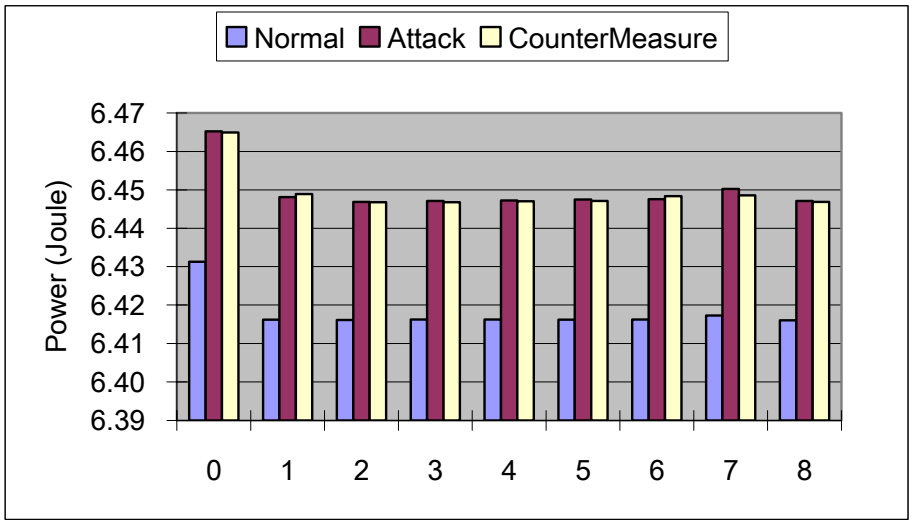


Figure 5.45: HELLO attack CPU Power Consumption

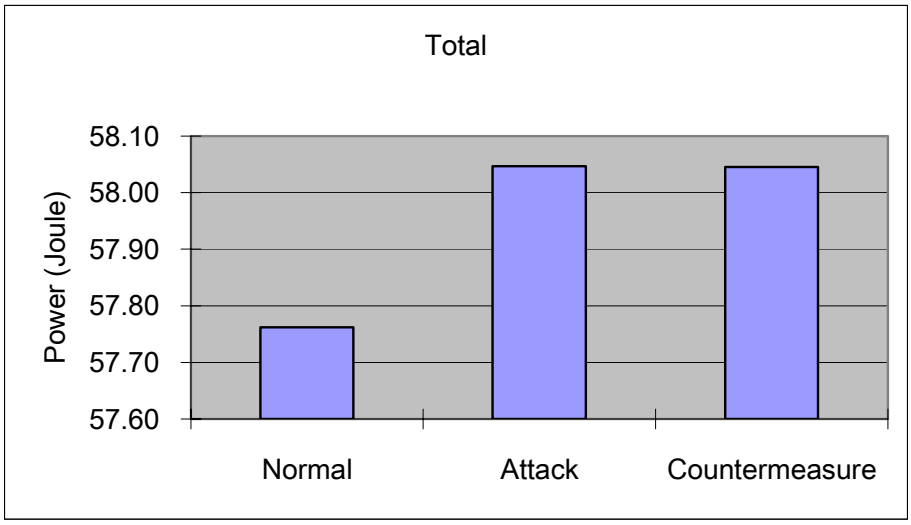


Figure 5.46: HELLO attack CPU Power Consumption - Overall

CPU power consumption is smaller in magnitude when compared to the radio traffic. This is because of the small number of tasks given to a CPU in a mote when a HELLO attack occurs. Countermeasures power levels are smaller when compared to the attacking modes. The CPU is only involved in maintaining the neighbor list.

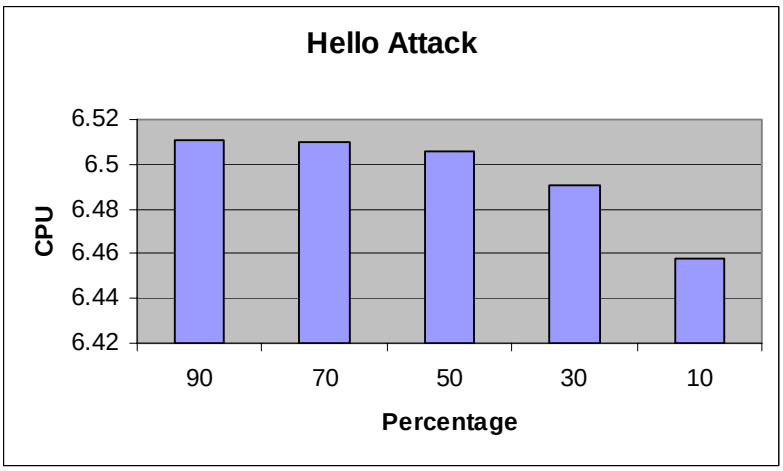


Figure 5.47: HELLO attack CPU Power Consumption

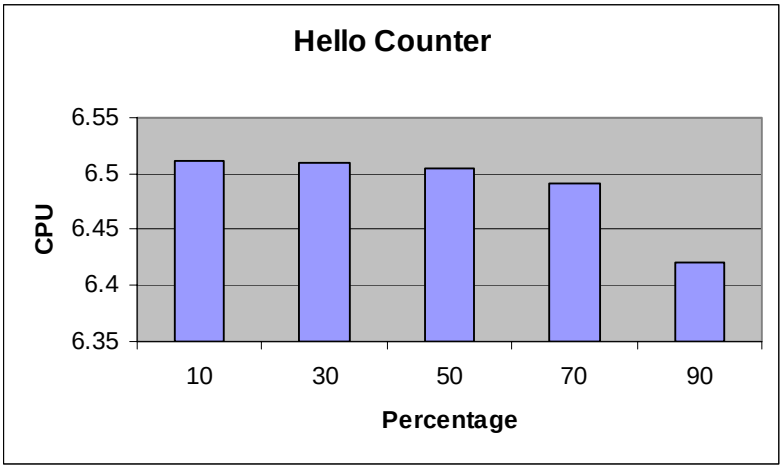


Figure 5.48: HELLO Countermeasure CPU Power Consumption

5.5.5.6 Conclusion

A hello attack costs very little power to counter. The overall power consumed in the countermeasure is much less when compared to the other forms of attacks. HELLO message acknowledgment limit is set to 25 in our experiments, bringing this number down would still limit the power consumption further.

The difference in power levels (Radio) is as much as 0.09% and 0.11 % on attacks and countermeasures. The difference between the two is not significant but the reason for a countermeasure being much higher is the flooding of the network with lot of HELLO messages. On the contrary, CPU power expenditure is lower for countermeasures compared to Radio power consumption.

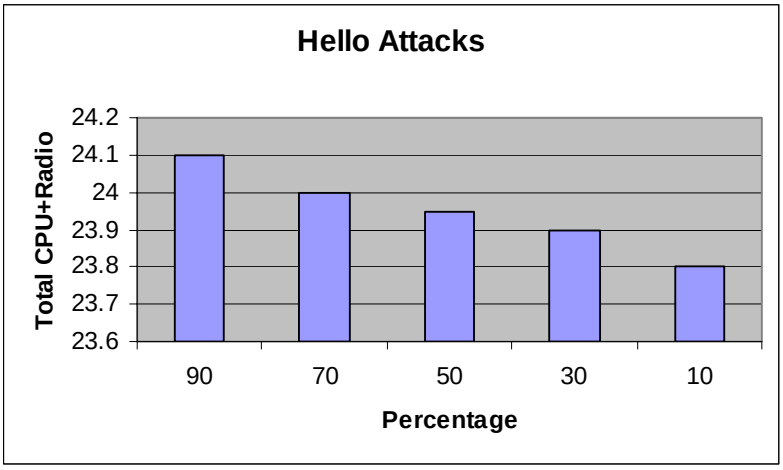


Figure 5.49: HELLO Attack Power Consumption – Overall

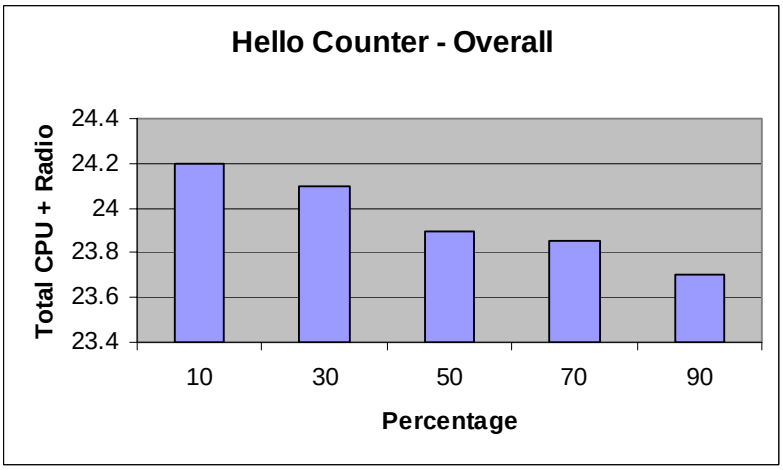


Figure 5.50: HELLO Countermeasure Power Consumption – Overall

5.6 Wormhole attack

A wormhole attack can be defined as one where two motes on a network being far apart from each other deceive the other motes on the network into thinking that the

distance is shorter to reach the base station if the packets travel between them. A decision to take that route would result in packets being lost or getting delayed. A wormhole attack is difficult to detect and countermeasure. In our experiment we decided to use different channels to simulate an attack. For example the two motes in question would be on channel 17 and the rest of the motes on the network would use channel 22. [11] Or they could be on a different frequency, 433 and 916 MHz. Theoretically this setup can work on a simulator but cannot be duplicated with real motes. Motes work on a fixed frequency and this is a limitation of the radio that's built onto them.

Similarly for countermeasure, it is impossible to check and modify a Channel ID of a mote during runtime. A node can only modify its own channel ID, by using the functions available in the CC2420 radio stack such as "TuneManual ()". A malicious node would not change its channel ID by itself in this regard.

To Change the channel in a mote

Changes in the Make File to include
CFLAGS += -DCC2420_DEF_CHANNEL=x
Where x is between channel 11 and 26

CFLAGS=-DCC2420_DEF_CHANNEL=22 make telos
CFLAGS=-DCC2420 DEF CHANNEL=13 make telos

Let node 8 and node 9 transmit data at 916MHz channel, and node 0 – 7 transmit data at 433MHz.

Node	CPU	Radio	Total (CPU+Radio)
0	6.5174162215937494	17.292089890568032	23.80950611
1	6.511739680535156	17.38051706500651	23.89225675
2	6.511564614624755	17.380540147680662	23.89210476
3	6.511588183932617	17.380497939982096	23.89208612
4	6.5118502423281255	17.38050924910482	23.89235949

5	6.511660555369873	17.380519677539063	23.89218023
6	6.5118372968234866	17.380595147241213	23.89243244
7	6.511551576320801	17.38052602746582	23.8920776
8	6.512465730357178	17.382710398746745	23.89517613
9	6.5143572003942865	17.408891925105795	23.92324913
Total	65.1260313	173.7473975	238.8734288

Table 5.6.1: Wormhole Power consumption (433MHz & 916MHz) – Average

Node	CPU	Radio	Total (CPU+Radio)
0	6.516963577461426	17.29209536669922	23.80905894
1	6.51114897164209	17.380584572745768	23.89173354
2	6.510851532647949	17.380632121752928	23.89148365
3	6.511435345177002	17.386137372965496	23.89757272
4	6.511041412078857	17.380480457080075	23.89152187
5	6.511179158916017	17.380578258683265	23.89175742
6	6.510981817732666	17.38201775579427	23.89299957
7	6.5113287342290045	17.385510646166992	23.89683938
8	6.511048169243897	17.380615702490235	23.89166387
9	6.510974629222657	17.38051327855631	23.89148791
Total	65.11695335	173.7291655	238.84611885

Table 5.6.1: Wormhole Power consumption 433MHz

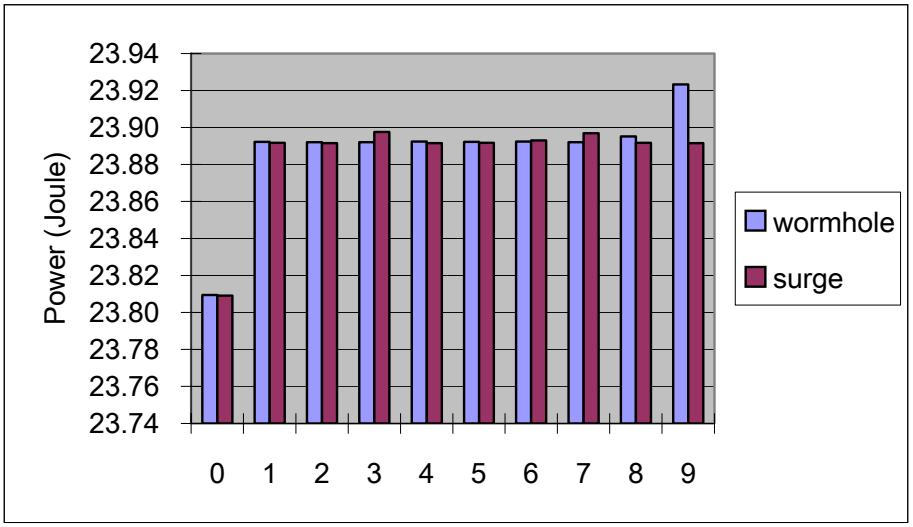


Figure 5.51: Wormhole Power Consumption - Comparison

The above setup yielded constant power consumption and erratic changes not influenced by both the frequencies. This is because in a real world experiment the two malicious motes would be far apart and working in a particular channel/frequency

thus deceiving the other motes into believing that they are actually closer, a shortcut to the base station.

5.6.1 Conclusion:

Wormhole attacks would consume more power with the malicious motes that are involved in the attack; this is because of the quantity of traffic that may propagate through them. Countermeasures if devised would be effective in stopping packets from reaching these motes altogether, they would need to maintain a routing table and work on finding the actual path to the base station. It would be interesting to see the effects of an actual Wormhole attack on such motes.

6. Conclusion and Future Work

Research on sensor networks has been on the mainstream for the past couple

of years. One of the major focuses has been on the security and the quantity of power they consume. Relating these two fields would yield us some valuable insight on how they would perform in a given scenario. The different attacks defined for sensors were simulated with the resulting countermeasures. Some of the attacks were damaging as in they consumed more power than usual and some of the countermeasures applied were effective in curtailing the power consumed. Sometimes it may be more effective to quarantine the network and restrict the influence of malicious nodes, instead of resorting to counter measures.

Countermeasures were effective in stopping the attack with minimal power expenditure for Spoof attack, compared to selective forwarding and Sinkhole attacks, where the power consumed was much higher. The power consumed in spoof countermeasures was less than that consumed during an attack enabling the motes to recover with minimal loss of power. With Selective forwarding the effects were just the opposite with more power being consumed during countermeasures. Hello attack/countermeasures showed a mixed mode of power consumption, with attacks and countermeasures consuming the same levels. The normal operations of most of these motes were documented using the Surge program which showed a consistent power level. Though the magnitude of power difference (E.g.: 0.1% - 0.6% for 10 minutes) in most of our experiments is not significant enough to alter the power consumption of the single sensor, the study when extrapolated to a bigger network with larger number of motes working for longer periods of time would offer significant benefits.

The decision to apply a countermeasure or not depends on the current power level left on the mote. It also depends on factors like the motes ability to recover from such an attack and provide seamless service for the rest of its operational life. Future

works in this area could be to develop effective detection and prevention mechanisms.

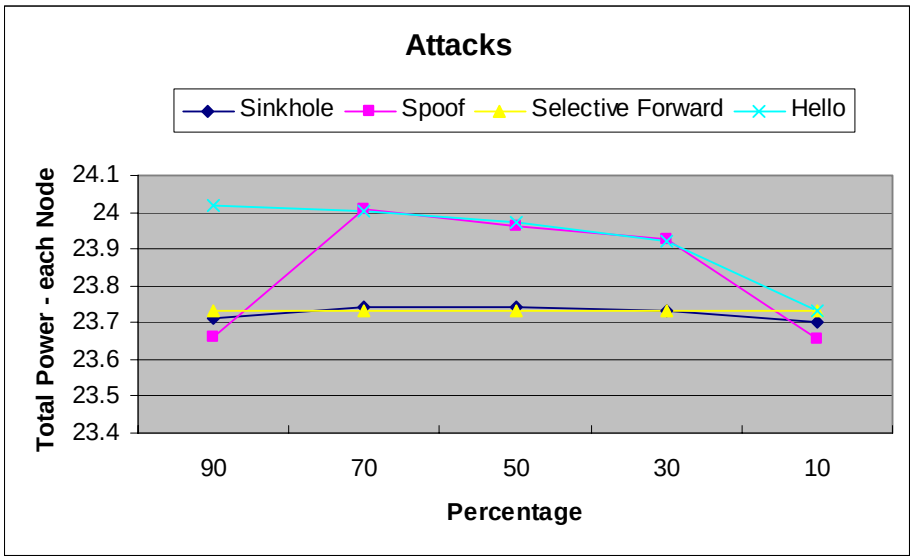


Figure 5.52: Comprehensive Attacks Power Consumption - Comparison

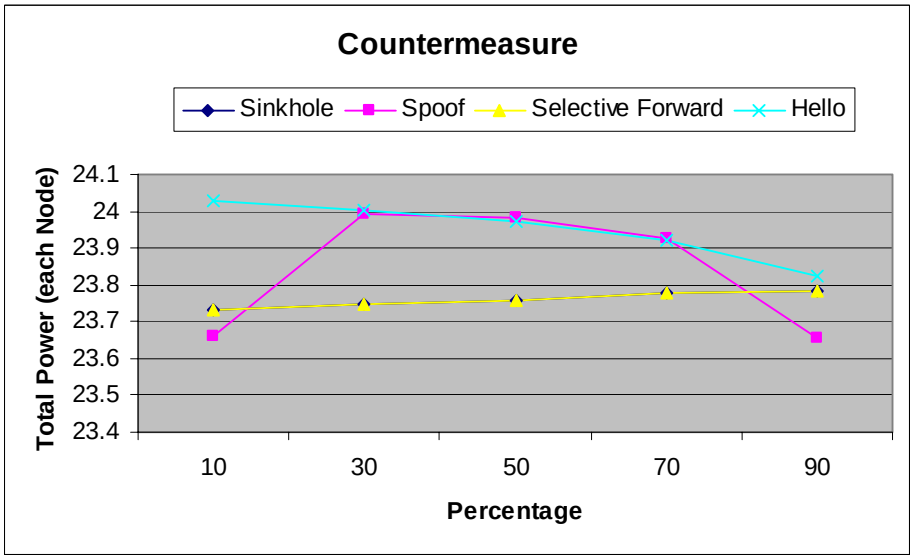


Figure 5.53: Comprehensive Countermeasure Power Consumption – Comparison

REFERENCES

[1] Chris Karlof and David Wagner. “Secure Routing in Sensor Networks: Attacks and Countermeasures”,, Ad hoc Networks, Vol 1, issues 2--3, pp. 293-315, Elsevier, September 2003

[2] M. Pirretti, S. Zhu, N. Vijaykrishnan, P. McDaniel, M. Kandemir and R. Brooks, “The Sleep Deprivation Attack in Sensor Networks: Analysis and Methods of Defense”, Proc. Conference on Innovations and Commercial Applications of Distributed Sensor Networks, October, 2005

[3] Victor Shnayder, Mark Hempstead, Borrong Chen, Geoff Werner Allen, and Matt Welsh, “Simulating the Power Consumption of Large-Scale Sensor Network Applications”, Proc. Conference on Embedded Network Sensor Systems, pp. 199 – 200 , 2004

[4] Prabal Dutta, Mike Grimmer, Anish Arora, Steven Bibyk, and David Culler, “Design of a Wireless Sensor Network Platform for Detecting Rare, Random, and Ephemeral Events”, Proc Fourth International Symposium on Information Processing in Sensor Networks pp 497 – 502, IPSN 2005

[5] J. Deng, R. Han, S. Mishra, "A Performance Evaluation of Intrusion Tolerant Routing in Wireless Sensor Networks",Proc IEEE 2nd International Workshop on Information Processing in Sensor Networks, 2003

[6] A. Ledeczi, P. Volgyesi, M. Maroti, G. Simon, G. Baloga, A. Nadas, B. Kusy, and S. Dora. “Multiple Simultaneous Acoustic Source Localization in Urban Terrain” Proc Fourth International Symposium on Information Processing in Sensor Networks pp 491-496 IPSN, 2005

[7] A. Sinha and A. Chandrakasan, "Joule Track - A web based tool for software energy profiling", Proc. Design Automation Conf., pp. 220--225, June 2001

[8] TinyOS Documentation, http://tinyos.net/tinyos-1.x/doc/ [last accessed - May 10, 2006]

[9] Chris Karlof,Naveen Sastry,David Wagner , TinyOS Documentation http://www.tinyos.net/tinyos-1.x/doc/tinysec.pdf [last accessed - May 10 , 2006]

[10] Chris Karlof,Naveen Sastry,David Wagner - TinySec: User Manual, http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html [last accessed - May 10, 2006]

[11]Tiny OS Resource on TinyOS.net MICA2 Radio Stack for TinyOS http://www.tinyos.net/tinyos-1.x/doc/mica2radio/CC1000.html [last accessed - May 10, 2006]

[12]Avrora Simulator on UCLA web server

http://compilers.cs.ucla.edu/avrora/releases/avrora-beta-1.6.0.jar [last accessed - May 10, 2006]

[13] Tiny OS Resource on TinyOS.net - Multi Hop Routing
http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html [last accessed - May 10, 2006]

[14] Qinghua Zhang, Pan Wang, Douglas S. Reeves, Peng Ning
"Defending against Sybil Attacks in Sensor Networks",Proc International Conference on Distributed Computing Systems Workshops, pp 185-191, 2005

[15] L. Hu and D. Evans, "Using Directional Antennas to Prevent Wormhole Attacks", Proceedings of Network and Distributed System Security Symposium (NDSS), 2004

[16] Chien – Liang Fok "Tiny OS Tutorial"
http://www.cse.wustl.edu/~lu/cs520s/slides/tinyos_tutorial.pdf [last accessed - May 10, 2006]

[17] James Newsome, Elaine Shi, Dawn Xiaodong Song, Adrian Perrig, "The sybil attack in sensor networks: analysis & defenses", Proc Information Processing in Sensor Networks, 259-268, 2004

[18] Issa Khalil, Saurabh Bagchi, Ness B. Shroff, "A Lightweight Countermeasure for the Wormhole Attack in Multihop Wireless Networks", Proc International Conference on Dependable Systems and Networks , pp. 612-621 , 2005

VITA

Karthikram Sheshachalam

Candidate for the Degree of

Masters of Science

Thesis: ENERGY CONSUMPTION DURING ATTACKS AND COUNTERMEASURES IN SENSOR NETWORKS.

Major Field: Computer Science.

Biographical:

Personal Data: Born in Coimbatore, India on June 28th, the son of Mr. V.G.Sheshachalam and Mrs.Renuka Sheshachalam

Education: Obtained Senior High School Diploma from Stanes H.S.S, India in May 1998. Received my Bachelors in Computer Science and Enggineering from Annamalai University,Annamalai Nagar, May 2002. Completed the requirements for Master of Science at Oklahoma State University, Stillwater, May 2006.

Experience: Jan 2004 – Dec 2004 Para-Professional, Technology Operations, Oklahoma State University, Stillwater.

Jan 2005 – Dec 2005 Graduate Assistant, CEAT Labs, Oklahoma State University, Stillwater.

Name: Karthikram Sheshachalam Date of Degree: May, 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: ENERGY CONSUMPTION DURING ATTACKS AND COUNTERMEASURES IN SENSOR NETWORKS

Pages of Study: 93 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

Sensors have found several useful applications. Sensor Networks are formed with a number of small low cost sensor devices which connect wirelessly to gather data. A sensor has the ability to collect data from its surroundings and transmit to a central base station where it can be processed into meaningful information. These devices have limited power supply on deployment and cannot be replaced or renewed. Although many secure aspects of sensor networks have been proposed, little work has been reported on the impact of malicious attacks and countermeasures on the energy levels. Our study looks at some aspects of efficiency and power consumption during attacks and countermeasures. Decisions based on the data would enable the nodes in the network to take an optimum course of action to conserve power and prolong their life.

Findings and Conclusions:

The attacks and countermeasures defined for Sensor Networks were designed and simulated using Avrora – a simulator and power analysis tool. The data gathered from this setup was analyzed and compared to find the best solution in defending a network. The results showed that only some of the countermeasures applied were economical with relation to power. Results showed that countermeasures for attacks such as Hello, Selective Forward and Sinkhole consumed more power than countermeasures to other attacks. This study also shows that attacks such as Hello and Spoof are more energy intensive than attacks such as Selective Forwarding, Sinkhole, Sybil and Wormhole Attacks.

ADVISER'S APPROVAL: Dr.Johnson P Thomas