UNIVERSITY OF CALGARY

Defending against Link Quality Routing Attacks in Wireless Sensor Networks

by

Islam Hegazy

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

September, 2011

UNIVERSITY OF
CALGARY

Abstract

Wireless sensor networks are gaining popularity due to their ease of deployment, low cost, portability, and scalability. However, the sensor nodes, which are the components of wireless sensor networks, are limited in communication, processing, and storage capabilities. They collect data from their surrounding environments and relay it using wireless communication to a central controller, which is more capable of processing and storing data. The wireless communication exposes the wireless sensor networks to different types of routing attacks.

The goals of this dissertation are: study the two types of link quality routing protocols, identify the vulnerabilities of each type, and develop intrusion detection mechanisms to detect any malicious node that uses the identified vulnerabilities. Analysis and simulation are adopted as the research methodology.

The first type of protocols requires the sensor nodes to cooperate to compute link qualities. An intrusion detection mechanism is proposed that introduces a gap in the sequence number of packets to detect a malicious node that advertises false link quality values. Hence, sensor nodes can expect the values of the link qualities of their neighbours to detect violators.

The second type of protocols does not require the sensor nodes to cooperate. However, a malicious node may use false values for the parameters of the routing protocol, such as advertising false routing cost or sending frequent beacons. An intrusion detection system with two modules is proposed to detect this malicious node. The first module applies the watchdog concept to detect false routing costs and the second module applies a state machine to test the frequency of broadcasting beacons.

The two intrusion detection systems and a routing protocol from each type are simulated in ns-2. The simulation results of the first intrusion detection system show that a

small gap in the sequence numbers helps the sensor nodes to detect the malicious node effectively. The simulation results of the second intrusion detection system show that the success of the watchdog mechanism is dependent on the density of the sensor nodes in the network. However, the state machine mechanism helps the sensor nodes to detect the malicious node regardless of their density.

Acknowledgements

First and foremost, I thank Allah (God) for giving me patience and knowledge to complete this work. Without his guidance and support, I would not have been who I am now.

I would like to thank my supervisors, Professor Reihaneh Safavi-Naini and Professor Carey Williamson, their insights and helpful comments helped this dissertation to come to light. Their invaluable directions enriched my academic thinking and writing. Their guidance and regular meetings enlightened me throughout this work. It is a privilege to be in their research groups.

I am grateful for the friends and colleagues in the iCORE Information Security lab (iCIS), Experimental Laboratory for Internet Systems and Applications (ELISA) lab, and the Department of Computer Science in general. My learning experience in those environments was exceptional.

My family has been a continuous support throughout my life. I would like to thank them, especially my parents, for raising me up as an independent person. I know that it is hard for them to have me living away but I am sure that they are proud of me. My words cannot express my gratitude towards you.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

4-bitle Four-Bit Link quality Estimator

ACK Acknowledgement

AODV Ad hoc On-demand Distance-Vector

BLQ Beacon Link Quality

BS Base Station

CPU Central Processing Unit

CTP Collection Tree Protocol

DLQ Data Link Quality

DoS Denial-of-Service

DSDV Destination-Sequenced Distance-Vector Routing

EBS Exclusion-Based System

ECC Elliptic Curve Cryptography

ETX Expected Transmission count

FIFO First In First Out

FP False Positive

GEAR Geographical and Energy Aware Routing

GPS Global Positioning System

ID Identifier

IDS Intrusion Detection System

LEACH Low Energy Adaptive Clustering Hierarchy

LEEP Link Estimation Exchange Protocol

LQI Link Quality Indicator

LRU Least Recently Used

MAC Medium Access Control

MANTIS MultimodAl system for NeTworks of In-situ wireless Sensors

MANET Mobile Ad hoc Network

MCFA Minimal Cost Forwarding Algorithm

MWE Multiple Winner algorithm

NAWMS Nonintrusive Autonomous Water Monitoring System

nesC Network Embedded Systems C

ns-2 Network Simulator-2

OSI Open Systems Interconnection

OSPF Open Shortest Path First

PRR Packet Reception Ratio

QoS Quality of Service

RandomLFSR Linear Feedback Shift Register pseudo random number generator

RandomMLCG Multiplicative Linear Congruential Generator

RETOS Resilient, Expandable, and Threaded Operating System

RF Radio Frequency

RIP Routing Information Protocol

RNP Required Number of Packets

RSSI Received Signal Strength Indicator

SNR Signal-to-Noise Ratio

SPIN Sensor Protocols for Information via Negotiation

Tcl Tool Command Language

TEP TinyOS Enhancement Proposal

THL Time Has Lived

TinyOS Tiny Operating System

TP True Positive

WMEWMA Window Mean Exponentially Weighted Moving Average

WSN Wireless Sensor Network

List of Publications

Islam Hegazy, Reihaneh Safavi-Naini, and Carey Williamson, "Exploiting Routing Tree Construction in CTP," in Proceedings of the 12^{th} International Workshop on Information Security Applications (WISA), 2011.

Islam Hegazy, Reihaneh Safavi-Naini, and Carey Williamson, "Towards Securing MintRoute in Wireless Sensor Networks," in Proceedings of the 1^{st} International Workshop on Data Security and PrivAcy in wireless Networks (D-SPAN) / the 2010 IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM), IEEE, June 2010.

Rennie deGraaf, Islam Hegazy, Jeffrey Horton, and Reihaneh Safavi-Naini, "Distributed Detection of Wormhole Attacks in Wireless Sensor Networks," in Proceedings of the 1^{st} International Conference on Ad Hoc Networks (ADHOCNETS), vol. 28 of LNICST, pp. 208-223, Springer Berlin / Heidelberg, September 2009.

Chapter 1

INTRODUCTION

1.1 Motivation

A Wireless Sensor Network (WSN) is a special type of wireless networks composed of small nodes that collect data from their surrounding environments. These nodes send the collected data to a central collector for further processing, decision making, and storage. This central controller is called a Base Station (BS) or a sink. The base station connects the WSN to an existing infrastructure where the user can access the collected data [1]. Recent technological advances in electro-mechanical systems, digital electronics, and wireless communication have enabled the design and development of these nodes. These nodes contain sensors to sense the environment, storage and processing units to handle the data, and a wireless module for communication. These nodes are called wireless sensor nodes or sensor nodes, for short. Advances in electro-mechanical systems enable the components of a wireless sensor node to fit into a small chip. Advances in digital electronics empower the small chips to process received data. Advances in wireless communications enable the use of Radio Frequency (RF) antennas to relay the collected data among the sensor nodes. These sensor nodes are small in size, low-cost, low-power, and multifunctional [2, 3]. Figure 1.1 compares a sensor node to a quarter dollar [1].

Although these advances in technology have helped in the wide use of WSNs, they have their shortfalls on the development of the wireless sensor nodes as well. The sensor nodes are highly constrained devices: they operate on batteries; and they have limited computational and storage capabilities. As a result, applications of WSNs should con-

[1]Image is courtesy of http://www.russnelson.com/wisan/

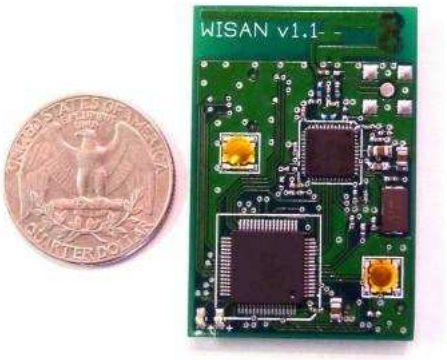


Figure 1.1: A sensor board compared to a quarter dollar

serve as much storage space and battery power as possible to prolong the lifetime of the network. Despite their limited capabilities, WSNs are gaining more attention in several application fields. WSNs can be used in: military applications, such as battlefield surveillance; in environmental applications, such as fire detection; health application, such as patient tracking; home applications, such as home automation; and other applications, such as interactive museums, managing inventory control, and vehicle tracking and detection [4].

Sensor nodes play the dual role of collecting data and relaying data from other sensor nodes until it reaches the base station. A WSN may be deployed in a vast area such that not all the sensor nodes can communicate directly with the base station. To overcome this limitation, the sensor nodes cooperate to deliver the data to the base station using a routing protocol. Since the sensor nodes use wireless communication to relay the collected data, they are susceptible to different types of attacks that target the routing protocol. These attacks are different from the attacks against wired networks because they exploit the wireless nature of communication in WSNs.

Attacks on WSNs take advantage of the wireless communication and the vulnerability of sensors to being captured. It can be difficult to guarantee physical security for the vast areas where WSNs are deployed; this enables attackers to capture and examine, modify,

or remove the sensors, and to insert devices such as malicious nodes or repeaters into the environment. Modified or malicious nodes can be used by the attacker to launch different types of attack against WSNs.

Because of the scarcity of the resources of sensor nodes, WSN routing protocols rarely incorporate trust or security mechanisms. An non-trusted protocol is itself a security threat to the WSN. Traditional security mechanisms that are developed for wired networks are not applicable for wireless sensor networks. First, some attacks are more prominent in wireless networks, such as the wormhole attack and the RF signal distortion. Second, some security mechanisms, such as asymmetric key encryption, are expensive to implement in the limited resources of the sensor nodes.

Even the security systems that are developed for wireless ad hoc networks may not be directly applicable to WSNs because of the limited capabilities of the sensor nodes and the different operations of both types of networks. Although WSNs share the property of no infrastructure with wireless ad hoc networks, adopting ad hoc routing protocols in WSNs may not be a feasible solution due to the inherent characteristics of WSNs.

Thus, new security systems are needed to protect WSNs from targeted attacks. Several research studies have focused on the security challenges of WSNs. Many of these studies have considered the attacks against the routing protocols in terms of the effect on the availability of data in the network. However, research efforts on these attacks tend to take a general approach and do not focus on specific WSN routing protocols. LIDeA [5] and LAID [6] are two general architectures for Intrusion Detection Systems (IDSs) in WSNs. Both architectures focus on the outcome of the attack and not on the frailties of the routing protocols. There are several WSN routing protocols that differ in their functions according to the network structure or the network operation. These differences allow the attackers to launch specific attacks against the WSN routing protocols.

1.2 Problem Statement

One common categorization of routing protocols in WSNs distinguishes shortest path routing protocols from link quality routing protocols. Shortest path or minimum hop routing protocols construct a routing tree with the shortest paths between the sensor nodes and the base station. These routes are not usually the best because the shortest path routing protocols do not consider other factors along the paths, such as route congestion or signal strength.

In link quality routing protocols, sensor nodes compute a quality metric for their links and use it as the routing metric for relaying data. Link qualities can be end-to-end delay, signal strength, available bandwidth, etc. This category of routing protocols may require cooperation between the sensor nodes to compute the best routes to the base station. Sensor nodes share information without any guarantee of the validity of this information. For example, in MintRoute [7, 8], each node exchanges its expected packet reception ratio with its neighbours. Then each node computes an estimate for the packet reception ratio to the base station through each neighbour individually. Each node chooses the route to the base station with the highest expected packet reception ratio. A malicious node can exploit this feature and manipulate the link qualities to lure good sensor nodes.

The aim of a malicious node is to attract as much traffic as possible from its surrounding neighbours. It may then launch another attack, such as a blackhole or selective forwarding attack [9], a sinkhole attack [10], or a wormhole attack [11, 12]. The strength of exploiting the link quality routing protocols stems from its transparency. The malicious node neither performs extra communication nor requires additional hardware. Neither encryption nor authentication can prevent this attack because the malicious node may be an existing node that has been compromised. In this case, the malicious node has legitimate keys to communicate with the other sensor nodes. Accordingly, other defence

mechanisms are needed to protect the WSN from such attacks.

1.3 Objective

The objective of this dissertation is to detect a malicious node that exploits the link quality routing protocols of WSNs. This dissertation focuses on:

- analyzing the different types of link quality routing protocols;
- investigating the possible vulnerabilities of each type of link quality routing protocols; and
- proposing and evaluating appropriate defence mechanisms.

This dissertation addresses the following questions:

- Can a security mechanism be implemented in WSNs without the help of cryptography?

  It is known that cryptography requires complex mathematical operation and powerful resources. Both are hard to achieve in the resource-limited sensor nodes. Securing WSNs without cryptography will be an advantage.

- Can the sensor nodes work independently to detect malicious nodes?

  Independent detection neither requires exchanging information among the sensor nodes nor sending it to the base station. Thus, it helps to eliminate extra traffic in the network. Therefore, less power consumption is required.

1.4 Contributions

The major contributions of this dissertation are:

- A comprehensive study of the two types of link quality routing protocols is presented.
- Existing vulnerabilities in the two types of protocols are identified.
- Two IDSs are proposed to detect malicious nodes that exploit the discovered vulnerabilities.
- A simulator for a protocol of each type is built to evaluate the proposed IDSs.

This dissertation adopts analysis and simulation as the research methodologies. Two simulators are built to simulate a malicious node that exploits the identified vulnerabilities of each type of link quality routing protocols. In addition, two IDSs are proposed and simulated to detect the malicious node. The success of the IDSs is measured by the percentage of data delivery at the base station before and after detection. A second metric is the number of true detections versus the number of false detections.

1.5 Thesis Organization

This thesis is organized as follows.

Chapter 2 provides a thorough background survey on WSNs. It gives an emphasis on the routing protocols in WSNs, their classification, and metrics. Security threats to WSNs are explained and then a survey of previous works that propose defence mechanisms against routing attacks ends the Chapter.

Chapter 3 focuses on the link quality routing protocols and their importance in WSNs. The Chapter gives an in depth analysis of the common link quality estimators that are used in link quality routing protocols. The adopted research methodologies used in this dissertation are explained next. The Chapter ends by explaining the simulation framework.

Chapter 4 explains the details of MintRoute. An explanation of the components of MintRoute and how they integrate together is provided. An explanation of how a malicious sensor node can violate the operations of MintRoute to attract network traffic to its favour is provided next. The proposed IDS and how it works transparently in a WSN using MintRoute is explained. Finally, the Chapter explains possible scenarios of a malicious node behaviour and how the proposed IDS is adapted to detect it.

Chapter 5 presents the simulation results of detecting MintRoute vulnerability. The Chapter begins with explaining the simulation setup and the configurations of the WSNs. The measurement metrics used to evaluate the proposed IDS are explained and finally, the outputs of the simulation are presented and discussed.

Chapter 6 focuses on the details of Collection Tree Protocol (CTP) and its vulnerabilities. An explanation of the components of CTP and how they integrate begins the Chapter. The vulnerabilities that a malicious node can exploit are highlighted. The Chapter ends with a proposed IDS to detect the vulnerabilities of CTP.

Chapter 7 presents the results of simulating CTP and its proposed IDS in Network Simulator-2 (ns-2). The configuration of the simulation environment is provided. The simulation results show that the proposed IDS detects the malicious nodes effectively.

Finally, Chapter 8 concludes the thesis and suggests future work.

Chapter 2

BACKGROUND AND RELATED WORK

2.1 Introduction

This Chapter gives an overview of the world of WSNs. It begins by giving a thorough background on WSN structures, advantages, and applications. Then the Chapter emphasizes the role of routing protocols in WSNs, their classifications, and their performance and cost metrics. The security threats that target WSNs are explained next. The Chapter ends with explaining how to secure WSNs using three main security areas: key distribution, cryptography, and intrusion detection.

2.2 Wireless Sensor Networks

A WSN contains dozens of wireless sensor nodes deployed randomly in an unattended environment to collect data. They can be deployed on the ground, in the air, in vehicles, or inside buildings. These sensor nodes are limited in resources, which mean that they cannot process the collected data or store it for long periods of time. Therefore, the wireless sensor nodes forward their collected data to a base station that is capable of processing and storing the collected data. The base station is usually far from the source of the data.

The base station collects the sensed data from the sensor nodes, and then it processes, analyzes, and stores the data. The data is made available to the end-user through connecting the base station to an enterprise network or the Internet. A WSN can have a single or multiple base stations with mobile or fixed scenarios [3]. In a mobile scenario, the sensor nodes sense and save data from their surroundings, while a base station roams

the vicinity of the WSN to collect the saved data. Fixed scenarios are more common in WSNs where the sensor nodes sense and forward their sensed data to the base station using a wireless multihop infrastructureless architecture [2]. To relay their data, the sensor nodes form a spanning tree rooted at the base station that connects all sensor nodes. For example, Figure 2.1 shows a hypothetical WSN installed in different buildings at the University of Calgary campus to monitor, for example, temperature levels.

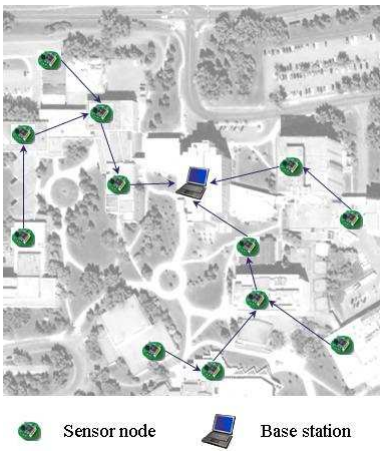


Figure 2.1: A routing tree rooted at the base station

A sensor node typically contains: a sensing chip to sense environmental or physical parameters, a microcontroller to process data and perform networking operations, a radio transceiver to send and receive data, and a power source to feed all other components. Figure 2.2 shows the typical components of a sensor node. The sensing chip has one or more sensors to sense the environmental parameters, such as temperature, pressure, or light intensity, or to sense physical parameters, such as blood pressure. The microcontroller provides the computational capabilities to the sensor node and it has access to a memory component. The radio transceiver provides the wireless communication capability and it has access to a small or an embedded antenna. The power source could be a number of AA batteries or solar panels. There could be other components in the sensor nodes, such as a Global Positioning System (GPS) circuit for localization purposes

[13, 14]. A number of sensor nodes has been developed in recent years including Imote2 [15], Iris [15], Mica2 [15], MicaZ [15], Telos [15], Cricket [16], and SunSPOT [17]. Although the sensor nodes communicate together using wireless communication, they may interface with wired networks with Ethernet, WiFi, USB, or serial ports. This interfacing enables programming the sensor nodes or gathering information from them [2].

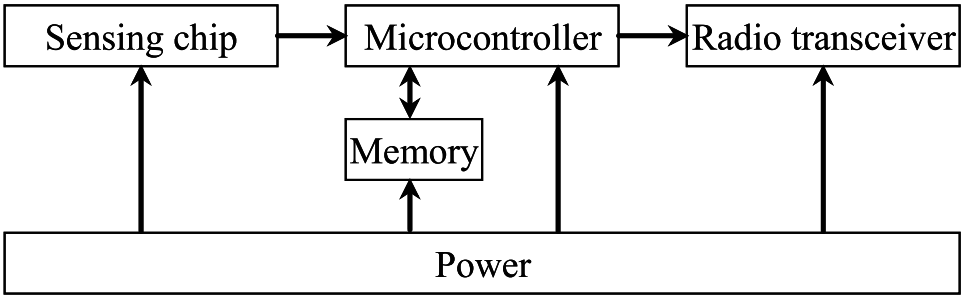


Figure 2.2: Main components of a sensor node

To build a practical WSN on a large scale in an unattended environment, the recent technological advances enable to have wireless sensor nodes that have the following features: small in size to achieve portability; low in cost to achieve feasibility; and low in consuming battery power to achieve prolonged lifetime [13]. However, the design of the sensor nodes put multiple challenges for developing software applications for WSNs [18]:

- *Limited resources*: The sensor nodes have limited processing, storage, and communication resources. In addition, the power supply is limited and the communication bandwidth is scarce.

- *Limited support for networking*: A WSN is like a peer-to-peer network, where each sensor node is both a router and an application host. Thus, the routing protocols are not generic and they are application centric.

- *Limited support for software development*: Given the limited capabilities of the sensor nodes, the developed software for the WSNs has to be simple, efficient, and

lightweight. Moreover, only software related to the application of the WSN should be installed on the sensor nodes.

2.2.1 WSNs and Ad hoc Networks

Ad hoc networks are a special type of wireless networks that have no infrastructure. Thus, the nodes of an ad hoc network must provide for services such as addressing, routing, and more [19]. Although one may think of a WSN as a type of ad hoc networks, they have some differences:

- The number of sensor nodes in a WSN can be orders of magnitude larger than the number of nodes in an ad hoc network [13].
- The sensor nodes are prone to failure because of their limited capabilities and low cost. However, the nodes in an ad hoc network, such as laptops, have more resources and stronger capabilities [13].
- Many of the applications of WSNs do not require mobility. Whereas, ad hoc networks will form a Mobile Ad hoc Network (MANET) if the nodes are mobile. The mobility of MANETs requires the routing protocols to be adaptable to mobility [13].
- WSNs can be data centric, which means that data is requested based on specific criteria. For example, the base station can query the sensor nodes to send their temperature reading if it is greater than 25°C [20].
- Sensor nodes in a WSN may aggregate data from adjacent sensor nodes since the data may have similar values [20].
- WSNs are application-specific, which affects the requirements of each WSN. Some applications may require mobility, others may require data aggregation, and others

may require random deployment [20].

Although both ad hoc networks and WSNs allow multihop communication, there are several distinctions between their communication patterns. Ad hoc networks allow the communication of any pair of nodes, whereas the communications in WSNs fall into three categories [11, 21]:

- *Many-to-one*: sensor nodes send their data to the base station communication for further processing and analysis. Also, they can send the data to cluster heads, where the cluster heads aggregate the data.
- *One-to-many*: base station sends queries or reprogramming tasks to all sensor nodes.
- *One-to-one*: base station sends specific requests to a sensor node. Also, a sensor node relays data to another sensor node.

Thus, the routing protocols of ad hoc networks are not directly applicable in WSNs and more specialized routing protocols are needed.

2.2.2 Operating Systems for WSNs

An operating system is software that resides between the hardware and the software applications. It provides basic programming abstractions to enable the applications to interact with hardware resources, schedule tasks, and resolve conflicts between contending applications that try to seize resources. Other duties of an operating system include memory management, file management, power management, and networking [22].

Like other computational devices, the sensor nodes need an operating system to function. However, the limited resources of the sensor nodes and the requirements of the applications for the WSNs put constraints on the capabilities of the operating systems

for WSNs. Researchers at UC Berkeley have led the development of operating systems for WSNs by developing Tiny Operating System (TinyOS) [23].

TinyOS is an event-driven operating system developed for sensor nodes that have limited resources. It has a small memory footprint and excellent power management. TinyOS is open-source software written in Network Embedded Systems C (nesC) [24, 25]. nesC is a dialect of the C programming language that supports event-driven programming. TinyOS meets the limited resource capabilities of the sensor nodes by excluding some of the common features of large operating systems, such as multithreading. These exclusions have led to the development of other operating systems to meet these capabilities, such as MANTIS [26], Contiki [27], SOS [28], and LA-TinyOS [29]. However, TinyOS has become the de facto industry standard operating system for WSNs [14].

The primary goal of MultimodAl system for NeTworks of In-situ wireless Sensors (MANTIS) is ease of use, while adhering to the limited resources of the sensor nodes. To achieve its goal, MANTIS is written entirely in standard C, which gives the programmers the features of large operating systems, such as multithreading, pre-emptive scheduling, and a standard network stack. Contiki is another event-driven operating system for WSNs that supports dynamic loading and replacement of programs and services. Unlike TinyOS, Contiki supports multithreaded applications. SOS takes a more dynamic approach to the design than TinyOS. In TinyOS, the sensor nodes run a single statically-linked image of the operating system. This makes it hard to run multiple applications or update them. SOS consists of dynamic modules that can be loaded or unloaded during run-time. LA-TinyOS extends TinyOS to support temporal and spatial locality for better event detection and lower energy consumption. It provides kernel-level support for the development of locality-aware tasks.

MagentOS [30] is another operating system for WSNs that employs a distributed model approach. MagentOS treats the whole WSN as a single computational device.

It partitions the application into small components and dynamically places them on the sensor nodes within a WSN. This partitioning reduces energy consumption, avoids hotspots, and increases the lifetime of the sensor nodes.

Resilient, Expandable, and Threaded Operating System (RETOS) [31] is another multithreaded operating system for WSNs. RETOS provides system resiliency, kernel extensibility, and dynamic configuration.

2.2.3 Protocols for WSNs

Like other types of networks, WSNs need layers of protocols to organize the functionalities of the sensor nodes. The protocol stack of the sensor nodes is based on the Open Systems Interconnection (OSI) reference model [32]. It comprises five layers out of the seven layers of the OSI reference model. Figure 2.3 shows the five layers of the WSN protocol stack.

The physical layer is responsible for the modulation, transmission, and reception mechanisms. These mechanisms help the physical layer to convert meaningful data into wireless signals and vice versa [2, 13].

The data link layer ensures reliable communication and provides error detection mechanisms. It also manages channel access to minimize packet collisions. These issues are resolved for the immediate neighbouring sensor nodes [2, 13].

The routing layer is responsible for establishing routes to the base station that satisfy the criteria of choosing a route. These criteria can be low energy consumption, or low delay, or a combination of desirable features. Once routes are established to the base station, sensed data is relayed on a multihop communication basis. The routing layer is responsible for maintaining alternative routes in case the primary route becomes unavailable [13].

The main goals of the transport layer are to achieve end-to-end reliable data transmission and to reduce network congestion. Since the goals of the transport layer are

fulfilled by the two end points of communication, the transport layer exists in the base station and it functions on the sender sensor node only [13].

Finally, the application layer contains the software application of the WSN. It defines the display format for the gathered data from the sensor nodes. The application layer exists in the base station only. The sensor nodes relay data only, and do not display or process it for the end-user [13].

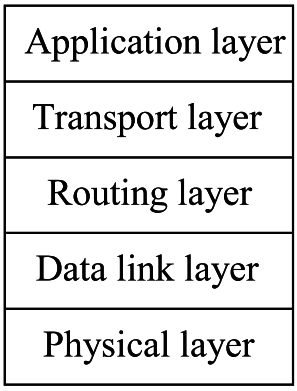


Figure 2.3: WSN protocol stack

2.2.4 Advantages of WSNs

Despite the limited resources and the challenges of deploying WSNs, their deployment comes with several advantages over wired networks:

- *Coverage of a large area*: Deploying a large number of sensor nodes can be helpful to cover a large area that will be costly to cover using traditional networks [20].
- *Ease of deployment*: WSNs can be deployed without any prior organization. As soon as the sensor nodes are deployed, they self-organize to form a WSN. This gives flexibility to deployment and reduces the cost and time of deployment [20].
- *Ease of scalability*: It is easy to deploy new sensor nodes in a WSN to increase the coverage area or to replace sensor nodes [33].

- *Fault tolerance*: Because of the dense deployment of sensor nodes, WSNs can operate reliably, even with the failure of some sensor nodes because multiple sensor nodes monitor the same area [20, 33].

- *Higher level of confidence*: Because an event may be detected and collected by more than one sensor node, the base station will have more confidence in the data received from the sensor nodes [33].

- *Improved sensing quality*: Local sensor nodes can collaborate to send a summary to the base station rather than sending all the data [33].

- *Less human interference*: Since human access to some environments may be difficult or undesirable, the sensor nodes can be scattered from an airplane in hostile environments. Also, due to their small size, sensor nodes have a small footprint on the environment [33].

- *Less total energy consumption*: A multihop WSN saves more power than a single-hop WSN. However, there is a tradeoff between the number of sensor nodes and total energy consumption. An excess number of sensor nodes may lead to more total energy consumption than a single-hop WSN [18].

2.2.5 Applications of WSNs

WSNs applications collect data from the sensor nodes in three reporting modes: *event-driven*, *periodic*, and *on-demand reporting*. In the *event-driven reporting* mode, the sensor nodes will sense and report data if an event occurs in their surrounding environment. For example, if a WSN is deployed in a forest to monitor forest fires, then the sensor nodes will report back to the base station if a fire occurs in the forest. Event-driven reporting is characterized by its real-time nature. Thus, the data should reach the base station as quickly as possible [3].

Unlike event-driven reporting, *periodic reporting* is used when the data is not urgent. The sensor nodes report sensed data to the base station at predefined periods of time. Further, in periodic reporting all the sensor nodes send their data to the base station, whereas in event-driven reporting, only the sensor nodes in the vicinity of the event report it to the base station. An example of periodic reporting application is habitat monitoring. A sensor node attached to a bear reports back to the base station at predefined periods of time to locate the bear [3].

On-demand reporting applications send requests to the sensor nodes to send their sensed data. For example, an application that monitors chemical pollutants in water may send a request to the sensor nodes to sense the pollutants and report the values to the base station when required [3].

Besides the three reporting modes, the sensor nodes apply one of two operations on data: *data gathering*, and *data aggregation*. *Data gathering* means that all sensor nodes send their data to the base station without any changes along the route. This data can be redundant, correlated, or inconsistent. *Data aggregation* combines data from different sensor nodes to eliminate redundancy and to minimize the number of transmissions [3]. In data aggregation, some sensors are designated as cluster heads. A cluster head sensor node gathers the data from its cluster, aggregates it, and sends a single value to the base station.

WSNs have been deployed in several fields. They can be used to monitor traffic, habitat, or chemical pollutants. They can be used in protecting the infrastructure, such as oil facilities, power grids, and water distribution facilities. Other fields such as health care systems, military, smart homes, and warehouse management also benefit from the deployment of WSNs [3, 13]. The following are some real-life implementations of WSNs.

WSNs in Health Care

Sensor nodes are attached to patients to monitor and analyze data, such as pulse and blood pressure. In emergency cases, the sensor nodes send alarm messages to a nearby base station. To minimize the probability of lost alarms and to speed the reporting time, multiple base stations could be installed in the premises so the sensor nodes can communicate with the nearest base station in a single hop [3].

The CodeBlue project [34] attaches sensor nodes to patients to monitor pulse rate, blood oxygen saturation, patient movements, and muscular activities. The miTag platform [35] includes pulse oximetry, blood pressure, and temperature sensors to monitor and assess vital signs of casualties.

WSNs in Habitat Monitoring

Human presence in the habitats of animals, birds, or plants can have undesired impacts. WSNs can help researchers in life sciences to monitor such habitats with minimum presence of humans. A WSN has been used to monitor the behaviour of petrels in the breeding season in Great Duck Island. The researchers have installed the WSN before the petrels migrated to the island for breeding. The WSN has been used to describe the usage pattern of the nesting burrows and the changes in the burrows and the surrounding environment during the breeding season [36].

ZebraNet [37] is a WSN that is used to track the pattern of Zebra movement, their interaction, and the impact of human development. Another WSN is provided in [38] to monitor and control the farms of swift birds.

WSNs in Environmental Monitoring

In the field of geology, WSNs can be used in dangerous environments. WSNs can be used to monitor the interior structure of a volcano or to differentiate true volcanic eruptions from other sources of noise. For example, a WSN has been used to monitor the eruptions

of volcanos in central Ecuador [39, 40].

The Columbia River Ecosystem (CORIE) [41] is a WSN deployed at the estuary of the Columbia River to measure water velocity, temperature, salinity, and depth. WSNs can also be used to warn against floods as explained in [42].

WSNs in Military

One of the tasks in military applications is to detect attacks by chemical or biological weapons before proceeding to a conflict zone. Such detection will help military personnel to plan for evacuation procedures or to avoid contaminated areas. An airplane can spread chemical sensor nodes in the suspected areas via the air. Once the sensor nodes reach the ground they form a WSN and start sensing the required parameters and send the readings to the base station [43].

Other military applications include the Boomerang sniper detection system [44] and VigilNet [45]. Boomerang uses acoustic sensor nodes to detect the sound of firing of snipers. VigilNet is a surveillance WSN for target tracking in harsh environments.

WSNs in Commercial Applications

The complex physical systems in airplanes can benefit from the usage of WSNs. A WSN can be installed in an airplane to monitor, for example, the status of the engines. The sensor nodes can sense various physical parameters, such as oil level and temperature. The values of these parameters can be transmitted to a base station in the cockpit or on the ground for further analysis. This will provide the maintenance crew with immediate diagnosis of the airplane [46].

WSNs have been used in civil engineering to monitor the health of structures by tracking the spatio-temporal patterns of vibrations [47]. The Nonintrusive Autonomous Water Monitoring System (NAWMS) [48] is developed using WSNs to detect water usage in houses.

2.2.6 Definitions of WSNs

WSNs are an interdisciplinary research area that draws contributions from several fields, such as networking and signal processing. In the following, the definitions of the key terms that are used throughout the dissertation are given.

- *Sensor*: It is a transducer that converts a physical phenomenon, such as temperature, sound, or motion into a digital form that can be further processed by software applications. There are different types of sensors that measure different phenomena. For example, thermal sensors measure temperature values, electrochemical sensors sense chemical components, and acoustic sensors sense sound waves [18, 49, 22].

- *Sensor node*: It is the main component of WSNs. It encompasses one or more sensors, a processor, a memory chip, a power supply, and a wireless modem. These components are called resources and they are mounted on one board [18].

- *Base station*: It is the central controller of a WSN. It receives the sensed data from the sensor nodes for analysis and processing. It may send queries to the sensor nodes to request data. The base station provides an interface for the end-user to deal with the WSN. It has more resources and more powerful processing capabilities than the sensor nodes. The base station is also called a sink [19].

- *Route*: It is the sequence of links that a packet traverses to reach the base station from a sensor node. The terms route and path are used interchangeably throughout the dissertation [19].

- *Routing*: It is the process by which the sensor nodes construct routes to the base station. The main goal of the routing process is to establish low-cost routes to the base station. The cost of a route is the sum of the costs of all links along that route. The routing process is performed by a routing protocol [32].

- *Cost metric*: It is the metric by which the routing protocol chooses the low-cost route to the base station. There is a number of route costs that the routing protocol needs to satisfy, such as shortest path, least packet loss, or available power.

- *Routing tree*: The routing tree represents the routes from the sensor nodes to the base station with one route only for each sensor node. In WSNs, the base station is always the root of the routing tree, i.e., the final destination of any route in the network.

- *Subtree*: It is a portion of the routing tree where all sensor nodes have their routes passing through a common sensor node. The subtree is identified by that common sensor node.

- *Beacon*: It is a packet sent by a sensor node to announce its presence and update neighbouring sensor nodes with its routing information. It is also called a route update.

- *Neighbour*: It is a sensor node that is within the communication range of another sensor node. Both sensor nodes communicate directly using a wireless link.

- *Next-hop*: It is the neighbouring sensor node chosen with respect to a routing protocol to relay the data to the base station. The next-hop is also called the *parent sensor node*. The sender of the data is called the *child sensor node*.

- *Forwarding*: It is the process of sending data from the child sensor node to the parent sensor node.

- *In-network*: It is a type of processing that occurs in the sensor nodes. It means that the sensor nodes do some processing on the data before sending it to the base station [18]. For example, a sensor node may collect some data and send the average only to the base station.

- *Unicast*: It is a type of transmission where packets are sent to one destination [22].
- *Broadcast*: It is a type of transmission where packets sent by a sensor node are received by all its neighbours [22].
- *Multihop*: It is a type of transmission where packets traverse multiple links to reach the final destination, which is the base station in case of WSNs.
- *Good sensor node*: It is a sensor node that follows the routing protocol and does not perform any malicious behaviour.
- *Malicious node*: It is an attacker node that compromises the routing protocol to its favour.

2.3 Routing Protocols for WSNs

The sensor nodes play a dual role of collecting data from their surrounding environments and sending it to the base station. Therefore, the sensor nodes need routing protocols to determine how the collected data will reach the base station. As mentioned before, the sensor nodes are limited in resources and computational capabilities. Since WSNs are mostly deployed in unattended environments, increasing their lifetimes becomes essential. As a result, the resource intensive routing protocols of traditional wired and wireless networks do not suit the WSNs for the following reasons [20]:

- WSNs suffer from high bit error rates, whereas traditional routing protocols assume highly reliable connections [13].
- WSNs are designed to be deployed randomly with no infrastructure. Thus, the sensor nodes should be self-organizing and cope with the resultant nodal distribution.

- WSNs are application specific, which means that their designs differ for each application. A WSN to monitor vehicular traffic has a different design from another WSN for weather monitoring.

- WSNs are stationary in most scenarios. Thus, there are no frequent topological changes. Even if mobility is permitted in a WSN, it is very limited.

- WSNs may apply data aggregation and in-network processing to minimize data transmission.

- WSNs are data-centric, which means data may be requested based on specific attributes. In attribute-based addressing, the sensor nodes are addressed by an attribute-value pair rather than by their unique Identifiers (IDs). For example, if the base station initiates a query for temperature $> 25°C$, then only sensor nodes with temperature readings above $25°C$ will report back their readings.

- WSN applications may be more interested in knowing the location of the sensor nodes rather than their IDs to determine the location of reported phenomena. A GPS may not be feasible for the resource-constrained devices, thus the sensor nodes use other methods to approximate their locations, such as triangulation [50].

The routing protocols for WSNs typically begin constructing routes to the base station by neighbour discovery. Sensor nodes send out beacons to announce their presence. Upon receiving beacons from their neighbours, the sensor nodes build local neighbour tables. These tables often include: each neighbour's ID, delay via that neighbour, location of that neighbour, and an estimate of link quality to that neighbour [13, 51]. Routing protocols specify a cost metric for the purpose of choosing a next hop. For example, next hop may be the neighbour with the lowest delay to deliver data to the base station or the closest neighbour to the base station. Although the routing protocols for WSNs look

simple, some challenges face their design [20, 52]:

- *Communication range*: Routing protocols should support multihop communication to overcome the limitation of short wireless ranges. Therefore, the path from a sensor node to the base station will likely contain multiple wireless hops.

- *Connectivity*: Routing protocols should handle the high connectivity resulting from the dense deployment of sensor nodes efficiently.

- *Control overhead*: Routing protocols may use control packets to maintain the routes between the sensor nodes and the base station. As the node density increases, the transmission of control packets increases in the wireless medium, leading to more latency and energy consumption. As a result, tradeoffs between conserving energy, latency, and maintaining routes may exist.

- *Energy consumption*: Routing protocols should be energy-efficient to conserve the battery power of the sensor nodes. For example, beacons or route updates should be small and infrequent to minimize the number and duration of transmissions.

- *Fault tolerance*: Routing protocols should be adaptive to failures that occur due to power depletion, physical damage, or environmental interference. They should be able to find new routes to the base station or adjust transmission powers to reduce energy consumption.

- *Limited resources*: Routing protocols should be lightweight and simple to meet the limited resources and capabilities of the sensor nodes.

- *Quality of Service (QoS)*: Routing protocols should be aware of QoS requirements. For example, in some health applications, it is critical to deliver the information about the patients as quickly as possible to the base station.

- *Random deployment*: Routing protocols should be able to deal with the randomness and nodal distribution of random deployment.

- *Scalability*: Routing protocols should be scalable to embrace new sensor nodes that replace failed sensor nodes or to embrace new ones to expand the WSN.

- *Security*: Routing protocols should balance between the security level and the energy consumption. Security is important in wireless communication to ensure data availability, confidentiality, and integrity.

- *Transmission media*: Routing protocols should consider wireless channel problems, such as fading and interference, and the scarce bandwidth. The available bandwidth for WSNs is low in the order of 1-100 kb/s.

2.3.1 Classification of WSN Routing Protocols

Many classifications have been proposed for the routing protocols of WSNs. Al-Karaki and Kamal [1] have proposed a general classification that is widely recognized. Other classifications tend to classify the routing protocols according to a specific attribute. For example, Acs and Buttyan [53] classify the routing protocols for WSNs according to how the next hop is chosen. Lotf and Ghazani [54] classify the routing protocols according to the knowledge of the sensor nodes about the network topology. We explain the details of the three classifications next.

According to the classification of Al-Karaki and Kamal [1], the routing protocols are classified according to the **network structure**, the **protocol operation**, or **how the source finds the destination**. Each of these classes are further divided into subclasses. Figure 2.4 shows the three classes and their subclasses.

The **network structure** routing protocols are further classified into *flat routing protocols*, *hierarchical routing protocols*, and *location-based routing protocols*. In *flat routing*

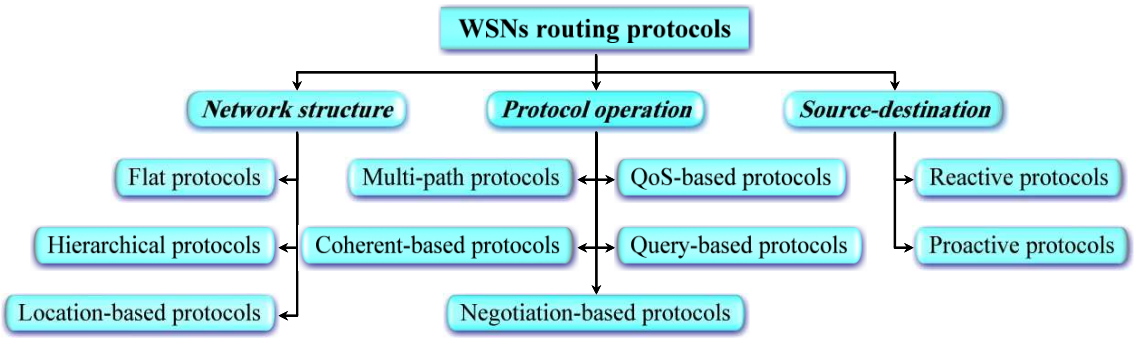


Figure 2.4: Classification 1 of WSNs routing protocols

protocols, all the sensor nodes play the same role in the network, such as data collection. Sensor Protocols for Information via Negotiation (SPIN) [55], Directed Diffusion [56], Energy Aware Routing [57], and Minimal Cost Forwarding Algorithm (MCFA) [58] are examples of flat routing protocols. In *hierarchical routing protocols*, the sensor nodes play different roles in the network, such as data collection and data aggregation. The sensor nodes form clusters with some nodes acting as cluster heads. Each cluster head aggregates the collected data from the sensor nodes in its cluster and sends a single report to the base station. Cluster heads can be more capable nodes or regular sensor nodes where they rotate roles periodically. Low Energy Adaptive Clustering Hierarchy (LEACH) [59] is one of the early hierarchical routing protocols for WSNs. Finally, in *location-based routing protocols*, the positions of the sensor nodes are considered when choosing the routes to the base station. Sensor nodes can estimate distances between them using, for example, signal strength. For example, Geographical and Energy Aware Routing (GEAR) [60] is a location-based routing protocol for WSNs.

The **protocol operation** class is divided into *multi-path routing protocols*, *QoS-based routing protocols*, *coherent-based routing protocols*, *query-based routing protocols*, and *negotiation-based routing protocols*. For fault tolerance purposes, *multi-path routing protocols* construct multiple paths between each sensor node and the base station. If the primary path becomes unavailable, then the sensor nodes can switch quickly to another

path. Directed Diffusion is a multi-path routing protocol for WSNs. The *QoS-based routing protocols* have to satisfy certain metrics when forwarding data, such as delay, or energy consumption. SPEED [61] ensures a certain speed to deliver collected data to the base station. In *coherent-based routing protocols*, the sensor nodes cooperate to process the forwarded data in the network. For example, since multiple sensor nodes sense the same phenomenon, many will have the same or similar readings. As a result, the sensor nodes can reduce the amount of traffic in the WSN by testing the ambient traffic to drop duplicate readings. Multiple Winner algorithm (MWE) [62] is an example of coherent-based routing protocols for WSNs. In a network that uses *query-based routing protocols*, such as Directed Diffusion, the sensor nodes do not send the collected data to the base station but they rather wait for the base station to request the data. The base station may request data from a special region of interest or it may request data with specific attributes, using attribute-based addressing. Instead of flooding the WSN with redundant traffic, sensor nodes using *negotiation-based routing protocols* exchange a series of negotiation messages to suppress the duplicate copies of data from being forwarded. SPIN provides a family of protocols that are negotiation-based.

Finally, the **source-destination** class is divided into *proactive routing protocols* and *reactive routing protocols* according to how the source finds a route to the destination. The *reactive routing protocols* are on-demand, which means that routes are constructed only if they are needed. TinyAODV [63] is an example of reactive routing protocols. On the other hand, *proactive routing protocols*, such as CTP [64], establish routes between the sensor nodes and the base station before the routes are needed, even if they are never used. A routing protocol may adopt a hybrid approach of both subclasses.

According to this classification, routing protocols for WSNs fall in a subclass in each of the three classes. For example, a routing protocol may be flat, multi-path, and proactive.

The classification of Acs and Buttyan [53] is based on the way the next hop is selected

to relay the data packets to the base station. This classification divides the routing protocols into five classes as shown in Figure 2.5.

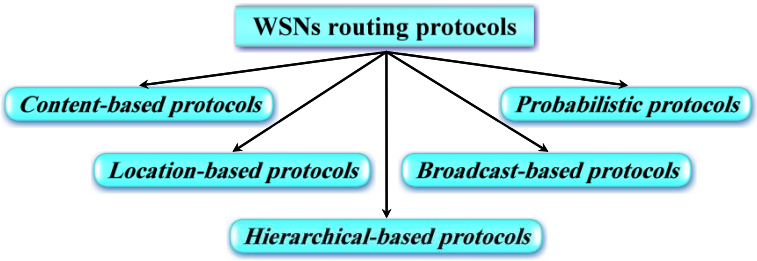


Figure 2.5: Classification 2 of WSNs routing protocols

Content-based routing protocols, such as Directed Diffusion, select routes to the base station based on the content of the queries sent by the base station to gather data. For example, only the sensor nodes in a region of interest establish routes to the base station.

Location-based routing protocols choose next hops based on their locations relative to the base station. If the sensor nodes do not know their locations, then they can use computational methods, such as triangulation, to estimate their physical locations. These methods add computation overhead on the routing protocol. GEAR is an example of these protocols.

Hierarchical-based routing protocols aim to reduce the communication overhead and to conserve power. Sensor nodes send their sensed data to other nodes in a higher level. These higher level nodes, cluster heads, aggregate the sensed data and send the result to the base station. Hierarchical-based routing protocols are mainly based on LEACH.

The sensor nodes implementing a **broadcast-based routing protocol** decide individually whether or not to forward a packet. If a sensor node decides to forward a packet, then it will simply broadcast it. Otherwise, it will drop the packet. For example to drop a packet, a sensor node compares its cost with the cost of the sender that is embedded

in the packet. MCFA is a representative routing protocol of this class.

Probabilistic routing protocols aim to load-balance the selection of next hops and to increase the robustness of the routes. For example, they can choose routes that have power levels above a certain threshold to prevent power depletion of heavily used sensor nodes. These protocols assume homogenous and randomly deployed sensor nodes. Energy Aware Routing is an example of probabilistic routing protocols.

A recent classification by Lotf and Ghazani [54] classifies WSNs routing protocols based on the knowledge of the sensor nodes about the network topology into **topology aware routing protocols** and **topology unaware routing protocols**. In **topology aware protocols**, the sensor nodes have a global view of the routing tree after the protocol is run. This means that each sensor node has a copy of the complete routing tree. On the other hand, in WSNs implementing **topology unaware protocols**, the sensor nodes have a local view of the routing tree. A local view means knowing neighbouring sensor nodes only. Figure 2.6 shows this classification and its subclasses.

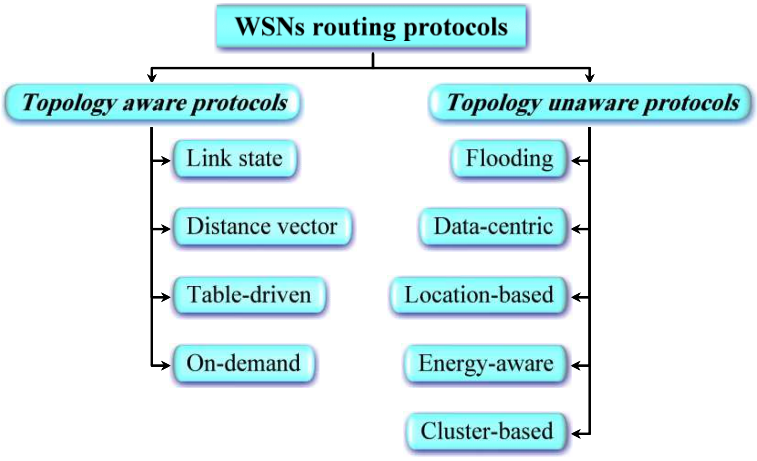


Figure 2.6: Classification 3 of WSNs routing protocols

Topology aware routing protocols are inherited from the routing protocols of wired and traditional wireless networks. These protocols were designed without the constraints of WSNs in mind. As a result, they are not popular in WSNs implementations

and their applicability is limited to small WSNs. This class contains four subclasses: *link state routing protocols*, such as Open Shortest Path First (OSPF) [65], *distance vector routing protocols*, such as Routing Information Protocol (RIP) [66], *table-driven routing protocols*, such as Destination-Sequenced Distance-Vector Routing (DSDV) [67], and *on-demand routing protocols*, such as Ad hoc On-demand Distance-Vector (AODV) [68]. *Link state routing protocols* broadcast local routing information to all nodes in the network, whereas distance vector protocols share network wide information with neighbouring nodes only. Both subclasses use the hop count metric to construct the shortest routes to the root node. *Table-driven routing protocols* and *on-demand routing protocols* are designed for traditional wireless networks. They address the problem of frequent unavailability of links due to the wireless medium or mobility of nodes. *Table-driven or proactive routing protocols* exchange routing tables periodically among the neighbouring nodes. *On-demand or reactive routing protocols* construct routes only when required by the source node. Neither subclass is suitable for WSNs since they are energy inefficient and computationally expensive, and they introduce a large amount of overhead in large networks.

Topology unaware routing protocols are designed mainly for the resource-limited WSNs. This class contains five subclasses: *flooding routing protocols*, *data-centric routing protocols*, *location-based routing protocols*, *energy-aware routing protocols*, and *cluster-based routing protocols*. In *flooding routing protocols*, nodes that receive route update packets or beacons rebroadcast them until a maximum number of hops is reached. *Data-centric routing protocols*, such as SPIN, depend on the base station to request data from the sensor nodes. When the base station initiates a request to collect data, only sensor nodes that have this data reply back to the base station. *Location-based routing protocols*, such as GEAR, are important in two cases. The first case is when the base station is interested in gathering data from a certain location. Then it sends a data query with

the location of interest and sensor nodes in that location reply back. The second case is where the sensor nodes themselves need location information to compute a routing metric, such as energy consumption. The objective of *energy-aware routing protocols*, such as energy aware routing, is to increase the lifetime of the network. Routes are chosen by means of certain probabilities to achieve low-energy consumption along the route. These probabilities use different metrics, such as energy consumed per packet or node's power level. Finally, *cluster-based routing protocols*, such as LEACH, aim to reduce the redundancy in sensed data and minimize the transmissions in the network to conserve the power. The sensor nodes are divided in two levels. Low-level sensor nodes sense data and send it to the high-level sensor nodes. High-level sensor nodes, cluster heads, aggregate the sensed data and send one report to the base station.

2.3.2 Performance Metrics for WSN Routing Protocols

To evaluate the performance of the different routing protocols for WSNs, there are several metrics to that can be used:

- *Data delivery*: This metric measures the total amount of data delivered to the base station compared to the total amount of data sent from the sensor nodes. This metric is also called end-to-end success rate [7].

- *Energy efficiency*: This metric indicates how much the routing protocol consumes energy [69].

- *Fault-tolerance*: This metric indicates how robust the routing protocol is against the failures of sensor nodes. For example, it can be achieved through data replication or having multiple paths to the base station [69].

- *Latency*: This metric computes the delay from the moment data is collected until it reaches the base station [69].

- *Path length*: This metric computes the number of hops from the source sensor node to the base station [70].
- *Path reliability*: This metric is the product of link qualities along the path from each sensor node to the base station. It measures the end-to-end reliability [7].
- *Routing overhead*: This metric estimates the consumed energy to establish a route from a sensor node to the base station [70].
- *Scalability*: This metric measures how scalable a routing protocol is to accommodate new sensor nodes in the WSN [69].
- *Stability*: This metric measures the total number of route changes in the WSN over a period of time [7].

2.3.3 Cost Metrics for WSN Routing Protocols

There are numerous cost metrics that routing protocols use to evaluate routes to the base station. This evaluation is done on a link-by-link basis where a neighbouring sensor node with the best link cost is chosen as the next hop. The cost of a route is the sum of all costs of all the links along that route. A single routing protocol can use one metric or a combination of metrics to evaluate the possible routes to the base station.

Hop count is a basic cost metric to evaluate the routes in a WSN. It is defined as the number of hops or links between one sensor node and another. Two adjacent sensor nodes have a hop count value of one. Hop count is not the best cost metric for evaluating the routes in WSNs because it does not consider the limited resources of the sensor nodes. For example, two adjacent nodes that are distant will have a hop count value of one if they communicate directly. These distant sensor nodes may perform worse than two closer sensor nodes because many retransmissions may be required due to the low

probability of receiving packets. In this case, a multihop route is better than the single hop route [3].

Power-aware Metrics

Using the hop count metric may lead to congestion in the WSN because a sensor node close to the base station will be included in many routes. This congestion leads to the rapid depletion of the battery power of a sensor node because it will transmit more forwarded traffic than other sensor nodes. Power-aware metrics solve this problem by distributing the routes on multiple sensor nodes depending on remaining power of batteries or energy required to transmit data. Power-aware metrics deplete the power of the batteries at the same time. This is more advantageous because a set of new nodes can be deployed at the same time instead of node-by-node deployment.

Reluctance is a power consumption metric that avoids sensor nodes with low remaining battery power. It is measured as the inverse of the remaining battery power at a sensor node. A sensor node with a low battery power level will have a high reluctance value. The best route to the base station is a route with sensor nodes that have a minimum threshold of remaining battery power [3, 71].

Maximum power available is another power metric. Only sensor nodes that have the highest battery power levels, i.e., battery power level, are considered when choosing the routes [72].

Maximum minimum power available prevents the depletion of low battery power sensor nodes if they are on the routes of very high battery power sensor nodes. This metric picks the route with minimum battery power that is larger than the minimum battery power of other routes [72].

Minimum energy is a metric that considers the energy required to transmit packets along the route. This metric chooses routes that require low energy to deliver data to

the base station [72].

Link Quality Metrics

Link quality cost metrics are used to choose the routes that achieve high probability of data delivery to the base station. For example, a route with a high collision rate will have a low probability of delivering data to the base station.

Received Signal Strength Indicator (RSSI), Link Quality Indicator (LQI), and Signal-to-Noise Ratio (SNR) are cost metrics that choose the routes according to signal strength or noise level in the environment [73].

Packet Reception Ratio (PRR) metric [73] estimates link qualities as the ratio of the number of successfully received packets to the number of transmitted packets. PRR chooses routes with high values.

Expected Transmission count (ETX) metric [74] chooses routes that minimize the expected number of transmissions and retransmissions required to deliver data to the base station. Best routes have low ETX values.

Required Number of Packets (RNP) metric [75] estimates link qualities as the ratio between transmitted and retransmitted packets to the number of successfully transmitted packets. Best routes have RNP with low values.

2.3.4 TinyOS Routing Protocols

Many of the proposed routing protocols for WSNs are not implemented. However, the operating systems for WSNs offer some new protocols or adaptations of traditional routing protocols. Four software WSN routing protocols are officially implemented in TinyOS: *AM_ROUTE*, *MintRoute*, *Multihop*, and *CTP*. The four protocols are flat and proactive, which means that all the sensor nodes play the same role in the network. The following is a brief overview of the four protocols.

1. *AM_ROUTE* is the basic protocol that was implemented in TinyOS version 1. The base station periodically starts the routing tree construction process by broadcasting a route update packet with hop count equal to zero. As the sensor nodes receive the route update packet, they rebroadcast it after increasing the hop count by one. Sensor nodes choose their parents as the first neighbour from which a route update packet is heard. Thus, sensor nodes almost always choose the shortest route to the base station. This process is repeated with every route update cycle. The shortest route is not always a reliable route because it may lead to congestion in the WSN [76, 77].

2. *MintRoute* solves the unreliability of AM_ROUTE by introducing link qualities in the route choice. Sensor nodes determine their link qualities with each neighbour according to the number of received and missed packets for the corresponding neighbour. The route to the base station is chosen as the route with the best send and receive qualities. Sensor nodes do not change their routes unless the qualities of the current parent go below certain thresholds [7]. The reliable route may be a long route to the base station. Thus, MintRoute may delay data delivery.

3. *Multihop* is a version of MintRoute that depends on hop counts rather than link quality. Multihop follows the same steps of MintRoute to build the routing tree, but the path to the base station is chosen as the path with the least hop count value. Send and receive qualities are only used as a tiebreaker when there are two paths with the same length [78].

4. *CTP* is a new routing protocol implemented in TinyOS version 2. It combines the properties of MintRoute and Multihop, choosing the shortest reliable path. The sensor nodes build their routes using a routing gradient, which is the expected number of transmissions to reach the base stations. CTP tries to confirm data

delivery by sending acknowledgements as replies to unicast packets [23].

2.4 Security Threats to WSNs

The sensor nodes of a WSN often rely on multihop wireless communication to deliver sensed data to a base station. WSNs are designed to be deployed randomly in unattended and uncontrollable environments. The limited resources, wireless communication, random deployment, and unattended and uncontrollable environments are all vulnerabilities to the security of WSNs. These vulnerabilities can be classified according to the OSI reference model as proposed in [79]. Figure 2.7 depicts this classification.

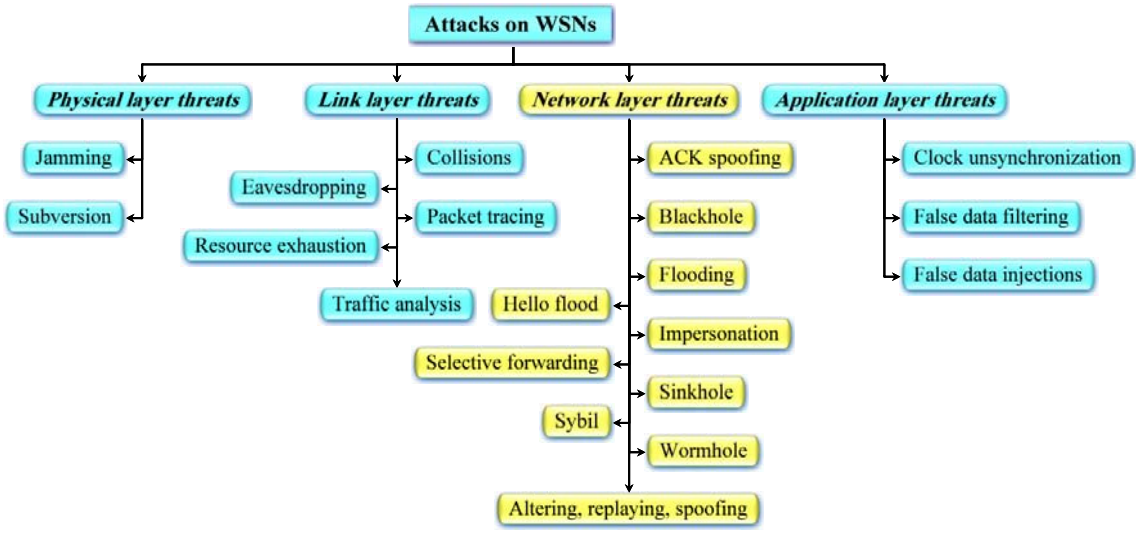


Figure 2.7: Attacks on WSNs

Physical Layer Threats

Due to the broadcasting nature of wireless communication and the non-tamper resistant sensor nodes, the physical layer of WSNs suffer more threats than wired networks.

- *Jamming*: It is a type of attack that interferes with the communication frequency to disrupt the communications in WSNs [12].

- *Subversion*: It is a type of attack in which an attacker tries to get physical access to a sensor node. Once gaining physical access, the attacker can damage it, steal its cryptographic information, or replicate it with an attacker-controlled sensor node [80].

Link Layer Threats

Threats to the link layer of WSNs target the mechanisms that take place in this layer, such as medium access, error control, and frame detection.

- *Collisions*: It is a type of attack in which the attacker causes collisions to control packets, such as Acknowledgement (ACK) packets. This attack may lead to the back-off of certain Medium Access Control (MAC) protocols [79].

- *Eavesdropping*: It is a type of attack in which the attacker monitors the communication between two sensor nodes to gain access to sensitive information [12].

- *Packet tracing*: It is a type of attack in which the attacker determines the location of the source sensor node of an overheard sensor node [81].

- *Resource exhaustion*: It is a type of attack in which the attacker performs Denial-of-Service (DoS) attacks by purposely introducing bogus information to deplete or exhaust the limited resources of the sensor nodes [79].

- *Traffic analysis*: It is a type of attack in which the attacker tries to determine the location of the base station by analyzing the volume of traffic at some sensor nodes. Sensor nodes close to the base station will have more traffic passing through [82].

Network Layer Threats

Threats in the network layer mainly aim to disrupt the routing protocol, for example by modifying the routing tree. Attacks in this category mainly affect the availability of data. This category of attacks is also called routing attacks.

- *ACK spoofing*: It is a type of attack that targets the routing protocols that acknowledge reception of data packets. If an attacker sends an ACK for an overheard data packet, it can convince the sender sensor node that a weak link is strong or a dead link is alive [79].

- *Blackhole and Selective forwarding*: It is a type of attack where the attacker drops all the traffic that passes through it. To hide its existence, the malicious node may launch a selective forwarding attack in which it only drops a portion of the traffic [9, 83, 84].

- *Flooding*: It is a type of attack that targets stateful routing protocols. Stateful routing protocols maintain the state of connections at communicating sensor nodes. An attacker may flood a sensor node with requests to open connections until its memory is exhausted and no more connections from good sensor nodes can be accepted [79].

- *HELLO flood*: It is a type of attack in which a malicious node uses a powerful signal to send or replay neighbour discovery packets. Sensor nodes that receive these packets assume that the malicious node is a neighbour. If the sensor nodes choose the path of the malicious node to forward packets, then their packets will go nowhere [11, 83].

- *Impersonation*: It is a type of attack in which a malicious node tries to convince the sensor nodes in its vicinity that a far sensor node is their neighbour and it has a good route to the base station. If the affected sensor nodes try to send their data to the impersonated node, their data will be lost and will not reach the base station [85].

- *Sinkhole*: It is a type of attack in which a malicious node tries to attract as much

network traffic as possible. The malicious node may then modify the traffic or perform traffic analysis. It may even perform a selective forwarding attack [10, 11, 86].

- *Sybil*: It is a type of attack in which a malicious node claims multiple identities for the same physical sensor node. Thus, the malicious node can induce the sensor nodes to route through the same physical sensor node or it can use the multiple identities to report false data to the base station. The Sybil attack affects fault-tolerance techniques, such as multi-path routing [11, 12, 87].

- *Wormhole*: It is a type of attack in which two colluding nodes create an out-of-band channel between themselves with the aim of transferring the network traffic from one part of the network to another distant part. The wormhole attack is a severe attack because it is not easily detected. The colluding nodes use a communication channel that is different from the one used by the sensor nodes. Moreover, the wormhole attack can be combined with other attacks, such as selective forwarding and sinkholes [11, 88].

- *Altering, replaying, spoofing*: These attacks target the routing information to disrupt the creation of the routing tree. For example, they can create loops, shorten or extend routes, attract or repel network traffic, generate fake error packets, or partition the network [13, 79].

Application Layer Threats

Many of the services that are provided by the application layer are targeted by the attacks in this category.

- *Clock unsynchronization*: It is a type of attack that aims at disrupting the time synchronization of the sensor nodes. This may affect the sleeping schedules of the

sensor nodes [79].

- *False data filtering*: It is a type of attack that targets the clustering applications. In this attack, the attacker resides in an aggregation point. Thus, the attacker can manipulate the aggregation result of the downstream data and also affect the overall computations at the base station [79].

- *False data injections*: It is a type of attack in which the attacker injects false data in the network. Eventually, this false data will reach the base station and it may skew the decisions or the analysis of the base station [79].

The attacks against WSNs can be launched by attackers that fall into three classes according to [11, 13]. An attacker can be identified by one subclass in each of the three classes. These classes are:

- *Mote-class/laptop-class attacker*: A mote-class attacker has access to sensor nodes with the same capabilities as the good sensor nodes. A laptop-class attacker has access to more powerful resources, such as laptops.

- *Insider/outsider attacker*: An insider attacker is a compromised sensor node in the WSN, and it has access to any security mechanisms used in the WSN. An outsider attacker is an implanted sensor node that does not have access to any of the security mechanisms used in the WSN.

- *Active/passive attacker*: An active attacker damages the WSN by actively attacking it. A passive attacker steals the WSN data by passively listening to the communications in the WSN.

2.5 Securing WSNs

Securing WSNs is challenging because they are subject to a wide range of attacks as discussed in Section 2.4. These attacks exploit their wireless communication, their insecure environments, and their limited resources. The resource-constrained sensor nodes preclude the use of resource-intensive security mechanisms. The security goals of WSNs are similar to the goals of other communication networks:

- *Access control*: It means restricting access to resources to good sensor nodes [89].
- *Authentication*: It means ensuring that a sensor node is who it claims to be and that the source of data is a good sensor node [11, 89].
- *Availability*: It means that access to resources and services are available whenever required [11, 79, 89].
- *Confidentiality*: It means protecting communication from unauthorized sensor nodes [79, 89].
- *Integrity*: It means that data packets are delivered without modification [11, 79, 89].
- *Non-repudiation*: It means ensuring that a sensor node does not refute its activities [79].

Countermeasures to security threats to any communication network should have the following requirements and WSNs are no different:

- *Backward secrecy*: It means that a newly added sensor node to a WSN cannot decrypt any previously transmitted secret data packet [89].
- *Efficiency*: It means considering the limited resources of the sensor nodes [89].

- *Forward secrecy*: It means that a removed sensor from a WSN cannot decrypt future secret data packets [89].
- *Freshness*: It means making sure that data is recent and not replayed [79, 89].
- *Scalability*: It means supporting a large number of sensor nodes [89].
- *Self-security*: It means that any additional hardware or software to secure the WSN must be secure itself [79].
- *Survivability*: It means providing a certain level of service in the presence of attacks or node failures [79, 89].

Traditional cryptography achieves many of the aforementioned goals but it is expensive to implement in WSNs. For example, traditional cryptographic algorithms add about 16-32 bytes of overhead to the 30-byte packets of WSNs [90]. In addition, sensor nodes provide 8-16 bits word size, which is insufficient for the arithmetic operations in the cryptographic algorithms [91]. Table 2.1 [91] provides an overview of the popular processor architectures for sensor nodes. It shows how limited these processors are. These challenges limit the choice of traditional cryptographic algorithms to implement. Special cryptographic algorithms that are not resource intensive must be developed for WSN.

Table 2.1: Processor architectures for sensor nodes

Platform	***Word size***	***Clock frequency***	***Cache inst./data***
Atmega103	8 bits	4 MHz	None
Atmega128	8 bits	16 MHz	None
M16C/10	16 bits	16 MHz	None
SA-1110	32 bits	206 MHz	16/8 KB
PXA250	32 bits	200-400 MHz	32/32 KB
UltraSparc2	32 bits	440 MHz	16/16 KB

Moreover, protection mechanisms based on cryptographic techniques are not sufficient because the vulnerability of nodes to capture allows the adversary to access cryptographic

keys; this enables the adversary to duplicate the sensor nodes or insert new ones. In addition, some attacks, such as the wormhole attack, can be launched even in the presence of encryption and/or authentication without the need to know the cryptographic keys. IDSs provide a complementary means of providing protection. An IDS monitors the system and attempts to detect malicious behaviour by searching for attack signatures (misuse detection) or abnormal behaviour (anomaly detection) patterns [84]. The misuse detection is deemed inappropriate for WSN due to the large memory requirement to store the signatures of the attacks.

Many IDSs for WSNs have been developed to detect different types of attacks. However, to the best of our knowledge, most of these IDSs focus on detecting the activities of malicious nodes, rather than the frailties of the routing protocols themselves. Routing attacks against WSNs have gained attention in the literature due to the severity of these attacks.

The countermeasures against the security threats to WSNs can be broadly classified into those that use cryptography and those that do not. For cryptographic systems, key management is an important component.

2.5.1 Key Management

Establishing cryptographic keys is essential for later secure communication in WSNs. The established keys should be resilient to attacks and flexible to update. Due to the limited resources of the sensor nodes, it is not feasible to use traditional key establishment schemes, such as Diffie - Hellman key exchange protocol [92] and key distribution centre [93]. Numerous key management schemes have been proposed for WSNs.

Key management schemes are classified into *static/predistribution schemes* and *dynamic schemes*. In either class, the schemes generate administrative keys that are assigned to the sensor nodes. The sensor nodes use the administrative keys to generate

pair-wise communication keys that are assigned to the communication links. In *static key schemes*, the administrative keys are generated and assigned to sensor nodes prior to their deployment. Static schemes assume that the administrative keys will not be changed once assigned to the sensor nodes. In *dynamic key schemes*, the administrative keys are assigned on demand or on detection of key capture. The advantages of dynamic key schemes over static key schemes are network survivability and scalability. Captured keys are replaced in a timely manner to revoke any communications with the attacker. WSNs can be expanded by adding new sensor nodes and assigning new keys without the risk of increasing the probability of key capture [94].

Two basic schemes of static key management are to assign a global key to all sensor nodes or to assign each sensor node a unique key with each other sensor node in the WSN, pair-wise scheme. The first scheme is vulnerable to the compromise of a single sensor node, and the second suffers from a huge storage requirement.

Eschenauer and Gligor [95] have proposed the first probabilistic key predistribution scheme. Their scheme depends on randomly loading each sensor node with a set of keys from a key pool before deployment. The goal of the scheme is to have each pair of sensor nodes share at least one key. This scheme has become the basis for many other key predistribution schemes. The q-composite scheme by Chan *et al.* [96] is an extension to the scheme of Eschenauer and Gligor. The q-composite scheme requires each pair of sensor nodes to share at least q keys. The virtual key ring by Vu *et al.* [97] is also based on the scheme of Eschenauer and Gligor. The virtual key ring allows sensor nodes that do not share any keys to establish pair-wise keys with the help of other sensor nodes. Most key predistribution schemes assume secure assignment of administrative keys to the sensor nodes. Message-In-a-Bottle by Kuo *et al.* [98] provides a secure mechanism to the initial assignment of administrative keys. It works with any key predistribution scheme as the initial step, but it requires the use of special hardware for the deployment

of keying information.

Many of the dynamic key schemes are based on the Exclusion-Based Systems (EBSs) [99]. EBSs provide secure and efficient rekeying mechanism. Rekeying, which is the process of generating replacement keys, occurs periodically or in the event of node capturing. Replacement keys are encrypted with all the keys unknown to the captured sensor nodes, and distributed to other sensor nodes that know the encryption keys. A drawback of EBSs is that a small number of captured nodes can collude to reveal all the keys in the WSN.

Eltoweissy *et al.* [100] have proposed the first use of EBS for key management in WSNs. Their system assigns the sensor nodes to cells in a virtual coordinate system. The sensor nodes in the same cell are assigned the same EBS key combination where rekeying occurs at the cell level. This system suffers from the collusion problem of EBSs. Younis *et al.* [101] have proposed another EBS system, called SHELL, for clustered WSNs that solves the collusion problem of EBSs. SHELL divides the key generation, assignment, and distribution between the base station and the cluster heads. Eltoweissy *et al.* [94] have presented LOCK as an improvement for SHELL. LOCK uses two layer EBS to perform rekeying to minimize communication overhead. In addition, it uses key polynomials instead of location-based assignment as in SHELL.

Other research efforts have addressed dynamic key schemes that are not based on EBSs, such as the scheme based on identity-based symmetric keying by Jolly *et al.* [102], and SAKE by Seshadri *et al.* [103].

2.5.2 Cryptography

After generating and assigning keys to the sensor nodes, cryptographic mechanisms are used to provide authentication, confidentially, integrity, and non-repudiation of communications in WSNs. Cryptographic mechanisms are divided into *public/asymmetric key*

cryptography and *symmetric key cryptography*.

In *public key cryptography*, each sensor node is assigned a public/private key pair. Public keys are made available to communicate with the owning sensor nodes, whereas the private keys are private to the owners. Anything that is encrypted or authenticated by one key is reversed or checked by its associated key. Public key cryptography mechanisms, such as RSA and Elliptic Curve Cryptography (ECC) [104], are believed to be infeasible in WSNs due to their large code size, intensive computations, and processing time [105, 106]. However, Gura *et al.* [107] have compared the performance of RSA and ECC on sensor nodes of 8-bit Central Processing Unit (CPU). They have concluded that ECC is more suitable for WSNs due to its efficiency with small key sizes, thus reducing processing and communication overhead. For example, ECC with key size of 160 bits provides the same security level as RSA with key size of 1024 bits [105].

Most of the research studies on public key cryptography for WSNs focus on the public key operations and assume that the private key operations are handled by the base station. The private key operations are still expensive to perform on the sensor nodes. Hence, symmetric key cryptography is more popular in WSNs [105].

In *symmetric key cryptography*, any two communicating sensor nodes use a single shared key. The major challenge for symmetric key cryptography is the distribution of shared keys as explained in Section 2.5.1. Ganesan *et al.* [91] have compared five popular symmetric key mechanisms, IDEA [108], RC4 [108], RC5 [109], SHA-1 [110], and MD5 [108, 111], on sensor nodes of CPU sizes 8, 16, and 32 bits. The results of their experiments show uniform cost for execution time and memory size for all sizes of CPUs. However, the hashing mechanisms SHA-1 and MD5 show higher overhead than the encryption mechanisms IDEA, RC4, and RC5.

Early research work on symmetric key cryptography for WSNs include μTESLA by Perrig *at al.* [112]. μTESLA provides asymmetry through delaying the disclosure of

symmetric keys. A sensor node generates a key chain and signs every packet with a key that is secret at the time of transmission. The sensor node broadcasts the secret key at a later time for other sensor nodes to authenticate all packets that are signed with that key. Karlof *et al.* [90] have proposed TinySec, the first fully-implemented symmetric key cryptography for WSNs. TinySec either provides authentication only or authentication and encryption. TinySec is incorporated in the official release of TinyOS. Luk *at al.* [113] have proposed MiniSec to solve the vulnerabilities of impersonation and replay attacks in TinySec.

2.5.3 Intrusion Detection

IDSs provide a second line of defence for WSNs. They complement the security of authentication and cryptography because these two are not enough to secure a WSN. Authentication and cryptography cannot prevent all possible attacks, such as insider attackers [79]. Authentication and cryptography are mainly concerned with the integrity and confidentiality of data and authenticity of sensor nodes, while IDSs are mainly concerned with availability of data and secure routing.

IDSs are classified according to the detection technique: *misuse-based detection*, *anomaly-based detection*, or *specification-based detection* [114]. *Misuse-based or signature-based detection* depends on the knowledge of attack signatures to detect known attacks. It cannot detect new attacks or attacks with unknown signatures. Misuse-based detection is deemed inappropriate for WSNs due to the huge storage requirement to store the signatures. *Anomaly-based detection* establishes profiles of normal behaviour usually by automated training. An anomalous behaviour will be detected if the sensor node deviates from the normal profiles. Although anomaly-based detection succeeds in detecting unknown attacks, it suffers from a high rate of false alarms. *Specification-based detection* is similar to anomaly-based detection but instead of the automated learning, an expert

sets the rules of the operation. If a sensor node violates the predefined rules, it will be declared malicious. Specification-based detection has a low rate of false alarms but it cannot detect malicious nodes that do not violate the rules.

IDSs are also classified according to the location of the system into *network-based IDS* and *host-based IDS*. A *network-based or centralized IDS* is located on the base station where the sensor nodes send network flow information for analysis and detection of malicious activities, such as the IDSs introduced by Ngai *et al.* in [10], Wang *et al.* [115], and Buttyan *et al.* [116]. Ngai *et al.* detect sinkhole attacks by identifying the sensor nodes that send inconsistent network flow information (next hops and costs) to the base station. Wang *et al.* detect wormhole attacks by searching for anomalies in distance estimates resulting from wormhole connections. Buttyan *et al.* assume that the base station knows the distribution of the sensor nodes a priori. The base station builds an estimated distribution graph and compares it to a hypothetical graph it builds from the neighbour lists received from the sensor nodes to detect wormhole attacks. Network-based IDSs have a global view of the WSNs and can detect colluding malicious nodes but it suffers from the extra traffic to send network flow information to the base station.

On the other hand, *host-based IDSs* generate less traffic but their decisions are local to the sensor nodes. A *host-based or decentralized IDS* is installed on the sensor nodes. Each sensor node analyzes its local traffic to detect any malicious or abnormal behaviour. Sensor nodes may share their local decisions to reach a consensus about a malicious node. Host-based IDSs are more common in WSNs, such as the systems introduced by Ioannis *et al.* [84], Krontiris *et al.* [86], Demirbas and Song [117], and deGraf *et al.* [118]. Ioannis *et al.* detect blackholes and selective forwarding attacks by a voting scheme. If a sensor node suspects a neighbouring sensor node, it will broadcast an alert. If the number of alerts from different sensor nodes exceeds a certain threshold, the sensor nodes will flag the neighouring sensor node as malicious. Krontiris *et al.* have extended the previous

IDS to detect sinkhole attacks. The sensor nodes broadcast the lists of neighbours in response to a suspicious activity. Nodes then compute the intersection of their neighbour lists, and if a single node remains, they flag it as a sinkhole. Demirbas and Song detect Sybil attacks by testing RSSI values periodically. A Sybil attacker claiming multiple IDs reveals the same RSSI value for more than one ID. deGraf *et al.* introduce a second layer of powerful nodes to detect wormhole attacks. This second layer of nodes is collocated with the sensor nodes according to a certain placement criterion. The placement criterion enables the intrusion detection nodes to detect wormholes in the WSN by eavesdropping on the communication between the sensor nodes.

2.6 Summary

This Chapter gave a detailed survey about WSNs. It highlighted the differences between ad hoc networks and WSNs. It also went through the famous operating systems for WSNs and explained the protocol stack of WSNs from the OSI reference model perspective. The survey on WSNs ended with the advantages and applications of WSNs. Then the Chapter put an emphasis on the routing protocols for WSNs. It surveyed some of the proposed classifications for WSN routing protocols. It explained different performance and cost metrics that affect the choice of the routing protocol. It emphasized the routing protocols of TinyOS since it is the defacto operating system for WSNs and the choice for this work. The Chapter went through the security threats that can be launched against WSN and explained a classification that divided them according to the protocol stack. The Chapter ended by explaining security measures that can be used to defend against the security threats to WSNs. Some of the proposed implementations of these measures were discussed.

This dissertation focuses on the threats that exploit link quality routing protocols.

The next Chapter explains link quality cost metrics in detail and states the problem that this dissertation solves. Then it explains the research methodology used in the solution.

Chapter 3

PROBLEM FORMULATION AND METHODOLOGY

3.1 Introduction

This Chapter provides a thorough survey of link quality routing protocols for WSNs. It begins by explaining what the link quality routing protocols are. The Chapter details the classifications of link quality estimators and goes over some of their comparisons. Then it gives a detailed description of the estimators of importance to this dissertation. The Chapter follows this survey by explaining the research methodology used in this dissertation. Finally, the Chapter ends with describing the simulation framework of the experiments in this dissertation.

3.2 Link Quality Routing Protocols

The sensor nodes in a WSN relay data to the base station using wireless communication. The wireless communication, especially low-powered, is known for its unreliability and fluctuations due to interference, collisions, multi-path effects, and obstacles. These factors affect the performance of the WSNs, namely topology control and routing. Consequently, traditional cost metrics, such as hop count, latency, and round trip time, fail to provide highly reliable routes in WSNs. Estimation of link qualities emerges as an important factor in the selection of stable routes. The more accurate the link quality estimation is, the more stable the routes will be [73, 119, 120]. Accurate estimation:

- improves data delivery to the base station;
- avoids excessive transmissions over low quality links; and

- minimizes route selection triggered by link failures.

Each sensor node estimates link qualities for a set of its neighbouring sensor nodes based on observing their transmitted packets. This observation can either be through packets addressed to the sensor nodes or packets observed on the wireless medium. Estimates are not necessarily symmetric nor static. They are not symmetric because the conditions influencing the computations of each sensor node, such as number of neighbours or proximity to obstacles, are different. Also, the estimates are not static because signal strength and interference change over time. In some cases, the sensor nodes may share their estimates with neighbouring sensor nodes to compute a link quality for both directions of a link. Figure 3.1 illustrates this concept. Two sensor nodes, S and R, are neighbours and thus, they share a wireless link. Estimates can be computed for inbound communication and outbound communication. From the perspective of sensor node S, inbound communication is the packets it overhears from sensor node R, while outbound communication is the packets that sensor node R overhears from sensor node S. Sensor node S can compute an estimate for the inbound communication but it cannot for the outbound communication because it does not know what sensor node R overhears. To compute a link quality for the wireless link, sensor node S may either use the inbound estimate only or the inbound and outbound estimates. The former case is called a *unidirectional link quality*. The latter case requires sensor node R to share its estimate with sensor node S, which is the outbound value from the perspective of sensor node S. Sensor node S uses both estimates to compute a *bidirectional link quality*.

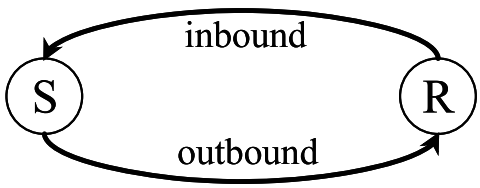


Figure 3.1: Directions of a wireless link

Link quality routing protocols construct more stable routing trees than shortest path routing protocols because simply hearing packets only is not enough to choose a next hop. Choosing routes depending on hearing packets will result in a few long hops but with low link reliability. Thus, it is better to have more shorter hops with high link reliability [119]. Link quality routing protocols will achieve the latter case by computing estimates for link qualities and choosing a neighbouring sensor node as a next hop if its link quality exceeds some threshold, say 75%. The component of link quality routing protocols that does these computations is called link quality estimator or simply estimator.

3.3 Link Quality Estimators

Several link quality estimators have been proposed for the routing protocols of WSNs. Some of these estimators compute unidirectional link qualitie and some compute bidirectional link qualities. Some of these estimators require the cooperation of sensor nodes to share their estimates and some do not. The success of the estimator to construct a stable and reliable routing tree faces some challenges [119]:

- *Speed*: The estimator should be fast so that it can adapt quickly to the changes in the links.
- *Stability*: The estimator should not be too sensitive to transient variations in the underlying connectivity so that the routing tree does not change chaotically.
- *Storage*: The estimator should use small memory space, and be simple and effective.

3.3.1 Classification of Link Quality Estimators

Link quality estimators are classified according to their mode of operation, the way they collect statistics about the wireless links, or the type of computed link quality.

Link quality estimators are classified into two modes of operation: *hardware-based estimators*, and *software-based estimators*. *Hardware-based estimators* choose routes faster and require no computation overhead because they compute link qualities directly from the radio module. This class includes the estimators LQI, RSSI, and SNR. Although hardware-based estimators are faster than software-based estimators, they are inaccurate because they are measured only for successfully received packets. *Software-based estimators* are preferred. They count or approximate the reception ratio or the average number of packet transmissions before any successful reception [73].

Link quality estimators collect statistics about the wireless links either through *active monitoring* or *passive monitoring*. *Active monitoring* means that the sensor nodes intentionally send probe packets to collect statistics about the wireless links. For example, sensor nodes broadcast beacons or route update packets periodically. *Passive monitoring* means that the sensor nodes infer the statistics about the links from the packets sent/received over them. For example, sensor nodes snoop data packets or ACK packets off the wireless links [70, 73].

Link quality estimators either compute *unidirectional qualities* or *bidirectional qualities*. *Unidirectional link qualities* are mostly computed independently without any cooperation among the sensor nodes. *Bidirectional link qualities* may require the sensor nodes to cooperate to share their estimates. However, some bidirectional link quality estimators work independently as well.

3.3.2 Related Work

There has been some research work exploring the characteristics of the wireless links in WSNs. Many of these works have proposed ways to improve the reliability of these wireless links. Other research work has been dedicated to comparing link quality estimators.

Baccour *et al.* [73] have compared the estimators PRR, ETX, RNP, Window Mean

Exponentially Weighted Moving Average (WMEWMA) [119], and Four-Bit Link quality Estimator (4-bitle) [121], in a simulated WSN using the CTP routing protocol. They have concluded that bidirectional estimators, ETX and 4-bitle, have shown better data delivery than the unidirectional ones. WMEWMA has shown the highest stability but the worst data delivery to the base station because of its high over-estimation. Over-estimation can be costly in terms of link reliability; 4-bitle has shown the lowest over-estimation and ETX comes next. Although PRR and RNP are the simplest estimators, they have the lowest performance among the five estimators.

Liu *et al.* [120] have evaluated the performance of ETX, 4-bitle, and RNP estimators in a WSN using CTP. They have concluded that RNP establishes shorter path length than ETX and 4-bitle. RNP has near 100% data delivery to the base station but RNP shows a slight degradation in high-density WSNs. Finally, they have found that ETX adds more routing overhead than the other two estimators.

In conclusion, there is no perfect estimator that works in all conditions. Each one has its advantages and drawbacks that must be considered per application or WSN. The following gives the details of the software-based estimators that are relevant to this dissertation.

3.3.3 Packet Reception Ratio Estimator

The PRR is the probability that a receiving sensor nodes receives a packet successfully. The PRR estimator is a passive unidirectional link quality estimator. PRR is computed at the receiving sensor node as the average of the number of successfully received packets to the total number of transmitted packets, for a window of w received packets, as shown in Equation 3.1. The numbers of missed and transmitted packets are inferred from the sequence numbers of successfully received packets. Large PRR values are better than small values.

$$PRR(w) = \frac{number\ of\ successfully\ received\ packets}{number\ of\ total\ transmitted\ packets} \tag{3.1}$$

3.3.4 Window Mean EWMA Estimator

WMEWMA approximates PRR and considers the effect of the previously estimated value on the new value. It is a unidirectional passive estimator at the receiving sensor node. WMEWMA updates the estimated link quality for a window of w successfully received packets with history factor $\alpha \in [0, 1]$, as shown in Equation 3.2. Large WMEWMA values are better than small values.

$$WMEWMA(\alpha, w) = \alpha \times WMEWMA + (1 - \alpha) \times PRR \tag{3.2}$$

3.3.5 Required Number of Packets Estimator

The RNP estimator is a unidirectional passive estimator at the sending sensor node for every w transmitted and retransmitted packets. RNP estimates the link quality as the ratio of the number of transmitted and retransmitted packets to the number of successfully received packets, as shown in Equation 3.3. RNP works in routing protocols that implement an acknowledgement scheme. It uses the received ACKs to determine the number of successfully transmitted packets. Small RNP values are better than large values.

$$RNP(w) = \frac{number\ of\ transmitted/retransmitted\ packets}{number\ of\ successfully\ received\ packets} \tag{3.3}$$

3.3.6 Expected Transmission Count Estimator

The ETX is a bidirectional active estimator. The choice of routing paths minimizes the expected number of transmissions/retransmissions required to deliver data to the base

station. The ETX estimator considers the effect of link loss ratios, asymmetry of the loss ratios, and interference along the path.

The ETX estimator considers the asymmetry of the links by computing forward and reverse packet reception, PRR_f and PRR_r, for each link respectively. The combination of both estimates gives an ETX value for the bidirectional link quality, as shown in Equation 3.4. The receiving sensor node computes the PRR_f, and the sending sensor node computes the PRR_r, for each w probe packets. Then the two sensor nodes exchange their PRR values to compute the bidirectional link quality. Small ETX values are better than large values.

$$ETX(w) = \frac{1}{PRR_f \times PRR_r} \tag{3.4}$$

3.3.7 Four-bit Estimator

The 4-bitle quality estimator is a bidirectional hybrid estimator that uses active and passive monitoring at the sending sensor node. Active monitoring depends on the sensor nodes broadcast beacons or route updates periodically. The sending sensor node uses Equation 3.5 to compute the inverse of WMEWMA for w_a received beacons from the receiving sensor node. The computed value is an estimation of the unidirectional link quality from the receiving sensor node to the sending sensor node.

$$EETX_a(w_a, \alpha) = \frac{1}{WMEWMA(w_a, \alpha)} \tag{3.5}$$

The sending sensor node estimates another unidirectional link quality from the sending sensor node to the receiving sensor node by passively monitoring the ACKs arriving from the receiving sensor node. Equation 3.6 shows the computation of this value.

$$EETX_p(w_p) = \frac{number\ of\ transmitted/retransmitted\ packets}{number\ of\ successfully\ received\ packets} \tag{3.6}$$

Finally, 4-bitle combines the two values by using an WMEWMA as shown in Equation 3.7, where EETX is replaced by either EETX_a or EETX_p.

$$four\_bit(w_a, w_p, \alpha) = \alpha \times four\_bit + (1 - \alpha) \times \ EETX \tag{3.7}$$

If the network traffic is heavy, then the passive monitoring will dominate. Otherwise, the active monitoring will dominate. Since data packets are acknowledged from the next hop (receiving sensor node) only, the 4-bitle value represents a bidirectional link quality for next hops only. For other neighbouring sensor nodes, the 4-bitle value represents a unidirectional link quality. Small four-bit values are better than large values.

3.4 Research Methodology

As explained in the previous sections, link quality estimators collect statistics about the wireless link either independently or by cooperation among the sensor nodes. Moreover, link quality routing protocols for WSNs may modify an estimator to fit their needs. For example, MintRoute modifies WMEWMA to a bidirectional version where the sensor nodes exchange their estimate values to compute bidirectional link qualities.

Link quality estimators are not immune against malicious attacks that can exploit them. A malicious node may share false information with its neighbouring sensor nodes to affect the computations of their estimates. Likewise, a malicious node may behave maliciously such that its neighbours infer incorrect statistics about their wireless links. In this dissertation, we aim to detect a malicious node that manipulates the link quality estimator of the routing protocol.

To achieve this detection, we have chosen MintRoute and CTP routing protocols to study. MintRoute represents routing protocols that require cooperation among the sensor nodes to compute bidirectional link qualities. On the other hand, CTP computes

bidirectional link qualities without the requirement of sensor nodes cooperation.

Firstly, we have studied the two protocols thoroughly and revealed vulnerabilities in their estimators. Secondly, we have investigated the scenarios that a malicious node can follow to exploit these vulnerabilities. Thirdly, we have designed detection mechanisms to detect the malicious node. To test the effectiveness of the proposed detection mechanisms, we have implemented and developed both protocols and the detection mechanisms in ns-2 [122], respectively. ns-2 was chosen to add on a project that was built with it. Actual implementation of the two IDSs is deferred to future work due to the lack of time and resources.

3.5 Simulation Framework

The ns-2 is an open source simulator for networking protocols written in the C programming language. We have chosen it to simulate MintRoute and CTP due to its functionality, large community, and flexibility to add new protocols. Unfortunately, ns-2 does not have the implementation of MintRoute and CTP in any of its official distributions. As a result, we have extended ns-2 version 2.31 to include them. The implementation of each protocol and its IDS are discussed in the respective chapters.

3.5.1 Implementing TinyOS MAC protocol

TinyOS implements a simple Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) [19] MAC protocol as shown in Algorithm 3.1. To avoid collisions that may occur due to simultaneous transmissions, every sensor node sleeps for a random time between 1 and 32 transmission times. Then the sensor node senses the channel before transmission to make sure that it is not in use. Otherwise, the sensor node sleeps again for a random time between 1 and 16 transmission times. We have extended ns-2 to include this MAC protocol as well.

Algorithm 3.1 MAC protocol of TinyOS

```
SLEEP between 1 and 32 transmission times
while channel is busy do
    SLEEP between 1 and 16 transmission times
BEGIN transmission
```

3.5.2 Implementing a TinyOS Application

To test MintRoute and CTP in ns-2, a traffic generator is required. We have written a simple application to reside in the application layer of the sensor nodes. This application sends a counter value at random during a specified send interval.

3.5.3 Linking It All together

We have written simulation scripts in Tool Command Language (Tcl) [123] to simulate different WSNs with and without the presence of a malicious node according to the following configurations:

- The sensor nodes have the same wireless range following the unit-disk graph model [3].
- The sensor nodes run the same routing protocol.
- The sensor nodes are deployed in an obstacle-free 2D simulation environment of size 100×100 units.
- The sensor nodes are distributed uniformly at random.
- The sensor nodes are stationary.
- The sensor nodes send data packets at random during a specific send interval, S, without any encryption or authentication mechanisms.
- The base station is stationary at the northwest corner of the simulation environment.

- The malicious node is single and stationary.
- The malicious node exists when the WSN is first deployed.
- The malicious node may be an implanted node or a compromised sensor node.
- The malicious node has the same capabilities and wireless range of the sensor nodes (mote-class attacker).
- The malicious node implements the same routing protocol as the sensor node.
- The malicious node may drop, modify, or divert the traffic that traverses it.

3.5.4 Performance and Detection Metrics

The performance of the malicious node is measured by the percent of data delivered to the base station. The simulator is instrumented to record every data packet sent from the sensor nodes and every data packet received at the base station. Forwarded data packets are not counted. In addition, the base station records all the data packets it receives, while disregarding duplicate data packets. The percentage of the data delivered to the base station is computed as the former value divided by the latter value.

The success of the detection mechanisms is measured by computing the True Positives (TPs) and False Positives (FPs). Table 3.1 shows the definition of each measurement.

Table 3.1: Notions of detection measurements

	Attack=True	***Attack=False***
Detection=True	TP	FP

3.6 Summary

In this Chapter, we explained the importance of link quality routing protocols for WSNs over shortest path routing protocols. Three classifications of the link quality estima-

tors were discussed after stating the challenges that faces the design of a link quality estimator. Namely, a link quality estimator should meet the three requirements: speed, stability, and storage. Two recent comparisons of the software-based link quality estimators were discussed. Relevant link quality estimators to this work were explained in details, specifically PRR, WMEWMA, RNP, ETX, and 4-bitle. After going through the link quality estimators, we discussed the research methodology followed in this dissertation. The Chapter ended with describing the framework for the experiments of this dissertation.

The next Chapter starts the analysis and simulation of vulnerabilities in MintRoute, starting with explaining the components of MintRoute.

Chapter 4

MINTROUTE AND ITS VULNERABILITIES

4.1 Introduction

This Chapter explains the components of MintRoute and how they operate together. Then a flaw in MintRoute, which makes it vulnerable to a special type of attack, is explained. A malicious node may use this flaw to influence the choice of routes. Then we propose an IDS to detect and isolate any malicious nodes that exploit this flaw. The Chapter ends with possible scenarios that may occur and how we enhance the proposed IDS to handle these scenarios.

4.2 Components of MintRoute

MintRoute is a flat proactive link quality routing protocol for WSNs that is implemented in TinyOS version 1. Sensor nodes broadcast route update packets periodically to announce their presence, their link qualities, and their routing information. Using this information, the sensor nodes cooperate to construct a routing tree rooted at the base station. MintRoute has six components to perform these tasks: *routing table*; *routing table manager*; *link quality estimator*; *parent selector*; *cycle detector*; and *transmission timer*. Each sensor node runs a local copy of these components. We explain the operation of each component and how they are integrated together in the following.

MintRoute uses a common packet format for all its outgoing packets. This format is shown in Table 4.1. TinyOS version 1 has a maximum packet size of 29 bytes. So any data included in an outgoing packet cannot exceed 22 bytes. This data can be either a route update packet or data collected by the sensor nodes.

Table 4.1: Common fields in MintRoute packets

Field	***Description***	***Size***
sourceAddr	ID of forwarding sensor node	2 bytes
originAddr	ID of originating sensor node	2 bytes
seqNo	sequence number of packet	2 bytes
hopCount	number of hops from base station	1 byte
data	data included in packet	22 bytes

4.2.1 Routing Table

The routing table is the core component of MintRoute. It contains the link qualities and the routing information of up to 16 neighbours. Table 4.2 shows the fields of one record in the routing table for a neighbouring sensor node. Most of the components of MintRoute access the values in the routing table to perform their operations.

Table 4.2: The fields in the routing table of MintRoute

Field	***Description***	***Size***
id	ID of neighbour	2 bytes
parent	ID of parent of neighbour	2 bytes
cost	cost of routing through neighbour	2 bytes
received	number of packets received from neighbour	2 bytes
missed	number of packets missed from neighbour	2 bytes
last SeqNo	last sequence number received from neighbour	2 bytes
sendEst	outbound link quality to neighbour	1 byte
recvEst	inbound link quality from neighbour	1 byte
child Liveliness	flag to indicate neighbour is a child	1 byte
liveliness	frequency of receiving/overhearing packets from neighbour	1 byte

4.2.2 Routing Table Manager

The routing table manager is responsible for inserting neighbours into the routing table and evicting neighbours from the routing table. When a packet is heard from a neighbour that has an entry in the routing table, the corresponding entry is updated accordingly. Otherwise, a new entry is created for the newly heard neighbour. In case the routing table is full, the table manager decides whether to drop the packet or evict a weak neighbour

from the routing table. The operations of the table manager can be summarized as:

- *Insertion*: This operation will insert a new neighbour into the routing table if space exists.

- *Reinforcement*: This operation updates the link qualities of an existing neighbour in the routing table. Algorithm 4.1 shows how the table manager updates the inbound link quality of an existing neighbour in the routing table.

Algorithm 4.1 Update($seqNo$)

SET Δ TO $seqNo$ - `lastSeqNo` - 1
SET `missed` TO `missed` + Δ
SET `lastSeqNo` TO $seqNo$
SET `received` TO `received` + 1

- *Eviction*: This operation uses a frequency algorithm [124] to reinforce existing neighbours as shown in Algorithm 4.2. If the routing table is full, the routing table manager will replace a neighbour whose `liveliness` = 0.

Algorithm 4.2 Evict($nghbrID$)

if a packet received from existing neighbour with ID $nghbrID$ **then**
    SET its `liveliness` TO MAX_COUNT
if timer expires **then**
    DECREMENT `liveliness` for all neighbours

4.2.3 Link Quality Estimator

MintRoute uses the WMEWMA link quality estimator to compute unidirectional inbound link qualities at the receiving sensor nodes. Sensor nodes then exchange their unidirectional link qualities to compute bidirectional link qualities. The sensor nodes collect link statistics by snooping on the communications on the links. They basically infer the number of transmitted and missed packets from the sequence numbers of snooped

packets. Thus, MintRoute requires each packet to hold a sender ID and a sequence number.

WMEWMA first computes a PRR value over a period of time t, as shown in Equation 4.1, for distinct neighbouring sensor nodes. MintRoute expects a minimum number of packets from each neighbouring sensor node per interval t. Faulty or congested sensor nodes may not be able to meet this minimum number. To minimize the probability that these sensor nodes are chosen as next hops, the number of received packets is divided by the maximum of the expected number and the actual number of transmitted packets. Then the PRR value is smoothed with an WMEWMA as explained in Section 3.3.4. MintRoute uses a history factor α with value 0.75, which means that the old WMEWMA value constitutes 75% of the new value. Finally, neighbouring sensor nodes exchange their unidirectional estimates to compute bidirectional link qualities in the parent selector component.

$$PRR(t) = \frac{number\ of\ successfully\ received\ packets}{MAX(expected\ packets,\ transmitted\ packets)} \tag{4.1}$$

Algorithm 4.3 shows the computations of the unidirectional estimates, where `total` = the number of transmitted packets, `EXPECTED` = the number of packets MintRoute expects in t, `newAvg` = PRR value, and `recvEst` = WMEWMA value. Since floating point operations require more resources for the sensor nodes, MintRoute overcomes this limitation by converting floating point operations to integer operations. This is achieved by multiplying `newAvg` by 255. Thus, the lowest link quality value is 0 and the highest link quality value is 255.

4.2.4 Parent Selector

The parent selector component runs periodically to compute bidirectional link qualities, select a parent, and send route update packets. Route update packets inform the

Algorithm 4.3 WMEWMA(α, t)

SET total TO `missed` + `received`
if total < EXPECTED **then**
    SET total TO EXPECTED

SET newAvg TO $255 \times \frac{\texttt{received}}{total}$

if 1st packet to receive or overhear from neighbour **then**
    SET `recvEst` TO newAvg
else
    SET `recvEst` TO $(1 - \alpha) \times$ `recvEst` $+ \alpha \times newAvg$

neighbouring sensor nodes of the current parent's ID, the current routing cost, and the latest unidirectional link qualities of the sensor node. Figure 4.1 shows the fields of a MintRoute route update packet. The `parent` field holds the ID of the current parent, the `cost` field holds the current routing cost to the base station, the `entriesCount` field holds the number of unidirectional link qualities sent in the current route update packet, and the `entries` is a list of link qualities where each link quality, `recvEst`, is associated with the corresponding neighbouring sensor node `neighbour`. Route update packets are limited in size so only good link qualities should be shared with neighbouring sensor nodes. The parent selector embeds link qualities that are greater than 100 in the route update packets. The higher the link quality is, the more reliable the link will be. By this condition, the parent selector shares its link qualities with the good link quality neighbours. If the number of good link quality neighbours exceeds the size of the route update packet, then the parent selector will use a round robin technique [19] to embed the qualities in the route update packets.

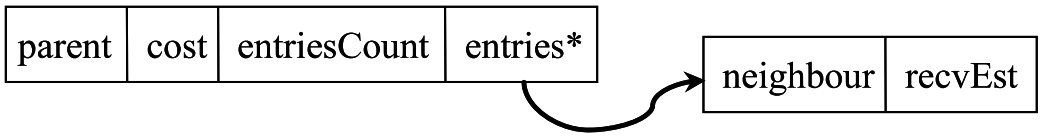


Figure 4.1: Fields of MintRoute route update packet

Upon receiving a route update packet, each neighbouring sensor node searches it for its ID. If it is found, then the sensor node will update the `sendEst` field in its routing table in the entry of the sending sensor node. Simply, the outbound link quality of a sensor node equals the inbound link quality of the neighbouring sensor node at the other end of the link. So, by sharing the inbound link qualities, the sensor nodes will have link qualities for their outbound links. Once a sensor node has values for its `sendEst` and `recvEst` for a neighbour, its parent selector runs Algorithm 4.4 to compute a bidirectional link quality and convert it into a routing cost for that neighbour, where `transEst` = bidirectional link quality, `linkCost` = inverse of the bidirectional link quality, `totalCost` = total routing cost through the corresponding neighbour.

Algorithm 4.4 Cost(*cost*, *sendEst*, *recvEst*)

if $sendEst > 0$ and $recvEst > 0$ **then**
  SET transEst TO $sendEst \times recvEst$
  SET linkCost TO $\frac{16777216}{transEst}$
  SET totalCost TO linkCost + $cost << 6$
  SET totalCost TO totalCost $>> 6$
else
  SET transEst TO ∞
  SET linkCost TO ∞
  SET totalCost TO ∞
RETURN totalCost

Again, to prevent floating point operations, the parent selector component multiplies `transEst` by 16777216, which is 2^{24}. Since the size of `sendEst` and `recvEst` can hold up to 256 values each, the size of their multiplication can hold up to 65536 values. However, the inverse of the bidirectional link quality results in the range [0, 1]. If the inverse is multiplied by some number, then it can be converted to a value greater than one. This magic number is 2^{24}, which moves the inverse values to the range [256, 16777216]. The shift operations in computing `totalCost` puts the values in the range [0, 255].

Parent selector calls Algorithm 4.4 with `cost` = 0 to compute local routing costs to neighbouring sensor nodes. To compute total routing costs to the base station through a neighbouring sensor node, the parent selector calls Algorithm 4.4 with `cost` = routing cost of neighbouring sensor node. Doing the computations for the highest `sendEst` and `recvEst` values, 255 each, `linkCost` will have a value of 4. This means that the cost of the best route increases by 4 with each link along it. The best parent to relay data to the base station is the one with the lowest total routing cost. The parent selector triggers a parent change operation in the following cases:

- New parent has a routing cost better than 75% of the current cost.
- Current parent's link quality drops below 25 (inbound or outbound).
- Base station is unreachable through the current parent (`liveliness` = 0).
- A cycle is detected.

4.2.5 Cycle Detector

The cycle detector checks each received packet to detect possible cycles. A cycle will be detected if a sensor node originates a data packet and receives it later from a child sensor node. Once a cycle is detected, the cycle detector triggers the parent selector to choose a new route to the base station. Thus, the cycle detector requires the IDs of the sensor nodes to be included in data packets.

4.2.6 Transmission Timer

The operations in MintRoute are timer-based. A timer is fired every update interval, U, to send a route update packet. This means that the parent selector is called every U to broadcast route update packets. However, the parent selection is not affected until after the link estimator is called. The link estimator is called every estimate interval E. By

default MintRoute sets $E = 10 \times U$. This relation between U and E means that 10 route update packets will hold the same `parent`, `cost`, and `recvEst` values until the next link estimator call.

4.3 Example of the Operations in MintRoute

MintRoute computes bidirectional link qualities based on how many packets are sent and received over the wireless links. The computed link qualities are estimations of packet reception ratios. In this section, we give a detailed example of the operations in MintRoute.

Suppose that two sensor nodes, A and B, are neighbours and they need to compute the estimates for their link qualities. Figure 4.2 shows the operations of the routing table manager and link estimator of sensor node A with respect to the relevant fields of neighbouring sensor node B. The lines on the right of the figure show the packets transmitted by B with their sequence numbers. A line with an arrow means that the packet is received or overheard by A. Otherwise, the lines indicate packets missed by A. From the figure, we see that A received/overheard five packets from B and missed three packets. When A receives a packet, its routing table manager executes Algorithm 4.1 to update the `received` and `missed` values corresponding to B, as shown in the figure. When the timer expires at the end of interval E, the link estimator executes Algorithm 4.3 to compute the `recvEst` values for all neighbours in the routing table and it resets the `received` and `missed` values. Sensor node B performs the same steps regarding sensor node A.

Subsequent route update packets from sensor node A will hold `recvEst` = 159 for sensor node B. Assume that A announces a routing cost of value 24 through a parent, which is not B. Upon receiving a route update packet from A, the routing table manager

Figure 4.2: Sensor node A computes its inbound link quality to sensor node B

in B searches for the `recvEst` of B in the packet. If it is found, then the routing table manager of B will update the corresponding routing table entry of A as shown in Figure 4.3. In other words, B uses `recvEst` as its `sendEst` to A. The same procedure is followed at A. Assuming that B has received or overheard five and missed five packets from A, B will compute a `recvEst` value of 127 to sensor node A using Algorithm 4.3.

When the parent selector is called, it executes Algorithm 4.4 to compute the routing costs for the neighbours in the routing table. Using the information in the entry of sensor node A, the parent selector of sensor node B will compute a routing cost through A of value 36. If A is offering the lowest routing cost, then the parent selector of B will select A as the parent. The next route update packets from B will announce that A is its parent and the routing cost is 36.

id	*cost*	*recvEst*	*sendEst*
A	24	127	159

Figure 4.3: Sensor node B updates its outbound link quality to sensor node A

4.4 Vulnerabilities of MintRoute to Link Quality Attacks

As explained in Section 4.3, sensor nodes that use MintRoute as the routing protocol in a WSN cooperate to compute link qualities for routes to the base station. Basically, a sensor node computes the qualities of its inbound links and shares the values with its neighbours. Each neighbouring sensor node will use its corresponding value as the estimate for its outbound link quality to the sending sensor node. The sensor nodes assume that all are trustworthy and they don't apply any trust mechanism.

A malicious node can make use of this trustworthiness and lie (i.e., exaggerate) about its inbound link qualities. Following the example in Section 4.3, suppose that sensor node A is a malicious node and it sends the highest possible value, i.e. 255, for its link quality with B. Sensor node B cannot refute or check the validity of this value. Basically, B knows how many packets it has sent out but it does not know how many packets the malicious node has overheard. A high link quality means that A has overheard a high percentage of the packets sent out by B. The highest link quality means that A has overheard all the packets from B and did not miss any of them. Having received this false information from A, B updates the entry of A in its routing table as shown in Figure 4.4.

id	*cost*	*recvEst*	*sendEst*
A	24	127	255

Figure 4.4: Sensor node B updates its outbound link quality with a false value

Now, when the parent selector is called at sensor node B, it will use the false information provided by sensor node A to compute a routing cost through A of value 32. However, this new value may not be sufficiently tempting to B to choose or change its parent to A. Sensor node A can act more maliciously by lying about its cost as well. In

MintRoute, each link in the path increases the routing cost by at least 4. So, if a link between a parent and a child sensor node has the best quality, the routing cost of the child sensor node will be greater than the routing cost of its parent by 4. Otherwise, the difference between the routing costs will be greater than 4. A malicious node can benefit from this property by decreasing its cost to the minimum, which is its parent cost plus 4. Thus, if A has room to decrease its cost to less than 24, then it will succeed in decreasing the routing cost through it as opposed to lying about link qualities only. A more vicious attacker announces a fake parent, i.e., fake route to the base station. In this case, the malicious node can decrease its routing cost as low as it desires. Assume that A has announced a routing cost of 12 through a fake parent X, which does not exist in the WSN. Using this false cost and the values of `sendEst` and `recvEst` in Figure 4.4, B will compute a routing cost of value 20. Now, A has a higher chance to be the parent of B because it is offering a lower routing cost than its true value, 36.

4.5 Detection Mechanisms for MintRoute Vulnerabilities

Wireless communication consumes more power than the other resources of the sensor nodes [11, 77]. Therefore, the objective of any WSN application is to avoid any extra or unnecessary communication. In this section, we will explain our proposed detection mechanism. Our detection mechanism does not require any extra communication among the sensor nodes. However, it requires extra storage space and extra computations. First, we explain how traditional mechanisms cannot defend against such malicious node. Second, we explain the proposed detection mechanism.

4.5.1 Traditional Detection Mechanisms

As explained earlier in the previous Section, a malicious node that exaggerates its inbound link qualities only may not be convincing to its neighbours. There are other factors that

influence the choice of the next hop:

- *Inbound link qualities*: The neighbours of the malicious node compute their inbound link qualities from the malicious node depending on how many packets they receive and miss from it. The malicious node cannot control the computations of these values. On the contrary, missing packets from the malicious node decreases its chances to be chosen as a parent for its neighbouring sensor nodes.

- *Routing cost*: A high routing cost through the malicious node may not be in favour of the efforts of the malicious node to attract traffic. Thus, the malicious node, to achieve its goal, has to decrease its routing cost. This may require the malicious node to announce a fake parent to decrease its routing cost without being detected.

Clearly, encryption or authentication cannot defeat this malicious node. If the malicious node is a compromised sensor node, then it will have authentic keys and will participate legitimately in any encryption or authentication protocol. Furthermore, the malicious node is not breaching the operation of the WSN, it is only announcing incorrect values for its computations.

Another approach is to send network flow information, such as the IDs of next hops and the numbers of packets sent, to the base station. The base station may infer from this data the malicious activity of a node that is not forwarding data or is attracting more traffic than expected. This approach requires a lot of communication overhead. This will deplete the power resources quickly, especially for sensor nodes close to the base station, which makes the WSN non-functional.

It is clear that a novel detection mechanism is needed to handle this type of malicious node.

4.5.2 A New MintRoute Detection Mechanism

The proposed detection scheme adds the ability to the sensor nodes to use their sequence numbers to detect neighbours that misrepresent their inbound link qualities. This detection mechanism works for the link quality routing protocols that require the cooperation between the sensor nodes to compute link qualities. We explain the proposed detection mechanism in terms of MintRoute.

A sensor node can play a "trick" by introducing an artificial gap in its sequence number space. This gap implies a lower bound on the number of missed packets that the neighbouring sensor nodes perceive. Now, the tricking sensor node has a minimum number of missed packets perceived by its neighbours, the number of packets it has sent, and the previous advertised `recvEst` value from each neighbour. The tricking sensor node uses Algorithm 4.3 to compute an estimate for the upper bound for the next `recvEst` advertisement from each neighbour. When the tricking sensor node receives a new link quality, it compares the received link quality with the estimated link quality of the corresponding neighbour. If the received link quality is larger than the estimated link quality, then the tricking sensor node will suspect the corresponding neighbour as malicious. The neighbours of a tricking sensor node cannot tell if packets have actually been missed or a sequence number gap trick is being played. If a malicious node is lying about its link quality, then it will ignore the indicated miss and announce a high link quality to the tricking sensor node. Once the malicious node is detected, the tricking sensor node can blacklist it so as not to consider it as a future next hop. If the tricking sensor node is a child of the malicious node, then it may change its route to avoid the malicious node. For example, let sensor nodes A and B be two neighbours and sensor node A wants to test if sensor node B is sending its true link quality. Assume the following:

- A has the sequence number value 25 to use in its next outgoing packet.
- A received `recvEst` = 255 in the last route update packet from B.
- A knows that B has reset its `received` and `missed` values after broadcasting its last route update packet.
- A sends 15 packets before receiving the next route update packet from B.

Sensor node A can play the sequence number gap trick by incrementing its current sequence number by two, for example. The next outgoing packet from A will hold the sequence number 27 instead of 25. When sensor node B overhears or receives the next packet from A, it will assume that it has missed two packets from A, following Algorithm 4.1. Thus, B will increment its `missed` value by two. Sensor node A can compute an upper bound for the next `recvEst` from B using Algorithm 4.3 with the following values `missed` = 2, `recevied` = 15, and `recvEst` = 255. Then the new `recvEst` from B should be less than or equal to 232. Thus, A succeeds in estimating an upper bound for the next `recvEst` from B without the help of B. Sensor node A can test all its neighbours in the same way.

The sensor nodes should perform the sequence number gap trick at most once per estimate interval at random. This ensures that only one sequence number gap is present in the computations of the neighbours so it does not severely affect the link qualities.

Time Synchronization Problem

Tight time-synchronization is not required in many WSN routing protocols. Accordingly, the tricking sensor node cannot determine the appropriate route update packets to test. An efficient scheme to overcome the synchronization problem is to record the sequence number at which the tricking sensor node has begun the sequence number gap trick. Also, the sensor nodes must announce the last sequence number they have used to compute the

`recvEst` value of each neighbouring sensor node. A tricking sensor node will only test the route update packets with last sequence number greater than the recorded sequence number.

Over-testing Problem

Testing unnecessary or extra route update packets may lead to an over-testing problem. The over-testing problem simply means blacklisting a neighbour because the received `recvEst` does not conform with the computed expected `recvEst` although a previous `recvEst` was correct. Figure 4.5 depicts this problem. The figure shows the timeline of a tricking sensor node with the times of performing two tricks Tr_1 and Tr_2 and the times of receiving the new `recvEst` values from a neighbouring sensor node, E_i. After the trick at Tr_1, the tricking sensor node uses the `recvEst` received at E_2 to test the truthfulness of the neighbouring sensor node, which passes the test. However, the tricking sensor node may receive a new `recvEst` at E_3 before performing a new trick. Using this new `recvEst` may lead to incorrect conclusion about the neighbouring sensor node. To solve the over-testing problem, the tricking sensor node should have a flag per neighbour to indicate when to test the received `recvEst`. So, when the tricking sensor node performs the trick at Tr_1 it sets the flags of all its neighbours. When the tricking sensor node receives a route update packet with a new `recvEst`, it resets the flag of the corresponding neighbour. Any subsequent route update packets from this neighbour will not be tested until a new trick is played.

Figure 4.5: The over-testing problem

Isolation Problem

An isolation problem may occur if the network has high traffic volume or is dense. The tricking sensor node may lose, due to collisions, the appropriat route update packets to test. This will lead to blacklisting the corresponding neighbour. Losing the appropriate route update packets from all the neighbours will lead to the tricking sensor node blacklisting all its neighbours. This ends up with the tricking sensor node with no route to the base station. To overcome the isolation problem, we propose a simple algorithm for removing the neighbours from the blacklist. Upon receiving the appropriate route update packets from neighbours, the tricking sensor node implements Algorithm 4.5. If a neighbouring sensor node fails the test, then the testing sensor node will set the corresponding flag `blackListed` = 255. Subsequent passes from the test decrease the value of `blackListed` by half. Once the value comes under a certain threshold, the tricking sensor node removes the corresponding sensor node from the blacklist.

Algorithm 4.5 Blacklist(*recvEst*, expected *recvEst*)

if *recvEst* > expected *recvEst* **then**
  SET blackListed TO 255
else
  SET blackListed TO $\frac{blacklisted}{2}$
  if blackListed < blackList_threshold **then**
    SET blackListed TO 0 ▷ remove neighbour from blackList

An alternative way to blacklist a node is to set an alert threshold for consecutive exaggerated `recvEst` values. Once a neighbour exceeds the threshold it will be blacklisted. However, this technique takes longer to blacklist a suspicious node than Algorithm 4.5.

4.6 Attacker's Strategies to Defeat Detection

We assume that the malicious node is an intelligent node that knows that a detection mechanism is running in the WSN. In this section, we show how the malicious node can try to escape detection. We first explain how to detect a naive malicious node that is not aware of the detection mechanism. Then we proceed with a more intelligent malicious node that knows the detection mechanism and how we adapt the proposed detection mechanism. Finally, the malicious node reaches the state of conformity.

4.6.1 MintRoute Scenario 1

In this scenario, the malicious node uses a very simple attack strategy, in which it always announces the highest possible link quality (255). It ignores any missed packets and advertises the highest possible link qualities, which makes it easily detectable. Figure 4.6 shows a hypothetical example of the behaviour of the malicious node and how a neighbour detects the anomalous behaviour. A tricking sensor node decides to perform the sequence number gap trick at the third estimate interval using the previous received link quality, 255, to expect the next link quality, 237. However, the tricking sensor node receives a higher link quality, 255, so it decides that its neighbouring sensor node is behaving maliciously.

4.6.2 MintRoute Scenario 2

While the malicious node in Scenario 1 is easily detected, it is reasonable to consider a more sophisticated behaviour. In particular, the malicious node in Scenario 2 is adaptive. It tries to guess which neighbour is performing the sequence number gap trick and when. It then advertises the correct link quality to the corresponding neighbour. As a result, the malicious node succeeds in avoiding being detected some of the time. Figure 4.7 [125] illustrates the behaviour of the malicious node against a single neighbouring sensor node

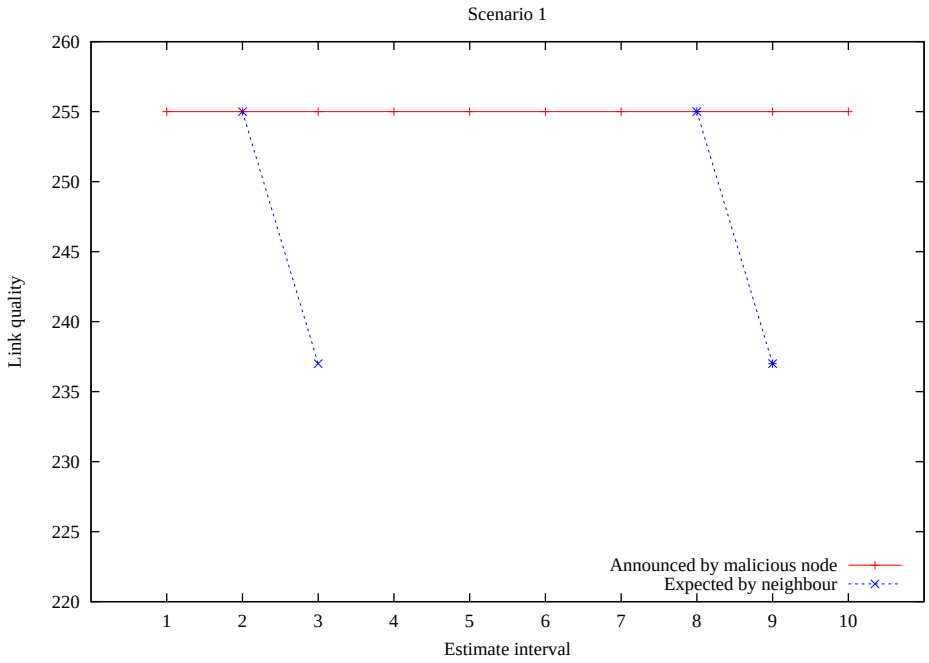


Figure 4.6: Behaviour of malicious node in MintRoute Scenario 1

that is performing the sequence number gap trick. We can see that at the third estimate interval, the malicious node has guessed that a neighbouring sensor node is performing the sequence number gap trick, so it advertises the correct link quality to this neighbour. Thus, the malicious node has succeeded in escaping detection. However, at the ninth estimate interval, the malicious node is not successful in guessing that the corresponding neighbour is performing the sequence number gap trick and it fails to avoid the detection.

To elaborate, the malicious node computes a threshold, τ, for the number of packets it overhears from each neighbouring sensor node. Then it decides whether to lie or not depending on the observed packet traffic relative to the threshold. In essence, if unusually many packets are "missing" and the number of observed packets exceeds the threshold, the malicious node suspects that the sequence number gap trick is being played, and tells the truth to avoid being detected. Mathematically, the malicious node may use a threshold such as $E[X] + \sigma$, where $E[X]$ is the expected number of packets, and σ is the standard deviation.

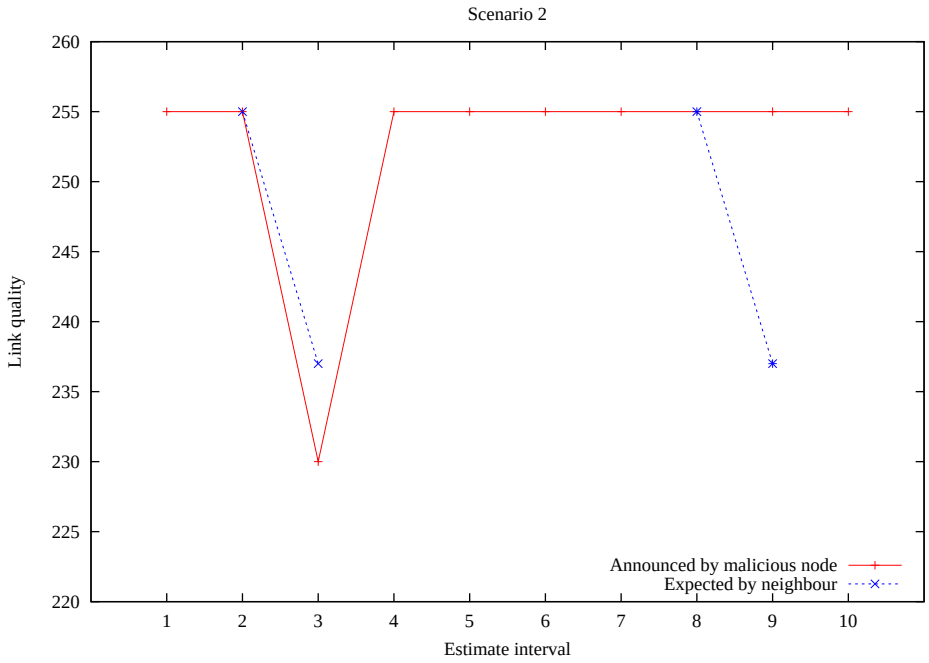


Figure 4.7: Behaviour of malicious node in MintRoute Scenario 2

To compute the expected number of packets to overhear from a neighbouring sensor node, the malicious node needs to know how many packets this neighbour should send during one estimate interval, E. The number of data packets can be computed from the sizes of E and the send interval, S. Equation 4.2 shows how many data packets the malicious node expects to receive from a single neighbouring sensor node in one estimate interval. Basically, it is the ratio of the size of estimate interval to the size of the send interval.

$$E[X] = \frac{E}{S} \tag{4.2}$$

Moreover, the sensor nodes forward the data of all the sensor nodes in their subtrees. Thus, there will be many data packets coming during the same send interval if each sensor node in the subtree sends one data packet. Assuming that the malicious node knows the subtree size, n, of each of its neighbouring sensor nodes, it computes the expected number of data packets from each neighbour as:

$$E[X] = n \times \frac{E}{S} \tag{4.3}$$

Next, the malicious node computes the variance, $Var(X)$, for each of its neighbouring sensor nodes depending on the corresponding $E[X]$.

However, this is not enough to determine the threshold. There are route update packets from each neighbouring sensor node. As per the default configuration of MintRoute, a sensor node sends ten route update packets during a single E. The malicious node computes the expected total number of packets as shown in Equation 4.4.

$$E[X] = 10 + n \times \frac{E}{S} \tag{4.4}$$

The final problem that faces the malicious node is how to know n, the subtree size, of each neighbouring sensor node. The malicious node uses the following features of MintRoute: data packets include the identifier of the originating sensor node; and no encryption mechanism is implemented. For any neighbouring sensor node that forwards a data packet, the malicious node can read the originator of the data packet. Then the malicious node assigns this originator to the subtree of the corresponding neighbour. At the end of the estimate interval and before computing the link qualities, the malicious node counts the number of sensor nodes in each subtree. Now, before advertising its link qualities, the malicious node can compute the thresholds to decide which link qualities should be exaggerated and which should be true. Since the subtrees may change, the malicious node resets n after advertising its link qualities and begins counting again in the new estimate interval.

4.6.3 MintRoute Scenario 3

One weakness of the sequence number gap trick used in Scenario 2 is that the sensor nodes only monitor the link quality advertisements immediately following a sequence

number gap trick, rather than continuously.

The obvious solution is to make the sensor nodes more vigilant. In particular, because link quality computations in MintRoute use an exponentially weighted moving average, it is impossible for a link quality estimate to jump dramatically from one estimate interval to the next. An enhanced detection mechanism can monitor link qualities more closely to detect malicious nodes at all times, as illustrated in Figure 4.8 [125]. In other words, when a tricking sensor node is not playing the sequence number gap trick, it assumes that all neighbouring sensor nodes have received all its outgoing packets and `missed` $= 0$. It computes the expected link qualities for all neighbouring sensor nodes and uses them to test the next advertisements of the neighbours.

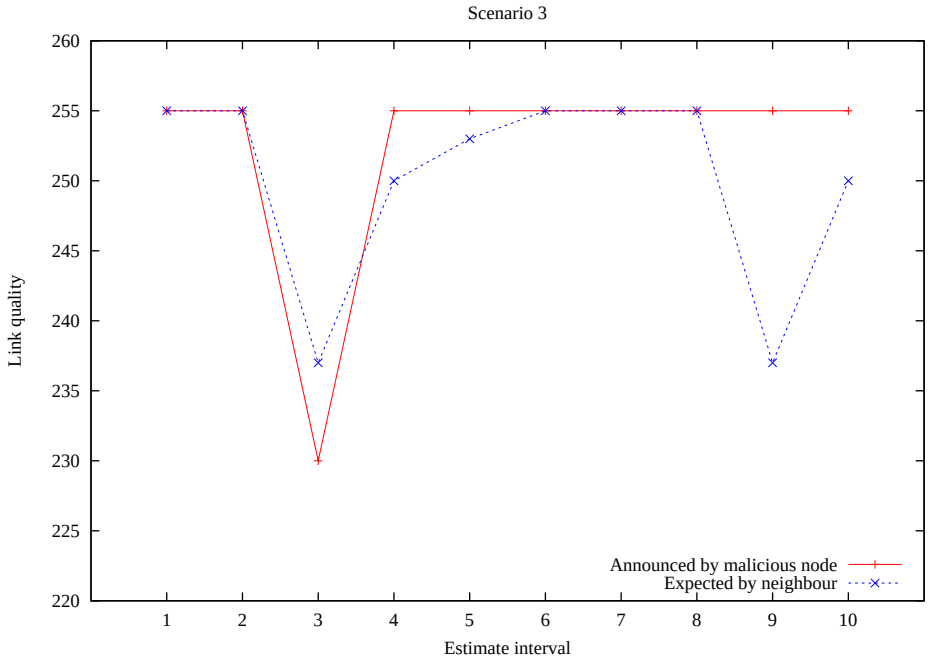


Figure 4.8: Behaviour of malicious node in MintRoute Scenario 3

4.6.4 MintRoute Scenario 4

The enhancement to the detection mechanism in Scenario 3 can also be applied to the malicious node itself. That is, following a (detected) sequence number gap trick, the

malicious node can cautiously increase its link quality back to the maximum so as to avoid suspicion, as shown in Figure 4.9 [125]. This strategy effectively converts the malicious node into a conformant node.

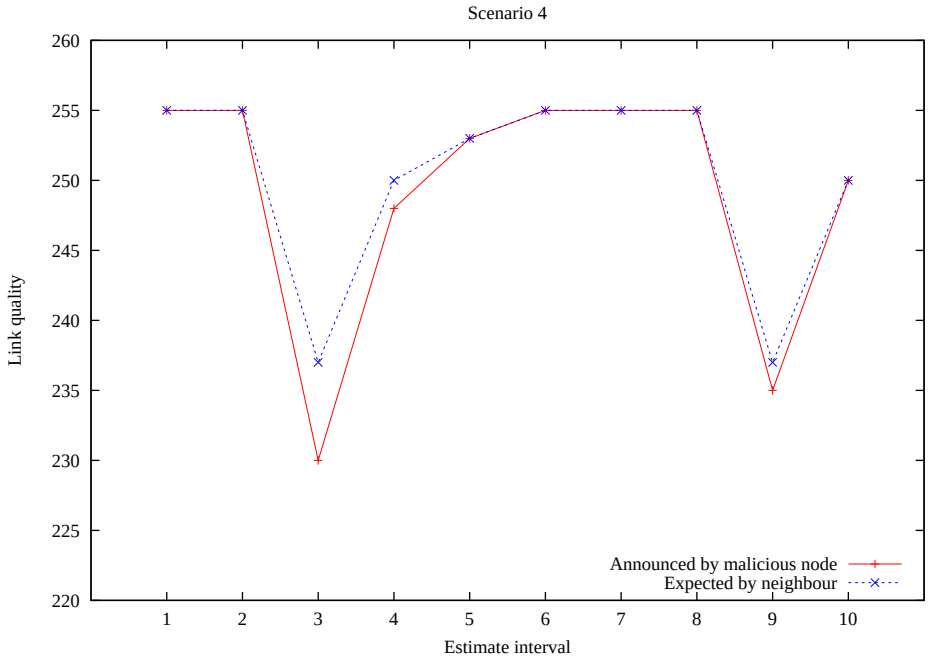


Figure 4.9: Behaviour of malicious node in MintRoute Scenario 4

4.7 Summary

In this Chapter, we explained the six components of MintRoute and showed how they cooperate to construct a routing tree using bidirectional link qualities. Then the Chapter proceeded with an example of the operations in MintRoute. The Chapter followed the example with vulnerabilities in MintRoute that a malicious node can use to influence the computation of link qualities to its favour. Simply, the malicious node can exaggerate its link qualities and the sensor nodes cannot validate or refute them. MintRoute assumes that all nodes are trustworthy. We explained why traditional security mechanisms cannot detect this malicious behaviour and the need to find a novel detection mechanism. Then

the proposed detection system was explained and discussed. The Chapter ended with possible scenarios that a malicious node can follow and how the proposed detection system can be adapted to bring malicious nodes to conformity.

The next Chapter explains how ns-2 is extended to support MintRoute and how the proposed IDS is implemented. It also discusses the simulation of MintRoute and the detection of the malicious node.

Chapter 5

MINTROUTE: SIMULATION MODEL AND RESULTS

5.1 Introduction

This Chapter describes the implementation of MintRoute and its IDS in ns-2. It begins by explaining the extensions made to ns-2. Then it describes the configuration of the simulation environment. The Chapter follows with the simulation of MintRoute WSNs where a malicious node exploits the routing protocol. Different scenarios that the malicious node can follow are simulated. In each scenario, the malicious node adapts its attack strategy to escape detection. In parallel, we explain how the IDS is modified to detect the adaptive malicious node. The simulation results show that the proposed IDS detects the malicious node effectively.

5.2 Extending ns-2 to Support MintRoute

In this section, the extensions made to ns-2 to support MintRoute and the proposed MintRoute IDS are explained.

5.2.1 Implementation of MintRoute

We have used the source code of MintRoute in TinyOS [126] and its implementation in the Castalia simulator [127] as a reference for our implementation. TinyOS implements MintRoute as one component with different functions to implement its six components. Therefore, we have implemented MintRoute as a single class in ns-2.

5.2.2 Implementation of MintRoute IDS

After having MintRoute running in ns-2, we have extended MintRoute itself to support the proposed detection mechanism. In addition to extending the code, which will be explained later, some new fields are required in the routing table. We have added the fewest possible extra bytes to the routing table of MintRoute as shown in Table 5.1.

Table 5.1: New fields in MintRoute routing table

Field	***Description***	***Size***
estSeqNo	`lastSeqno` used to calculate neighbour `recvEst`	2 bytes
usedSeqNo	sequence number neighbour used to calculate `recvEst`	2 bytes
flags2	0, TRICK, 0, 0, 0, 0, 0, 0	1 byte
blackList	trust value	1 byte

Storage Space Overhead

To solve the time synchronization problem as explained in Section 4.5.2, two fields are required: `estSeqNo` and `usedSeqNo`. The `estSeqNo` field is used to hold the `lastSeqNo` value used to compute the `recvEst` value of a neighbouring sensor node. Then the value of `estSeqNo` is sent with the `recvEst` value in the same route update packet. The sensor nodes cannot simply send the value of `lastSeqNo` because they may overhear a new route update packet between the computation and the transmission of `recvEst`. Thus, `estSeqNo` is used to resolve this conflict. The second field `usedSeqNo` is populated when the sensor nodes overhear the route update packets. This field is populated with the value of `estSeqNo` received in the route update packets. Finally, to complete the solution of the time synchronization problem, a global variable `trickSeqNo` is required to hold the sequence number at which a tricking sensor node starts the sequence gap trick. Only neighbouring sensor nodes that have `usedSeqNo` greater than `trickSeqNo` are tested.

To solve the over-testing problem, the `TRICK` bit in the `flag2` field is used to determine the neighbours that are already tested in the current trick. When a tricking sensor node

starts a sequence number gap trick, it sets this bit for all the neighbouring sensor nodes in its routing table. When a neighbour is tested, the tricking sensor node resets the `TRICK` bit in the corresponding routing table entry. Other bits in the `flag2` field are reserved for future purposes.

Finally, the `blackList` field is used to determine blacklisted neighbours. In addition, its value solves the isolation problem by removing good sensor nodes from the blacklist.

In total, 98 extra bytes of memory are required per sensor node. The additional fields in the routing table take 96 bytes for the 16 entries plus two bytes for the global variable, `trickSeqNo`.

The malicious node does not play the sequence gap trick so it does not blacklist any of its neighbouring sensor nodes. Thus, the malicious node requires the `estSeqNo` field only to send the values of `lastSeqNo` used in the computations of `recvEst` values. However, the malicious node needs to keep track of the size of the subtrees of its neighbouring sensor nodes. Every sensor node that is sending a data packet in a subtree should be counted once during a single estimate interval. To achieve this, the malicious node has an array of bytes for each neighbouring sensor node. The array size equals the size of the WSN. The indices of the array represent the IDs of the sensor nodes in the WSN. When a sensor node sends a data packet, its corresponding index in the array is set to 1. With every new estimate interval the malicious node resets all the indices of the arrays of its neighbouring sensor nodes to begin a new count. This reset is important to accommodate the changes that may occur in the routing tree. Hence, the storage requirement for the malicious node is dependent on the size of the WSN.

Packet Overhead

The other modification to MintRoute comes in its packet format. As explained in Section 4.2, a MintRoute packet has a data field that can hold up to 22 bytes. In the case

of route update packets, the data portion holds the routing packet shown previously in Figure 4.1. The routing packet has a fixed part of size five bytes and a variable part that can hold up to 17 bytes. Each neighbour/`recvEst` pair requires three bytes and thus, the routing packet can hold up to five neighbour/`recvEst` pairs. If a sensor node has more than five neighbours, the routing update packets will hold the neighbour/`recvEst` pairs in a round robin fashion. The modified route update packets have the triplet neighbour/`recvEst`/`estSeqNo`. This modification occupies five bytes and thus, the modified route update packets hold three neighbour/`recvEst`/`estSeqNo` triplets only.

Code Overhead

Algorithm 5.1 shows the pseudo-code of the proposed IDS. When a sensor node, *node*, receives a route update packet, *pkt*, from its neighour, *nghbr*, it goes through Algorithm 5.1 to test the `recvEst` value. The *expEst* variable is computed using Algorithm 4.3. Currently, the *blacklist threshold* is set to 1. This threshold can be used if we do not want to blacklist a neighbouring sensor node after a single test failure. When the parent selector selects a next hop, it considers only the neighbouring sensor nodes that are not blacklisted.

Algorithm 5.2 shows how the malicious node builds the subtrees of its neighbouring sensor nodes. When the malicious node receives a data packet, *pkt*, from its neighbouring sensor node, *nghbr*, it will check if the origin sensor node has sent a data packet before. If it is the first data packet, then the malicious node will set the array index at the position of the origin ID. To know the subtree size of any of its neighbours, the malicious node counts the number of cells that are set in the corresponding array. Then the malicious node can compute the expected number of packets to overhear from each neighbouring sensor node.

When the estimate interval of the malicious node expires, it updates the estimates

Algorithm 5.1 ReceiveUpdatePkt(*node*, *nghbr*, *pkt*)

if *node* ID included in *pkt* **then**
  if *node* playing trick **then**
    if *pkt* holds a new `recvEst` for *node* & `usedSeqNo` in *pkt* > `trickSeqNo` **then**
      COMPUTE expEst ▷ Call Algorithm 4.3
      if `recvEst` > expEst **then**
        INCREMENT alert
        if alert ≥ blacklist threshold **then**
          SET *nghbr* `blackList` TO 255
      else
        SET *nghbr* `blackList` TO $\frac{nghbr\ \texttt{blackList}}{2}$
        RESET alert
      RESET `TRICK` bit for *nghbr*
  else if *node* not playing trick & *pkt* holds a new `recvEst` for *node* **then**
    COMPUTE expEst ▷ Call Algorithm 4.3
    if `recvEst` > expEst **then**
      INCREMENT alert
      if alert ≥ blacklist threshold **then**
        SET *nghbr* `blackList` TO 255
    else
      RESET alert
else
  DROP *pkt*

Algorithm 5.2 UpdateSubtree(*nghbr*, *dPkt*)

READ origin FROM *dPkt*
if index origin in array of *nghbr* is not set **then**
  SET index origin in array of *nghbr* TO 1

of the link qualities of its neighbouring sensor nodes. However, the malicious node calls Algorithm 5.3 to guess which neighbours are playing the sequence number gap trick. First, it calls Algorithm 5.4 to compute the expected number of packets, ex, to receive from neighbouring sensor node *nghbr* and the associated variance, $varX$. Next it computes the standard deviation, *stdDev*. To determine if *nghbr* is playing the sequence number gap trick or not, the malicious node compares the number of received packets from *nghbr* to the threshold (ex + stdDev). If the number of received packets from *nghbr*

exceeds the threshold, then the malicious node will send the true link quality to *nghbr*. Otherwise, the malicious node will send an exaggerated link quality. Algorithm 5.3 can be modified to reflect the different thresholds that the malicious node may use.

Algorithm 5.3 GuessTrick(*nghbr*)

```
COMPUTE ex and varX of nghbr                          ▷ Call Algorithm 5.4
RESET subTree of nghbr                  ▷ Set cells in subtree of nghbr to 0
SET stdDev TO square root of varX       ▷ Compute the standard deviation
if EX + stdDev < number of packets from nghbr then
    SEND true link quality to nghbr
else
    SEND false link quality to nghbr
```

Algorithm 5.4 implements Equation 4.4 to compute the expected number of packets and variance for each neighbouring sensor node *nghbr* of the malicious node. The algorithm begins by counting the number of sensor nodes in the subtree of *nghbr*. This number is counted as the number of cells that are set to 1 in the subtree of *nghbr*. Then the algorithm computes the expected number of data packets to receive from *nghbr*. The algorithm follows with the computation of the variance and finally, it updates the expected number of packets to include the default number of route update packets, EXPECTED_UPDATES, to receive from *nghbr* during a single estimate interval. EXPECTED_UPDATES is set to 10 by default in MintRoute.

Algorithm 5.4 Expectations(*nghbr*)

COMPUTE subTreeSize of *nghbr*
if subTreeSize > 0 **then**
    SET ex TO $\frac{estimate\ interval}{send\ interval} \times$ subTreeSize
    SET varX TO variance of ex
    SET ex TO ex + EXPECTED_UPDATES
else
    SET ex TO 0
    SET varX TO 0

5.3 Setup of MintRoute Simulation Environment

Following the simulation framework presented in Section 3.5, we have written Tcl simulation files to simulate 10 different topologies (same physical locations) with 20 sensor nodes each. MintRoute seeds the random number generator with the current time, so we have seeded the random number generator of ns-2 with the current time as well. Accordingly, for the same physical locations of the sensor nodes a different routing tree is constructed with every new run of the simulation file. We have simulated each topology 25 times. As a result, the malicious node has had the same number of neighbouring sensor nodes, as shown in Table 5.2, but its subtree is different in each run for the same topology depending on the current time. In each simulation run, the following parameters are fixed:

- Wireless communication range = 40 units
- Route update interval, $U = 5$ time units
- Estimate interval, $E = 50$ time units
- Data send interval, $S = 20$ time units
- Simulation time = 5000 time units
- Blacklist removal threshold = 16
- Sequence number gap = 1, 2, 3, 4, 5

Given the previous values, the sensor nodes compute the link qualities 100 times during a single simulation run. The simulation time is sufficient to have a stable and complete routing tree. The blacklist threshold requires a blacklisted sensor node to have five consecutive true link qualities to be removed from the blacklist. The results of the

Table 5.2: Neighbourhood sizes of malicious node in MintRoute simulations

Network ID	***Size of neighbourhood***
1	8
2	7
3	8
4	8
5	11
6	11
7	11
8	12
9	9
10	13

simulations represent the average values calculated from the 25 independent runs for each topology.

The base station initiates the construction of the routing tree by broadcasting a route update packet. Once its neighbouring sensor nodes receive this broadcast, they broadcast their own route update packets. Then their neighbouring sensor nodes will do the same until all the sensor nodes in the WSN broadcast their route update packets. This means that sensor nodes closer to the base station construct routes to the base station before far sensor nodes.

5.3.1 Setup of Malicious node

The malicious node has been placed in the centre of the network with the same configuration as the good sensor nodes. To be more efficient, we have configured the malicious node to announce a routing cost that is lower than the lowest routing cost in its communication range. To achieve this, the malicious node waits for the first neighbour to announce its routing cost to the base station. Most probably, this will be the closest neighbour to the base station with the lowest routing cost. The malicious node immediately announces a lower cost to a fake parent to lure the other neighbouring sensor nodes. Forwarding traffic to a fake parent makes the malicious node look legitimate. On

the other hand, it helps to compute the percentage of data delivery at the base station with and without the presence of the malicious node to measure its effect.

The malicious node does not perform the sequence number gap trick, so as not to affect the computations of its neighbours. This reinforces the probability of choosing the malicious node when they compute routing costs to the base stations.

Finally, the sensor nodes are configured to test route update packets, with or without the sequence gap trick, after 100 time units. This delay ensures that the routing tree is built without interruption.

5.3.2 Setup of Performance Metrics

The performance of the malicious node is measured as the percentage of data delivered to the base station as described in Section 3.5.4.

The success of MintRoute IDS is measured by computing the number of detections that occurred in the WSN and how many of these detections are true. Equation 5.1 shows the computation of the TP and FP ratios.

$$true\ detection = \frac{\sum_{i=0}^{r-1} TP}{r}$$
$$false\ detection = \frac{\sum_{i=0}^{r-1} FP}{r} \tag{5.1}$$

where r = number of runs, i.e., 25.

There are factors other than the sequence number gap trick that may affect these measurements. These factors can cause a mismatch between an advertised link quality and an expected link quality:

- A sensor node starts a second sequence number gap trick shortly after a first one. The second trick cancels the first trick.

- Sensor nodes do not advertise link qualities that are below 100 (as stated by MintRoute). Accordingly, the neighbouring sensor nodes will not receive any link qualities to test. The first link quality to test may arrive later than the trick.

- The number of neighbours is greater than what the route update packets can accommodate in one estimate interval. This means that neighbouring sensor nodes may not be updated in one estimate interval and new link qualities may be computed without advertising the old values.

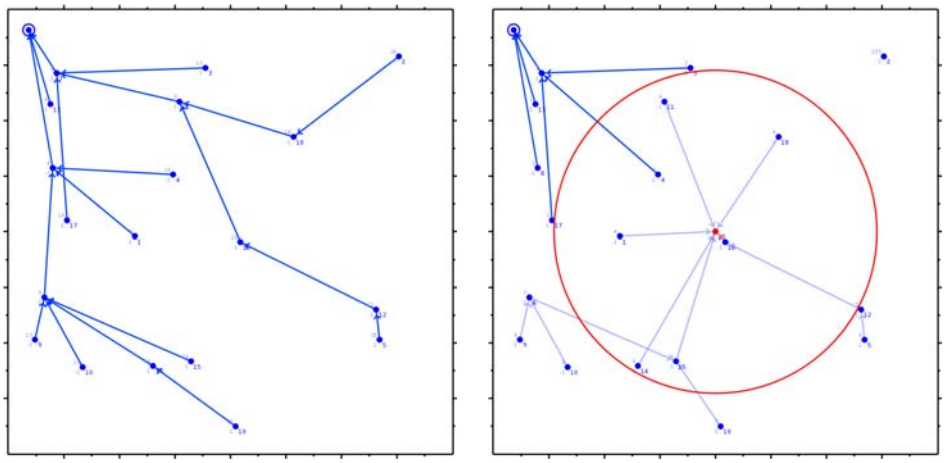
5.4 Results of MintRoute Simulation

We incrementally present the scenarios in which a malicious node exploits the MintRoute routing protocol. In each scenario, we discuss the performance of the malicious node by measuring data delivery at the base station and the success of detection by measuring TP and FP values.

When the sensor nodes detect the malicious node, they change their parents to good sensor nodes. A high percentage of data delivery means that the sensor nodes detect the malicious node quickly. Accordingly, a large proportion of data reaches the base station intact. A low percentage of data delivery means that the malicious node is successful in escaping detection.

We discuss the simulation results for the percentage of data delivery for the ten simulated topologies. However, for the success of detection, we discuss the simulation results of network 2 because the neighbourhood of the malicious node is the smallest. Accordingly, the graphs of the simulation results are less dense. However, all the observations stated in this Section can be stated for the other nine networks as well. In network 2, the malicious node has seven neighbouring sensor nodes with the IDs: 1, 4, 11, 14, 15, 16, and 18. Figure 5.1 [125] shows the topology of network 2 with and without the malicious

node.



(a) Before attack (b) After attack

Figure 5.1: MintRoute Network 2

5.4.1 Simulation of MintRoute Scenario 0

As a baseline, we compare the data delivery at the base station in the different topologies with and without the presence of the malicious node. In both cases, the sensor nodes do not run the proposed IDS.

First, the sensor nodes build a routing tree rooted at the base station without the presence of the malicious node in the network. The best route in MintRoute is computed as the route with the highest packet reception ratio, interpreted as the lowest total routing cost. MintRoute offers a stable and robust routing tree that changes only when the quality of a link deteriorates. The simulation results show over 99% data delivery to the base station in WSNs using MintRoute. See the green bars in Figure 5.2.

Second, a malicious node is introduced to the network with the aim of diverting as much traffic as possible. To achieve this goal, the malicious node lies about its cost and the estimates of its inbound link qualities. The malicious node uses a very simple attack strategy, in which it always announces the highest possible link quality (255). The

simulation results show that the percentage of data delivery drops to 20% - 60% of the WSN traffic, depending on the size of the neighbourhood of the malicious node. See the blue bars in Figure 5.2 [125].

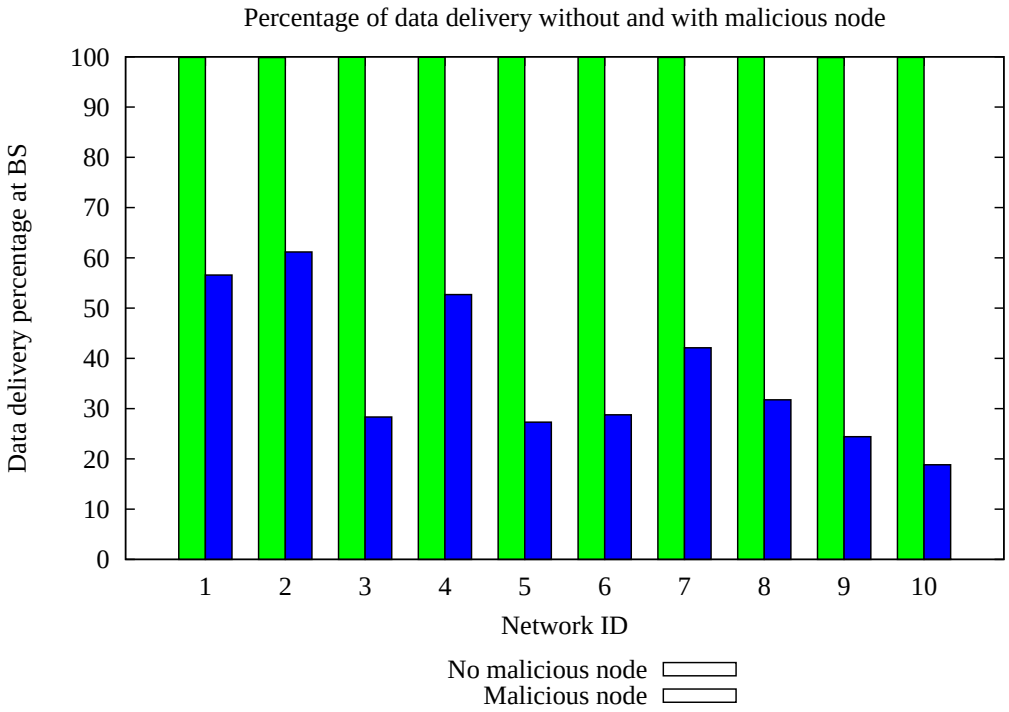


Figure 5.2: MintRoute Scenario 0: data delivery

5.4.2 Simulation of MintRoute Scenario 1

In this scenario, the malicious node behaves as in Scenario 0. However, the sensor nodes use the sequence number gap trick to try to detect it. The sensor nodes are configured to play the sequence number gap trick once (at a random time) during each interval E. This ensures that only one sequence number gap trick is used in the (per-neighbour) computations of the malicious node and, at the same time, it does not severely affect the link qualities. Since the malicious node is passive, it ignores any missed packets and announces the highest possible link quality, which makes it easily detectable.

Effectiveness of Malicious Node

Figure 5.3 [125] shows that the percentage of data delivery is restored to 98% or better in each of the 10 WSN topologies tested. In other words, the sequence number gap trick is a simple and effective means to detect the malicious node. We can also see that the sequence number gap size has no effect on the percentage of data delivery. Since the malicious node is passive, it is detected regardless of the gap size. The slight decrease in the percentage of data delivery can be attributed to the difference in time between when the malicious node announces its route cost and the time the neighbours start the sequence gap trick. The later the sequence number gap trick is played, the higher the possibility that the malicious node disrupts more traffic.

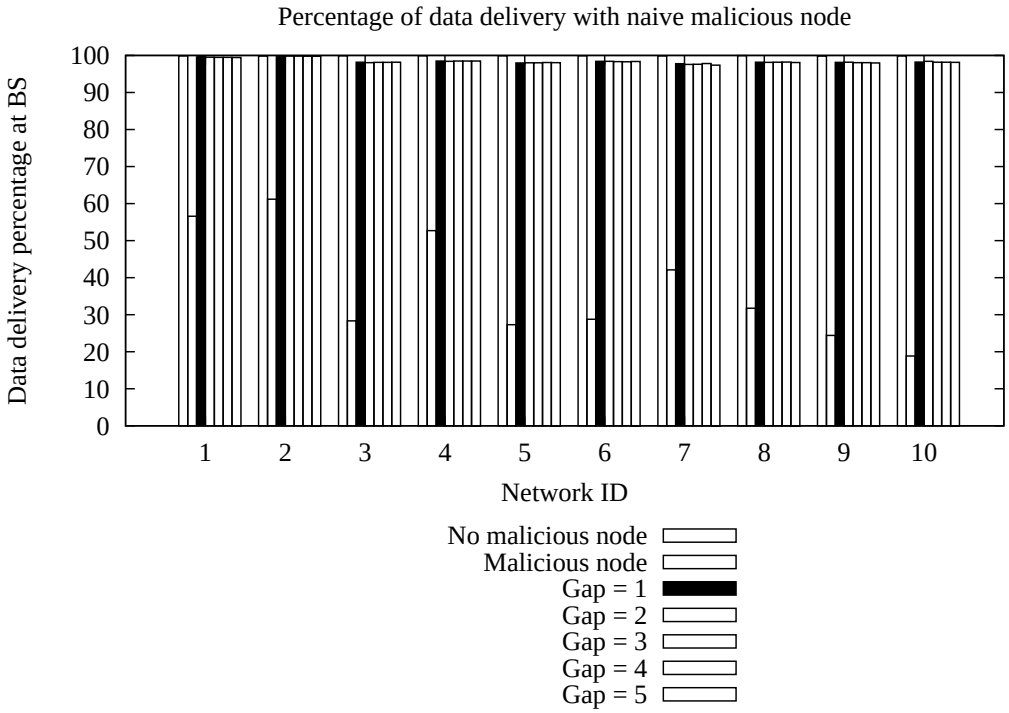


Figure 5.3: MintRoute Scenario 1: data delivery

Success of Detecting Malicious Node

Figure 5.4 depicts the average detection results when the sensor nodes used sequence number gap of size 1. We can see that all the neighbours of the malicious node had successfully detected it with TP between 60% and 80% on average. Although the malicious node exaggerates its link qualities all the time, it is not detected all the time due to the aforementioned factors in Section 5.3.2. Specifically, the malicious node is not detected when a sensor node performs two sequence number gap tricks within a short period of time. The computations of the first trick will be canceled by the second trick and will not detect the malicious node. FP counts for a negligible amount of the detections due to mistakenly suspecting a good sensor node as malicious. For example, sensor node 8 suspected a good neighbour as malicious 6 times in 6 runs only.

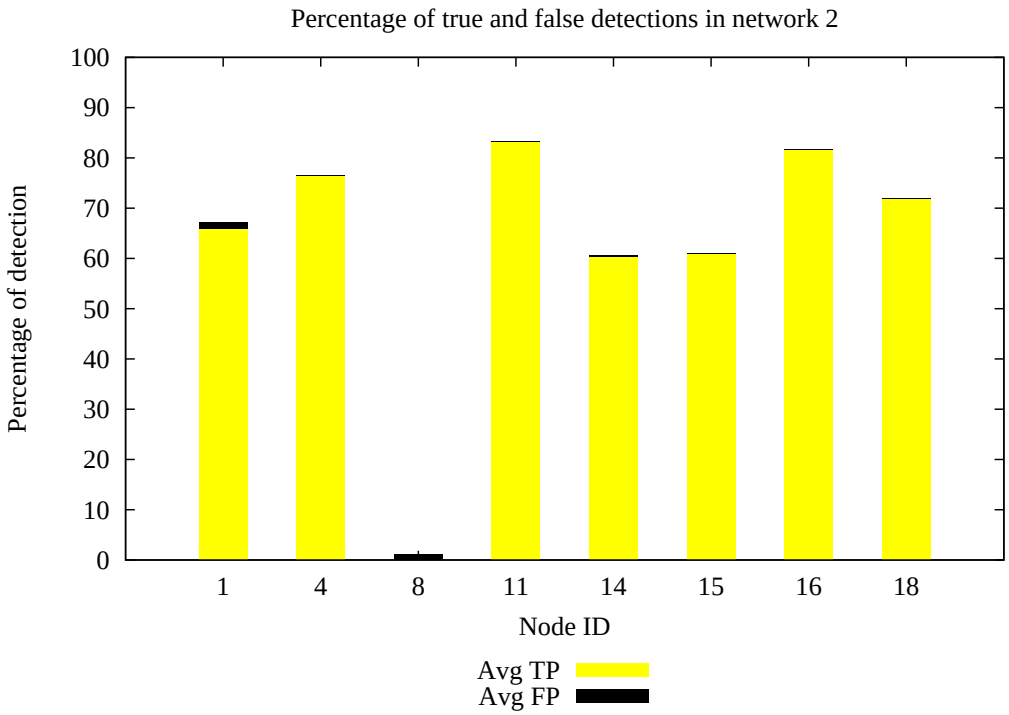


Figure 5.4: MintRoute Scenario 1: average detection in network 2, gap size = 1

The same behaviour is observed with larger sequence number gaps. However, there is an increase in the number of the sensor nodes that have FP. Since a larger sequence

number gap can affect the link qualities, link qualities that go under 100 are not advertised. Accordingly, a mismatch may occur between a sequence number gap trick and the appropriate received link quality to test. See Figure 5.5.

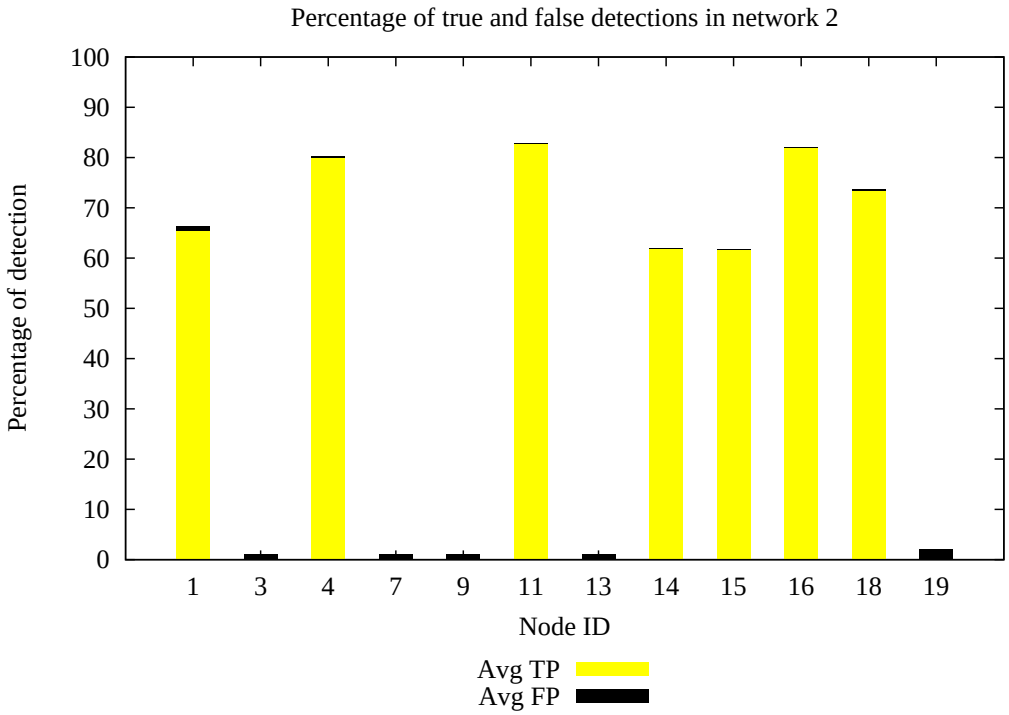


Figure 5.5: MintRoute Scenario 1: average detection in network 2, gap size = 5

5.4.3 Simulation of MintRoute Scenario 2

We present the results of the adaptive malicious node here. The malicious node tries to guess when the sequence number gap trick is being played, and tell the truth about its link quality in this case. It is then able to avoid detection by the sequence number gap trick some of the time.

Effectiveness of Malicious Node

The malicious node computes a threshold, τ, to represent the number of packets it expects to hear from each neighbour. Then it decides whether to lie or not to the

corresponding neighbour depending on the observed traffic relative to the threshold. In essence, if unusually many packets are "missing", the malicious node suspects that a trick is being played, and tells the truth in order to avoid detection. Figure 5.6 [125] shows the percentage of data delivery at the base station when the malicious node uses $\tau = E[X] + \sigma$, where $E[X]$ is the expected number of packets, and σ is the standard deviation.

With a sequence number gap of size 1, the simulation results show that the percentage of data delivery is at least 97% across the 10 WSN topologies. However, the percentage of data delivery drops sharply as the sequence number gap size increases. The malicious node can detect large sequence number gaps and thus, it lies less often. In other words, the sequence number gap trick is most effective with a small sequence number gap of size 1.

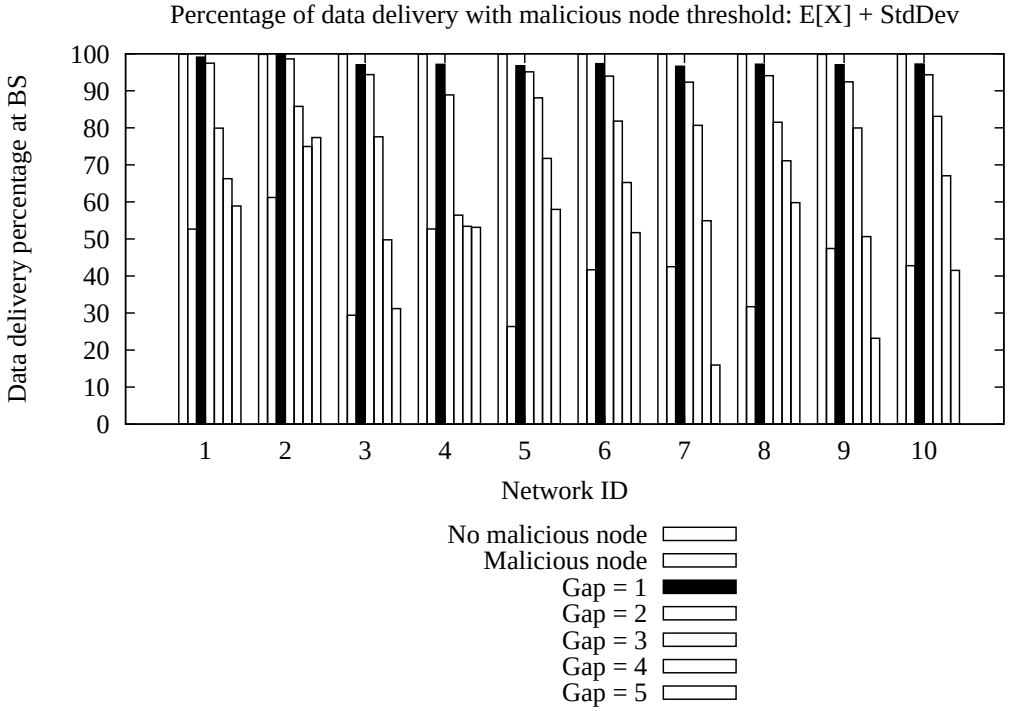


Figure 5.6: MintRoute Scenario 2: data delivery, $\tau = E[X] + \sigma$

Figures 5.7 and 5.8 show the simulation results with the malicious node using different

thresholds. It uses $\tau = E[X] + \frac{\sigma}{2}$ in the former graph. In the latter graph, it uses a threshold of `missed` $= 0$. The latter graph means that if a single packet is perceived missed, then the malicious node announces the true link quality for the corresponding neighbour. We can see that the tighter the threshold, the more successful the malicious node is in escaping the detection. However, the tighter threshold is not in favour of the malicious node because it will announce its real link quality values, which may not be attractive to its neighbours. In WSNs, losses are frequent and link qualities are weaker. Thus, if the malicious node does not exaggerate its link qualities, it may not look appealing to the neighbouring sensor nodes.

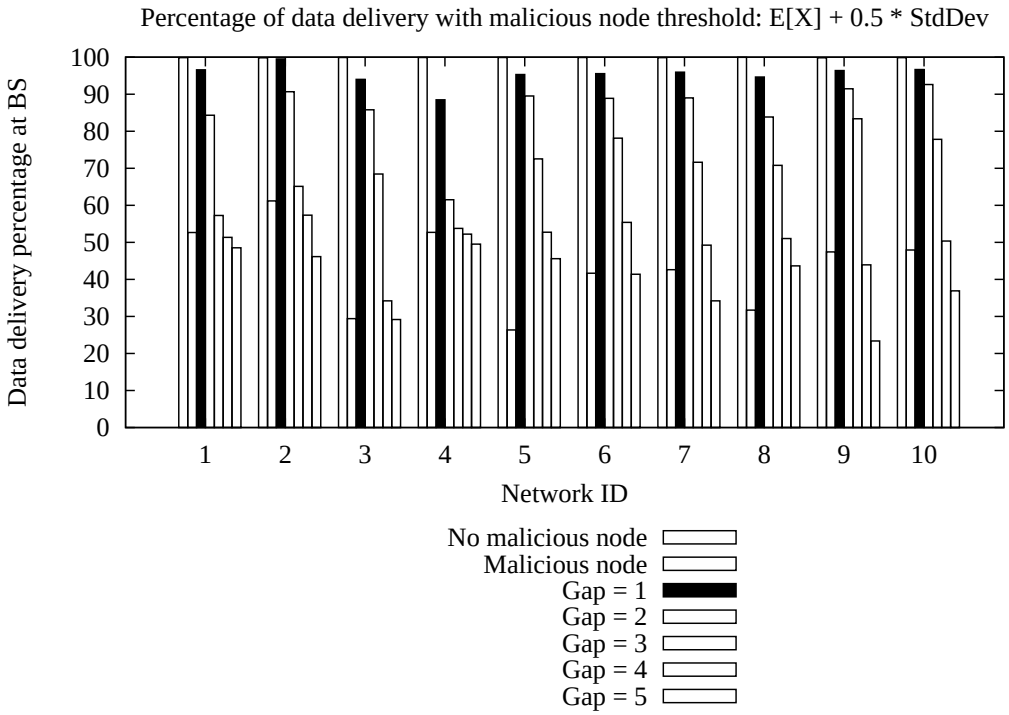


Figure 5.7: MintRoute Scenario 2: data delivery, $\tau = E[X] + \frac{\sigma}{2}$

Success of Detecting Malicious Node

The malicious node in Scenario 2 is more malignant and tries to guess when a neighbour is performing the sequence number gap trick hoping to escape detection. We show the

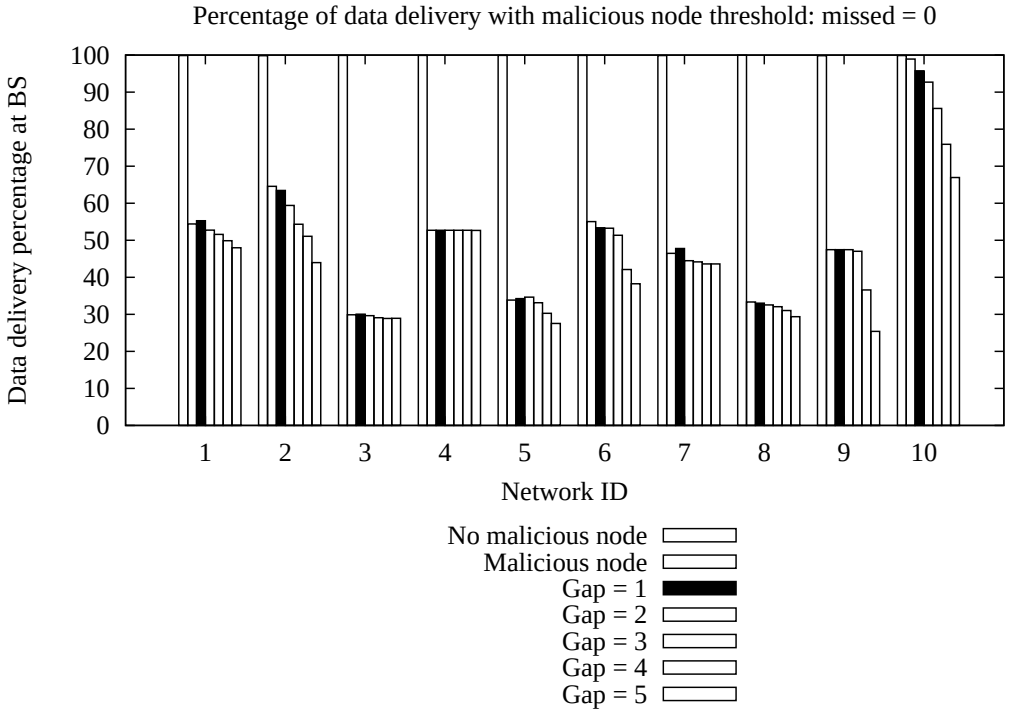


Figure 5.8: MintRoute Scenario 2: data delivery, $\tau = missed = 0$

performance of the sensor nodes against the different thresholds that the malicious node may use.

Figure 5.9 illustrates the behaviour of the sensor nodes when the malicious node uses the threshold $\tau = E[X] + \sigma$. We can see a decrease in the average number of TP, which means that the malicious node is successful in escaping the detection. On the other hand, the average number of FP stays almost the same since this is due to a miss from a good sensor node.

The malicious node is more successful in escaping detection when it uses a tighter threshold as shown in Figure 5.10. It can even escape detection completely with the tightest threshold as shown in Figure 5.11. However, as we said before, this tightest threshold may deprive the malicious node from luring its neighbours if its real link qualities are not good.

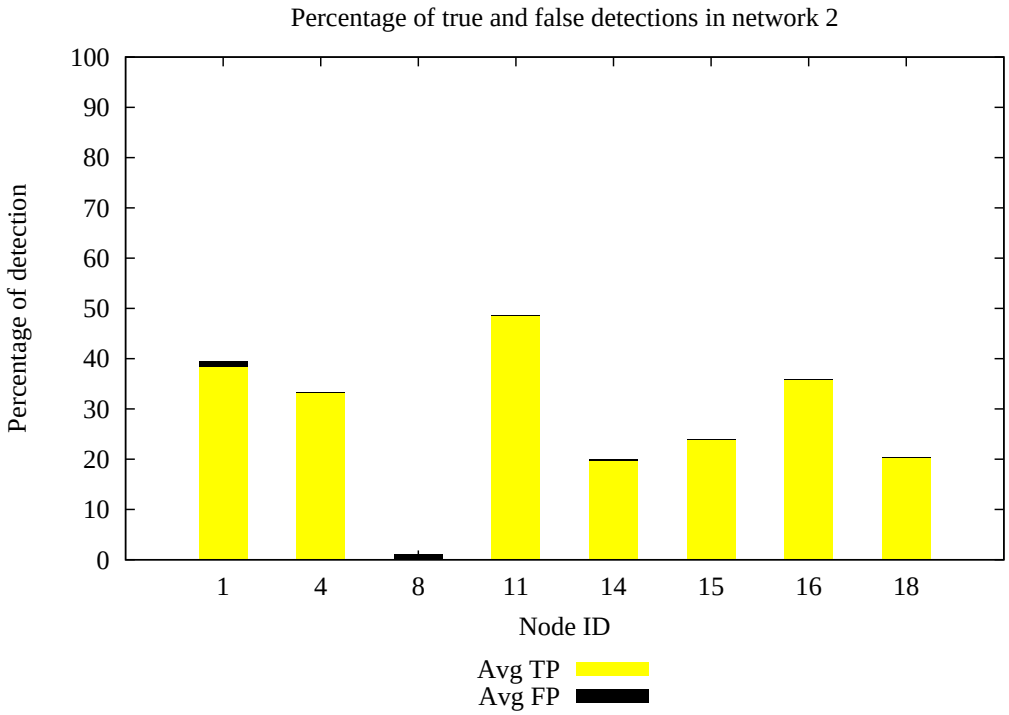


Figure 5.9: MintRoute Scenario 2: average detection in network 2, $\tau = E[X] + \sigma$

5.4.4 Simulation of MintRoute Scenario 3

Link quality computations in MintRoute use an exponentially weighted moving average. As a result, it is impossible for a link quality estimate to jump dramatically from one interval to the next. Knowing that the malicious node is not naive, the sensor nodes should consider this feature of MintRoute and be more vigilant.

In the previous scenarios, the sensor nodes test the malicious node's link qualities advertisement immediately following a sequence number gap trick, rather than continuously. An enhanced detection mechanism can monitor link qualities continuously to detect adaptive malicious nodes.

Effectiveness of Malicious Node

The simulation results for this scenario show that the enhanced detection mechanism works well. For a sequence number gap of size 1, Figure 5.12 [125] shows that the

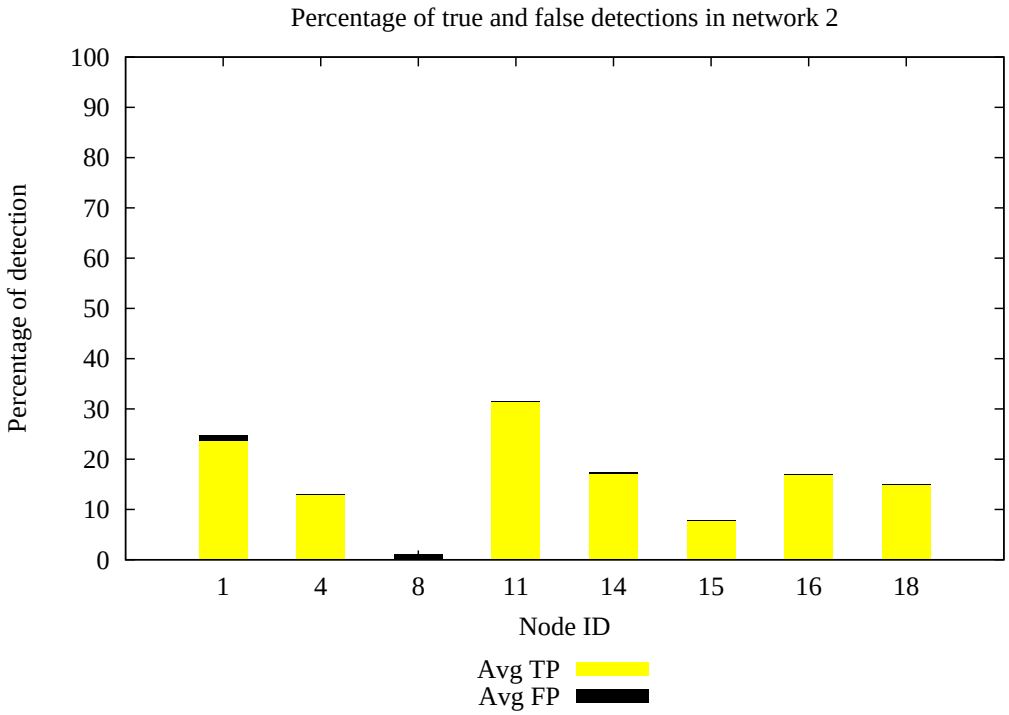


Figure 5.10: MintRoute Scenario 2: average detection in network 2, $\tau = E[X] + \frac{\sigma}{2}$

percentage of data delivery is about 98% across the 10 WSN topologies with malicious node's threshold $\tau = E[X]+\sigma$. Furthermore, the approach works even for larger sequence number gap sizes, though there is still a slight decline in effectiveness as the gap size increases.

Moreover, the vigilant sensor nodes are more successful in detecting the malicious node that uses tighter thresholds as illustrated in Figures 5.13 and 5.14. Since the malicious node goes back to exaggerating its link qualities after guessing the sequence number gap trick being performed, a sensor node that is always testing link qualities will detect this malicious behaviour.

The enhancement to the IDS mechanism in Scenario 3 can also be applied to the malicious node itself. That is, following a (detected) sequence number gap trick, the malicious node can cautiously increase its link quality back to the maximum so as to avoid

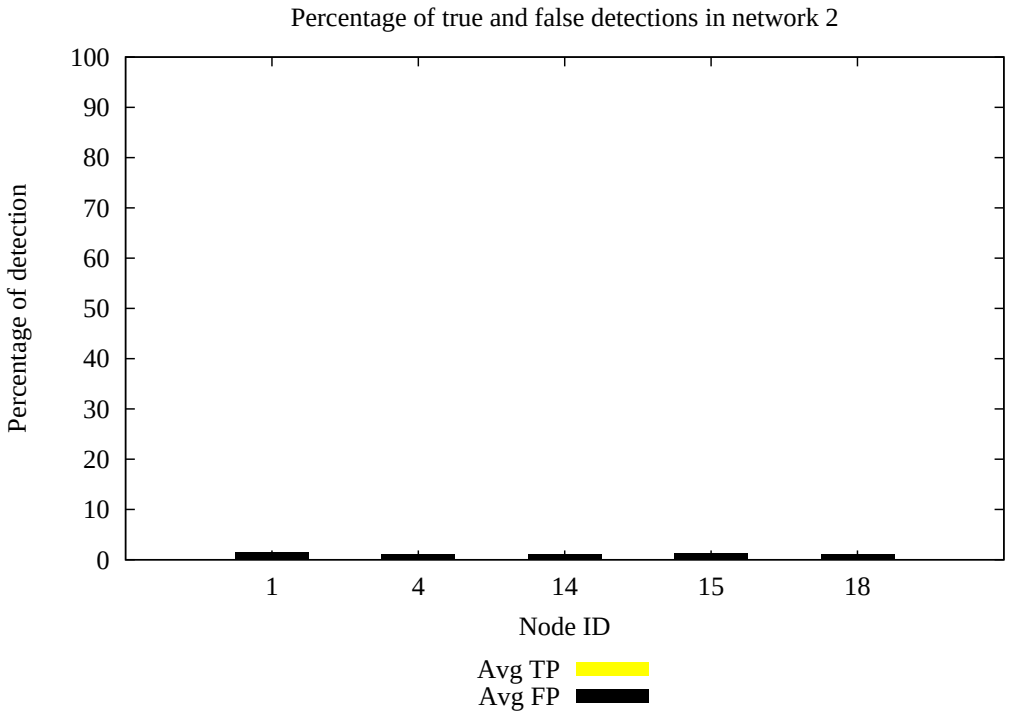


Figure 5.11: MintRoute Scenario 2: average detection in network 2, $\tau = missed = 0$

suspicion. This strategy has effectively converted the malicious node into a conformant node.

Success of Detecting Malicious Node

Figures 5.15, 5.16, and 5.17 show the improvements in detecting the malicious node when the sensor nodes test all the received link qualities. The latter figure shows that the malicious node is also detected when the sensor nodes do not play the sequence number gap trick. We can also see an increase in the number of false detections. This is due to testing at all times so a missed packet may cause a tricking sensor node to suspect its good neighbouring sensor node.

Although the low number of true detections do not confirm the high percentage of data delivery in Scenario 3, it is because of the blacklisting mechanism of our IDS and the high threshold of MintRoute to change to a new parent. It was computationally

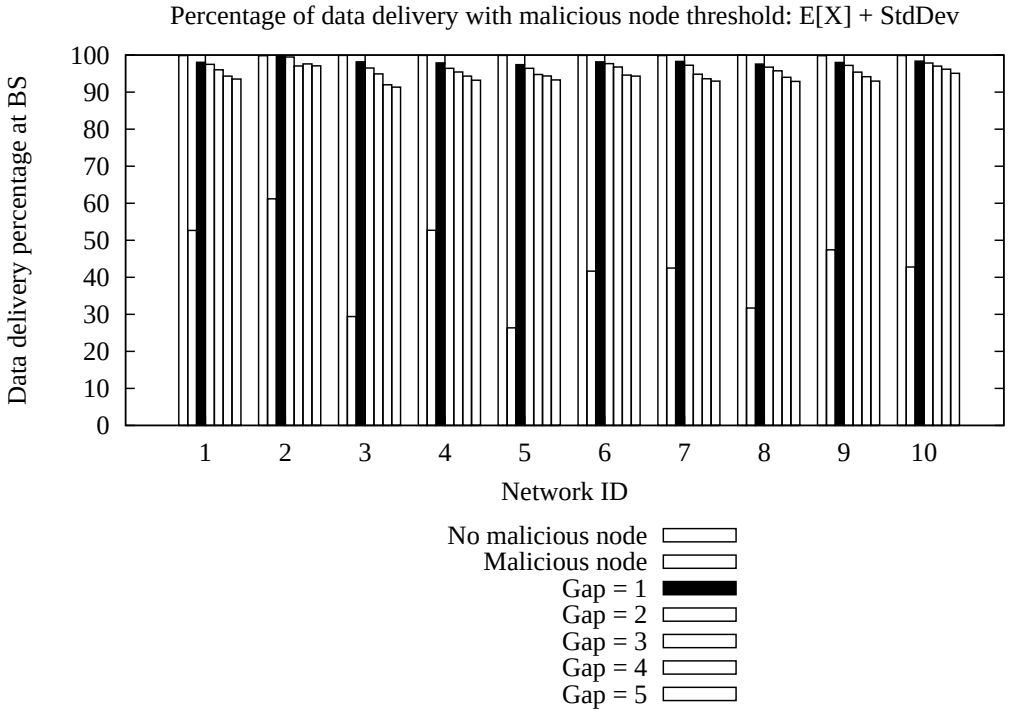


Figure 5.12: MintRoute Scenario 3: data delivery, $\tau = E[X] + \sigma$

found that a malicious node reaches the actual link qualities after advertising five true link qualities. Hence, we set the blacklist removal threshold to 16, which means once the malicious node is blacklisted it needs five true advertised link qualities to be removed. After the removal of the node from the blacklist, it needs to offer a route cost that is 75% (MintRoute default) better than the current cost to be considered as a next hop. Thus, once the malicious node is blacklisted, it is hard to choose it again as a next hop unless all other route costs are low.

5.5 Summary

In this Chapter, we explained the extensions to ns-2 to support MintRoute routing protocol and the proposed IDS for the vulnerability of cooperative link quality routing protocols for WSNs. Then the Chapter went through several scenarios where the malicious

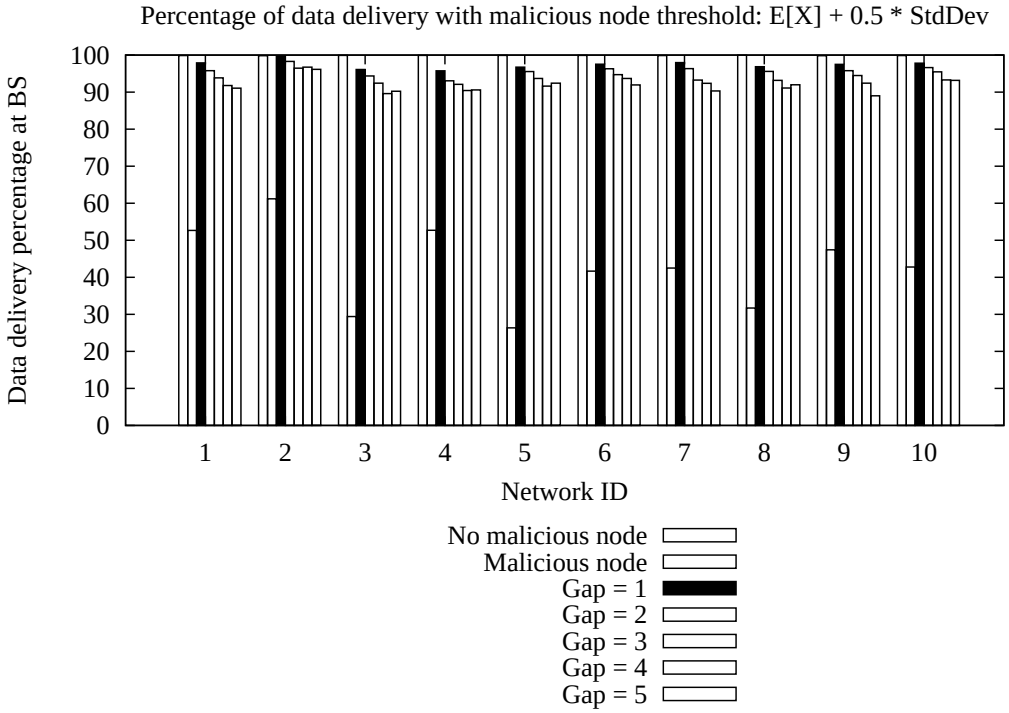


Figure 5.13: MintRoute Scenario 3: data delivery, $\tau = E[X] + \frac{\sigma}{2}$

node adapted its attack to be more vicious. The malicious node built different thresholds in different scenarios for the expected number of packets from each neighbouring sensor node to escape the detection. If the number of overheard packets exceeded the threshold, the malicious node would advertise the true link quality to the corresponding neighbouring sensor node. In each scenario, the IDS was adapted to detect the new behaviour of the malicious node. The IDS tested different sizes of the sequence number gap. The simulation results showed that the tighter the threshold of the malicious node, the lower the success of the IDS. However, smaller size of the sequence number gap were more helpful in detecting the malicious node.

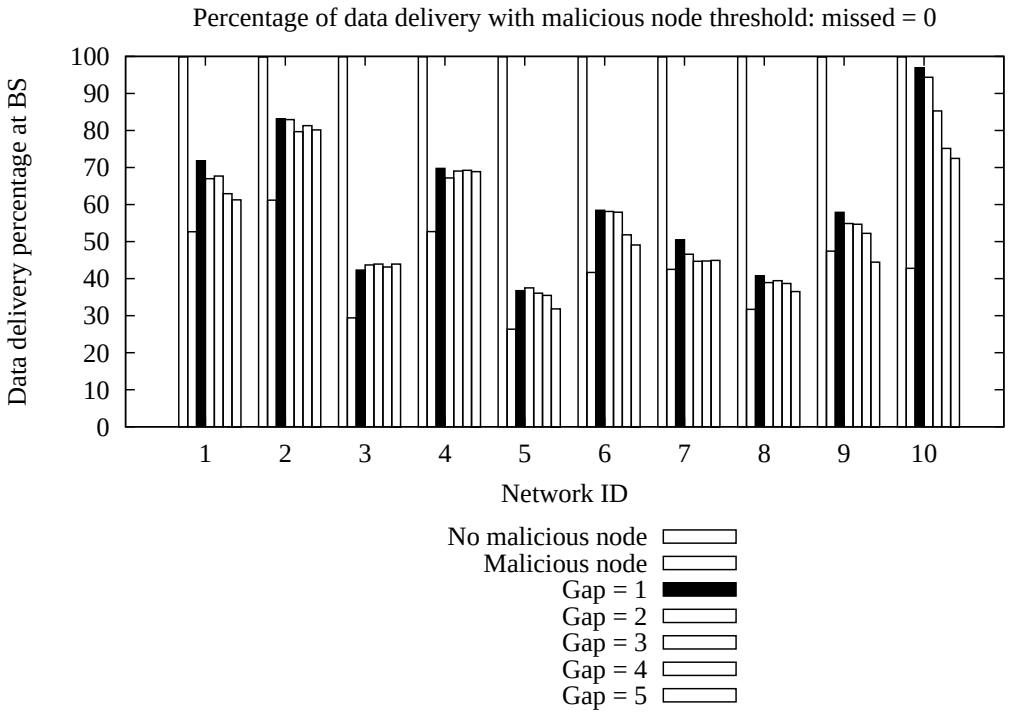


Figure 5.14: MintRoute Scenario 3: data delivery, $\tau = missed = 0$

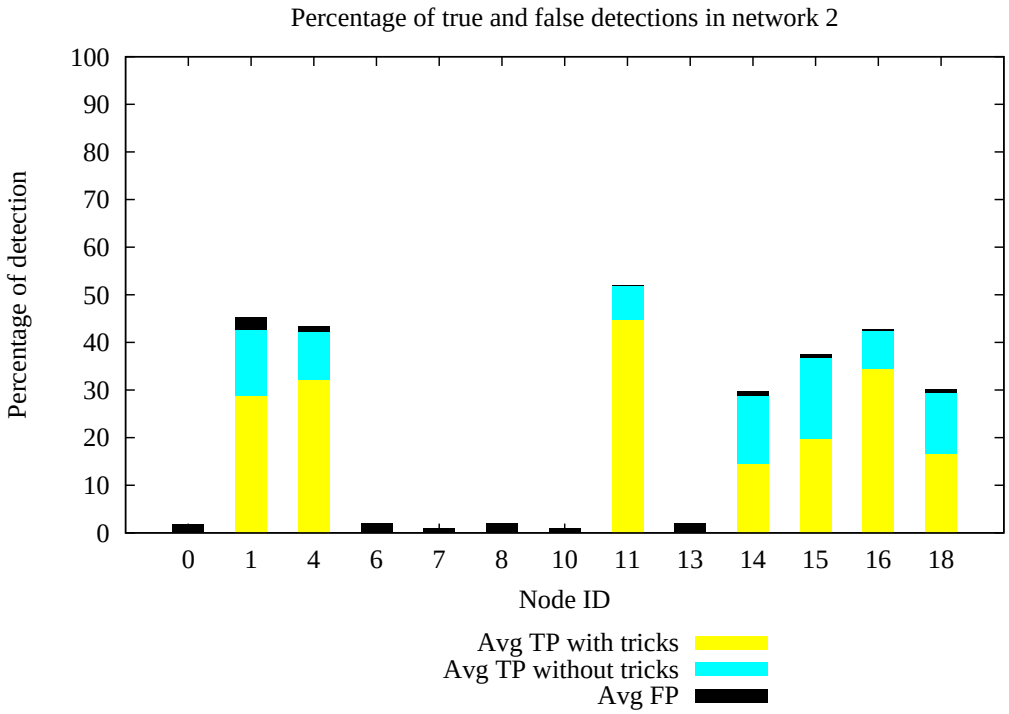


Figure 5.15: MintRoute Scenario 3: average detection in network 2, $\tau = E[X] + \sigma$

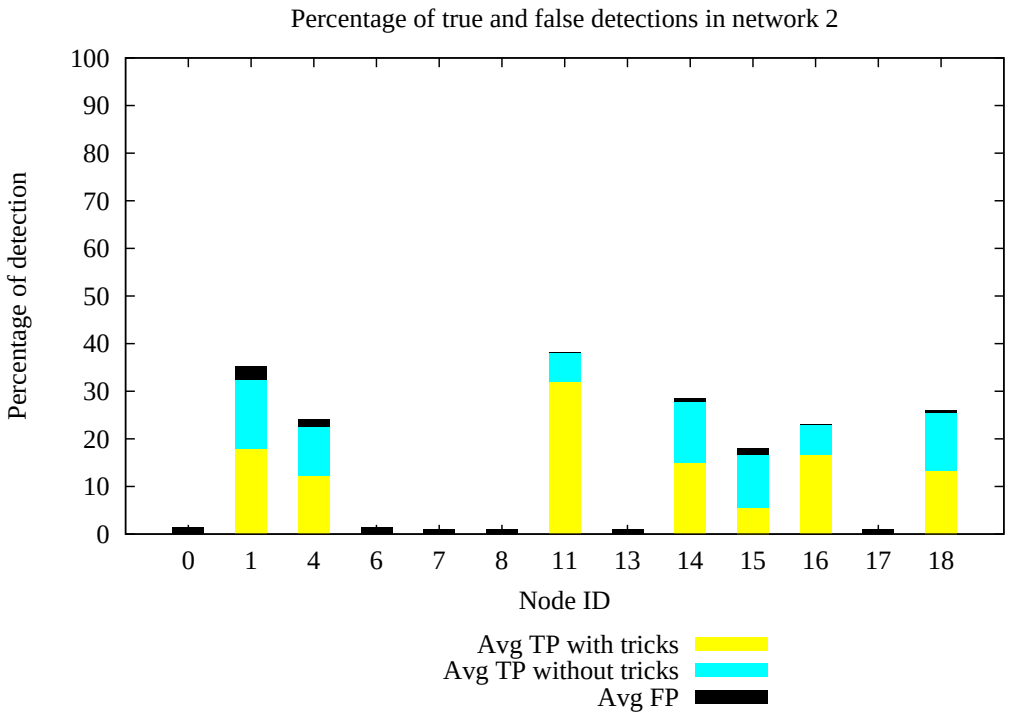


Figure 5.16: MintRoute Scenario 3: average detection in network 2, $\tau = E[X] + \frac{\sigma}{2}$

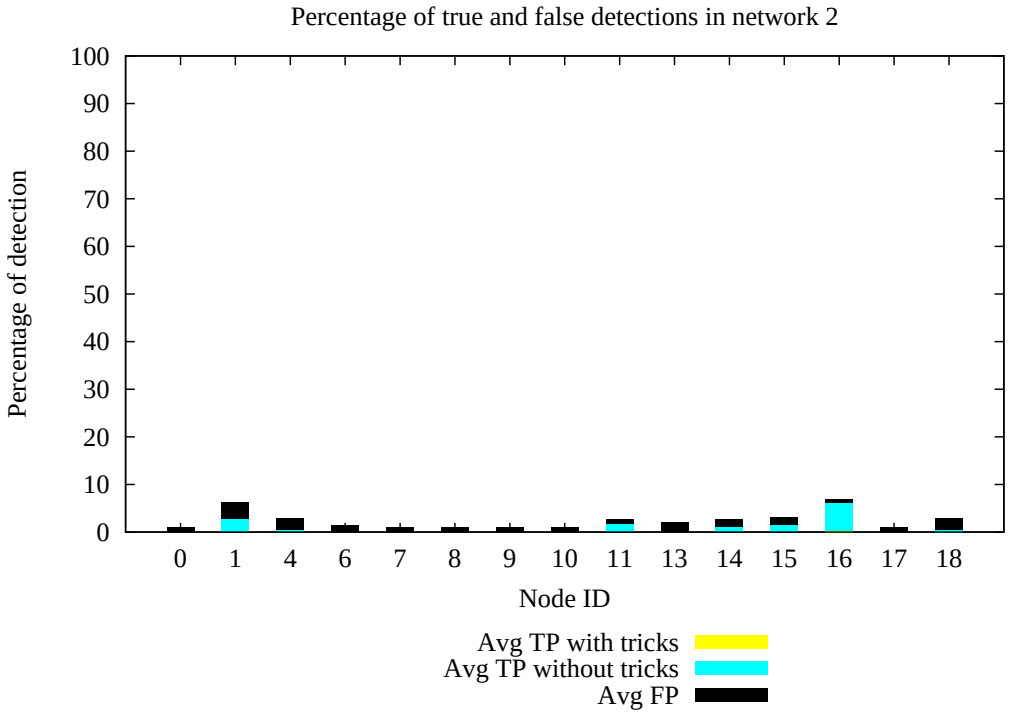


Figure 5.17: MintRoute Scenario 3: average detection in network 2, $\tau = missed = 0$

Chapter 6

COLLECTION TREE PROTOCOL AND ITS VULNERABILITIES

6.1 Introduction

This Chapter provides the details of CTP and the possible vulnerabilities related to link qualities that a malicious node can exploit. The Chapter begins by explaining the components of CTP and how they integrate together followed by a detailed example. Then we point out the vulnerabilities that exploit the computations of link qualities in CTP. With each vulnerability, the behaviour of the malicious node is explained. Finally, the Chapter ends with a proposed IDS to detect these attacks.

6.2 Modules of CTP

CTP is a link quality routing protocol that computes routes to a single or a small number of base stations in a WSN. It is a best-effort protocol that implements several mechanisms to improve data delivery but it does not guarantee 100% delivery [64]. The specification of CTP is provided in TinyOS Enhancement Proposal (TEP) 123 [128] and its implementation is available in the TinyOS 2.1 distribution. CTP is a data collection protocol that fulfills the primitives of data collection that are set in TEP 119 [129]:

- Estimate link quality of 1-hop links.
- Detect and repair routing loops.
- Detect and suppress duplicate packets.

Although it may look easy to fulfill the data collection primitives, not all collection protocols offer high data delivery ratios. The performance of the link quality protocols is affected by the instability of wireless links because link qualities may vary significantly and quickly over time. In addition, link estimation is often based on correctly received packets, which may introduce bias in the estimation. CTP addresses these problems and provides a high data delivery ratio to the base station [64, 130, 131].

CTP is a flat proactive routing protocol for WSNs. It is a hybrid link quality protocol, which means that it computes bidirectional link qualities for some sensor nodes and unidirectional link qualities for others. CTP computes bidirectional link qualities differently from MintRoute. In CTP, bidirectional link qualities are computed independently without any cooperation among the sensor nodes. The computations of CTP are performed in three modules: the link quality estimator module, the forwarding engine module, and the routing engine module. The three modules reside in the network layer of the protocol stack. The three modules interact together and with other layers through well-defined interfaces.

Figure 6.1 shows a conceptual view of the interaction between the three modules. The link quality estimator computes link quality estimates between neighbouring sensor nodes. The forwarding engine is responsible for sending and forwarding data packets and receiving ACKs in the WSN. Sent/Forwarded data packets and received ACKs are passed to the link estimator to update the outbound link qualities. The routing engine manages transmission and reception of beacons (route update packets). The routing engine passes the received beacons to the link estimator module to update the inbound link qualities. It also accepts link quality estimates from the link quality estimator.

Sensor nodes implementing CTP exchange beacons to construct routing trees. They also relay data packets to report collected data to the base station. In addition, CTP relies on receiving ACKs to indicate the successful reception of data packets.

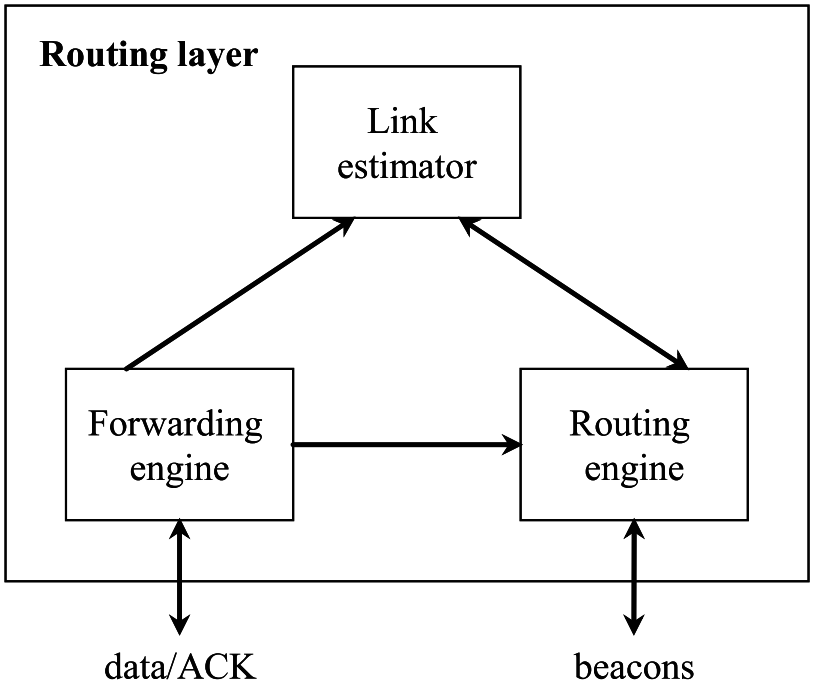


Figure 6.1: Modules of CTP

6.2.1 Link Quality Estimator

CTP uses the 4-bitle as its default link quality estimator. 4-bitle is a hybrid estimator that uses active and passive monitoring at the sending sensor node. Active monitoring requires the sensor nodes to broadcast periodic beacons. As a result of this broadcast, sensor nodes receive routing information from all their neighbouring sensor nodes. Thus, 4-bitle computes inbound link qualities for all the neighbouring sensor nodes using the received beacons. Unlike active monitoring, passive monitoring infers statistics about the links from the packets being sent on these links. Since CTP requires the transmission of ACKs to acknowledge the successful reception of data packets, 4-bitle computes outbound link qualities using the received ACKs. Since the sensor nodes transmit data packets to a chosen next hop, the outbound link quality is computed only for the next hop. Accordingly, 4-bitle combines the inbound and outbound link qualities for next hop into a bidirectional link quality. All other neighbouring sensor nodes have a unidirectional

link quality, the inbound link quality.

4-bitle separates the computations of inbound and outbound link qualities in two modules, Beacon Link Quality (BLQ) and Data Link Quality (DLQ), respectively, as shown in Figure 6.2. The BLQ converts the number of received and missed beacons into inbound link qualities for the sending sensor node using the inverse of WMEWMA. To determine the number of missed packets, CTP requires the use of sequence numbers in the beacons. The DLQ uses the number of transmitted packets to compute outbound link qualities for the sending sensor node using RNP. The outputs of BLQ and DLQ are combined together into one value using WMEWMA. If the WSN has high traffic volume, then DLQ will dominate the estimates. Otherwise, BLQ will dominate [121].

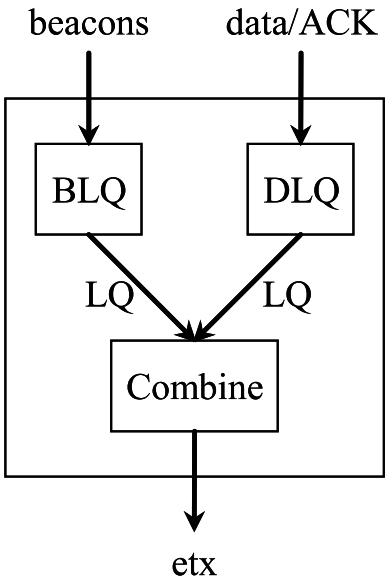


Figure 6.2: Components of link quality estimator

4-bitle has a neighbour table that holds information of 10 neighbouring sensor nodes. Each entry in the table occupies 11 bytes, requiring 110 bytes to save the whole table. The fields of each entry are described in Table 6.1. Beacons delivered to the link estimator from the routing engine update the `lastSeq`, `rcvCnt`, and `failCnt` fields. The `lastSeq` is updated with the sequence number of the last received beacon after all other

fields are updated. The `rcvCnt` field is incremented with every received beacon from the corresponding neighbour. The `failCnt` field is incremented if the difference between the new sequence number and the `lastSeq` is greater than one. The BLQ component uses the fields `lastSeq`, `rcvCnt`, `failCnt`, and `inQuality` to compute the inbound link qualities. Meanwhile, `dataTotal` and `dataSuccess` fields are updated when the forwarding engine passes the data packets and ACKs to the link estimator, respectively. The `dataTotal` field is incremented whenever a data packet is transmitted and the `dataSuccess` field is incremented whenever an ACK is received. The DLQ accesses the `dataSuccess` and `dataTotal` fields to compute the outbound link qualities. The `etx` field holds the combination of both the inbound and outbound link qualities of the corresponding neighbour.

Table 6.1: The fields in the neighbour table of CTP link estimator

Field	***Description***	***Size***
ll_addr	identifier of neighbour	2 bytes
lastSeq	sequence number of last received beacon	1 byte
rcvCnt	number of beacons received after last BLQ update	1 byte
failCnt	number of beacons missed after last BLQ update	1 byte
flags	state of this entry	1 byte
inQuality	inbound quality in the range [1, 255]. 1 bad, 255 good	1 byte
etx	quality of the link	2 bytes
dataSuccess	number of successful data packets sent after last DLQ update	1 byte
dataTotal	number of transmission attempts after last DLQ update	1 byte

The DLQ component computes the outbound link quality for every five data packets sent or forwarded to the corresponding neighbour. The BLQ component computes the inbound link quality for every three beacons sent from the corresponding neighbour. Sensor nodes may miss packets from their neighbours due to collision or environmental factors. Algorithms 6.1 and 6.2 show the steps for computing DLQ and BLQ, respectively.

For the outbound link quality computations in DLQ, if s out of t data packets are acknowledged, then the outbound link quality will be $\frac{t}{s}$. If $s = 0$, then the outbound link quality will be the number of failed transmissions since the last successful transmission.

Algorithm 6.1 DLQ()

if `dataTotal` ≥ 5 **then**
  if `dataSuccess` $= 0$ **then**
    SET q TO $10 \times$ `dataTotal` ▷ number of failed transmissions
  else
    SET q TO $\frac{10 \times \texttt{dataTotal}}{\texttt{dataSuccess}}$ ▷ RNP value
    SET `dataSuccess` TO 0
    SET `dataTotal` TO 0
  SET `etx` TO $\frac{9 \times \texttt{etx} + q}{10}$

The computation of inbound link qualities is analogous but it is smoothed with the inverse of a WMEWMA with $\alpha = 0.9$ [121]. We can see that both computations, DLQ and BLQ, are scaled to avoid floating point operations, which are expensive in resource-limited devices.

4-bitle follows the Link Estimation Exchange Protocol (LEEP) [132] packet format. LEEP provides a standard format for beacons or route update packets that link quality protocols should use. Thus, the routing engine sends the generated routing engine packet to the link estimator to encapsulate it in a LEEP packet to form the beacon packet. Figure 6.3 shows the fields of a CTP beacon along with the size of each field in bytes. Following is the description of each field [132]:

- *header*: The header of the packet contains the *num* and *seq* fields. The *num* holds the number of link information entries in the trailer of the packet. The *seq* is the sequence number of the packet. This number is incremented with every new beacon.

- *payload*: This is usually the routing information received from the routing engine.

- *trailer*: The trailer contains multiple entries from the neighbour table. Each entry in the trailer holds the `inQuality` value, lq, of a neighbouring sensor node, *node id*

Algorithm 6.2 BLQ()

SET total TO `rcvCnt` + `failCnt`
if total ≥ 3 **then**
  SET newEst TO $\frac{250 \times \texttt{rcvCnt}}{total}$
  SET `inQuality` TO $\frac{9 \times \texttt{inQuality} + newEst}{10}$
  SET q TO $\frac{2500}{\texttt{inQuality}}$
  if q > 250 **then**
    SET q TO 65535 ▷ largest possible value
  SET `rcvCnt` TO 0
  SET `failCnt` TO 0
SET `etx` TO $\frac{9 \times \texttt{etx} + q}{10}$

field. A single CTP beacon may hold up to n entries that bring the total size of the beacon to the maximum size allowed by TinyOS. TinyOS version 2 allows packets of at most 28 bytes. Since the header occupies two bytes, the payload occupies five bytes, and one entry occupies three bytes, a single CTP beacon can hold up to seven link information entries. Although CTP does not include the exchanged `inQuality` values in the computation of link qualities, 4-bitle still adds them to the beacons to conform with LEEP.

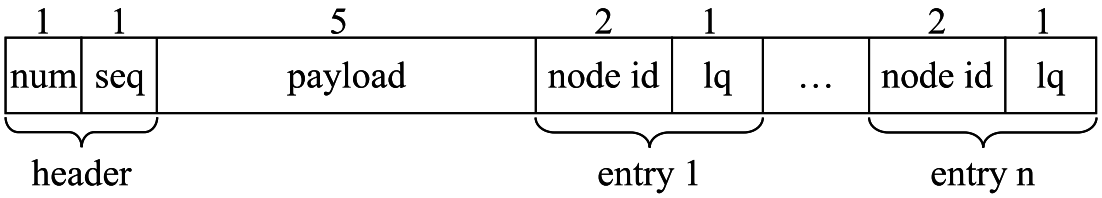


Figure 6.3: Format of CTP beacon

Since CTP computes outbound and inbound link qualities differently, it is important to study the behaviour of both link qualities. Figures 6.4 and 6.5 show the values for

`etx` computed manually for a WSN of two sensor nodes. In a stable WSN, the DLQ component produces a constant `etx` value of one, as shown in Figure 6.4. A stable WSN means a network with no collisions or packet losses where the routing tree is not changed frequently. Each step of DLQ requires at least five data packets to be transmitted. On the other hand, Figure 6.5 shows an example of `etx` value if the link estimator uses BLQ only. In a stable WSN, it will take BLQ 25 computations to converge to the minimum value of `etx` which is 10. Each computational step of the BLQ component requires at least three beacons to be perceived (missed or received) with a total of 75 beacons to reach the minimum `etx`. Combining BLQ and DLQ helps `etx` to converge more quickly to the minimum value.

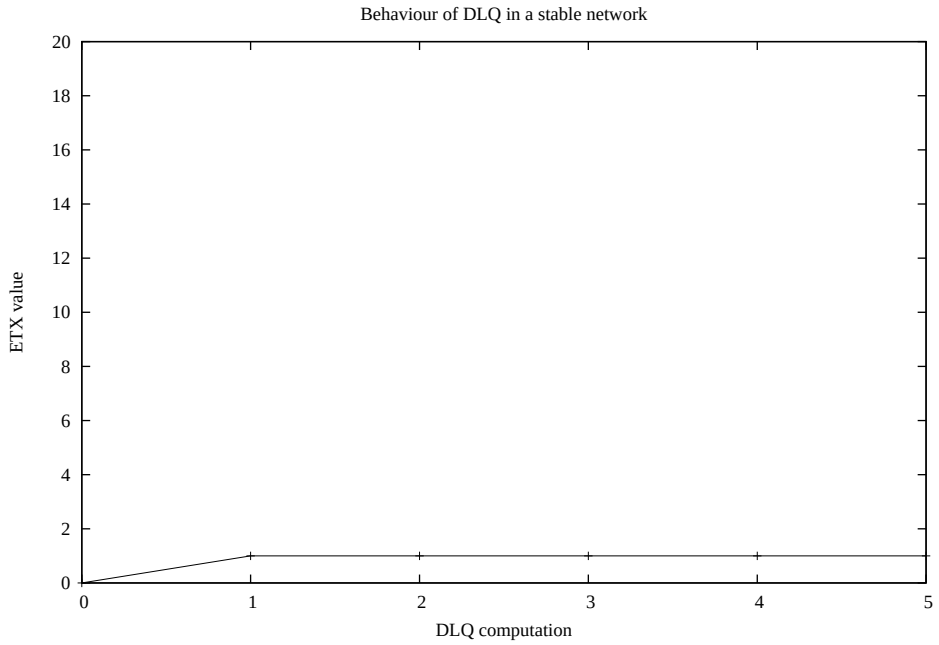


Figure 6.4: etx values with the DLQ component only

6.2.2 Routing Engine

The routing engine establishes the routing tree by choosing the best routes to the base station. The best route is the one with the lowest path cost. Path cost is interpreted as

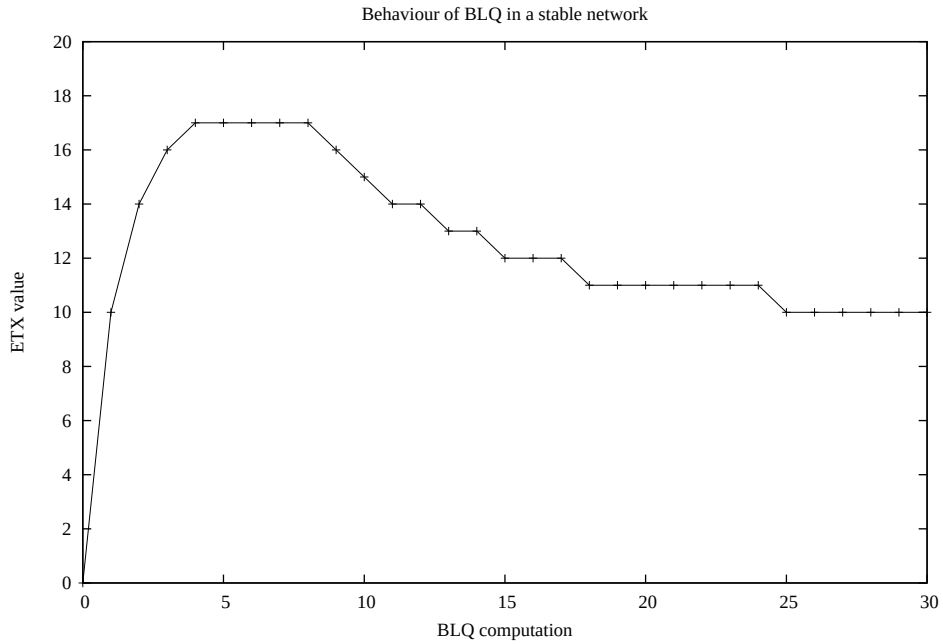


Figure 6.5: etx values with the BLQ component only

the sum of the `etx` values of all the links along the path to the base station. To compute the path cost of a sensor node, the routing engine adds the `etx` value of the link to a neighbouring sensor node computed by the link estimator to the path cost announced by this neighbour. Sensor nodes choose the path of the neighbouring sensor node offering the lowest path cost. The routing engine broadcasts the path cost and the next hop adaptively in beacons. The routing engine holds routing information of 10 neighbouring sensor nodes in a routing table.

Routing table

The routing engine maintains a routing table that holds the path cost values of some of the neighbouring sensor nodes. Table 6.2 shows the fields of the routing table. It is important for the routing engine to know the parents of the neighbouring sensor nodes so that the routing engine does not choose a child sensor node as the next hop. The `cost` value saved in the routing table is the path cost announced by the corresponding

neighbour in its beacons. The routing engine adds this value to the `etx` value in the neighbour table of the link estimator to determine the best route to the base station. The `haveHeard` flag is updated whenever a beacon is heard from the corresponding neighbour. This field is important to keep the routing table updated with potential next hop sensor nodes. If a sensor node does not receive beacons from a neighbouring sensor node, it shall remove this neighbour from the table so as not to consider it for a next hop choice. The `congested` field is set if a received data packet or a received beacon has the `C` bit set. This means that the route through the corresponding neighbour is congested and the neighbour cannot handle any more traffic. Thus, the routing engine will not consider the corresponding neighbour in the route choice process.

Table 6.2: The fields in the routing table of CTP routing engine

Field	***Description***	***Size***
neighbour	identifier of neighbour	2 bytes
parent	identifier of parent of neighbour	2 bytes
cost	path cost of this neighbour	2 byte
haveHeard	flag to indicate that beacons are received from neighbour	1 byte
congested	flag to indicate that route through neighbour is congested	1 byte

It is important to synchronize the routing table of the routing engine with the neighbour table of the link estimator. Otherwise, the routing table may hold entries for neighbouring sensor nodes that the link estimator has decided to evict due to poor link qualities. Thus, the link estimator shall update the routing engine if it evicts any neighbour from its neighbour table. In addition, the routing engine informs the link estimator of the current next hop to pin it in the neighbour table.

The routing engine generates packets in the format shown in Figure 6.6. The following is an explanation of each field in the packet:

- P: Pull bit. When the P bit is set, it means that the routing engine is requesting the routing information of the neighbouring sensor nodes as soon as possible.

- C: Congestion bit. If the forwarding engine has no more space in the send queue, it will inform the routing engine to set the C bit in the next beacon. Since the beacons are broadcast, it is more convenient to set the C bit in the beacons. This informs all the neighbours of the status of the send queue to change their routes to an uncongested one.

- *parent*: This field holds the ID of the next hop of this sensor node.

- *cost*: This field holds the value of the path cost of this sensor node.

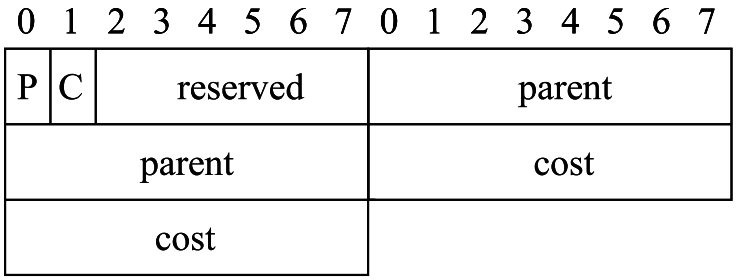


Figure 6.6: Format of CTP routing engine packet

Adaptive Beacons

Routing protocols of WSNs typically broadcast beacons or route update packets at fixed interval [7]. Having a small interval updates the routing information frequently and eliminates routing loops quickly but it uses more bandwidth and energy. Large update intervals can let topological inconsistencies persist for long periods of time [64]. The routing engine of CTP applies an adaptive beaconing strategy to achieve both fast recovery and low bandwidth and energy usage. The routing engine implements the Trickle algorithm [133] to broadcast beacons.

Trickle's purpose is to propagate code in a wireless network reliably and efficiently. It uses a randomized timer to broadcast a code summary to the local neighbourhood during an adaptive transmission interval. On top of this randomized timer, Trickle adopts two

other mechanisms: suppression of transmission, and adaptation of transmission interval. If a sensor node hears a code summary that is identical to its own code summary, it will suppress its own transmission to eliminate the propagation of redundant code. When the transmission interval expires, the sensor node doubles its interval up to a maximum value τ_{max}. On the other hand, when the sensor node hears a code summary that is older than its own code summary, it shrinks its transmission interval to a minimum value τ_{min} so that its newer code propagates quickly in the WSN.

Unlike Trickle, there is no global code or routing metric to share among the sensor nodes in a CTP WSN. Thus, the routing engine of CTP eliminates the suppression mechanism of Trickle but it implements the adaptive transmission interval for sending beacons. The routing engine sets τ_{min} to 128 time units and τ_{max} to 512,000 time units. The routing engine resets the transmission interval to τ_{min} in three cases [64, 131, 134]:

- When the forwarding engine receives a data packet to forward with a lower path cost, it assumes the existence of topology inconsistency or a possible routing loop. The forwarding engine sends a request to the routing engine to broadcast a beacon as soon as possible.

- When the routing engine finds a new route that has a significantly lower path cost than the current value, it shrinks its beacon transmission interval. Thus, a beacon is sent as soon as possible to update the neighbouring sensor nodes of the new value. A significant drop in the path cost may mean that this sensor node has a better route to the base station. Resetting the beacon transmission interval propagates this information to the neighbouring sensor nodes quickly. Significant means having a path cost value that is 20 points lower than the current value.

- When a packet is received with the P bit set, it means that the sending sensor node has joined the network recently and it requests topology information to populate

its routing and neighbour tables. The routing engine cancels the current beacon transmission interval and schedules a new minimum interval to update the new sensor node as soon as possible.

The operations of the routing engine are controlled by a timer. This timer is fired twice: at the beginning of a beacon transmission interval, and at random during the interval. Figure 6.7 shows how this timer works. At the beginning of the beacon transmission interval, τ_b, the timer is fired and it calls the functions `decayInterval()` and `chooseAdvertiseTime()`, D and C respectively. The `decayInterval()` function computes the new beacon transmission interval by doubling the old interval. When the beacon transmission interval reaches the maximum interval, the new intervals use the maximum value without doubling. The `chooseAdvertiseTime()` function computes a random time during the beacon transmission interval to broadcast the beacon. The random time is always computed in the second half of the beacon transmission interval. Having a silent period in the first half of the beacon transmission interval eliminates the short-listen effect [133]. Eliminating the short-listen effect helps in suppressing the propagation of redundant code. Although this suppression is not implemented in the current version of CTP, its developers intend to implement it in future versions. They believe that after a few number of neighbouring sensor nodes advertise their costs no better routes can be found to the base station.

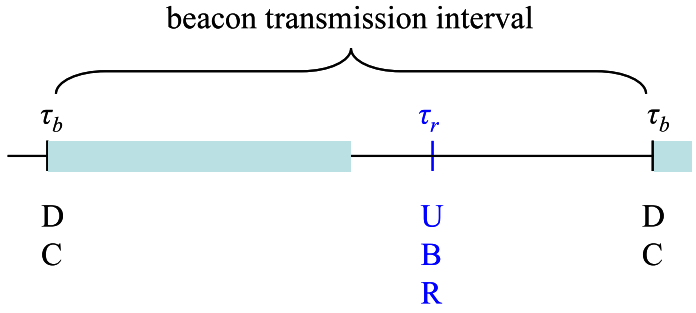


Figure 6.7: Beacon transmission intervals of CTP

When the timer fires during the beacon transmission interval, τ_r, at random, it calls the functions `updateRouteTask()`, U, `sendBeaconTask()`, B, and `remainingInterval()`, R. If it is the first call to the function `updateRouteTask()`, the routing engine will choose, as the next hop, the neighbouring sensor node that offers the lowest path cost to the base station. Otherwise, if a next hop is chosen before, then the routing engine will determine whether to change it or not. This decision depends on the difference between the new path cost and the old path cost. The `sendBeaconTask()` function prepares a beacon for broadcast. However, this preparation takes place only if the sensor node has a path to the base station. The function prepares the beacon by filling some of the fields of the beacon from the routing table. Then it passes the beacon to the link estimator to add the header and footer of the link estimator as explained in Section 6.2.1. Finally, the `remainingInterval()` is called to compute the remaining time until the next beacon transmission interval. At the end of the remaining interval, the timer is fired again at τ_b and the sequence of operations begins again for the new interval.

6.2.3 Forwarding Engine

The forwarding engine is the top level interface of CTP to the upper layers of the protocol stack of WSNs. It is responsible for the transmission of data packets, suppression of duplicate packets, detection of loops, and timing of transmissions [129]. The forwarding engine accepts data packets from the application layer and enqueues them for transmission. It also accepts data packets from neighbouring sensor nodes for forwarding. The forwarding engine maintains a send queue to buffer data packets ready for transmission and a transmit cache to buffer successfully transmitted data packets.

Transmission of Data Packets

Before transmission, the forwarding engine asks the routing engine about the next hop and the path cost to embed in the data packets. If there is no route to the base station,

the forwarding engine will suppress the transmission. Once a path is found to the base station, the routing engine informs the forwarding engine to resume transmission of data packets.

Suppression of Duplicate Data Packets

The forwarding engine removes a sent/forwarded data packet from the send queue and adds it to the transmit cache when an ACK is received. The transmit cache is important in case duplicate data packets arrive slower than the draining rate of the send queue. A duplicate data packet may be received again if the sending sensor node misses the ACK from the forwarding sensor node. When the forwarding engine receives a data packet to forward, it checks if this data packet has been received before. There are two places to search for duplicate data packets: the send queue, and the transmit cache. The data packet would be in the send queue if the forwarding engine did not send it yet or an ACK is not received from the receiving node. The transmit cache holds data packets that are sent and acknowledged. It was found empirically that a transmit cache of size four is sufficient to suppress most of the duplicates [64].

Detection of Routing Loops

Routing loops are possible in WSN. A routing loop occurs when the new route includes a descendant sensor node [128]. There are two cases where routing loops can occur. First, routing loops occur when a sensor node chooses a route that has a significantly higher path cost than its older route. A sensor node may choose this route in response to losing connectivity with its previous parent. Second, the forwarding engine receives a data packet with a path cost that is smaller than its own. It will assume the sending sensor node has stale routing topology information and a routing loop may exist [64]. The forwarding engine detects routing loops by comparing the ID of the originating sensor node and the sequence number of a newly received data packet with the data packets in

the send queue and the transmit cache.

Transmission Timer

Finally, a timer controls the transmission of data packets from the send queue. When the forwarding engine transmits a data packet, it waits for a short period of time to receive an ACK. If an ACK is not received within the allotted time, then the forwarding engine will retransmit the data packet. The forwarding engine will retransmit an unacknowledged data packet at most 32 times before it finally decides to discard it.

6.3 Example of the Operations in CTP

As mentioned in the previous section, CTP computes an inbound link quality using active monitoring of beacons and computes an outbound link quality using passive monitoring of data packets and ACKs. The two link qualities are combined into a bidirectional link quality. Next, we give the details of how the CTP modules cooperate to compute the link qualities.

Suppose that two sensor nodes, A and B, are neighbours. Assume that A is the parent sensor node and B is the child sensor node. We show how the link estimator and the routing engine of sensor node B build the neighbour and routing tables, respectively. Table 6.3 shows an excerpt of the evolution of the entry of A in the neighbour table of B. The table shows the relevant fields in the computations of the `etx` value, namely, `lastSeq`, `rcvCnt`, `failCnt`, `dataSuccess` as $dataS$, `dataTotal` as $dataT$, `etx`, and `inQuality`. For ease of explanation, column s represents the sequence of actions taken by the link estimator of sensor node B. The *action* column represents the processing of sensor node B. The following explains each action:

- *init*: This is the initial state of any entry in the neighbour table. All the fields are initialized to 0 except `lastSeq` because the sequence numbers start at 0.

- b_A: This is a beacon received from sensor node A. With each received beacon, the routing engine passes the beacon to the link estimator to update the fields `rcvCnt` = `rcvCnt` + 1, `failCnt` = $seqNo$ - `lastSeq` - 1, and `lastSeq` = $seqNo$, where $seqNo$ is the sequence number of the received beacon. The first three beacons update `rcvCnt`, as seen in steps 1, 2, and 3.

- *BLQ*: The BLQ component computes the `etx` and inbound link quality, `inQuality`, values whenever `rcvCnt` + `failCnt` ≥ 3. At step 4, the BLQ computes the values 10 and 25 for `etx` and `inQuality`, respectively, using Algorithm 6.2.

- *data*: The data action means that sensor node B sent a data packet to its next hop, sensor node A in our example. When a data packet is sent, the forwarding engine updates the link estimator to increment the `dataTotal` field, as shown in steps 5, 11, and n-2.

- *ACK*: The ACK means that sensor node A received the data packet from sensor node B and sent back an ACK. When the forwarding engine of sensor node B receives the ACK, it updates the link estimator to increment the `dataSuccess` field, as shown in steps 6, 13, and n-1. The ACK in CTP is sent in a stop-and-wait fashion. This means that a sensor node sends a data packet and waits for its ACK before sending another data packet. However, this may lead to a deadlock if the ACK is lost or the data packet itself did not reach the next hop. To solve this deadlock, the forwarding engine tries to transmit unacknowledged data packets up to 32 times. This transmission is transparent to the link estimator, which means that with every transmission the link estimator updates the `dataTotal` field regardless of a new transmission or a retransmission.

- *DLQ*: this action means that the link estimator calculates the outbound link quality from the sender, sensor node B, to the receiver, sensor node A. For every five data

packets transmitted, Algorithm 6.1 is called to compute the outbound link quality, as shown in step n.

Table 6.3: Example of CTP neighbour table

s	***action***	***lastSeq***	***rcvCnt***	***failCnt***	***dataS***	***dataT***	***etx***	***inQuality***
0	init	-1	0	0	0	0	0	0
1	b_A	0	1	0	0	0	0	0
2	b_A	1	2	0	0	0	0	0
3	b_A	2	3	0	0	0	0	0
4	BLQ	2	0	0	0	0	10	25
5	data	2	0	0	0	1	10	25
6	ACK_A	2	0	0	1	1	10	25
7	b_A	3	1	0	1	1	10	25
8	b_A	5	2	1	1	1	10	25
9	BLQ	5	0	0	1	1	15	39
10	b_A	6	1	0	1	1	15	39
11	data	6	1	0	1	2	15	39
12	b_A	7	2	0	1	2	15	39
13	ACK_A	7	2	0	2	2	15	39
14	b_A	8	3	0	2	2	15	39
15	BLQ	8	0	0	2	2	17	60
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
n-2	data	.	.	.	4	5	13	.
n-1	ACK_A	.	.	.	5	5	13	.
n	DLQ	.	.	.	0	0	12	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Unlike the link estimator, the routing engine adds to the routing table only the neighbouring sensor nodes that have paths to the base station. The routing engine infers this information from the received beacons before passing them to the link estimator. Suppose that sensor node A announced a path cost of 20, and its entry in the routing table of sensor node B looks as shown in Table 6.4.

Sensor node B infers from this entry that its neighbour sensor node A has a path to the base station through a sensor node, for example C. This path has a cost of 20 and

Table 6.4: Example of CTP routing table

neighbour	***parent***	***cost***	***haveHeard***	***congested***
A	C	20	1	0

it is not congested. This entry is not updated as often as the entry of sensor node A in the neighbour table. With every received beacon from sensor node A, the link estimator updates its entry in the neighbour table. On the other hand, the routing engine updates the entry of sensor node A in the routing table only if the parent is changed, the path cost changes, or the path is congested. The routing engine will update the `haveHeard` field if the route through sensor node A is lost, i.e., sensor node A does not send any more beacons.

To compute the cost of its path to the base station, sensor node B adds the path cost of sensor node A to the `etx` value of A computed by the link estimator. It does the same process for all other neighbouring sensor nodes in the routing table. Sensor node B chooses the neighbour that offers the lowest path cost to the base station. Suppose that the beacon that arrived in step 7, in the neighbour table, holds the path cost of A. This beacon forces the routing engine to create an entry for A in the routing table. Later, when the `updateRouteTask()` function is called, the routing engine will have a path to the base station through sensor node A with cost 30. If this is the lowest path cost to the base station, the routing engine will mark sensor node A as the next hop to the base station. It is worth mentioning that other neighbouring sensor nodes may offer a lower path cost to the base station in the future. However, the routing engine will not change the next hop unless it does not receive any more beacons from the current next hop. Also, it will change the next hop if another neighbouring sensor node offers a path cost that is lower than the current path cost by 15 (CTP default) or more. In its future beacons, B will broadcast a path cost of value 30 through parent sensor node A.

6.4 Vulnerabilities of CTP to Link Quality Attacks

In this section, we explain three basic vulnerabilities that a malicious node may exploit to disrupt the computations in CTP.

As explained in MintRoute, a malicious node may send fake link qualities to a neighbouring sensor node. Since the computations of link qualities in MintRoute depend on the cooperation between sensor nodes, fake link qualities may affect these computations. This type of malicious node attack is not effective in CTP because the sensor nodes do not exchange link quality information. Each sensor node in CTP computes its link qualities independent of the other sensor nodes. Thus, the malicious node has to find other methods to compromise CTP.

Figure 6.8 [135] depicts the types of link qualities that a CTP sensor node computes for its neighbouring sensor nodes. The figure shows sensor node S and its three neighbouring sensor nodes N_1, N_2, and N_3. Sensor node S has chosen N_1 as its parent. The figure shows the types of link qualities with respect to S. Sensor node S computes the inbound link qualities for all its neighbours since this quality depends on the beacons being broadcast from them. Since the computation of the outbound link quality depends on receiving ACKs from the parent, S computes the outbound link quality for sensor node N_1 only. In other words, the inbound link quality is computed for all neighbouring sensor nodes, whereas the outbound link quality is computed for the next hop sensor node only.

Knowing that each sensor node computes its link qualities independently, a malicious node cannot influence the computation of link qualities by sending false information to its neighbours. In addition, since inbound and outbound link qualities are computed differently, the malicious node can either manipulate the up-link quality or influence the computation of the down-link quality, where the up-link quality is the link from the malicious node to the base station, and the down-link is the link from the malicious node

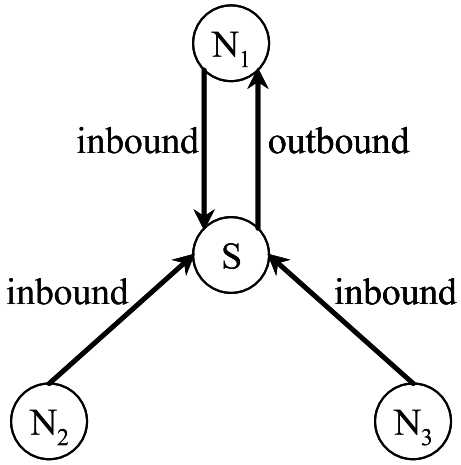


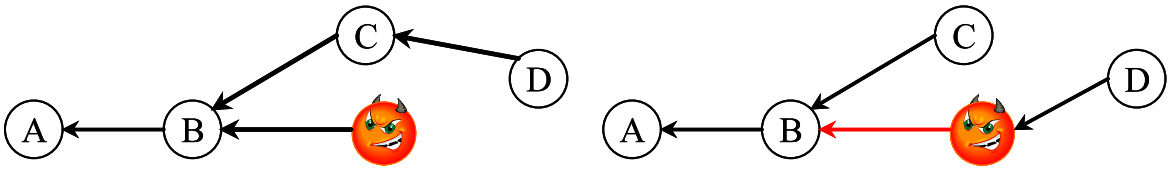
Figure 6.8: Types of link qualities in CTP

to a neighbouring sensor node [136]. Manipulation of up-link quality can be achieved by broadcasting false path cost, for example. Influencing the down-link quality can be achieved by manipulating the beacon transmission interval. Moreover, the malicious node can combine both strategies for maximum efficiency. Next, we describe three basic scenarios followed by a combined scenario in which a malicious node may compromise CTP.

6.4.1 CTP Scenario 1

In Scenario 1, the malicious node manipulates the up-link quality. In this scenario, the malicious node lies about the quality of the link to its parent. Thus, when the routing engine of the malicious node chooses a parent, it will add the lowest possible cost to the value of the path cost of the parent. The malicious node then broadcasts the computed value as its path cost. The malicious node aims to convince its neighbouring sensor nodes that it has a low cost route to the base station, as shown in Figure 6.9. The true local link quality between the malicious node and its parent, sensor node B, may not offer the lowest path cost to sensor nodes C and D, as shown in Figure 6.9(a). However, if the malicious node manipulates the `etx` value, it may offer a better route to D, as shown in

Figure 6.9(b).



(a) Before compromising local link (b) After compromising local link

Figure 6.9: Behaviour of malicious node in CTP Scenario 1

6.4.2 CTP Scenario 2

The second way to manipulate up-link quality is to lie about the value of the path cost. The malicious node adopts a more vicious behaviour by faking its path cost. The malicious node escalates its attack because scenario 1 suffers from ineffectiveness, especially when the malicious node is located far from the base station. In that case, the local link quality has a small contribution to the computation of the path cost of the malicious node. Thus, if the path cost of the parent sensor node is already high, it is doubtful that faking the local link quality will offer a low path cost for the malicious node.

When the malicious node advertises a low path cost to the base station, its path looks more attractive to the other sensor nodes, as shown in Figure 6.10. The fake path cost value offers a better route for sensor nodes C and D.

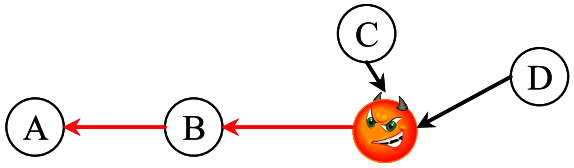


Figure 6.10: Behaviour of malicious node in CTP Scenario 2

6.4.3 CTP Scenario 3

In Scenario 3, the malicious node influences the computations of down-link qualities. Influencing the computations for the down-links means that the malicious node interferes in the computations of the outbound links or the inbound link qualities of its neighbouring sensor nodes.

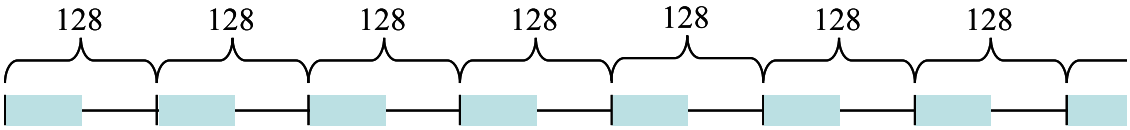
Influencing the computations of the outbound link qualities is not advantageous to the malicious node. First, it is not necessary that all the neighbouring sensor nodes have an outbound link quality to the malicious node. Only the neighbours that choose the malicious node as their next hop will have outbound link qualities to it. To manipulate the computations of the outbound link qualities of its children sensor nodes, the malicious node shall not acknowledge the reception of their data packets. However, this behaviour will lead to a negative outcome for the malicious node because its children will have low outbound link qualities to it. This means that the malicious node will not be a favoured next hop in the future.

It is more advantageous for the malicious node to influence the computations of the inbound link qualities of its neighbouring sensor nodes. Since inbound link qualities are computed for all neighbouring sensor nodes, regardless of which one is the next hop, the malicious node will have a larger impact than targeting outbound link qualities. Since the computation of inbound qualities depends on beacons, the malicious node has to manipulate its beacon broadcast to influence the computations of its neighbouring sensor nodes. A malicious node implementing Scenario 3 takes advantage of the count-based property of the computations and the adaptive beacon transmission intervals. In CTP, a sensor node computes its inbound link quality for a neighbouring sensor node for every three beacons received from this neighbour. These beacons should be received in adaptive intervals, which means that the time difference between any two beacons should increase over time. A malicious node may compromise the beacon transmission interval

by always using the minimum beacon transmission interval. Thus, the neighbouring sensor nodes will receive more frequent beacons from the malicious node than the other good sensor nodes.

For example, Figure 6.11 [135] compares the beacon transmission intervals of a good sensor node versus a malicious node. We can see that if a malicious node uses the minimum beacon transmission interval, 128 time units, it will send seven beacons for every three beacons sent by a good sensor node. This means that the inbound link qualities for the malicious node will be computed twice versus once for the other sensor nodes. This helps the malicious node to reinforce the inbound link qualities of its children sensor nodes and attract other neighbouring sensor nodes. This behaviour does not look suspicious because, in CTP, sensor nodes use the minimum beacon transmission interval if they do not have routes to the base station.

(a) Beacon transmission intervals of a good sensor node



(b) Beacon transmission intervals of a malicious sensor node

Figure 6.11: Behaviour of malicious node in CTP Scenario 3

6.4.4 CTP Combined Scenarios

A more vicious node would combine the basic scenarios to lure more good sensor nodes. It can combine Scenarios 1 and 2, Scenarios 1 and 3, or Scenarios 2 and 3. However, Scenario 2 subsumes Scenario 1, which means that exaggerating the path cost implies lying about the local link quality as well. This means that Scenario 2 is more powerful

than Scenario 1. Accordingly, we only consider the latter combination in which the malicious node combines Scenarios 2 and 3.

If a malicious node broadcasts low path cost in frequent beacons, then it will be more convincing to its neighbouring sensor nodes to route their sensed data through it. Moreover, these neighbouring sensor nodes will be offering low path costs as well and thus, more sensor nodes will join the subtree of the malicious node. For the ease of explanation, we call the combined attack, Scenario 4.

6.5 Detection Engine of CTP Vulnerabilities

Since the nature of attacks on CTP differs from one scenario to another, we have divided the detection of the attacks on CTP into three modules. These modules correspond to the detection of the three basic Scenarios 1, 2, and 3. The combined scenario is detected by any of the detection modules of the constituent scenarios.

6.5.1 Detection Module 1

Detection module 1 is responsible for detecting malicious nodes that follow Scenario 1. Although we did not implement this module, we give its details for the completion of the detection engine.

The detection of Scenario 1 malicious node is not straightforward since the computations of the local links are internal for each sensor node. Moreover, the computations are not shared with other sensor nodes. Therefore, it is hard to compute expectations for the link quality values. However, the parent sensor node can discover the deception of the malicious node by performing the sequence number gap trick as follows:

- introduce a sequence number gap in its beacons. Thus, the parent sensor node will have a maximum threshold for the value of `rcvCnt` of the malicious node and a

minimum threshold for the value `failCnt`;

- use the two values to compute an expected value for the inbound link quality computed by the malicious node; and
- add its own path cost to the expected value and compare the result with the announced value from the malicious node.

Similarly, the parent node performs the same steps for all its children sensor nodes. The drawback of this procedure is that only one sensor node can discover the malicious node, which is its parent. Moreover, if the malicious node is faking its parent, then the malicious node will go undetected.

6.5.2 Detection Module 2

Detection module 2 takes advantage of the common neighbouring sensor nodes between a parent and its child to detect malicious nodes that follow Scenario 2. To detect the malicious node, the common neighbouring sensor nodes between the malicious node and its parent sensor node work as watchdogs for the advertised path cost values. They take advantage of the following property: since a child sensor node adds the `etx` value of its parent to the value of path cost of its parent, the path cost of the child sensor node must be greater than the path cost of its parent. Accordingly, if the path cost of the malicious node is less than or equal to the path cost of its parent, then the common neighbours will flag it.

To be more vicious, the malicious node can claim a fake next hop. In such case, the malicious node escapes detection because there are no common neighbours between it and its parent.

6.5.3 Detection Module 3

Detection module 3 detects malicious nodes that implement Scenario 3 by measuring the arrival times of beacons and checking that they exhibit an adaptive behaviour. However, the detection is not straightforward because there are cases where a sensor node resets or does not double its beacon transmission interval as explained in Section 6.2.2.

Figure 6.12 [135] shows a state machine that implements detection module 3, where *cost* is the path cost of the received beacon, EB is an expected beacon that arrives in an adaptive fashion, UEB is an unexpected beacon, and *alert* is a counter of beacon violations. Each sensor node in a WSN should have a local copy of this state machine to detect malicious nodes independently. This state machine considers the following factors in CTP:

- At least three beacons are required to compute a link quality.
- Sensor nodes will use the minimum beacon transmission interval if they do not have routes to the base station.
- After establishing routes to the base station, sensor nodes should adapt their beacon transmission interval.

The state machine tests only beacons from sensor nodes that have routes to the base station, with $cost > 0$. After starting to monitor the arrival of beacons, the state machine requires three consecutive beacons from the same neighbouring sensor node to blacklist it. This threshold is important to ensure that the malicious node is not considered for parent choice after computing the link qualities. To punish malicious nodes and to prevent the isolation problem, the state machine decreases the value of the *alert* variable for every conformant beacon. On the other hand, it will increase the value if the beacon is suspicious. A blacklisted sensor node that has its *alert* value reach 0 is removed from the blacklist.

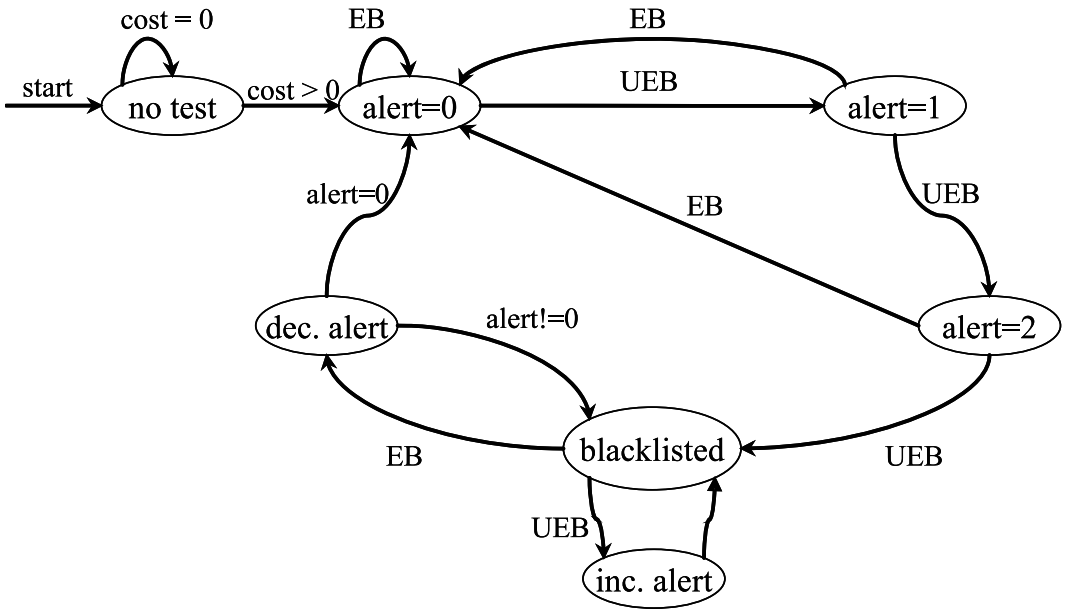


Figure 6.12: Detection Module 3 of CTP Scenario

6.6 Summary

In this Chapter, we explained the details of CTP and showed how they are integrated to build routing trees. CTP comprises three modules that cooperate to build a robust routing tree. However, the sensor nodes in a CTP WSN do not cooperate to build the routing trees as they do in MintRoute. This makes it harder for a malicious node to share fake values to influence the computations of its neighbouring sensor nodes to its favour. However, we pointed out three basic scenarios in which a malicious node can compromise the computations of the link qualities in CTP. In two of these scenarios, the malicious node manipulated its path cost and in the third, it manipulated its beacon transmission intervals. Combined scenarios where a malicious node combined more than one basic scenario were also discussed. Finally, the Chapter discussed a proposed detection mechanism that isolates the malicious node from the rest of the WSN.

The next Chapter discusses the simulation model and results of the three CTP scenarios in ns-2.

Chapter 7

CTP: SIMULATION MODEL AND RESULTS

7.1 Introduction

This Chapter discusses the simulation results of CTP in ns-2. The Chapter begins by explaining the implementation of CTP and its supporting algorithms in ns-2. Next, the Chapter describes the setup of the simulation environment and the configuration of the sensor nodes, the malicious node, and the proposed IDS. The simulation of attack scenarios on CTP comes next. In that part of the Chapter, the effect of each attack on CTP is explained followed by a discussion about the simulation results of the proposed IDS.

7.2 Extending ns-2 to Support CTP

In this section, we explain the extensions to ns-2 to support CTP and the proposed CTP IDS. Additional components that do not exist in ns-2 are also implemented. These components are: a random number generator, and CTP specific data structures.

7.2.1 Implementation of CTP

We have used the source code of CTP in TinyOS and the papers [128, 129] as references for the implementation of CTP in ns-2.

CTP implements its three modules (the link estimator, the routing engine, and the forwarding engine) separately where they interact together through defined interfaces. This implementation gives the flexibility to replace any of them without affecting the others. For example, we may use the link quality estimator WMEWMA instead of 4-bitle

without affecting either the implementations or the operations of the routing engine and the forwarding engine.

To accommodate this flexibility, we have implemented CTP in ns-2 as four separate classes. Three of these classes implement the three modules and the fourth links them together.

7.2.2 Implementation of TinyOS Random Number Generator

CTP requires a random number generator to schedule the transmission of beacons. TinyOS offers two algorithms for generating random numbers: Linear Feedback Shift Register pseudo random number generator (RandomLFSR), and Multiplicative Linear Congruential Generator (RandomMLCG). RandomLFSR is faster, but the generated numbers have less randomness [137]. CTP uses RandomMLCG seeded with the IDs of the sensor nodes. Several random number generators are implemented in ns-2 but neither RandomLFSR nor RandomMLCG is implemented. Accordingly, we have implemented RandomMLCG in ns-2 to use in CTP simulations. The algorithm of RandomMLCG is described in Section A.1 of Appendix A.

7.2.3 Implementation of CTP Data Structures

CTP uses several data structures in TinyOS that have different implementation in ns-2 or do not exist at all. These data structures are *the queue*, *the message pool*, and *the message cache*. *The queue* is a general First In First Out (FIFO) structure that has a limited size. Data packets are enqueued at the end of the queue and dequeued from the front of the queue. *The message pool* is a dynamic memory component with a limited size. The message pool adds new elements to any empty space in it. Finally, *the message cache* is a CTP specific data structure that implements the Least Recently Used (LRU) cache algorithm [138] to store the signatures of CTP data packets. We have implemented

the three data structures in ns-2. The algorithms of the three data structures are given in Sections A.2, A.3, and A.4 of Appendix A, respectively.

7.2.4 Implementation of CTP IDS

As explained in Section 6.5, CTP IDS is divided into three modules, one for each of the basic scenarios. As discussed, a malicious node that follows Scenario 2 implies following Scenario 1. In addition, Scenario 2 is more powerful than Scenario 1. Accordingly, we have neither implemented Scenario 1 nor detection module 1. Detection modules 2 and 3 are implemented in routines `Test2()` and `Test3()`, respectively.

Since both detection modules 2 and 3 test the path costs or the arrival time of beacons, it is more convenient to implement them in the routing engine. The routing engine is responsible for computing path costs and scheduling beacon transmissions. Thus, when the routing engine receives a beacon, it calls `Test2()` and `Test3()` to test any anomalous behaviour.

Storage Space Overhead

To implement the detection modules, extra information about the received beacons is required. A logical place to save this information is the routing table of the routing engine since the detection modules are implemented in the routing engine. However, the routing table holds information about the best ten routes to the base station only. The entries are inserted and evicted depending on the routes offered by the neighbouring sensor nodes. On the contrary, the detection routines depend on permanent information about the routes of the neighbouring sensor nodes. As a result, we have added a new table, `beaconInfo`, to the routing engine to hold the extra information. This table enables the detection routines to keep the required information about neighbouring sensor nodes at all times. In addition, it would make it easy to separate the detection engine into a separate entity if required. The fields of the `beaconInfo` table are shown in Table 7.1.

Table 7.1: Fields of the `beaconInfo` table

Field	***Size***	***Description***
neighbour	2 bytes	ID of neighbouring sensor node
lastSeq	1 byte	sequence number of last received beacon
cost	2 bytes	value of the path cost to the base station
prevBeaconTime	2 bytes	arrival time of previous beacon
nextMinDiff	4 bytes	min. difference between received beacon & next beacon
nextMaxDiff	4 bytes	max. difference between received beacon & next beacon
alert2	1 byte	alerts raised from detection module 2
alert3	1 byte	alerts raised from detection module 3
blackList	1 byte	flag to indicate if neighbour is blacklisted

The `beaconInfo` table consists of 9 fields that occupy 18 bytes per entry. For each neighbouring sensor node, the routing engine extracts the `neighbour`, `lastSeq`, and `cost` fields from the beacons to create an entry for the sending neighbouring sensor node. Hence, the first eligible beacon from `neighbour` is used to initialize its entry. If there is an entry for `neighbour`, the routing engine will update the latter two fields. The routing engine updates the value of `prevBeaconTime` with the arrival time of the last received beacon after the beacon has gone through all the detection routines.

Detection module 2 uses the `cost` field to compare the values announced by a child sensor node and its parent sensor node under the condition that both are neighbours of the testing sensor node. If a neighbouring sensor node is suspected, then detection module 2 will increment the value of `alert2`.

Detection module 3 uses the values of the fields `prevBeaconTime`, `nextMinDiff`, and `nextMaxDiff` to test the arrival time of the last received beacon against the arrival time of the previous beacon - for `neighbour`. Then it updates the values of `nextMinDiff` and `nextMaxDiff` according to the outcome of the test. If the current beacon fails the test, detection module 3 will increment the value of `alert3`.

A testing sensor node will blacklist a neighbouring sensor node if the number of raised alerts exceeds a certain threshold. The `blacklist` field informs the routing engine that

the corresponding neighbour is not an option for a next hop choice.

The size of the table depends on the size of the neighbourhood of each sensor node. TinyOS does not have a dynamic memory allocation mechanism. To overcome this problem, we have fixed the size of the table in our experiments depending on the size of the WSN as explained in the next section.

Code Overhead

Algorithms 7.1, 7.2, 7.3, 7.4, 7.5, and 7.6 show the pseudo-code of the algorithms added to the routing engine of CTP. Algorithm 7.1, `TestBeacon()`, describes the code to be executed by the routing engine. Algorithm 7.2, `Test2()`, implements detection module 2 of CTP IDS and Algorithm 7.3, `Test3()`, implements detection module 3. The other three algorithms are helper routines that are called by the former three algorithms.

When a sensor node receives a *beacon* from a neighbouring sensor node *nghbr*, its routing engine calls `TestBeacon`(*nghbr*, *beacon*). If *nghbr* has a path to the base station - known from the cost value in *beacon*, then `TestBeacon()` will create and initialize an entry for *nghbr* in `beaconInfo` table. Having a path means that the next beacons from *nghbr* should arrive in adaptable times. If beacons are missed, then `TestBeacon()` will adapt the intervals `nextMinDiff` and `nextMaxDiff` depending on the number of missed beacons. For example, missing one beacon adapts the intervals once, and missing two beacons adapts the intervals twice. The `cost` of *nghbr* is set to ∞ because missed beacons may have held a new path cost value. Resetting `alert2` and `alert3` is important because the IDS tests consecutive beacons. If none of the previous conditions is satisfied, then `Test2()` and `Test3()` will be called to test the path cost value and the arrival time of *beacon*, respectively. Finally, `prevBeaconTime` of *nghbr* is updated with the arrival time of *beacon*.

`Test2()` checks the value of the path cost of *nghbr*. It tests the property that the

Algorithm 7.1 TestBeacon(*nghbr*, *beacon*)

```
READ path cost of nghbr FROM beacon
if path cost > 0 then
    if first time to hear from nghbr then
        CREATE an entry in beaconInfo table for nghbr
        SET lastSeqNo of nghbr in beaconInfo TO sequence number of beacon
        SET cost of nghbr in beaconInfo TO cost value in beacon
        SET nextMinDiff of nghbr in beaconInfo TO 128
        SET nextMaxDiff of nghbr in beaconInfo TO 320
        SET alert2 of nghbr in beaconInfo TO 0
        SET alert3 of nghbr in beaconInfo TO 0
        SET blackList of nghbr in beaconInfo TO false
    else if missed beacons from nghbr then
        for the number of missed beacons do
            CALL AdaptInterval(nghbr, 0)          ▷ Adapt min and max intervals
        CALL DecAlert(nghbr, 0)                   ▷ Reset alert2 and alert3
        SET cost of nghbr in beaconInfo TO ∞
    else
        CALL Test2(nghbr, beacon)
        CALL Test3(nghbr, beacon)
    SET prevBeaconTime of nghbr in beaconInfo TO arrival time of beacon
```

path cost of a child sensor node must be greater than the path cost of its parent. To proceed with the test, `Test2()` checks that the parent of *nghbr* is a neighbour to the testing sensor node, and it has an entry in `beaconInfo` table. Afterwards, if *nghbr* does not satisfy the aforementioned property, then `Test2()` will call `IncAlert()` to increment `alert2`. Otherwise, it will call `DecAlert()` to decrement `alert2` and check the status of the blacklist.

Algorithm 7.2 Test2(*nghbr*, *beacon*)

```
Read parent of nghbr from beacon
if parent in routing table then
    READ path cost of parent FROM routing table
    READ path cost of nghbr FROM beacon
    if cost of nghbr ≤ cost of parent then
        CALL IncAlert(nghbr, 2)                   ▷ Increment alert2
    else
        CALL DecAlert(nghbr, 2)                   ▷ Decrement alert2
```

`Test3()` tests the arrival time of beacons relative to the `nextMinDiff` and `nextMaxDiff` intervals. First, it computes the time difference between the arrival times of the current beacon and the previous beacon. If this difference is within the current intervals, then `Test3()` will call `DecAlert()` to decrement any raised `alert3` and will adapt the current intervals for the arrival of the next beacon. If the difference is within half of the current intervals, then `Test3()` will increment `alert3` only. This case may occur because *nghbr* may not adapt its beacon transmissions if any of its neighbouring sensor nodes does not have a route to the base station yet and it sets the `P` bit in its beacons. Thus, *nghbr* will not adapt its beacon transmissions until this neighbour unsets the `P` bit. This may take up to three unadapted beacons from *nghbr* because afterwards this neighbour will have *nghbr*, at least, to select as a parent. Since the testing sensor node may not be able to verify this case, it increments `alert3` for *nghbr*. It does not adapt the intervals for *nghbr* because the next beacon should be adapted to satisfy the current intervals. The third case occurs when the arrival difference falls in an overlap of two sets of intervals. For example, a difference of 600 falls in the sets [256, 640] and [512, 1280]. If the beacon belongs to the second set and the IDS classifies it to the first set, then the intervals will not be adapted correctly for the arrival of the next beacon. The next beacon should be in the interval [1024, 2560] which is four times greater than the current interval determined by the IDS. In such case, `Test3()` increments `alert3` for *nghbr* and adapts the intervals twice. In the final case, *nghbr* resets its beacon transmission interval either because it lost its route to the base station or one of its neighbours has the `P` bit set in its beacons. `Test3()` resets the intervals for *nghbr* accordingly since none of the previous conditions is satisfied.

`IncAlert()` increments the value of `alert2` or `alert3` depending on the value of the *status* parameter. If the number of raised alerts is ≥ 3 then `IncAlert()` will blacklist *nghbr*.

Algorithm 7.3 Test3(*nghbr*, *beacon*)

```
arrival difference = arrival time of beacon - arrival time of previous beacon
if nextMinDiff < arrival difference < nextMaxDiff then
    CALL DecAlert(nghbr, 3)                                  ▷ Decrement alert3
    CALL AdaptInterval(nghbr, 0)      ▷ Adapt intervals according to current values
else if nextMinDiff / 2 < arrival difference < nextMaxDiff / 2 then
    CALL IncAlert(nghbr, 3)                                  ▷ Increment alert3
else if nextMinDiff * 2 < arrival difference < nextMaxDiff * 2 then
    CALL IncAlert(nghbr, 3)                                  ▷ Increment alert3
    CALL AdaptInterval(nghbr, 0)      ▷ Adapt intervals according to current values
    CALL AdaptInterval(nghbr, 0)      ▷ Adapt intervals according to current values
else
    CALL AdaptInterval(nghbr, 1)                ▷ Reset intervals for next beacon
    CALL IncAlert(nghbr, 3)
```

Algorithm 7.4 IncAlert(*nghbr*, *status*)

```
if status = 2 then
    INCREMENT alert2 of nghbr in beaconInfo
else if status = 3 then
    INCREMENT alert3 of nghbr in beaconInfo
if alert2 + alert3 = 3 then
    BLACKLIST nghbr
```

`DecAlert()` will decrement the values of `alert2` or `alert3` if *nghbr* is blacklisted. If the number of raised alerts reaches zero, then *nghbr* will be removed from the blacklist. If *nghbr* is not blacklisted, then `DecAlert()` will reset the value of `alert2` or `alert3` depending on the value of the *status* parameter.

Finally, `AdaptInterval()` is responsible for adapting the `nextMinDiff` and `nextMaxDiff` intervals to test the arrival times of beacons. CTP uses a minimum beacon transmission interval of size 128 time units and a maximum of 512,000 time units. The routing engine doubles the beacon transmission interval after sending every beacon until the maximum is reached. Then it keeps the maximum interval for future beacons. Due to the adaptive behaviour, the minimum difference between the arrival of two beacons is 128 and the maximum difference is 768,000.

Algorithm 7.5 DecAlert(*nghbr*, *status*)

```
if status = 2 then
    if nghbr is blacklisted then
        DECREMENT alert2 of nghbr in beaconInfo
        if alert2 + alert3 = 0 then
            REMOVE nghbr from blacklist
    else
        SET alert2 of nghbr in beaconInfo TO 0
else if status = 3 then
    if nghbr is blacklisted then
        DECREMENT alert3 of nghbr in beaconInfo
        if alert2 + alert3 = 0 then
            REMOVE nghbr from blacklist
    else
        SET alert3 of nghbr in beaconInfo TO 0
else if status = 0 then
    if nghbr is blacklisted then
        DECREMENT alert2 of nghbr in beaconInfo
        DECREMENT alert3 of nghbr in beaconInfo
        if alert2 + alert3 = 0 then
            Remove nghbr from blacklist
    else
        SET alert2 of nghbr in beaconInfo TO 0
        SET alert3 of nghbr in beaconInfo TO 0
```

The malicious node that will send frequent beacons modifies Algorithm 7.7 to use the minimum beacon transmission interval. Non-malicious node should double their beacon transmission intervals then choose a random time during the new interval to broadcast their beacons. The malicious node omits the first operation and chooses the random time only. Hence, the malicious node never adapts its initial beacon transmission interval, which is the minimum allowed beacon transmission interval.

7.3 Setup of CTP Simulation Environment

In this section, we explain the settings of the simulation environment used to simulate WSNs that run CTP as the routing protocol.

Algorithm 7.6 AdaptInterval(*nghbr*, *status*)

```
if status = 0 then
    if nextMinDiff = 131072 and netMaxDiff = 327680 then
        SET nextMinDiff of nghbr in beaconInfo TO 256000
        SET nextMaxDiff of nghbr in beaconInfo TO 643072
    else if nextMinDiff < 256000 and netMaxDiff < 643072 then
        SET nextMinDiff of nghbr in beaconInfo TO nextMinDiff * 2
        SET nextMaxDiff of nghbr in beaconInfo TO nextMaxDiff * 2
    else if netMaxDiff = 643072 then      ▷ Set maximum values for both differences
        SET nextMinDiff of nghbr in beaconInfo TO 256000
        SET nextMaxDiff of nghbr in beaconInfo TO 768000
else                                                    ▷ Reset both differences
    SET nextMinDiff of nghbr in beaconInfo TO 128
    SET nextMaxDiff of nghbr in beaconInfo TO 320
```

Algorithm 7.7 DecayInterval()

```
if malicious node then
    CHOOSE random time to send beacon
else                                    ▷ original code for non-malicious nodes
    DOUBLE the beacon transmission interval
    if beacon transmission interval > maximum transmission interval then
        SET beacon transmission interval TO maximum transmission interval
    CHOOSE random time to send beacon
```

CTP differs from MintRoute in the way it seeds the random number generator. CTP uses the IDs of the sensor nodes to seed the random number generator. As a result, simulating the same network topology multiple times produces the same routing tree. Seeding the random number generator with a fixed value produces the same sequence of random number values. Thus, the transmission times of the beacons and data packets will always be the same for the same topology. Accordingly, we have written simulation files to simulate 100 different WSNs.

As explained in Section 6.2.1, the `etx` in a stable WSN requires 75 beacons to reach the minimum link quality of value 10. According to the adaptive behaviour of the beacon transmission, the 75 beacons require 32,780,160 time units. Therefore, the simulation time for each experiment has been set to 40,000,000 time units. This time is sufficient to

have the 75 beacons from each sensor node reach the minimum link qualities. Moreover, the simulation time allows for the transmission of more than 75 beacons, specifically around 89 beacons per sensor node.

We have conducted simulation experiments with different WSN densities, data transmission intervals, and locations of the malicious node. Following the simulation framework of Section 3.5, the next subsections describe the parameters of the framework.

7.3.1 Setup of the WSNs

We have simulated WSNs of different sizes namely, 25, 50, and 100 sensor nodes, to evaluate the proposed IDS. The wireless range of the sensor nodes differs in each size to have at least 95% of the sensor nodes connected. Table 7.2 shows the different wireless ranges of the sensor nodes in the different sizes of the WSNs. We have simulated 100 different WSNs for each network size.

Table 7.2: Wireless ranges of sensor nodes in CTP simulations

Size of WSN	***Range of sensor nodes***
25 nodes	45 units
50 nodes	35 units
100 nodes	25 units

Since CTP is designed to work in low traffic WSNs [128], we have chosen 100,000 and 1,000,000 time units as two long data transmission intervals. The data transmission interval means that every sensor node generates one data packet at random during that interval.

7.3.2 Setup of Malicious Node

To have different neighbourhood sizes, the malicious node has been placed at different locations in the simulation area along the diagonal from the base station. The locations

of the malicious node depend on its wireless range as shown in Table 7.3. Figure 7.1 [135] shows the locations of the malicious node relative to the base station.

Table 7.3: Physical locations of the malicious node vs. its wireless range

Range	***Physical locations***
45 units	(45, 45), (55, 55), (65, 65)
35 units	(35, 35), (45, 45), (55, 55), (65, 65), (75, 75)
25 units	(25, 25), (35, 35), (45, 45), (55, 55), (65, 65), (75, 75), (85, 85)

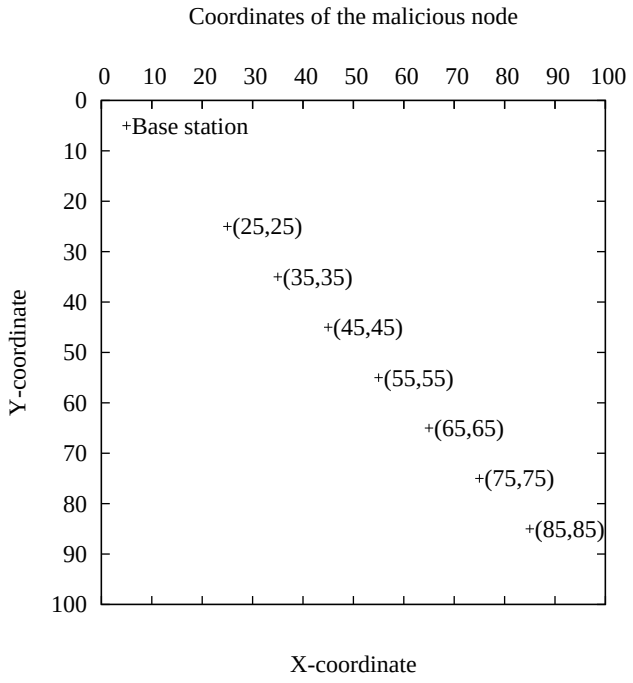


Figure 7.1: Locations of the malicious node relative to the base station

In all scenarios, the malicious node is configured to discard the data packets that it receives from its children sensor nodes. This mechanism helps in measuring data delivery at the base station before and after the attack. However, we assume that the malicious node can perform any malicious behaviour. So as not to deteriorate the link qualities of its children, the malicious node acknowledges the reception of data packets.

7.3.3 Setup of CTP IDS

To detect malicious nodes effectively, the IDS needs an adequate size of the `beaconInfo` table with respect to the neighbourhood size. A fine-tuned size allows an easy detection of malicious nodes. An underestimated size may result in failing to track the beacons of some neighbouring sensor nodes and thus, the IDS may fail to detect the malicious node. An overestimated size results in wasting the scarce storage space.

Since we assume the unit disk model for the wireless ranges and randomly uniform distributed sensor nodes, we can compute an expected value for the number of neighbouring sensor nodes as shown in Equation 7.1.

$$\begin{aligned} expected\ no.\ of neighbours &= density\ of\ sensor\ nodes \times area\ of\ wireless\ coverage \\ &= \frac{number\ of\ nodes}{simulation\ area} \times \pi r^2 \end{aligned} \tag{7.1}$$

where *number of nodes* = size of the WSN plus one for the malicious node, *simulation area* $= 100^2$, $\pi = \frac{22}{7}$, and r = wireless range of sensor nodes. The size of the `beaconInfo` table is set to the expected number of neighbours ahead of running experiments. Table 7.4 shows the expected size of the `beaconInfo` table for each network size.

Table 7.4: Size of `beaconInfo` table for the different sizes of WSNs

Number of nodes	***Wireless range***	***Size of* BeaconInfo**
26	45 units	$16.50 \approx 17$
51	35 units	$19.25 \approx 20$
101	25 units	$19.83 \approx 20$

7.3.4 Setup of Performance Metrics

The performance of the malicious node is measured as the percentage of data delivered to the base station as described in Section 3.5.4.

The success of the IDS is measured as the percentage of TP and the FP is computed as follows. The simulator is instrumented to record the number of neighbours of each

sensor node in a separate file. In addition, each sensor node records the ID of the neighbours that it suspects as malicious. For each WSN, TP is computed as the number of neighbours that suspect the malicious node divided by the total number of neighbours of the malicious node, as shown in Equation 7.2. Then a global average TP is computed as the average of all the tp_is of the 100 WSNs, as shown in Equation 7.3.

$$tp_i = \frac{number\ of\ neighbours\ that\ suspect\ m}{number\ of\ neighbours\ of\ m} \tag{7.2}$$

where tp_i is the TP for WSN i, and m is the malicious node.

$$TP = \frac{\sum_{i=1}^{NTs} tp_i}{NTs} \tag{7.3}$$

where NTs = number of WSNs, 100.

We compute FP from the perspective of good and suspected links. A link between any two sensor nodes, A and B, is asymmetric. This means that the link quality from A to B is different from the link quality from B to A. The link quality value depends on the number of packets a sensor node receives from its neighbouring sensor node. Thus, A may suspect B as malicious but B may not suspect A as malicious. For a sensor node, the number of neighbours represents the number of outbound links from this sensor node. Thus, to compute the number of links in a WSN, we count the number of neighbours of each good sensor node in the WSN. We subtract the number of neighbours of the malicious node from the number of links. The number of neighbours of the malicious node constitutes the number of links from the neighbours to the malicious node, which is computed in TPs. We compute a FP for each simulated WSN and a global average FP for all simulated WSN as shown in Equations 7.4 and 7.5, respectively.

$$fp_i = \frac{number\ of\ suspected\ links}{number\ of\ links} \tag{7.4}$$

$$FP = \frac{\sum_{i=1}^{NTs} fp_i}{NTs} \tag{7.5}$$

7.4 Results of CTP Simulation

This Section presents the simulation of the different scenarios that a malicious node can follow to exploit CTP. In each scenario, we discuss the effectiveness of the malicious node without running the IDS. We follow this discussion by presenting the simulation results of running the IDS accompanied by a discussion on the effectiveness of the malicious node and the success of the IDS in detecting it.

We have adjusted the simulator to use the detection routines, `Test2()` and `Test3()`, separately. So when the malicious node is configured to follow Scenario 2, `Test2()` is turned on. When it is following Scenario 3, `Test3()` is turned on. The combined scenario, Scenario 4, can be tested either with `Test2()` or `Test3()`. Separation of the detection routines facilitates the measurement of their effectiveness.

7.4.1 Simulation of CTP Scenario 0

Scenario 0 is used as the baseline for the comparisons of the effectiveness of the malicious node in the other scenarios. The malicious node in Scenario 0 does not follow any of the attack scenarios on CTP; however, it still drops data packets that it receives from its children sensor nodes. The decrease in data delivery between Scenario 0 and any of the other scenarios measures the success of the malicious node in diverting data from the base station.

The difference in the percentage of data delivery in Scenario 0 between the two data transmission intervals is negligible, less then 0.75% in all cases. Thus, the graphs in the following depict the data transmission interval 100,000 time units of Scenario 0 for the

sake of readability.

7.4.2 Simulation of CTP Scenario 1

In Scenario 1, the malicious node lies about the `etx` value to its parent. In other words, the path cost of the malicious node is always its parent's path cost plus 10, the minimum value for `etx`.

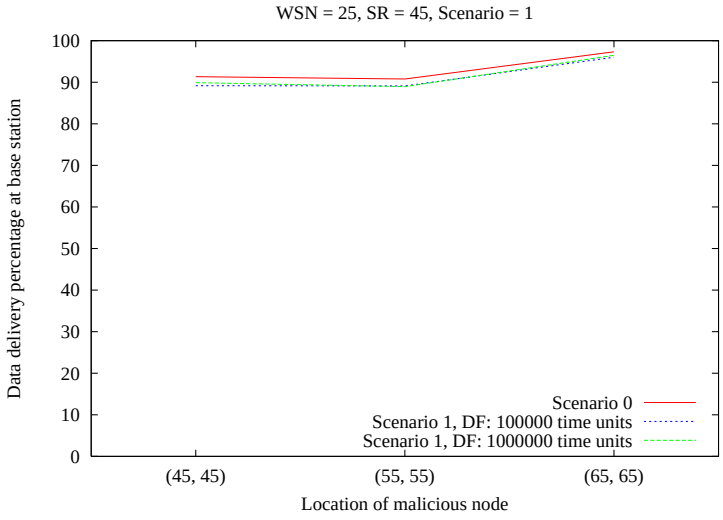
Effectiveness of Malicious Node without Running the IDS

The simulation results of Scenario 1 show that the effectiveness of the malicious node is affected by its location in the WSN. A malicious node closer to the base station means that it is included in many routes. Accordingly, the malicious node discards more traffic in close locations than in far locations. We can see in Figure 7.2 that the percentage of data delivery increases in locations far from the base station in the three sizes of the WSNs, where WSN = size of WSN, SR = wireless range of the sensor nodes, and DF = data transmission interval.

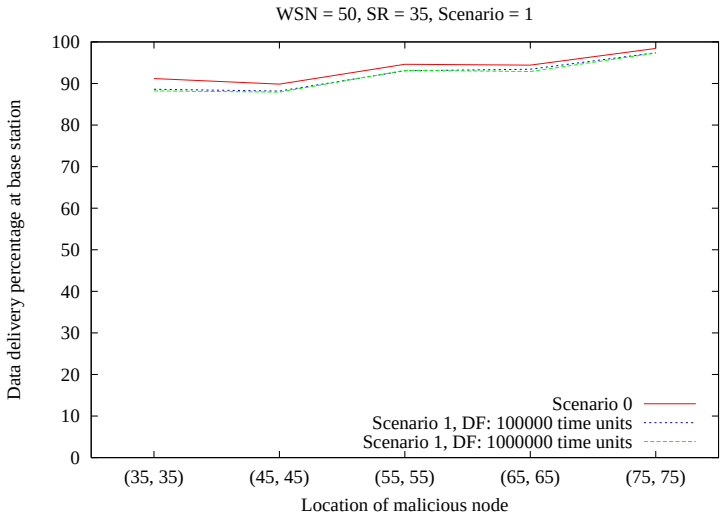
The simulation results also show that the gain of the malicious node in Scenario 1 is small. Figure 7.2 shows that the percentage of data delivery in Scenario 1 tends to get closer to the percentage of data delivery in Scenario 0. This means that the malicious node fails to attract more sensor nodes to its subtree by faking its `etx` values. The gain of the malicious node is measured as the drop in the percentage of data delivery. This is measured by subtracting the percentage of the data delivery value in Scenario 1 from the value in Scenario 0. Table 7.5 shows the maximum drops in the percentage of data delivery for each WSN configuration.

7.4.3 Simulation of CTP Scenario 2

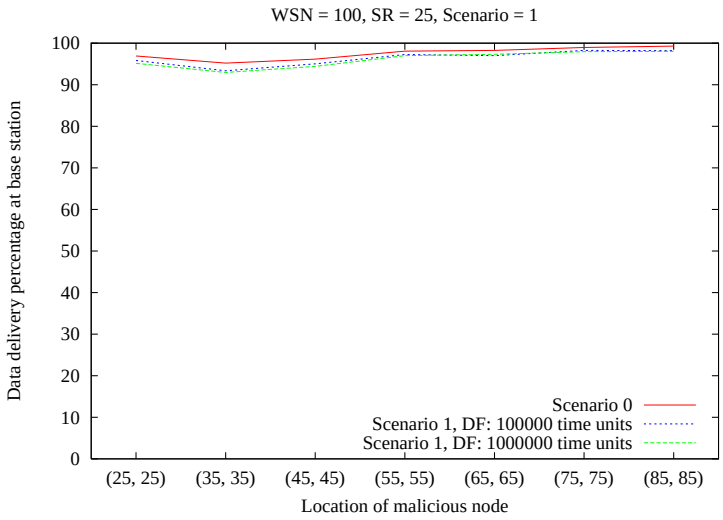
The malicious node in Scenario 2 lies about its path cost. In the experiments of Scenario 2, we have configured the malicious node to announce a path cost value of 20. The



(a) 25-node WSN



(b) 50-node WSN



(c) 100-node WSN

Figure 7.2: Effectiveness of malicious node in CTP Scenario 1 without IDS running

Table 7.5: Percentage of drop in data delivery for CTP Scenario 1

Size of WSN	***Drop***	***Data frequency***	***Location of malicious node***
25-node	2.16%	100,000	(45, 45)
25-node	1.92%	1,000,000	(55, 55)
50-node	2.57%	100,000	(35, 35)
50-node	2.94%	1,000,000	(35, 35)
100-node	1.84%	100,000	(35, 35)
100-node	1.88%	1,000,000	(45, 45)

malicious node does not choose the lowest path cost value, 10, because this means that the malicious node is a child of the base station. Thus, it can be easily detected if the sensor nodes know the ID of the base station.

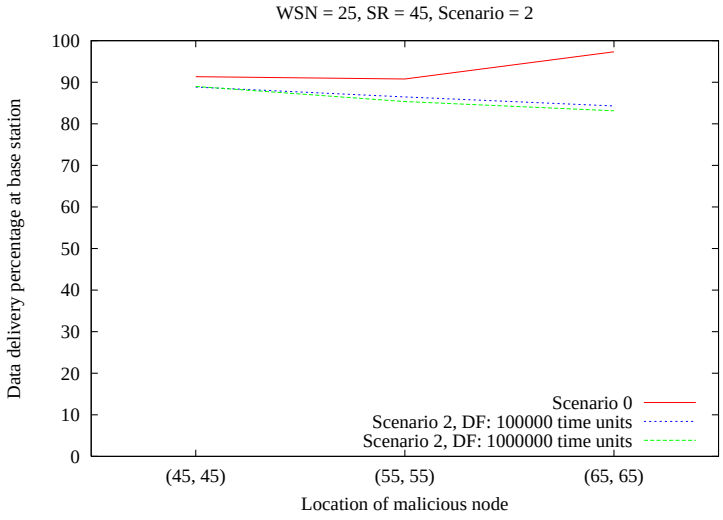
Effectiveness of Malicious Node without Running the IDS

Figure 7.3 shows the simulation results of Scenario 2. As seen in the figure, the percentage of data delivered to the base station is proportional to the distance of the malicious node from the base station. This means that the malicious node is more effective in locations far from the base station. It succeeds to enlarge its subtree when it is far from the base station because the path cost it offers is lower than the other path costs in the same area. Table 7.6 shows that the maximum drops in the percentage of data delivery occur between the centre of the WSN and the far edge from the base station.

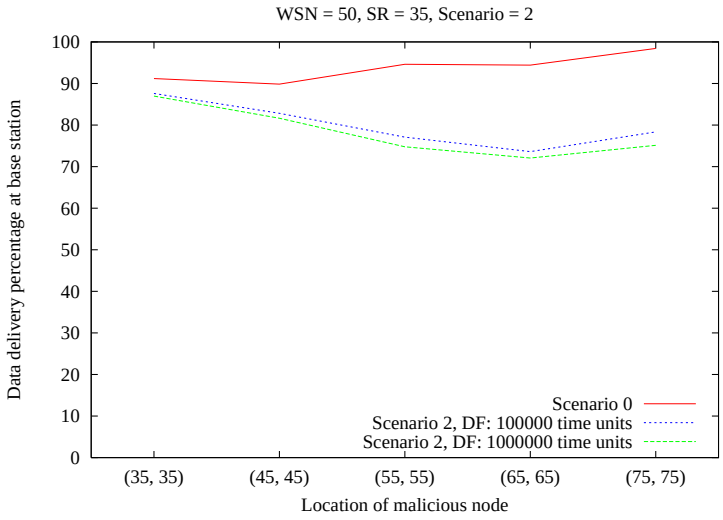
Table 7.6: Percentage of drop in data delivery for CTP Scenario 2

Size of WSN	***Drop***	***Data frequency***	***Location of malicious node***
25-node	13.04%	100,000	(65, 65)
25-node	14.53%	1,000,000	(65, 65)
50-node	20.81%	100,000	(65, 65)
50-node	23.35%	1,000,000	(75, 75)
100-node	31.00%	100,000	(65, 65)
100-node	34.60%	1,000,000	(65, 65)

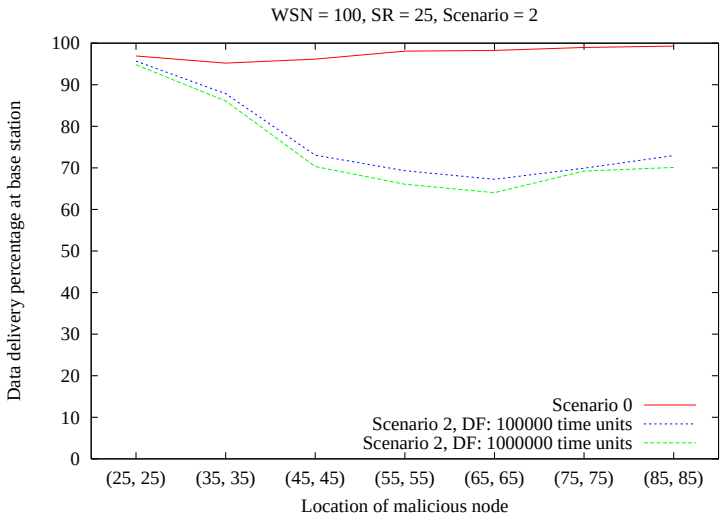
In locations close to the base station, the sensor nodes find comparable path costs to the base station. Thus, if a sensor node chooses a good neighbouring sensor node as its next hop, then the path cost of the malicious node is not competitive enough to trigger a



(a) 25-node WSN



(b) 50-node WSN



(c) 100-node WSN

Figure 7.3: Effectiveness of malicious node in CTP Scenario 2 without IDS running

parent switch. The efficiency of the malicious node increases away from the base station, where it reaches the maximum beyond the centre of the simulation environment. Around the center of the WSN, the path costs are higher so the low path cost of the malicious node is an incentive for the neighbouring sensor nodes to join its subtree. Moreover, these neighbouring sensor nodes will offer low path costs, which will encourage more sensor nodes to join the subtree of the malicious node. This phenomenon is reversed near the edge of the WSN where fewer sensor nodes can join the subtree of the malicious node. Mainly, these sensor nodes are the neighbouring sensor nodes only. These neighbouring sensor nodes are the ones near the edges of the WSN. Consequently, they cannot expand the subtree of the malicious node as they do near the center of the WSN.

Also, the figure shows that the malicious node is more successful in low-traffic WSNs. In a low-traffic WSN, 1,000,000 time units, a small number of lost packets will have an impact on the percentage of data delivered to the base station; however, the impact is smaller on high-traffic WSNs, 100,000 time units. In general, the drop in the percentage of data delivery increases with the size of the WSN because more traffic is generated due to the increased number of sensor nodes. See Table 7.6.

Effectiveness of Malicious Node with Running the IDS

Next, we discuss the effectiveness of the malicious node when `Test2()` is running. Since `Test2()` uses the watchdog concept, the detection of the malicious node depends on the number of common neighbouring sensor nodes with its parent. One can expect that the larger the number of common neighbours is, the more successful `Test2()` will be. However, some of the common neighbours are not children of the malicious node. As a result, they cannot detect the malicious node and thus, they do not change their parents to good ones.

Figure 7.4 [135] shows the effectiveness of the malicious node when `Test2()` is running.

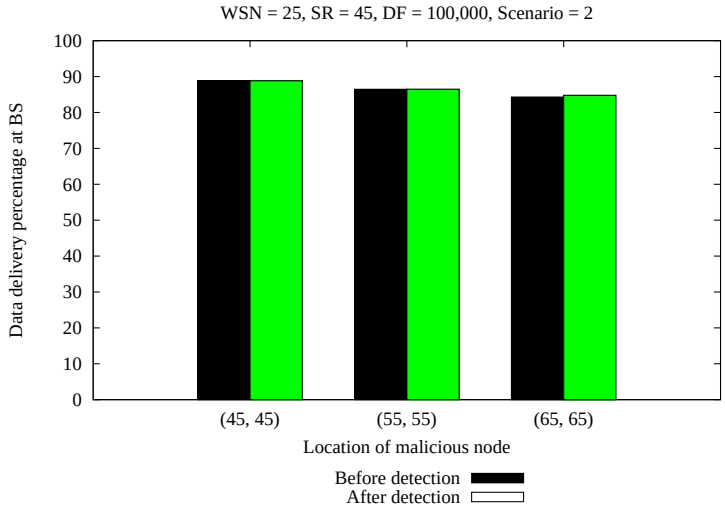
We can see that in the sparse WSN of 25 sensor nodes the effectiveness of the malicious node is not affected. First, the number of common neighbours between the malicious node and its parent is small so a small number of sensor nodes can detect the malicious node. Out of the small number of neighbours that can detect the malicious node, an even smaller number may be its children. Accordingly, most of the children of the malicious node cannot detect it and they do not change their malicious parent to a good one.

As the WSNs get denser, the number of common neighbouring sensor nodes between the malicious node and its parent increases. In addition, the number of children of the malicious node increases so a larger number of the common neighbours are children of the malicious node. These children that detect the malicious node switch their parents to good ones so the percentage of data delivery at the base station increases. Figure 7.4(b) shows that the percentage of data delivery will rise from 78.36% to 81.65% if the malicious node is placed at (75, 75). Whereas, it rises from 72.95% to 86.51 % at location (85, 85), see Figure 7.4(c).

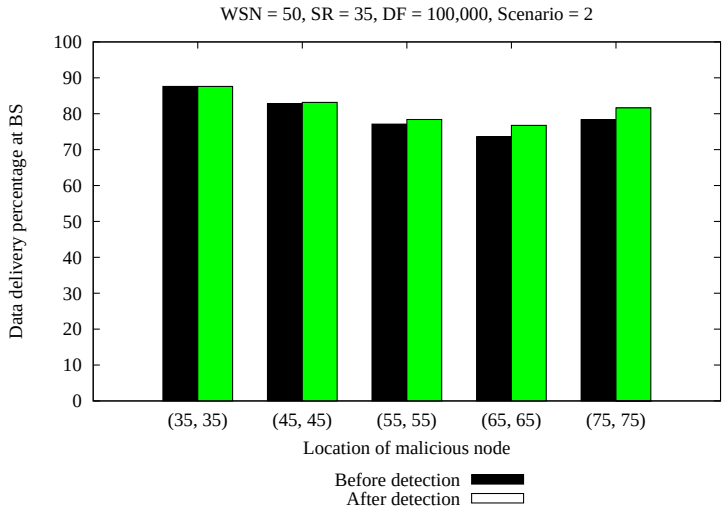
Success of Detecting Malicious Node

As discussed previously, the larger number of the common neighbouring sensor nodes helps to detect the malicious node. However, the path cost offered by the malicious node and the location of the malicious node in the WSN affect detection as well. Figure 7.5 shows that the detection of the malicious node increases as the distance between the malicious node and the base station increases.

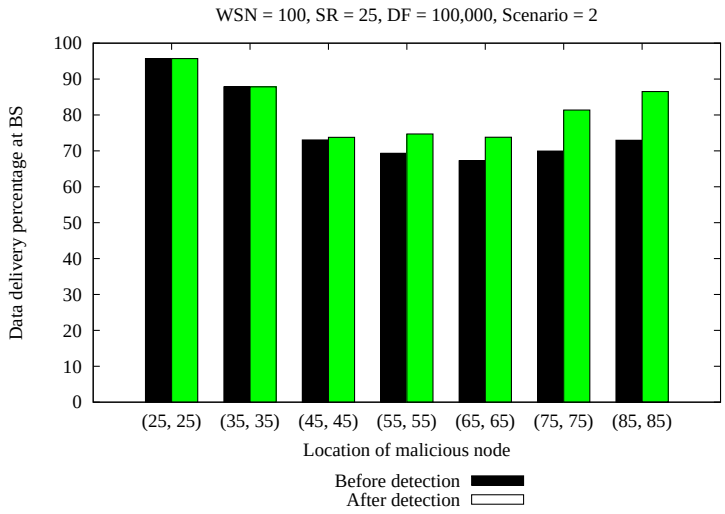
The malicious node is configured to announce a path cost value of 20. So, if the malicious node is in a location where this value is not suspicious, then no neighbouring sensor node can detect it. The locations (45, 45) in Figure 7.4(a), (35, 35) in Figure 7.4(b), and (25, 25) in Figure 7.4(c) satisfy this condition. In these locations, the malicious node is close to the base station and its parent is 1-hop away from the base station. Hence,



(a) Data delivery in 25-node WSN

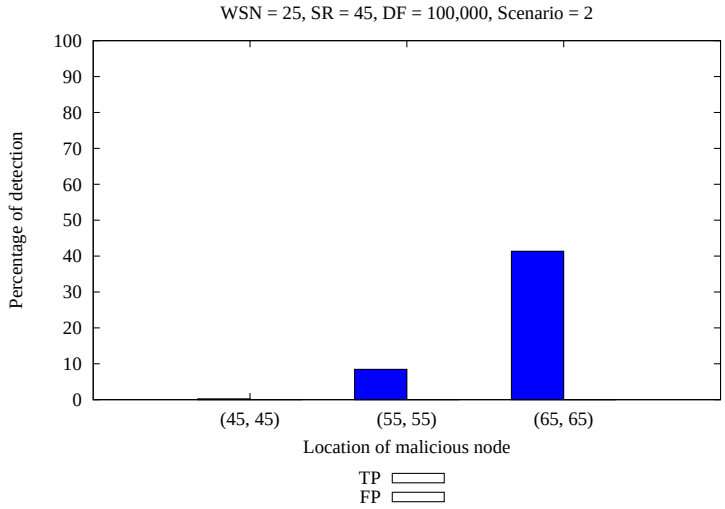


(b) Data delivery in 50-node WSN

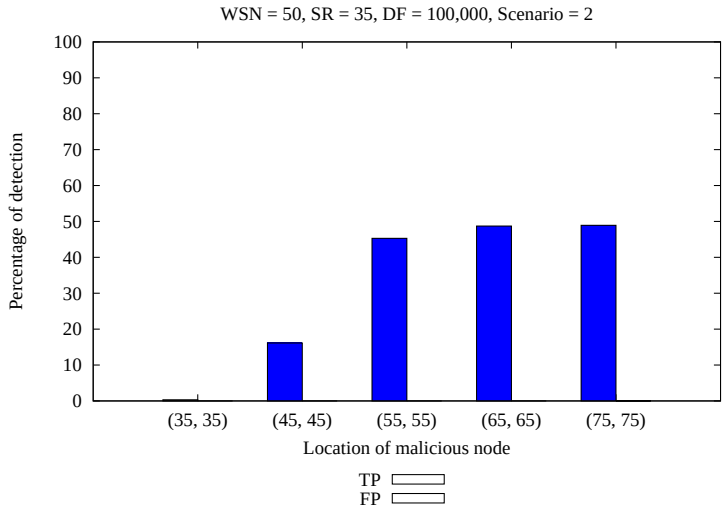


(c) Data delivery in 100-node WSN

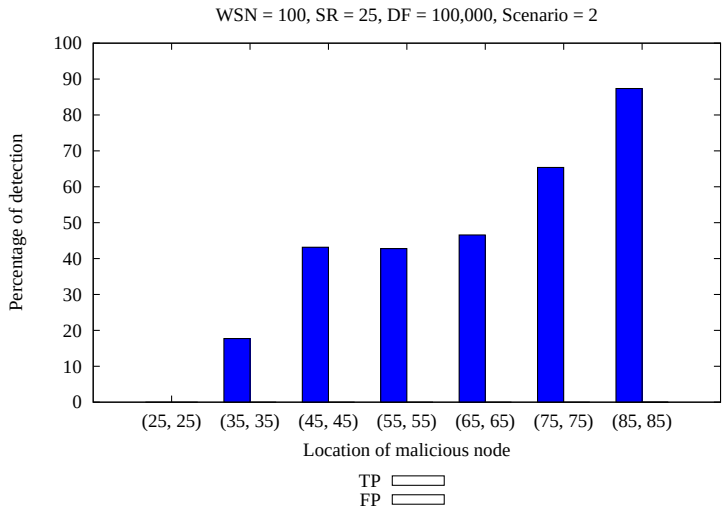
Figure 7.4: Effectiveness of malicious node in CTP Scenario 2 with IDS running



(a) True & false detections in 25-node WSN



(b) True & false detections in 50-node WSN



(c) True & false detections in 100-node WSN

Figure 7.5: Success of detecting malicious node in CTP Scenario 2

the parent of the malicious node has lower path cost due to its proximity to the base station. Accordingly, the common neighbouring sensor nodes cannot find any violations of the path cost for the malicious node. If the malicious node is located far from the base station, the path cost of its parent will get higher and the common neighbours can now detect it. At the farthest location from the base station (85, 85) in Figure 7.5(c), the malicious node is detected 87.37% of the time.

Although Figure 7.5 shows that at least 40% of the neighbours of the malicious node are able to detect it, the simulation results of data delivery confirm that most of these neighbours are not children of the malicious node. To rectify the poor effectiveness of `Test2()`, the sensor nodes must exchange their lists of suspected neighbours. To do this exchange in a secure fashion, the sensor nodes must:

- encrypt the exchanged list so that the malicious node does not discover that it is detected;
- have a trust mechanism to ensure that sensor nodes do not slander good sensor nodes; and
- minimize the size of the list to decrease the volume of additional traffic and to prevent quick battery depletion.

Clearly, the three items do not meet the stated constraints of this dissertation of not using encryption, performing in-node processing, and eliminating any additional traffic in the WSN.

7.4.4 Simulation of CTP Scenario 3

In Scenario 3, the malicious node announces its real path cost but it does not adapt its beacon transmission interval after sending each beacon. Thus, it tries to influence the computation of the `etx` values of its neighbouring sensor nodes by sending frequent

beacons. The malicious node is configured to use the minimum beacon transmission interval, 128 time units, for each beacon transmission.

Effectiveness of Malicious Node without Running the IDS

By sending frequent beacons, the malicious node influences the computations of the `etx` values of its neighbouring sensor nodes. The frequent beacons help them to reach the minimum `etx` value to the malicious node quickly. However, their choice of the malicious node as the next hop depends on the value of its path cost, which depends on how far the malicious node is from the base station. Figure 7.6 shows that the malicious node is more successful in locations close to the base station.

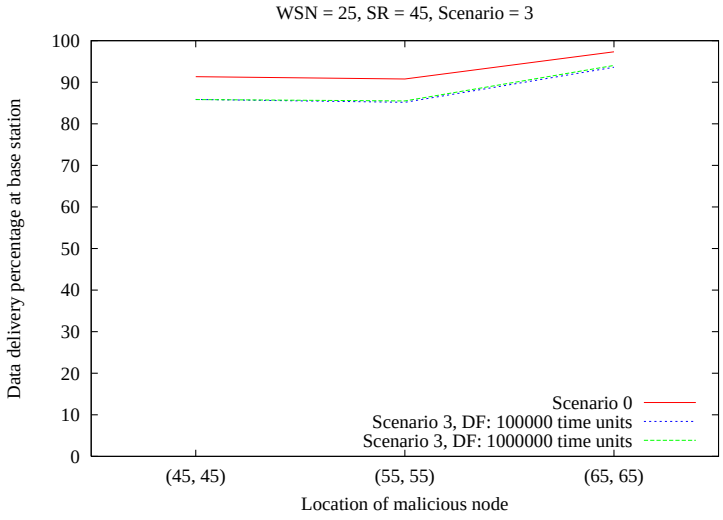
Near the base station, the malicious node has a low path cost value. Consequently, the routes through the malicious node encourage the neighbouring sensor nodes to join the subtree of the malicious node. Far from the base station, the routing cost of the malicious node gets higher. As a result, the low `etx` values of the neighbouring sensor nodes are not incentive enough to lure them to join the subtree of the malicious node. Table 7.7 shows that the maximum drops in the percentage of data delivery occur in locations close to the base station.

Table 7.7: Percentage of drop in data delivery for CTP Scenario 3

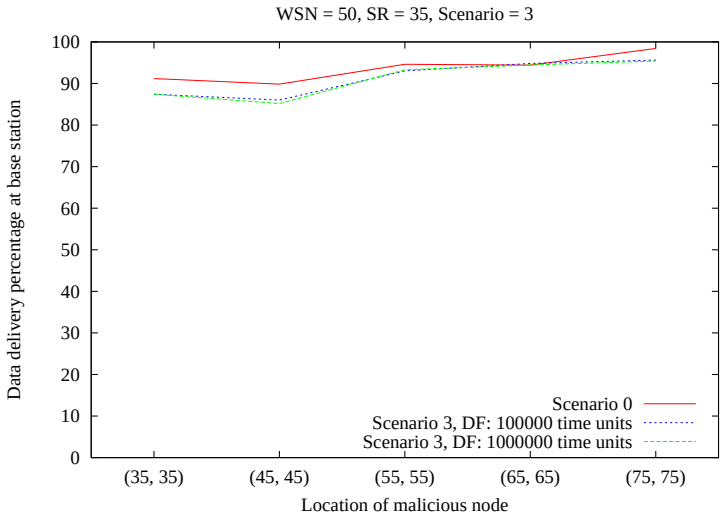
Size of WSN	***Drop***	***Data frequency***	***Location of malicious node***
25-node	5.61%	100,000	(55, 55)
25-node	5.83%	1,000,000	(45, 45)
50-node	3.84%	100,000	(45, 45)
50-node	4.00%	1,000,000	(45, 45)
100-node	11.00%	100,000	(35, 35)
100-node	10.55%	1,000,000	(35, 35)

Effectiveness of Malicious Node with Running the IDS

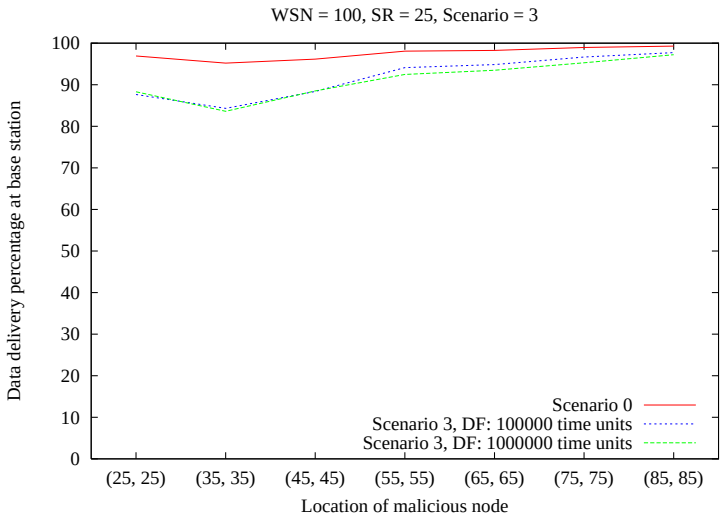
`Test3()` implements a state machine to test the arrival of beacons from neighbouring sensor nodes. A sensor node that does not adapt its beacon transmission interval for



(a) 25-node WSN



(b) 50-node WSN



(c) 100-node WSN

Figure 7.6: Effectiveness of malicious node in CTP Scenario 3 without IDS running

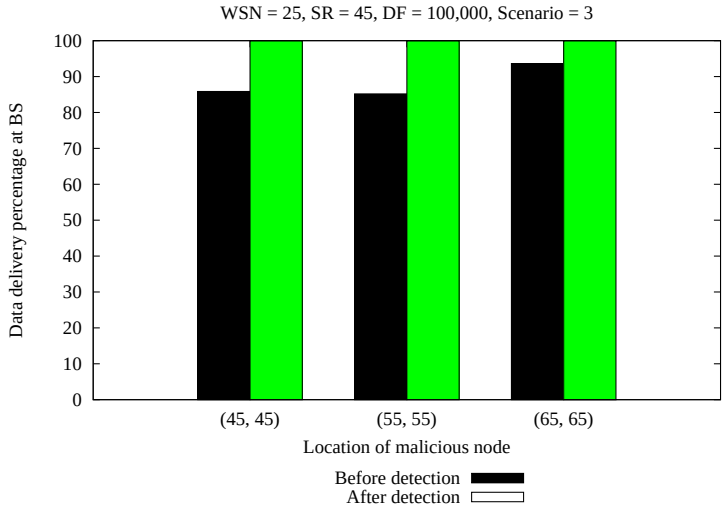
three consecutive transmissions will be blacklisted by its neighbouring sensor nodes.

Figure 7.7 shows that the percentage of data delivery is boosted to almost 100% in the three sizes of the WSNs. Since each sensor node is running a local copy of `Test3()`, all the neighbouring sensor nodes detect the malicious node. The high percentage of data delivery means that most of the neighbouring sensor nodes are able to detect the malicious node. It also means that all of them have blacklisted the malicious node and the children sensor nodes have switched their malicious parent to good ones.

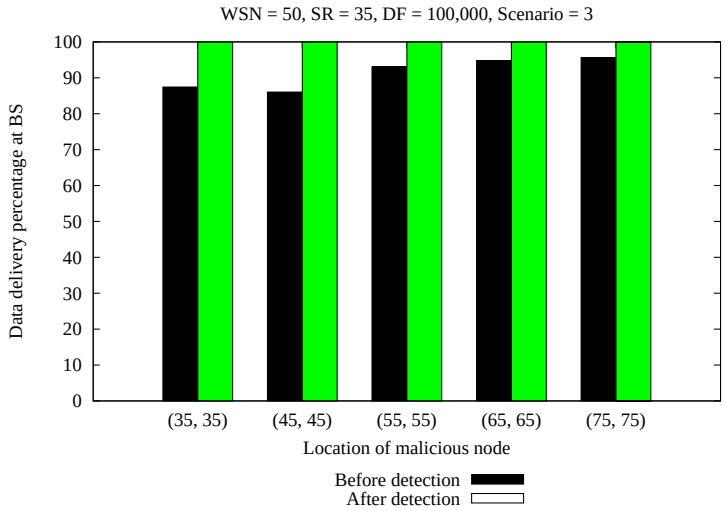
As explained earlier and as shown by the black bars in Figure 7.7, the malicious node is more effective near the base station in Scenario 3, which means that it has more children and a larger subtree. Accordingly, when the children sensor nodes detect their parent sensor node as malicious, they change their parent to a good one and succeed in avoiding the malicious node. At far locations from the base station, the malicious node is not as effective, and the black bars are higher. This means that the subtree of the malicious node is smaller, but when the small number of children sensor nodes change their malicious parent the percentage of data delivery rises to 100%.

Success of Detecting Malicious Node

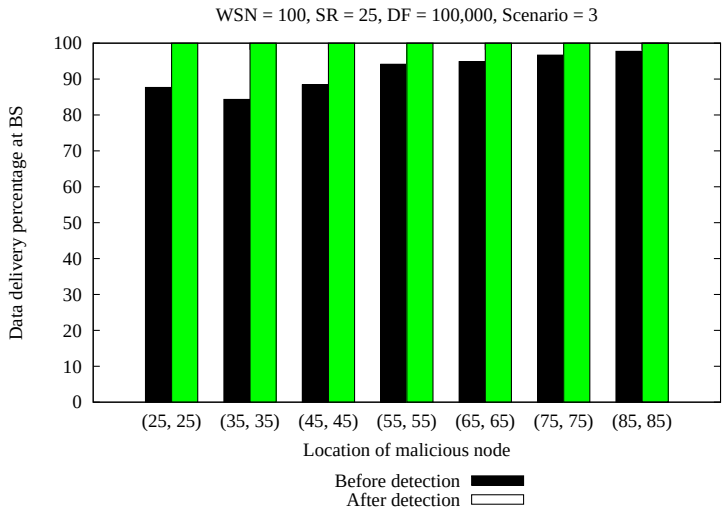
Figure 7.8 shows that `Test3()` detects the malicious node with a success percentage of at least 95%. The lowest percentage occurs in dense networks, 100-node, where packet collisions are high. The state machine of `Test3()` requires three consecutive violations for the beacon transmission intervals to blacklist a sensor node. Thus, with high packet collisions, especially when the malicious node broadcasts more beacons than usual, missing a single packet reinitiates the detection process. Although the low success does not support the high percentage of data delivery, the neighbouring sensor nodes that fail to detect the malicious node are not necessarily the children of the malicious node. Thus, their failure to detect the malicious node does not affect the data delivered to the base



(a) Data delivery in 25-node WSN



(b) Data delivery in 50-node WSN



(c) Data delivery in 100-node WSN

Figure 7.7: Effectiveness of malicious node in CTP Scenario 3 with IDS running

station.

7.4.5 Simulation of CTP Scenario 4

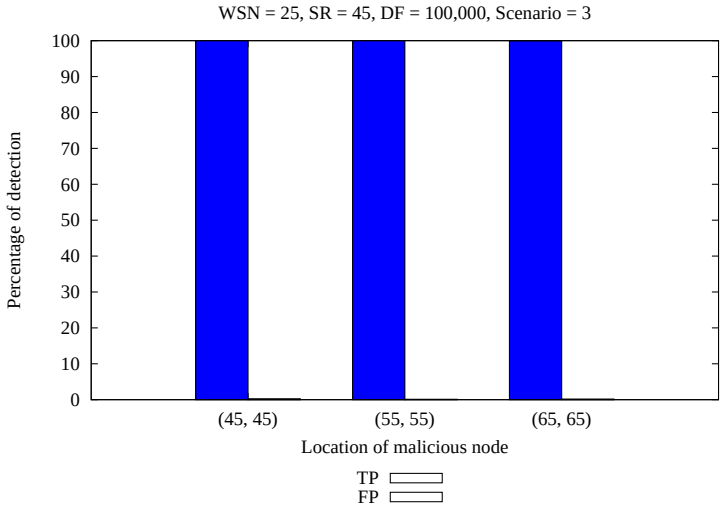
Scenario 4 is the combination of Scenarios 2 and 3. The malicious node in Scenario 4 announces a low path cost value with frequent beacons. The malicious node is configured to announce 20 for its path cost using the minimum beacon transmission interval of 128 time units.

Effectiveness of Malicious Node without Running the IDS

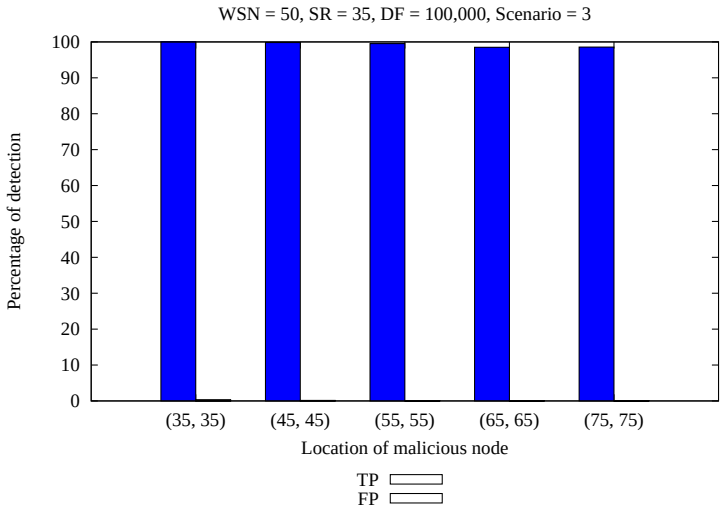
One can expect that the simulation results of Scenario 4 will be similar to the simulation results of Scenarios 2 and 3 combined. However, Figures 7.9 and 7.10 show that the drop in the percentage of data delivery exceeds the combined drop from Scenarios 2 and 3. Combining Scenarios 2 and 3 amplifies the drop to more than double the summation of their drops in some cases. Figures 7.9 and 7.10 show the percentage of data delivery in Scenario 0 as a reference. The effectiveness of Scenario 4 is compared to the summation and double the summation of the drop in the percentage of data delivery in Scenarios 2 and 3. Table 7.8 shows the maximum drops in the percentage of data delivered to the base station.

Table 7.8: Percentage of drop in data delivery for CTP Scenario 4

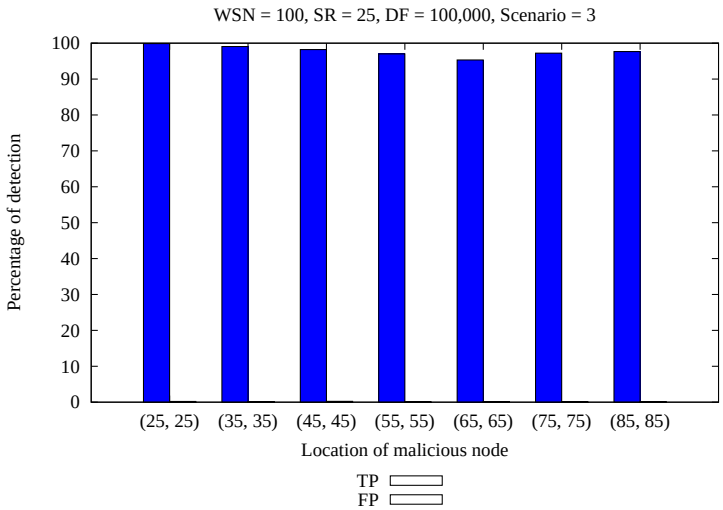
Size of WSN	***Drop***	***Data frequency***	***Location of malicious node***
25-node	37.05%	100,000	(65, 65)
25-node	40.08%	1,000,000	(65, 65)
50-node	40.63%	100,000	(65, 65)
50-node	52.04%	1,000,000	(55, 55)
100-node	54.54%	100,000	(55, 55)
100-node	67.14%	1,000,000	(45, 45)



(a) True & false detections in 25-node WSN

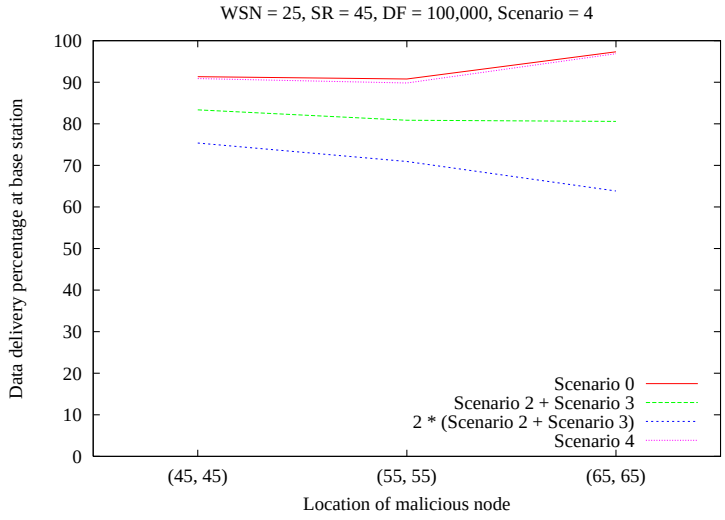


(b) True & false detections in 50-node WSN

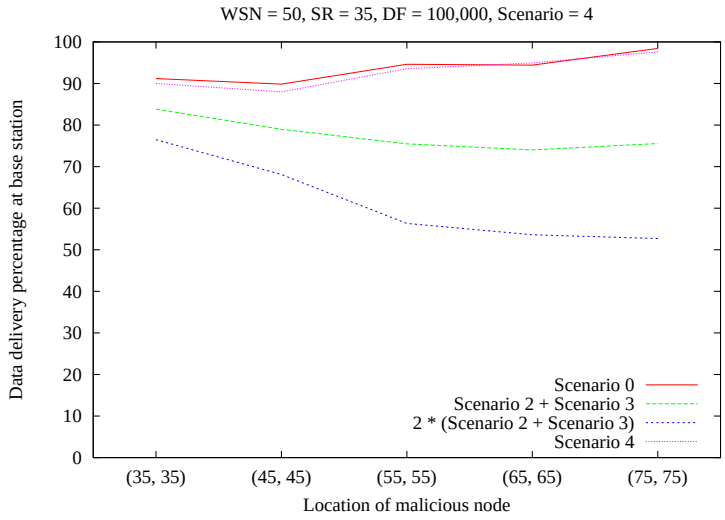


(c) True & false detections in 100-node WSN

Figure 7.8: Success of detecting malicious node in CTP Scenario 3



(a) 25-node WSN, 100,000 time units



(b) 50-node WSN, 100,000 time units

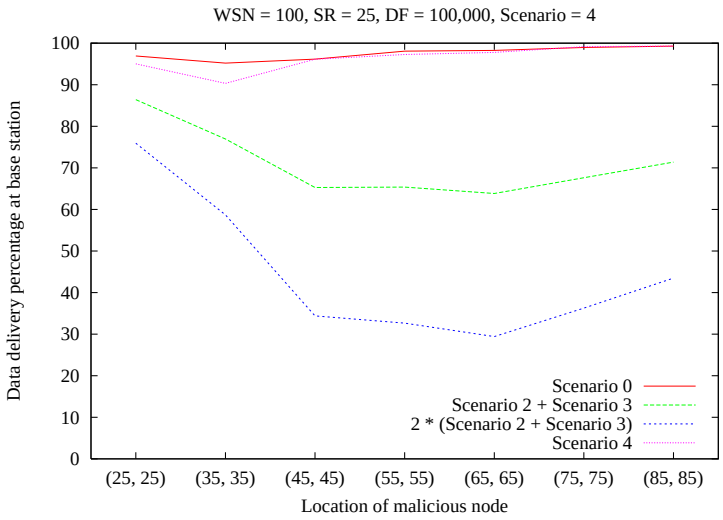
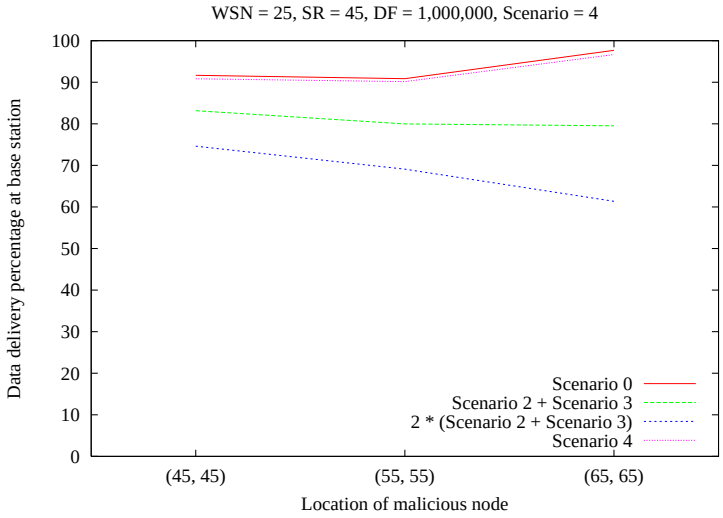
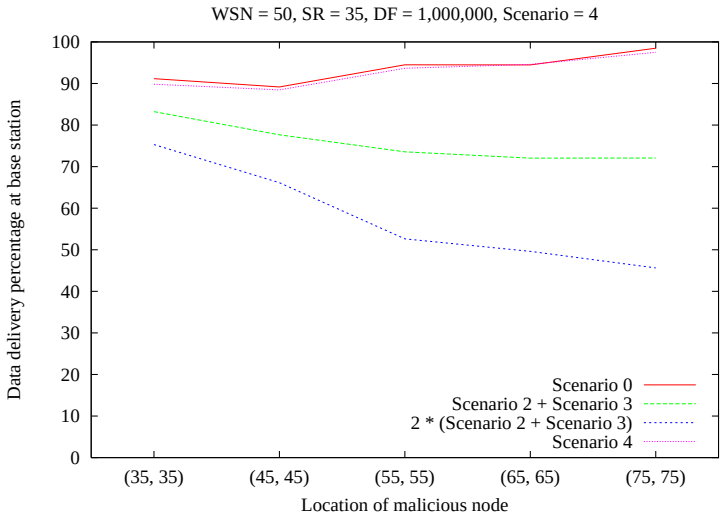


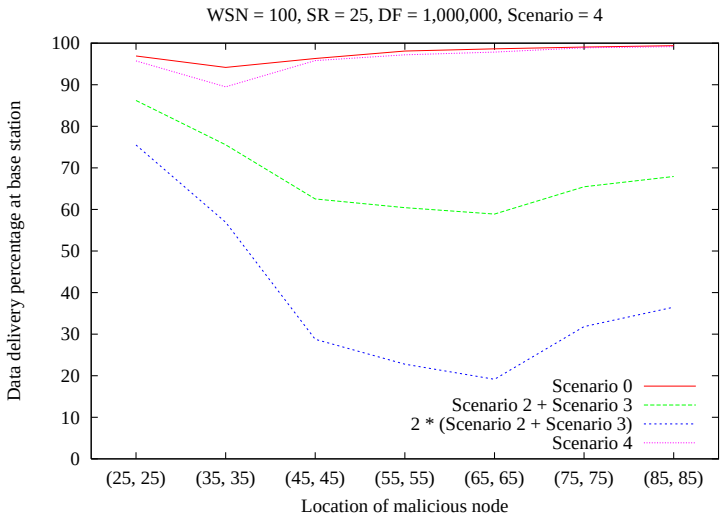
Figure 7.9: Effectiveness of malicious node in CTP Scenario 4 in high-traffic WSNs



(a) 25-node WSN, 1,000,000 time units



(b) 50-node WSN, 1,000,000 time units



(c) 100-node WSN, 1,000,000 time units

Figure 7.10: Effectiveness of malicious node in CTP Scenario 4 in low-traffic WSNs

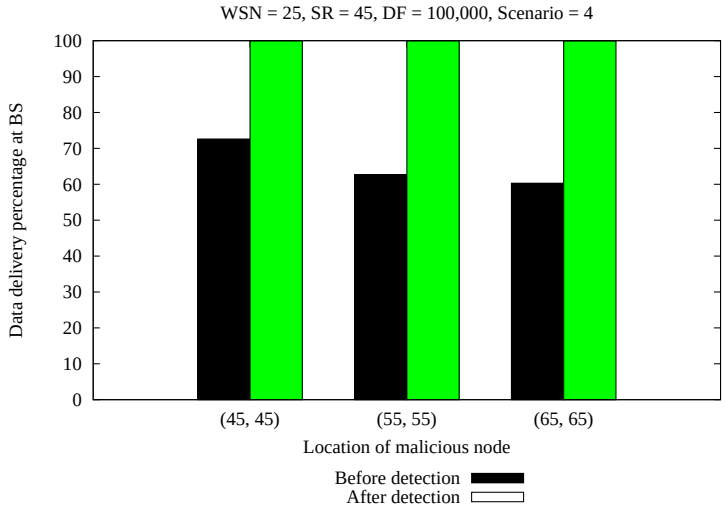
Effectiveness of Malicious Node with Running the IDS

Since Scenario 4 combines Scenarios 2 and 3, it can be detected with `Test2()` or `Test3()`. However, `Test3()` is more successful in detecting the malicious node. Therefore, we discuss the simulation results of detecting the malicious node under `Test3()`.

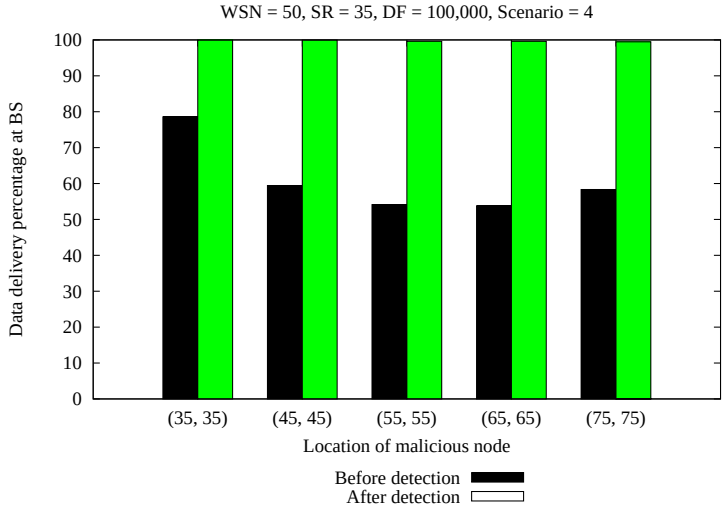
The effectiveness of the malicious node when `Test3()` is running depends on its location and the size of the WSN. In the 25-node and 50-node WSNs, the percent of data delivery is almost 100%, see Figures 7.11(a) and 7.11(b). However, in the 100-node WSN, the percentage of data delivery can be as low as 95% especially in locations far from the base station, see location (65, 65) in Figure 7.11(c).

By broadcasting low path cost value in frequent beacons, the malicious node attracts more sensor nodes to its subtree as explained previously and as shown by the black bars in Figure 7.11 [135]. As a consequence of its success, more data packets are routed to the malicious node. This leads to higher packet collisions in the vicinity of the malicious node, which affects the computations of link qualities and path costs of its neighbours.

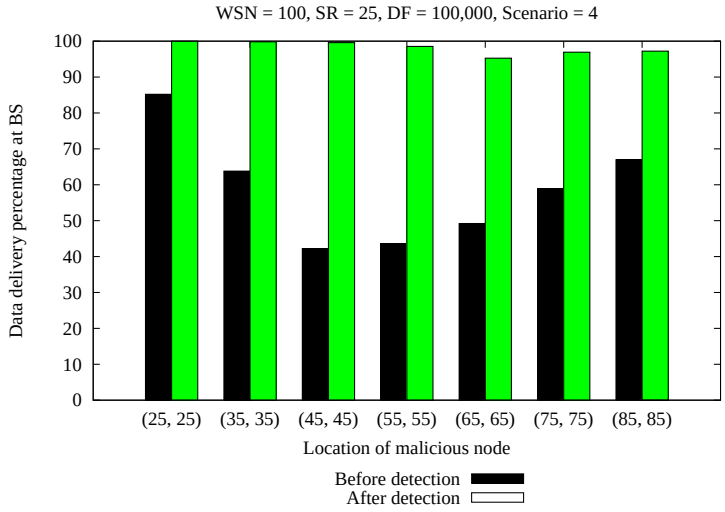
The high packet collisions lead to a phenomenon that occurred in dense WSNs, 100-node. Between the centre of the WSN and its far edge from the base station, the parent of the malicious node forms a loop by joining the subtree of the malicious node. This happens because the parent sensor node suffers from high path costs due to packet collisions and it finds the best path through one of the children of the malicious node. This loop cannot be broken because the malicious node is configured to drop data packets passing through it. Therefore, the parent sensor node will never find its data packet returning back to it. We can see that the percentage of data delivery at the base station is as low as 95.27% when the malicious node is at location (65, 65), Figure 7.11(c).



(a) Data delivery in 25-node WSN



(b) Data delivery in 50-node WSN



(c) Data delivery in 100-node WSN

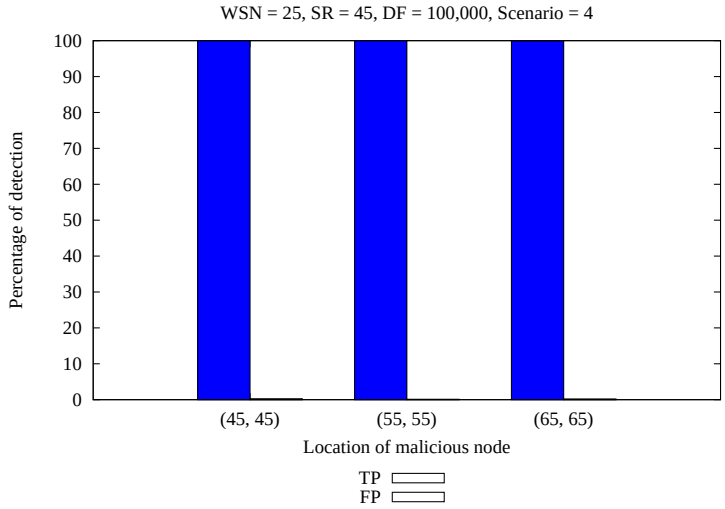
Figure 7.11: Effectiveness of malicious node in CTP Scenario 4 with IDS running

Success of Detecting Malicious Node

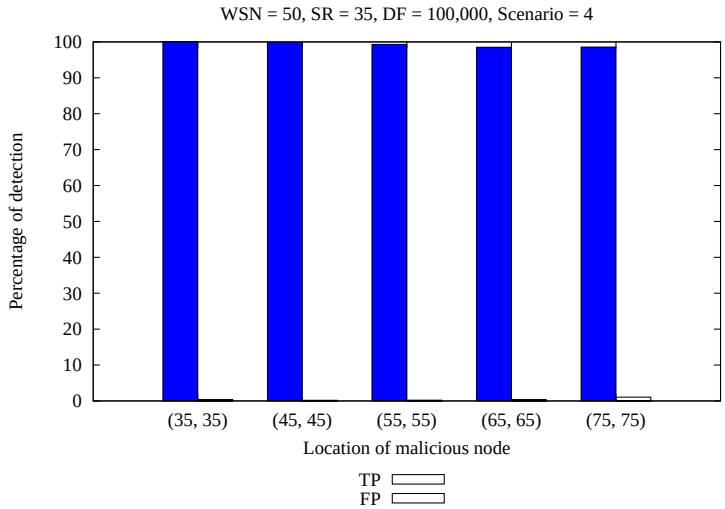
Figure 7.12 shows that `Test3()` achieves almost the same success percentage in Scenario 4 as it does in Scenario 3. However, some false detections do occur in Scenario 4 especially in far locations from the base station in dense WSNs. Collisions lead to resetting the beacon transmission intervals or missing the appropriate beacons to test.

7.5 Summary

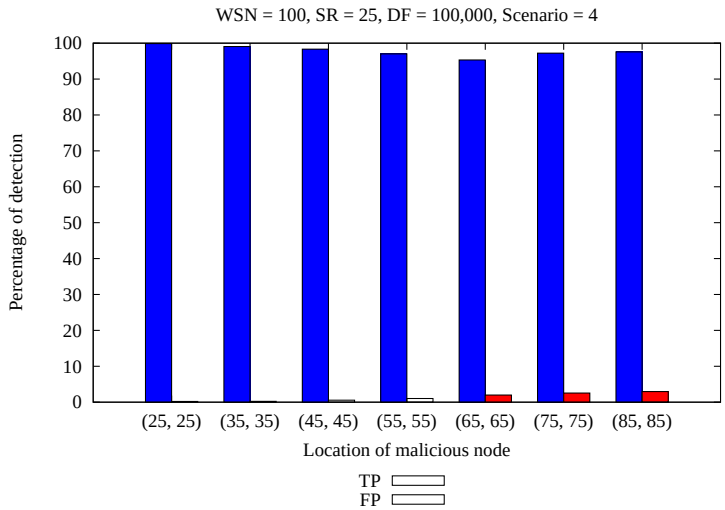
This Chapter presented and discussed the simulation results of attack scenarios on CTP routing protocol. It began by explaining the extensions made to ns-2 to support CTP and the proposed IDS followed by explaining the setup of the simulation environment. Three attack scenarios on CTP were discussed. In the first scenario, a malicious node broadcast a low path cost value to the base station. The effectiveness of the malicious node increased in locations far from the base station where the path costs were higher than the value offered by the malicious node. A detection module was proposed to detect this type of malicious node by comparing the path costs of children sensor nodes to their parent sensor nodes. A child sensor node must have a path cost greater than its parent's. The success of this module depended on the location of the malicious node and the density of the WSN. In the second attack scenario, the malicious node violated the beacon transmission intervals by using the minimum interval to send frequent beacons. The effectiveness of the malicious node depended on its location in the WSN. In locations close to the base station, the malicious node had low path cost so the frequent beacons helped in luring more sensor nodes. In far locations, the malicious node had high path cost so it is not as effective. A detection module was proposed that tests the arrival times of beacons. A node that did not adapt its beacon transmission interval was detected as malicious. The third scenario combined the previous two scenarios. The malicious node



(a) True & false detections in 25-node WSN



(b) True & false detections in 50-node WSN



(c) True & false detections in 100-node WSN

Figure 7.12: Success of detecting malicious node in CTP Scenario 4

in the last scenario was more vicious because it combined a low path cost value with frequent beacons. Due to its success, the second detection module was tested to detect the malicious node in the combined scenario.

The next Chapter concludes the dissertation and suggests directions for future work.

Chapter 8

CONCLUSIONS AND FUTURE WORK

8.1 Summary

WSNs are becoming an important component in our daily life. They are used in a broad range of applications, such as military, medical, or environmental monitoring. They are composed of resource-constrained devices called sensor nodes that communicate wirelessly. These sensor nodes are scattered in an environment to collect data from their surroundings. To relay the collected data to a central controller, the sensor nodes depend on a routing protocol. Routing protocols for WSNs sacrifice the security aspects to meet the limited resources of the sensor nodes. Hence, WSNs are susceptible to different types of routing attacks, ranging from stealing collected data to injecting false data.

Routing protocols for WSNs aim to deliver data to a central controller reliably. To achieve an acceptable level of reliability, routing protocols have to fulfill certain cost metrics. Some routing protocols are concerned with conserving the power resources of the sensor nodes. Others are concerned with choosing routes that achieve the lowest latency to deliver data to the central controller. Others choose routes that comprise reliable wireless links. This dissertation studies protocols of the third type.

Link quality routing protocols choose routes to the base station that have reliable links. This reliability can be measured as the ratio of packet reception or the number of required transmissions and retransmissions. These protocols may require the cooperation of the sensor nodes to compute link qualities or the sensor nodes may compute the qualities independently. This dissertation studies MintRoute as a representative of the former group, and CTP as a representative of the latter group.

Analysis and simulation are adopted as the research methodology in this dissertation. The two routing protocols are analyzed for possible vulnerabilities that may be used to exploit the computations of link qualities. Then detection mechanisms are proposed to detect malicious nodes that exploit these vulnerabilities. Finally, both protocols and their proposed IDSs are simulated in ns-2 to detect the effectiveness of the IDSs. The effectiveness of the malicious node is measured as the percentage of data delivered to the base station when the malicious node is configured to drop data packets. The success of the IDSs is measured as the percentage of true detections versus the percentage of false detections.

8.1.1 Cooperative Link Quality Routing Protocols

Chapter 4 discusses a vulnerability in cooperative link quality routing protocols that a malicious node can use to influence the computations of link qualities to its favour. MintRoute is chosen as a case study of this type of link quality protocols for WSNs. The sensor nodes in a WSN using MintRoute cooperate to compute bidirectional link qualities without any trust mechanism. Thus, if a sensor node advertises an incorrect link quality, no sensor node can refute or validate the claimed link quality. The malicious node may exaggerate the values of its link qualities to convince its neighbouring sensor nodes that it has a better route to the base station, thus luring them to send their traffic through it.

Then an IDS is proposed to detect any malicious nodes that may use this vulnerability. The proposed IDS uses the sequence numbers of MintRoute packets to validate the advertised link qualities. Sensor nodes can compute an upper bound on the advertised link qualities by introducing an artificial gap in their sequence numbers. This gap in the sequence numbers gives the tricking sensor node a minimum view of what its neighbours perceive as missed packets. If the neighbours only miss the size of the sequence number

gap, then their advertised link qualities will match the expected link qualities. Otherwise, their link qualities will be different than the expected link qualities. Each sensor node in the WSN runs a local copy of the proposed IDS and each reaches a decision about a malicious node on its own. Therefore, the proposed IDS does not add any extra communications in the WSN, which is a great asset. However, it requires extra storage space for an additional 98 bytes per sensor node.

Chapter 4 ends with discussing possible scenarios that a malicious node may exhibit and how the proposed IDS is enhanced to face them. We assume that the malicious node is adaptive and that it guesses the neighbouring sensor nodes that perform the sequence number gap trick. By this guess, the malicious node may avoid being detected by advertising the correct link qualities. Accordingly, the proposed IDS is adapted to detect the malicious node even when no sensor node is performing the sequence number gap trick.

Chapter 5 discusses the simulation results of MintRoute IDS. If MintRoute IDS succeeds in detecting the malicious node, then the children of the malicious node will change their routes to good ones. Thus, the percentage of data delivery to the base station shall increase. The simulation results show that MintRoute IDS is very successful when the malicious node is a simple one that always advertises exaggerated link qualities. However, the performance of the IDS degrades when the malicious node is adaptive. The malicious node tries to escape detection by building thresholds for the expected number of packets from its neighbours. If the number of packets exceeds the threshold, the malicious node will advertise the true link quality to the corresponding neighbour. The tighter the threshold is, the lower the performance of the IDS will be. However, the tighter threshold means that sensor nodes may not choose the malicious node route if its link qualities are not good enough. This degradation in performance is very apparent with the large sequence number gaps because it is easier to guess that the sequence

number gap trick is being played. The simulation results of the enhanced IDS where the sensor nodes test the link qualities at all times show that the enhanced IDS outperforms the basic IDS with the different thresholds of the malicious node.

MintRoute IDS has shown far more true detections than false detections. However, the average number of true detections decreases with the tighter threshold of the malicious node, but the enhanced IDS boosts the number back. The simulation results show that large sequence number gaps increase false detections because low link qualities are not advertised, which cause mismatches between expected link qualities and advertised link qualities. Also, the enhanced IDS has shown an increase in the average number of false detections because all the advertisements are tested, and not only when the sequence number gap trick is being played.

8.1.2 Non-cooperative Link Quality Routing Protocols

Chapter 6 describes three possible scenarios that a malicious node may follow to exploit non-cooperative link quality routing protocols for WSNs. CTP is chosen as a case study of this type of link quality routing protocols. In addition, a combined scenario is explained. In CTP, sensor nodes do not cooperate to compute link qualities, but a malicious node may manipulate some parameters of CTP to influence the computations of other sensor nodes. In the first scenario, the malicious node advertises a low path cost value to the base station by lying about the link quality to its parent. In the second scenario, the malicious node advertises a low total path cost to the base station. In the third scenario, the malicious node sends frequent beacons by fixing the beacon transmission intervals and not adapting them with every beacon. The final scenario combines the second and third scenarios. The aim of the malicious node in all scenarios is to convince more sensor nodes that it has a low path cost to the base station.

An IDS that is composed of three detection modules is proposed to detect the attack

scenarios on CTP. The first detection module detects the first scenario by implementing the sequence number gap trick. However, since the sensor nodes do not share their link quality computations, only the parent of the sensor node can implement the trick and test the advertised path cost of the malicious node. The second detection module detects the second scenario by testing the following property: the path cost of a child sensor node must be greater than the path cost of its parent. The third detection module implements a state machine to detect the third scenario. The state machine tests the arrival times of beacons and checks that they follow an adaptive behaviour. The combined scenario can be detected by either the second detection module or the third detection module.

Chapter 7 presents the results of simulating CTP and its IDS. The malicious node in Scenario 1 failed to convince its neighbouring sensor nodes of its path to the base station. Basically, lying about the link quality to its parent is not effective because this link constitutes one portion of the total path to the base station. However, in the second scenario when the malicious node lies about its total path cost to the base station, it lures more sensor nodes especially in locations far from the base station. When the malicious node offers a low path cost in a vicinity where the path costs are high, it will be easier to lure more sensor nodes. Sending frequent beacons only in the third scenario will not be beneficial if the path cost of the malicious node is high. So, in locations far from the base station the malicious node is not as successful as in close locations because its path cost is higher. A malicious node that combines the second and third scenarios succeeds in luring more sensor nodes.

The success of the second detection module to detect the malicious node that follows the second scenario depends on the value of path cost offered by the malicious node and the density of the network. If the malicious node is offering a path cost that is greater than its parent's, then the IDS will not be able to detect it. This case occurs close to the base station where the sensor nodes have low path costs. In far locations, it will be

easy to detect the malicious node if it is offering a lower path cost than the sensor nodes in vicinity. Also, the number of sensor nodes in the WSN affects the detection since the second detection module depends on the number of common neighbours between the malicious node and its parent. If this number is large, then the IDS will detect the malicious node easily. However, the effect of this detection on the malicious node depends on the number of common neighbours that are children of the malicious node. If the number of children that detect the malicious node is large, then they will change their parent to a good one.

The success of the third detection module depends on the amount of traffic in the WSN. More traffic leads to more packet collisions, which causes the third detection module to reset the detection process. Thus, the third detection module has its lowest detection in dense networks where more sensor nodes generate more traffic.

8.2 Discussion

This dissertation shows that sensor nodes can detect a malicious node that violates the link quality routing protocols independently and without sharing any information or forwarding it to the base station. In addition, it has shown that cryptography is not required to ensure data availability.

In the case of routing protocols that require cooperation between the sensor nodes to compute link qualities, the sensor nodes can introduce a gap in their sequence numbers to compute expected link quality values for their neighbours. Any neighbour that shares a link quality that does not conform with its corresponding expected link quality value will be detected as malicious.

In the case of routing protocols that do not require the sensor nodes to cooperate to compute link qualities, the sensor nodes can use the watchdog concept or a state machine

to detect a malicious node that uses incorrect values for the parameters of the link quality protocol.

Since the proposed detection mechanisms for both types of routing protocols do not require the sensor nodes to share detection results, no extra communication is generated in the WSNs. Thus, both IDSs are energy efficient. In addition, no cryptographic mechanisms are necessary to secure transmission of detection results.

MintRoute IDS tests neighbouring sensor nodes that are in the routing table only. Other neighbouring sensor nodes are not considered for parent choice and thus, there is no need to test their link qualities. In total, MintRoute IDS adds 98 bytes to the memory of the sensor nodes to operate efficiently. Thus, MintRoute IDS scales constantly with the size of the WSN.

On the other hand, CTP IDS depends on the history of beacons broadcast of each neighbouring sensor node. It needs to keep track of last beacon broadcast of all neighbouring sensor nodes and not just the ones in the routing table. To achieve this, CTP IDS creates an additional table to keep this information. Each entry in the new table requires 18 bytes. Hence, CTP IDS scales linearly with the size of the WSN.

8.3 Future Work

This dissertation focuses on detecting a single stationary malicious node that exploits the vulnerabilities of link quality routing protocols for WSN. The two types of link quality routing protocols are investigated for the possible vulnerabilities. An IDS is proposed to detect the vulnerabilities of each type of link quality routing protocols. Two routing protocols and the two IDSs are implemented and simulated in ns-2 to evaluate the performance of the IDSs. The simulation experiments show promising results. However, there are other topics that are worth investigation.

Larger WSN sizes

The proposed IDSs are tested in relatively small WSNs. A WSN may contain thousands of sensor nodes. As the number of sensor nodes gets larger, more packet collisions occur due to the larger traffic volume, and each sensor node has a larger number of neighbouring sensor nodes.

With more packet collisions, the IDSs may miss the appropriate packets for the detection test. This may lead to suspecting good sensor nodes as malicious or may lead to failing to detect the malicious node. Since both IDSs save extra information about the sensor nodes, the number of neighbours that a sensor node can track is limited by the available memory size. Accordingly, more simulation experiments are required to evaluate the effects of packet collisions and neighbourhood sizes on the performance of the IDSs.

Clustered WSNs

Some WSNs may use cluster-based or hierarchical routing protocols. Cluster-based routing protocols operate differently than the flat routing protocols studied in this dissertation. In cluster-based routing protocols, some sensor nodes, cluster heads, aggregate data collected from the members of their clusters and send one report to the base station. Cluster-based routing protocols put a challenge for the sensor nodes after they detect a malicious node that acts as a cluster head. In this dissertation, the sensor nodes are configured to change routes when they detect the malicious node. This behaviour gives us the ability to measure the effect of the malicious node before and after the detection. However, this behaviour is not feasible in cluster-based routing protocols because the sensor nodes have no other choice but to route their data through the malicious node. Post detection behaviour of sensor nodes is important to investigate to make sure that the sensor nodes can communicate with the rest of the WSN.

Multiple Malicious Nodes

The simulation results show that both IDSs are effective in detecting a single malicious node that exploits link quality routing protocols for WSNs. Multiple malicious nodes can pose a greater threat to WSNs. If they are implanted in an area that has a low density of sensor nodes, their detection may lead to undesirable consequences. First, if there is no other route to the base station, the good sensor nodes will have no choice but to route through the malicious nodes. Second, if few other routes exist, they may become congested if the good sensor nodes change their routes to them. The study of the behaviours of the good sensor nodes after detection is important.

Traffic-expelling Malicious Node

Another type of malicious nodes is a node that tries to expel or minimize the amount of traffic that passes through it. The malicious node tries to conserve its battery power to live longer than the other sensor nodes. It waits until some of its neighbouring sensor nodes consume their batteries and die. Afterwards, the malicious node can claim the IDs of the dead neighbours (Sybil attack) or attract the traffic from the remaining neighbours (sinkhole attack). To expel traffic, the malicious node needs only to announce low link qualities. When the neighbouring sensor nodes compute route costs to the base station, the route through the malicious node will have a low link quality and will not be chosen. When the malicious node decides the time to start its attack, it can announce or exaggerate its true link qualities.

Bibliography

[1] J. N. Al-Karaki and A. E. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey," *IEEE Wireless Communications*, vol. 11, pp. 6–28, December 2004.

[2] I. F. Akyildiz and M. C. Vuran, *Wireless Sensor Networks*. Ian F. Akyildiz Series in Communications and Networking, West Sussex, U.K.: John Wiley & Sons, 2010.

[3] A. Nayak and I. Stojmenovic, *Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordiantion and Data Communications*. Hoboken, NJ, USA: John Wiley & Sons, 2010.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, vol. 38, pp. 393–422, March 2002.

[5] I. Krontiris, T. Giannetsos, and T. Dimitriou, "LIDeA: A Distributed Lightweight Intrusion Detection Architecture for Sensor Networks," in *Proceedings of the 4^{th} International Conference on Security and Privacy in Communication Networks (SecureComm)*, (New York, NY, USA), pp. 20:1–20:10, ACM, 2008.

[6] S. Misra, K. I. Abraham, M. S. Obaidat, and P. V. Krishna, "LAID: A Learning Automata-based Scheme for Intrusion Detection in Wireless Sensor Networks," *Security and Communication Networks*, vol. 2, pp. 105–115, March/April 2009.

[7] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks," in *Proceedings of the 1^{st} International Conference on Embedded networked sensor systems (SenSys)*, (New York, NY, USA), pp. 14–27, ACM, 2003.

[8] A. Woo, *A Holistic Approach to Multihop Routing in Sensor Networks*. PhD thesis, Computer Science Division, University of California, Berkeley, CA, USA, 2004.

[9] B. Yu and B. Xiao, "Detecting Selective Forwarding Attacks in Wireless Sensor Networks," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, (Washington, DC, USA), pp. 351–351, IEEE Computer Society, 2006.

[10] E. C. H. Ngai, J. Liu, and M. R. Lyu, "On the Intruder Detection for Sinkhole Attack in Wireless Sensor Networks," in *Proceedings of IEEE International Conference on Communications (ICC)*, vol. 8, pp. 3383–3389, IEEE, 2006.

[11] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," in *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications (SNPA)*, pp. 113–127, IEEE, 2003.

[12] E. Shi and A. Perrig, "Designing Secure Sensor Networks," *IEEE Wireless Communications*, vol. 11, pp. 38–43, December 2004.

[13] F. Hu and X. Cao, *Wireless Sensor Networks: Principles and Practice*. Boca Raton, FL, USA: CRC Press, 2010.

[14] T. J. Dishongh and M. McGrath, *Wireless Sensor Networks for Healthcare Applications*. Norwood, MA, USA: Artech House, 2010.

[15] Crossbow technology, "http://www.xbow.com/."

[16] The Cricket Indoor Location System, "http://cricket.csail.mit.edu/."

[17] Sun SPOT World, "http://www.sunspotworld.com/."

[18] F. Zhao and L. J. Guibas, *Wireless Sensor Networks: An Information Processing Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2004.

[19] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. Boston, MA, USA: Pearson Education, 5^{th} ed., 2010.

[20] C. D. M. Cordeiro and D. P. Agrawal, *Ad Hoc & Sensor Networks: Theory and Applications*. Singapore: World Scientific Publishing, 2006.

[21] A. Hac, *Wireless Sensor Networks Designs*. West Sussex, UK: John Wiley & Sons, 2003.

[22] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Wiley Series on Wireless Communications and Mobile Computing, West Sussex, UK: John Wiley & Sons, 2010.

[23] TinyOS, "http://www.tinyos.net/."

[24] nesC, "http://nescc.sourceforge.net/."

[25] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming Language Design and Implementation (PLDI)*, (New York, NY, USA), pp. 1–11, ACM, 2003.

[26] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han, "MANTIS: System Support for MultimodAl NeTworks of In-situ Sensors," in *Proceedings of the 2^{nd} ACM international conference on Wireless sensor networks and applications (WSNA)*, (New York, NY, USA), pp. 50–59, ACM, 2003.

[27] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 29^{th} Annual IEEE International Conference on Local Computer Networks (LCN)*, (Washington, DC, USA), pp. 455–462, IEEE Computer Society, 2004.

[28] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A Dynamic Operating System for Sensor Nodes," in *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, (New York, NY, USA), pp. 163–176, ACM, 2005.

[29] T.-Y. Huang, K.-Y. Hou, H.-Y. Yu, E. T. Chu, and C. King, "LA-TinyOS: A Locality-Aware Operating System for Wireless Sensor Networks," in *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC)*, (New York, NY, USA), pp. 1151–1158, ACM, 2007.

[30] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. D. Kim, B. Zhou, and E. G. Sirer, "On the Need for System-Level Support for Ad hoc and Sensor Networks," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 1–5, April 2002.

[31] H. Cha, S. Choi, I. Jung, H. Kim, H. Shin, J. Yoo, and C. Yoon, "RETOS: Resilient, Expandable, and Threaded Operating System for Wireless Sensor Networks," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, (New York, NY, USA), pp. 148–157, ACM, 2007.

[32] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 4th ed., 2011.

[33] I. Stojmenovic, *Handbook of Sensor Networks: Algorithms and Architectures*. Wiley Series on Parallel and Distributed Computing, Hoboken, NJ, USA: John Wiley & Sons, 2005.

[34] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton, "CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care," in *Proceedings of the Workshop on Applications of Mobile Embedded Systems (WAMES)*, 2004.

[35] T. Gao, C. Pesto, L. Selavo, Y. Chen, J. Ko, J. Lim, A. Terzis, A. Watt, J. Jeng, B. rong Chen, K. Lorincz, and M. Welsh, "Wireless Medical Sensor Networks in Emergency Response: Implementation and Pilot Results," in *Proceedings of the 2008 IEEE Conference on Technologies for Homeland Security (HST)*, pp. 187–192, IEEE, 2008.

[36] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, (New York, NY, USA), pp. 88–97, ACM, 2002.

[37] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein, "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," *ACM SIGPLAN Notices*, vol. 37, pp. 96–107, October 2002.

[38] A. Othman, K. Lee, H. Zen, W. Zainal, and M. M. Sabri, "Wireless Sensor Networks for Swift Bird Farms Monitoring," in *Proceedings of the International Conference on Ultra Modern Telecommunications & Workshops (ICUMT)*, pp. 1–7, IEEE, 2009.

[39] J. J. Geoffrey Werner-Allen, M. Ruiz, J. Lees, and M. Welsh, "Monitoring Volcanic Eruptions with a Wireless Sensor Network," in *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN)*, pp. 108–120, IEEE, 2005.

[40] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees, "Deploying a Wireless Sensor Network on an Active Volcano," *IEEE Internet Computing*, vol. 10, pp. 18–25, March 2006.

[41] Center for Coastal Margin Observation and Prediction, "http://www.ccalmr.ogi.edu/CORIE/."

[42] E. A. Basha, S. Ravela, and D. Rus, "Model-Based Monitoring for Early Warning Flood Detection," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys)*, (New York, NY, USA), pp. 295–308, ACM, 2008.

[43] Y. Li, M. T. Thai, and W. Wu, *Wireless Sensor Networks and Applications*. Signals and Communication Technology, New York, NY, USA: Springer, 2008.

[44] Boomerang Shooter Detection System, "http://www.bbn.com/products_and_services /boomerang/."

[45] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, P. V. Qing Cao, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance," *ACM Transactions on Sensor Networks*, vol. 2, pp. 1–38, February 2006.

[46] H. Bai, M. Atiquzzaman, and D. Lilja, "Wireless Sensor Network for Aircraft Health Monitoring," in *Proceedings of the 1st International Conference on Broadband Networks (BroadNets)*, pp. 748–750, IEEE, 2004.

[47] K. Chintalapudi, J. P. Tat Fu, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri, "Monitoring Civil Structures with a Wireless Sensor Network," *IEEE Internet Computing*, vol. 10, pp. 26–34, March 2006.

[48] Y. Kim, T. Schmid, Z. M. Charbiwala, J. Friedman, and M. B. Srivastava, "NAWMS: Nonintrusive Autonomous Water Monitoring System," in *Proceedings of the 6th ACM Conference on Embedded Setwork Sensor Systems (SenSys)*, (New York, NY, USA), pp. 309–322, ACM, 2008.

[49] J. Vetelino and A. Reghu, *Introduction to Sensors*. Boca Raton, Florida, USA: CRC Press, 2011.

[50] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less Low-Cost Outdoor Localization for Very Small Devices," *IEEE Personal Communications*, vol. 7, pp. 28–34, October 2000.

[51] J. A. Stankovic, "Wireless Sensor Networks," technical report, Department of Computer Science, University of Virginia, Charlottesville, VA, USA, June 2006.

[52] M. Ilyas and I. Mahgoub, *Handbook of Sensor Networks Compact Wireless and Wired Sensing Systems*. Boca Raton, FL, USA: CRC Press, 2005.

[53] G. Acs and L. Buttyan, "A Taxonomy of Routing Protocols for Wireless Sensor Networks," *Hiradstechnika*, vol. LXII, pp. 32–40, January 2007.

[54] J. J. Lotf and S. H. H. N. Ghazani, "Overview on Routing Protocols in Wireless Sensor Networks," in *Proceedings of the 2^{nd} International Conference on Computer Engineering and Technology (ICCET)*, vol. 3, pp. V3–610–V3–614, 2010.

[55] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," in *Proceedings of the 5^{th} Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, (New York, NY, USA), pp. 174–185, ACM, 1999.

[56] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *Proceedings of the 6^{th} Annual International Conference on Mobile Computing and Networking (MobiCom)*, (New York, NY, USA), pp. 56–67, ACM, 2000.

[57] R. C. Shah and J. M. Rabaey, "Energy Aware Routing for Low Energy Ad Hoc Sensor Networks," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, vol. 1, pp. 350–355, IEEE, 2002.

[58] F. Ye, A. Chen, S. Lu, and L. Zhang, "A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks," in *Proceedings of the 10th International Conference on Computer Communications and Networks (ICCCN)*, pp. 304–309, IEEE, 2001.

[59] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," in *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, vol. 2, (Washington, DC, USA), IEEE Computer Society, 2000.

[60] Y. Yu, R. Govindan, and D. Estrin, "Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wreless Sensor Networks," Technical Report UCLA/CSD-TR-01-0023, Computer Science Department, University of California, Los Angeles, CA, USA, May 2001.

[61] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks," in *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, (Washington, DC, USA), pp. 46–55, IEEE Computer Society, 2003.

[62] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie, "Protocols for Self-Organization of a Wireless Sensor Network," *IEEE Personal Communications*, vol. 7, pp. 16–27, October 2000.

[63] TinyAODV implementation, TinyOS source code repository, "http://cvs.sourcefourge.net/viewcvs.py/tinyos/tinyos-1.x/contrib/hsn/."

[64] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, (New York, NY, USA), pp. 1–14, ACM, 2009.

[65] J. Moy, "Open Shortest Path First." RFC 2328, IETF, April 1998.

[66] C. Hedrick, "Routing Information Protocol." RFC 1058, IETF, June 1988.

[67] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, (New York, NY, USA), pp. 234–244, ACM, 1994.

[68] C. E. Perkins, E. M. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing." RFC 3561, IETF, July 2003.

[69] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, "A Taxonomy of Wireless Micro-Sensor Network Models," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, pp. 28–36, April 2002.

[70] O. Gnawali, M. Yarvis, J. Heidemann, and R. Govindan, "Interaction of Retransmission, Blacklisting, and Routing Metrics for Reliability in Sensor Network Routing," in *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, pp. 34–43, IEEE, 2004.

[71] I. Stojmenovic and X. Lin, "Power-Aware Localized Routing in Wireless Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, pp. 1122–1133, November 2001.

[72] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine*, vol. 40, pp. 102–114, August 2002.

[73] N. Baccour, A. Koubaa, M. B. Jamaa, H. Youssef, M. Zuniga, and M. Alves, "A Comparative Simulation Study of Link Quality Estimators in Wireless Sensor Networks," in *Proceedings of the 17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 1–10, IEEE, 2009.

[74] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for MultiHop Wireless Routing," in *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom)*, (New York, NY, USA), pp. 134–146, ACM, 2003.

[75] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin, "Temporal Properties of Low Power Wireless Links: Modeling and Implications on Multi-Hop Routing," in *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, (New York, NY, USA), pp. 414–425, ACM, 2005.

[76] J. Hill, "A Software Architecture Supporting Networked Sensors," Master's thesis, Deptartment of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA, 2000.

[77] J. Hill, R. Szewczky, A. Woo, S. Hollar, D. Vuller, and K. Pister, "System Architecture Directions for Networked Sensors," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pp. 93–104, 2000.

[78] S. Datema, "A Case Study of Wireless Sensor Network Attacks," Master's thesis, Delft University of Technology, Delft, Netherlands, September 2005.

[79] Y. Zhang and P. Kitsos, *Security in RFID and Sensor Networks.* Wireless Networks and Mobile Communications, Boca Raton, FL, USA: CRC Press, 2009.

[80] A. Perrig, J. Stankovic, and D. Wagner, "Security in Wireless Sensor Networks," *Communications of the ACM*, vol. 47, pp. 53–57, June 2004.

[81] P. Kamat, Y. Zhang, W. Trappe, and C. Ozturk, "Enhancing Source-Location Privacy in Sensor Network Routing," in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, (Washington, DC, USA), pp. 599–608, IEEE Computer Society, 2005.

[82] J. Deng, R. Han, and S. Mishra, "Countermeasures Against Traffic Analysis Attacks in Wireless Sensor Networks," in *Proceedings of the 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm)*, (Washington, DC, USA), pp. 113–126, IEEE Computer Society, 2005.

[83] Y. Wang, G. Attebury, and B. Ramamurthy, "A Survey of Security Issues in Wireless Sensor Networks," *IEEE Communications Surveys & Tutorials*, vol. 8, 2nd Quarter 2006.

[84] I. Krontiris, T. Dimitriou, and F. C. Freiling, "Towards Intrusion Detection in Wireless Sensor Networks," in *Proceedings of the 13th European Wireless Conference (EW)*, 2007.

[85] I. Onat and A. Miri, "An Intrusion Detection System for Wireless Sensor Networks," in *Proceedings of IEEE International Conference on Wireless and Mobile Computing, Networking and Communication (WiMob)*, vol. 3, pp. 253–259, IEEE, 2005.

[86] I. Krontiris, T. Dimitriou, T. Giannetsos, and M. Mpasoukos, "Intrusion Detection of Sinkhole Attacks in Wireless Sensor Networks," in *Proceedings of the 3rd*

International Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors), LNCS, (Berlin / Heidelberg, Germany), Springer, 2007.

[87] J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil Attack in Sensor Networks: Analysis & Defenses," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN)*, (New York, NY, USA), pp. 259–268, ACM, 2004.

[88] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3, pp. 1976–1986, IEEE, 2003.

[89] Z. Li and G. Gong, "A Survey on Security in Wireless Sensor Networks," Technical Report CACR 2008-20, Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada, 2008.

[90] C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, (New York, NY, USA), pp. 162–175, ACM, 2004.

[91] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu, "Analyzing and Modeling Encryption Overhead for Sensor Network Nodes," in *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA)*, (New York, NY, USA), pp. 151–159, ACM, 2003.

[92] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transaction on Information Theory*, vol. 22, pp. 644–654, November 1976.

[93] D. Liu and P. Ning, *Security for Wireless Sensor Networks*. New York, NY, USA: Springer, 2007.

[94] M. Eltoweissy, M. Moharrum, and R. Mukkamal, "Dynamic Key Management in Sensor Networks," *IEEE Communications Magazine*, vol. 44, pp. 122–130, April 2006.

[95] L. Eschenauer and V. D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, (New York, NY, USA), pp. 41–47, ACM, 2002.

[96] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy (SSP)*, (Washington, DC, USA), pp. 197–213, IEEE Computer Society, 2003.

[97] T. M. Vu, R. Safavi-Naini, and C. Williamson, "Securing Wireless Sensor Networks against Large-scale Node Capture Attacks," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, (New York, NY, USA), pp. 112–123, ACM, 2010.

[98] C. Kuo, M. Luk, R. Negi, and A. Perrig, "Message-In-a-Bottle: User-Friendly and Secure Key Deployment for Sensor Nodes," in *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys)*, (New York, NY, USA), ACM, 2007.

[99] M. Eltoweissy, M. H. Heydari, L. Morales, and I. H. Sudborough, "Combinatorial Optimization of Group Key Management," *Journal of Network and Systems Management*, vol. 12, pp. 33–50, March 2004.

[100] M. Eltoweissy, A. Wadaa, S. Olariu, and L. Wilson, "Group Key Management

Scheme for Large-Scale Wireless Sensor Networks," *Ad Hoc Networks*, vol. 3, pp. 668–688, September 2005.

[101] M. F. Younis, K. Ghumman, and M. Eltoweissy, "Location-Aware Combinatorial Key Management Scheme for Clustered Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, pp. 865–882, August 2006.

[102] G. Jolly, M. C. Kuscu, P. Kokate, and M. Younis, "A Low-Energy Key Management Protocol for Wireless Sensor Networks," in *Proceedings of the 8^{th} IEEE International Symposium on Computers and Communication (ISCC)*, vol. 1, pp. 335–340, IEEE, 2003.

[103] A. Seshadri, M. Luk, and A. Perrig, "SAKE: Software Attestation for Key Establishment in Sensor Networks," in *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, vol. 5067 of *LNCS*, (Berlin / Heidelberg, Germany), pp. 372–385, Springer, 2008.

[104] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Hoboken, NJ, USA: John Wiley & Sons, 2^{nd} ed., 1996.

[105] J. Sen, "A Survey on Wireless Sensor Network Security," *International Journal of Communication Networks and Information Security*, vol. 1, pp. 55–78, August 2009.

[106] J. Zhang and V. Varadharajan, "Wireless Sensor Network Key Management Survey and Taxonomy," *Journal of Network and Computer Applications*, vol. 33, pp. 63–75, March 2010.

[107] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," in *Proceedings of the Workshop*

on Cryptographic Hardware and Embedded Systems (CHES), vol. 3156 of *LNCS*, (Berlin / Heidelberg, Germany), pp. 925–943, Springer, 2004.

[108] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.

[109] R. L. Rivest, "The RC5 Encryption Algorithm," in *Proceedings of the 2nd International Workshop on Fast Software Encryption (FSE)*, vol. 1008 of *LNCS*, (Berlin / Heidelberg, Germany), pp. 86–96, Springer, 1995.

[110] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)." RFC 3174, IETF, September 2001.

[111] R. Rivest, "The MD5 Message-Digest Algorithm." RFC 1321, IETF, April 1992.

[112] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pp. 189–199, ACM, 2001.

[113] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: A Secure Sensor Network Communication Architecture," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, (New York, NY, USA), pp. 479–488, ACM, 2007.

[114] A. A. Ghorbani, W. Lu, and M. Tavallaee, *Network Intrusion Detection and Prevention: Concepts and Techniques*, vol. 47 of *Advances in Information Security*. New York, NY, USA: Springer, 2010.

[115] W. Wang and B. Bhargava, "Visualization of Wormholes in Sensor Networks," in *Proceedings of the ACM Workshop on Wireless Security (WiSe)*, (New York, NY,

USA), pp. 51–60, ACM, 2004.

[116] L. Buttyn, L. Dra, and I. Vajda, "Statistical Wormhole Detection in Sensor Networks," in *Proceedings of the 2nd European Workshop on Security and Privacy in Ad Hoc and Sensor Networks (ESAS)*, vol. 3813 of *LNCS*, (Berlin / Heidelberg, Germany), pp. 128–141, Springer, 2005.

[117] M. Demirbas and Y. Song, "An RSSI-based Scheme for Sybil Attack Detection in Wireless Sensor Networks," in *Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks (WOWMOM)*, (Washington, DC, USA), pp. 564–570, IEEE Computer Society, 2006.

[118] R. deGraaf, I. Hegazy, J. Horton, and R. Safavi-Naini, "Distributed Detection of Wormhole Attacks in Wireless Sensor Networks," in *Proceedings of the 1st International Conference on Ad Hoc Networks*, vol. 28 of *LNICST*, (Berlin / Heidelberg, Germany), pp. 208–223, Springer, 2009.

[119] A. Woo and D. Culler, "Evaluation of Efficient Link Reliability Estimators for Low-Power Wireless Networks," Technical Report UCB/CSD-03-1270, Electrical Engineering and Computer Science Department, University of California, Berkeley, CA, USA, 2003.

[120] T. Liu, A. Kamthe, L. Jiang, and A. Cerpa, "Performance Evaluation of Link Quality Estimation Metrics for Static Multihop Wireless Sensor Networks," in *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 1–9, IEEE Communications Society, 2009.

[121] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, "Four-Bit Wireless Link Estimation," in *Proceedings of the 6th Workshop on Hot Topics in Networks (HotNets)*,

2007.

[122] Network Simulator, ns-2, "http://www.isi.edu/nsnam/ns/."

[123] Tool Command Language, Tcl, "http://www.tcl.tk/."

[124] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency Estimation of Internet Packet Streams with Limited Space," in *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, (London, UK), pp. 348–360, Springer-Verlag, 2002.

[125] I. Hegazy, R. Safavi-Naini, and C. Williamson, "Towards Securing MintRoute in Wireless Sensor Networks," in *Proceedings of the 2010 IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, pp. 1–6, IEEE, 2010.

[126] MintRoute Protocol, "http://www.tinyos.net/tinyos-1.x/tos/lib/MintRoute/."

[127] Castalia A simulator for WSNs, "http://castalia.npc.nicta.com.au/."

[128] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo, *TinyOS Enhancement Proposal (TEP) 123: The Collection Tree Protocol (CTP)*, August 2006.

[129] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, *TinyOS Enhancement Proposal (TEP) 119: Collection*, February 2006.

[130] U. Colesanti and S. Santini, "A Performance Evaluation of the Collection Tree Protocol Based on its Implementation for the Castalia Wireless Sensor Networks Simulator," Technical Report 681, Department of Computer Science, ETH, Zurich, Switzerland, August 31 2010.

[131] O. Gnawali, R. Fonseca, K. Jamieson, and P. Levis, "CTP: Robust and Efficient Collection through Control and Data Plane Integration," Technical Report SING-08-02, University of Southern California, UC Berkeley, MIT Computer Science and Artificial Intelligence Laboratory, Stanford University, USA, 2008.

[132] O. Gnawali, *TinyOS Enhancement Proposal (TEP) 124: The Link Estimation Exchange Protocol (LEEP)*, February 2006.

[133] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, (Berkeley, CA, USA), pp. 15–28, USENIX Association, 2004.

[134] The Collection Tree Protocol (CTP), "http://www.tinyos.net/tinyos-2.x/tos/lib/net/ctp/."

[135] I. Hegazy, R. Safavi-Naini, and C. Williamson, "Exploiting Routing Tree Construction in CTP," in *Proceedings of the 12th International Workshop on Information Security Applications (WISA)*, LNCS, (Berlin / Heidelberg, Germany), Springer, August 2011.

[136] Y. Pan and Y. Xiao, *Ad Hoc and Sensor Networks*, vol. 2 of *Wireless Networks and Mobile Computing*. New York, NY, USA: Nova Science Publishers, 2006.

[137] P. Levis, *TinyOS Programming*, October 2006.

[138] P. J. Denning, "The Working Set Model for Program Behavior," *Communications of the ACM*, vol. 11, pp. 323–333, May 1968.

[139] A. A. Puntambekar, *Data Structures and Algorithms*. Pune, India: Technical Publications Pune, 2009.

Appendix A

A.1 Algorithm of RandomMLCG

The algorithm of acRandomMLCG comprises three routines. The first routine, *InitRandomMLCG()*, initializes the seed of the random number generator. The second routine, *Rand16()*, returns a two-byte random number and, finally, the third routine, *Rand32()*, returns a four-byte random number.

Algorithm A.1 Routines of the RandomMLCG algorithm

```
procedure InitRandomMLCG(s):
    SET seed TO (uint32_t)(s + 1)

procedure Rand16():
    return (uint16_t)Ran32()

procedure Rand32():
    SET tmpSeed TO (uint64_t)(33614U × (uint64_t)seed)
    SET q TO (uint32_t)tmpSeed
    SET q TO q >> 1
    SET p TO (uint32_t)(tmpSeed >> 32)
    SET mlcg TO p + q
    SET value TO ((mlcg & 0x80000000) ≠ 0)
    if value is true then
        SET mlcg TO mlcg & 0x7FFFFFFF
        SET mlcg TO mlcg + 1
    SET seed TO mlcg
    RETURN mlcg
```

A.2 Algorithm of the Queue

The queue algorithm consists of three routines that implement a simple FIFO queue. To utilize the space efficiently, the queue algorithms implement a circular queue [139]. The

first routine, *Enqueue*(), simply adds an element to the end of the queue. The second routine, *Dequeue*(), returns the first element of the queue and empties its space. The last routine, *Element*(), returns the queue element at the *index* position in the queue without deleting it. In the case of CTP, the queue element is of type CTP data packet.

Algorithm A.2 Routines of the queue algorithm

```
procedure ENQUEUE(queue element):
    if size of queue ≤ maximum queue size then
        ADD queue element at tail of queue
        INCREMENT tail of queue
        if tail = maximum queue size then
            SET tail TO 0
        INCREMENT size of queue
        RETURN SUCCESS
    else
        RETURN FAIL

procedure DEQUEUE():
    SET t TO element at head of queue
    if queue is not empty then
        INCREMENT head of queue ▷ it is a circular queue so move the head forward
        if head of queue = maximum queue size then
            SET head of queue TO 0
        DECREMENT size of queue
    RETURN t

procedure ELEMENT(index):
    SET index TO index + head of queue
    if index ≥ maximum queue size then
        SET index TO index - maximum queue size
    RETURN element from queue at index
```

A.3 Algorithm of the Message Pool

The message pool algorithm implements a general dynamic memory allocation structure and it contains three routines. The *InitPool*() routine initializes the size of the pool.

Elements from the pool are returned with the *Get*() routine and new elements are added with the *Put*() routine. The pool element is of type CTP data packet.

Algorithm A.3 Routines of the message pool algorithm

```
procedure InitPool(S):
    SET size TO s
    SET free TO size
    SET index TO 0

procedure Get():
    if there is free space in the pool then
        SET rval TO element at index in the pool
        SET element at index TO null
        DECREMENT free
        INCREMENT index
        if index = size of pool then
            SET index TO 0
        RETURN rval
    RETURN NULL

procedure Put(newVal):
    if free ≥ size then
        RETURN fail
    else
        SET emptyIndex TO index + free
        if emptyIndex ≥ size then
            SET emptyIndex TO emptyIndex - size
        SET pool element at emptyIndex TO newVal
        INCREMENT free
        RETURN SUCCESS
```

A.4 Algorithm of the Message Cache

The message cache stores the signatures of the CTP data packets. The signature of a packet contains its origin, sequence number, type, and Time Has Lived (THL). The message cache algorithm contains 4 routines. *Init*() initializes the size of the cache. *Lookup*1() returns the index of a packet if the packet exists in the cache. *Insert*() inserts

a new packet in the cache. Inserting a new element in a full cache will replace the oldest element. Inserting an element already in the cache will update its signature and age. *Remove*() removes a packet from the cache.

Algorithm A.4 Routines of the LRU cache algorithm

```
procedure InitCache(S):
    SET size TO s
    SET first TO 0
    SET count TO 0

procedure Lookup1(message):
    for i = 0 to count-1 do
        SET index TO (i + first) mod size;
        if signature of message = signature of stored message at index then
            BREAK
    RETURN i

procedure Lookup2(message):
    RETURN (Lookup1(m) < count)

procedure Remove(index):
    if index ≥ count then
        RETURN
    if index = 0 then
        SET first TO (first + 1) mod size                ▷ shift all by moving first
    else
        for j = index to j < count-1 do                  ▷ shift all elements down
            MOVE cache element at [(j + first + 1) mod size] TO [(j + first) mod size]
    DECREMENT count

procedure Insert(message):
    if count = size then
        SET i TO Lookup2(message)  ▷ if message is in cache, remove it temporarily
        REMOVE (i mod count)           ▷ otherwise, remove the first item in cache
    STORE signature of message at location first + count in cache
    INCREMENT count
```