Detection of Wormhole Attack in Wireless Networks

Aninda Sarker Rahul
ID : 1304103

October, 2018

Bachelor of Science in Computer Science and Engineering

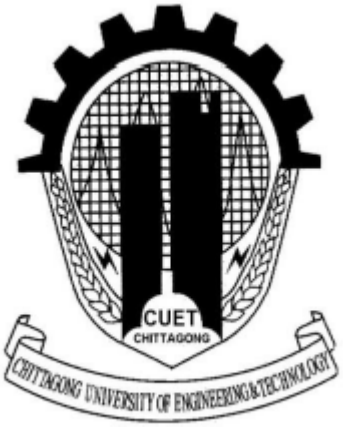
Detection of Wormhole Attack in Wireless Networks

Aninda Sarker Rahul
ID : 1304103

October, 2018

Department of Computer Science & Engineering
Chittagong University of Engineering & Technology
Chittagong-4349, Bangladesh.

Detection of Wormhole Attack in Wireless Networks



This thesis is submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science & Engineering.

Aninda Sarker Rahul

ID : 1304103

Supervised by
Mir Md. Saki Kowsar
Assistant Professor
Department of Computer Science & Engineering (CSE)
Chittagong University of Engineering & Technology (CUET)

Department of Computer Science & Engineering
Chittagong University of Engineering & Technology
Chittagong-4349, Bangladesh.

The thesis titled **"Detection of Wormhole Attack in Wireless Networks"** submitted by Roll No. 1304103, Session 2016-2017 has been accepted as satisfactory in fulfillment of the requirement for the degree of Bachelor of Science in Computer Science & Engineering (CSE) as B.Sc. Engineering to be awarded by the Chittagong University of Engineering & Technology (CUET).

Board of Examiners

1. _____ Chairman
Mir Md. Saki Kowsar
Assistant Professor
Department of Computer Science & Engineering (CSE)
Chittagong University of Engineering & Technology (CUET)

2. _____ Member
Professor Dr. Mohammed Moshiul Hoque (Ex-officio)
Head
Department of Computer Science & Engineering (CSE)
Chittagong University of Engineering & Technology (CUET)

3. _____ Member
Professor Dr. Md. Ibrahim Khan (External)
Professor
Department of Computer Science & Engineering (CSE)
Chittagong University of Engineering & Technology (CUET)

Statement of Originality

It is hereby declared that the contents of this thesis is original and any part of it has not been submitted elsewhere for the award of any degree or diploma.

Signature of the Supervisor
Date:

Signature of the Candidate
Date:

Acknowledgment

First of all thanks to God for his great blessings on me to complete this project successfully. There after I am expressing my gratitude to my honorable project Supervisor Mir Md. Saki Kowsar, Assistant Professor, Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, for his valuable suggestion, constructive advice, encouragement and sincere guidance in my entire project.

It was a real opportunity to work with him. I learned a lot from him. I am grateful that he invested so much of his precious time with us. It was really an amazing experience.

I am also expressing my gratitude to Professor Dr. Mohammed Moshiul Hoque, honorable head of the Department of Computer Science and Engineering for his kind support. I am highly indebted to all of my teachers for their valuable effort in teaching us for last four and a half years.

This thesis would not be possible without the support of our faculty staffs especially Mr. Osman Billah, Mr. Shafikul Islam, Mr. Liton Kar, Mr. Provatosh and others.

I would like to give thanks to my family, friends, seniors, juniors, batchmates for their constant support and motivation. I thank them for giving me confidence and drive for pursuing my degree.

Special thanks to Moinul Islam Bappi,Shourav Sinha Klington and Raihan Roman for working day and night for the completion of the thesis. I am looking forward to work with them in future again.

Abstract

Wireless Sensor Networks (WSNs) provide flexible infrastructures for numerous applications like healthcare,industry automation,surveillance and defence. Wormhole attack is one of the most dangerous attack which can distabilize or disable wireless sensor networks. In order to provide a stable and uninterrupted packet sending experience to a network, Wormhole attack detection is required to maintain the network connection stable. Ideally, Wormhole attack detection should be completely transparent to legitimate users in a network. However the current IEEE 802.11 standards do not detect the Wormhole attack well. In this thesis current network layer attacks scheme is analyzed and an efficient method is proposed. Finally, it is implemented in network simulator 3 and analyzed. The analysis shows that the proposed scheme reduces the packet loss and improves the overall network performance.

Existing solutions on reducing the packet loss in WSNs ignore one important factor for the long handoff delay.Data can be lost during the multihop transmission resulting in increase in packet loss and the data collection becomes incomplete[17]. Studies have revealed that standard hand-off on IEEE 802.11 WLANs increase a latency of the order of hundreds of milliseconds to several seconds. Moreover the discovery step in the handoff process accounts for more than 99% of this latency.

Ad-hoc wireless network signals are not really strong as compared to the wireless connections which uses routers to function properly. On discovery steps, multiple paths between source and destination is discovered using AODV routing protocol. Discovering multipath comes with a number of disadvantages like link failure, congestion error etc. To overcome those disadvantages, the AODV protocol is modified to select the main path for data transmission based on the time on the time of routing establishment

The feasibility of the proposed scheme to support fast handoff in WSNs has been demonstrated through computer simulations under different network conditions. The results from the simulations show that the latency associated with handoff can be reduced by using this technique. This scheme can improve the overall performance by increasing packet delivery fraction and throughput and reducing ETE delay.

In conclusion, it can be said that the latency in the link layer is reduced by introducing an efficient and powerful technique which also improves the overall performance.

Contents

List of Figures

List of Tables

List of Abbreviations

WSN	Wireless Sensor Network
SN	Sensor Node
MR	Mesh Router
MC	Mesh Client
AP	Access Point
AODV	Ad-hoc On-Demand Distance Vector
NS-3	Network Simulator 3

Chapter 1

Introduction

Wireless sensor networks(WSNs) consist of many interconnected self-controlled device(i.e sensor nodes) that are used in a collective manner to monitor and/or contorl environmental phenomena in local or remote environments[8]. Sensor nodes which are spacially distributed communicate with their peers in order to send aggregated data to the base station efficiently. WSN is a spacial kind of Mobile Ad-hoc network that has gained popularity for its versatile application in military and civil domains such as battlefield monitoring, tracking objects, healthcare and home automation. Due to the broadcast nature of the transmission medium and fact that sensor nodes often operate in hostile environments. WSN are vulnerable to variety of security attacks[1]. This chapter contais some introductory information on wormhole attack in wireless sensor network, motivation of our work, challanges of implementing our work and our objectives.

1.1 Wormhole Attack in Wireless Sensor Network

The wormhole attack is recognized as one of the most dangerous security threats for WSNs. This attack has one or more malicious node and a tunnel between them. The attacking nodes capture the packets from one location and transfers them to other distant location node which distributes them locally[1]. The tunnel can be established in many ways e.g in-band and out-of-band channel.Routing mechanisms which rely on the knowledge about distance between nodes can get confuse because because wormhole nodes fake a route that is shorter than the original one within the network[1].

Wireless sensor networks are susceptible to wide range of security attacks due to the multi-hop nature of the transmission medium. Also, wireless sensor networks have an additional vulnerability because nodes are generally deployed in a hostile or unprotected environment. From [2] a summarization of possible attacks in different layers with respect to ISO-OSI model are shown in the Table 1.1

Layer	Attacks
Physical Layer	**Denial of Service, Tampering**
Data Link Layer	**Jamming, Collision, Traffic manipulation**
Routing/Network Layer	**Wormhole, Sinkhole, Flooding**

Table 1.1: Different types of Layering based attacks

For the nature wireless transmission the attacker can create a wormhole even for packets not addressed to itself, since it can overhear them in wireless transmission and tunnel them in wireless transmission and tunnel them to the colluding attacker at the opposite end of wormhole[7].

However, in wireless sensor networks, mobility, limited bandwidth, routing functionalities etc. associated with each node, present many new opportunities for launching a Wormhole attack. Wormhole attack is classified into four models[3] :-

Encapsulation: Here a malicious node at one part of the network overhears the RREQ packet. It is then tunnel through a low latency link with the help of normal node, to the second colluding malicious node at a distance near to the destination node. Once this packet is received by the second malicious code, the legitimate neighbour of the node drops any further legitimate requests from a legitimate neighbour node. This result to the routes between the source and the destination go through the wormhole link, because it has broadcast itself has the fastest route. It prevents legitimate nodes from discovering legitimate paths more than two hops away.

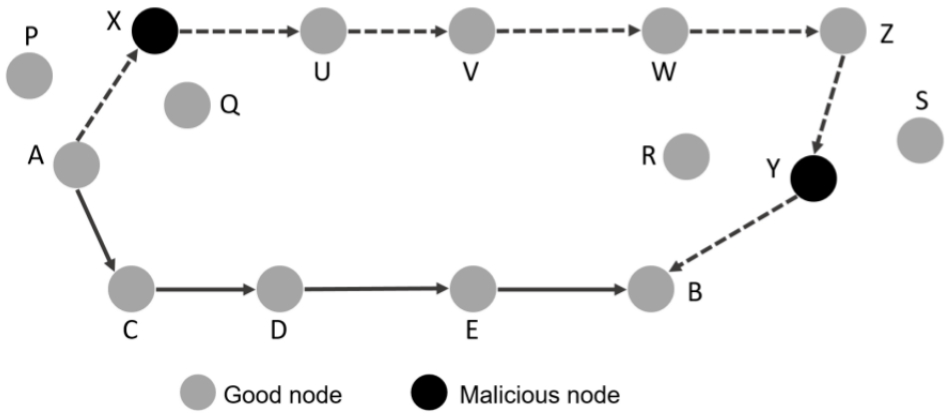


Figure 1.1: Encapsulation Wormhole

Packet Relay: This is another type of wormhole attack where malicious relays packet between source and destination nodes. Unlike encapsulation, this type of wormhole attack can be launched using only one malicious node.

Out-of-band Channel: As the name suggest is a type of worm-hole attack that uses a long range directional wireless link or a wired link. It is a very difficult attack to launch because its needs a specialized hardware.

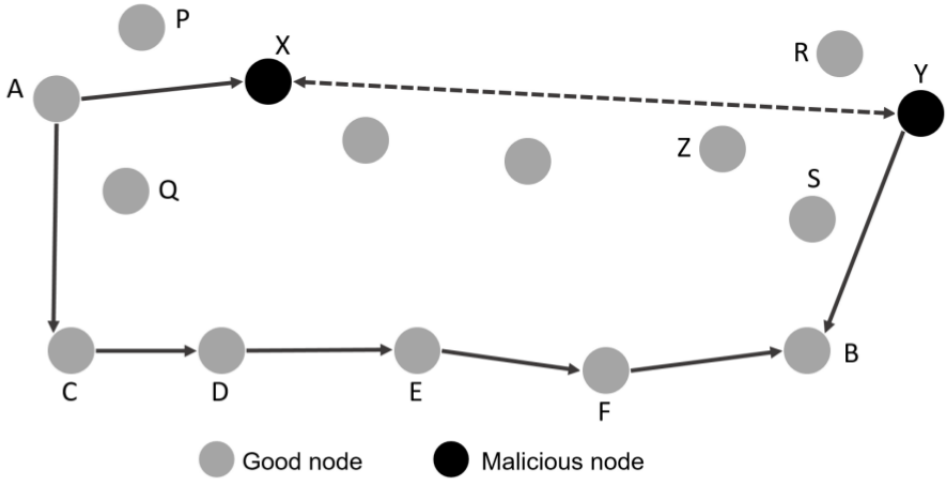


Figure 1.2: Out-of-band Wormhole

High Power Transmission: In this mode of attack, a single ma- licious node can create a wormhole without colluding node. when this single malicious node received a RREQ, it rebroadcasts the request at a very high power level capability compared to normal node, thereby attracting normal nodes to overhear this RREQ and further on broadcast the packet towards destination.

1.2 Background or Previous works

Security in WSNs is still in its infancy as very little attention has been devoted thus so far to this topic by the research community and so it has become more vulnerable to various types of attacks.

An efficient method for detecting wormhole attacks against the routing functionality of network is proposed in [11]. The author propose an algorithm which is meant to secure each link. In this algorithm, each node considers the distance which separates it from its direct neighbors. This estimate is performed using an exchange of message simultaneously by using a radio interferometry generating ultrasonic waves. Then each node exchanges the information of these values of calculated distances. Once these data are exchanged, each node runs a set of geometric tests on the local data thus obtained, in order to detect false links present due to the Wormhole attack. The disadvantage of this approach is that each node must be equipped with a second ultrasound radio, allowing the estimation of distances between neighboring nodes.

In paper[12], a statistical approach is proposed, known as SWAN, in which each sensor collects a recent number of neighbors. A wormhole attack is identified if the current number of neighbors exhibits an unusual increase, compared to the previous neighborhood counts taken outside of the wormhole zones. This is a distributed approach so that it doesn't cause any overhead, unlike a centralized approach. However, this schemes has been designed for and perform better in a uniformly distributed network, but their performance is in question for networks in which sensors are distributed non-uniformly.

In [6], Hu and Evans propose a solution to wormhole attacks for ad hoc networks in which all nodes are equipped with directional antennas. When directional antennas are used, nodes use specific 'sectors' of their antennas to communicate with each other. Therefore, a node receiving a message from its neighbor has some information about the location of that neighbor, which knows the relative orientation of the neighbor with respect to itself. This extra bit information makes wormhole discovery much easier than in networks with exclusively omni-directional antennas. This approach does not require either location information or clock synchronization, and is more efficient with energy. They use directional antenna and consider the packet arrival direction to defend the attacks. They use the neighbor verification methods and verified neighbors are really neighbors and only accept messages from verified neighbors. But it has the drawback that the need of the directional antenna is impossible for sensor networks.

HU et al [4] describe a defense based on the leashes of the packet, where the distance of a message route is limited, each message having a timestamp and a location of its transmitter. The receiver compares this information with its own location and timestamp to check if the intervals of transmissions are exceeded. However, this proposal presents two disadvantages: It requires a coordinate system such as the GPS in order to obtain the geographic information about each node; It requires a precise synchronization of clocks between different nodes in order to use timely data.

In WSNs are vulnerable to several kinds of attacks because of their inherent attributes such as the open communication medium. Malicious sensor devices can launch attacks to disrupt the network routing operations, then putting the entire sensor network at risk. Many techniques have been suggested for Wormhole attack detection and characterization. Most of these techniques have one or more limitations. Network visualization method can be effective against Wormhole attack but it requires central coordination and the mobility is not studied for this method.

1.3 Present state and Contribution

The objective of this thesis is to develop a Procedure to detect Wormhole attack in Infrastructure based WSNs. After a study and analysis of the wireless sensor network Wormhole attack detection procedure, the detection process was divided into two phases: neighborhood sensing and malicious node detection. A fast

Wormhole attack detection scheme have been developed to provide a novel use of the channel in wireless sensor network. This detection scheme may be implemented by upgrading the protocol of wireless sensor network, no hardware upgrade is required. NS-3 simulations were used in order to verify the feasibility of the proposed scheme. The results presented in chapter 5 indicate that the latency associated with Wormhole attack can be efficiently detect by using the proposed technique. The performance of the proposed scheme was also analyzed. The results show that the scheme continued to successfully operate under different network conditions.

1.4 Motivation

security in wireless sensor network system is one of the main concerns to provide protected communication between mobile nodes in strange environment. Unlike the wired line networks, the unique characteristics of WSN create a number of nontrivial challenges to security design like open peer-to-peer network architecture, shared wireless medium, inflexible resources constraints and highly dynamic network topology.

Guarding against Wormhole attack is a critical component of any WSN security system. Security services in WSNs are needed to protect from attacks and to ensure the security of the information. The wireless channel is accessible to both intended and unintended users. There is no well-defined place where traffic monitoring or access control mechanisms can be brought into life. As a result, there is no clear boundary that separates the inside network from the outside world.

1.5 Prospects of the problem

Our main prospects of this project is to detect the Wormhole attack in infrastructure base wireless sensor network and improve the overall performance.

1.6 Organization of the Project

The remainder of the report is organized as follows. In the next chapter, an overview of our project related terminologies and contains brief discussion on previous works that is already implemented with their limitations. Chapter three describes the working procedure of our proposed system. In Chapter 4, we have illustrated our implementation of the project in details. Chapter 5 focuses on the experimental result of the proposed system. The thesis concludes with a summary of research contributions and future plan of our work in chapter 6. This thesis contains an appendix intended for persons who wish to explore the source code.

Chapter 2

Literature Review

Wormhole attack detection is an essential issue to ensure continuous communications in wireless sensor networks (WSNs). The Wormhole attack performance in WSNs can be largely degraded by the packet loss which dropped the valuable information by neighborhood sensing at each sensor node, especially when the backbone traffic volume is high. In this chapter, we present studies on the terminologies related to the project which are important to understand. This chapter also contains brief discussion on previous works that is already implemented along with their limitations.

2.1 Wireless Sensor Network

A Wireless Sensor Network is one kind of wireless network includes a large number of circulating, self-directed, minute, low powered devices named sensor nodes called motes. These networks certainly cover a huge number of spatially distributed, little, battery-operated, embedded devices that are networked to caringly collect, process, and transfer data to the operators, and it has controlled the capabilities of computing & processing. Nodes are the tiny computers, which work jointly to form the networks.

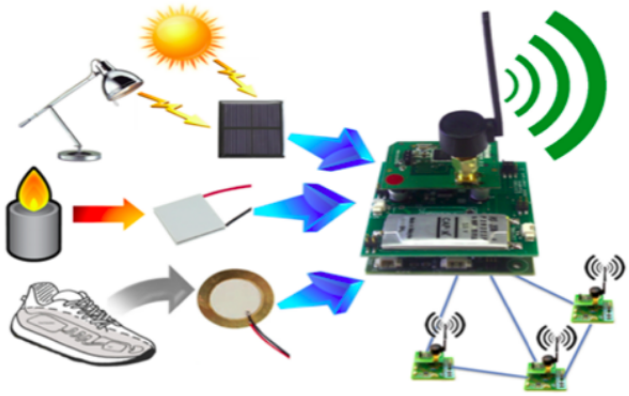


Figure 2.1: Wireless Sensor Network

The sensor node is a multi-functional, energy efficient wireless device. The applications of motes in industrial are widespread. A collection of sensor nodes collects the data from the surroundings to achieve specific application objectives. The communication between motes can be done with each other using transceivers. In a wireless sensor network, the number of motes can be in the order of hundreds/ even thousands. In contrast with sensor n/ws, Ad Hoc networks will have fewer nodes without any structure.

2.1.1 Network Architecture

The most common WSN architecture follows the OSI architecture Model. The architecture of the WSN includes five layers and three cross layers. Mostly in sensor n/w we require five layers, namely application, transport, n/w, data link & physical layer. The three cross planes are namely power management, mobility management, and task management. These layers of the WSN are used to accomplish the n/w and make the sensors work together in order to raise the complete efficiency of the network.

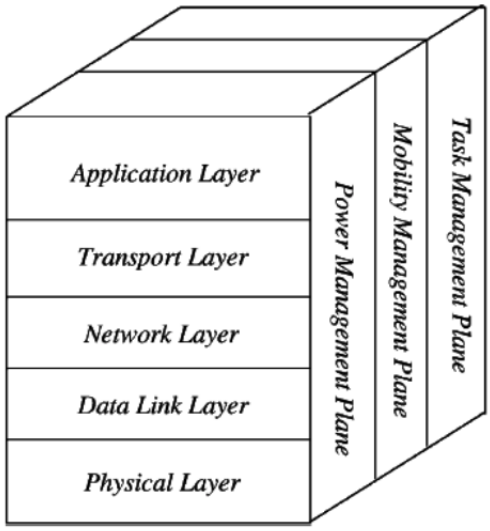


Figure 2.2: Wireless Sensor Network Architecture

Application Layer: The application layer is liable for traffic management and offers software for numerous applications that convert the data in a clear form to find positive information. Sensor networks arranged in numerous applications in different fields such as agricultural, military, environment, medical, etc.

Transport Layer: The function of the transport layer is to deliver congestion avoidance and reliability where a lot of protocols intended to offer this function are either practical on the upstream. These protocols use dissimilar mechanisms for loss recognition and loss recovery. The transport layer is exactly needed when a system is planned to contact other networks.

Network Layer: The main function of the network layer is routing, it has a lot of tasks based on the application, but actually, the main tasks are in the power conserving, partial memory, buffers, and sensor don't have a universal ID and have to be self-organized.

Data Link Layer: The data link layer is liable for multiplexing data frame detection, data streams, MAC, & error control, confirm the reliability of point–point (or) point– multipoint.

Physical Layer: The physical layer provides an edge for transferring a stream of bits above physical medium. This layer is responsible for the selection of frequency, generation of a carrier frequency, signal detection, Modulation & data encryption.

WSN Network Topologies

For radio communication networks, the structure of a WSN includes various topologies like the ones given below.

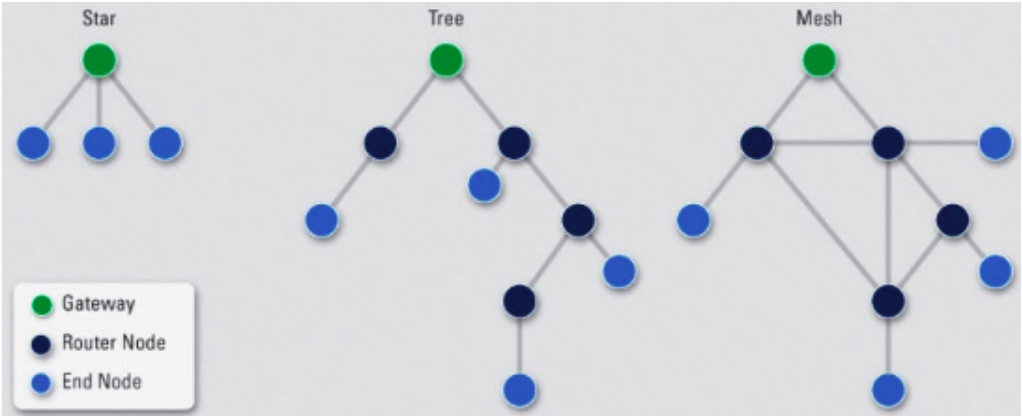


Figure 2.3: Wireless Sensor Network Topologies

Star Topologies: Star topology is a communication topology, where each node connects directly to a gateway. A single gateway can send or receive a message to a number of remote nodes. In star topologies, the nodes are not permitted to send messages to each other. This allows low-latency communications between the remote node and the gateway (base station).

Tree Topologies: Tree topology is also called as cascaded star topology. In tree topologies, each node connects to a node that is placed higher in the tree, and then to the gateway. The main advantage of the tree topology is that the expansion of a network can be easily possible, and also error detection becomes easy. The disadvantage with this network is that it relies heavily on the bus cable; if it breaks, all the network will collapse.

Mesh Topologies: The Mesh topologies allow transmission of data from one node to another, which is within its radio transmission range. If a node wants to send a message to another node, which is out of radio communication range, it needs an intermediate node to forward the message to the desired node. The advantage with this mesh topology includes easy isolation and detection of faults in the network. The disadvantage is that the network is large and requires huge investment.

2.1.2 Characteristics

There have been several characteristics of WSNs. Some of characteristics are explained as follows:

- **Low cost:** In the WSN normally hundreds or thousands of sensor nodes are deployed to measure any physical environment. In order to reduce the overall cost of the whole network the cost of the sensor node must be kept as low as possible.
- **Energy efficient:** Energy in WSN is used for different purpose such as computation, communication and storage. Sensor node consumes more energy compare to any other for communication. If they run out of the power they often become invalid as we do not have any option to recharge. So, the protocols and algorithm development should consider the power consumption in the design phase.
- **Computational power:** Normally the node has limited computational capabilities as the cost and energy need to be considered.
- **Communication Capabilities:** WSN typically communicate using radio waves over a wireless channel. It has the property of communicating in short range, with narrow and dynamic bandwidth. The communication channel can be either bidirectional or unidirectional. With the unattended and hostile operational environment it is difficult to run WSN smoothly. So, the hardware and software for communication must have to consider the robustness, security and resiliency.
- **Distributed sensing and processing:** the large number of sensor node is distributed uniformly or randomly. WSNs each node is capable of collecting, sorting, processing, aggregating and sending the data to the sink. Therefore the distributed sensing provides the robustness of the system.
- **Dynamic network topology:** In general WSN are dynamic network. The sensor node can fail for battery exhaustion or other circumstances, communication channel can be disrupted as well as the additional sensor node may be added to the network that result the frequent changes in the network topology. Thus, the WSN nodes have to be embedded with the function of reconfiguration, self adjustment.

- **Multi-hop communication:** A large number of sensor nodes are deployed in WSN. So, the feasible way to communicate with the sinker or base station is to take the help of a intermediate node through routing path. If one need to communicate with the other node or base station which is beyond its radio frequency it must me through the multi-hop route by intermediate node.
- **Application oriented:** WSN is different from the conventional network due to its nature. It is highly dependent on the application ranges from military, environmental as well as health sector. The nodes are deployed randomly and spanned depending on the type of use.
- **Robust Operations:** Since the sensors are going to be deployed over a large and sometimes hostile environment. So, the sensor nodes have to be fault and error tolerant. Therefore, sensor nodes need the ability to self-test, self-calibrate, and self repair.
- **Security and Privacy:** Each sensor node should have sufficient security mechanisms in order to prevent unauthorized access, attacks, and unintentional damage of the information inside of the sensor node. Furthermore, additional privacy mechanisms must also be included.
- **Small physical size:** sensor nodes are generally small in size with the restricted range. Due to its size its energy is limited which makes the communication capability low.

2.1.3 Advantages of Wireless Mesh Network

The advantages of WSN over other networks are very significant and have great deal of importance. There are some unique features compared to other network in WSN. These features are explained below:

- Network arrangements can be carried out without immovable infrastructure.
- Apt for the non-reachable places like mountains, over the sea, rural areas and deep forests.
- Flexible if there is a casual situation when an additional workstation is required.
- Execution pricing is inexpensive.
- It avoids plenty of wiring.
- The data processing is pretty fast.
- Easy installation and uninstall.
- It might provide accommodations for the new devices at any time.
- It can be opened by using a centralized monitoring.

Building a network without any wire brings a great deal of advantage. In most the case bigger network don't really use any wire. The internet we use in our daily life is a realistic example for this. It is seen that most of the network are inter connected with each other wirelessly creating a mesh topology which is also called seamlessly. It is cheap as it don't use any wire. In WSN the nodes automatically adjust themselves according to the situation, so there is no need for network administrator if there is a problem regarding nodes or the network. WSN nodes can communicate with their neighboring nodes as well without going back to the central device, which increases its data processing speed. According to the requirement WSN nodes can be installed or uninstalled. Like all other wireless networks standards, WSN also uses one of those standards. Being a new technology it does not require new Wi-Fi standard. WSNs are very much tolerant to faults, if couple of nodes in a network fails, the communication will always keep on going.

2.1.4 Application

Wireless sensor networks may comprise of numerous different types of sensors like low sampling rate, seismic, magnetic, thermal, visual, infrared, radar, and acoustic, which are clever to monitor a wide range of ambient situations. Sensor nodes are used for constant sensing, event ID, event detection & local control of actuators. The applications of wireless sensor network mainly include health, military, environmental, home, & other commercial areas.

- Military Applications
- Health Applications
- Environmental Applications
- Home Applications
- Commercial Applications
- Area monitoring
- Health care monitoring
- Environmental/Earth sensings
- Air pollution monitoring
- Forest fire detection
- Landslide detection
- Water quality monitoring
- Industrial monitoring

In addition to the above applications, these systems has low power consumption, low cost and is a convenient way to control real-time monitoring for unprotected agriculture and habitat. Moreover, it can also be applied to indoor living monitoring, greenhouse monitoring, climate monitoring and forest monitoring. These approaches have been proved to be an alternative way to replace the conventional method that use men force to monitor the environment and improves the performance, robustness, and provides efficiency in the monitoring system.

2.2 IEEE 802.11s

In this section a detailed explanation of how the IEEE 802.11s works is presented.

2.2.1 Network design

In 802.11, an extended service set (ESS) consists of multiple basic service sets (BSSs) connected through a distributed system (DS) and integrated with wired LANs. The DS service (DSS) is provided by the DS for transporting MAC service data units (MSDUs) between APs, between APs and portals, and between stations within the same BSS 5 that choose to involve DSS. The portal is a logical point for letting MSDUs from a non-802.11 LAN to enter the DS. The ESS appears as single BSS to the logical link control layer at any station associated with one of the BSSs. As is explained in, the 802.11 standard has pointed out the difference between independent basic service set (IBSS) and ESS. IBSS actually has one BSS and does not contain a portal or an integrated wired LANs since no physical DS is available. Thus, an IBSS cannot meet the needs of client support or Internet access, while the ESS architecture can. However, IBSS has its advantage of self-configuration and ad-hoc networking. Thus, it is a good strategy to develop schemes to combine the advantages of ESS and IBSS. The solution being specified by IEEE 802.11s is one of such schemes. In 802.11s, a meshed wireless LAN is formed via ESS mesh networking. In other words, BSSs in the DS do not need to be connected by wired LANs. Instead, they are connected via wireless mesh networking possibly with multiple hops in between. Portals are still needed to interconnect 802.11 wireless LANs and wired LANs.

2.2.2 WSN formation and management

There are four elements that characterize a wireless sensor network:

- Gateway
- Relay Node
- Sink Node
- Sensor

Together these four elements define a profile. If we compare the structure of Mesh Network on IEEE 802.11s with heterogeneous hierarchical wireless sensor

net-work, we can find that they are very similar. Mesh Portal can work as a gateway and provide access to other networks,Mesh Portal is similar to the sink node of wireless sensor network, Mesh Access Point is similar to the cluster head of heterogeneous hierarchical network, and mobile client terminals are similar to the common nodes in wireless sensor network[21]. Most of the nodes of wireless sensor network are powered by batteries, so their node energy is restricted and the calculating capability and storage capacity are limited. So the network protocols of WiFi Wireless Mesh Network can only be used in the new generation of remote AMR network after being optimized. Since the default routing protocol of Wireless Mesh is HWMP, we should reduce the energy consumption of nodes running with HWMP.

2.3 IEEE 802.11s model in NS-3

This section provides an explanation of how is implemented the 802.11s wireless sensor networking model in the Network Simulator 3 (NS-3) and which features or characteristics are supported and which not. First, NS3 is briefly explained to see why this simulator has been chosen. The model used has been the one developed by the Wireless Software R&D Group of IITP RAS and included in NS-3 from the release 3.6. Although it is based in the IEEE P802.11s/D3.0, for the aim of this research, the characteristics used and analyzed are not different from the ones present in the last draft of 802.11s.

2.3.1 Network Simulator 3

NS-3 is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. NS-3 is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use. It is a tool aligned with the simulation needs of modern networking research allowing researchers to study Internet protocols and large-scale systems in a controlled environment. The following trends is how Internet research is being conducted are responded by NS-3:

- **Extensible software core:** written in C++ with optional Python interface and an extensively documented API (doxygen).
- **Attention to realism:** model nodes more like a real computer and support key interfaces such as sockets API and IP/device driver interface (in Linux).
- **Software integration:** conforms to standard input/output formats (pcap trace output, NS-2 mobility scripts, etc.) and adds support for running implementation code.
- **Support for virtualization and testbeds:** Develops two modes of integration with real systems:
  -Virtual machines run on top of ns-3 devices and channels
  -NS-3 stacks run in emulation mode and emit/consume packets over real devices.

- **Flexible tracing and statistics:** decouples trace sources from trace sinks so we have customizable trace sinks.
- **Attribute system:** controls all simulation parameters for static objects, so you can dump and read them all in configuration files.
- **New models:** includes a mix of new and ported models.

To sum up, NS-3 tries to avoid some problems of its predecessor, NS-2, which is still being used by many researchers, but it has some important lacks such as: interoperability and coupling between models, lack of memory management, debugging of split language objects or lack of realism (in the creation of packets for example). Mainly, the new available high fidelity IEEE 802.11 MAC and PHY models together with real world design philosophy and concepts made NS-3 the choice for developing this 802.11s model as well as for carrying out this research.

2.3.2 Model design

To meet these requirements imposed by 802.11s of supporting multiple interfaces (wire- less devices) and also different mesh networking protocol stacks, WS RD Group designed and implemented a runtime configurable multi-interface and multi-protocol Mesh STA architecture.

Supported features

The most important features supported are the implementation of the Peering Management Protocol, the HWMP and the ALM. A part from the functionality described in section2.3, the PMP includes link close heuristics and beacon collision avoidance. HWMP includes proactive and on-demand modes, unicast/broadcast propagation of management traffic and, as an extra functionality not specified yet in the draft, multi-radio extensions. However, for the moment RANN mechanism is implemented but there is no support, so only the PREQ can be used.

Unsupported features

The most important feature not implemented is Mesh Coordinated Channel Access (MCCA). Internetworking using a Mesh Access Point or a Portal is not implemented neither, but this functionality is not needed to evaluate the performance in the creation of mesh networks. As other less relevant features not implemented we can point out the security, power safe mode and although multi-radio operation is supported, no channel assignment protocol is proposed.

2.3.3 Model implementation

The description of which modules are implemented in C++ and how they interconnect with each other is presented in appendix B. The explanation is in a high-level in order to see which modules and classes need to be accessed or created when designing a mesh network, but the low-level code structure is not described. For

more information on each module and a more detailed low level explanation, please check NS-3 documentation under Doxygen [19]. First is explained the way the MAC-layer routing is implemented presenting the most important classes. Then are analyzed the class MeshHelper (used to create a 802.11s network easier) and MeshPointDevice (developed to create mesh point devices). They provide some functions to configure the different parameters of the network and its devices, so the main parameters and the way to configure them is studied.

2.4 Wormhole attack in 802.11b WSN

Due to multihop routing in wireless sensor network, it is prone to various types of attacks. Wormhole attack in an IEEE 802.11bWSN occurs when a mobile STA sends data to the destination beyond the radio range and receives data by another mobile STA in MN. During the data sending and receiving process, management of frames are exchanged between the STA and the MN. Consequently, there is a latency involved in the Wormhole attack process during which the STA is unable to send or receive traffic because of malicious attribute of a node in WSN which prevent service to the legitimate users and drop packets in Wormhole attack.

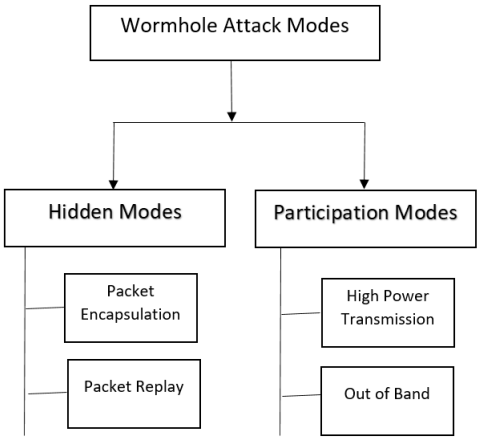


Figure 2.4: Taxonomy of Wormhole attack

2.4.1 Wormhole using Packet Encapsulation

Here several nodes exist between two malicious nodes and data packets are encapsulated between the malicious nodes. Hence it prevents nodes on way from incrementing hop counts. The packet is converted into original form by the second end point. This mode of wormhole attack is not difficult to launch since the two ends of wormhole do not need to have any cryptographic information, or special requirement such as high-power source or high bandwidth channel.

2.4.2 Wormhole using Packet Relay

One or more malicious nodes can launch packet-relay-based wormhole attacks. In this type of attack malicious node replays data packets between two far nodes and this way fake neighbours are created.

2.4.3 Wormhole using Out-of-Band Channel

This kind of wormhole approach has only one malicious node with much high transmission capability in the network that attracts the packets to follow path passing from it. The chances of malicious nodes present in the routes established between sender and receiver increases in this case. Also this type is referred as "black hole attack" in the literature.

2.4.4 Wormhole using High Power Transmission

In this mode of attack, a single ma- licious node can create a wormhole without colluding node. when this single malicious node received a RREQ, it rebroadcasts the request at a very high power level capability compared to normal node, thereby attracting normal nodes to overhear this RREQ and further on broadcast the packet towards destination.

2.5 Related Works

Many Wormhole attack detection schemes have been proposed in the researched literature for WSN networks. An efficient method for detecting wormhole attacks against the routing functionality of network is proposed in [11] in which the author propose an algorithm which is meant to secure each link.The disadvantage of this approach is that each node must be equipped with a second ultrasound radio, allowing the estimation of distances between neighboring nodes. In paper[12], a statistical approach is proposed, known as SWAN, in which each sensor collects a recent number of neighbors. In [6], Hu and Evans propose a solution to wormhole attacks for ad hoc networks in which all nodes are equipped with directional antennas. HU et al [4] describe a defense based on the leashes of the packet, where the distance of a message route is limited, each message having a timestamp and a location of its transmitter. But this proposal presents two disadvantages: It requires a coordinate system such as the GPS in order to obtain the geographic information about each node; It requires a precise synchronization of clocks between different nodes in order to use timely data.

2.6 Chapter Summary

This chapter has discussed what Wormhole attack is and why detection is important to WSNs and also outlines some fundamental aspects of the operation of IEEE 802.11 WLANs and WSNs. The chapter ended with discussion of related works. The following chapters will further outline the technical details of the proposed scheme and its implementation, as well as an analysis of its performance along with the comparing with the existing system.

Chapter 3

Methodology

In this chapter, a detailed description of the proposed Dos attack detection methodology is given. Besides an analysis of the existing Dos attack detection procedure is provided.

3.1 Detection, Prevention and Reactive AODV

In[**?**], they proposed DPRAODV. In this scheme, AODV protocol is modified to have a new control packet called ALARM and a threshold value. A threshold value is the average of the difference of destination sequence number in the routing table and sequence number in the RREP packet.

In the usual operation of AODV, the value of sequence number is checked in the routing table by the node that receives a RREP packet. The sequence number of RREP packet has to be higher than the sequence number value in the routing table in order for RREP to be accepted. In DPRAODV, there is an extra threshold value that is compared to RREP sequence number, and if RREP sequence number is greater than the threshold value, then the sender is considered malicious and added to the black list. The neighbouring nodes are notified using an ALARM packet so that the RREP packet from the malicious node is not processed and gets blocked. Automatically, the threshold value gets updated using the data collected in the time interval. This updating of the threshold value helps to detect and stop black hole attacks.

The ALARM packet contains the black list that has a malicious node, so that the neighbouring nodes reject any RREP packet from a malicious node. Any node that receives a RREP packet checks the black list and if the reply comes from a node that has been blacklisted, it is ignored and further replies from that node will be discarded. Thus the ALARM packet isolates a malicious node from the network.

3.2 Malicious Node Detection

The malicious node detection technique is responsible for detecting the Black Hole nodes in the network. Initially, the Black Hole detector initializes the malicious node detection process. First, it broadcasts the spoofed RREQ packets. As discussed above, the spoofed RREQ packet contains the non existence source id and the TTL value set to 1. Then this spoofed RREQ packet is broadcast to all the other nodes in the network. The broadcasted Honeypot spoofed RREQ packet waits for the reply from the neighbor nodes. If any neighbor replies to this packet, those nodes are marked as Black Hole nodes in the routing table. The reason is, since the normal nodes which are not malicious will not reply to this spoofed RREQ packet. So the routing table updates this Black Hole node information by marking it as malicious.

3.3 Route Lookup in Network Layer

In order to resolve the route, the AODV calls the modified Route Lookup function. This algorithm is very important, because it detects the Black Hole attacks by checking the node id. If the malicious node replies that, it has the route towards the non-existence node, then that vulnerable (Black Hole) node is marked as malicious. In order to find a Black Hole node, a detection flag is set on the routing table. If the detection flag is true then, it is observed that the malicious node id is marked. Thus, routing via the malicious node is avoided. The Route Lookup algorithm for the network layer is responsible for updating the reply from the neighbor nodes. The node which replies to the spoofed RREQ packet is identified as the Black Hole node. Then, the node is marked as malicious in RTF and this information is updated in the routing table. Hence the above route lookup algorithm is responsible which marks the malicious node ids in the routing table.

3.4 Isolation in Network Layer

The isolation technique is responsible for isolating the malicious node from the network. This technique is important, because it prevents broadcasting routes via the malicious node. A flag is set as malicious, and the nodes which reply to the non-existence node id are marked as malicious.

3.5 Intrusion Detection System AODV (IDSAODV)

In[**?**], they proposed IDSAODV in order to decrease the impact of black hole. This is achieved by altering the way normal AODV updates the routing process. The routing update process is modified by adding a procedure to disregard the route that is established first.

The tactic applied in this method is that the network that is attacked has many RREP packets from various paths, so is assumed that the first RREP packet is

generated by a malicious node. The assumption is based on the fact that a black hole node does not look up into its routing table before sending a RREP packet. Therefore, to avoid updating routing table with wrong route entry, the first RREP is ignored.

This method improves packet delivery but it has limitations that; the first RREP can be received from an intermediate node that has an updated route to the destination node, or if RREP message from a malicious node can arrive second at the source node, the method is not able to detect the attack.

3.6 Enhanced AODV (EAODV)

In[**?**],they proposed EAODV. Similar to IDSAODV, EAODV allows numerous RREPs from various paths to lighten the effect of black hole attack. This method makes an assumption that eventually the actual destination node will send a RREP message, so the source node overlooks all previous RREP entries, including the ones from malicious node and takes the latest RREP packet.

The source node keeps on updating its routing table with RREPs being received until a RREP from the actual destination is received. Then all RREPs get analysed and suspicious nodes are detected and isolated from the network. The limitation to this method is that it adds two processes that increase delay and exhaust energy of the nodes.

3.7 Secure AODV (SAODV)

In[**?**], they proposed a secure routing protocol SAODV that addresses black hole attack in AODV. The difference between AODV and SAODV is that in SAODV, there are random numbers that are used to verify the destination node. An extra verification packet is introduced in the route discovery process. After receiving a RREP packet, the source node stores it in the routing table and immediately sends a verification packet using reverse route of received RREP. The verification packet contains a random number generated by source node.

When two or more verification packets from the source node are received at the destination node, coming from different routes, the destination node stores them in its routing table and checks whether the contents contain the same random numbers. If the verification packets contain same random numbers along different paths, the destination node sends verification confirm packet to the source node which contains random number generated by destination node.

If verification confirm packet contains different random numbers, the source node will wait until at least two or more verification confirm packets contain same random numbers. When the source node receives two or more verification confirm packet with same random numbers, it will use the shortest route to send data to the

destination node. The security mechanism in this protocol is that malicious node cannot pretend to be destination node and send correct verification confirm packet to the source node.

3.8 Solution utilizing network redundancy

This is an improvement of the normal AODV. The solution proposes that the source node does not immediately start sending data packets after receiving a route reply. It waits to receive other route replies from nearby nodes to confirm that they contain the same next hop information. This solution uses an assumption that there are redundant routes that can be used to reach the destination node. When a RREP packet arrives at the source node, the full path is extracted and the source node waits for another RREP packet. The routes from other RREP packets are compared with the route extracted from the first RREP, and they must have shared hops. If there are no shared hops in the routes, the source node takes the routes to be untrustworthy and waits for more RREP packets until there are shared hops or until the expiration of the routing timer. Even though this solution assures a safe route, it increases the time delay and messages will never be forwarded to the destination node if there are no shared hops in the paths.

3.9 Proposed Methodology

The proposed methodology is based on calculation on the hop by hop to find out Dos attacks like Blackhole and Greyhole attacks.

In order to detect Dos, attack a malicious node has been created in wireless mesh network which attributes is drop all packets for Blackhole attack and selectively drop some packets for Greyhole attack. The specific value of those thresholds can be calculated based probability of packet dropping for those attacks.

The DoS attack detection algorithm is divided into the several steps as shown in Figure 4:

Header addition: Add a 16-bit header in each packet for source id and destination id. First 8-bit is used for source id and last 8-bit is used for destination id.
Collection: Find the route for sending packet from source to destination in wireless mesh network by OSLR algorithm. Also collect the forwarding and received packet for all node in wireless mesh network.
Calculation: For forwarding node calculate the forwarding packet FP and for receiving node calculate the receiving packet RP. Determination: Determine forwarded and received packet difference. If the difference is zero, then there is no types of attack. So, send the next packet.

Packet analysis: Analyse the packet.If the forwarded and received packet difference is less than threshold value which is for attacker, then send next packet.If the forwarded and received packet difference is greater than threshold value which is

for channel and other losses, then save the node id, from routing table. Then,calculate the packet loss and also calculate the probability of attack. If the probability of attack is greater than the probability of Blackhole attack, then notify that there is Blackhole attack and send message to all mesh nodes and update the attack table. If the probability of attack is greater than the probability of Grayhole attack, then notify that there is Grayhole attack and send message to all mesh nodes and update the attack table.

For Blackhole attack:

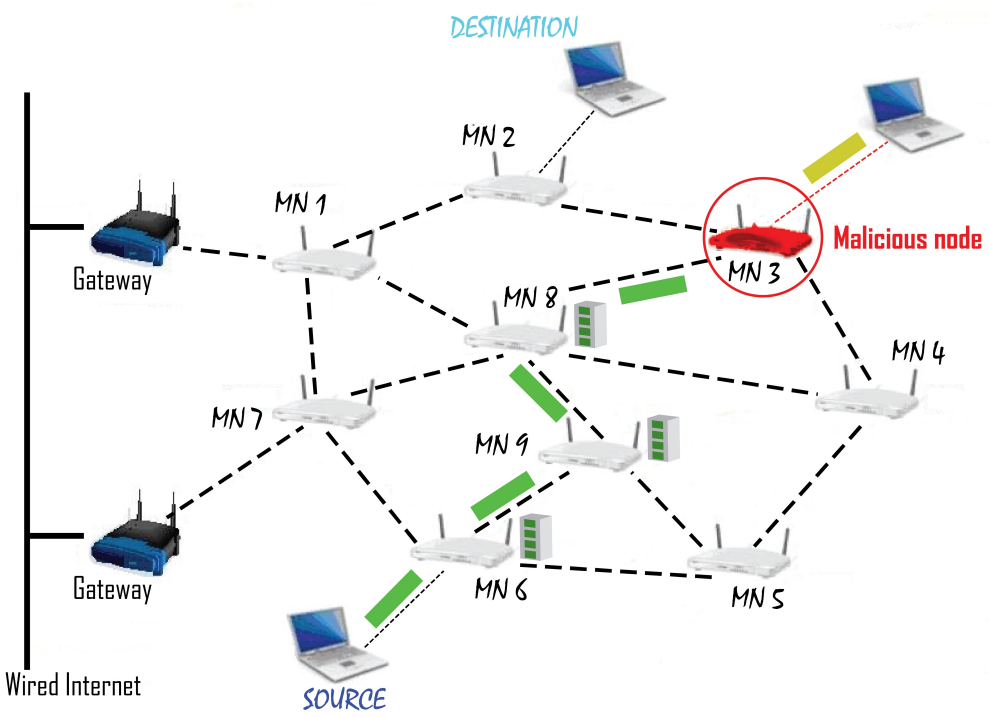


Figure 3.1: Blackhole attack in wireless mesh network

At time threshold 1, the Blackhole attack detection process will be triggered in advance in order to detect the Blackhole attack.

For Greyhole attack: At time threshold 2, the Greyhole attack detection

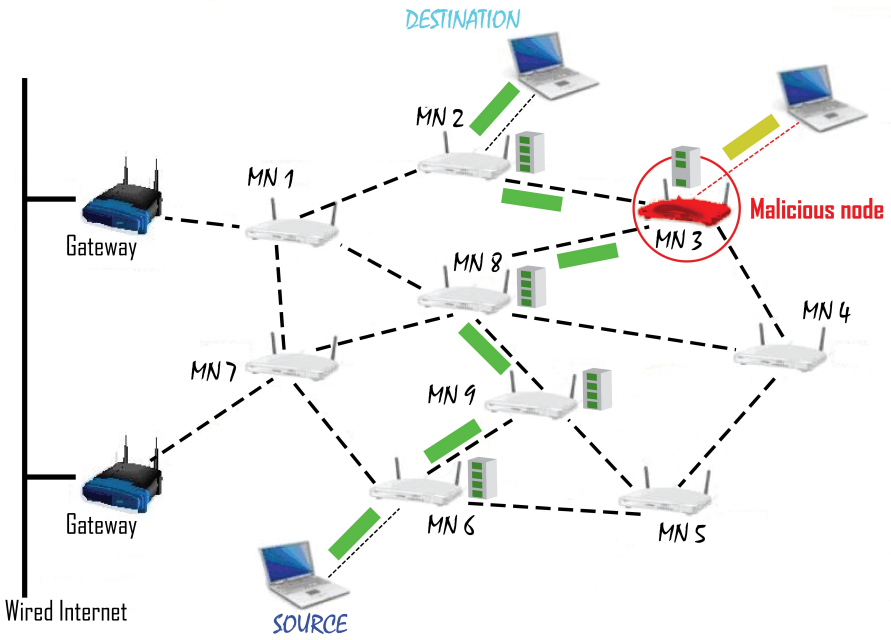


Figure 3.2: Greyhole attack in wireless mesh network

process will be triggered in advance in order to detect the Greyhole attack.

Methodology :

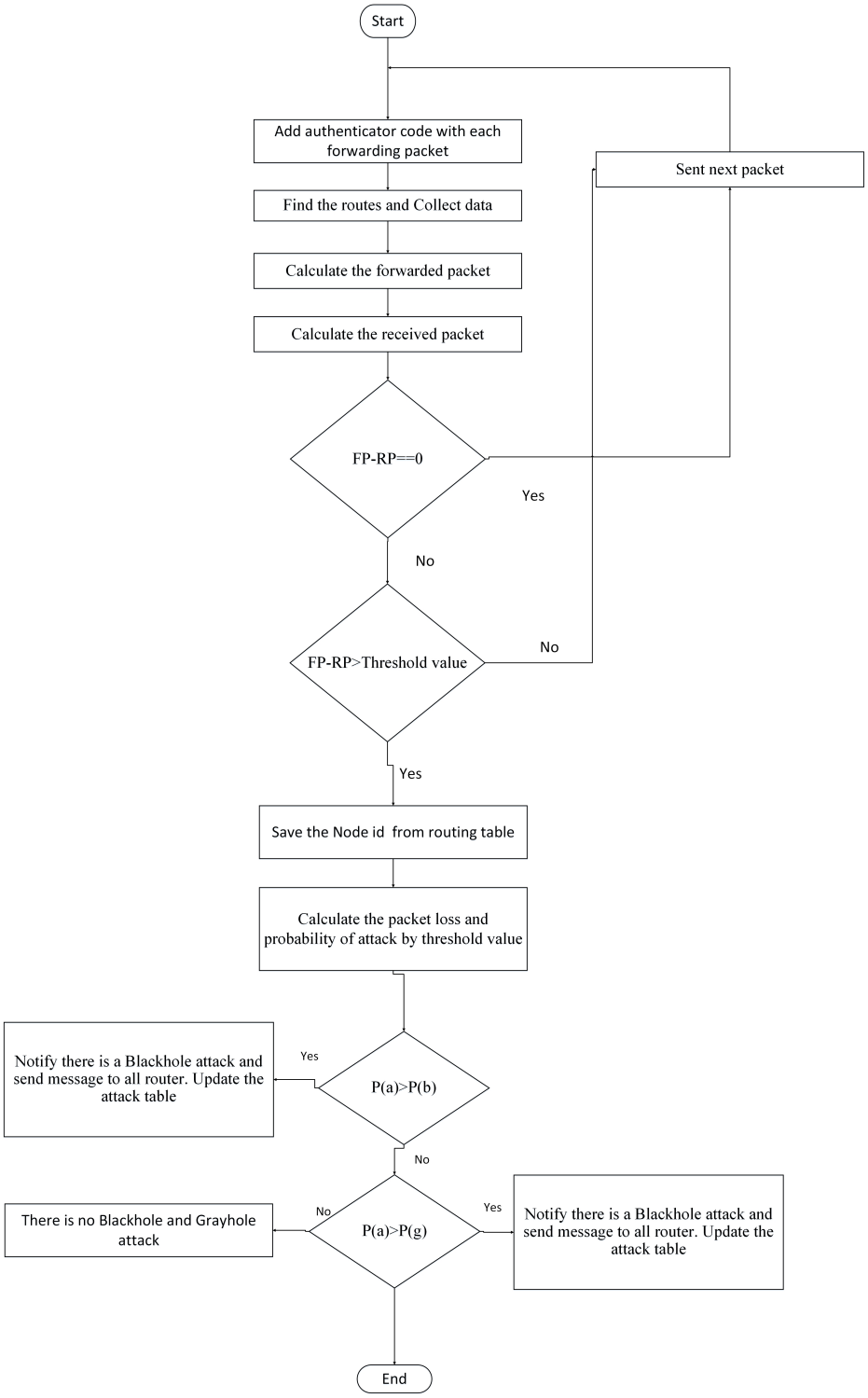


Figure 3.3: Flow chart for DoS attack detection mechanism.

3.10 Chapter Summary

This chapter has outlined the different phases of this study including Dos attack detection analysis and the operation of the proposed scheme. A detailed description of the scheme was given also. The next chapter will outline the implement of the scheme in NS-3

Chapter 4

Implementation

This study evaluated the black hole attack impact on performance of WMN using AODV protocols. It further compared the performance of AODV under black hole attack. The solutions that have been previously proposed to combat effects of black hole attack, which were tested using the base protocol AODV, were studied and the study tries to determine the solution that performs better than others. This is achieved by using network simulator version 3 (NS-3) to simulate Wireless mesh network scenarios that include black hole node. It can be very expensive to carry out a networking research by setting up an actual network with several computers and routers. Network simulators save a lot of money and time in accomplishing network research goals that is why a simulator-based approach has been chosen for this study. The disadvantage of simulation is that some factors have to be estimated because it is not possible to accurately duplicate the whole world inside a computer model. Thus simulation over simplifies real network scenarios. There exists a variety of network simulation tools that are used in research, but NS-3 has been selected for this study because the protocols under study (AODV) have not been implemented in NS-3. Also NS-3 is distributed freely and is an open source environment which allows the creation of new protocols, and modification of existing ones, so it is possible to introduce a black hole attack in NS-3 by modifying its source code. Moreover, NS-3 is well documented and user online support is provided. This chapter explains the implementation of the research study on NS-3 simulation tool stipulating in detail the parameters used in the simulation and outlining the changes made to the NS-3 source code to introduce black hole attack.

4.1 Implementation Tools

The necessary tools to implement this system can be divided in to two categories. Hardware & Software as described below:

- Hardware Requirements
  - Personal Computer with basic configuration
-

Software Tools

- Operating System:Ubuntu 14.04 LTS
- Network Simulator 3 version 3.23
- NetAnim
- PyViz
- Wireshark
- Flow Monitor

4.2 Implementation Details

As discussed in the previous chapter, a Dos attack detection scheme have been developed for reducing packet loss in wireless mesh network. It is implemented in NS-3 in order to analyze its performance through experiments. The basis of the scheme is to decrease the total packet loss by detecting the Blackhole and Greyhole attack.

4.3 Simulation Parameters

- Number of Nodes: 15
- Simulation Time: 80s
- Mobility Model: Constant Position Mobility Model, Random Walk Mobility Model
- Routing Protocol: AODV Routing Protocol
- Size of packets in UDP ping: 1024 bytes
- Packet interval: 0.1 sec
- Data Rate of Wireless links: 250Kbps
- Data Rate of Wireless Mesh links: 500Kbps
- Data Rate of CSMA connection: 100Mbps
- Node distance: 50 meters

4.4 Simulation of black hole attack and greyhole attack

A malicious node is introduced to both AODV to implement a black hole and greyhole by modifying NS-3 C++ source code as shown below. A malicious node attracts the packets and discards them.

4.4.1 AODV Modifications

i. Declared malicious node variable in aodv.h file.
bool malicious;

ii.Initialized the variable to false in aodv.cc constructor function to show that initially all nodes are not malicious.
malicious= false;

iii.In AODV.cc route handling function, the following code was added to maliciously drop packets.

if(IsMalicious)//When malicious node receives packet it drops the packet.
std :: cout ¡¡"Blackhole Attack! in wireless mesh network Packet dropped
. .
return false;

4.5 Simulation Visualization

The network model used in our simulation is shown in Figure 4.1. The mesh backbone size varies when mesh routers are added in the network. The link between Mesh Router and all other connections are wireless.

The figure 4.1 shows the normal behavior of in the wireless mesh network and the figure 4.2 shows the malicious attribute in the wireless mesh network.

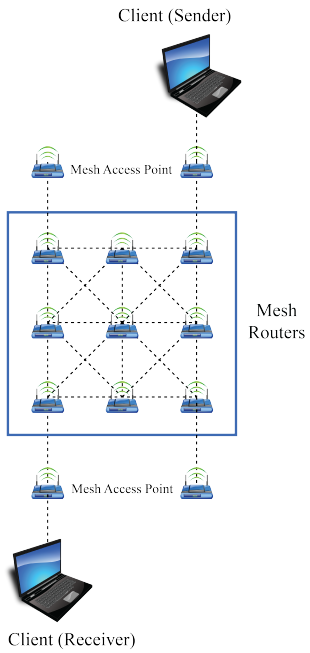


Figure 4.1: Normal Network Model for Simulation

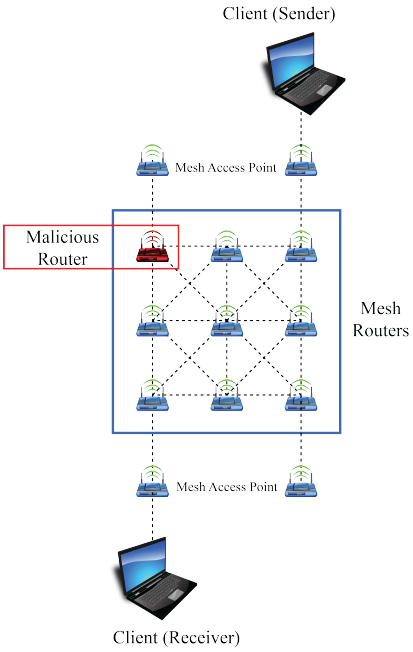


Figure 4.2: Malicious Network Model for Simulation

For the analysis and evaluation ns-3 (network simulator 3) and for the visualization NetAnim and PyViz is used. Wireshark is used to analyze the signaling packet and data packets.

The network model shown above is simulated in PyViz as below:

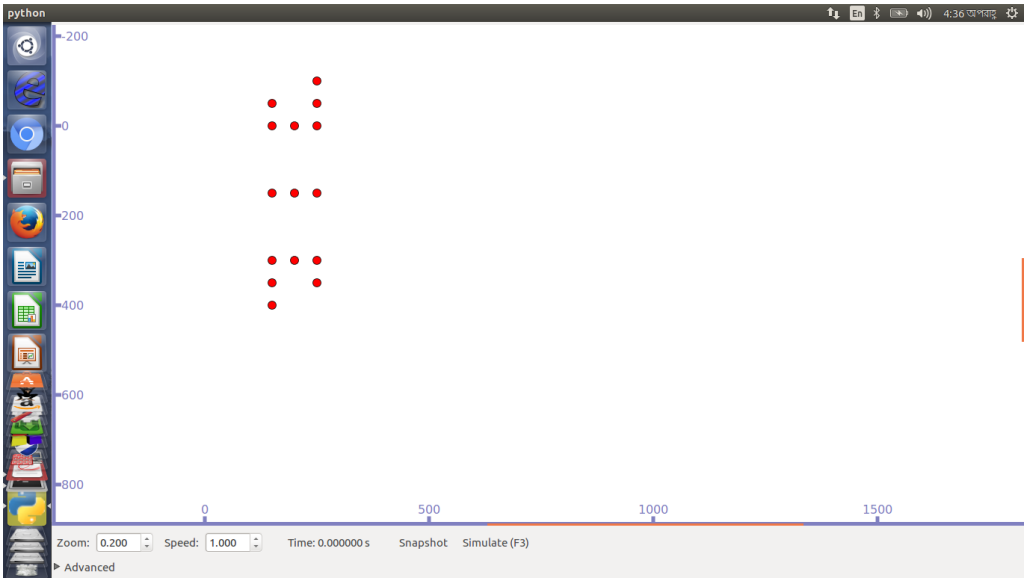


Figure 4.3: Network model

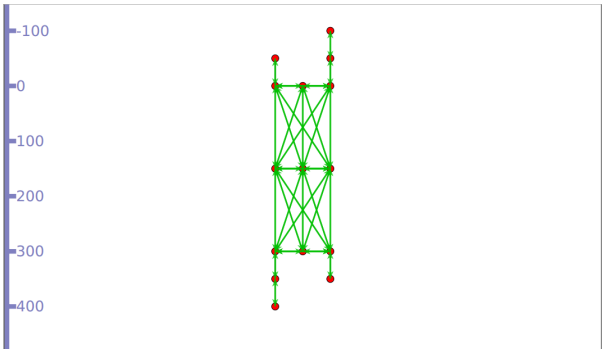


Figure 4.4: Active probing

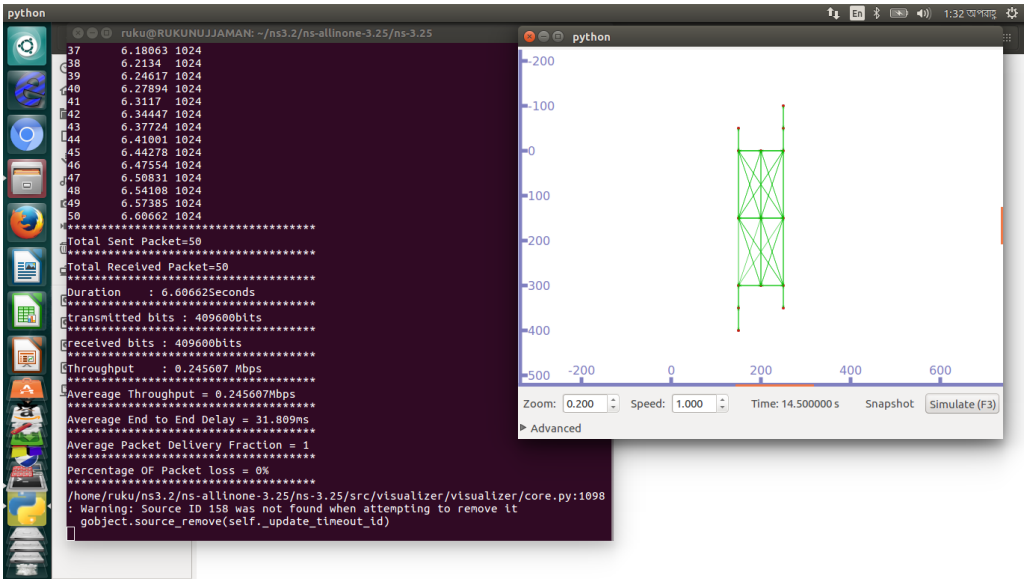


Figure 4.5: Transmission of Data Packets in normal mode

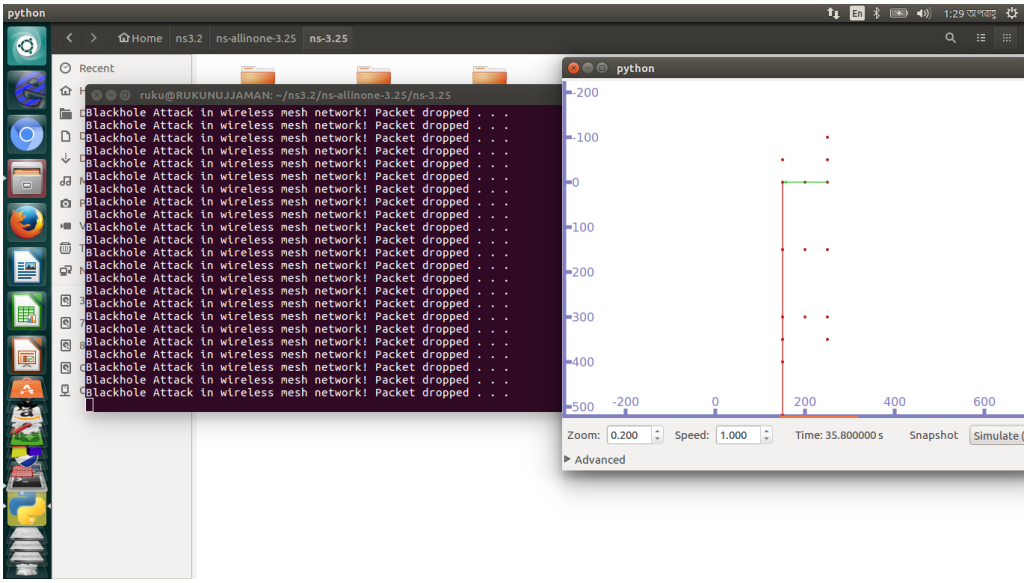


Figure 4.6: Transmission of data packets Under Blackhole attack

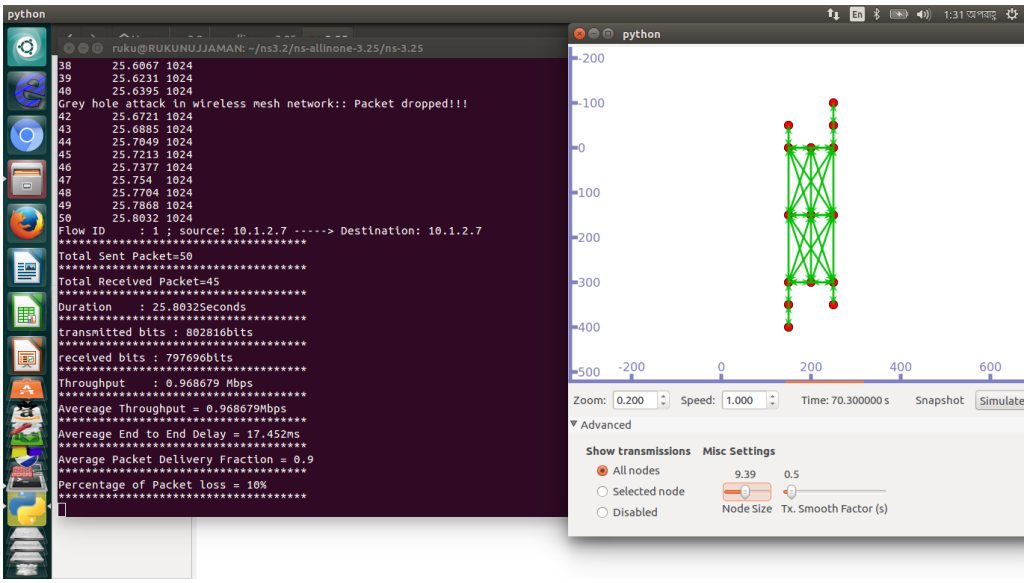


Figure 4.7: Transmission of data packets Under Greyhole attack

4.6 Chapter Summary

In this chapter, the details of the implementation of the scheme is given in NS-3.The next chapter will present the results generated from the simulations.

Chapter 5

Simulation Results and Analysis

In this section the simulation results are presented in detail and an explanation is provided.

5.1 Parameters for Evaluating Simulation Model

The following parameters are needed for evaluating our simulation.

Average Throughput: Number of bits received divided by the difference between the arrival time of the first packet and the last one.

$$Throughput = \frac{Bits\ Received}{timeLastRxPacket - timeFirstTxPacket}$$

Average Packet Delivery Fraction (PDF): Number of packets received divided by the number of packets transmitted.

$$PDF = \frac{No.\ of\ Packets\ Received}{No.\ of\ Packets\ Transmitted}$$

Average end-to-end Delay: The sum of the delay of all received packets divided by the number of received packets.

$$ETE\ Delay = \frac{\sum Delay\ of\ all\ received\ packets}{number\ of\ received\ packets}$$

5.2 Performance of Proposed Method

5.2.1 Average End to End Delay

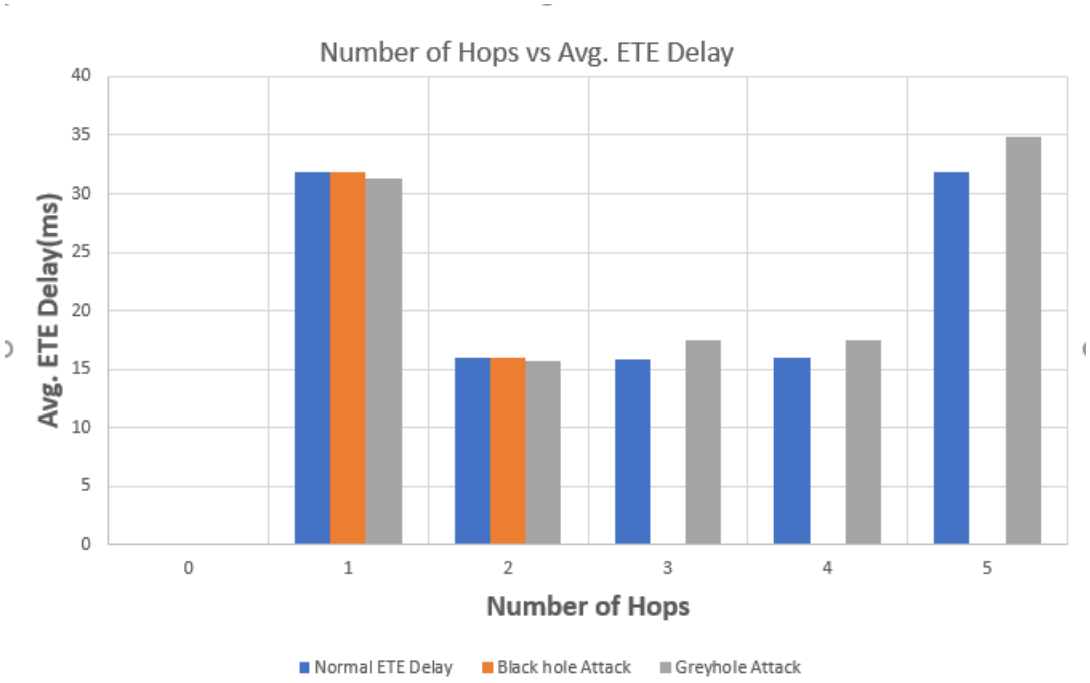


Figure 5.1: Number of Hops vs. Avg. ETE Delay

5.2.2 Average Throughput

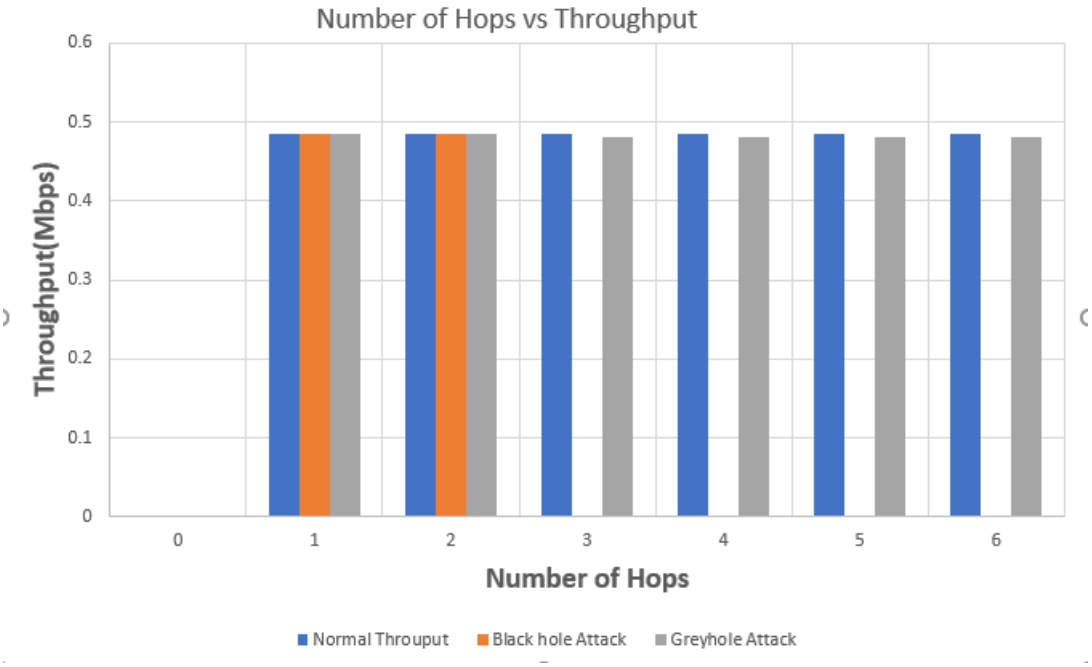


Figure 5.2: Number of Hops vs. Avg. Throughput

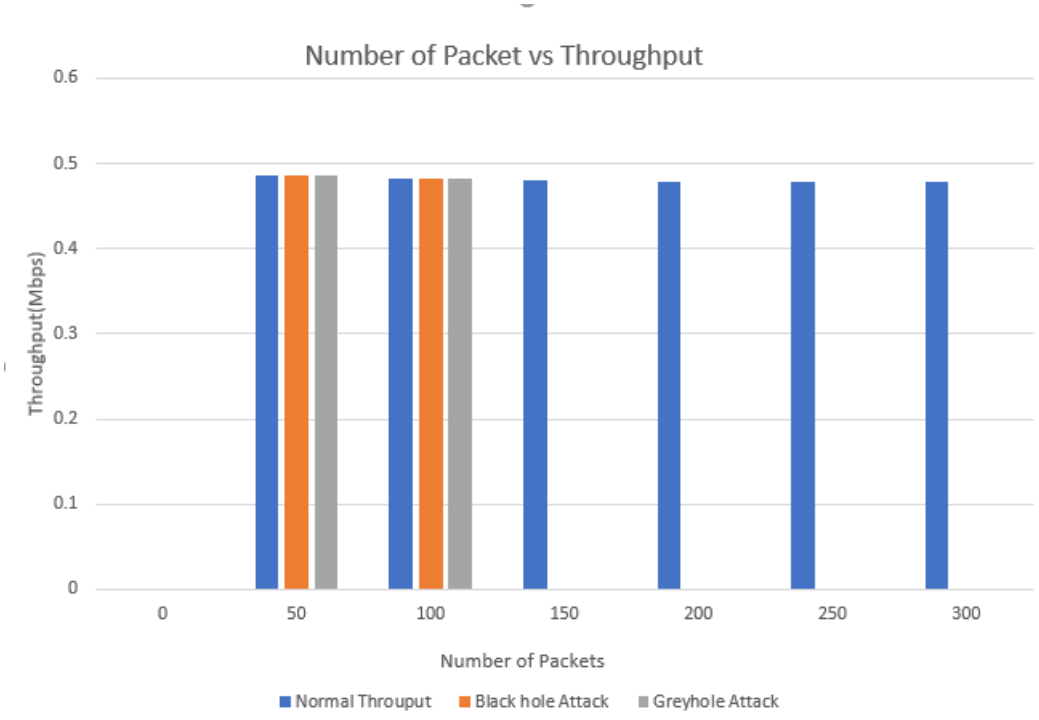


Figure 5.3: Number of MCs vs. Avg. Throughput

5.2.3 Packet delivery fraction(PDF)

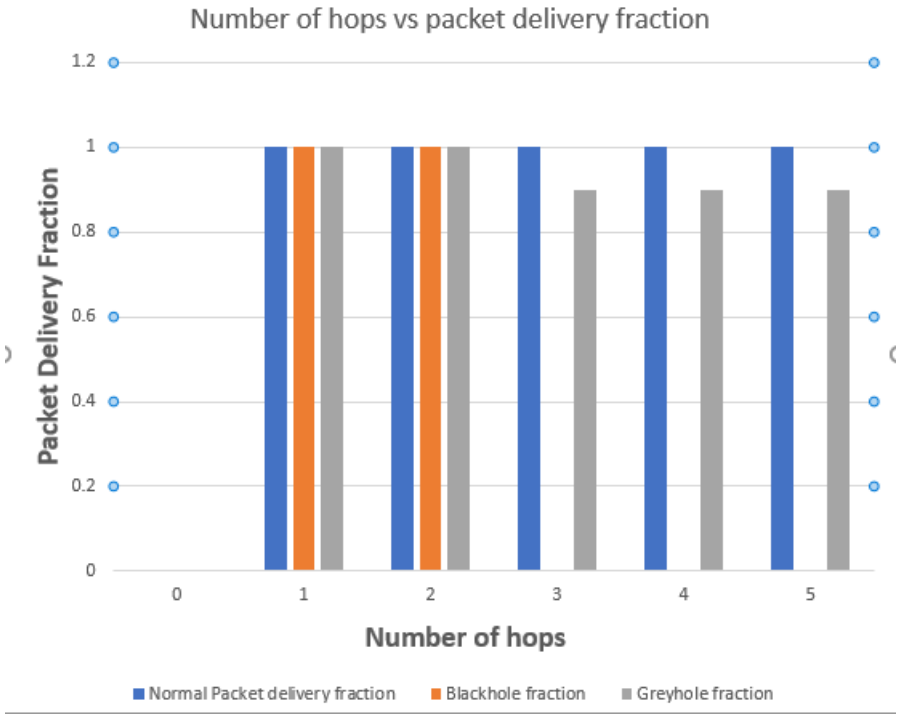


Figure 5.4: Number of Hops vs. PDF

No. of hops	Normal	Blachole attack	Greyhole attack
0	0	0	1
1	1	1	1
2	1	1	0.90
3	1	0	0.90
4	1	0	0.90
5	1	0	0.90

Table 5.1: No. of hops vs Packet delivery fraction

5.3 Comparison with Traditional Dos attack

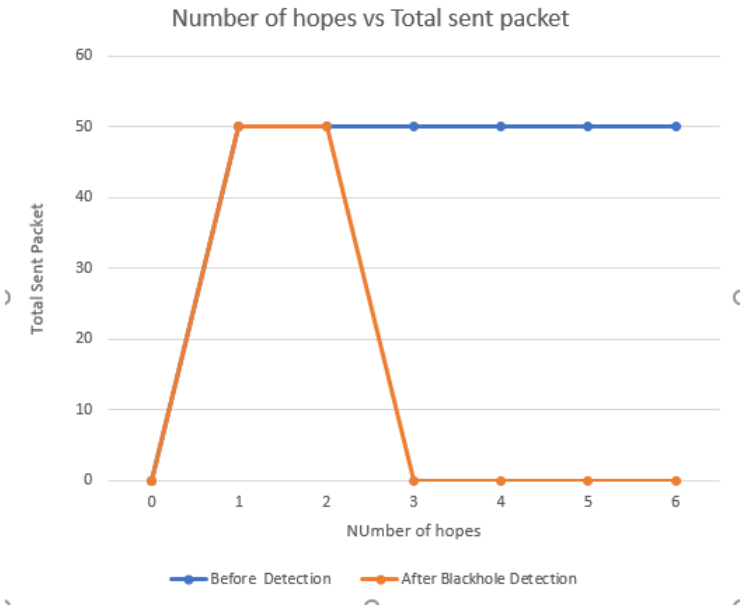


Figure 5.5: Number of hopes vs. Number of Packets

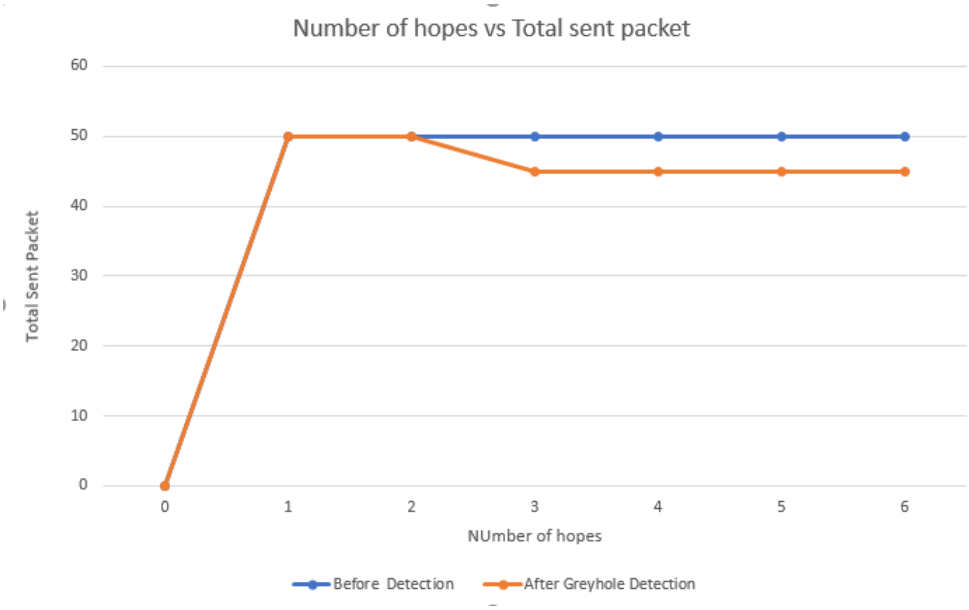


Figure 5.6: Number of hopes vs. Number of Packets

5.3.1 Average End to End Delay

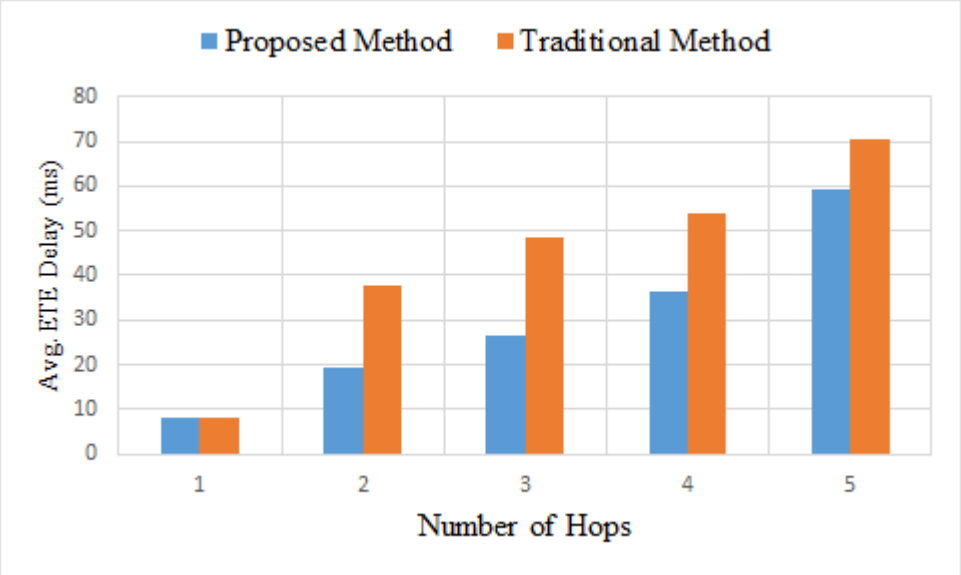


Figure 5.7: Number of Hops vs. Avg. ETE Delay

In terms of average end to end delay, our proposed method performs better than the traditional method with the increasing number of hops.

5.3.2 Average Throughput

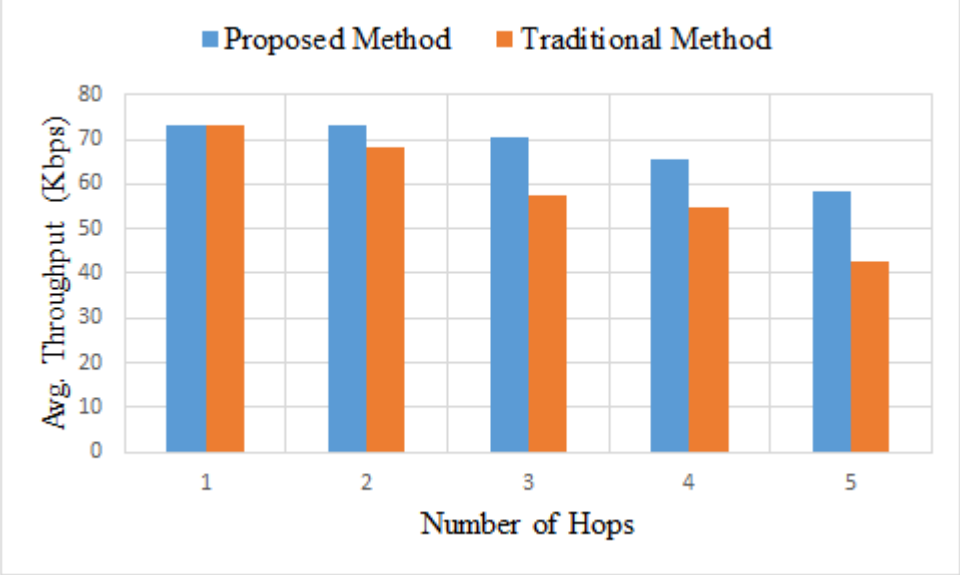


Figure 5.8: Number of Hops vs. Avg. Throughput

From the above figure we can see that our proposed method gives better throughput than traditional method.

5.3.3 Packet delivery fraction(PDF)

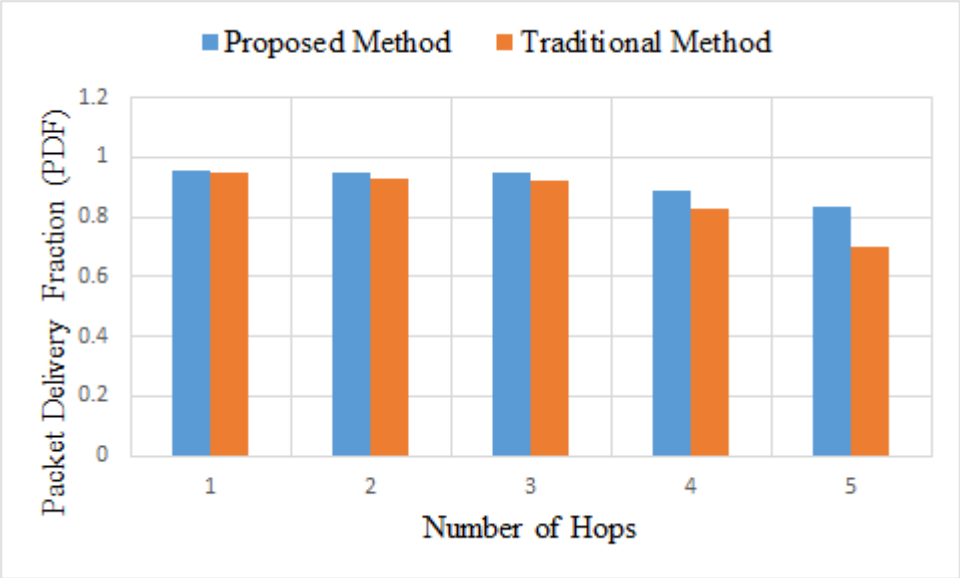


Figure 5.9: Number of Hops vs. PDF

In terms of packet delivery fraction, our proposed method performs better than the traditional method with the increasing number of hops.

From the above information, it is clearly shown that the proposed scheme performs better than traditional Dos attack scheme in terms of packet delivery ratio, end to end delay and throughput.

5.4 Chapter Summary

The objective for the simulation work was to verify the feasibility of the scheme and to compare its latency with the current standard. From the results presented above, the conclusion can be made that this scheme shows the better performance in finding the next AP for STA to associate with when handoff is required compared to other scan techniques.

Chapter 6

Conclusion

This chapter contains an overview of the system and its limitations with future recommendations.

6.1 Findings of the Work

In this thesis, a practical Dos attack management scheme have been developed, to manage the transmitted packet. Theoretically, this scheme can reduce the latency associated with Dos attack in a network. A set of simulation studies were conducted in order to investigate the performance of the scheme in an IEEE 802.11. In the computer simulations, NS-3 was used to implement the theoretical procedures of the scheme and to simulate the scheme under different network scenarios in order to verify the feasibility of the scheme. Over the course of simulation, the effectiveness of our scheme was demonstrated by comparing it to the IEEE 802.11 standard Dos attack and other schemes. The following main observations were made:

- The proposed scheme can reduce the ETE delay by 31.54%.
- It can improve the overall performance by increasing Packet Delivery Fraction by 5.81% and Throughput by 15.06%.

6.2 Future Works

In this work a Dos attack scheme for Infrastructure WMNs has been developed and analyzed. Although this scheme has shown improved Dos attack latency in WMN, further analysis of the scheme under different network conditions could be performed. There are some limitations that should be pointed out which concern the experimental setup.

Appendix A

Source Code

Normal Mode Source Code:

```
#include "ns3/aodv-module.h"
#include "ns3/netanim-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mesh-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/olsr-helper.h"
#include "ns3/point-to-point-module.h"
#include "ns3/csma-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/mobility-module.h"
#include "myapp.h"
#include "ns3/simulator.h"
#include "ns3/nstime.h"
#include "ns3/command-line.h"
#include "ns3/random-variable-stream.h"
#include <iostream>
#include <sstream>
#include <fstream>

NS_LOG_COMPONENT_DEFINE ("Blackhole");

using namespace ns3;


    int        m_xSize=3;
    int        m_ySize=3;
    int        m_sta=1;
    int        m_ap=1;
    uint16_t   m_packetSize=1024;
    uint16_t   m_NumOfPacket=50;

int packetCount,received_bits=0,transmitted_bits=0,endtoenddelay;
float first_transmittedpacket,last_transmittedpacket;
```

```
40 float throughput,localThrou,pdf,sum_of_ete_delay;
41
42 FlowMonitorHelper flowmon;
43 Ptr<FlowMonitor> monitor;
44
45 uint64_t simulationTime = 30; //seconds
46
47 void ReceivePacket(Ptr<const Packet> p, const Address & addr)
48 {
49       packetCount++;
50       std::cout << packetCount << "\t" << Simulator::Now().GetSeconds
     () << "\t" << p->GetSize() <<"\n";
51
52     if(packetCount==1)
53     {
54         first_transmittedpacket=Simulator::Now().GetSeconds();
55     }
56
57     transmitted_bits+=p->GetSize()*8;
58     received_bits+=p->GetSize()*8;
59
60     if(packetCount==m_NumOfPacket)
61     {
62
63         last_transmittedpacket=Simulator::Now().GetSeconds();
64         sum_of_ete_delay=last_transmittedpacket -
    first_transmittedpacket;
65         //std::cout << "Flow ID      : " << 1 << " ; " <<"source: "<<
    Address.addr << " ------> " <<"Destination: "<<Address.addr<< std::
    endl;
66         std::cout<<"***********************************"<<std::endl;
67         std::cout<<"Total Sent Packet="<<m_NumOfPacket<<std::endl;
68         std::cout<<"***********************************"<<std::endl;
69         std::cout<<"Total Received Packet="<<(packetCount)<<std::endl;
70         std::cout<<"***********************************"<<std::endl;
71         std::cout << "Duration     : " <<Simulator::Now().GetSeconds
    ()<< "Seconds" << std::endl;
72         std::cout<<"***********************************"<<std::endl;
73         std::cout << "transmitted bits : " <<transmitted_bits<< "bits"
     << std::endl;
74         std::cout<<"***********************************"<<std::endl;
75         std::cout << "received bits : " <<(received_bits)<< "bits" <<
    std::endl;
76         std::cout<<"***********************************"<<std::endl;
77         std::cout << "Throughput     : " << (received_bits)/(Simulator
    ::Now().GetSeconds() - first_transmittedpacket)/1024/1024 << "
    Mbps" <<std::endl;
78         localThrou = (received_bits)/(Simulator::Now().GetSeconds()
    - first_transmittedpacket)/1024/1024;
79         pdf = (double)(packetCount)/m_NumOfPacket;
80         std::cout<<"***********************************"<<std::endl;
81         std::cout<<"Avereage Throughput = "<<localThrou<<"Mbps"<<std::
    endl;
82         std::cout<<"***********************************"<<std::endl;
83         std::cout<<"Avereage End to End Delay = "<<sum_of_ete_delay
    *1000/(packetCount)<<"ms"<<std::endl;
```

```
84          std::cout<<"**********************************"<<std::endl;
85          std::cout<<"Average Packet Delivery Fraction = "<<pdf<<""<<std
    ::endl;
86          std::cout<<"**********************************"<<std::endl;
87          std::cout<<"Percentage OF Packet loss = "<<(1-pdf)*100<<"%"<<
    std::endl;
88          std::cout<<"**********************************"<<std::endl;
89          packetCount=0;
90          transmitted_bits=0;
91          received_bits=0;
92          sum_of_ete_delay=0;
93      }
94
95  }
96
97
98  int main (int argc, char *argv[])
99  {
100   Time::SetResolution (Time::NS);
101   LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
102   LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
103
104   bool enableFlowMonitor = false;
105   std::string phyMode ("DsssRate1Mbps");
106
107   CommandLine cmd;
108   cmd.AddValue ("EnableMonitor", "Enable Flow Monitor",
      enableFlowMonitor);
109   cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
110   cmd.Parse (argc, argv);
111     ApplicationContainer clientApps;
112
113
114     NS_LOG_INFO ("Create nodes.");
115     NodeContainer nc_meshNodes,nc_sta1, nc_sta2,nc_ap11, nc_ap12,
      nc_ap13, nc_ap14,nc_ap11Mbb1, nc_ap12Mbb1,nc_ap13Mbb1,
116                   nc_ap14Mbb1,not_malicious,malicious; // ALL Nodes
117
118     NetDeviceContainer meshDevices,staApDevices1,staApDevices2,
      staDevices1,staDevices2, apDevices1,apDevices2,ap11Mbb1Devices,
      ap12Mbb1Devices, ap13Mbb1Devices,ap14Mbb1Devices;
119     Ipv4InterfaceContainer interfaces,staAp_interfaces1,
      staAp_interfaces2,staAp_interfaces3,staAp_interfaces4,
      csmaInterfaces1,
120                            csmaInterfaces2,csmaInterfaces3,
      csmaInterfaces4;
121
122     CsmaHelper csma1, csma2, csma3, csma4;
123
124
125 //**********************************station device creation
    ****************************************
126     nc_sta1.Create(m_sta);
127     nc_sta2.Create(m_sta);
128
```

```
129 //**********************************mesh device creation
    ****************************************
130     nc_meshNodes.Create(m_xSize * m_ySize);
131
132     not_malicious.Add(nc_meshNodes.Get(1));
133     not_malicious.Add(nc_meshNodes.Get(2));
134     not_malicious.Add(nc_meshNodes.Get(3));
135     not_malicious.Add(nc_meshNodes.Get(4));
136     not_malicious.Add(nc_meshNodes.Get(5));
137     not_malicious.Add(nc_meshNodes.Get(6));
138     not_malicious.Add(nc_meshNodes.Get(7));
139     not_malicious.Add(nc_meshNodes.Get(8));
140     malicious.Add(nc_meshNodes.Get(0));
141
142 //**********************************Accesspoint device creation
    ****************************************
143     nc_ap11.Create(m_ap);
144     nc_ap12.Create(m_ap);
145     nc_ap13.Create(m_ap);
146     nc_ap14.Create(m_ap);
147
148     nc_ap11Mbb1=NodeContainer(nc_ap11, nc_meshNodes.Get(6));
149     nc_ap12Mbb1=NodeContainer(nc_ap12, nc_meshNodes.Get(0));
150     nc_ap13Mbb1=NodeContainer(nc_ap13, nc_meshNodes.Get(8));
151     nc_ap14Mbb1=NodeContainer(nc_ap14, nc_meshNodes.Get(3));
152
153     // Set up mesh
154     WifiHelper wifi;
155
156     YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
157     wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11);
158
159     YansWifiChannelHelper wifiChannel ;
160     wifiChannel.SetPropagationDelay ("ns3::
    ConstantSpeedPropagationDelayModel");
161     wifiChannel.AddPropagationLoss ("ns3::
    TwoRayGroundPropagationLossModel",
162                                     "SystemLoss", DoubleValue(1),
163                                     "HeightAboveZ", DoubleValue(1.5));
164
165     wifiPhy.Set ("TxPowerStart", DoubleValue(33));
166     wifiPhy.Set ("TxPowerEnd", DoubleValue(33));
167     wifiPhy.Set ("TxPowerLevels", UintegerValue(1));
168     wifiPhy.Set ("TxGain", DoubleValue(0));
169     wifiPhy.Set ("RxGain", DoubleValue(0));
170     wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue(-61.8));
171     wifiPhy.Set ("CcaMode1Threshold", DoubleValue(-64.8));
172
173     wifiPhy.SetChannel (wifiChannel.Create ());
174
175     // Add a non-QoS upper mac
176     NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
177
178     NetDeviceContainer devices;
179     meshDevices = wifi.Install (wifiPhy, wifiMac, nc_meshNodes);
180
```

```
181
182 // ******************************Setup WiFi for network
    1**********************************
183     WifiHelper wifi1 = WifiHelper::Default();
184     wifi1.SetStandard (WIFI_PHY_STANDARD_80211b);
185     wifi1.SetRemoteStationManager("ns3::AarfWifiManager");
186
187     NqosWifiMacHelper mac1 = NqosWifiMacHelper::Default();
188
189     Ssid ssid1 = Ssid("network-1");
190     mac1.SetType("ns3::StaWifiMac",
191                  "Ssid", SsidValue(ssid1),
192                  "ActiveProbing", BooleanValue(true));
193
194     staDevices1 = wifi1.Install(wifiPhy, mac1, nc_sta1.Get(0));
195
196
197     // Setup AP for network 1
198     mac1.SetType("ns3::ApWifiMac",
199                  "Ssid", SsidValue(ssid1));
200
201     apDevices1.Add(wifi1.Install(wifiPhy, mac1, nc_ap11.Get(0)));
202
203     staApDevices1.Add(staDevices1);
204     staApDevices1.Add(apDevices1);
205     //*****************************************End wifi
    1*******************************************
206
207
208     // ***********************************Setup WiFi for network
    2**********************************
209     WifiHelper wifi2 = WifiHelper::Default();
210     wifi2.SetStandard (WIFI_PHY_STANDARD_80211b);
211     wifi2.SetRemoteStationManager("ns3::AarfWifiManager");
212
213
214     NqosWifiMacHelper mac2 = NqosWifiMacHelper::Default();
215
216     Ssid ssid2 = Ssid("network-3");
217     mac2.SetType("ns3::StaWifiMac",
218                  "Ssid", SsidValue(ssid2),
219                  "ActiveProbing", BooleanValue(true));
220
221     staDevices2 = wifi2.Install(wifiPhy, mac2, nc_sta2);
222
223
224     // Setup AP for network 1
225     mac2.SetType("ns3::ApWifiMac",
226                  "Ssid", SsidValue(ssid2));
227
228     apDevices2.Add(wifi2.Install(wifiPhy, mac2, nc_ap14.Get(0)));
229
230     staApDevices2.Add(staDevices2);
231     staApDevices2.Add(apDevices2);
232
```

```
233     //*****************************************End wifi
        2*****************************************
234
235
236 //*********************************Property for csma1
        *****************************************
237     csma1.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
238     csma1.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
        );
239     ap11Mbb1Devices = csma1.Install (nc_ap11Mbb1);
240
241 //*********************************Property for csma2
        *****************************************
242     csma2.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
243     csma2.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
        );
244     ap12Mbb1Devices = csma2.Install (nc_ap12Mbb1);
245
246 //*********************************Property for csma3
        *****************************************
247     csma3.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
248     csma3.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
        );
249     ap13Mbb1Devices = csma3.Install (nc_ap13Mbb1);
250
251 //*********************************Property for csma4
        *****************************************
252     csma4.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
253     csma4.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
        );
254     ap14Mbb1Devices = csma4.Install (nc_ap14Mbb1);
255
256
257
258 //*********************************mobility for mesh
        *****************************************
259
260     MobilityHelper mobility;
261     Ptr<ListPositionAllocator> positionAlloc = CreateObject <
        ListPositionAllocator>();
262     positionAlloc ->Add(Vector(150, 0, 0)); // node0
263     positionAlloc ->Add(Vector(200, 0, 0)); // node1
264     positionAlloc ->Add(Vector(200, 300, 0)); // node2
265     positionAlloc ->Add(Vector(150,300, 0)); // node3
266     positionAlloc ->Add(Vector(150, 150, 0)); // node4
267     positionAlloc ->Add(Vector(200, 150, 0)); // node5
268     positionAlloc ->Add(Vector(250, 0, 0)); // node6
269     positionAlloc ->Add(Vector(250,150, 0)); // node7
270     positionAlloc ->Add(Vector(250, 300, 0)); // node8
271     mobility.SetPositionAllocator(positionAlloc);
272     mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
273     mobility.Install(nc_meshNodes);
274
275 //*********************************mesh accesspoint 1 mobility
        *****************************************
276     MobilityHelper mobility1;
```

```
277     Ptr<ListPositionAllocator> positionAlloc1 = CreateObject <
        ListPositionAllocator >();
278     positionAlloc1 ->Add(Vector(250, -50, 0)); // ap11
279     mobility1.SetPositionAllocator(positionAlloc1);
280     mobility1.SetMobilityModel("ns3::ConstantPositionMobilityModel");
281     mobility1.Install(nc_ap11);
282
283 //**********************************mesh accesspoint 2 mobility
        ****************************************
284     MobilityHelper mobility2;
285     Ptr<ListPositionAllocator> positionAlloc2 = CreateObject <
        ListPositionAllocator >();
286     positionAlloc2 ->Add(Vector(150, -50, 0)); // ap12
287     mobility2.SetPositionAllocator(positionAlloc2);
288     mobility2.SetMobilityModel("ns3::ConstantPositionMobilityModel");
289     mobility2.Install(nc_ap12);
290
291 //**********************************mesh accesspoint 3 mobility
        ****************************************
292     MobilityHelper mobility3;
293     Ptr<ListPositionAllocator> positionAlloc3 = CreateObject <
        ListPositionAllocator >();
294     positionAlloc3 ->Add(Vector(250, 350, 0)); // ap13
295     mobility3.SetPositionAllocator(positionAlloc3);
296     mobility3.SetMobilityModel("ns3::ConstantPositionMobilityModel");
297     mobility3.Install(nc_ap13);
298
299 //**********************************mesh accesspoint 4 mobility
        ****************************************
300     MobilityHelper mobility4;
301     Ptr<ListPositionAllocator> positionAlloc4 = CreateObject <
        ListPositionAllocator >();
302     positionAlloc4 ->Add(Vector(150, 350, 0)); // ap13
303     mobility4.SetPositionAllocator(positionAlloc4);
304     mobility4.SetMobilityModel("ns3::ConstantPositionMobilityModel");
305     mobility4.Install(nc_ap14);
306
307 //**********************************station node 1 mobility
        ****************************************
308     MobilityHelper mobility5;
309     Ptr<ListPositionAllocator> positionAlloc5 = CreateObject <
        ListPositionAllocator >();
310     positionAlloc5 ->Add(Vector(250, -100, 0)); // ap13
311     mobility5.SetPositionAllocator(positionAlloc5);
312     mobility5.SetMobilityModel("ns3::ConstantPositionMobilityModel");
313     mobility5.Install(nc_sta1);
314
315 //**********************************station node 2 mobility
        ****************************************
316     MobilityHelper mobility6;
317     Ptr<ListPositionAllocator> positionAlloc6 = CreateObject <
        ListPositionAllocator >();
318     positionAlloc6 ->Add(Vector(150, 400, 0)); // ap13
319     mobility6.SetPositionAllocator(positionAlloc6);
320     mobility6.SetMobilityModel("ns3::ConstantPositionMobilityModel");
321     mobility6.Install(nc_sta2);
```

```
322
323 //  Enable AODV
324     AodvHelper aodv;
325
326     // Set up internet stack
327     InternetStackHelper internetStack;
328     internetStack.SetRoutingHelper (aodv);
329     internetStack.Install (not_malicious);
330     internetStack.Install (malicious);
331     internetStack.Install (nc_sta1);
332     internetStack.Install (nc_sta2);
333     internetStack.Install (nc_ap11);
334     internetStack.Install (nc_ap12);
335     internetStack.Install (nc_ap13);
336     internetStack.Install (nc_ap14);
337
338
339 //**********************************Setup IP Address
    ******************************************
340     Ipv4AddressHelper address;
341     NS_LOG_INFO ("Assign IP Addresses.");
342     address.SetBase ("10.1.2.0", "255.255.255.0");
343     interfaces = address.Assign (meshDevices);
344
345
346     address.SetBase ("10.1.4.0", "255.255.255.0");
347     csmaInterfaces1 = address.Assign (ap11Mbb1Devices);
348
349     address.SetBase ("10.1.5.0", "255.255.255.0");
350     csmaInterfaces2 = address.Assign (ap12Mbb1Devices);
351
352     address.SetBase ("20.1.4.0", "255.255.255.0");
353     csmaInterfaces3 = address.Assign (ap13Mbb1Devices);
354
355     address.SetBase ("20.1.5.0", "255.255.255.0");
356     csmaInterfaces4 = address.Assign (ap14Mbb1Devices);
357
358     address.SetBase ("10.1.3.0", "255.255.255.0");
359     staAp_interfaces1 = address.Assign (staApDevices1);
360
361     address.SetBase ("20.1.3.0", "255.255.255.0");
362     staAp_interfaces2 = address.Assign (staApDevices2);
363
364
365
366     NS_LOG_INFO ("Create Applications.");
367
368 //*****************************************client to nc_ap11
    ******************************************
369     uint16_t sinkPort3 = 7;
370     Address sinkAddress3 (InetSocketAddress (staAp_interfaces1.
    GetAddress (1), sinkPort3)); // interface of n3
371     PacketSinkHelper packetSinkHelper3 ("ns3::UdpSocketFactory",
    InetSocketAddress (Ipv4Address::GetAny (), sinkPort3));
372     ApplicationContainer sinkApps3 = packetSinkHelper3.Install (
    nc_ap11.Get (0)); //n3 as sink
```

```
373     sinkApps3.Start (Seconds (0.));
374     sinkApps3.Stop (Seconds (10.));
375
376     Ptr<Socket> ns3UdpSocket3 = Socket::CreateSocket (nc_sta1.Get (0),
        UdpSocketFactory::GetTypeId ()); //source at n1
377
378     Ptr<MyApp> app3 = CreateObject<MyApp> ();
379     app3->Setup (ns3UdpSocket3, sinkAddress3, 1024, m_NumOfPacket,
      DataRate ("250Kbps"));
380     nc_sta1.Get (0)->AddApplication (app3);
381     app3->SetStartTime (Seconds (5.));
382     app3->SetStopTime (Seconds (10.));
383
384 //*****************************************nc_ap11 to  mesh node
      6*****************************************
385     uint16_t sinkPort = 6;
386     Address sinkAddress (InetSocketAddress (csmaInterfaces1.GetAddress
        (1), sinkPort)); // interface of n3
387     PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort));
388     ApplicationContainer sinkApps = packetSinkHelper.Install (
      nc_meshNodes.Get (6)); //n3 as sink
389     sinkApps.Start (Seconds (10.));
390     sinkApps.Stop (Seconds (20.));
391
392     Ptr<Socket> ns3UdpSocket = Socket::CreateSocket (nc_ap11.Get (0),
      UdpSocketFactory::GetTypeId ()); //source at n1
393
394     Ptr<MyApp> app = CreateObject<MyApp> ();
395     app->Setup (ns3UdpSocket, sinkAddress, m_packetSize, m_NumOfPacket
      , DataRate ("500Kbps"));
396     nc_ap11.Get (0)->AddApplication (app);
397     app->SetStartTime (Seconds (15.));
398     app->SetStopTime (Seconds (20.));
399
400 //***************************************** mesh node 6 to mesh node
      3*****************************************
401     uint16_t sinkPort1 = 5;
402     Address sinkAddress1 (InetSocketAddress (interfaces.GetAddress (5)
      , sinkPort1)); // interface of n3
403     PacketSinkHelper packetSinkHelper1 ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort1));
404     ApplicationContainer sinkApps1 = packetSinkHelper1.Install (
      nc_meshNodes.Get (5)); //n3 as sink
405     sinkApps1.Start (Seconds (20.));
406     sinkApps1.Stop (Seconds (30.));
407
408     Ptr<Socket> ns3UdpSocket1 = Socket::CreateSocket (nc_meshNodes.Get
        (6), UdpSocketFactory::GetTypeId ()); //source at n1
409
410     Ptr<MyApp> app1 = CreateObject<MyApp> ();
411     app1->Setup (ns3UdpSocket1, sinkAddress1, m_packetSize,
      m_NumOfPacket, DataRate ("500Kbps"));
412     nc_meshNodes.Get (6)->AddApplication (app1);
413     app1->SetStartTime (Seconds (25.));
414     app1->SetStopTime (Seconds (30.));
```

```
415 //****************************************** mesh node 6 to mesh node
      3*************************************
416     uint16_t sinkPort9 = 3;
417     Address sinkAddress9 (InetSocketAddress (interfaces.GetAddress (3)
      , sinkPort9)); // interface of n3
418     PacketSinkHelper packetSinkHelper9 ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort9));
419     ApplicationContainer sinkApps9 = packetSinkHelper9.Install (
      nc_meshNodes.Get (3)); //n3 as sink
420     sinkApps9.Start (Seconds (30.));
421     sinkApps9.Stop (Seconds (40.));
422
423     Ptr<Socket> ns3UdpSocket9 = Socket::CreateSocket (nc_meshNodes.Get
      (5), UdpSocketFactory::GetTypeId ()); //source at n1
424
425     Ptr<MyApp> app9 = CreateObject<MyApp> ();
426     app9->Setup (ns3UdpSocket9, sinkAddress9, m_packetSize,
      m_NumOfPacket, DataRate ("500Kbps"));
427     nc_meshNodes.Get (5)->AddApplication (app9);
428     app9->SetStartTime (Seconds (35.));
429     app9->SetStopTime (Seconds (40.));
430 //****************************************** mesh node 3 to ap13
      ****************************************
431     uint16_t sinkPort2 = 4;
432     Address sinkAddress2 (InetSocketAddress (csmaInterfaces4.
      GetAddress (1), sinkPort2)); // interface of n3
433     PacketSinkHelper packetSinkHelper2 ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort2));
434     ApplicationContainer sinkApps2 = packetSinkHelper2.Install (
      nc_meshNodes.Get (3)); //n3 as sink
435     sinkApps2.Start (Seconds (40.0));
436     sinkApps2.Stop (Seconds (50.));
437
438     Ptr<Socket> ns3UdpSocket2 = Socket::CreateSocket (nc_ap14.Get (0),
      UdpSocketFactory::GetTypeId ()); //source at n1
439
440     Ptr<MyApp> app2 = CreateObject<MyApp> ();
441     app2->Setup (ns3UdpSocket2, sinkAddress2, m_packetSize,
      m_NumOfPacket, DataRate ("500Kbps"));
442     nc_ap14.Get (0)->AddApplication (app2);
443     app2->SetStartTime (Seconds (45.));
444     app2->SetStopTime (Seconds (50.));
445
446 //****************************************nc_ap13 to sta2
      ****************************************
447     uint16_t sinkPort4 = 2;
448     Address sinkAddress4 (InetSocketAddress (staAp_interfaces2.
      GetAddress (1), sinkPort4)); // interface of n3
449     PacketSinkHelper packetSinkHelper4("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort4));
450     ApplicationContainer sinkApps4 = packetSinkHelper4.Install (
      nc_ap14.Get (0)); //n3 as sink
451     sinkApps4.Start (Seconds (50));
452     sinkApps4.Stop (Seconds (60.));
453
```

```
454     Ptr<Socket> ns3UdpSocket4 = Socket::CreateSocket (nc_sta2.Get (0),
        UdpSocketFactory::GetTypeId ()); //source at n1
455
456     Ptr<MyApp> app4 = CreateObject<MyApp> ();
457     app4->Setup (ns3UdpSocket4, sinkAddress4, m_packetSize,
        m_NumOfPacket, DataRate ("250Kbps"));
458     nc_sta2.Get (0)->AddApplication (app4);
459     app4->SetStartTime (Seconds (55.));
460     app4->SetStopTime (Seconds (60.));
461
462 //*****************************************flow monitor
        ******************************************
463     monitor = flowmon.InstallAll();
464
465 //*****************************************Trace Received Packet
        ******************************************
466     Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3
        ::PacketSink/Rx", MakeCallback (&ReceivePacket));
467     //Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::
        WifiNetDevice/Mac/MacTxDrop", MakeCallback(&MacTxDrop));
468     //Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::
        WifiNetDevice/Mac/MacRxDrop", MakeCallback(&MacRxDrop));
469     monitor->CheckForLostPackets ();
470
471     Simulator::Stop (Seconds(simulationTime+1));
472     Simulator::Run ();
473
474     Simulator::Destroy ();
475
476     return 0;
477 }
```

Blackhole Attack Mode Source Code:

```
1  #include "ns3/aodv-module.h"
2  #include "ns3/netanim-module.h"
3  #include "ns3/core-module.h"
4  #include "ns3/network-module.h"
5  #include "ns3/internet-module.h"
6  #include "ns3/applications-module.h"
7  #include "ns3/mobility-module.h"
8  #include "ns3/wifi-module.h"
9  #include "ns3/mesh-module.h"
10 #include "ns3/ipv4-global-routing-helper.h"
11 #include "ns3/olsr-helper.h"
12 #include "ns3/point-to-point-module.h"
13 #include "ns3/csma-module.h"
14 #include "ns3/netanim-module.h"
15 #include "ns3/flow-monitor-module.h"
16 #include "ns3/mobility-module.h"
17 #include "myapp.h"
18
19 #include <iostream>
20 #include <sstream>
21 #include <fstream>
22
23 NS_LOG_COMPONENT_DEFINE ("Blackhole");
```

```
24
25 using namespace ns3;
26
27 uint32_t MacTxDropCount, MacRxDropCount, PhyTxDropCount,
     PhyRxDropCount;
28 int packetCount;
29
30 void MacTxDrop(Ptr<const Packet> p)
31 {
32   NS_LOG_INFO("Packet Drop");
33   MacTxDropCount++;
34 }
35
36 void MacRxDrop(Ptr<const Packet> p)
37 {
38   NS_LOG_INFO("Packet Drop");
39   MacRxDropCount++;
40 }
41
42 void PrintDrop()
43 {
44   //std::cout << Simulator::Now().GetSeconds() << "\t" <<
       MacTxDropCount << "\t"<< MacRxDropCount << "\t" << PhyTxDropCount
       << "\t" << PhyRxDropCount << "\n";
45   std::cout << Simulator::Now().GetSeconds() << "\t" << MacTxDropCount
       << "\t"<< MacRxDropCount << "\n";
46   Simulator::Schedule(Seconds(20.0), &PrintDrop);
47 }
48
49 void PhyTxDrop(Ptr<const Packet> p)
50 {
51   NS_LOG_INFO("Packet Drop");
52   PhyTxDropCount++;
53 }
54 void PhyRxDrop(Ptr<const Packet> p)
55 {
56   NS_LOG_INFO("Packet Drop");
57   PhyRxDropCount++;
58 }
59 void ReceivePacket(Ptr<const Packet> p, const Address & addr)
60 {
61   packetCount++;
62   std::cout << packetCount << "\t" << Simulator::Now ().GetSeconds ()
       << "\t" << p->GetSize() <<"\n";
63 }
64
65 //****************************************Flow monitor
     ****************************************
66 void ThroughputMonitor(FlowMonitorHelper *fmhelper, Ptr<FlowMonitor>
     flowMon, double m_totalTime)
67 {
68     double localThrou = 0;
69     double pdf = 0;
70     double sent_pack=0;
71     double received_pack=0;
72
```

```
73
74      std::map<FlowId, FlowMonitor::FlowStats> flowStats = flowMon->
        GetFlowStats();//FlowId is index type of flowState and FlowMonitor
        ::FlowStats is data type of flowStates
75      Ptr<Ipv4FlowClassifier> classing = DynamicCast<Ipv4FlowClassifier>
        (fmhelper->GetClassifier());//Classifies packets by looking at
        their IP and TCP/UDP headers.
76 //From these packet headers, a tuple (source-ip, destination-ip,
        protocol, source-port, destination-port) is created, and a unique
        flow identifier is assigned for each different tuple combination
77
78
79      for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator
        stats = flowStats.begin(); stats != flowStats.end(); stats++)
80      {
81
82
83          Ipv4FlowClassifier::FiveTuple fiveTuple = classing->FindFlow(
        stats->first);
84
85          std::cout << "Flow ID      : " << stats->first << " ; " <<"
        source: "<< fiveTuple.sourceAddress << " -----> " <<"Destination:
        "<<          fiveTuple.destinationAddress << std::endl;
86          std::cout << "Tx Packets = " << stats->second.txPackets << std
        ::endl;
87          std::cout << "Rx Packets = " << stats->second.rxPackets << std
        ::endl;
88          std::cout << "Delay sum = " << stats->second.delaySum << std::
        endl;
89
90
91          sent_pack+=stats->second.txPackets;
92          received_pack+=stats->second.rxPackets;
93          //average end to end delay
94
95          std::cout << "Transmitted Bits: " << stats->second.txBytes *
        8.0 << " bits" << std::endl;
96          std::cout << "Received Bits: " << stats->second.rxBytes * 8.0
        << " bits" << std::endl;
97          std::cout << "Duration     : " << (stats->second.
        timeLastRxPacket.GetSeconds() - stats->second.timeFirstTxPacket.
        GetSeconds()) << " Seconds" << std::endl;
98          std::cout << "Throughput     : " << (stats->second.rxBytes *
        8.0)/(stats->second.timeLastRxPacket.GetSeconds() - stats->second.
        timeFirstTxPacket.GetSeconds())/1024/1024 << " Mbps" << std::endl;
99          std::cout << "Average ETE Delay = " << stats->second.delaySum/
        stats->second.rxPackets << std::endl;
100         std::cout << "-----------------------------------------" << std::
        endl;
101
102         localThrou = (stats->second.rxBytes * 8.0)/(stats->second.
        timeLastRxPacket.GetSeconds() - stats->second.timeFirstTxPacket.
        GetSeconds())/1024/1024;
103         pdf = (double)stats->second.rxPackets/stats->second.txPackets;
104     }
105
```

```
106
107     if(Simulator::Now().GetSeconds()==m_totalTime)
108     {
109         std::cout<<"***********************************"<<std::endl;
110         std::cout<<"Avereage Throughput = "<<localThrou<<"Mbps"<<std::
        endl;
111         std::cout<<"Average Packet Delivery Fraction = "<<pdf<<""<<std
        ::endl;
112         std::cout<<"***********************************"<<std::endl;
113         std::cout<<"Total Sent Packet="<<sent_pack/2<<std::endl;
114         std::cout<<"***********************************"<<std::endl;
115         std::cout<<"Total Received Packet="<<received_pack/2<<std::
        endl;
116         std::cout<<"***********************************"<<std::endl;
117     }
118     else
119     {
120         Simulator::Schedule(Seconds(1.0), ThroughputMonitor, fmhelper,
        flowMon, m_totalTime);
121     }
122
123 }
124
125 int main (int argc, char *argv[])
126 {
127   bool enableFlowMonitor = false;
128   std::string phyMode ("DsssRate1Mbps");
129   ApplicationContainer clientApps;
130
131   uint64_t simulationTime = 100; //seconds
132
133   CommandLine cmd;
134   cmd.AddValue ("EnableMonitor", "Enable Flow Monitor",
      enableFlowMonitor);
135   cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
136   cmd.Parse (argc, argv);
137
138 //
139 // Explicitly create the nodes required by the topology (shown above).
140 //
141   NS_LOG_INFO ("Create nodes.");
142   NodeContainer nc_meshNodes,nc_sta1, nc_sta2,nc_ap11, nc_ap12,nc_ap13
      , nc_ap14,nc_ap11Mbb1, nc_ap12Mbb1,nc_ap13Mbb1,
143                 nc_ap14Mbb1,not_malicious,malicious; // ALL Nodes
144
145   NetDeviceContainer meshDevices,staApDevices1,staApDevices2,
      staDevices1,staDevices2, apDevices1,apDevices2,ap11Mbb1Devices,
      ap12Mbb1Devices, ap13Mbb1Devices,ap14Mbb1Devices;
146   Ipv4InterfaceContainer interfaces,staAp_interfaces1,
      staAp_interfaces2,staAp_interfaces3,staAp_interfaces4,
      csmaInterfaces1,
147                          csmaInterfaces2,csmaInterfaces3,
      csmaInterfaces4;
148
149    CsmaHelper csma1, csma2, csma3, csma4;
150
```

```
151
152 //**********************************station device creation
    ****************************************
153     nc_sta1.Create(1);
154     nc_sta2.Create(1);
155
156 //**********************************mesh device creation
    ****************************************
157     nc_meshNodes.Create(3*3);
158
159         not_malicious.Add(nc_meshNodes.Get(1));
160         not_malicious.Add(nc_meshNodes.Get(2));
161         not_malicious.Add(nc_meshNodes.Get(3));
162         not_malicious.Add(nc_meshNodes.Get(4));
163         not_malicious.Add(nc_meshNodes.Get(5));
164         not_malicious.Add(nc_meshNodes.Get(6));
165         not_malicious.Add(nc_meshNodes.Get(7));
166         not_malicious.Add(nc_meshNodes.Get(8));
167         malicious.Add(nc_meshNodes.Get(0));
168
169 //**********************************Accesspoint device creation
    ****************************************
170     nc_ap11.Create(1);
171     nc_ap12.Create(1);
172     nc_ap13.Create(1);
173     nc_ap14.Create(1);
174
175     nc_ap11Mbb1=NodeContainer(nc_ap11, nc_meshNodes.Get(6));
176     nc_ap12Mbb1=NodeContainer(nc_ap12, nc_meshNodes.Get(0));
177     nc_ap13Mbb1=NodeContainer(nc_ap13, nc_meshNodes.Get(8));
178     nc_ap14Mbb1=NodeContainer(nc_ap14, nc_meshNodes.Get(3));
179
180   // Set up mesh
181   WifiHelper wifi;
182
183   YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
184   wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11);
185
186   YansWifiChannelHelper wifiChannel ;
187   wifiChannel.SetPropagationDelay ("ns3::
      ConstantSpeedPropagationDelayModel");
188   wifiChannel.AddPropagationLoss ("ns3::
      TwoRayGroundPropagationLossModel",
189                                   "SystemLoss", DoubleValue(1),
190                                  "HeightAboveZ", DoubleValue(1.5));
191
192   wifiPhy.Set ("TxPowerStart", DoubleValue(33));
193   wifiPhy.Set ("TxPowerEnd", DoubleValue(33));
194   wifiPhy.Set ("TxPowerLevels", UintegerValue(1));
195   wifiPhy.Set ("TxGain", DoubleValue(0));
196   wifiPhy.Set ("RxGain", DoubleValue(0));
197   wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue(-61.8));
198   wifiPhy.Set ("CcaMode1Threshold", DoubleValue(-64.8));
199
200   wifiPhy.SetChannel (wifiChannel.Create ());
201
```

```
202   // Add a non-QoS upper mac
203   NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
204
205   NetDeviceContainer devices;
206   meshDevices = wifi.Install (wifiPhy, wifiMac, nc_meshNodes);
207
208
209 // ********************************Setup WiFi for network
      1***********************************
210    WifiHelper wifi1 = WifiHelper::Default();
211    wifi1.SetStandard (WIFI_PHY_STANDARD_80211b);
212    wifi1.SetRemoteStationManager("ns3::AarfWifiManager");
213
214    NqosWifiMacHelper mac1 = NqosWifiMacHelper::Default();
215
216    Ssid ssid1 = Ssid("network-1");
217    mac1.SetType("ns3::StaWifiMac",
218                 "Ssid", SsidValue(ssid1),
219                 "ActiveProbing", BooleanValue(true));
220
221    staDevices1 = wifi1.Install(wifiPhy, mac1, nc_sta1.Get(0));
222
223
224    // Setup AP for network 1
225    mac1.SetType("ns3::ApWifiMac",
226                 "Ssid", SsidValue(ssid1));
227
228    apDevices1.Add(wifi1.Install(wifiPhy, mac1, nc_ap11.Get(0)));
229
230    staApDevices1.Add(staDevices1);
231    staApDevices1.Add(apDevices1);
232    //*****************************************End wifi
      1*****************************************
233
234
235    // ***********************************Setup WiFi for network
      2***********************************
236    WifiHelper wifi2 = WifiHelper::Default();
237    wifi2.SetStandard (WIFI_PHY_STANDARD_80211b);
238    wifi2.SetRemoteStationManager("ns3::AarfWifiManager");
239
240
241    NqosWifiMacHelper mac2 = NqosWifiMacHelper::Default();
242
243    Ssid ssid2 = Ssid("network-3");
244    mac2.SetType("ns3::StaWifiMac",
245                 "Ssid", SsidValue(ssid2),
246                 "ActiveProbing", BooleanValue(true));
247
248    staDevices2 = wifi2.Install(wifiPhy, mac2, nc_sta2);
249
250
251    // Setup AP for network 1
252    mac2.SetType("ns3::ApWifiMac",
253                 "Ssid", SsidValue(ssid2));
254
```

```
255     apDevices2.Add(wifi2.Install(wifiPhy, mac2, nc_ap14.Get(0)));
256
257     staApDevices2.Add(staDevices2);
258     staApDevices2.Add(apDevices2);
259
260     //*****************************************End wifi
        2*****************************************
261
262
263 //*********************************Property for csma1
        ****************************************
264     csma1.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
265     csma1.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
        );
266     ap11Mbb1Devices = csma1.Install (nc_ap11Mbb1);
267
268 //*********************************Property for csma2
        ****************************************
269     csma2.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
270     csma2.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
        );
271     ap12Mbb1Devices = csma2.Install (nc_ap12Mbb1);
272
273 //*********************************Property for csma3
        ****************************************
274     csma3.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
275     csma3.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
        );
276     ap13Mbb1Devices = csma3.Install (nc_ap13Mbb1);
277
278 //*********************************Property for csma4
        ****************************************
279     csma4.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
280     csma4.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
        );
281     ap14Mbb1Devices = csma4.Install (nc_ap14Mbb1);
282
283
284
285 //*********************************mobility for mesh
        ****************************************
286
287         MobilityHelper mobility;
288         Ptr<ListPositionAllocator> positionAlloc = CreateObject <
        ListPositionAllocator>();
289         positionAlloc ->Add(Vector(150, 0, 0)); // node0
290         positionAlloc ->Add(Vector(200, 0, 0)); // node1
291         positionAlloc ->Add(Vector(200, 300, 0)); // node2
292         positionAlloc ->Add(Vector(150,300, 0)); // node3
293         positionAlloc ->Add(Vector(150, 150, 0)); // node4
294         positionAlloc ->Add(Vector(200, 150, 0)); // node5
295         positionAlloc ->Add(Vector(250, 0, 0)); // node6
296         positionAlloc ->Add(Vector(250,150, 0)); // node7
297         positionAlloc ->Add(Vector(250, 300, 0)); // node8
298         mobility.SetPositionAllocator(positionAlloc);
299         mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel"
```

```
    );
300         mobility.Install(nc_meshNodes);
301
302 //**********************************mesh accesspoint 1 mobility
    ****************************************
303         MobilityHelper mobility1;
304         Ptr<ListPositionAllocator> positionAlloc1 = CreateObject <
    ListPositionAllocator>();
305         positionAlloc1 ->Add(Vector(250, -50, 0)); // ap11
306         mobility1.SetPositionAllocator(positionAlloc1);
307         mobility1.SetMobilityModel("ns3::ConstantPositionMobilityModel
    ");
308         mobility1.Install(nc_ap11);
309
310 //**********************************mesh accesspoint 2 mobility
    ****************************************
311         MobilityHelper mobility2;
312         Ptr<ListPositionAllocator> positionAlloc2 = CreateObject <
    ListPositionAllocator>();
313         positionAlloc2 ->Add(Vector(150, -50, 0)); // ap12
314         mobility2.SetPositionAllocator(positionAlloc2);
315         mobility2.SetMobilityModel("ns3::ConstantPositionMobilityModel
    ");
316         mobility2.Install(nc_ap12);
317
318 //**********************************mesh accesspoint 3 mobility
    ****************************************
319   MobilityHelper mobility3;
320   Ptr<ListPositionAllocator> positionAlloc3 = CreateObject <
      ListPositionAllocator>();
321   positionAlloc3 ->Add(Vector(250, 350, 0)); // ap13
322   mobility3.SetPositionAllocator(positionAlloc3);
323   mobility3.SetMobilityModel("ns3::ConstantPositionMobilityModel");
324   mobility3.Install(nc_ap13);
325
326 //**********************************mesh accesspoint 4 mobility
    ****************************************
327   MobilityHelper mobility4;
328   Ptr<ListPositionAllocator> positionAlloc4 = CreateObject <
      ListPositionAllocator>();
329   positionAlloc4 ->Add(Vector(150, 350, 0)); // ap13
330   mobility4.SetPositionAllocator(positionAlloc4);
331   mobility4.SetMobilityModel("ns3::ConstantPositionMobilityModel");
332   mobility4.Install(nc_ap14);
333
334 //**********************************station node 1 mobility
    ****************************************
335   MobilityHelper mobility5;
336   Ptr<ListPositionAllocator> positionAlloc5 = CreateObject <
      ListPositionAllocator>();
337   positionAlloc5 ->Add(Vector(250, -100, 0)); // ap13
338   mobility5.SetPositionAllocator(positionAlloc5);
339   mobility5.SetMobilityModel("ns3::ConstantPositionMobilityModel");
340   mobility5.Install(nc_sta1);
341
342 //**********************************station node 2 mobility
```

```
      ****************************************
343   MobilityHelper mobility6;
344   Ptr<ListPositionAllocator> positionAlloc6 = CreateObject <
        ListPositionAllocator>();
345   positionAlloc6 ->Add(Vector(150, 400, 0)); // ap13
346   mobility6.SetPositionAllocator(positionAlloc6);
347   mobility6.SetMobilityModel("ns3::ConstantPositionMobilityModel");
348   mobility6.Install(nc_sta2);
349
350 //  Enable AODV
351   AodvHelper aodv;
352   AodvHelper malicious_aodv;
353
354
355   // Set up internet stack
356   InternetStackHelper internetStack;
357   internetStack.SetRoutingHelper (aodv);
358   internetStack.Install (not_malicious);
359      internetStack.Install (nc_sta1);
360     internetStack.Install (nc_sta2);
361   internetStack.Install (nc_ap11);
362     internetStack.Install (nc_ap12);
363    internetStack.Install (nc_ap13);
364     internetStack.Install (nc_ap14);
365
366
367   malicious_aodv.Set("IsMalicious",BooleanValue(true)); // putting *
        false* instead of *true* would disable the malicious behavior of
        the node
368   internetStack.SetRoutingHelper (malicious_aodv);
369   internetStack.Install (malicious);
370
371 //**********************************Setup IP Address
      ****************************************
372     Ipv4AddressHelper address;
373     NS_LOG_INFO ("Assign IP Addresses.");
374     address.SetBase ("10.1.2.0", "255.255.255.0");
375     interfaces = address.Assign (meshDevices);
376
377
378     address.SetBase ("10.1.4.0", "255.255.255.0");
379     csmaInterfaces1 = address.Assign (ap11Mbb1Devices);
380
381     address.SetBase ("10.1.5.0", "255.255.255.0");
382     csmaInterfaces2 = address.Assign (ap12Mbb1Devices);
383
384     address.SetBase ("20.1.4.0", "255.255.255.0");
385     csmaInterfaces3 = address.Assign (ap13Mbb1Devices);
386
387     address.SetBase ("20.1.5.0", "255.255.255.0");
388     csmaInterfaces4 = address.Assign (ap14Mbb1Devices);
389
390     address.SetBase ("10.1.3.0", "255.255.255.0");
391     staAp_interfaces1 = address.Assign (staApDevices1);
392
393     address.SetBase ("20.1.3.0", "255.255.255.0");
```

```
394     staAp_interfaces2 = address.Assign (staApDevices2);
395
396
397
398   NS_LOG_INFO ("Create Applications.");
399
400 //*****************************************client to nc_ap11
      ****************************************
401    uint16_t sinkPort3 = 7;
402   Address sinkAddress3 (InetSocketAddress (staAp_interfaces1.
      GetAddress (1), sinkPort3)); // interface of n3
403   PacketSinkHelper packetSinkHelper3 ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort3));
404   ApplicationContainer sinkApps3 = packetSinkHelper3.Install (nc_ap11.
      Get (0)); //n3 as sink
405   sinkApps3.Start (Seconds (0.));
406   sinkApps3.Stop (Seconds (30.));
407
408   Ptr<Socket> ns3UdpSocket3 = Socket::CreateSocket (nc_sta1.Get (0),
      UdpSocketFactory::GetTypeId ()); //source at n1
409
410   // Create UDP application at n1
411   Ptr<MyApp> app3 = CreateObject<MyApp> ();
412   app3->Setup (ns3UdpSocket3, sinkAddress3, 1024, 50, DataRate ("500
      Kbps"));
413   nc_sta1.Get (0)->AddApplication (app3);
414   app3->SetStartTime (Seconds (5.));
415   app3->SetStopTime (Seconds (30.));
416
417 //*****************************************nc_ap11 to  mesh node
      6*****************************************
418   uint16_t sinkPort = 6;
419   Address sinkAddress (InetSocketAddress (csmaInterfaces1.GetAddress
      (1), sinkPort)); // interface of n3
420   PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort));
421   ApplicationContainer sinkApps = packetSinkHelper.Install (
      nc_meshNodes.Get (6)); //n3 as sink
422   sinkApps.Start (Seconds (15.));
423   sinkApps.Stop (Seconds (30.));
424
425   Ptr<Socket> ns3UdpSocket = Socket::CreateSocket (nc_ap11.Get (0),
      UdpSocketFactory::GetTypeId ()); //source at n1
426
427   // Create UDP application at n1
428   Ptr<MyApp> app = CreateObject<MyApp> ();
429   app->Setup (ns3UdpSocket, sinkAddress, 1024, 50, DataRate ("500Kbps"
      ));
430   nc_ap11.Get (0)->AddApplication (app);
431   app->SetStartTime (Seconds (25.));
432   app->SetStopTime (Seconds (30.));
433
434 //***************************************** mesh node 6 to mesh node
      3*****************************************
435   uint16_t sinkPort1 = 5;
436   Address sinkAddress1 (InetSocketAddress (interfaces.GetAddress (3),
```

```
      sinkPort1)); // interface of n3
437   PacketSinkHelper packetSinkHelper1 ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort1));
438   ApplicationContainer sinkApps1 = packetSinkHelper1.Install (
      nc_meshNodes.Get (3)); //n3 as sink
439   sinkApps1.Start (Seconds (30.));
440   sinkApps1.Stop (Seconds (45.));
441
442   Ptr<Socket> ns3UdpSocket1 = Socket::CreateSocket (nc_meshNodes.Get
      (6), UdpSocketFactory::GetTypeId ()); //source at n1
443
444   // Create UDP application at n1
445   Ptr<MyApp> app1 = CreateObject<MyApp> ();
446   app1->Setup (ns3UdpSocket1, sinkAddress1, 1024, 50, DataRate ("500
      Kbps"));
447   nc_meshNodes.Get (6)->AddApplication (app1);
448   app1->SetStartTime (Seconds (35.));
449   app1->SetStopTime (Seconds (45.));
450
451 //****************************************** mesh node 3 to ap13
      ******************************************
452   uint16_t sinkPort2 = 4;
453   Address sinkAddress2 (InetSocketAddress (csmaInterfaces4.GetAddress
      (0), sinkPort2)); // interface of n3
454   PacketSinkHelper packetSinkHelper2 ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort2));
455   ApplicationContainer sinkApps2 = packetSinkHelper2.Install (
      nc_ap14Mbb1.Get (0)); //n3 as sink
456   sinkApps2.Start (Seconds (45.0));
457   sinkApps2.Stop (Seconds (60.));
458
459   Ptr<Socket> ns3UdpSocket2 = Socket::CreateSocket (nc_meshNodes.Get
      (3), UdpSocketFactory::GetTypeId ()); //source at n1
460
461   // Create UDP application at n1
462   Ptr<MyApp> app2 = CreateObject<MyApp> ();
463   app2->Setup (ns3UdpSocket2, sinkAddress2, 1024, 50, DataRate ("500
      Kbps"));
464   nc_meshNodes.Get (3)->AddApplication (app2);
465   app2->SetStartTime (Seconds (50.));
466   app2->SetStopTime (Seconds (60.));
467
468 //******************************************nc_ap13 to sta2
      ******************************************
469  uint16_t sinkPort4 = 8;
470   Address sinkAddress4 (InetSocketAddress (csmaInterfaces4.GetAddress
      (1), sinkPort4)); // interface of n3
471   PacketSinkHelper packetSinkHelper4("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort4));
472   ApplicationContainer sinkApps4 = packetSinkHelper4.Install (nc_sta1.
      Get (0)); //n3 as sink
473   sinkApps4.Start (Seconds (50.));
474   sinkApps4.Stop (Seconds (75.));
475
476   Ptr<Socket> ns3UdpSocket4 = Socket::CreateSocket (nc_ap14.Get (0),
      UdpSocketFactory::GetTypeId ()); //source at n1
```

```
477
478   // Create UDP application at n1
479   Ptr<MyApp> app4 = CreateObject<MyApp> ();
480   app4->Setup (ns3UdpSocket4, sinkAddress4, 1024, 50, DataRate ("500
        Kbps"));
481   nc_ap14.Get (0)->AddApplication (app4);
482   app4->SetStartTime (Seconds (65.));
483   app4->SetStopTime (Seconds (75.));
484
485
486
487
488       //Ptr<OutputStreamWrapper> routingStream = Create<
        OutputStreamWrapper> ("blackholewmn.routes", std::ios::out);
489       //aodv.PrintRoutingTableAllAt (Seconds (45), routingStream);
490
491  // Trace Received Packets
492   Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3::
        PacketSink/Rx", MakeCallback (&ReceivePacket));
493  // Trace Collisions
494   //Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::
        WifiNetDevice/Mac/MacTxDrop", MakeCallback(&MacTxDrop));
495   //Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::
        WifiNetDevice/Mac/MacRxDrop", MakeCallback(&MacRxDrop));
496   //Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::
        WifiNetDevice/Phy/PhyRxDrop", MakeCallback(&PhyRxDrop));
497   //Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::
        WifiNetDevice/Phy/PhyTxDrop", MakeCallback(&PhyTxDrop));
498
499 //
500 // Calculate Throughput using Flowmonitor
501 //
502   FlowMonitorHelper flowmon;
503   Ptr<FlowMonitor> monitor = flowmon.InstallAll();
504
505    //ThroughputMonitor(&flowmon,monitor ,simulationTime);
506    Simulator::Stop (Seconds(simulationTime+1));
507 //
508 // Now, do the actual simulation.
509 //
510   NS_LOG_INFO ("Run Simulation.");
511
512   PrintDrop();
513   monitor->CheckForLostPackets ();
514
515   Simulator::Run ();
516   Simulator::Destroy ();
517
518   return 0;
519 }
```

Greyhole Attack Mode Source Code:

```
1 #include "ns3/aodv-module.h"
2 #include "ns3/netanim-module.h"
3 #include "ns3/core-module.h"
4 #include "ns3/network-module.h"
```

```
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mesh-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/olsr-helper.h"
#include "ns3/point-to-point-module.h"
#include "ns3/csma-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/mobility-module.h"
#include "myapp.h"
#include "ns3/simulator.h"
#include "ns3/nstime.h"
#include "ns3/command-line.h"
#include "ns3/random-variable-stream.h"
#include <iostream>
#include <sstream>
#include <fstream>

NS_LOG_COMPONENT_DEFINE ("Blackhole");

using namespace ns3;


    int        m_xSize=3;
    int        m_ySize=3;
    int        m_sta=1;
    int        m_ap=1;
    uint16_t   m_packetSize=1024;
    uint16_t   m_NumOfPacket=50;

int packetCount,received_bits=0,transmitted_bits=0;
float throughput,localThrou,pdf;
float first_transmittedpacket,last_transmittedpacket,sum_of_ete_delay;

FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor;

uint64_t simulationTime = 30; //seconds

int drop_packet=5;
float random_value1,random_value2,random_value3,random_value4,
    random_value5;
int  random_value11,random_value22,random_value33,random_value44,
    random_value55;

void ReceivePacket(Ptr<const Packet> p, const Address & addr)
{
    InetSocketAddress transport = InetSocketAddress::ConvertFrom (addr
    );

    std::map<FlowId, FlowMonitor::FlowStats> flowStats1 = monitor->
    GetFlowStats();

```

```
57      if(packetCount==1)
58      {
59          first_transmittedpacket=Simulator::Now ().GetSeconds ();
60      }
61    if(transport.GetIpv4 () == "10.1.2.7")
62   {
63       if(random_value11==packetCount)
64      {
65        std::cout<<"Grey hole attack in wireless mesh network:: Packet
      dropped!!!"<<std::endl;
66        packetCount++;
67       }
68      else if(random_value22==packetCount)
69      {
70        std::cout<<"Grey hole attack in wireless mesh network:: Packet
      dropped!!!"<<std::endl;
71        packetCount++;
72       }
73      else if(random_value33==packetCount)
74      {
75
76        std::cout<<"Grey hole attack in wireless mesh network:: Packet
      dropped!!!"<<std::endl;
77        packetCount++;
78       }
79      else if(random_value44==packetCount)
80      {
81
82        std::cout<<"Grey hole attack in wireless mesh network:: Packet
      dropped!!!"<<std::endl;
83        packetCount++;
84       }
85      else if(random_value55==packetCount)
86      {
87        std::cout<<"Grey hole attack in wireless mesh network:: Packet
      dropped!!!"<<std::endl;
88        packetCount++;
89       }
90       else
91       {
92        packetCount++;
93        std::cout << packetCount << "\t" << Simulator::Now ().GetSeconds
       () << "\t" << p->GetSize() <<"\n";
94        if(packetCount==m_NumOfPacket)
95      {
96
97          last_transmittedpacket=Simulator::Now ().GetSeconds ();
98          sum_of_ete_delay=last_transmittedpacket -
      first_transmittedpacket;
99
100        std::cout << "Flow ID      : " << 1 << " ; " <<"source: "<<
      transport.GetIpv4 ()  << " ------> " <<"Destination: "<<transport.
      GetIpv4 () << std::endl;
101         std::cout<<"***********************************"<<std::endl;
102         std::cout<<"Total Sent Packet="<<m_NumOfPacket<<std::endl;
103         std::cout<<"***********************************"<<std::endl;
```

```
104         std::cout<<"Total Received Packet="<<(packetCount-drop_packet)
    <<std::endl;
105         std::cout<<"*************************************"<<std::endl;
106         std::cout << "Duration     : " <<Simulator::Now ().GetSeconds
    ()<< "Seconds" << std::endl;
107         std::cout<<"*************************************"<<std::endl;
108         std::cout << "transmitted bits : " <<transmitted_bits<< "bits"
     << std::endl;
109         std::cout<<"*************************************"<<std::endl;
110         std::cout << "received bits : " <<(received_bits-(1024*
    drop_packet))<< "bits" << std::endl;
111         std::cout<<"*************************************"<<std::endl;
112         std::cout << "Throughput     : " << (received_bits-(1024*
    drop_packet))/(Simulator::Now ().GetSeconds () -
    first_transmittedpacket)/1024/1024 << " Mbps" <<std::endl;
113         localThrou = (received_bits-(1024*drop_packet))/(Simulator::
    Now ().GetSeconds () - first_transmittedpacket)/1024/1024;
114         pdf = (double)(packetCount-drop_packet)/m_NumOfPacket;
115         std::cout<<"*************************************"<<std::endl;
116         std::cout<<"Avereage Throughput = "<<localThrou<<"Mbps"<<std::
    endl;
117         std::cout<<"*************************************"<<std::endl;
118         std::cout<<"Avereage End to End Delay = "<<sum_of_ete_delay
    *1000/(packetCount-drop_packet)<<"ms"<<std::endl;
119         std::cout<<"*************************************"<<std::endl;
120         std::cout<<"Average Packet Delivery Fraction = "<<pdf<<""<<std
    ::endl;
121         std::cout<<"*************************************"<<std::endl;
122         std::cout<<"Percentage of Packet loss = "<<(1-pdf)*100<<"%"<<
    std::endl;
123         std::cout<<"*************************************"<<std::endl;
124         std::cout<<"Greyhole Attack In wireless mesh network.....
    Simulation Stopped"<<std::endl;
125         std::cout<<"*************************************"<<std::endl;
126         packetCount=0;
127         transmitted_bits=0;
128         received_bits=0;
129         sum_of_ete_delay=0;
130       return;
131      }
132       }
133
134      transmitted_bits+=p->GetSize()*8;
135      received_bits+=p->GetSize()*8;
136 }
137 else
138 {
139        packetCount++;
140        std::cout << packetCount << "\t" << Simulator::Now ().GetSeconds
       () << "\t" << p->GetSize() <<"\n";
141        if(packetCount==m_NumOfPacket)
142      {
143         last_transmittedpacket=Simulator::Now ().GetSeconds ();
144         sum_of_ete_delay=last_transmittedpacket -
    first_transmittedpacket;
145
```

```
146         std::cout << "Flow ID      : " << 1 << " ; " <<"source: "<<
    transport.GetIpv4 ()  << " ------> " <<"Destination: "<<transport.
    GetIpv4 () << std::endl;
147         std::cout<<"***********************************"<<std::endl;
148         std::cout<<"Total Sent Packet="<<m_NumOfPacket<<std::endl;
149         std::cout<<"***********************************"<<std::endl;
150         std::cout<<"Total Received Packet="<<(packetCount-drop_packet)
    <<std::endl;
151         std::cout<<"***********************************"<<std::endl;
152         std::cout << "Duration     : " <<Simulator::Now ().GetSeconds
    ()<< "Seconds" << std::endl;
153         std::cout<<"***********************************"<<std::endl;
154         std::cout << "transmitted bits : " <<transmitted_bits<< "bits"
     << std::endl;
155         std::cout<<"***********************************"<<std::endl;
156         std::cout << "received bits : " <<(received_bits)<< "bits" <<
    std::endl;
157         std::cout<<"***********************************"<<std::endl;
158         std::cout << "Throughput     : " << (received_bits)/(Simulator
    ::Now ().GetSeconds () - first_transmittedpacket)/1024/1024 << "
    Mbps" <<std::endl;
159         localThrou = (received_bits)/(Simulator::Now ().GetSeconds ()
    - first_transmittedpacket)/1024/1024;
160         pdf = (double)(packetCount)/m_NumOfPacket;
161         std::cout<<"***********************************"<<std::endl;
162         std::cout<<"Avereage Throughput = "<<localThrou<<"Mbps"<<std::
    endl;
163         std::cout<<"***********************************"<<std::endl;
164         std::cout<<"Avereage End to End Delay = "<<sum_of_ete_delay
    *1000/(packetCount)<<"ms"<<std::endl;
165         std::cout<<"***********************************"<<std::endl;
166         std::cout<<"Average Packet Delivery Fraction = "<<pdf<<""<<std
    ::endl;
167         std::cout<<"***********************************"<<std::endl;
168         std::cout<<"Percentage of Packet loss = "<<(1-pdf)*100<<"%"<<
    std::endl;
169         std::cout<<"***********************************"<<std::endl;
170         packetCount=0;
171         transmitted_bits=0;
172         received_bits=0;
173       return;
174      }
175       }
176
177      transmitted_bits+=p->GetSize()*8;
178      received_bits+=p->GetSize()*8;
179 }
180
181
182 int main (int argc, char *argv[])
183 {
184   Time::SetResolution (Time::NS);
185   LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
186   LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
187
188   bool enableFlowMonitor = false;
```

```
189   std::string phyMode ("DsssRate1Mbps");
190
191   CommandLine cmd;
192   cmd.AddValue ("EnableMonitor", "Enable Flow Monitor",
        enableFlowMonitor);
193   cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
194   cmd.Parse (argc, argv);
195     ApplicationContainer clientApps;
196
197
198           Ptr<UniformRandomVariable> uv = CreateObject<
        UniformRandomVariable> ();
199           random_value1=uv->GetValue ();
200           random_value1=(random_value1*100)/2;
201           random_value11= (int)random_value1;
202
203           random_value2=uv->GetValue ();
204           random_value2=(random_value2*100)/2;
205           random_value22= (int)random_value2;
206
207           random_value3=uv->GetValue ();
208           random_value3=(random_value3*100)/2;
209           random_value33= (int)random_value3;
210
211           random_value4=uv->GetValue ();
212           random_value4=(random_value4*100)/2;
213           random_value44= (int)random_value4;
214
215           random_value5 = uv->GetValue ();
216           random_value5=(random_value5*100)/2;
217           random_value55= (int)random_value5;
218
219
220
221     NS_LOG_INFO ("Create nodes.");
222     NodeContainer nc_meshNodes,nc_sta1, nc_sta2,nc_ap11, nc_ap12,
        nc_ap13, nc_ap14,nc_ap11Mbb1, nc_ap12Mbb1,nc_ap13Mbb1,
223                   nc_ap14Mbb1,not_malicious,malicious; // ALL Nodes
224
225     NetDeviceContainer meshDevices,staApDevices1,staApDevices2,
        staDevices1,staDevices2, apDevices1,apDevices2,ap11Mbb1Devices,
        ap12Mbb1Devices, ap13Mbb1Devices,ap14Mbb1Devices;
226     Ipv4InterfaceContainer interfaces,staAp_interfaces1,
        staAp_interfaces2,staAp_interfaces3,staAp_interfaces4,
        csmaInterfaces1,
227                            csmaInterfaces2,csmaInterfaces3,
        csmaInterfaces4;
228
229     CsmaHelper csma1, csma2, csma3, csma4;
230
231
232 //************************************station device creation
        ****************************************
233     nc_sta1.Create(m_sta);
234     nc_sta2.Create(m_sta);
235
```

```
236 //**********************************mesh device creation
    ****************************************
237     nc_meshNodes.Create(m_xSize * m_ySize);
238
239     not_malicious.Add(nc_meshNodes.Get(1));
240     not_malicious.Add(nc_meshNodes.Get(2));
241     not_malicious.Add(nc_meshNodes.Get(3));
242     not_malicious.Add(nc_meshNodes.Get(4));
243     not_malicious.Add(nc_meshNodes.Get(5));
244     not_malicious.Add(nc_meshNodes.Get(6));
245     not_malicious.Add(nc_meshNodes.Get(7));
246     not_malicious.Add(nc_meshNodes.Get(8));
247     malicious.Add(nc_meshNodes.Get(0));
248
249 //**********************************Accesspoint device creation
    ****************************************
250     nc_ap11.Create(m_ap);
251     nc_ap12.Create(m_ap);
252     nc_ap13.Create(m_ap);
253     nc_ap14.Create(m_ap);
254
255     nc_ap11Mbb1=NodeContainer(nc_ap11, nc_meshNodes.Get(6));
256     nc_ap12Mbb1=NodeContainer(nc_ap12, nc_meshNodes.Get(0));
257     nc_ap13Mbb1=NodeContainer(nc_ap13, nc_meshNodes.Get(8));
258     nc_ap14Mbb1=NodeContainer(nc_ap14, nc_meshNodes.Get(3));
259
260     // Set up mesh
261     WifiHelper wifi;
262
263     YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
264     wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11);
265
266     YansWifiChannelHelper wifiChannel ;
267     wifiChannel.SetPropagationDelay ("ns3::
    ConstantSpeedPropagationDelayModel");
268     wifiChannel.AddPropagationLoss ("ns3::
    TwoRayGroundPropagationLossModel",
269                                     "SystemLoss", DoubleValue(1),
270                                     "HeightAboveZ", DoubleValue(1.5));
271
272     wifiPhy.Set ("TxPowerStart", DoubleValue(33));
273     wifiPhy.Set ("TxPowerEnd", DoubleValue(33));
274     wifiPhy.Set ("TxPowerLevels", UintegerValue(1));
275     wifiPhy.Set ("TxGain", DoubleValue(0));
276     wifiPhy.Set ("RxGain", DoubleValue(0));
277     wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue(-61.8));
278     wifiPhy.Set ("CcaMode1Threshold", DoubleValue(-64.8));
279
280     wifiPhy.SetChannel (wifiChannel.Create ());
281
282     // Add a non-QoS upper mac
283     NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
284
285     NetDeviceContainer devices;
286     meshDevices = wifi.Install (wifiPhy, wifiMac, nc_meshNodes);
287
```

```
288
289 // ******************************Setup WiFi for network
    1**********************************
290     WifiHelper wifi1 = WifiHelper::Default();
291     wifi1.SetStandard (WIFI_PHY_STANDARD_80211b);
292     wifi1.SetRemoteStationManager("ns3::AarfWifiManager");
293
294     NqosWifiMacHelper mac1 = NqosWifiMacHelper::Default();
295
296     Ssid ssid1 = Ssid("network-1");
297     mac1.SetType("ns3::StaWifiMac",
298                  "Ssid", SsidValue(ssid1),
299                  "ActiveProbing", BooleanValue(true));
300
301     staDevices1 = wifi1.Install(wifiPhy, mac1, nc_sta1.Get(0));
302
303
304     // Setup AP for network 1
305     mac1.SetType("ns3::ApWifiMac",
306                  "Ssid", SsidValue(ssid1));
307
308     apDevices1.Add(wifi1.Install(wifiPhy, mac1, nc_ap11.Get(0)));
309
310     staApDevices1.Add(staDevices1);
311     staApDevices1.Add(apDevices1);
312     //*****************************************End wifi
    1*****************************************
313
314
315     // ***********************************Setup WiFi for network
    2**********************************
316     WifiHelper wifi2 = WifiHelper::Default();
317     wifi2.SetStandard (WIFI_PHY_STANDARD_80211b);
318     wifi2.SetRemoteStationManager("ns3::AarfWifiManager");
319
320
321     NqosWifiMacHelper mac2 = NqosWifiMacHelper::Default();
322
323     Ssid ssid2 = Ssid("network-3");
324     mac2.SetType("ns3::StaWifiMac",
325                  "Ssid", SsidValue(ssid2),
326                  "ActiveProbing", BooleanValue(true));
327
328     staDevices2 = wifi2.Install(wifiPhy, mac2, nc_sta2);
329
330
331     // Setup AP for network 1
332     mac2.SetType("ns3::ApWifiMac",
333                  "Ssid", SsidValue(ssid2));
334
335     apDevices2.Add(wifi2.Install(wifiPhy, mac2, nc_ap14.Get(0)));
336
337     staApDevices2.Add(staDevices2);
338     staApDevices2.Add(apDevices2);
339
340     //*****************************************End wifi
```

```
    2*****************************************

//*********************************Property for csma1
    *****************************************
    csma1.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
    csma1.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
    );
    ap11Mbb1Devices = csma1.Install (nc_ap11Mbb1);

//*********************************Property for csma2
    *****************************************
    csma2.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
    csma2.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
    );
    ap12Mbb1Devices = csma2.Install (nc_ap12Mbb1);

//*********************************Property for csma3
    *****************************************
    csma3.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
    csma3.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
    );
    ap13Mbb1Devices = csma3.Install (nc_ap13Mbb1);

//*********************************Property for csma4
    *****************************************
    csma4.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
    csma4.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560))
    );
    ap14Mbb1Devices = csma4.Install (nc_ap14Mbb1);



//*********************************mobility for mesh
    *****************************************

    MobilityHelper mobility;
    Ptr<ListPositionAllocator> positionAlloc = CreateObject <
    ListPositionAllocator>();
    positionAlloc ->Add(Vector(150, 0, 0)); // node0
    positionAlloc ->Add(Vector(200, 0, 0)); // node1
    positionAlloc ->Add(Vector(200, 300, 0)); // node2
    positionAlloc ->Add(Vector(150,300, 0)); // node3
    positionAlloc ->Add(Vector(150, 150, 0)); // node4
    positionAlloc ->Add(Vector(200, 150, 0)); // node5
    positionAlloc ->Add(Vector(250, 0, 0)); // node6
    positionAlloc ->Add(Vector(250,150, 0)); // node7
    positionAlloc ->Add(Vector(250, 300, 0)); // node8
    mobility.SetPositionAllocator(positionAlloc);
    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(nc_meshNodes);

//*********************************mesh accesspoint 1 mobility
    *****************************************
    MobilityHelper mobility1;
    Ptr<ListPositionAllocator> positionAlloc1 = CreateObject <
```

```
    ListPositionAllocator>();
385     positionAlloc1->Add(Vector(250, -50, 0)); // ap11
386     mobility1.SetPositionAllocator(positionAlloc1);
387     mobility1.SetMobilityModel("ns3::ConstantPositionMobilityModel");
388     mobility1.Install(nc_ap11);
389
390 //**********************************mesh accesspoint 2 mobility
    ****************************************
391     MobilityHelper mobility2;
392     Ptr<ListPositionAllocator> positionAlloc2 = CreateObject<
    ListPositionAllocator>();
393     positionAlloc2->Add(Vector(150, -50, 0)); // ap12
394     mobility2.SetPositionAllocator(positionAlloc2);
395     mobility2.SetMobilityModel("ns3::ConstantPositionMobilityModel");
396     mobility2.Install(nc_ap12);
397
398 //**********************************mesh accesspoint 3 mobility
    ****************************************
399     MobilityHelper mobility3;
400     Ptr<ListPositionAllocator> positionAlloc3 = CreateObject<
    ListPositionAllocator>();
401     positionAlloc3->Add(Vector(250, 350, 0)); // ap13
402     mobility3.SetPositionAllocator(positionAlloc3);
403     mobility3.SetMobilityModel("ns3::ConstantPositionMobilityModel");
404     mobility3.Install(nc_ap13);
405
406 //**********************************mesh accesspoint 4 mobility
    ****************************************
407     MobilityHelper mobility4;
408     Ptr<ListPositionAllocator> positionAlloc4 = CreateObject<
    ListPositionAllocator>();
409     positionAlloc4->Add(Vector(150, 350, 0)); // ap13
410     mobility4.SetPositionAllocator(positionAlloc4);
411     mobility4.SetMobilityModel("ns3::ConstantPositionMobilityModel");
412     mobility4.Install(nc_ap14);
413
414 //**********************************station node 1 mobility
    ****************************************
415     MobilityHelper mobility5;
416     Ptr<ListPositionAllocator> positionAlloc5 = CreateObject<
    ListPositionAllocator>();
417     positionAlloc5->Add(Vector(250, -100, 0)); // ap13
418     mobility5.SetPositionAllocator(positionAlloc5);
419     mobility5.SetMobilityModel("ns3::ConstantPositionMobilityModel");
420     mobility5.Install(nc_sta1);
421
422 //**********************************station node 2 mobility
    ****************************************
423     MobilityHelper mobility6;
424     Ptr<ListPositionAllocator> positionAlloc6 = CreateObject<
    ListPositionAllocator>();
425     positionAlloc6->Add(Vector(150, 400, 0)); // ap13
426     mobility6.SetPositionAllocator(positionAlloc6);
427     mobility6.SetMobilityModel("ns3::ConstantPositionMobilityModel");
428     mobility6.Install(nc_sta2);
429
```

```
430 //  Enable AODV
431     AodvHelper aodv;
432
433     // Set up internet stack
434     InternetStackHelper internetStack;
435     internetStack.SetRoutingHelper (aodv);
436     internetStack.Install (not_malicious);
437     internetStack.Install (malicious);
438     internetStack.Install (nc_sta1);
439     internetStack.Install (nc_sta2);
440     internetStack.Install (nc_ap11);
441     internetStack.Install (nc_ap12);
442     internetStack.Install (nc_ap13);
443     internetStack.Install (nc_ap14);
444
445
446 //**********************************Setup IP Address
    ******************************************
447     Ipv4AddressHelper address;
448     NS_LOG_INFO ("Assign IP Addresses.");
449     address.SetBase ("10.1.2.0", "255.255.255.0");
450     interfaces = address.Assign (meshDevices);
451
452
453     address.SetBase ("10.1.4.0", "255.255.255.0");
454     csmaInterfaces1 = address.Assign (ap11Mbb1Devices);
455
456     address.SetBase ("10.1.5.0", "255.255.255.0");
457     csmaInterfaces2 = address.Assign (ap12Mbb1Devices);
458
459     address.SetBase ("20.1.4.0", "255.255.255.0");
460     csmaInterfaces3 = address.Assign (ap13Mbb1Devices);
461
462     address.SetBase ("20.1.5.0", "255.255.255.0");
463     csmaInterfaces4 = address.Assign (ap14Mbb1Devices);
464
465     address.SetBase ("10.1.3.0", "255.255.255.0");
466     staAp_interfaces1 = address.Assign (staApDevices1);
467
468     address.SetBase ("20.1.3.0", "255.255.255.0");
469     staAp_interfaces2 = address.Assign (staApDevices2);
470
471
472
473     NS_LOG_INFO ("Create Applications.");
474
475 //****************************************client to nc_ap11
    ******************************************
476     uint16_t sinkPort3 = 7;
477     Address sinkAddress3 (InetSocketAddress (staAp_interfaces1.
    GetAddress (1), sinkPort3)); // interface of n3
478     PacketSinkHelper packetSinkHelper3 ("ns3::UdpSocketFactory",
    InetSocketAddress (Ipv4Address::GetAny (), sinkPort3));
479     ApplicationContainer sinkApps3 = packetSinkHelper3.Install (
    nc_ap11.Get (0)); //n3 as sink
480     sinkApps3.Start (Seconds (0.));
```

```
481     sinkApps3.Stop (Seconds (10.));
482
483     Ptr<Socket> ns3UdpSocket3 = Socket::CreateSocket (nc_sta1.Get (0),
        UdpSocketFactory::GetTypeId ()); //source at n1
484
485     Ptr<MyApp> app3 = CreateObject<MyApp> ();
486     app3->Setup (ns3UdpSocket3, sinkAddress3, 1024, m_NumOfPacket,
      DataRate ("250Kbps"));
487     nc_sta1.Get (0)->AddApplication (app3);
488     app3->SetStartTime (Seconds (5.));
489     app3->SetStopTime (Seconds (10.));
490
491 //*****************************************nc_ap11 to  mesh node
      6*****************************************
492     uint16_t sinkPort = 6;
493     Address sinkAddress (InetSocketAddress (csmaInterfaces1.GetAddress
        (1), sinkPort)); // interface of n3
494     PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort));
495     ApplicationContainer sinkApps = packetSinkHelper.Install (
      nc_meshNodes.Get (6)); //n3 as sink
496     sinkApps.Start (Seconds (10.));
497     sinkApps.Stop (Seconds (20.));
498
499     Ptr<Socket> ns3UdpSocket = Socket::CreateSocket (nc_ap11.Get (0),
      UdpSocketFactory::GetTypeId ()); //source at n1
500
501     Ptr<MyApp> app = CreateObject<MyApp> ();
502     app->Setup (ns3UdpSocket, sinkAddress, m_packetSize, m_NumOfPacket
      , DataRate ("500Kbps"));
503     nc_ap11.Get (0)->AddApplication (app);
504     app->SetStartTime (Seconds (15.));
505     app->SetStopTime (Seconds (20.));
506
507 //***************************************** mesh node 6 to mesh node
      3*****************************************
508     uint16_t sinkPort1 = 5;
509     Address sinkAddress1 (InetSocketAddress (interfaces.GetAddress (5)
      , sinkPort1)); // interface of n3
510     PacketSinkHelper packetSinkHelper1 ("ns3::UdpSocketFactory",
      InetSocketAddress (Ipv4Address::GetAny (), sinkPort1));
511     ApplicationContainer sinkApps1 = packetSinkHelper1.Install (
      nc_meshNodes.Get (5)); //n3 as sink
512     sinkApps1.Start (Seconds (20.));
513     sinkApps1.Stop (Seconds (30.));
514
515     Ptr<Socket> ns3UdpSocket1 = Socket::CreateSocket (nc_meshNodes.Get
        (6), UdpSocketFactory::GetTypeId ()); //source at n1
516
517     Ptr<MyApp> app1 = CreateObject<MyApp> ();
518     app1->Setup (ns3UdpSocket1, sinkAddress1, m_packetSize,
      m_NumOfPacket, DataRate ("500Kbps"));
519     nc_meshNodes.Get (6)->AddApplication (app1);
520     app1->SetStartTime (Seconds (25.));
521     app1->SetStopTime (Seconds (30.));
522
```

```
523 if(drop_packet==0)
524 {
525
526     //************************************** mesh node 6 to mesh
    node 3**************************************
527     uint16_t sinkPort9 = 3;
528     Address sinkAddress9 (InetSocketAddress (interfaces.GetAddress (3)
    , sinkPort9)); // interface of n3
529     PacketSinkHelper packetSinkHelper9 ("ns3::UdpSocketFactory",
    InetSocketAddress (Ipv4Address::GetAny (), sinkPort9));
530     ApplicationContainer sinkApps9 = packetSinkHelper9.Install (
    nc_meshNodes.Get (3)); //n3 as sink
531     sinkApps9.Start (Seconds (30.));
532     sinkApps9.Stop (Seconds (40.));
533
534     Ptr<Socket> ns3UdpSocket9 = Socket::CreateSocket (nc_meshNodes.Get
    (5), UdpSocketFactory::GetTypeId ()); //source at n1
535
536     Ptr<MyApp> app9 = CreateObject<MyApp> ();
537     app9->Setup (ns3UdpSocket9, sinkAddress9, m_packetSize,
    m_NumOfPacket, DataRate ("500Kbps"));
538     nc_meshNodes.Get (5)->AddApplication (app9);
539     app9->SetStartTime (Seconds (35.));
540     app9->SetStopTime (Seconds (40.));
541 //************************************** mesh node 3 to ap13
    ****************************************
542     uint16_t sinkPort2 = 4;
543     Address sinkAddress2 (InetSocketAddress (csmaInterfaces4.
    GetAddress (1), sinkPort2)); // interface of n3
544     PacketSinkHelper packetSinkHelper2 ("ns3::UdpSocketFactory",
    InetSocketAddress (Ipv4Address::GetAny (), sinkPort2));
545     ApplicationContainer sinkApps2 = packetSinkHelper2.Install (
    nc_meshNodes.Get (3)); //n3 as sink
546     sinkApps2.Start (Seconds (40.0));
547     sinkApps2.Stop (Seconds (50.));
548
549     Ptr<Socket> ns3UdpSocket2 = Socket::CreateSocket (nc_ap14.Get (0),
     UdpSocketFactory::GetTypeId ()); //source at n1
550
551     Ptr<MyApp> app2 = CreateObject<MyApp> ();
552     app2->Setup (ns3UdpSocket2, sinkAddress2, m_packetSize,
    m_NumOfPacket, DataRate ("500Kbps"));
553     nc_ap14.Get (0)->AddApplication (app2);
554     app2->SetStartTime (Seconds (45.));
555     app2->SetStopTime (Seconds (50.));
556
557 //**************************************nc_ap13 to sta2
    ****************************************
558     uint16_t sinkPort4 = 2;
559     Address sinkAddress4 (InetSocketAddress (staAp_interfaces2.
    GetAddress (1), sinkPort4)); // interface of n3
560     PacketSinkHelper packetSinkHelper4("ns3::UdpSocketFactory",
    InetSocketAddress (Ipv4Address::GetAny (), sinkPort4));
561     ApplicationContainer sinkApps4 = packetSinkHelper4.Install (
    nc_ap14.Get (0)); //n3 as sink
562     sinkApps4.Start (Seconds (50));
```

```
563     sinkApps4.Stop (Seconds (60.));
564
565     Ptr<Socket> ns3UdpSocket4 = Socket::CreateSocket (nc_sta2.Get (0),
        UdpSocketFactory::GetTypeId ()); //source at n1
566
567     Ptr<MyApp> app4 = CreateObject<MyApp> ();
568     app4->Setup (ns3UdpSocket4, sinkAddress4, m_packetSize,
       m_NumOfPacket, DataRate ("250Kbps"));
569     nc_sta2.Get (0)->AddApplication (app4);
570     app4->SetStartTime (Seconds (55.));
571     app4->SetStopTime (Seconds (60.));
572
573 }
574
575 //*****************************************flow monitor
       *****************************************
576     monitor = flowmon.InstallAll();
577
578 //*****************************************Trace Received Packet
       *****************************************
579     Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3
       ::PacketSink/Rx", MakeCallback (&ReceivePacket));
580     //Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::
       WifiNetDevice/Mac/MacTxDrop", MakeCallback(&MacTxDrop));
581     //Config::ConnectWithoutContext("/NodeList/*/DeviceList/*/$ns3::
       WifiNetDevice/Mac/MacRxDrop", MakeCallback(&MacRxDrop));
582     monitor->CheckForLostPackets ();
583
584     Simulator::Stop (Seconds(simulationTime+1));
585     Simulator::Run ();
586
587     Simulator::Destroy ();
588
589     return 0;
590 }
```

Appendix B

NS-3 802.11s modules

In this appendix, the modules implemented in C++ and how they interconnect with each other is presented.

B.1 MeshHelper

void SetSpreadInterfaceChannels

Parameters: (ChannelPolicy)
Set the channel policy which can be SPREAD CHANNELS or ZERO CHANNEL: Spread or not spread frequency channels of MP interfaces. If set to true different non-overlapping 20MHz frequency channels will be assigned to different mesh point interfaces.

void SetStackInstaller

Parameters: (std::string type, std::string n0 = "", const AttributeValue & v0 = Empty- AttributeValue (),...)
You need to tell which Mesh stack do you want, in this case Dot11sStack, so you can use the characteristics of the 802.11s. n0 and v0 are the name and the value of the attribute to set, respectively. For example you can set the root node in the mesh network.

void SetNumberOfInterfaces

Parameters: (uint32 t nInterfaces)
Set a number of interfaces in a mesh network.

NetDeviceContainer Install

Parameters: (const WifiPhyHelper & phyHelper, NodeContainer c)
Install 802.11s mesh device and protocols on given node list. The phyHelper is the Wifi PHY helper and c is the list of nodes to install. This function returns the list of created mesh point devices, see MeshPointDevice (see next section).

void SetMacType

Parameters: (std::string n0 = "", const AttributeValue & v0 = EmptyAttributeValue (),...)
Uses the class MeshWifiInterfaceMac and n0 and v0 are the name and the value of the attribute to set, respectively. The values that can be set are the next ones:

- BeaconInterval : Beacon Interval. Initial value: 0.5 seconds
- RandomStart: Window when beacon generating starts (uniform random) in seconds. Initial value: 0.5 seconds
- BeaconGeneration: Enable/Disable Beaconing. Initial value: enabled
- TxOkHeader : The header of successfully transmitted packet.
- TxErrHeader : The header of unsuccessfully transmitted packet.

This class uses the RegularWifiMac class where you can set the QoS support which enable 802.11e/WMM-style (By default is disable). And at the same time this class has as parent class WifiMac which we can use to modify values like CTS timeout, ACK timeout, SIFTS, EIFS-DIFS, duration of a slot, PIFS or the Ssid.

void SetRemoteStationManager

Parameters: (std::string type, std::string n0 = " ", const AttributeValue & v0 = Empty- AttributeValue (),...)
With this function, using the variable type we define which station manager do we want. A part from a constant bit rate value, the following rate control algorithms implemented in NS-3: AARF, AARF-CD, AMRR, ARF, CARA, Ideal, Minstrel, ONOE and RRAA. The one selected by default is the ARF. They all use has as parent class WifiRemoteStationManager, and using the n0 and v0 you can set the maximum number of retransmission attempts for an RTS and data packets as well as the threshold to decide when to use a RTS/CTS handshake before sending a data packet and the one to decide when to fragment them. As described in IEEE Std. 802.11-2007, Section 9.2.6. and 9.4. This value will not have any effect on some rate control algorithms. Here we can also set a wifi mode for non-unicast transmissions.

void SetNumberOfInterfaces

Parameters: (uint32 t nInterfaces)
Set a number of interfaces in a mesh network.

void SetStandard

Parameters: (enum WifiPhyStandard standard)
Allows you to select the following standards: 802.11 with 5 or 10 Mhz, 802.11a and 802.11b. The one set by default is the 802.11a.

MeshPointDevice

Mesh point is a virtual net device which is responsible for aggregating and coordinating real devices (mesh interfaces), and hosting all mesh-related level 2 protocols. One of hosted L2 protocols must implement L2RoutingProtocol interface and is used for packets forwarding. From the level 3 point of view MeshPointDevice is similar to a bridge network device, but the packets, which going through may be changed (because L2 protocols may require their own headers or tags).

void AddInterface

Parameters: (Ptr <NetDevice>port)
Attach new interface to the station. Interface must support 48-bit MAC address and only MeshPointDevice can have IP address, but not individual interfaces.

bool SetMtu

Parameters: (const uint16 t mtu)
Set the MAC-level Maximum Transmission Unit in bytes and returns whether the MTU value was within legal bounds. Override for default MTU defined on a per-type basis.

void SetRoutingProtocol

Parameters: (Ptr <MeshL2RoutingProtocol>protocol)
Register the mesh routing protocol to be used by this mesh point. Protocol must be already installed on this mesh point.