SECURING DSR PROTOCOL DEFENDING AGAINST SYBIL ATTACK IN MANET

A MAIN PROJECT REPORT

Submitted by

SHARADH R	**AC19UCS103**
SUBASH A L	**AC19UCS116**
VIGNESH K	**AC19UCS135**

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

ADHIYAMAAN COLLEGE OF ENGINEERING, (AUTONOMOUS)

DR. M. G. R NAGAR, HOSUR- 635 130

ANNA UNIVERSITY::CHENNAI 600 025

APRIL 2023

BONAFIDE CERTIFICATE

Certified that this main project report **"SECURING DSR PROTOCOL FOR DEFENDING SYBIL ATTACK IN MANET"** is the bonafide work of **SHARADH R (AC19UCS103), SUBASH A L (AC19UCS116) ,VIGNESH K (AC19UCS135)** who carried out the project work under my supervision.

SIGNATURE

Dr. G. FATHIMA, M.E., Ph.D.,

HEAD OF THE DEPARTMENT

PROFESSOR

Department of CSE,
Adhiyamaan College of Engineering,
Dr. M.G.R Nagar,
Hosur-635130.

SIGNATURE

Mrs. D.M.VIJAYA LAKSHMI, M.E.,

SUPERVISOR,

ASSISTANT PROFESSOR

Department of CSE,
Adhiyamaan College of Engineering,
Dr. M.G.R Nagar,
Hosur-635130.

Submitted for the Main Project VIVA- VOCE Examination held on **…….**at **Adhiyamaan College of Engineering**, **Hosur.**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

It is one of the most efficient tasks in life to choose appropriate words to express one's gratitude to the beneficiaries. We are very much grateful to God who helped us through the project and how moulded us into what we are today.

We are grateful to our beloved principal **Dr. G. RANGANATH, M.E., Ph.D.,** Adhiyamaan College of Engineering (Autonomous), Hosur for providing the opportunity to do this work on the premises.

We extend our hearty gratitude to **Dr. G. FATHIMA, M.E., Ph.D.**, Head of the Department, Department of Computer Science And Engineering, Adhiyamaan College of Engineering, Hosur, for her guidance and valuable suggestions and encouragement throughout this project.

We acknowledged our gratitude to **Mrs.D.M.VIJAYA LAKSHMI, M.E.** Assistant professor, Department of Computer Science And Engineering, Adhiyamaan College of Engineering (Autonomous), Hosur whose immense Support, encouragement and valuable guidance made us complete this project successfully.

We also extend our thanks to the Project Coordinator **Dr. B.GOPINATHAN M.E., Ph.D.,** and all Staff Members for their support in completing this project successfully.

Finally, we would like to thank our parents, without their motivation and Support would not have been possible for us to complete this project successfully.

ABSTRACT

Mobile ad hoc networks (MANETs) are infrastructure-free networks in which nodes are free to move in any direction. These networks use specific routing protocols that can create a path between nodes that are not within transmission range of each other. Because MANETs are easy to configure, they are mostly used in areas where infrastructure is not available, such as military and rescue operations, etc. Due to the open approach, MANETs are always vulnerable to external and internal attacks such as Denial of Service (DoS), Flooding, Worms hole, Black hole, Gray hole Sinkholes etc. There is no central point of administration. In this project, we focus on the Sybil attack. An SYBIL attack is an individual type of association layer attack in a specially selected mobile network. In this attack, a fake hub is introduced that has a direct path to reach the target. So it collects the entire packet from the source and drops it. Nowadays, it is very difficult to secure the network against such attacks. The Dynamic Source Routing (DSR) algorithm uses caching concepts to store all newly created routing paths in mobile ad hoc networks. Route caching aggressively uses DSR. With source routing, it is possible to cache each overhead path without causing loops. Forwarding nodes cache the source path from a packet and forward it for future use. The destination also meets all the requirements. Thus, the source learns many alternative paths to the destination, which are stored in the cache. Here, we propose a novel approach to prevent Sybil attacks in DSR based on route caching. In this approach, once a Sybil node is detected in the MANET during path construction, we pass the Sybil node id to the DSR path function. In this function, routes are

ready to be added to the route cache; however, adding each route to the route cache is decided by parsing those routes for the presence of a Sybil node id. This process uses only the normal time of the caching process. In this project, we propose a cache-based Sybil attack prevention algorithm for DSR routing protocols in MANET. Simulations in NS-2 show that our proposed mechanism greatly reduces the packet drop rate with a very low false positive rate.

SECURING DSR PROTOCOL FOR DEFENDING SYBIL ATTACK IN MANET

LIST OF ABBREVIATIONS

ACRONYM	ABBREVIATION
NAM	Network animator
NS2	Network simulator
TCL	Tool command language,
MANET	Mobile Adhoc Network
DSR	Dynamic Source Routing (DSR)
DSDV	Destination-Sequenced Distance-Vector Routing)
MAC	Medium access control
TCP	Transmission Control Protocol
ADOV	Ad-hoc On-demand Distance Vector
WSNs	Wireless sensor networks

LIST OF FIGURES

CHAPTER - 1

INTRODUCTION

1.1 Project Overview

Wireless sensor networks (WSNs) are an emerging technology consisting of small, low-power devices that integrate limited computation, sensing and radio communication capabilities. The deployment of the Wireless Sensor Network is mostly done for information collecting and remote monitoring. WSN's are typically used in an open, uncontrolled environment, often in hostile territories. The open nature of the wireless communication channels, the lack of infrastructure, the fast deployment practices, and the hostile environments where sensor nodes are deployed, make them vulnerable to a wide range of security attacks.

1.2 Purpose

A Sybil attack is a type of security threat in which a malicious node in a MANET impersonates multiple identities and pretends to be multiple nodes in the network. Such attacks can lead to various security issues, including the disruption of network services, the interception of confidential data, and the injection of false information into the network. The DSR protocol is a widely used routing protocol in MANETs that enables nodes to dynamically discover and maintain routes to other nodes in the network. Our project aims to propose a solution to secure the DSR protocol against Sybil attacks by implementing authentication mechanisms that can verify the identity of nodes in the network.

CHAPTER - 2

2. LITERATURE SURVEY

The Dynamic Source Routing (DSR) protocol is a popular routing protocol used in Mobile Ad hoc Networks (MANETs). However, like any other wireless network, MANETs are vulnerable to a variety of attacks, and the Sybil attack is one of the most significant threats that can compromise the DSR protocol's security.

A Sybil attack involves an attacker creating multiple fake identities, or nodes, within the network, and then using these identities to overwhelm the routing protocol's routing tables. By doing so, the attacker can manipulate the network's routing paths and potentially disrupt communication between legitimate nodes.

To defend against Sybil attacks in the DSR protocol, several research studies have proposed different mechanisms. Here are a few examples:

1. "A Sybil Attack Detection Mechanism for DSR in MANETs" by A. M. Al-Salihy, et al. (2017): This paper proposes a mechanism for detecting Sybil attacks in the DSR protocol. The mechanism is based on monitoring the routing messages exchanged between nodes and identifying patterns that are indicative of Sybil attacks.
2. "A Sybil Attack Defense Mechanism Based on Trust and Reputation in DSR Protocol" by F. Al-Turjman, et al. (2015): This paper proposes a defence mechanism for the DSR protocol based on trust and reputation. The mechanism involves assigning reputation scores to nodes based on their behaviour in the network and using these scores to select trustworthy routes.

3. "A Resource-Based Sybil Attack Detection Mechanism for DSR Protocol in MANETs" by Y. Chen, et al. (2016): This paper proposes a resource-based mechanism for detecting Sybil attacks in the DSR protocol. The mechanism involves testing the resource capabilities of nodes and identifying those that have insufficient resources to support multiple identities.
4. "A Collaborative Sybil Attack Detection Mechanism in DSR-Based MANETs" by X. Zhou, et al. (2018): This paper proposes a collaborative mechanism for detecting Sybil attacks in the DSR protocol. The mechanism involves nodes sharing information about their neighbours and using this information to identify suspicious behaviour.
5. "A Reputation-Based Sybil Attack Detection Mechanism for DSR in MANETs" by S. Lee, et al. (2014): This paper proposes a reputation-based mechanism for detecting Sybil attacks in the DSR protocol. The mechanism involves assigning reputation scores to nodes based on their behaviour in the network and using these scores to identify suspicious behaviour.
6. "A Novel Sybil Attack Detection Scheme for DSR in MANETs" by M. A. Azab, et al. (2016): This paper proposes a novel mechanism for detecting Sybil attacks in the DSR protocol. The mechanism involves monitoring the routing messages exchanged between nodes and identifying anomalies that are indicative of Sybil attacks.
7. "A Distributed Sybil Attack Detection Mechanism for DSR in MANETs" by T. Xiang, et al. (2018): This paper proposes a distributed mechanism for detecting Sybil attacks in the DSR protocol. The mechanism involves nodes sharing information about their neighbours and using this information to identify suspicious behaviour.
8. "A Location-Based Sybil Attack Detection Mechanism for DSR in MANETs" by W. Lu, et al. (2016): This paper proposes a location-based mechanism for

detecting Sybil attacks in the DSR protocol. The mechanism involves using location information to verify the physical presence of nodes and detect fake identities.

9. "A Social Network Analysis-Based Sybil Attack Detection Mechanism for DSR in MANETs" by L. Zhang, et al. (2019): This paper proposes a social network analysis-based mechanism for detecting Sybil attacks in the DSR protocol. The mechanism involves analysing the communication patterns of nodes to identify suspicious behaviour.
10. "A Lightweight Sybil Attack Detection Mechanism for DSR in MANETs" by M. Li, et al. (2017): This paper proposes a lightweight mechanism for detecting Sybil attacks in the DSR protocol. The mechanism involves monitoring the routing messages exchanged between nodes and identifying patterns that are indicative of Sybil attacks, without requiring significant computational resources.

CHAPTER - 3

SYSTEM ANALYSIS

3.1. Existing System

- In the existing system, The core objective of this existing system is to provide comparative performance analysis on the routing protocols using a ViSim tool.
- This tool focuses on MANET (Mobile Ad-Hoc Network) protocols for proactive - DSDV (Destination-Sequenced Distance-Vector Routing) and ADOV(Ad-hoc On-demand Distance Vector).
- Throughput, Goodput (packets and packet size) and Routing load (packets and packet size) are the parameters to make a comparison between the routing protocols for performance analysis.

Disadvantages:

- Due to security checks during the routing mechanism, the delay has been increased by 0.2 to 0.5 seconds.
- The quality of service of the mitigation technique used in this existing method to improve network performance is quite low compared to other techniques.

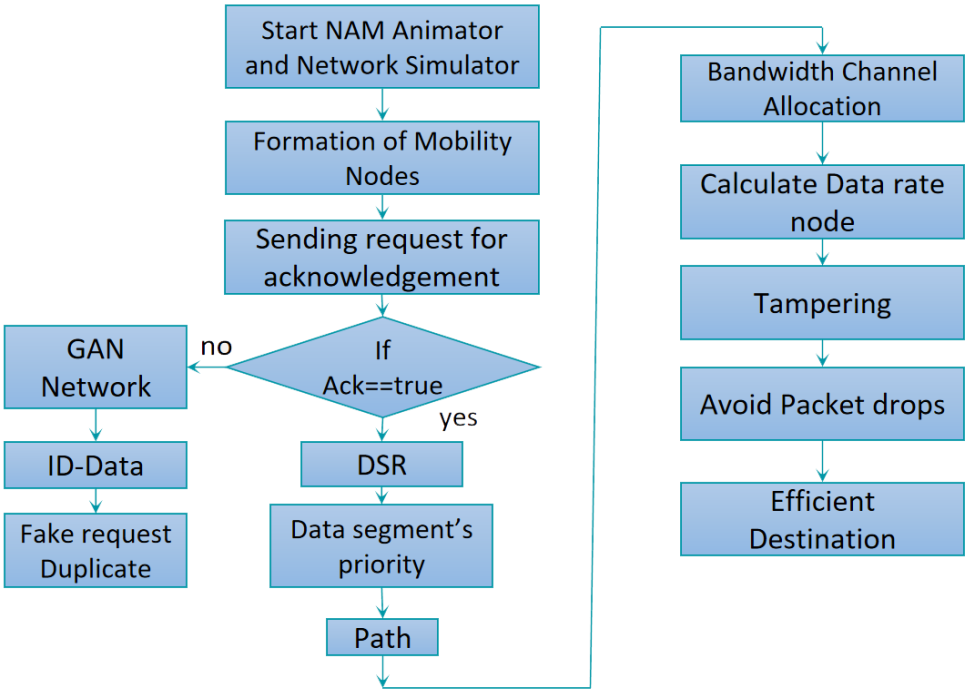
3.2. Proposed System

In this project, In this proposed technique, we present a new approach to prevent Sybil attacks in DSR using DSR's route cache mechanism. When detecting a Sybil node or a misbehaving node, during path construction processing, we need to get the Sybil node ID and pass it to add to the DSR path function. Functional paths are ready to be added to the route cache, but before adding each path to the route cache, we parse those paths for the presence of the Sybil node id. If a Sybil node appears in a path, we simply need to list that path and add all the remaining paths for the intended source-destination pairwise communication.

Advantages:

- This process uses only the normal time of the caching process.
- This minimises latency compared to previous Sybil attack detection mechanisms.
- The packet drop rate is drastically reduced.

3.3. System Architecture



MODULES

1. NAM animator and Network Simulator Module
2. Acknowledgement request Module
3. DSR Module
4. Dumping Black node Module
5. Destination Module

3.4. MODULES DESCRIPTION

3.4.1. NAM Animator

NAM is a Tcl/TK-based animation tool for displaying network simulation routes and real-world packet routes. It has a graphical interface that can provide

information such as the number of dropped packets on each link. The network animator "NAM" which is located in the initial position and starts moving the node wirelessly started in 1990 as a simple tool to animate packet trace data. Nam started at LBL. NAM was developed in collaboration with the VINT project. It is currently developed as an open-source project hosted on Sourceforge.

- This trace data is usually derived as output from a network simulator such as ns or from actual network measurements such as using tcpdump.
- We can run NAM either with a command 'name'.
- where " is the name of the NAM trace file that was generated by NS or can be run directly from a Tcl simulation script to visualise node motion.
- The NAM window is shown in the following figure.

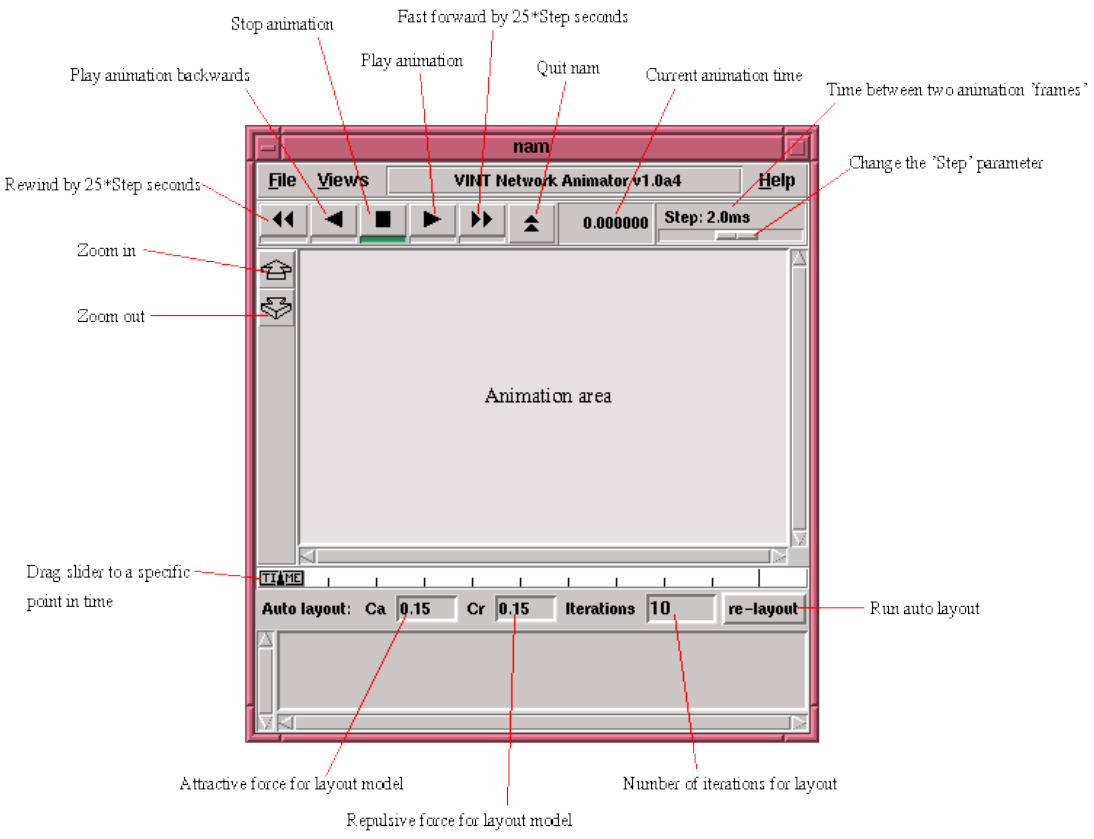


Fig. no: 1 NAM Animator Window

- We can use NAM in a network simulation by creating a name trace file and then running the name trace file on a TCL script.

Network Simulator

Initialization of a network simulator involves setting up the initial conditions and parameters for the simulated network. This process is typically done before running any simulations, and it ensures that the simulated network is configured correctly and ready to run. The specific steps involved in initialising a network simulator will depend on the particular simulator being used, as well as the specific network being simulated. However, some common tasks involved in network simulator initialization include:

1. **Defining the network topology:** This involves specifying the physical layout of the network, including the nodes, links, and other network components.
2. **Configuring network devices:** This involves setting up the various devices that make up the network, including routers, switches, and other networking equipment.
3. **Configuring network protocols:** This involves setting up the various network protocols that will be used by the simulated network, such as TCP/IP, BGP, OSPF, and so on.
4. **Defining traffic patterns:** This involves specifying the types of traffic that will be simulated on the network, including the traffic sources, destinations, and volumes.
5. **Setting up performance metrics:** This involves defining the performance metrics that will be used to evaluate the performance of the simulated network, such as throughput, latency, and packet loss.

Once the network simulator has been initialised, it can be used to run simulations and analyse the performance of the simulated network under different conditions.

3.4.2. Acknowledgement request Module

The acknowledgement request process is an important aspect of data transmission in computer networks. It ensures that the data sent by one device is received by the other device and that the data is accurate and complete. The process involves the following steps:

1. **Sender sends data:** The sender device sends the data to the receiver device.
2. **A receiver receives data:** The receiver device receives the data sent by the sender.
3. **The receiver sends acknowledgement:** Once the receiver has received the data, it sends an acknowledgement to the sender to confirm that the data has been received successfully.
4. **Sender receives acknowledgement:** The sender device receives the acknowledgement sent by the receiver, which confirms that the data has been received successfully.
5. **Resend data:** If the sender does not receive an acknowledgement within a certain period, it assumes that the data was not received successfully and resends the data.

The acknowledgement request process is used in various protocols such as the Transmission Control Protocol (TCP) to ensure reliable data transmission. In TCP, the sender sends a packet of data and waits for an acknowledgement from the receiver before sending the next packet. If the sender does not receive an acknowledgement within a certain period, it resends the packet. This process continues until all the packets have been successfully transmitted and acknowledged by the receiver.

3.4.3. DSR Module

Dynamic Source Routing (DSR) is a routing protocol used in wireless ad-hoc networks. It is a reactive protocol that allows nodes in the network to dynamically discover and maintain routes to other nodes in the network. DSR is designed to operate in environments where network topology may change frequently and unpredictably. In DSR, each node maintains a route cache that contains information about the routes to other nodes in the network. When a node wants to send a packet to another node, it first checks its route cache to see if it has a valid route to the destination. If a valid route is found, the node uses it to forward the packet. If a valid route is not found, the node initiates a route discovery process. The route discovery process in DSR involves broadcasting a route request packet to all nodes in the network. The route request packet contains the source and destination addresses of the packet, as well as a unique identifier for the request. When a node receives a route request packet, it checks its route cache to see if it has a valid route to the destination. If it does, it responds with a route reply packet that contains the route information. If it does not have a valid route, it rebroadcasts the route request packet to all of its neighbours.

As the route request packet is forwarded through the network, each node that receives it adds its address to the route in the packet. When the packet reaches the destination node, it sends a route reply packet back to the source node. The route reply packet contains the complete route from the source to the destination. Once the source node receives the route reply packet, it adds the route information to its route cache and uses it to forward the original packet to the destination node. The route

information is also cached by all of the intermediate nodes along the route so that future packets can be forwarded more efficiently.

One of the advantages of DSR is that it allows for multiple routes to be discovered and maintained between any two nodes in the network. This provides increased network reliability and fault tolerance. Additionally, DSR can be used with a variety of different wireless network technologies, making it a flexible and adaptable protocol for a wide range of applications.
DSR is a reactive routing protocol, meaning that it only creates routes when needed. This is in contrast to proactive routing protocols, which create routes ahead of time, even if they are not needed at the moment. In a wireless ad-hoc network, where the topology can change frequently, a reactive protocol like DSR can be more efficient and scalable.

In DSR, each node maintains a route cache that contains information about the routes to other nodes in the network. The route cache is organised as a table, with each entry containing the destination address and the route information. The route information includes a sequence of nodes that must be traversed to reach the destination, as well as a timestamp indicating when the route was last used.

When a node wants to send a packet to a destination, it first checks its route cache to see if it has a valid route. If a valid route is found, the packet is forwarded using that route. If a valid route is not found, the node initiates a route discovery process. The route discovery process begins with the node broadcasting a route request packet to all of its neighbours. The route request packet contains the source and destination addresses of the packet, as well as a unique identifier for the request. When a node receives a route request packet, it checks its route cache to see if it has a valid route

to the destination. If it does, it responds with a route reply packet that contains the route information. If it does not have a valid route, it rebroadcasts the route request packet to all of its neighbours.

As the route request packet is forwarded through the network, each node that receives it adds its address to the route in the packet. When the packet reaches the destination node, it sends a route reply packet back to the source node. The route reply packet contains the complete route from the source to the destination. Once the source node receives the route reply packet, it adds the route information to its route cache and uses it to forward the original packet to the destination node. The route information is also cached by all of the intermediate nodes along the route so that future packets can be forwarded more efficiently. One of the benefits of DSR is that it supports multiple routes between any two nodes in the network. This provides increased network reliability and fault tolerance. DSR also has low overhead, since route discovery is only initiated when a valid route is not found in the route cache. However, DSR can be less efficient than proactive protocols in networks with high mobility, since route discovery and route maintenance can consume significant bandwidth and processing resources.

3.4.4. Dumping Black node Module

The term "dumping Black node" is not commonly used, and without more context, it's difficult to provide a specific explanation. However, I will explain what a black node is in a general sense. In computer science, a black node is a type of node used in certain data structures, such as red-black trees or B-trees. These data structures are used to organise and store large sets of data efficiently.

- In a red-black tree, a black node is a node that is coloured black, which is used to balance the tree. The red-black tree is a type of self-balancing binary search tree where each node has either a red or black colour assigned to it.
- The tree is balanced in such a way that the height of the tree is always logarithmic to the number of nodes in the tree. This balance ensures that the tree remains efficient and that searches, insertions, and deletions can be performed in logarithmic time.
- In a B-tree, a black node is a node that is not a leaf node and has the maximum number of keys that the data structure can hold.
- B-trees are a type of self-balancing tree data structure that can store a large number of keys in a single node.
- B-trees are often used in file systems and databases, where large amounts of data need to be accessed quickly.

Without more context, it's difficult to provide a specific explanation of "dumping Black node." It's possible that this term could refer to the process of removing a black node from a data structure or the action of writing the contents of a black node to a file or other storage medium. However, this is speculation, and without more information, it's impossible to provide a more detailed explanation.

Dumping in computer science generally refers to the process of transferring the contents of memory to a storage device such as a file, disk, or tape. Memory dumping is often performed for diagnostic purposes or to save the system state before shutting down or restarting a computer. A memory dump typically contains the contents of the computer's RAM (random access memory) at a specific point in time. This data can be useful for diagnosing system crashes, software bugs, or other issues that may be difficult to reproduce. Memory dumps can also be used to analyse system performance or to monitor the behaviour of certain applications.

In addition to memory dumps, other types of dumps exist in computer science. For example, a database dump is a copy of a database that is typically used for backup or migration purposes. A network dump, also known as a packet capture, is a record of the network traffic that has passed through a particular network interface.

In summary, "dumping" in computer science refers to the process of transferring the contents of memory or other data to a storage device for analysis, backup, or other purposes. Without further context, it is difficult to provide a more specific explanation of "dumping Black node."

3.4.5. Destination Module

To transmit data from a source to a destination, the data needs to be sent over a series of interconnected devices or network nodes. To ensure that the data reaches its destination efficiently and with minimal data loss, the network needs to select an optimal route for data transmission. This involves evaluating the available routes based on factors such as the distance, bandwidth, latency, and reliability of each path. The network then selects the route that offers the highest data transmission rate and the least amount of data loss.

Various routing protocols such as Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) are commonly used in computer networks to select the most efficient route for data transmission. These protocols help to ensure that data is delivered quickly and reliably to its destination, while also minimising the risk of data loss or corruption during transmission.

In a computer network, data is typically transmitted from a source device to a destination device through a series of network nodes. These network nodes can include routers, switches, and other networking devices. The path that the data takes through the network is known as the routing path.

The routing path that data takes can have a significant impact on the speed and reliability of the data transmission. For example, if the routing path has a high level of congestion or network traffic, the data transmission may be slowed down or even interrupted. Similarly, if the routing path has a lot of network errors or packet loss, the data may be corrupted or lost during transmission.

To avoid these issues and ensure the efficient transmission of data, computer networks use various routing protocols. These protocols help to select the most optimal route for data transmission based on several factors, including:

1. Distance: The distance between the source and destination devices, and the distance between network nodes along the routing path.
2. Bandwidth: The amount of data that can be transmitted over a network connection in a given period.
3. Latency: The amount of time it takes for data to travel from the source to the destination device.
4. Reliability: The likelihood of errors or data loss occurring during transmission.

Routing protocols such as OSPF and BGP use these and other factors to determine the most efficient route for data transmission. They also allow network administrators to configure various network parameters, such as the priority of certain types of traffic or the routing path for specific destinations.

Overall, routing is an essential component of computer networking and plays a critical role in ensuring the fast, reliable transmission of data across networks of all sizes.

CHAPTER - 4

SYSTEM REQUIREMENT

4.1. Hardware Requirement

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the system does and not how it should be implemented.

- Processors: Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM
- SSD/HDD : 1 TB
- Operating systems : Linux*(Ubuntu)

4.2. Software Requirement

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification.

- Front end : NAM (Network Animator)
- Back end : TCL
- Software : NS2 (Network Simulator)
- Platform : Linux (Ubuntu)

4.3. Software Description

Nam (Network Animator)

NAM is a Tcl/TK-based animation tool for viewing network simulation traces and real-world packet trace data. The design theory behind NAM was to create an animator that can read large animation data sets and be extensible enough so that it could be used in different network visualisation situations. Under this constraint, NAM was designed to read simple animation event commands from a large trace file. To handle large animation data sets a minimum amount of information is kept in memory. Event commands are kept in the file and reread from the file whenever necessary.

The first step to using NAM is to produce the trace file. The trace file contains topology information, e.g., nodes, links, as well as packet traces. The detailed format is described in the section. Usually, the trace file is generated by ns. During an ns simulation, the user can produce topology configurations, layout information, and packet traces using tracing events in ns. However, any application can generate a NAM trace file.

When the trace file is generated, it is ready to be animated by NAM. Upon startup, NAM will read the trace file, create a topology, pop up a window, do layout if necessary, and then pause at time 0. Through its user interface, NAM provides control over many aspects of animation. These functionalities will be described in detail in the USER INTERFACE section.

Nam Command Line Options

nam [-g \<geometry\>] [-t \<graphInput\>] [-i \<interval\>] [-j \<startup time\>]

[-k \<initial socket port number\>] [-N \<application name\>] [-c \<cache size\>]

[-f \<configuration file\>] [-r initial animation rate]

[-a] [-p] [-S]

[\<tracefile(s)\>]

Command Line Options

-g — Specify geometry of the window upon startup.

-t — Instruct NAM to use tk graph, and specify input file NAME for tk graph.

-i — [Information for this option may not be accurate] Specify rate (real) milliseconds as the screen update rate. The default rate is 50ms (i.e., 20 frames per second). Note that the X server may not be able to keep up with this rate, in which case the animation will run as fast as the X server allows it to (at 100% cpu utilisation).

-N — Specify the application name of this NAM instance. This application name may later be used in peer synchronisation.

-c — The maximum size of the cache used to store 'active' objects when doing animating in reverse.

-f — Name of the initialization files to be loaded during startup. In this file, users can define functions which will be called in the trace file. An example for this is the 'link-up' and 'link-down' events of dynamic links in ns. (Refer to $ns rtmodel for detail, and tcl/ex/simple-dyn.tcl in your ns direc-tory for example). Example

initialization files can be found at ex/sample.nam.tcl and ex/dynamicnam.conf.

-a	Create a separate instance of nam.
-p	Print out NAM trace file format.
-S	Enable synchronous X behaviour so it is easier for graphics debugging. For UNIX systems running X only.
<tracefile>	is the name of the file containing the trace data to be animated. If <tracefile> cannot be read, nam will try to open <tracefile>.nam.

Animation Objects

Nam does animation using the following building blocks which are defined below:

Node

Nodes are created from 'n' trace events in a trace file. It represents a source, host, or router. Nam will skip over any duplicate definitions for the same node. A node may have three shapes, (circle, square, and hexagon), but once created it cannot change its shape. Nodes can change their colour during animation. Nodes can be labelled.

Link

Links are created between nodes to form a network topology. Internally NAM links consist of 2 simplex links. The trace event 'l' creates two simplex links and does another necessary setup. Therefore, from a user's perspective, all links are duplex links. Links can be labelled and also can change colour during the animation. Links can be labelled as well.

Queue

Queues need to be constructed in NAM between two nodes. A NAM queue is associated with only one edge of a duplex link. Queues are visualised as stacked packets. Packets are stacked along a line, the angle between the line and the horizontal line can be specified in the queue trace event.

Packet

Packets are visualised as a block with an arrow. The direction of the arrow shows the flow direction of the packet. Queued packets are shown as little squares. A packet may be dropped from a queue or a link. Dropped packets are shown as falling rotating squares and disappear at the end of the screen. Unfortunately, due to NAM's design, dropped packets are not visible during backward animation.

Agent

Agents are used to separating protocol states from nodes. They are always associated with nodes. An agent has a name, which is a unique identifier of the agent. It is shown as a square with its name inside and is drawn next to its associated node.

Tool Command Language

Tcl is a shortened form of Tool Command Language. John Ousterhout of the University of California, Berkeley, designed it. It is a combination of a scripting

language and its own interpreter that gets embedded to the application, we develop with it.

Tcl was developed initially for Unix. It was then ported to Windows, DOS, OS/2, and Mac OS X. Tcl is much similar to other unix shell languages like Bourne Shell (Sh), the C Shell (csh), the Korn Shell (sh), and Perl.

It aims at providing the ability for programs to interact with other programs and also for acting as an embeddable interpreter. Even though the original aim was to enable programs to interact, you can find full-fledged applications written in Tcl/Tk.

Features of Tcl

The features of Tcl are as follows −

- Reduced development time.
- Powerful and simple user interface kit with integration of TK.
- Write once, run anywhere. It runs on Windows, Mac OS X, and almost on every Unix platform.
- Quite easy to get started for experienced programmers; since, the language is so simple that they can learn Tcl in a few hours or days.
- You can easily extend existing applications with Tcl. Also, it is possible to include Tcl in C, C++, or Java to Tcl or vice versa.
- Have a powerful set of networking functions.
- Finally, it's open source, free, and can be used for commercial applications without any limit.

Local Environment Setup

If you are willing to set up your environment for Tcl, you need the following two software applications available on your computer

- Text Editor
- Tcl Interpreter.

Text Editor

This will be used to type your program. Examples of a few text editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi. The name and version of a text editor can vary on different operating systems. For example, Notepad will be used on Windows, and vim or vi can be used on windows as well as Linux or UNIX.

The files you create with your text editor are called source files and contain program source code. The source files for Tcl programs are named with the extension ".tcl".

Before starting your programming, make sure you have one text editor in place and you have enough experience to write a computer program, save it in a file, build it, and finally execute it.

The Tcl Interpreter

It is just a small program that enables you to type Tcl commands and have them executed line by line. It stops the execution of a TCL file, in case, it encounters an error, unlike a compiler that executes fully.

Let's have a helloWorld.The TCL file is as follows. We will use this as the first program, we run on a platform you choose.

```
#!/usr/bin/tclsh

puts "Hello World!"
```

Installation on Linux

Most of the Linux operating systems come with Tcl inbuilt and you can get started right away in those systems. In case it's not available, you can use the following command to download and install Tcl-Tk.

```
$ yum install tcl tk
```

Now, we can build and run a Tcl file say helloWorld.tcl by switching to folder containing the file using 'cd' command and then execute the program using the following steps −

```
$ tclsh helloWorld.tcl
```

We can see the following output −

```
$ hello world
```

As you know, Tcl is a Tool command language, commands are the most vital part of the language. Tcl commands are built into the language with each having its predefined function. These commands form the reserved words of the language and cannot be used for another variable naming. The advantage of these Tcl commands is that you can define your implementation for any of these commands to replace the original built-in functionality. Each of the Tcl commands validates the input and it reduces the work of the interpreter. Tcl command is a list of words, with the first word representing the command to be executed. The next words represent the arguments. To group the words into a single argument, we enclose multiple words with "" or {}.

The syntax of Tcl command is as follows −

```
commandName argument1 argument2 ... argumentN
```

Let's see a simple example of Tcl command −

```
#!/usr/bin/tclsh
puts "Hello, world!"
```

When the above code is executed, it produces the following result −

```
Hello, world!
```

In the above code, ‘puts’ is the Tcl command and "Hello World" is the argument1. As said before, we have used "" to group two words.
Let's see another example of Tcl command with two arguments −

```
#!/usr/bin/tclsh
```

puts stdout "Hello, world!"

When the above code is executed, it produces the following result −

```
Hello, world!
```

In the above code, ‘puts’ is the Tcl command, ‘stdout’ is argument1, and "Hello World" is argument2. Here, stdout makes the program to print in the standard output device.

Network Simulator

Network Simulator (NS) is simply a discrete event-driven network simulation tool for studying the dynamic nature of communication networks. Network Simulator 2 (NS2) provides substantial support for the simulation of different protocols over wired and wireless networks. It provides a highly modular platform for wired and wireless simulations supporting different network elements, protocols, traffic, and routing types.

NS2 is a simulation package that supports several network protocols including **TCP, UDP, HTTP,** and **DHCP** and these can be modelled using this package. In addition, several kinds of network traffic types such as constant bit rate **(CBR)**, available bit rate **(ABR),** and variable bit rate (VBR) can be generated easily using this package. It is a very popular simulation package in academic environments.
NS2 has been developed using the **C++** programming language and **OTcl. OTcl** is a relatively new language that uses object-oriented aspects. It was developed at **MIT** as an object-oriented extension of the Tool command language **(Tcl).**

Features of NS2

1. It is a discrete event simulator for networking research.
2. It provides substantial support to simulate a bunch of protocols like TCP, FTP, UDP, https and DSR.
3. It simulates wired and wireless networks.
4. It is primarily Unix based.
5. Uses TCL as its scripting language.
6. Otcl: Object oriented support
7. Tclcl: C++ and otcl linkage
8. Discrete event scheduler

Basic Architecture

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up the simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TclCL

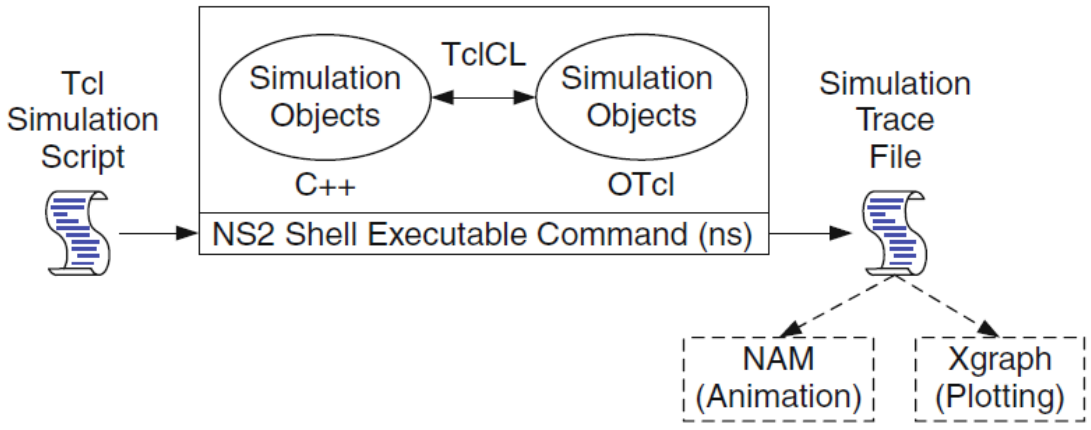


Fig no: 2 Basic architecture of NS.

MANET in NS2

Unlike infrastructure-based wireless networks, a mobile ad hoc network, or MANET, does not depend on fixed infrastructure for its network traffic. A MANET is an autonomous and short-lived association of a group of mobile nodes that communicate with each other via wireless links. A node can communicate directly with nodes that lie within its communication range. If a node wants to communicate with a node that is not directly in its communication range, it uses intermediate nodes as routers. From a simulation perspective, the primary component in designing a mobile advertising network is the mobility model, while other components include node configuration, random topology, and communication model. In the mobility model, the mobility of a node from one location to another can be enabled using the "saddest" keyword in the Tool Command Language (TCL) script. The specifications for the node's target location include the x, and y coordinates along with the velocity. Nodes are configured with the components of the channel, network interface, radio propagation model, medium access control (MAC), ado routing protocol, interface queue, link layer, topographic object, and antenna type. In a dynamic topology, the neighbours of each node vary according to the location of that particular node. Nodes in the ado network communicate using a communication model. Sample14.tcl illustrates the design of a mobile ad network that consists of 3 mobile nodes. The movements of mobile nodes are limited to an area of 500mX500m with a pause time of 3s. Data transfer is established between nodes using a UDP agent and CBR traffic. These intermediate routers forward packets generated by other nodes to their destination.

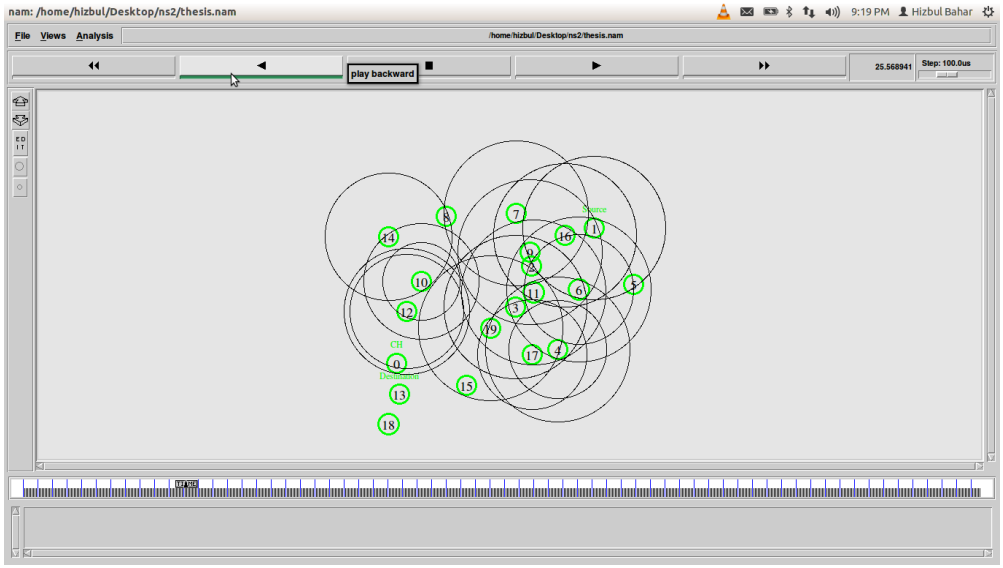


Fig no: 3 Manet in NS2

LINUX (Ubuntu)

Overview

Ubuntu is by far the most popular Linux distribution for running web servers from the sites they analyze, "used by 47.3% of all sites that use Linux", and Ubuntu alone powers more sites than Microsoft Windows, which powers 28.2% of all sites. , or 39% of the share held by Unix (which includes Linux and therefore Ubuntu). All Linux/Unix distributions have a combined performance of more than twice the number of hosts than Windows for websites based on W3Techs numbers. Ubuntu and Debian alone (on which Ubuntu is based, with the same package manager and therefore managed in the same way) account for 65% of all Linux distributions for web services; Ubuntu usage surpassed Debian (for such server usage) in May 2016. Ubuntu is the most popular Linux distribution among the top 1000 sites, gaining about 500 of the top 10 million sites daily.

Features of Ubuntu

The default installation of Ubuntu includes a wide variety of software that includes Libre Office, Firefox, Thunderbird, Transmission and a few light games such as Sudoku and Chess.

- Many other software packages are accessible from Ubuntu's built-in software (formerly Ubuntu Software Center) as well as from other APT-based package management tools.
- Many other software packages that are no longer installed by default, such as Evolution, GIMP, Pidgin, and Synaptic, are still available in the repositories and 5 can be installed using the main tool or any other APTbased package management tool.
- There are also snap and flatpak packages for various distributions, both of which allow software such as Microsoft software to be installed on most major Linux operating systems (such as any currently supported version of Ubuntu and Fedora).

Just like Windows, iOS, and Mac OS, Linux is an operating system. One of the most popular platforms on the planet, Android, is powered by the Linux operating system. An operating system is a software that manages all of the hardware resources associated with your desktop or laptop. To put it simply, the operating system manages the communication between your software and your hardware. Without the operating system (OS), the software wouldn’t function.

The Linux operating system comprises several different pieces:

1. **Bootloader –** The software that manages the boot process of your computer. For most users, this will simply be a splash screen that pops up and eventually goes away to boot into the operating system.
2. **Kernel –** This is the one piece of the whole that is actually called 'Linux'. The kernel is the core of the system and manages the CPU, memory, and peripheral devices. The kernel is the lowest level of the OS.
3. **Init system –** This is a sub-system that bootstraps the user space and is charged with controlling daemons. One of the most widely used init systems is systemd, which also happens to be one of the most controversial. It is the init system that manages the boot process, once the initial booting is handed over from the bootloader (i.e., GRUB or GRand Unified Bootloader).
4. **Daemons –** These are background services (printing, sound, scheduling, etc.) that either startup during boot or after you log into the desktop.
5. **Graphical server –** This is the subsystem that displays the graphics on your monitor. It is commonly referred to as the X server or just X.
6. **Desktop environment –** This is the piece that the users actually interact with. There are many desktop environments to choose from (GNOME, Cinnamon, Mate, Pantheon, Enlightenment, KDE, Xfce, etc.). Each desktop environment includes built-in applications (such as file managers, configuration tools, web browsers, and games).
7. **Applications –** Desktop environments do not offer the full array of apps. Just like Windows and macOS, Linux offers thousands upon thousands of high-quality software titles that can be easily found and installed. Most modern Linux distributions include App Store-like tools that centralise and simplify application installation. For example, Ubuntu Linux has the Ubuntu Software Center which allows you to quickly search among the thousands of apps and install them from one centralised location.

CHAPTER - 5

SYSTEM IMPLEMENTATION

5.1. Module List

1. NAM animator and Network Simulator Module
2. Acknowledgement request Module
3. DSR Module
4. Dumping Black node Module
5. Destination Module n

5.2. Experimental Results And Evaluation

In this project we enter the commands in the terminal to access the network simulator to initiate the process.The network simulator starts with initially the nodes set by the user ,here 35 nodes are used and time latency is set to 4-5ms.the nam animator checks for the miscellaneous nodes and the packet drop takes place.the energy level of the nodes varies and drop where it can identified through changing colour of the nodes.The graphs would be executed as a result after the execution of the process.

5.3.1. End-To-End Delay

End-to-end delay in networking refers to the amount of time it takes for a packet to be transmitted from the source device to the destination device. It includes the time it takes for the packet to travel over the network, as well as any processing time that occurs at each intermediate device along the way, such as routers or switches.End-to-end delay can be affected by a variety of factors, such as network congestion, network topology, distance between devices, and

processing delays at intermediate devices. To calculate the end-to-end delay in Mobile Ad hoc Networks (MANETs) in NS2, you can use the trace files generated during the simulation. The end-to-end delay is the time taken for a packet to travel from the source to the destination in the network. The end-to-end delay in AODV can be affected by several factors, including: 1. Route discovery time: AODV uses a reactive routing protocol, which means that routes are only established when needed. When a node wants to send a packet to a destination, it initiates a route discovery process to find a route to the destination. The time it takes to discover a route can affect the end-to-end delay. 2. Route maintenance time: AODV also has a mechanism to maintain the established routes. If a route breaks due to a node moving out of range or due to network congestion, AODV will try to find an alternate route to the destination. This process can also affect the end-to-end delay. 3. Transmission time: The time it takes for a packet to be transmitted from one node to another can also affect the end-to-end delay. This time 20 includes the time it takes for the packet to be sent, received, and processed by the nodes in the network. 4. Congestion: If the network is congested, the end-to-end delay can increase significantly. Congestion occurs when there is more traffic in the network than the network can handle, leading to packet loss, delays, and other performance issues. In general, the end-to-end delay in AODV can vary depending on the size and topology of the network, the traffic load, and the mobility of the nodes. However, AODV is designed to provide efficient and reliable routing in MANETs, and the end-to-end delay is typically within acceptable limits for most applications.

5.2.2. Latency Rate

Latency rate in a network simulator refers to the amount of time it takes for a packet to be transmitted from the source node to the destination node in a network simulation. It is an important metric for evaluating the performance of network protocols and algorithms, particularly for real-time applications that require low latency, such as video conferencing, online gamingLatency rate in network simulator is an important metric for evaluating the quality of service (QoS) of a network, and is often used as a key performance indicator (KPI) for network performance.In the context of Mobile Ad hoc Networks (MANETs), latency rate can be affected by various factors such as network topology, mobility pattern, traffic load, routing protocol, and network congestion. 22 Therefore, it is important to carefully design and configure your MANET simulation scenario to accurately reflect your research objectives.To calculate the latency rate in a MANET simulation, you can use the trace files generated during the simulation. You can analyze the trace files using a tool like awk or grep to extract information about packet delays and calculate the latency rate.

5.2.3. Data Transmission Between Nodes

Networks can be physical or logical. A physical computer network is a real network comprising the cable and devices that send data back and forth. Logical networks are software representations of a physical network. They are built on top of a physical network.Computer networks aim to share information and resources among multiple digital devices. The internet is an example of a computer network. It is made up of many smaller computer networks. Computer networks make things like video streaming, social networks and cloud networks possible. A node is a point of intersection/connection within a data communication network. In an environment

where all devices are accessible through the network, these devices are all considered nodes. The individual definition of each node depends on the type of network it refers to.Once the route is established, the source node can start transmitting data to the destination node. The data is typically divided into packets, which are transmitted from the source node to the destination node hop by hop, where each hop represents a wireless link between two adjacent nodes. When a node receives a packet, it checks the destination address in the packet header to determine whether it is the intended recipient. If the node is the destination node, it receives the packet and sends an acknowledgement (ACK) back to the source node. If the node is not the destination node, it forwards the packet to the next hop in the route towards the destination. 23 The process of data transmission continues until the entire data packet reaches the destination node. During the transmission, each node in the route may retransmit the packet if it does not receive an ACK from the next hop within a certain timeout period, or if it detects errors in the packet. In summary, data transmission between nodes in a MANET involves the establishment of a route between the source and destination nodes, division of data into packets, and hop-by-hop transmission of the packets through wireless links until they reach the destination node.

CHAPTER - 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1. Conclusion

We presented a new technique for detecting black hole attacks in wireless sensor networks in this project. Our proposed system implements the DSR protocol as a routing protocol. It does not require additional hardware, node location, or send any information to the base station. No extra communication is used for the purpose of detecting and isolating black hole attacks. The DSR protocol is a more efficient routing protocol generally, as it uses cache memory to save all the possible routes that can be enhanced for communication purposes. Further, the use of GAN network is applied to mitigate the security by identifying the fake ID data of the miscellaneous nodes. Proposed techniques are also applicable when black hole nodes advertise high-quality links, strong transmitted power, etc. In such cases, too, our system manages to detect and isolate such attacks in wireless sensor networks.

6.2. Future Scope

The proposed approach can be further improved by incorporating machine learning and artificial intelligence techniques to detect and prevent Sybil attacks. Furthermore, the proposed approach can be extended to other routing protocols used in MANETs.

CHAPTER - 7

APPENDIX

7.1. Source Code

dsr.tcl

```
#Fixing the coordinate of simulation area
set val(x) 500
set val(y) 500
# Define options
set val(chan)    Channel/WirelessChannel ;
# channel type
set val(prop)    Propagation/TwoRayGround ;
# radio-propagation model
set val(netif)   Phy/WirelessPhy ;
# network interface type
set val(mac)     Mac/802_11 ;
# MAC type
set val(ifq)     Queue/DropTail/PriQueue ;
# interface queue type
set val(ll)  LL ;
# link layer type
set val(ant)
Antenna/OmniAntenna ;
# antenna model
set val(ifqlen) 50 ;
```

```
# max packet in ifq
set val(nn)  15 ;
# number of mobilenodes
set val(rp)  DSR ;
# routing protocol
set val(x)   500 ;
# X dimension of topography
set val(y)   500 ;
# Y dimension of topography
set val(stop)    10.0 ;
# time of simulation end

# Simulator Instance Creation
set ns [new Simulator]

#Creating nam and trace file:
set tracefd       [open dsr.tr w]
set namtrace      [open dsr.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
```

```
# general operational descriptor- storing the hop
details in the network
create-god $val(nn)

# configure the nodes
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON

# Node Creation

for {set i 0} {$i < 15} {incr i} {

set node_($i) [$ns node]
$node_($i) color black
```

```
}


for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}

for {set i 0} {$i < 3 } { incr i } {
           set bb [expr $i*10]
           set cc [expr $i*60]
                  $node_($i) set X_ $bb
                  $node_($i) set Y_ $cc
           #$node_($i) random-motion 0

             }

for {set i 5} {$i < 10 } { incr i } {
           set bb [expr $i*60]
           set cc [expr ($i-5)*60]
                  $node_($i) set X_ $bb
                  $node_($i) set Y_ $cc
           #$node_($i) random-motion 0

             }

for {set i 10} {$i < 12 } { incr i } {
```

```
            set bb [expr ($i-3)*60]
            set cc [expr ($i-10)*60]
                   $node_($i) set X_ $bb
                   $node_($i) set Y_ $cc
            #$node_($i) random-motion 0

              }

for {set i 13} {$i < 15 } { incr i } {
            set bb [expr ($i-2)*25]
            set cc [expr ($i-10)*60]
                   $node_($i) set X_ $bb
                   $node_($i) set Y_ $cc
            #$node_($i) random-motion 0

              }

for {set i 3} {$i < 5 } { incr i } {
            set bb [expr $i]
            set cc [expr ($i+1)*60]
                   $node_($i) set X_ $bb
                   $node_($i) set Y_ $cc
            #$node_($i) random-motion 0

              }

$node_(12) set X_ 150
```

```
$node_(12) set Y_ 80

$ns at 4.5 "malicious_find"
proc malicious_find {} {
    global node_
            $node_(5) label "sybil attacker_found"
            $node_(1) color red
            $node_(1) label "sybil attacker node"
            $node_(3) color red
            $node_(3) label "sybil attacker node"
            $node_(13) color red
            $node_(13) label "sybil attacker node"
            $node_(7) color red
            $node_(7) label "sybil attacker node"
            $node_(9) color red
            $node_(9) label "sybil attacker node"
            #$ns at 6.0 "$node_($b) color red"
            #$ns at 6.0
}

$ns at 0.1 "source_find"
proc source_find {} {
    global node_
    $node_(5) color green
    $node_(5) label "SOURCE"
}
```

```
$ns at 1.0 "ack_request"
proc ack_request {} {
    global node_
    $node_(5) label "ACK requested"
}

$ns at 2.0 "ack_receive"
proc ack_receive {} {
    global node_
    $node_(5) label "ACK received"
}

$ns at 3.0 "ack_receive"
proc ack_receive {} {
    global node_
    $node_(5) label "finding_destination"
}

$ns at 4.0 "destination"
proc ack_receive {} {
    global node_
    $node_(5) label "destination_found"
    $node_(11) label "DESTINATION"
    $node_(11) color blue
}

$ns at 5.0 "data_send_receive"
```

```
proc data_send_receive {} {
    global node_
    $node_(5) label "data_sending"
    $node_(11) label "data_receiving"
}


$ns at 6.0 "trying_to_connect"
proc trying_to_connect {} {
    global node_
            $node_(1) color darkmagenta
            $node_(1) label "fake_request"
            $node_(3) color darkmagenta
            $node_(3) label "fake_request"
            $node_(13) color darkmagenta
            $node_(13) label "fake_request"
            $node_(7) color darkmagenta
            $node_(7) label "fake_request"
            $node_(9) color darkmagenta
            $node_(9) label "fake_request"
            #$node_($b) label "trying_to_connect"
            #$ns at 6.0 "$node_($b) color red"
            #$ns at 6.0
}


$ns at 7.0 "ARMKEY_send"
```

```
proc ARMKEY_send {} {
    global node_
    $node_(5) label "GAN_applied"
    $node_(11) label "attack_mitigated"
}


$ns at 9.5 "data_transferred"
proc data_transferred {} {
    global node_
    $node_(5) label "data_sent"
    $node_(11) label "data_received"
}

# dynamic destination setting procedure..
$ns at 0.0 "destination"
proc destination {} {
      global ns val node_
      set time 1.0
      set now [$ns now]
      for {set i 0} {$i<$val(nn)} {incr i} {
            set xx [expr rand()*500]
            set yy [expr rand()*400]
            $ns at $now "$node_($i) setdest $xx $yy
12.0"
      }
      $ns at [expr $now+$time] "destination"
```

```
}

#*****************************Defining Communication
Between node0 and all nodes
***************************

#Setup a TCP connection
# Defining a transport agent for sending
set tcp [new Agent/TCP]
#set udp [new Agent/UDP]

$tcp set class_ 2
# Attaching transport agent to sender node
$ns attach-agent $node_(5) $tcp
#$ns attach-agent $node_($i) $udp

# Defining a transport agent for receiving
set sink [new Agent/TCPSink]
#set null [new Agent/Null]

# Attaching transport agent to receiver node
$ns attach-agent $node_(11) $sink
#$ns attach-agent $node_(0) $null

#Connecting sending and receiving transport agents
$ns connect $tcp $sink
#$ns connect $udp $null
```

```
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Defining Application instance
#set cbr [new Application/Traffic/CBR]

# Attaching transport agent to application agent
#$cbr attach-agent $udp

#Packet size in bytes and interval in seconds
definition
#$cbr set packetSize_ 512
#$cbr set interval_ 0.1

# data packet generation starting time
#$ns at 1.0 "$cbr start"
$ns at 5.0 "$ftp start"

$ns at 1.0 "$ns trace-annotate \"ack_request send\""
$ns at 2.0 "$ns trace-annotate \"ack_ request receiver
\""
$ns at 3.0 "$ns trace-annotate \"finding_destination\""
```

```
$ns at 4.0 "$ns trace-annotate \"Destination found\""
$ns at 5.0 "$ns trace-annotate \"Data send receive\""
$ns at 6.0 "$ns trace-annotate \"trying to connect the
nodes\""
$ns at 6.1 "$ns trace-annotate \"sybil attacker nodes
were found as n1 n3 n5 n7 n9 n13\""
$ns at 7.0 "$ns trace-annotate \" GAN key send
#5678966789JHG6\""
$ns at 8.8 "$ns trace-annotate \"GAN key received
applied #5678966789JHG6 to transfer between in source
to destination\""
$ns at 9.5 "$ns trace-annotate \" Data_received\""

# data packet generation ending time
#$ns at 6.0 "$cbr stop"
#$ns at 6.0 "$ftp stop"

#stop procedure..
$ns at $val(stop) "stop"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
exec nam dsr.nam &
}
$ns run
```

7.2. Output Screenshots

The secure transmission date from source to destination is completed without any interruption and reduces packet drop and time delay. The number of nodes is present in the NAM animator which is located in the initial position and starts moving the node wirelessly.

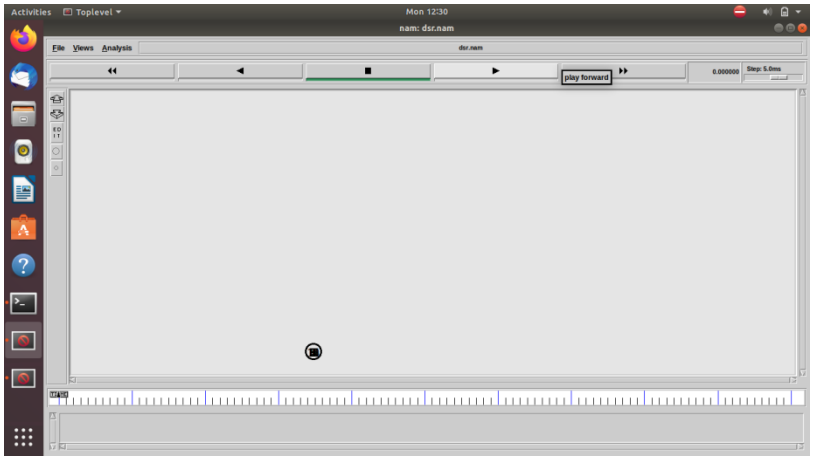


Fig no: 4 Initial node formation in Nam animator.

The nodes are moving randomly and assign node name weather check the source node are presented in TCL code , the node number assign 5th node and send that ask request to destination.

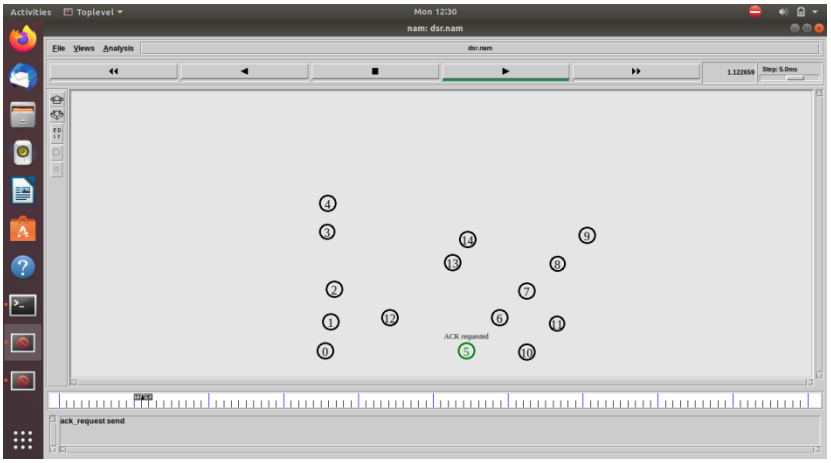


Fig no: 5 Random moving node

Here the source node sends to ack request to destination which will conform to the destination and in predefined code are user defined nodes.

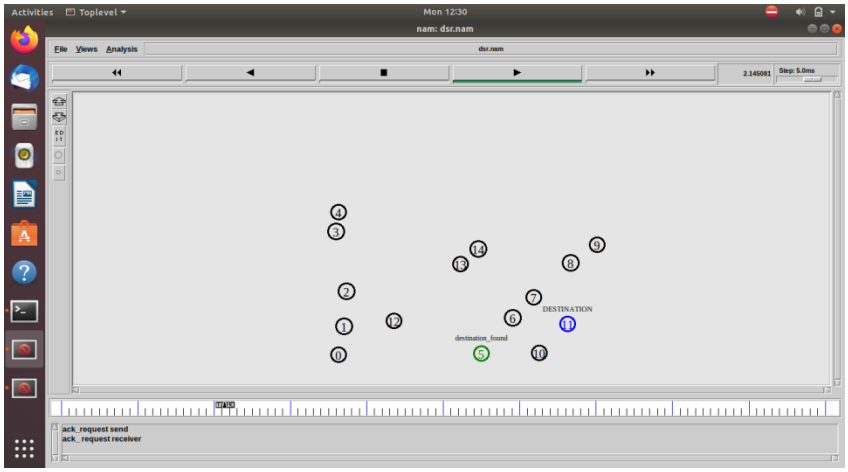


Fig no: 6 Source and destination unitization.

The nodes are randomly and the source send the data to destination the here some sybil attacker nodes are formed to avoid the data transmission from source to destination in MANET (Mobile -Ad hoc network).

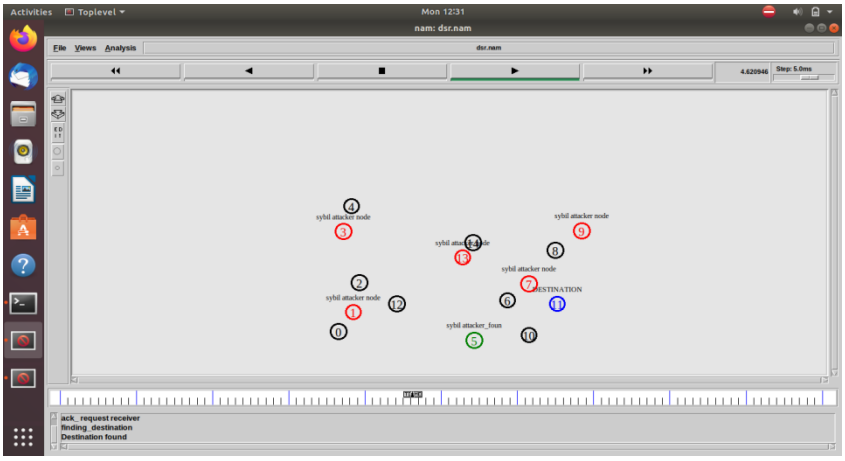


Fig no: 7 Sybil attacker node formed

The fake nodes are sending data to the source which acts as a destination node. Here the source node sends some GAW key which contains 16 digital passkeys which are used to confirm the destination without any interruption.

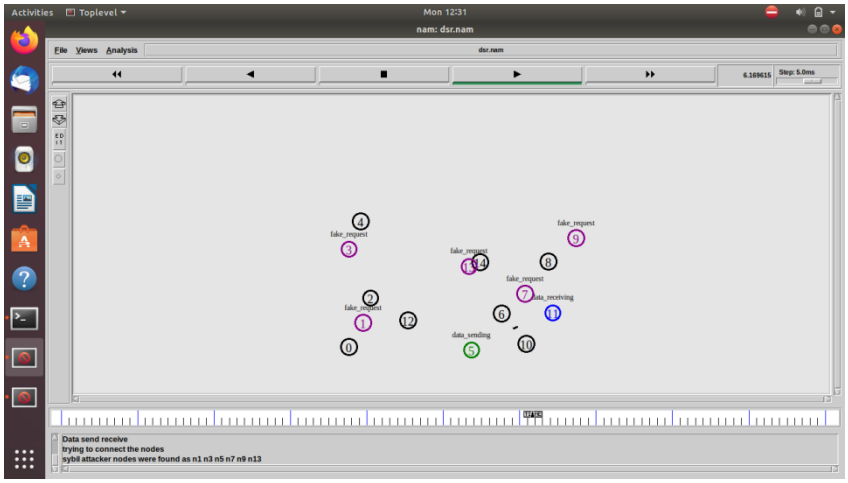


Fig no: 8 Fake request to source node.

Finally, the nodes conform to the destination and transmit the data without time delay and also reduce packet drop while transferring the data.

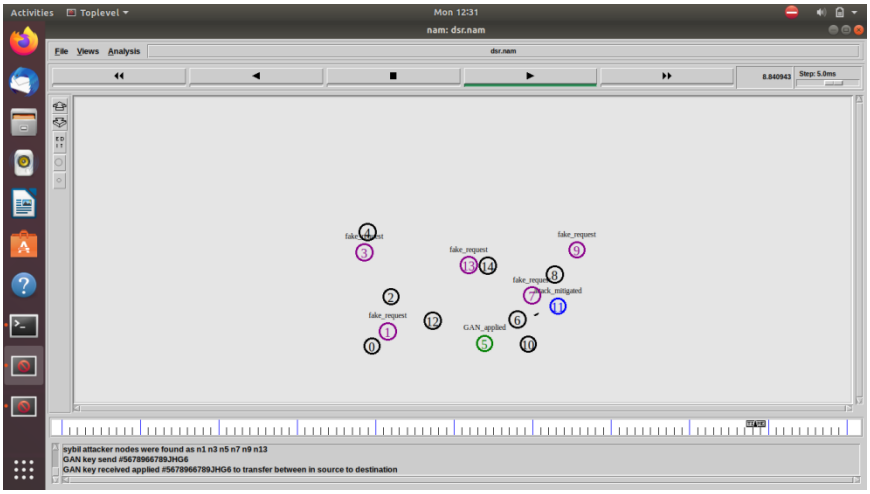


Fig no: 9 data sending and data receiving nodes.

CHAPTER - 8

REFERENCE

[1] A. Mouiz, A. Badri, Evaluating the Power Consumption of Routing Protocols for Wireless Sensor Networks under Different Metric Parameters, IEEE paper 2020.

[2] S.H. Park, S. Cho, and J.R. Lee, Energy-Efficient Probabilistic Routing Algorithm for Internet of Things. In 2019 4th International Conference on Recent Trend on Electronics, Information, Communication & Technology, IEEE.

[3] S. S. Roy, D. Puthal, S. Sharma.Building a Sustainable Internet of Things: Energy Efficient Routing Using Low-Power Sensors Will Meet the Need, IEEE 2021.

[4] S. Gupta and V. Grover, "Survey of intrusion detection techniques in LEACH",2019.

[5] S. Ramachandran and V. Shanmugam, "An approach to secure leach using tesla-based certificate",2021.

[6]. Martins, D., Guyen net, H.: Wireless Sensor Network Attacks and Security Mechanisms: A Short Survey. 2010 13th International Conference on Network-Based Information Systems. pp. 313–320. IEEE (2020).

[7]. Pandey, A., Tripathi, R.C. A Survey on Wireless Sensor Networks Security. Int. J. Comput. Appl. IJCA. 3, 43–49 (2019).

[8]. Ngai, E.C.H., Liu, J., Lyu, M.R.: An efficient intruder detection algorithm against sinkhole attacks in wireless sensor networks. Comput. Commun. 30, 2353–2364 (2021).

[9].Tumrongwittayapak, C., Varakulsiripunth, R.: Detecting Sinkhole attacks in wireless sensor networks. ICROS-SICE International Joint Conference. pp. 1966–1971 (2019).

[10]. Tumrongwittayapak, C., Varakulsiripunth, R.: Detecting sinkhole attack and selective forwarding attack in wireless sensor networks. 2009 7th International Conference on Information, Communications and Signal Processing (ICICS). pp. 1–5. IEEE (2020).

[11]. Coppolino, L., D'Antonio, S., Romano, L., Spagnuolo, G.: An Intrusion Detection System for Critical Information Infrastructures using Wireless Sensor Network technologies. 2010 5th International Conference on Critical Infrastructure (CRIS). pp. 1–8. IEEE (2019).

[12]. Krontiris, I., Dimitriou, T., Giannetsos, T., Mpasoukos, M.: Intrusion detection of sinkhole attacks in wireless sensor networks. In: Kutyłowski, M., Cichoń, J., and Kubiak, P. (eds.) ALGOSENSORS'07 Proceedings of the 3rd international conference on Algorithmic aspects of wireless sensor networks. pp. 150–161. Springer-Verlag, Wroclaw, Poland (2020).