Chapter – 4

IMPLEMENTATION OF PROPOSED ROUTING SECURITY SCHEMES FOR MANETS

CHAPTER -4

Chapter - 1. Implementation of Proposed Routing Security Schemes for MANETs

CHAPTER-4

IMPLEMENTATION OF PROPOSED ROUTING SECURITY SCHEMES FOR MANETS

As a part of this investigation three security frameworks are proposed as an extension to the traditional AODV routing protocol with an objective to overcome the limitations of the on hand security methods using an intrusion detection system and cryptographic security approaches purely in the context of MANETs. The new security extensions of the AODV routing protocol are named as a Game Theoretic Approach to Secure AODV (GTASA), Security enabled AODV (SEA) and Elliptic Curve Cryptography Enabled AODV (ECCEA). The GTASA and SEA routing protocols are implemented using the intrusion detection system schemes and ECCEA protocol is implemented using key authorization schemes of Cryptography, specifically to compete with blackhole attacks in the context of MANETs.

This chapter presents the implementation of new security schemes in the popular network simulator NS2 [69], [70], [71]. The graceful design of NS2 is its open source property and modifiability that facilitate us to adapt with the new protocols to extend the simulator. Let us first have the discussion on the NS2 simulator and then the discussion is extended to the implementation part of the new security routing approaches.

4.1. THE NETWORK SIMULATOR (NS-2)

The Network Simulator (V-2.35), popularly known as NS2, is basically an event driven simulation tool. It is very much helpful in investigating the vibrant environment of communication systems. The simulation of wireless as well as wired network functions and protocols can be completed using NS2. It allows users to specify such network protocols, and facilitate the way to simulate their equivalent behaviors. Since its invention in 1989, it has gained so much popularity among the networking research community because of its flexibility and modular nature. After then, several revisions have manifested the mounting maturity of the tool, thanks to its users for their extensive contributions in the field. The prominent users of NS2 include Cornell University, the University of California, National Science Foundation (NSF) and finally the group of researchers who are continuously working to keep NS2 flexible and strong.

4.1.1. The Fundamental Architecture of NS2

The fundamental architecture of NS2 is illustrated in fig 4.1. NS2 comprises of two important programming languages: C++ and OTcl (Object oriented Tool Command Language). The C++ as a back end defines the in-house mechanism of the simulation objects and OTcl sets up a simulation as a front end by assembling and configuring the objects, also scheduling discrete events. The TclCL is used to link together the C++ and the OTcl.

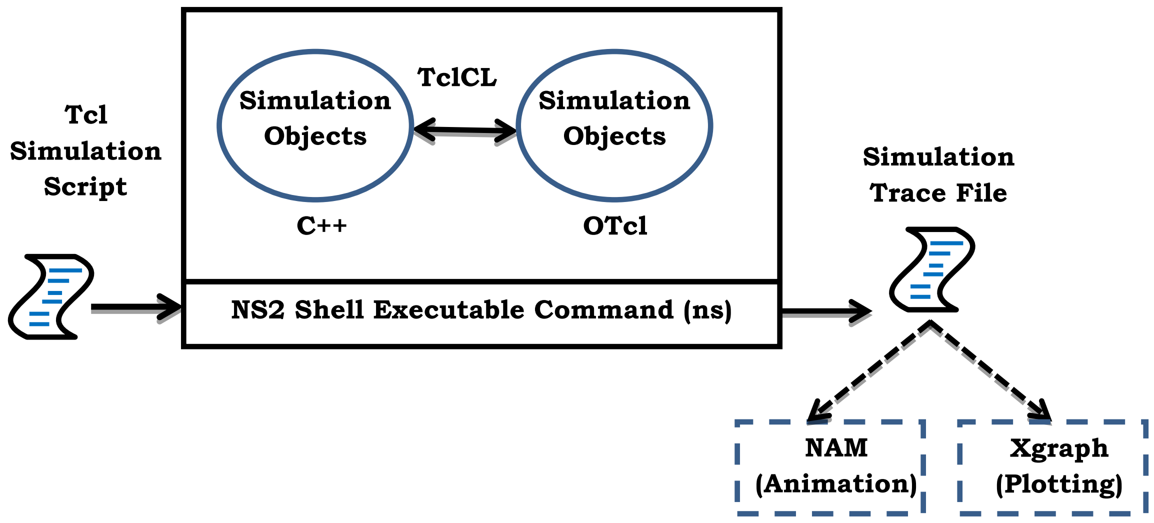


Fig.4.1: Basic architecture of NS

The NS2 simulator provides numerous built in C++ objects. The simulation can be set up using a Tcl simulation script with the help of these C++ objects. But, for the advanced users these objects may be inadequate. They necessitate scripting their own C++ objects, and put together all these objects using an interface configuration of OTcl. NS2 outputs both animation based and text based simulation results after the simulation is over. For the interpretation of these results the graphical and interactive tools such as Gnuplot [72] and Network Animator (NAM) [73] are used. The users can take out a suitable subset of text based data and modernize it to a more reasonable appearance to investigate a precise performance of the network.

4.1.2. Time Driven Simulation

In this type of simulation, the simulation clock is advanced exactly by a predetermined period of *Δ* time slots. After the advancement of every clock the simulation always looks for the events that may have occurred during this predetermined interval. If any, such events are handled as if they occurred at the end of this interval. The fundamental idea behind time advancement in this simulation approach is illustrated figure 4.2. The bent arrows stand for such time advances, and a, b, and c spot the happenings of particular events. No event has occurred in the first interval, while the second interval contains event "a", which is treated once the interval is over. The major problem with this class of simulation is described in the fifth interval, where numerous events b and c are assumed to occur exactly at the end of the interval (at 5*Δ*). To solve this problem a method that can decide which event must be treated first is needed. One such method to come out of this difficulty is to decrease the interval simulation so that each interval may possibly contain only one event. However, this new method of simulation puts a lot of computational overhead on the simulator. Hence this type of simulation is not suitable for network models whose events might be likely to take place over an arbitrary period of time.

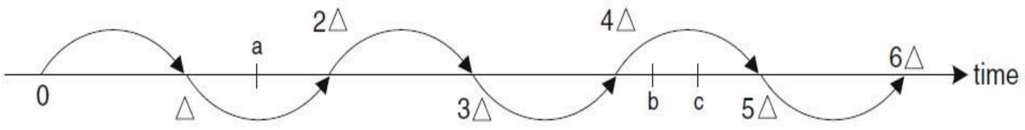


Fig 4.2: The illustration of time driven simulation

4.1.3. Event Driven Simulation

As the name suggests this class of simulation [75], [76], is initiated and run by a set of events. The list of all planned events is usually stored and rationalized throughout the simulation process. In fact the critical loop of this simulation procedure is the sequencing of all the events from the list and treat them one after the other until the list is empty.

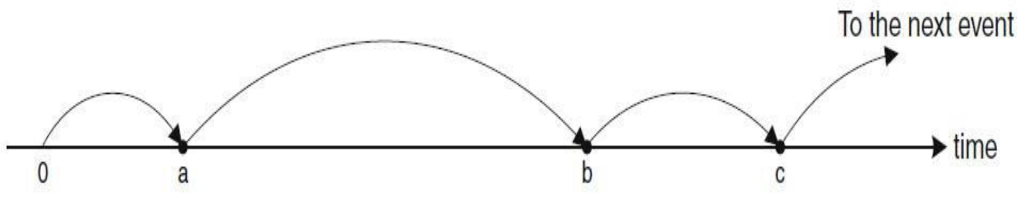


Fig 4.3: Event driven simulation

4.1.4. Installation of NS2

NS2 is an open tool and can be obtained from the web at free of cost. It is so portable to run on different platforms, including Linux or UNIX, Mac systems and Windows. As it is developed in the UNIX environment, without any surprise, it has the smoothest traverse there, and so does its installation also. One can obtain NS2 source files in two forms: the component-wise and the all-in-one suite. The users will get all the necessary components along with few optional components with the all-in-one pack.

The present all-in-one suite consists of NS release 2.35, OTcl release 1.14, Tcl/Tk release 8.5.8, and TclCL release 1.20 as the main components and NAM release 1.14, Zlib version 1.2.3 and Xgraph version 12.1 as the optional components.

4.1.5. The Installation of an All-in-one NS2 Pack on UNIX Related Environment

This pack can easily be installed on the UNIX related environment like Ubuntu 12.04 by simply running the “install” command and following the instructions afterwards. The only prerequisite is a system installed with a C++ compiler. The commands **Shell>./install** and **Shell>./validate** illustrates how this pack of NS2 can be installed and validated, respectively. The validation process involves the running of operational scripts a numerous times to confirm the required functionalities of the installed components.

The following steps clearly describe the complete installation procedure of NS2.

Open the option dash home - terminal.

1. Sudo apt-get update.
2. Sudo build-essential.
3. Sudo apt-get install build-essential.
4. Sudo apt-get libxll-dev
5. Sudo apt-get install libxll-dev
6. Sudo apt-get libxmu-dev
7. Sudo apt-get build essential autoconf apt-get auto make libxmu-dev.
8. ./install
9. ./validate

4.1.6. The Installation of an All-in-one NS2 Pack on Windows Related Environment

A bit of tweaking is required to install and run NS2 on operating systems based on windows. Importantly, the initiative is to craft a windows environment to emulate the functionality of the UNIX type of environment.

The Cygwin is one such admired program that can carry out this work. After the installation of Cygwin the same installation procedure as that of UNIX environment can be used. To make the installation simple, it is advisable to use the all-in-one suit of NS2. It is worth enough to note that the default Cygwin does not install all the required packages to run NS2. It is up to the user to install the necessary packages manually. Cygwin is an open source and accessible online. Different editions are available and may install diverse default packages.

4.1.7. Scenario Generation

Scenario generation can be done using Setdest command. The different input parameters of setdest command are as follows.

“./setdest -n <No of nodes> -p <pause time> -s <Maxspeed> -t <simtime> -x <max x> -y <max y> Scenario file>”

- -n - Number of generating nodes in the scenario will be 0 to n-1
- -p – Pause time
- -s – The maximum speed assigned to the mobile nodes
- -t – Total simulation time
- -x - Space along x direction
- -y – Space along y direction

• After running the command a scenario will be generated. Pipe the scenario in the file as described below.

Example:/setdest -p 1 -s 10 -t 100 -P 1 -x 700 -y 700 > scen- exp1

For more details one can go through the in-built source code of setdest.cc in NS2.

4.1.8. CBR Traffic Generation

This can be done with the help of cbrgen.tcl executing file with “ns” command. Open the terminal and run the command “ns” cbrgen.tcl [-type udp or tcp] [-nn number of nodes] [-seed value of seed] [-mc number of mobile connections] [-rate rate] in cmu-scen-gen directory of NS2.

- -type - Traffic type (udp or tcp)
- -nn - The number of mobile nodes (0 to nn)
- -seed - Random variable generation which is used to create random scenarios
- -mc - The maximum number of connections;
- -rate - It is the inverse of the packet transmission interval

After running the command an udp or tcp [76], [77] traffic will be generated. Then pipe the traffic in file as illustrated below.

Example:

```
“ns cbrgen.tcl -type udp  -nn 20  -seed 2 -mc  5 -rate .50  > cbr-exp1”
```

For more details one can go through the in-built source code of cbrgen.tcl in cmu-scen-gen directory of NS2.

4.1.9. The Concept of NS2 Simulation

The simulation process of NS2 [78], [79], [80] comprises two important stages. They are network configuration and simulation stages.

Network Configuration Stage:

The NS2 constructs a network and generates an initial chain of events that comprises events which are planned to take place at specific times in this stage of the simulation. These types of events are referred as *at-event*. In Tcl simulation every line prior to the execution of an instant procedure runs {} of any simulation object is related to this stage.

Simulation Stage:

This stage includes only one particular line that calls the instance procedure run {} of the simulator object. Traditionally, this single line contributes too much of the simulation. In this stage, the simulator moves all along the sequence of events and runs each event in a sequential order. At this juncture, the instant procedure run {} starts the simulation by *dispatching* the events in sequence. Upon dispatch an event, the simulator moves downward the sequence and attempts to dispatch the next event. The same procedure repeats until the last event of instance procedure halt {} is dispatched that suggests the final part of the simulation.

4.2. SIMULATION ENVIRONMENT

As a part of this investigation, three new routing protocols called GTASA (Game Theoretic Approach to Secure AODV), SEA (Security Enabled AODV) and ECCEA (Elliptic Curve Cryptography Enabled AODV) are introduced as the security extensions to the conventional AODV routing protocol to compete with the blackhole attacks in the context of MANETs. The crisis is investigated by the way of gathering data from obtainable resources and by the way of optimal simulation which gives proper results. In addition, the simulation results are appropriately analyzed to make the decisions on their basis. The fashionable discrete event network simulator NS2 (V 2.35) is used to carry the simulations in Ubuntu-12.04 operating environment. The NS2 permits the users to build their own routing protocols as per the requirement and to compare its performance with the on hand protocols. To assess the performance of a protocol in MANET [81], [82], [83], [84] it is required to investigate it under realistic conditions, particularly counting the movement of mobile nodes. The creation of mobility models and traffic are required for the performance evaluation. The following parameters as listed in table 4.1 are used to perform the simulation.

Parameters	Value
Simulator	Ns-2.35
Data packet size	512 byte
Simulation time	100 sec
Environment size	700 x 700
Number of nodes	Ranging between 20 to 100
Transmission range	250m
Observation parameters	PDR, Delay, Throughput, NRL
Traffic Type	CBR Traffic
Mobility	5 & 10 m/s
Blackhole Nodes	1 & 3
Seed Values	1, 2, 3, 4 & 5
Pause time	0 s
Routing Protocols	AODV, GTASA, SEA, and ECCEA

Table 4.1: Simulation parameters

4.2.1. System configuration

A computer with the following configurations is used to run the simulation and to analyze the facts generated by the NS.

Processor	1.60 GHz CORE DUO 1GB x2 cache
Hard disk	512GB
RAM Memory	4GB
Operating system	UBUNTU 12.04

Table 4.2 System configuration

4.2.2. The Mobility Models

There exist a wide variety of mobility models proposed by Manzoni and Sanchez [85], [86], [87]. In the present investigation the Random Way Point (RWM) mobility model is used during the simulation. The speed variation of mobile nodes over time and their movement can be described using the mobility models during the simulation of a routing protocol. The RWM model proposed by Johnson and Maltz is the popular mobility model that is widely used to implement and examine the simulation of routing protocols due to its availability and simplicity. The every mobile node waits for some fixed time interval called pause time and randomly selects one location at the starting point of the simulation. Till the expiry of pause time a mobile node opts a new arbitrary destination consequent for staying at its previous position. A mobile node travels across the region at an arbitrary speed spread uniformly between the minimum and maximum speeds of the nodes. Until the simulation is finished the process of opting arbitrary destination at arbitrary speed is recurred again and again. Hence there is no constraint in selecting the speed, direction and destination of a node irrespective of the one hop nodes.

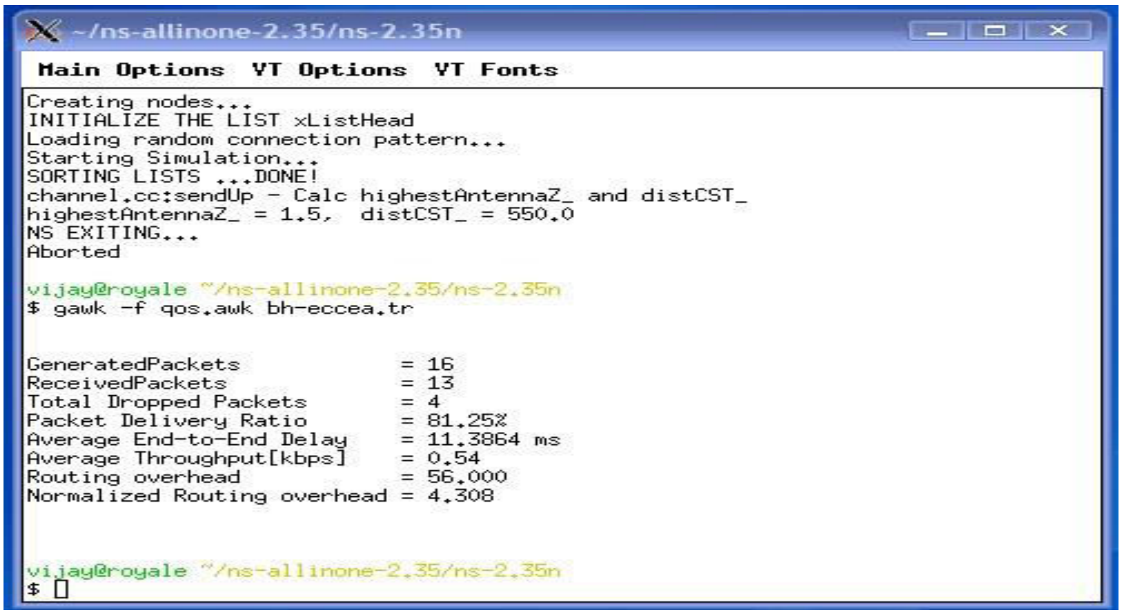
4.2.3. The Way to Analysis Simulation Results

This section presents running process, NAM simulation, Trace file analysis of the executions and GNU plot graphs. The traffic and connection patterns of MANET are created using different number of nodes, the seed value and node mobility's as follows.

Nodes	: 10, 20, 30, 40, 50, 60, 70, 80, 90,100
Seed	: 1, 2, 3, 4, 5
Mobility	: 5, 10

4.2.4. Simulation Running Procedure

Open the terminal and change the directory to the location where the program is stored. Type command name as "ns programname.tcl", for instance ns blackaodv30-3-1-5.tcl and press enter as shown in the figure below.



```
Creating nodes...
INITIALIZE THE LIST xListHead
Loading random connection pattern...
Starting Simulation...
SORTING LISTS ...DONE!
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
NS EXITING...
Aborted

vijay@royale ~/ns-allinone-2.35/ns-2.35n
$ gawk -f qos.awk bh-eccea.tr

GeneratedPackets              = 16
ReceivedPackets               = 13
Total Dropped Packets         = 4
Packet Delivery Ratio         = 81.25%
Average End-to-End Delay      = 11.3864 ms
Average Throughput[kbps]      = 0.54
Routing overhead              = 56.000
Normalized Routing overhead = 4.308


vijay@royale ~/ns-allinone-2.35/ns-2.35n
$
```

Fig 4.4: Running of sample Tcl file

4.2.5. NAM Analysis

The screen chart of figure 4.5 illustrates the node distribution using RWP mobility model. There are totally 20 nodes, three out of which are blackhole nodes. The CBR/UDP traffic scenario is generated using setdest and the connection scenario is generated using cbrgen.tcl script. For the scenario generation seed value is set to 1 and mobility is set to 5.

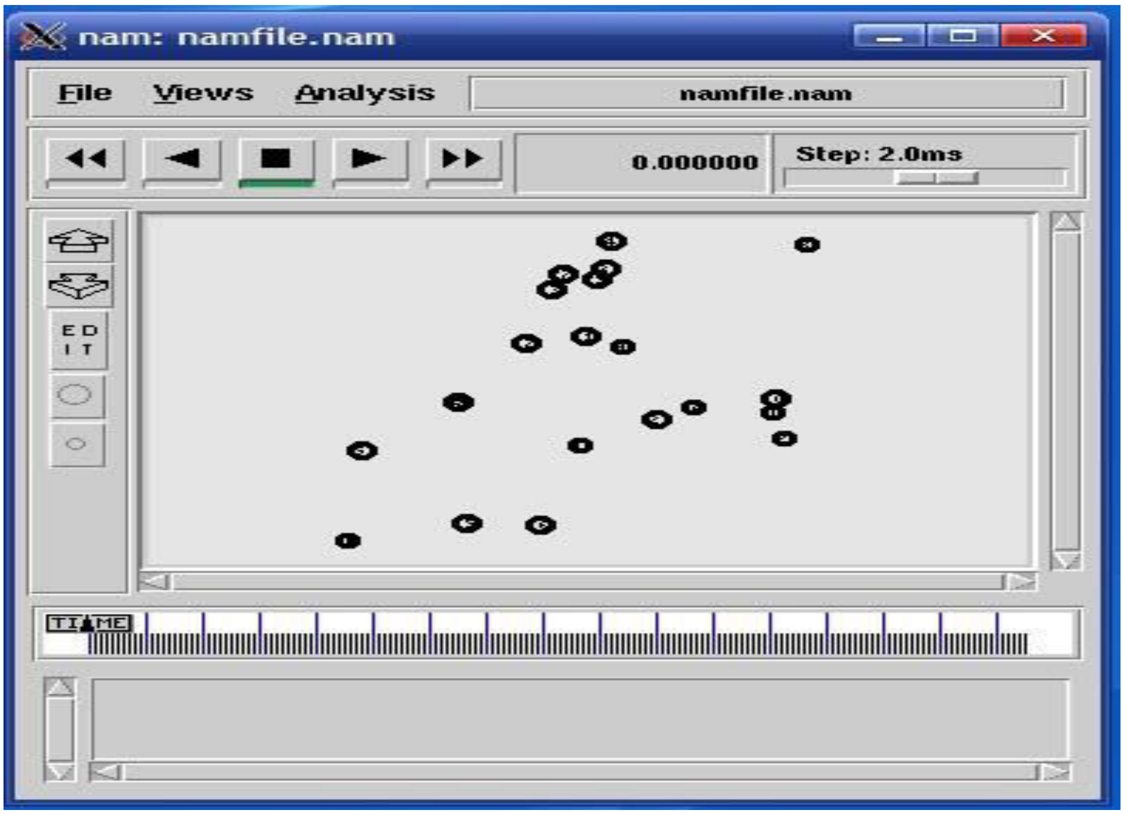
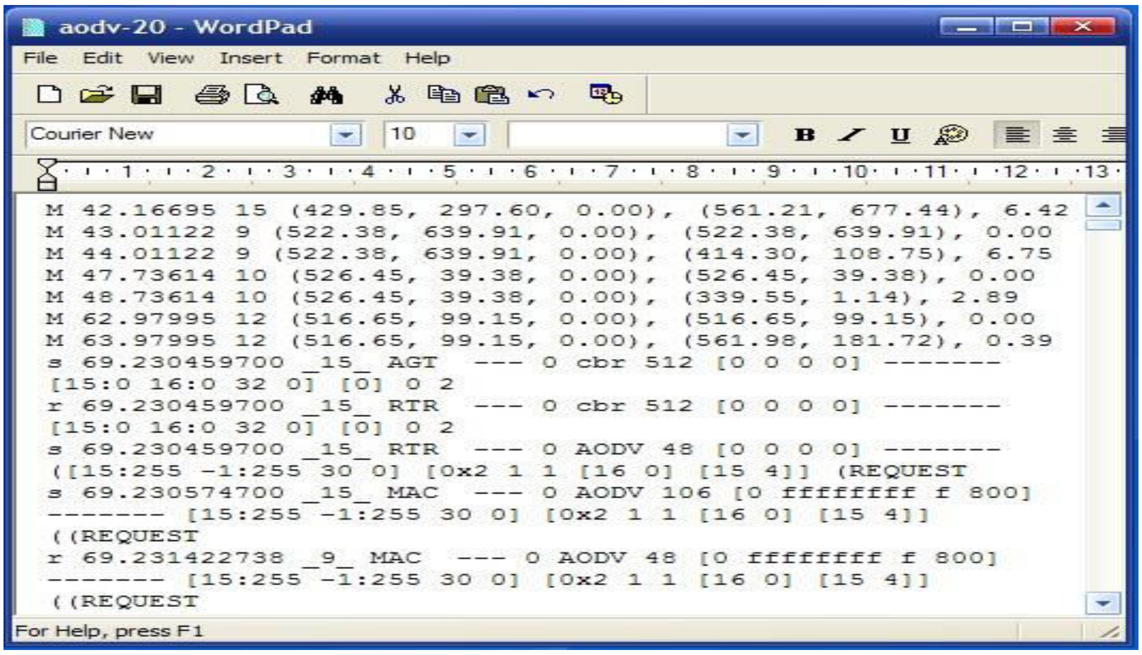


Fig 4.5: Screen shot of NAM

4.2.6. Trace File Analysis

The Trace file describes the process how nodes replicate the data items to neighboring nodes. It shows the detailed information about the data replication. It also shows the trace level, such as MAC, RTR, AGT levels.

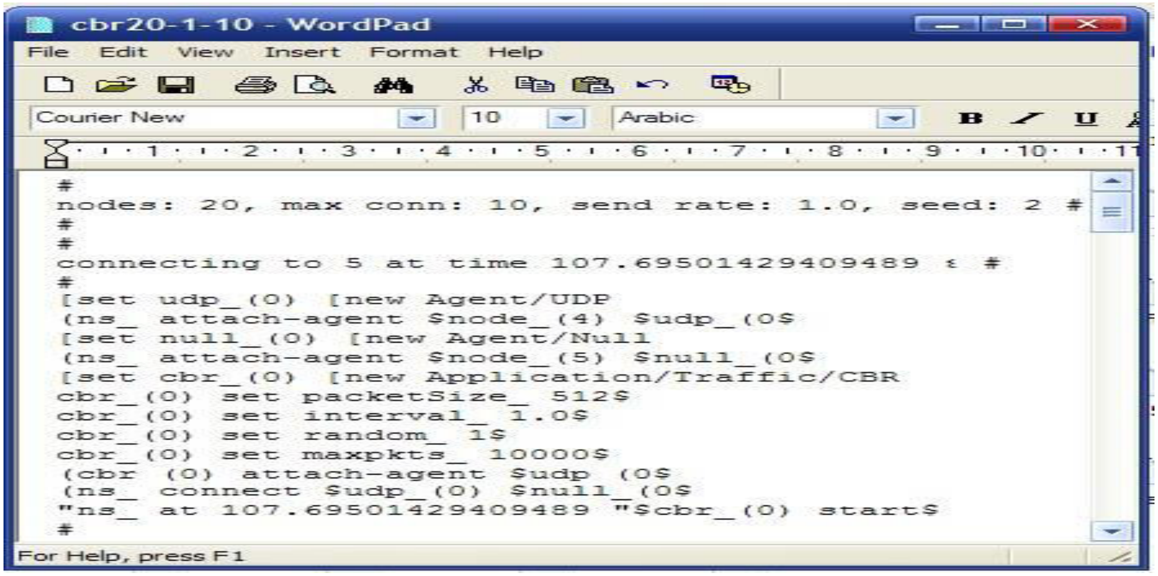
The below trace file illustrates how to send, receive and forward the packets. The nodes send and forward packets to its neighboring nodes and also receive packets from its neighboring nodes.



```
M 42.16695 15 (429.85, 297.60, 0.00), (561.21, 677.44), 6.42
M 43.01122 9 (522.38, 639.91, 0.00), (522.38, 639.91), 0.00
M 44.01122 9 (522.38, 639.91, 0.00), (414.30, 108.75), 6.75
M 47.73614 10 (526.45, 39.38, 0.00), (526.45, 39.38), 0.00
M 48.73614 10 (526.45, 39.38, 0.00), (339.55, 1.14), 2.89
M 62.97995 12 (516.65, 99.15, 0.00), (516.65, 99.15), 0.00
M 63.97995 12 (516.65, 99.15, 0.00), (561.98, 181.72), 0.39
s 69.230459700 _15_ AGT  --- 0 cbr 512 [0 0 0 0] -------
[15:0 16:0 32 0] [0] 0 2
r 69.230459700 _15_ RTR  --- 0 cbr 512 [0 0 0 0] -------
[15:0 16:0 32 0] [0] 0 2
s 69.230459700 _15_ RTR  --- 0 AODV 48 [0 0 0 0] -------
([15:255 -1:255 30 0] [0x2 1 1 [16 0] [15 4]] (REQUEST
s 69.230574700 _15_ MAC  --- 0 AODV 106 [0 ffffffff f 800]
------- [15:255 -1:255 30 0] [0x2 1 1 [16 0] [15 4]]
((REQUEST
r 69.231422738 _9_ MAC  --- 0 AODV 48 [0 ffffffff f 800]
------- [15:255 -1:255 30 0] [0x2 1 1 [16 0] [15 4]]
((REQUEST
```

Fig 4.6: Trace file analysis to send, receive and forward packets

4.2.7. Connection Pattern Scenario Generation



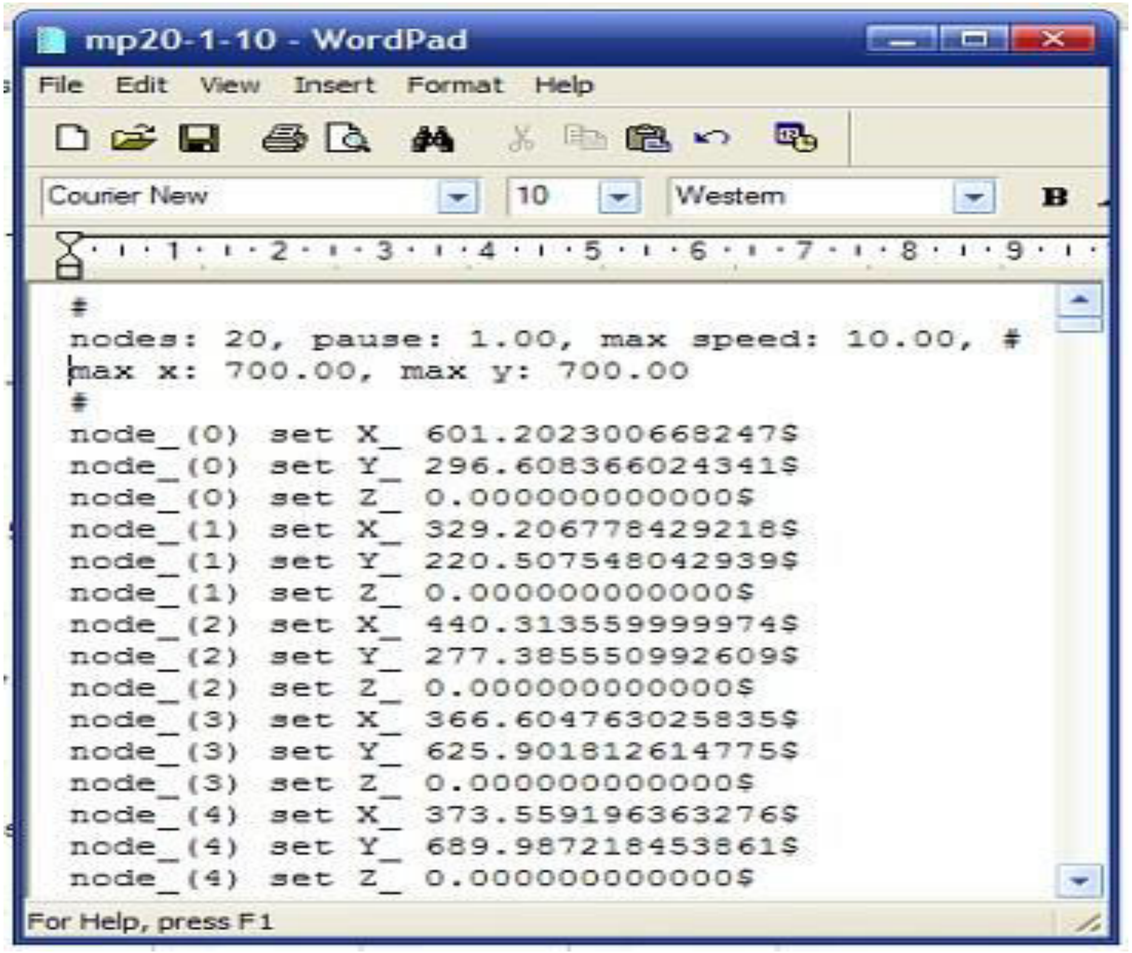
```
#
nodes: 20, max conn: 10, send rate: 1.0, seed: 2 #
#
#
connecting to 5 at time 107.69501429409489 ɛ #
#
[set udp_(0) [new Agent/UDP
(ns_ attach-agent $node_(4) $udp_(0$
[set null_(0) [new Agent/Null
(ns_ attach-agent $node_(5) $null_(0$
[set cbr_(0) [new Application/Traffic/CBR
cbr_(0) set packetSize_ 512$
cbr_(0) set interval_ 1.0$
cbr_(0) set random_ 1$
cbr_(0) set maxpkts_ 10000$
(cbr (0) attach-agent $udp (0$
(ns_ connect $udp_(0) $null_(0$
"ns_ at 107.69501429409489 "$cbr_(0) start$
#
```

Fig 4.7: cbrgen.tcl generated file

The connection pattern scenario file can be generated using built in cbrgen.tcl program of NS2. This is responsible for generating constant bit rate (CBR) traffic in a network. The figure shown above denotes the data present in the connection pattern file of wireless ad hoc network. This particular file gives us the entire information about the nodes, connections, seed values and send rate.

4.2.7.1. Traffic Scenario File

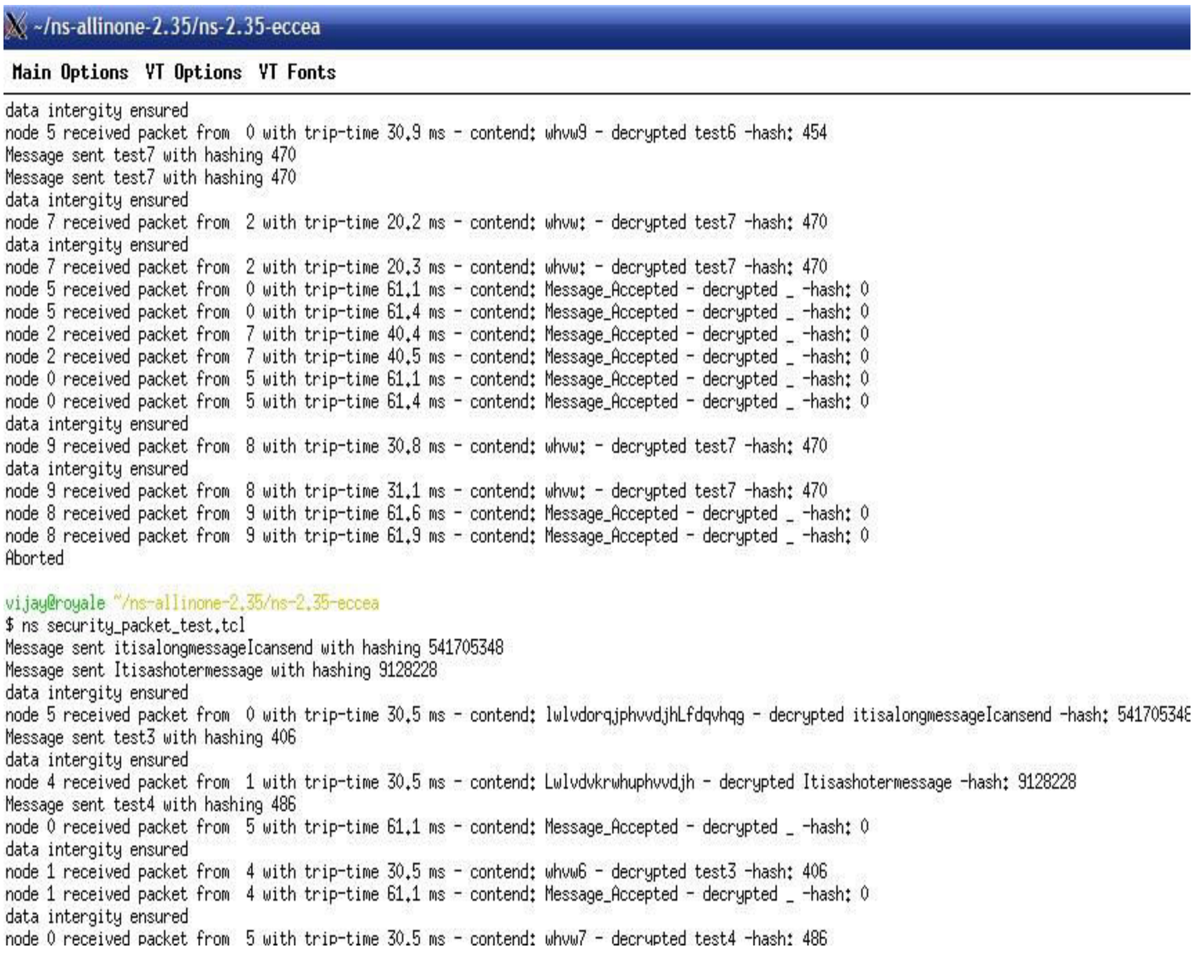
The traffic scenario file is created using a built in setdest program of NS2. This is responsible for creating mobility of nodes [88], [89] in a network. The figure shown below represents the data present in the traffic scenario file.



```
#
nodes: 20, pause: 1.00, max speed: 10.00, #
max x: 700.00, max y: 700.00
#
node_(0) set X_ 601.202300668247$
node_(0) set Y_ 296.608366024341$
node_(0) set Z_ 0.000000000000$
node_(1) set X_ 329.206778429218$
node_(1) set Y_ 220.507548042939$
node_(1) set Z_ 0.000000000000$
node_(2) set X_ 440.313559999974$
node_(2) set Y_ 277.385550992609$
node_(2) set Z_ 0.000000000000$
node_(3) set X_ 366.604763025835$
node_(3) set Y_ 625.901812614775$
node_(3) set Z_ 0.000000000000$
node_(4) set X_ 373.559196363276$
node_(4) set Y_ 689.987218453861$
node_(4) set Z_ 0.000000000000$
```

Fig 4.8: Generated Traffic scenario file

4.2.8. Data Integrity & Confidentiality through Hash function and Authentication techniques


~/ns-allinone-2.35/ns-2.35-eccea
Main Options VT Options VT Fonts

```
data intergity ensured
node 5 received packet from  0 with trip-time 30.9 ms - contend: whvw9 - decrypted test6 -hash: 454
Message sent test7 with hashing 470
Message sent test7 with hashing 470
data intergity ensured
node 7 received packet from  2 with trip-time 20.2 ms - contend: whvw: - decrypted test7 -hash: 470
data intergity ensured
node 7 received packet from  2 with trip-time 20.3 ms - contend: whvw: - decrypted test7 -hash: 470
node 5 received packet from  0 with trip-time 61.1 ms - contend: Message_Accepted - decrypted _ -hash: 0
node 5 received packet from  0 with trip-time 61.4 ms - contend: Message_Accepted - decrypted _ -hash: 0
node 2 received packet from  7 with trip-time 40.4 ms - contend: Message_Accepted - decrypted _ -hash: 0
node 2 received packet from  7 with trip-time 40.5 ms - contend: Message_Accepted - decrypted _ -hash: 0
node 0 received packet from  5 with trip-time 61.1 ms - contend: Message_Accepted - decrypted _ -hash: 0
node 0 received packet from  5 with trip-time 61.4 ms - contend: Message_Accepted - decrypted _ -hash: 0
data intergity ensured
node 9 received packet from  8 with trip-time 30.8 ms - contend: whvw: - decrypted test7 -hash: 470
data intergity ensured
node 9 received packet from  8 with trip-time 31.1 ms - contend: whvw: - decrypted test7 -hash: 470
node 8 received packet from  9 with trip-time 61.6 ms - contend: Message_Accepted - decrypted _ -hash: 0
node 8 received packet from  9 with trip-time 61.9 ms - contend: Message_Accepted - decrypted _ -hash: 0
Aborted

vijay@royale ~/ns-allinone-2.35/ns-2.35-eccea
$ ns security_packet_test.tcl
Message sent itisalongmessageIcansend with hashing 541705348
Message sent Itisashotermessage with hashing 9128228
data intergity ensured
node 5 received packet from  0 with trip-time 30.5 ms - contend: lwlvdorqjphvvdjhLfdqvhqg - decrypted itisalongmessageIcansend -hash: 54170534
Message sent test3 with hashing 406
data intergity ensured
node 4 received packet from  1 with trip-time 30.5 ms - contend: Lwlvdvkrwhuphvvdjh - decrypted Itisashotermessage -hash: 9128228
Message sent test4 with hashing 486
node 0 received packet from  5 with trip-time 61.1 ms - contend: Message_Accepted - decrypted _ -hash: 0
data intergity ensured
node 1 received packet from  4 with trip-time 30.5 ms - contend: whvw6 - decrypted test3 -hash: 406
node 1 received packet from  4 with trip-time 61.1 ms - contend: Message_Accepted - decrypted _ -hash: 0
data intergity ensured
node 0 received packet from  5 with trip-time 30.5 ms - contend: whvw7 - decrypted test4 -hash: 486
```

Figure 4.9: Secure Message with encryption and decryption technique

4.2.9. GNU Plot – Graphs

Gnuplot is an open source, interactive, command driven and data plotting program. The following commands are used to plot the graph.

```
> gnuplot               - to enter the gnuplot
> set xrange [0-100]    - to determine x- axis range
> set xtics    10       - to divide the x- axis range into equal parts
> set xlabel “value”    - to denote the values taken on x- axis
> set yrange [0-10]     - to determine y- axis range
```

> set ytics 1	- to divide the y- axis range into equal parts
> set ylabel "value"	- to denote the values taken on y- axis
> set title	- to set the title of the graph
> to plot	- plot "file" using 1:2 with linespoints t "file" lw 2 lc 1 ps 1(lc : line color, lw : line width, ps: point size)

4.2.10. Quality of service parameters for performance evaluation of routing protocols

The performance comparison of the protocols AODV, GTASA, SEA and ECCEA is made by evaluating the following QOS performance parameters [90], [91], [92].

- **Packet Delivery Ratio (PDR)**

It is the ratio of the number of packets received by the destination to the number packet originated from the source. This shows the level of data delivered to the destination.

PDR = Packets received / Packets sent

The larger value of the PDR denotes the better performance of the routing protocol

- **Throughput**

It is the mean rate of triumphant packet delivery over a transmission channel. The data may be delivered over a physical or logical, or it can be passed through a specific network node. The throughput normally considered in data packets per time slot or data packets per second and is sometimes bits per second.

- **Average end-to end delay (AEED)**

It is the average time taken by a data packet to travel across the network from source to destination. The queue waiting in data packet transmission and delay caused by the route discovery process is also considered. The information packets that are productively delivered to the destinations are only counted.

AEED = Sum of (arrive time – send time) / Total number of connections

The lesser value of AEED means better performance of the protocol.

- **Normalized Routing Overhead**

It can be defined as the ratio of the total control information transmitted to the total number of packets received. That is the routing overload per unit information delivered productively to the destination.

4.3. Implementation of a New Routing Protocol in NS2 to Simulate a Node with Blackhole Behavior

The implementation procedure of a new MANET Unicast Routing Protocol in NS2 is discussed in [93], [94], [95]. The details discussed in these papers are so much helpful for the proposed implementation. In this work, the nodes that exhibit blackhole behavior is used in MANETs that use AODV protocol. A new routing protocol that can take part in the AODV messaging is needed to incorporate the blackhole features into a node. This new routing protocol implementation is described below in detail.

All the routing protocols in NS2 are installed in the "ns-2.35" directory. The work is initiated by duplicating the existing AODV protocol in this directory and the name of the directory is changed as "**bh-aodv**". All the names of the files except "*aodv_packet.h*" in the directory that are labeled as "aodv" are changed to "**bh-aodv"** such as *bh-aodv.cc*, *bh-aodv.h*, *bh-aodv.tcl*, *bh-aodv_rqueue.cc*, *bh-aodv_rqueue.h* etc. The "*aodv-packet.h*" file is not copied into the bh-aodv directory as AODV and bh-AODV protocols will communicate each other the same AODV packets. In the directory, all classes, functions, structures, constants and variable names in all the files except structure names that belong to bh-packet.h file are changed. These two protocols AODV and bh-AODV are actually the same except the change of labels.

After these changes, the two common files "\tcl\lib\\ *ns-lib.tcl*" and "\makefile" in the basic directory of the "**ns-2.35**" that are used in NS-2 globally are changed to integrate the new bh-AODV protocol into the simulator. The new protocol agents are coded as a procedure in the first file *ns-lib.tcl.* This new agents are scheduled at the start of the simulation and are assigned to the nodes that uses bh-AODV protocol and the second file which is tailored "\makefile" in the root directory of "**ns-2.35**". With this the implementation of a new routing protocol labeled as bh-AODV is completed, but blackhole behavior has not yet been incorporated into this new routing protocol. To add blackhole behavior into the new AODV protocol some changes are made in bh-aodv/bh-aodv.cc file to incorporate blackhole features. After making all these changes, the NS2 is recompiled to create object files. Once the compilation is finished, the new test bed is ready to simulate the blackhole attack in AODV protocol. The same procedure is repeated to create the new test beds for the simulation of proposed secure routing protocols GTASA, SEA and ECCEA.