Security in Mobile Ad hoc Networks

by

Ankit K. Joshi

A Project Report Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Supervised by

Dr. Hans-Peter Bischof

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences Rochester Institute of Technology Rochester, New York

March 2013
The project “Security in Mobile Ad hoc Networks” by Ankit K. Joshi has been examined and approved by the following Examination Committee:

Dr. Hans-Peter Bischof  
Assistant Professor  
Project Committee Chair

Dr. Stanislaw Radziszowski  
Associate Professor
Acknowledgments

The successful completion of Master’s Project is attributed to the knowledge, support and guidance from few people.

I would like to take this opportunity to express my profound gratitude to the Committee Chair Prof. Hans-Peter Bischof for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. His willingness to share his vast knowledge provided me the right approach and desired impetus to start and complete the assigned project. A special vote of thanks to Prof. Stanislaw Radziszowski who ensured that my approach and implementation of the project was accurate from the beginning stage itself.

My thanks and appreciation also extends to my family and friends for their unwavering support and encouragement which motivates me to achieve my desired goals.
Abstract

Security in Mobile Ad hoc Networks

Ankit K. Joshi

Supervising Professor: Dr. Hans-Peter Bischof

Wireless ad hoc network does not have any centralized server but is a decentralized, independent of any pre-established infrastructure network. The nodes in the network are themselves responsible for routing the packets from the source to the destination. These nodes are also responsible to make the transfer of packets secure. This type of network is very vulnerable to different types of attacks. Thus, all the nodes in the network also have to make sure that there are no malicious nodes in the network, and if found, these nodes should be isolated and dealt with by the members of the network.

This project establishes a wireless ad-hoc network with the help of a simulator (assuming that all the packets reach the destination). The nodes created in the simulator are made to communicate with each other and also few malicious nodes are created in the network so that we can test if the purpose of the project, i.e. to isolate the malicious nodes and to deliver the packet from source to destination, works properly. Several tests have been run to check if the project is working properly.

The rest of the report gives detailed information about the simulator used, OTcl language, graphs, issues in such networks, security system and analysis.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>iv</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Related Work</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Routing Protocols</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Security Issues and Prevention methods</td>
<td>6</td>
</tr>
<tr>
<td>2.2.1 Security Issues</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2 Prevention Methods</td>
<td>7</td>
</tr>
<tr>
<td>3 Implementation</td>
<td>10</td>
</tr>
<tr>
<td>3.1 Network Simulator</td>
<td>10</td>
</tr>
<tr>
<td>3.1.1 Installing NS on Red Hat 9 / Windows XP</td>
<td>11</td>
</tr>
<tr>
<td>3.1.2 Working on Network Simulator</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Implementation of Routing Protocols</td>
<td>17</td>
</tr>
<tr>
<td>3.2.1 Route Discovery</td>
<td>17</td>
</tr>
<tr>
<td>3.2.2 Multicast Tree Construction</td>
<td>20</td>
</tr>
<tr>
<td>3.2.3 Maintaining Multicast Tree</td>
<td>21</td>
</tr>
<tr>
<td>3.2.4 Revocating Group Member</td>
<td>23</td>
</tr>
<tr>
<td>3.2.5 Merging Tree</td>
<td>24</td>
</tr>
<tr>
<td>3.2.6 Simulation Environment</td>
<td>25</td>
</tr>
<tr>
<td>3.3 Implementing Security in MANETs</td>
<td>25</td>
</tr>
<tr>
<td>3.3.1 2-way Diffie Hellman</td>
<td>25</td>
</tr>
<tr>
<td>3.3.2 n-Party Diffie-Hellman Key Exchange</td>
<td>27</td>
</tr>
<tr>
<td>4 Analysis</td>
<td>31</td>
</tr>
<tr>
<td>4.1 Hypothesis</td>
<td>31</td>
</tr>
<tr>
<td>4.2 Approach</td>
<td>31</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Multicast Route Discovery [17] .................................................. 5
2.2 Tree formation [14] ................................................................. 8
2.3 Group formation in K-Dimensional Tree [14] ............................. 8
2.4 Mechanism proposed in [22] ....................................................... 9

3.1 NAM editor window ............................................................... 13
3.2 NAM simulation window ........................................................ 13
3.3 Structure of NS2 directory ....................................................... 16
3.4 Route Request from Source to Destination ............................... 18
3.5 Route Response from Destination to Source ............................ 19
3.6 Diffie Hellman Example ......................................................... 26
3.7 Upflow stage ........................................................................ 29
3.8 Broadcast Stage ..................................................................... 29
3.9 Response Stage ..................................................................... 29
3.10 Final Broadcast stage ............................................................. 30

4.1 Broadcast message by nodes in the network ............................ 35
4.2 Movement of nodes in the network .......................................... 36
4.3 Node Join time ...................................................................... 36
4.4 Node leave time .................................................................... 37
4.5 Key Generation Time ............................................................. 37
4.6 Communication Overhead ....................................................... 38
4.7 Latency caused by the increase in number of receivers ............. 38
Chapter 1

Introduction

In recent years there have been new technologies coming up like: Bluetooth communication, IEEE 802.11 also known as Hyperlan, which help in advancement of communication. These new technologies or networks are also known as ad hoc networks. Ad hoc networks can be established without the need of any pre-established infrastructure. All the nodes in this network are mobile and thus, can move from one place to another. These nodes can communicate with each other until and unless they remain in the range of the network. As soon as they leave the range, the communication is lost. Mobile Ad hoc Networks also known as MANETs is a network where these mobile networks communicate using multi-hop methodology.

Unlike the wired networks such as LAN, MANETs do not have any centralized coordination or control. It is a decentralized network. All the nodes in the network used for communication and control of the network. These nodes coordinate with each other and help in communicating with each other. MANETs are widely used and deployed when the other communication systems are destroyed due to war, natural calamities like earthquakes, flood, etc... During such times, MANETs become very useful to setup immediate communication.

The nature of these networks make them more susceptible to different types of security issues. Various common issues faced by such networks are: interception, interference, and passive eavesdropping. The major disadvantage for such a network is when the node in the network itself becomes selfish and does not pass the packet to another node.

To protect such networks from various issues, many different types of methods have
been used in the literature. Most of them depend on they key management system. These schemes use public key infrastructure or private key infrastructure [21]. Asymmetric key algorithms are used in public key mechanism, whereas, symmetric key algorithms are used in private key mechanism. Few methods make use of both public and private key mechanisms. where public key is visible to everyone in the network, but private key is shared only between the nodes communicating with each other.

Most of the authors have specified use of Ad-hoc On-deman Deistance Vector (AODV) protocol for communication between nodes in MANETs. This method is called as a unicast routing protocol i.e. the source can communicate to only one destination at a time. Also, AODV is a very good protocol when mobility of the nodes in the network is very less. It becomes difficult for this protocol to work properly for the nodes with higher mobility. The major disadvantage here is that if the source has to send same packets to more than one node, it has to send the packet again to the second node only after the packet reaches the first node. This results in more time consumption.

To avoid all these factors, this project tries to establish a Multicast Ad-hoc On-demand Distance Vector Protocol (MAODV). This method helps the destination to send the packets to more than one destination simultaneously. This method also adapts the best part of AODV i.e. the On Demand approach of finding routes [20]. In this method, the routes are found if and only if the source wants to send a packet to the destination. Another property of AODV used in this MAODV is the Route Request (RREQ) and the Route Response (RREP) messages sent by the source and the destination. This will be explained in detail later.
Chapter 2

Related Work

The literature specifies many routing protocols and also different ways to protect them from malicious nodes and attacks. Wikipedia defines Mobile Ad hoc networks as "Mobile Ad hoc Network (MANET) is a self-organizing infrastructure less network of mobile devices connected without physical wire." [18]. The feature of such networks is that all the nodes in the network are mobile nodes, hence, they can move anywhere in the network without any restrictions. For this reason, maintaining the network become more and more difficult, as it requires to reconfigure itself on the fly. This makes protecting the nodes and the network a more difficult job.

The most common method used for protecting such networks is the Key Management system. The key management scheme used in most of the paper use either Public Key Infrastructure or the Private Key Infrastructure. The public key models use asymmetric key algorithms, whereas, private key models use symmetric key algorithms. Many of the asymmetric algorithms also make use of the Public and Private key pairs for security reasons. In such mechanisms, the Public key is visible to all the member of the network, whereas, the private key is visible only to the nodes communicating with each other.

2.1 Routing Protocols

The routing protocols can be categorized into two different types: Unicast Routing Protocols, where there is only one sender and one receiver. If the sender wants to send the same packet to more than one nodes, it has to resend the packet to the second node after sending
the packet to the first node. This may create communication overhead and the communication time required may also be more. The second type is the Multicast Routing Protocols where one node can send the same packets to more than one receivers. This helps in reducing the communication overhead and time. Many papers like [10], [7], [17], [3], [16], [11] have discussed various types of Multicast Protocols for Ad hoc systems.

The paper [10] explains a hypercube based multicast routing protocol. In this method, the nodes are organized in teams. This organization depends on the common interest of each node. Also, a hierarchical approach is used in this method making communication simpler. But the disadvantage is that, according to the paper, a node has only 35% chance of being successful in forwarding the packets. Also, this method requires to find the location of the node regularly, this causes the battery of the mobile nodes to drain quickly as they are constantly communicating for providing their location.

Few papers like [3], [16] describe cluster based multicast routing protocol. The method described in [3] is a fully distributed method. In this method, the network is divided into a group of nodes. These groups are called as clusters. Each cluster has cluster heads which is responsible for maintaining the clusters and communication between different clusters. The disadvantage with this method is that, even when this method is really good for limited number of members in clusters (according to paper 8 member in each clusters), but if the number of members in each cluster is increased, the result may not be good compared to other systems. So for a smaller network, this method is useful, but for larger network, this mechanism won’t provide the required results. The method described in [16] is similar to the method described in [3] but the main difference is that in [16], the clusters are created parallel. This reduces the time taken to form the network and groups. Again, this mechanism is good for a smaller network, but when the network size increases, the method does not provide the required results.

Another paper [7] provides a method based on the original AODV routing protocol. The only difference is that, in the proposed scheme, the protocol is Mobility Aware. This helps the nodes to find better routes between them. This scheme is not suitable for Multicast
Routing but is very helpful in unicast routing protocols where mobility is very high. The future work on this method can be to implement it on a multicast routing protocol.

Just like [3], another paper [11] proposes a scheme where the protocol is based on the location-based information of the nodes in the network. This method makes use of the Global Positioning System (GPS) to keep track of the location of the nodes. The network is divided into strips (vertical and horizontal). The GPS is used to locate the nodes in particular strips. The nodes themselves update their locations periodically. This may result in causing the battery of the devices to reduce rapidly. It is good for handling a very large network but again the power management is the major problem as these devices in ad hoc networks do not have very large power backups.

Of all the methods, I have modified the original AODV routing protocol and created the Multicast Ad-hoc On-demand Distance Vector (MAODV) routing protocol. This method adapts the mechanism used by AODV protocol to create the network. The only difference is that, here, the communication is done in multicast fashion i.e. more than two nodes can communicate with each other.

Just like the original AODV protocol, the methods also makes use of the RREQ (Route Request) and RREP (Route Response) messages. The node wanting to send the message send the RREQ message to the destination and the node receiving the message sends back the RREP message as an acknowledgement. The other nodes in the route pass the message till the message reaches the destination.

![Figure 2.1: Multicast Route Discovery [17]](image-url)
The figure above gives an idea of how a sender sends RREQ message to the two destinations simultaneously. The sender sends RREQ message to all the nodes in the network. The message is sent by the node till it reaches the destination nodes. As soon as the destination nodes receive the RREQ message, they respond to the source node through the reverse route with the RREP message. This lets the source understand that the destination nodes are ready and the route has been established. In the mechanism used in this project also applies the same procedure to find the routes between the source and the destination. The implementation of the project, the simulator used, the maintenance of tree, addition of nodes, finding the route everything is explained detailed in the Next Chapter i.e. Implementation.

2.2 Security Issues and Prevention methods

The mobile nature of the network, lack of infrastructure and non-centralized security systems make the network more vulnerable and susceptible to attacks than the wired networks. Various types of security methods are proposed in the literature but most use the key management systems. The key distribution or management schemes are divided into symmetric key or asymmetric key systems.

2.2.1 Security Issues

Few common attacks on Mobile Ad-hoc Networks are explained in [12]:

- Sybil Attack, where a malicious node creates multiple identities to manipulate keys to its advantage, thus, violating confidentiality, authentication and non-repudiation principles [12].

- Masquerade Attack, where identity of a legitimate node is forged by a malicious node again violating the above principles. Such attacks can also be used as Man-In-Middle (MIM) attacks.

- Lack of Cooperation, is where one node does not forward any packets to the upstream node. Here, such nodes are called selfish nodes, which use the resources of the
network without cooperating with any network operations. These nodes may not necessarily be malicious nodes but are harmful for the network.

Paper [12] proposes a scheme to avoid such attacks. This scheme is called as survivability, where the key is available even when the network is compromised. Well this may help in letting the flow of the network not get disturbed but, if the key is available even during such times, the network may still be at risk if the key reaches the malicious node.

Many papers like [8], [13], [22], [6], [9], [3], [17], [4], [5], [14] propose much better schemes. Most of these methods schemes provide good security for communication between nodes in the network. While selecting a key distribution scheme, one has to keep in mind few very important points.

- **Forward Secrecy**: A node joining the network should not be able to compute the prior group key used.

- **Backward Secrecy**: A node leaving the network should not be able to compute the future group keys.

- **Key Independence**: For group key based systems, any node which is not the part of the group should not be able to derive the group key through any information.

- **Group Key Secrecy**: For group key based systems, any adversary should not be able to compute the group key. This can result in compromising the network security.

### 2.2.2 Prevention Methods

The most common prevention method used for securing such networks and reducing risks is the Key Management System. Various papers propose different types of key management schemes. The papers like [4], [14] make use of the Tree based key management schemes. The method proposed in [4] is based on tree like P2P system where the nodes in the network themselves form trees by randomly ordering the mobile node IDs in the topology [4]. The root node is known as the Associated Trusted Mobile Node (ATMn), the head of the tree is
known as the Trusted Mobile Node (TdMn) and the other nodes are known as the Truster Mobile Nodes (TrMn). The method is good as the key generation and regeneration time required is less. But the major drawback is that the method has communication overhead and it does not support large networks or highly mobile networks. Figure below shows the general formation of a tree based mechanism

A K-Dimensional Tree for key-management is proposed in [14]. According to the authors, this method reduces the memory requirement by 50% when compared to other tree-based systems. The node joining and leaving procedure also does not require many key changes as compared to the other systems. The major disadvantage would be that, the number of groups formed would be very high as each group consists of mostly 3 nodes only. This will results in communication overhead. The group formation in K-Dimensional Tree is shown in figure 2.3

![Figure 2.2: Tree formation [14]](image1)

![Figure 2.3: Group formation in K-Dimensional Tree [14]](image2)
The Public Key Infrastructure proposed in [5] is based on distribution of authentication responsibilities to several Certificate Authorities (CAs) [5]. For authenticating the nodes in the network, a certification graph is used. The nodes with the maximum cliques are selected as CAs. The procedure used for managing key is a bit lengthier in this method. This results in wasting a little time for generating or regenerating the keys.

From all the methods, I have used the method proposed in [22] which makes use of the reliable Diffie-Hellman theorem for n nodes. This method is also known as an n-party Diffie-Hellman theorem. According to the authors of this paper, the method creates two types of trees: Red Tree and Blue Tree. This method makes sure that none of the nodes in the network are left behind, thus, covering all the nodes in the network. Also, there are few intermediate nodes which fall in both Red and Blue trees. These intermediate nodes make it possible for the nodes from different trees to communicate with each other. Figure below gives and idea of the mechanism proposed in [22]

![Mechanism proposed in [22]](image)

Figure 2.4: Mechanism proposed in [22]

The method I have implemented in this project also makes use of the n-party Diffie Hellman theorem. It is a modification of the original 2-party Diffie Hellman theorem. In the method implemented, the key exchange is done between n nodes and not just the 2 communicating nodes. This helps the purpose of Multicasting, where a source node can communicate with more than one destination simultaneously. The detailed implementation is explained later in the next chapter.
Chapter 3

Implementation

For this project, I have used the Network Simulator (NS 2.26) with Red Hat 9. The Network simulator can be used to add the nodes and create a virtual network for the nodes. The use of Red Hat 9 is because the best and most reliable simulator was NS 2.24 and it works only with Red Hat 9. So with the help of VM-Player, I was able to install the Red Hat 9 Operating System on my Windows 7 OS and then install the network simulator. The detailed explanation of the installation is provided later.

3.1 Network Simulator

The network simulator used for this project is the NS 2.26 version which is a very stable simulator. Network simulators are very helpful in creating virtual networks to test and work with the network agents. NS can be used to model the behavior of a network by calculating the characteristics of the network. According to wikipedia, “A network simulator is a piece of software or hardware that predicts the behavior of a network, without an actual network being present” [19]. The network simulator has a built-in tool NAM which is a visualization trace tool. It has GUI interface which can be used to generate NS scripts. This helps in reducing the manual errors while creating links between nodes and creating the node itself. The languages supported by NS 2.26 are OTcl and C++. I have used OTcl and C++ for this project, where I had to bind C++ with OTcl. NAM can also be used for comparing various results and presenting them in form of charts. This helps in understanding the simulation and results easily.
3.1.1 Installing NS on Red Hat 9 / Windows XP

This section provides the information about how to install NS on Red Hat machine. The installation procedure for any Linux operating system is similar to the one explained below. For a windows Operating System, the installation is different. The later part of this section provides the information about installing NS on Windows machines.

Red Hat 9 Machine:

The first step is to login to the machine as root.

userid : root
password : ****** [your selected password]

After you log in, you must download the simulator from the following website:
http://www.isi.edu/nsnam/dist/ [1]

Select the required simulator (in our case, ns-allinone-2.26.tar.gz). Once downloaded follow the steps below:

- Place the file in /usr/src

- In the terminal window, navigate to /usr/src directory by entering: cd /usr/src.

- Unpack the file by using the command: tar -xvzf ns-allinone-226.tar.gz. The folder will be unpacked to /usr/src/ns-allinone-2.26 and will create a new folder named ns-allinone-2.26

- Navigate to this directory cd nsallinone-2.26.

- Now install the simulator by entering ./install. This will install the whole simulator in your system.

The installation may take some time. After the installation is done, validate the installation by using ./validate command

The simulator may not be always run through root user. So to make the simulator available to all the users, a bash script has to be used. This file must be created and placed
The name of the script file must be run-ns.sh and the contents of the script are as follows:

```
#!/bin/sh

p1=/usr/src/ns-allinone-2.26/bin
p2=/usr/src/ns-allinone-2.26/otcl-1.0a8
p3=/usr/src/ns-allinone-2.26/lib
p4=/usr/src/ns-allinone-2.26/lib/tcl8.3
p5=/usr/src/ns-allinone-2.26/lib/tk8.3

export PATH

$PATH

LD_LIBRARY_PATH=$p3:$p4:$p5
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH
LD_LIBRARY_PATH

ns $1
```

With the help of the above script, you can run the ns simulator for any user by typing `run-ns.sh` in the terminal window. To make sure that this script file can be used by everyone, just check the properties of the script file and allow all users to execute the file.

Similarly, to run NAM for users other than the root, the above script can be copied and renamed to run-nam.sh and only one small change is required. Change the last line i.e. `ns $1` to `nam $1`. And then type `run-nam.sh` in the terminal window. The last line is basically used for invoking the simulator and file name is used to pass as an argument. Figures below show the GUI of NAM.
The above figure shows the window where you can edit an already existing ‘nam’ file or create a new ‘nam’ file. You can add nodes, links, agents and so on. The figure below shows the window where you can simulate an already created ‘nam’ file. It provides the functionality to change the speed of the simulation, fast forward or rewind the simulation, stop, pause and record the animation.
Windows:

For Windows Operating System, one needs to install a substitute for terminal i.e. Cygwin. For Windows XP, the stable version of network simulator is the NS-2.33 and NS 2.34 and Cygwin versions 1.5.25 and 1.7.1. You also need to download gcc version 3.4.4.xxx. Windows 7 can handle NS-2.35 with Cygwin 1.7.1 and gcc version 4.3.4. Following steps will help in installing the simulator on Windows OS:

- Install using the setup.exe file
- Follow the on-screen instructions
- Do not install the default packages. Search for gcc4 and other packages which may be required.
- Open Cygwin and download the ns2allinone2.35-RC3 package and keep in the home directory. To check the home directory type `pwd`. Extract the package using: `tar -xvf ns-allinone-2.35-RC3.tar.gz`
- Install the package by navigating to `cd ns-allinone-2.35-RC3` and typing `./install`
- The installation will complete after few minutes. The Windows version does not allow us to verify the installed packages. The installation creates a bash file “.bashrc”. This file is located in the home directory. You need to add the following lines in the .bashrc file:

```bash
NSHOME=~/ns-allinone-2.35-RC3
OTCL_LIB=${NSHOME}/otcl-1.14
NS2_LIB=${NSHOME}/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:OTCL_LIB:NS2_LIB
export TCL_LIBRARY=${NSHOME}/ns-2.35:${NSHOME}/name-1.15
```
This would complete the installation of Network Simulator NS-2.35 on Windows XP. To check if everything is installed correctly, type `ns`. If everything is working properly, the NS command prompt “%” should appear on the screen. This can be called as validation of the installed package.

If for any reason, the NS command prompt “%” does not appear, try and create a link to the NS2 executable:

cd /usr/local/bin

ln -s /home/username/ns-allinone-2.35-RC3/ns-2.35/ns.exe // Now it should work properly.

### 3.1.2 Working on Network Simulator

The Network Simulator supports two languages: **OTcl** and **C++** [2]. OTcl is the base for object-oriented programming and C++ is the base for kernel part of NS2. In order to build the network on the simulator and to create nodes, OTcl is used. It becomes easy to configure the network parameters. GUI provided by NAM makes node creation, links between nodes, design of the topology easy. C++ program can be used for single nodes i.e. if you want to write a particular code of each nodes, this programming language become very useful. C++ can also be used to overwrite the existing protocols of NS2 [2]. Selecting the programming method depends on the requirements such as: OTcl is easy to code but slow to execute, whereas, C++ is slow to code by fast to execute.

NS2 makes an easy choice for this project as it supports most of the network protocols like TCP, MAC, Ad-hoc Routing like DSDR, DSR, AODV, TORA, Multicast protocols, etc....

The simulator also has built-in preprocessing components (*traffic and topology generators* and post-processing components (*simple trace analysis available in Tcl or in most cases in Perl*).

The structure of NS2’s directory can be seen below:
Writing OTcl code is very simple. Few examples of how to create a simulator using OTcl code are given below:

#Create a simulator object:

```otcl
set ns [new simulator]
```

#Create nodes:

```otcl
set n0 [ns node]
set n1 [ns node]
```

This creates nodes n0 and n1 in the network.

#Create links between nodes:

```otcl
$ns duplex-link $n0 $n1 2Mb 10ms DropTail
```

The above statement creates a two-way link between the nodes n0 and n1. This link has maximum speed of 2Mb for 10ms.

The above statements are helpful in creating the network. There are also codes for analyzing the network which we have created. We can trace the events taking place between nodes. To do this, we need to turn the tracing ON for specific links. For example:

# Turn on tracing:

```otcl
$ns_trace-queue $n0 $n1
```
3.2 Implementation of Routing Protocols

The network creates different multicast groups and provides each group with separate unique multicast addresses. These groups are organized using tree structures. These tree structures consist of nodes and routers. The node which first constructs the group is called the group leader and is responsible for maintaining the group tree. This is done by periodically broadcasting a “Hello” message to each member of the group.

The basic idea behind knowing the routes and the next hop nodes in the network is done by keeping track of the nodes in the group and network with the help of the tables. There are three tables maintained by the nodes. Since MAODV is based on the original AODV, we make use of the unicast function i.e. the nodes maintain a Unicast Route Table. Now, since this is multicast routing protocol, the nodes also have to keep track of multicast routes whenever the destination address is the multicast address. For this purpose, the nodes also maintain a table i.e. the Multicast Route Table. All the nodes in the group must know their group leaders and the addresses. This is necessary so that the nodes can reply to the “HELLO” message sent by the group leader periodically. For this purpose, the nodes have to maintain the third table i.e. the Group Leader Table.

The following sections will explain the ways by which the routes can be discovered.

3.2.1 Route Discovery

Since we make use of the original AODV, there needs to be a way to find unicast routes and for MAODV, we need multicast routes. For this purpose, both the methods are described in detail below.

Unicast Route Discovery

This is the part of AODV routing protocol, where, the source has to reach a specific node in the network. NS2 provides a built-in standard implementation. There are few things to be specified for this implementation in this project.
- Only single-hop method is used for detecting neighboring nodes.
- The broken links can be detected only at MAC layer.
- Method to repair the broken links is not implemented.
- The source node is allowed to discover new routes even when the broken link is detected.

**Multicast Route Discovery**

MAODV needs to make sure that the addresses of all the destination nodes must be recorded even when the node is not the part of the multicast tree. The nodes which are not the part of the multicast tree must find a unicast route to the node in the multicast tree in order to send and receive packets. For this purpose, the Unicast approach of AODV is used. The diagram below explains the procedure used to find the route in AODV.

![Diagram](image)

**Figure 3.4: Route Request from Source to Destination**

In figure 3.2, the source sends the route request to the destination by sending RREQ (Route Request) message to the neighboring nodes which do the same till the message
reaches the destination. When the message reaches the destination, it includes the node number and the hops it has taken to reach the node. The destination then replies using the route where minimum hopes are used.

![Route Response from Destination to Source](image)

**Figure 3.5: Route Response from Destination to Source**

As shows in figure 3.3, the destination replies using the path (6,5,1) which has minimum hops (3 hops). This method helps in finding the best route between the communicating nodes.

When a node is sending a multicast message, all the nodes in the network must know and check if they are in the multicast tree. If the nodes are not the members of the multicast tree, they need to find the address of the next hop by making use of the Unicast Route Table. With the help of this table, the nodes can find the address of the next multicast address. If it gets the information, the data packets are sent to the next hop address but if the information is not found, the node send a separate RREP message back to the source in case the source still requires this node for route discovery. But if the node is in the multicast tree, the Multicast Route Table is used by the node to forward the packets [23].
During the multicast route discovery, if a broken link is found, a separate RREP message is sent to the source node so that it can discover a new route.

### 3.2.2 Multicast Tree Construction

The method used for Constructing the Tree is same as that used by AODV to find routes i.e. by sending the RREQ and RREP messages between the nodes. Also, a messages is used to finish the tree construction. This message is the MACT (Multicast Route Activation) message [23].

The nodes who want to join the multicast group (if that node is not the member of that group), it sends a RREQ messages sent with the help of a flag i.e. the Join flag (RREQ-J). The node, before sending this message, has to create an entry in its Multicast Route Table and show itself as the member of the group. But this entry will not have any group leader or any next hop sequence. The RREQ-J message can be sent directly by a node wanting to join the group to the group leader if that node gets the information of the leader, otherwise, the message has to be broadcasted to the network. If the message is broadcasted, the reverse route (from group leader to the sender node) is inserted in the Unicast Route Table.

The RREP-J message is sent back to the source only by the group member of the potential group or the member of the bigger multicast group members. With the help of the Unicast Route Table, the RREP-J is sent back via the reverse route [23]. After receiving this message, the node edits the entry in the Multicast Route Table with the address of the Group leader, and also the upstream and downstream entries. This makes the node the part of the multicast group. The route between this node and the Group Leader may not be the best route possible. So, there are possibilities that the node may receive another RREP-J message. This RREP-J message will provide the information of the number of hops it took from Group Leader to the node. If the number of hops is less then the previous route, the node updates the entry in the table with this new route. If the new route is not better than the one already existing between the node and the leader, the new RREP-J message will be discarded.
The main part is adding the new branch to the tree. This is done with the help of the MACT message. This is done after the node receives the RREP-J message. The node, checks for the RREP-J message periodically after every RREP\_WAIT\_TIME [23]. If the node has received the RREP-J message, it acknowledges the message (for the first time), establishes the route between the node and the leader and send back the MACT-J (Multicast Route Activation Join) message. The message is sent to the upstream node from where it received the RREP-J message. Thus, establishing a upstream hop for itself and a down-stream hop for the other node. This adds the node to the tree. Also, all the nodes receiving the MACT-J message, must update its Multicast Route Table and also the Unicast Route Table, with the next hop address.

3.2.3 Maintaining Multicast Tree

Maintaining a multicast tree is much more complicated then maintaining a unicast tree. Multicast tree maintenance includes, sending periodically HELLO message to the group members, keeping track of the neighboring nodes, selecting group leaders, revocating members that are acting maliciously and merging trees or branches.

Periodic Hello Message

As explained earlier, the group leader periodically broadcasts the HELLO message to its group members. All the nodes receiving this message, updates it Group Leader Table. This helps the nodes to know the address of the Group Leader also the route towards the Group Leader. If the node receiving the message is not he tree member, it retransmits the HELLO message. On the other hand, the node receiving the HELLO message is the part of the tree and is getting the message from its own upstream, can make use of this message and update the route sequence number, the group leader, and the number of hops from the group leader. If the member receives this message from a node other than its own upstream node, it checks if the group leader is the same. If it is the same, it discards the message and awaits for the new message from its own upstream node. This message can also initiate
Tree Merge. This will be explained in the Tree Merge section.

**Connectivity with Neighbor Nodes**

The MAC layer link breakage cannot be detected in Multicast Routing Protocols because the message are broadcasted through the trees. For this purpose, each node in the tree are supposed to send periodically another Hello-N message. This can cause overhead, but to reduce this overhead, the nodes can delay sending this message if it has just received a data packet or any other broadcast message from its neighbors.

If a node does not receive any HELLO-N message from its upstream node for 3 HELLO-N intervals, it detects the link is broken. After knowing that the link to its upstream is broken, the node deletes the entry from its Multicast Route Table and it becomes the broadcasting node which wants to find another route and branch to reconnect to the Group Leader. So this node sends the RREQ-J message and the process continues just as explained in the Multicast Tree Construction section. If this node receives the RREP-J message, then the new branch is added to the tree and the new upstream node updates its Route Table with the address of the new downstream node.

But if the node does not receive any RREP-J message, it means that the tree partitioning has occurred and a new Multicast tree is created. The Group Leader of this new Multicast Tree is selected. This is explained in the section below.

**Selecting Group Leaders**

Group leaders have to be selected not only when the new tree is created but also when the current Group Leaders leave the tree or the network. There is also one more situation when the Group Leader wants to opt out and does not want to be the leader anymore, it will ask the other group member to select another leader. All these situations are discussed below.

- If a new tree is created due to partitioning of tree as explained in the above section, a new Group Leader must be selected. The best node to become the group leader when
the new tree is created due to link breakage is the node that has no upstream node. This node will have only the downstream links with all the other nodes.

- When the Group Leader is leaving the group tree, it sends a message to its downstream nodes stating that it is leaving the tree. So the downstream links have to select a new Group Leader. Once the node receives the message it deletes the record of its upstream node from its Multicast Route Table.

- If for any reason, the group leader wants to give up the leadership, it again sends the message to all its downstream nodes. If there are more than one downstream nodes, the leaving group leader send the message to both the downstream nodes and it itself become downstream nodes to one of the nodes which become the Group Leader. The node which become the Group Leader, makes the changes from upstream to downstream in its Multicast Group leader, whereas, the other nodes in the tree only updates its Group Leader Table.

After the new Group Leader is selected, it send the HELLO message to all its group members. After the members receive this message, they realize that the Group Leader has changed. So, they also update their Group Leader Table with the address of the new Group Leader. They also update the route in their Multicast Route Table towards the new Group Leader.

### 3.2.4 Revocating Group Member

Revocation can be done by any member of the multicast tree including the group leader. For the group leader revocation, the group leader changes its identity to be a router i.e. any other node in the network which is not the member of the multicast tree. These router nodes are used by multicast tree nodes just to route the message from source to destination when the route includes these routers. The new group leader is selected in the manner described in the above section.
If the revocation node is not the group leader, it first discards the membership by changing its identity to the router. After changing its identity, it has to check if it has any downstream nodes. If there is a downstream node, it has to remain in the group to connect the members [23]. If there is no downstream node, it can revoke without any need to remain in the group.

Once the member is revoked, it sends a MACT-R revocation message to its upstream node so that this upstream node can update its Multicast Route Tree to delete the entry of the revoked node. This upstream node becomes the leaf node.

### 3.2.5 Merging Tree

For this purpose, I have used the method described in [15]. Tree merge is required when a new tree is established by the group member of the multicast tree due to revocation of a member node. When a multicast tree member receives a HELLO from a Group Leader which has a bigger address than its own group leader. The method used is not exactly as described in [15].

The method proposed in [15] does not allow any member of the other group member to initiate the merge, but in the method used in this project, any tree member in the group can initiate tree merge. For this purpose, the method used is similar to the method used for joining the node to the tree. The tree member that is initiating the tree merge sends a RREQ message to the group leader with the larger address. The only difference is that the RREQ message is sent with the repair flag (RREQ-R) rather than the join flag.

This RREQ-R message is propagated by the tree member initiating the merge till it reaches the Group Leader with the larger address. As soon as the group leader receives this message and approves the node to join the tree, it replies the node with the RREP-R message letting the node know that the trees can be merged. This establishing the reverse route and the branch is established to the new tree. After this tree merge is done, the new group leader sends the the HELLO message to all the nodes in the tree and all the nodes in the tree (the new members of the merged tree) update their Group Leader Table and also
the leaf node where the tree is joined, updates its Multicast Tree Table with the downstream node.

### 3.2.6 Simulation Environment

The simulation has been done with the help of network simulator NS2.26 on Red Hat 9 Operating System. The simulation environment is as follows:

- The network area selected is 1000 x 1000 meters
- Number of nodes in the environment are 50
- Duration for simulation: 200 sec
- Mobility Model: Random Waypoint Model with pause time, node movement speed (max 10Mbps)
- Each node in the multicast tree is the multicast receiver.
- Each node in the network may not be the part of the multicast tree.
- The simulation shows only multicast traffic.

### 3.3 Implementing Security in MANETs

The method used in this project is similar to the method proposed in [22]. The main difference is that, unlike the proposed method in [22], the mechanism used in this project does not make 2 different trees. There is only one tree for managing the keys.

#### 3.3.1 2-way Diffie Hellman

*n-Party* Diffie Hellmen, also known as Group Diffie Hellmen is an extension of the original 2-way Diffie Hellman theorem. First, let us understand the basic 2-way Diffie Hellman algorithm with the help of two users Alice and Bob. Suppose Alice and Bob want to communicate with each other. The steps they have to take are as follows:
• First, both of them will have to decide on a large Prime Number $p$ and a large Public Key $q$

• Alice selects a secret integer (known only to itself) ‘a’ and calculates the its key:
  \[ A = q^a \mod p \]

• Bob also selects a secret integer ‘b’ and calculates its key:
  \[ B = q^b \mod p \]

• Both Alice and Bob exchange the keys.

• After exchanging the keys, a secret key ‘s’ is calculated by both Alice and Bob
  Alice: \[ s = B^a \mod p \]
  Bob: \[ s = A^b \mod p \]

• This shared key calculated by Alice and Bob will be same and would be used to send and receive packets between each other.

The figure below provides an example with a better understanding:

![Figure 3.6: Diffie Hellman Example](image)
The Private Key (K) generated by both Alice and Bob will be same. This helps in encrypting and decrypting the data packet without the information of the private key to be exchanged.

The above mentioned method is very good and efficient when 2 nodes are communicating with each other. For more nodes to communicate with each other, a similar mechanism is adopted and used in this project. This is explained in the section below.

### 3.3.2 n-Party Diffie-Hellman Key Exchange

The \textit{n-party} Diffie Hellman Key Exchange mechanism is based on the 2-way Diffie Hellman algorithm explained in the previous section. The method adopted is very similar to the one above for more than 2 nodes communicating with each other. There are many types of Group Diffie Hellman protocols in the literature. The first three protocols assumed that the groups in the network would not change i.e. there would be no addition of new members or leaving of any members from the group. Later, new protocols were proposed which did not assume the networks to be static. They worked on the addition and deletion of nodes from the network. In this project, the Group Diffie Hellman algorithm used also takes care of the addition of new members and deletion of the members from the network.

#### Notations that would be used in the later part of this section:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>number of participants in the protocol</td>
</tr>
<tr>
<td>( i,j,k )</td>
<td>indices of group members, ( {i,j,k} \in [1,n] )</td>
</tr>
<tr>
<td>( M_i )</td>
<td>( i^{th} ) group member, where, ( i \in [1,n] )</td>
</tr>
<tr>
<td>( q )</td>
<td>order of the algebraic group</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>exponential base; generator in the algebraic group delimited by ( q )</td>
</tr>
<tr>
<td>( N_i )</td>
<td>random exponent generated by group member ( M_i )</td>
</tr>
<tr>
<td>( S, T )</td>
<td>subsets of ( {N_1 \ldots N_n} )</td>
</tr>
<tr>
<td>( \Pi(S) )</td>
<td>product of all elements in subset ( S )</td>
</tr>
<tr>
<td>( K_n )</td>
<td>group key shared among ( n ) members</td>
</tr>
</tbody>
</table>

(we also use \( K \) when \( n \) is obvious)
The above notation were done with the help of [?]. Just like the 2-way Diffie Hellman mechanism, the \textit{n-party} Diffie Hellman mechanism also calculates the group key but in this case, all the members of the group help in generating the key. For example, if there are 4 members in the group, say \textit{X1, X2, X3, X4} the key is generated as follows:

\[ \{ \alpha^{x2x3x4}, \alpha^{x1x2x4}, \alpha^{x1x3x4}, \alpha^{x1x2x3}, \alpha^{x1x2x3x4} \} \]

Similar to the 2-party Diffie Hellman theorem, the node in \textit{n-party} Diffie Hellman also have to follow steps in order to generate the group key.

- All the participants \textit{'n'} in the group agree on a cyclic group \textit{‘G’}, of order \textit{q} and base \textit{\alpha}

- Each member \textit{M_i} in the group selects a random value \textit{N_i} such that \textit{N_i \in G}

- Each member computes \textit{\alpha^{N_i} \mod q}, \ldots \ldots, \textit{\alpha^{N_i N_{i-1}} \mod q}, \textit{\alpha^{N_i N_{i-1} N_{i+1}} \mod q}, \textit{\alpha^{N_i N_{i-1} N_{i+1} N_n} \mod q}

- Finally, \textit{M_i} computes the shared key

\[ K = \alpha^{N_i \ldots N_n} \mod q \]

The above steps makes sure that the protocol is secure. Now the major problem is what subset of key \textit{(S)} to distribute and how? This depends on the type of Group Diffie Hellman a person is using. In this project, we are using the method where a node addition and deletion is taken care off and also there is a broadcast and response stage before the final broadcast stage.

- In the \textbf{Upflow} stage, the member \textit{M_i} [i \in [1,n-2]] receives a set \{ \textit{\alpha^{N_i}}, \ldots \ldots, \textit{\alpha^{N_i N_{i-1}}, \alpha^{N_i N_{i-1} N_{i+1}}, \alpha^{N_i N_{i-1} N_{i+1} N_n}} \} and forwards to \textit{M_{i+1}} \{ \textit{\alpha^{N_i}}, \ldots \ldots, \textit{\alpha^{N_i N_{i-1}}, \alpha^{N_i N_{i-1} N_{i+1}}, \alpha^{N_i N_{i-1} N_{i+1} N_n}} \}, where \textit{i \in [1, n-1]}
• In the **Broadcast** stage, \( M_{n-1} \) broadcasts \( \{ \alpha^{N_1}, \ldots, \alpha^{N_1 N_{i-1}}, \alpha^{N_1 N_{i-1} N_{i+1}}, \alpha^{N_1 N_{i-1} N_{i+1} N_m} \} \) to \( M_i \), where \( i \neq n-1 \)

• In the **Response** stage, each member \( M_i \) factors out its own component and forwards \( \{ \alpha^{N_1}, \ldots, \alpha^{N_1 N_{i-1}}, \alpha^{N_1 N_{i-1} N_{i+1}}, \alpha^{N_1 N_{i-1} N_{i+1} N_n} \} \) to \( M_n \)

• Now come the final stage again a **Broadcast** stage where \( M_n \) raises every input to power of \( N_n \) and broadcasts the resulting set \( \{ \alpha^{N_1}, \ldots, \alpha^{N_1 N_{i-1}}, \alpha^{N_1 N_{i-1} N_{i+1}}, \alpha^{N_1 N_{i-1} N_{i+1} N_n} \} \)
\[ \alpha_{N_1 N_{i-1} N_{i+1} N_n} \} \text{ to each member } M_i \ (i < n) \]

Figure 3.10: Final Broadcast stage

The best part is that there is no complication in generating the key when a new member is added. It considers the new node as \( M_{n+1} \) and re-generates the key.

Similarly, when a node \( M_n \) is deleted, the secret key is generated and the protocol is re-executed from the second stage.
Chapter 4

Analysis

4.1 Hypothesis

Most of the methods in the literature provide either only the routing protocol or the security required for the protocol. Also, most of the methods have communication overhead, key generation / re-generation time. All this result in delaying the communication and increase the wait time. Most of the methods in the literature are not suitable for large networks or for high mobile nodes. The mechanism used in this project, is suitable for large networks and also for highly mobile nodes. There is also less key generation / regeneration time and the communication overhead is not as much as some other systems. This helps in reducing the communication delay, increase the efficiency for communication, find the best path for communication between the source and the destination.

For this purpose, I have used the Multicast Ad-hoc On-demand Distance Vector (MAODV) protocol with the security provided using \textit{n-Party} Diffie Hellman mechanism. MAODV provides a better routing mechanism for multiple mobile nodes communicating with each other. The \textit{n-Party} Diffie Hellman mechanism based on the original 2-way Diffie Hellman theorem, provides a better way of securing the nodes and thus, reducing the chances of compromising the network.

4.2 Approach

For routing purpose, the MAODV protocol used is based on the original AODV protocol. The original AODV is modified to work for the multicast communication. There are total
eighteen files in AODV protocol.

1. aodv.h
2. aodv.cc
3. aodv_mcast.cc
4. aodv_mtable_aux.cc
5. aodv_mtable_aux.h
6. aodv_mtable.cc
7. aodv_mtable.h
8. aodv_packet.h
9. aodv_requeue.cc
10. aodv_requeue.h
11. aodv_rtable.cc
12. aodv_rtable.h
13. cmu-trace.cc
14. wireless-phy.cc
15. wireless-phy.h
16. node.cc
17. node.h
18. ns-mcast.tcl
4.2.1 MAODV Installations

- From the files mentioned above, the first 12 files must be put under the directory “./ns-2.26/aodv/”

- The file cmu-trace.cc must be put under directory “./ns-2.26/trace/”

- Files wireless-phy.cc and wireelss-phy.h must be put under directory “./ns-2.26/mac/”

- node.cc and node.h must be put under directory “./ns-2.26/common/”

- Copy ns-mcast.tcl to “./ns-2.26/tcl/mcast/”

Changes in Files

The contents of the files and the changes made are explained below:

- The aodv.h file needs an addition of the multicast configuration as “#define MULTICAST”.

- For UDP agents to receive multicast packets, one “#define UPPER_LEVEL_RECEIVE” needs to be added in aodv.h

- The aodvmcast.o, aodv_table.o and aodv_mtabl.aux.o must be added in the compilation list of the top-level Makefile. This make file is found under the ns-2.26 directory.

After making changes in the above mentioned files and putting the files under their respective directories, the new MAODV installation is almost done. The only step left is now creating the new MAODV implementation. This can be done by typing ”make clean” under the directory “./ns-2.26/” in the terminal window. This command will clear or remove all the obj files [23].

Once all the obj files are cleared or removed, we need to recompile NS2 with “make”. This will give the new implementation for MAODV and help in multicast routing.
4.2.2 Scenarios

The scenario for the movement of mobile nodes can be set as follows: 

```
```

For example: 

```
./setdest -n 50 -p 100 -s 60 -t 200.00 -x 500 -y 500 > scene-500x500-S60-N50
```

For testing purpose, 6 such scenarios were created:

1. x - 500, y - 500, s - 10, n - 50
2. x - 500, y - 500, s - 20, n - 50
3. x - 500, y - 500, s - 30, n - 50
4. x - 500, y - 500, s - 40, n - 50
5. x - 500, y - 500, s - 50, n - 50
6. x - 500, y - 500, s - 60, n - 50

The area selected for every experiment was same i.e. 500 x 500, also the number of nodes were kept same (50). Only the simtime was changed to make sure that everything was working as per the requirements even when the simulation went on for longer duration.

For analysis purpose, a script was written which would check the delay time when the number of attackers increase in the network. The script reads as follows: 

```
exec xgraph -bb -P -m -t AttackersVsAvg_Delay -x Attackers -y Avg_Delay Delay-rec10 Delay-rec20 -geometry 800x400
```

*xgraph* is a method used with nam to create graphs and give compared output of the simulation. It is a built-in feature of NS2. Delay-rec10 and Delay-rec20 are the output files which will show the delay in process as the number of attackers increase. Here we are comparing the results of two scenarios i.e. when simtime is 10 (Delay-rec10) and 20 (Delay-rec20). Similarly, we can compare the result of any scenarios and regarding any
two ratios. The best part of xgraph is that after plotting the graph, you can directly save it as pdf or image or any format as required. The plotting is done easily by either reading the numbers from a text file or by writing a code (for plotting when the code is running).

4.2.3 Screens

In this section, I have provided screen-shots of almost everything possible. Figure 4.1 shows the start of the simulation where all the nodes in the network are broadcasting the “HELLO” message and the groups are being created.

![Figure 4.1: Broadcast message by nodes in the network](image)

In figure 4.2, the nodes are moved from their original place. In the simulation process, the nodes are allowed to move within the network and the speed is kept to max 10 Mbps. If compared to figure 4.1, it is clear that the nodes have changed their position and have moved to some other random position. This is set with the help of the built-in function of NS2 i.e. random_mobility function.
Figure 4.2: Movement of nodes in the network

Figure 4.3 compares the time taken by both the methods for the nodes to join the network. The $n$-Party Diffie Hellman algorithm starts well by utilizing less time than its counterpart, but as the group members increase, the time taken by TDH is much less than the one implemented in this project.
*n-Party* Diffie Hellman takes much less time to regenerate the key when a node leaves the network as shown in figure 4.4

![Figure 4.4: Node leave time](image)

Time taken to regenerate the key after every interval is also much less than its counterpart. This is because, we do not re-generate the key if the nodes have communicated or the key was recently regenerated due to a node leaving or joining the network.

![Figure 4.5: Key Generation Time](image)
Due to the process used for generation and re-generation of key, the communication overhead is also reduced. This is showed in figure 4.6

![Communication Overhead](image)

**Figure 4.6: Communication Overhead**

![Latency caused by the increase in number of receivers](image)

**Figure 4.7: Latency caused by the increase in number of receivers**

Figure 4.7 shows that the latency increases a bit when the number of receivers increase
at the start, but then again with high number of nodes, the latency is not very high. This also proves the fact that method is suitable even for large networks.

4.3 Future Work

As mentioned in section 2.1, the method proposed by the authors in [7] can be modified to work for multicast routing mechanism also. Since that method is well suited for AODV i.e. the unicast mechanism, it would be helpful if this method can be used for multicast mechanism also. It uses a special technique for knowing if the mobile has high speed or low speed. Also, the author proposed two sub-methods 1.) Where the receiver decides if a particular node must be included in the reverse path and 2.) Where the nodes themselves decide if they want to be included in the communication path. Knowing the mobility of the nodes makes taking these decisions simpler.

The method implemented in this project is useful for authenticating the nodes in the network, adding them to the group but one thing that can be improvised is the Intrusion Detection Mechanism. Also, the experiments could be carried out for high number of nodes like 200 or 300.
Chapter 5

Conclusions

Mobile Ad-hoc Networks are most vulnerable and highly susceptible to attacks due to the nature of these networks. The high mobility of nodes make them change locations randomly and at varying speeds. For this purpose, the routing protocols and the security measures must be able to handle the highly mobile nodes. In this paper, I have used the most common protocol AODV and modified it for working in the multicast environment. This method is known as MAODV which helps in communication between multiple nodes. The security method used is also based on the very popular 2-party Diffie Hellman theorem and modified it to work with multicast routing protocol. This security method is known as $n$-Party Diffie Hellman theorem. This mechanism generates and re-generates the keys for group of nodes instead of only 2 nodes.

The test cases implemented proved that most of the hypothesis were true but the only place where the test cases failed was when the nodes have to join the network. The graphs provide statistical proof and also comparison between the implemented method and another Tree-based Diffie Hellman algorithm, which is close to the $n$-Party Diffie Hellman algorithm. In every case (except first) our method has outperformed the TDH algorithm.

Finally, to conclude, the implemented mechanism is better than most of the methods in the literature. The method used for route discovery is efficient and less costly. Also the time taken for generation and re-generation of keys is less when compared to other methods.
Bibliography

[1] Index of nsnam.


