

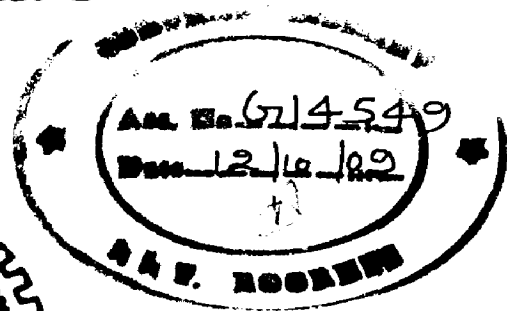
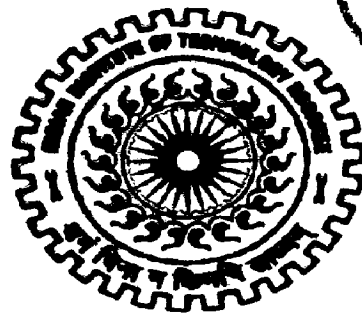
DETECTION AND MITIGATION OF WORMHOLE ATTACKS IN MANET

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*
of
MASTER OF TECHNOLOGY
in
INFORMATION TECHNOLOGY

By

TIRUMALESH. C



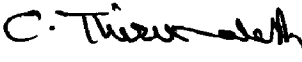
**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2009**

Candidate's Declaration

I hereby declare that the work being presented in the dissertation report titled "DETECTION AND MITIGATION OF WORMHOLE ATTACKS IN MANET" in partial fulfillment of the requirement for the award of the degree of Master of Technology in Information Technology, submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, is an authenticate record of my own work carried out under the guidance of Dr. Kum kum Garg, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee.

I have not submitted the matter embodied in this dissertation report for the award of any other degree.

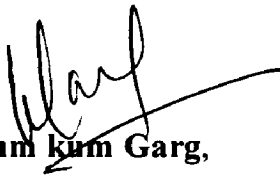
Dated: 29/6/2009
Place: IIT Roorkee.


(Tirumalesh .C)

Certificate

This is to certify that above statements made by the candidate are correct to the best of my knowledge and belief.

Dated: 29/6/2009
Place: IIT Roorkee.

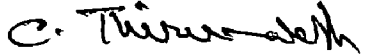

Dr. Kum Kum Garg,
Professor,
Department of Electronics and
Computer Engineering, IIT Roorkee,
Roorkee - 247667 (India).

ACKNOWLEDGEMENTS

I am thankful to Indian Institute of Technology Roorkee for giving me this opportunity. It is my privilege to express thanks and my profound gratitude to my supervisor Prof. Kum kum Garg for her invaluable guidance and constant encouragement throughout the dissertation. I was able to complete this dissertation in this time due to constant motivation and support obtained from Prof. Kum kum Garg.

I am also grateful to the staff of Network security laboratory for their kind cooperation extended by them in the execution of this dissertation. I am also thankful to all my friends who helped me directly and indirectly in completing this dissertation.

Most importantly, I would like to extend my deepest appreciation to my family for their love, encouragement and moral support. Finally I thank God for being kind to me and driving me through this journey.


(TIRUMALESH. C)

In a Mobile ad-hoc network (MANET), because of its dynamic nature, all the nodes must cooperate with each other and participate in routing. Most existing routing protocols for Ad-hoc networks rely on this cooperation. Wormhole attacks are among the most severe attacks on MANETs, in which two or more colluding attackers tunnel packets from one place to another. In particular, if attackers selectively tunnel control packets, the nodes near the attackers choose this tunnel and are prevented from using alternative routes.

In this thesis, we have proposed a multi-path routing protocol, which is a modified version of single-path Dynamic Source Routing (DSR) protocol. A multi-path routing protocol provides good defense against an attack like traffic analysis, which can be performed after a wormhole has been established. Another important effect of a wormhole attack is packet dropping. To address this problem, we have proposed a security extension to this multipath routing protocol. This extension is based on fixed size RREPLY messages. With such a multipath routing and its security extension, we can detect and mitigate in-band wormhole attacks in MANETs and Sensor networks. The proposed technique has less overhead on source and destination nodes as well as on intermediate nodes. It is also possible to isolate the attackers from the network and prevent throughput of the network from dropping.

The proposed technique has been simulated on the java based Jist-Swans simulator using various scenarios. The results are shown using the animator Inspect, a network visualization tool for ns2, using open source libraries g++ 4.2.1, gtkglext 1.2.0, and OpenGL 2.0 on a Linux based core2quad desktop.

CONTENTS

CANDIDATE DECLARATION.....	I
ACKNOWLEDGEMENTS.....	II
ABSTRACT.....	III
LIST OF TABLES.....	IV
LIST OF FIGURES.....	V
LIST OF ACRONYMS.....	VI
1. INTRODUCTION	
1.1. Introduction and Motivation.....	1
1.2. Problem statement.....	1
1.3. Organization of the thesis.....	2
2. INTRODUCTION TO MOBILE AD-HOC NETWORKS	
2.1. Overview.....	3
2.2. Dynamic Source Routing Protocol (DSR).....	5
2.3. Differences with traditional networks.....	7
2.4. Possible Attacks.....	8
3. WORMHOLE ATTACKS	
3.1. Overview.....	10
3.2. Types of wormhole attacks.....	11
3.3. Wormhole attacks effects.....	12
3.3.1. DoS attack.....	13
3.3.2. Cache poisoning.....	13
3.3.3. Sinkhole attack.....	13
3.3.4. Traffic analysis.....	13
3.4. Difficulties in detection.....	13
3.5. Existing prevention techniques.....	14
3.5.1. Packet leashes.....	14
3.5.2. Delay based.....	15
3.5.3. Statistical methods.....	16
3.5.4. Neighbor List.....	16
3.5.5. Wormhole detection based on packet dropping.....	16

3.5.6. Network visualization.....	17
3.5.7. Directional antennas.....	17
3.6. Limitations of existing techniques.....	18
4. MULTIPATH DSR PROTOCOL.....	
4.1. Introduction.....	20
4.2. Modifications to DSR.....	22
4.2.1. Route discovery at source node.....	22
4.2.2. RREQ processing at intermediate nodes.....	23
4.2.3. Example Neighborhood table updating scenarios.....	27
4.2.3.1. Scenario1	27
4.2.3.2. Scenario2.....	28
4.2.3.3. Scenario3.....	29
4.2.4. RREPLY at destination and intermediate nodes.....	30
4.3. Mitigation of wormhole attack effects.....	30
4.3.1. DoS attack.....	30
4.3.2. Cache poisoning, indirect Sinkhole attack and Traffic analysis.....	31
5. SECURITY EXTENSION.....	
5.1. Extensions to proposed MDSR.....	32
5.1.1. Fixed RREPLY messages.....	32
5.1.2. Suspicious table.....	33
5.2. Mechanism.....	33
5.3. Analysis.....	34
6. SIMULATION.....	
6.1. JiST-Swans.....	35
6.2. Simulation parameters.....	37
6.3. Simulation network.....	38
6.4. Metrics for evaluation.....	40
6.5. Analysis of Results.....	40
7. CONCLUSION.....	
7.1. Summary of work done.....	45
7.2. Suggestions for further work.....	45
7.3. Contributions.....	46

REFERENCES

APPENDIX: CODE LISTING

LIST OF TABLES

2.1 MANET vs. Traditional wired and cellular networks	7
4.1 RREQ at Source node	23
4.2 Seen-RREQ Table	24
4.3 Neighborhood Table	24
6.1 Field Parameters	37
6.2 Physical layer parameters	38
6.3 Protocols used	38

LIST OF FIGURES

2.1 Mobile Ad-hoc Network	3
2.2 RREQ broadcasting from node A to node E to discover route	5
2.3 Possible attacks in MANETs	9
3.1 MANET with two malicious nodes forming a tunnel	10
3.2 In-band wormhole attack	11
4.1 RREQ propagation in DSR	22
4.2 Flow chart of RREQ processing at intermediate nodes	26
4.3 Scenario1	27
4.4 Scenario2	28
4.5 Scenario3	29

LIST OF ACRONYMS

MANET	Mobile Ad-hoc Network
DSR	Dynamic Source Routing
MH	Mobile Host
DSDV	Destination-Sequenced Distance-Vector
OLSR	<i>Optimized Link State Routing</i>
TBRPF	Topology Dissemination Based on Reverse Path Forwarding
AODV	Ad Hoc on-Demand Distance Vector
TORA	Temporally Ordered Routing Algorithm
ABR	Associativity Based Routing
SSR	Signal Stability Routing
ZRP	Zone-Based Hierarchical Link-State Routing Protocol
RREQ	Route request Packet
RREPLY	Route reply packet
ACK	Acknowledgment packet
CSMA/CD	Carrier sense multiple access with collision detection
IP	Internet Protocol
IDS	Intrusion detection system
MDSR	Multipath DSR
TTL	Time to live
MTU	Maximum Transfer unit
MDSR-Se	Multipath DSR with security extension

1. INTRODUCTION

1.1 Introduction and Motivation

Mobile Ad-Hoc networks (MANETs), are networks which requires very minimal or no infrastructure. These networks can be formed very quickly using wireless mobile hosts (MH). MANET is one that comes together as needed, not necessarily with any support from the existing infrastructure or any other kind of fixed stations. A MANET consists of mobile platforms (e.g., a router with multiple hosts and wireless communications devices) herein referred to as "nodes" which are free to move about arbitrarily [1].

MANETs have a wide range of applications, especially in military operations, emergency and disaster relief efforts. However MANETs are more vulnerable to security attacks than conventional wired and wireless networks due to the dynamic topology, distributive and co-operative sharing of channel and power and computation constraints [2].

Wormhole attacks are one of the most powerful attacks in MANETs since they involve the cooperation between two or more malicious nodes that participate in network routing [3]. One attacker, say node M1, captures routing traffic at one point of the network and tunnels them to another point in the network, say node M2. Node M2 then selectively injects tunneled traffic back into the network. The connectivity of the nodes that have established routes over the wormhole link is completely under the control of the colluding attackers [4].

1.2 Problem statement

The main difference between MANET and other wireless networks occurs at the network layer. This is because in a MANET all the nodes participate in routing. In order to widely deploy Mobile Ad hoc networks, a good routing protocol is very important. In this thesis work, we analyze the effects of different wormhole attacks and propose a multipath routing protocol with an extension to detect and mitigate wormhole attacks.

1.3 Organization of thesis

Including this introductory chapter, this report contains 7 chapters

In chapter 2 we present an overview of MANET and their routing protocols and also discuss Dynamic Source Routing Protocol and various attacks that are possible on MANET.

In chapter 3 we discuss the Wormhole attacks in detail. We also discuss why the wormhole attack is hard to detect and present existing techniques to mitigate wormhole attacks, and also their limitations.

In chapter 4 we present our proposed modifications to DSR to make it multi-path and analyze the modifications.

In chapter 5 we discuss an extension to the proposed multipath DSR to detect and mitigate in-band wormhole attacks.

Chapter 6 presents the simulation parameters and scenarios and also the simulation results obtained in detail.

Chapter 7 concludes this thesis by giving limitations of work done and suggestions for further work.

2. Introduction to Mobile Ad-hoc Networks

2.1 Overview

A Mobile ad hoc network (MANET) is a collection of wireless mobile nodes, dynamically forming a temporary network without the use of any existing network infrastructure or centralized administration. The nodes are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a stand-alone fashion, or may be connected to the Internet. Multi hop, mobility, large network size combined with device heterogeneity, bandwidth, and battery power constraints make the design of adequate routing protocols a major challenge for MANETs [1]. Figure 2.1 shows a typical MANET.

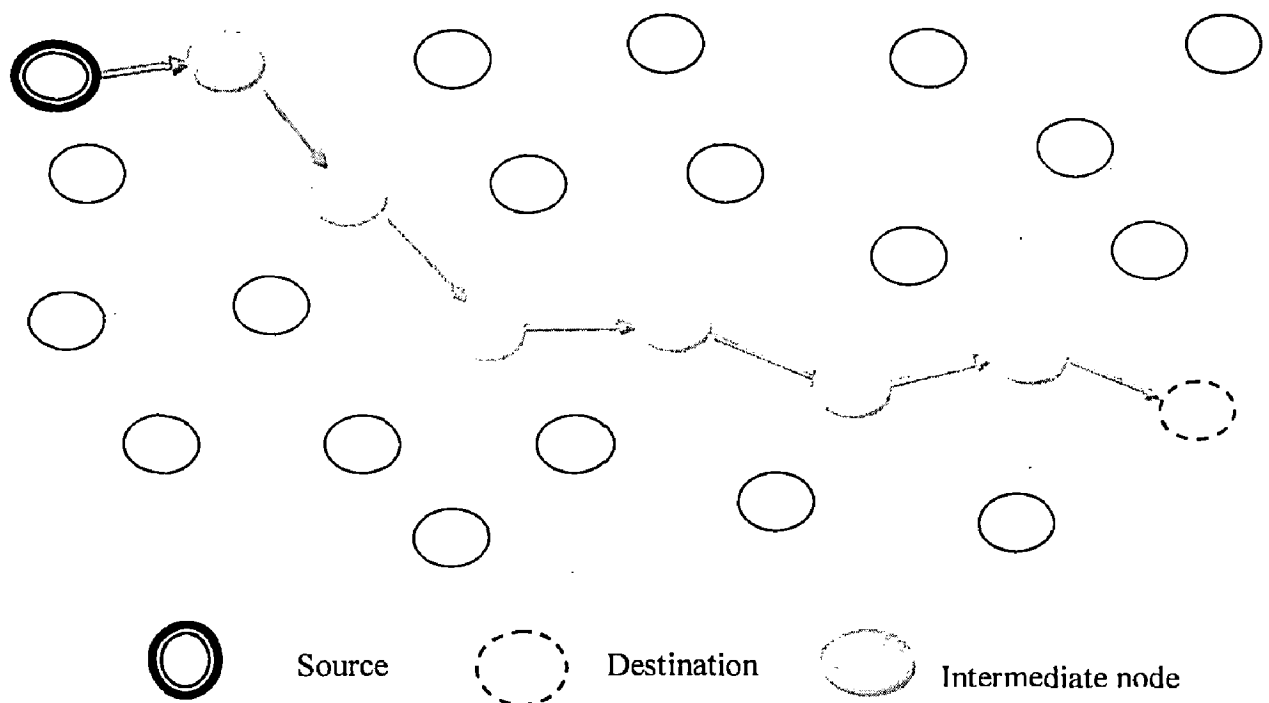


Figure: 2.1 Mobile Ad-hoc Network

In Figure 2.1, source node is shown transmitting the data to a destination node with the help of intermediate nodes. All the intermediate nodes must cooperate with the sender

and receiver to transfer data. This means that each intermediate node has full access to the packets flowing through it.

There are mainly 3 types of routing protocols [15] in MANETs.

1. Proactive routing protocols
2. Reactive routing protocols and
3. Hybrid routing Protocols

The Proactive routing protocol always maintains a route to each and every node in the network. Route creation and maintenance are performed through both periodic and event-driven messages. Various proactive protocols are Destination-Sequenced Distance-Vector (DSDV), Optimized Link State Routing (OLSR), and Topology Dissemination Based on Reverse Path Forwarding (TBRPF). Proactive routing protocols suffer from scalability because of periodic messages [1], which needs to be sent even if the network is static. These messages consume a lot of bandwidth and power.

The Reactive routing protocol reduces overhead as the route between two nodes is discovered only when it is needed. There are different reactive routing protocols such as Dynamic Source Routing (DSR) [2], Ad Hoc On-Demand Distance Vector (AODV), Temporally Ordered Routing Algorithm (TORA), Associativity Based Routing (ABR), and Signal Stability Routing (SSR). Reactive routing protocols mainly use route discovery and route reply messages to find routes between two nodes whenever it is necessary. Route discovery messages are broadcast.

In addition to proactive and reactive routing protocols, another class of unicast routing protocols that can be identified is hybrid protocols. The Zone-Based Hierarchical Link-State Routing Protocol (ZRP) is an example of a hybrid protocol that combines both proactive and reactive approaches, thus trying to bring together the advantages of the two approaches.

In this thesis work we use DSR as our routing protocol because of its scalability, low overhead and the ability to adapt to mobility.

2.2 Dynamic Source Routing protocol (DSR)

The Dynamic Source Routing protocol (DSR) [2] is a simple and efficient routing protocol designed specifically for use in multi-hop, wireless ad hoc networks of mobile nodes. DSR allows the network to be completely self-organizing and self-configuring, without the need for any existing network infrastructure or administration. The protocol is composed of the two main mechanisms of "Route Discovery" and "Route Maintenance", which work together to allow nodes to discover and maintain routes to arbitrary destinations in the ad hoc network.

All aspects of the protocol operate entirely on demand, allowing the routing packet overhead of DSR to scale automatically to only what is needed to react to changes in the routes currently in use. Other advantages of the DSR protocol include easily guaranteed loop-free routing, operation in networks containing unidirectional links, use of only "soft state" in routing, and very rapid recovery when routes in the network change. The DSR protocol is designed mainly for mobile ad hoc networks of up to about two hundred nodes and is designed to work well even with very high rates of mobility [2].

When some source node originates a new packet addressed to some destination node, it places in the header of the packet a "source route" giving the sequence of hops that the packet is to follow on its way to the destination. Normally, the sender will obtain a suitable source route by searching its "Route Cache" of routes previously learned; if no route is found in its cache, it will initiate the Route Discovery protocol to dynamically find a new route to this destination node.

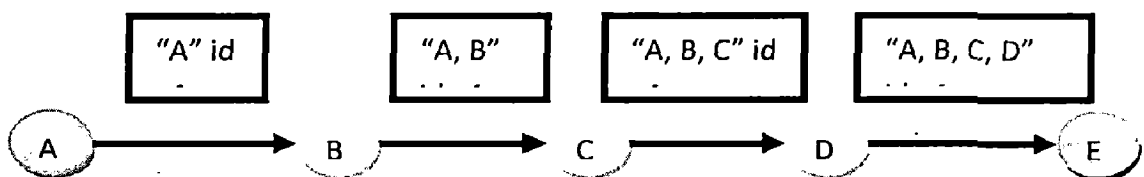


Figure 2.2 RREQ broadcasting from node 'A' to node 'E' to discover route

For example, in Figure 2.2, node A is attempting to discover the route to node E, A broadcasts a packet called RREQ. All the nodes that are within the transmission range of

A receive this broadcast packet. Node sends route reply message if it is the intended destination node, otherwise it checks whether it has already seen a route request from this source with the same ID. If yes, it drops the packet, otherwise it forwards it. In the above example, nodes B, C, D forward the RREQ packet. Node E sends a route reply (RREPLY) by placing the route information that is identified with RREQ message.

When originating or forwarding a packet using a source route, each node transmitting the packet is responsible for confirming that data can flow over the link from that node to the next hop. For example, in the situation shown in Figure 2.2, node A has originated a packet for node E using a source route through intermediate nodes B, C, and D. In this case, node A is responsible for the link from A to B, node B is responsible for the link from B to C, node C is responsible for the link from C to D, and node D is responsible for the link from D to E. The nodes fulfill this responsibility either using built-in acknowledgments like MAC layer CTS (clear to send) or using promiscuous mode if available. Otherwise the node can explicitly request DSR software to send an ACK message. If a node fails to receive an ACK for a fixed period of time, it sends a route error message back to the source, stating that the link is broken. In this case, the source chooses another path from the cache or broadcasts a RREQ message.

In addition to the basic mechanism DSR RFC [2] mentions the following optimizations.

1. Caching overheard Routing information
2. Replying RREQs with cached routes.
3. RREQ by hop limits
4. Packet salvaging
5. Queued packets destined over a broken link
6. Automatic route shortening
7. Increased spreading of route error messages
8. Flow state extension

In our thesis work we propose a multi-path routing protocol based on DSR. DSR by itself gives the source node options in route paths. With our proposed modifications, the number of available distinct paths will increase.

2.3 Differences with traditional networks

Different types of devices are used as nodes in ad hoc networks. For example, we can have PDA-like devices, mobile phones, two-way pagers, sensors, or laptop computers with different capabilities in terms of maximum transmission power, energy availability, mobility patterns, and QoS requirements. Thus Ad hoc networks are generally heterogeneous in terms of nodes and services offered. In terms of energy and power, one has to consider not only node heterogeneity, but also varying communication ranges, such as sleeping or active modes and the existence of energy supplies. Ad hoc networks also raise new issues concerning security and privacy [5].

Traditional and Cellular networks	Mobile ad-hoc networks
Infrastructure networks	Infrastructure less networks
Fixed pre allocated cells and base stations	No base station and rapid deployment
Routing decisions taken by limited number of trusted nodes.	Routing decisions taken by all the nodes in the network.
Static backbone network topology	Highly dynamic network topology
Stable connectivity	Irregular connectivity
Detailed planning before backbone network installed	Ad-hoc networks automatically forms and adopt to the changes
High setup cost	Cost-effective
High setup time	Very less setup time

Table 2.1: MANET vs. Traditional wired and cellular networks

Table 2.1 lists some of the main differences of MANETs with wired and cellular networks.

Wireless multiple accesses can be categorized into random access (e.g., CSMA and CSMA with Collision Detection [CSMA/CD]) and controlled access. Random access is suitable for ad hoc networks because of lack of infrastructure support [1].

The main functionalities of networking protocols need to be redesigned for MANETs. Current solutions like Mobile IP, generally adopted to manage mobile terminals in infrastructure networks, are inadequate and new approaches need to be found for mobile management.

Moreover, very minor changes are required in the transport layer. The main changes occur in the network layer.

2.4 Possible attacks

Figure 2.3 shows different attacks possible on MANETs. Among different attacks, attacks on MAC layer are common to all wireless networks. Attacks on transport layer and application layer are common to all wired and wireless networks.

Attacks on routing protocols are unique to MANETs. Among different attacks on network layer colluding attacks are hard to detect. In colluding attack two or more attackers cooperate to launch the attack.

Some of the attacks can be launched by outsider nodes and some of the attacks can be launched only by insider nodes. Attacks in which attackers does not require to alter packets can be launched by outsider nodes. This is possible because, MANETs operate on wireless medium, in which every node can overhear the transmission if the node is within the transmission range of transmitting node.

In this thesis work we concentrate on wormhole attacks. These are colluding attacks and can be launched by outsider nodes without altering the packet. Wormhole attack can be launched against all the routing protocols of MANET. In the next chapter we study more about the wormhole attacks.

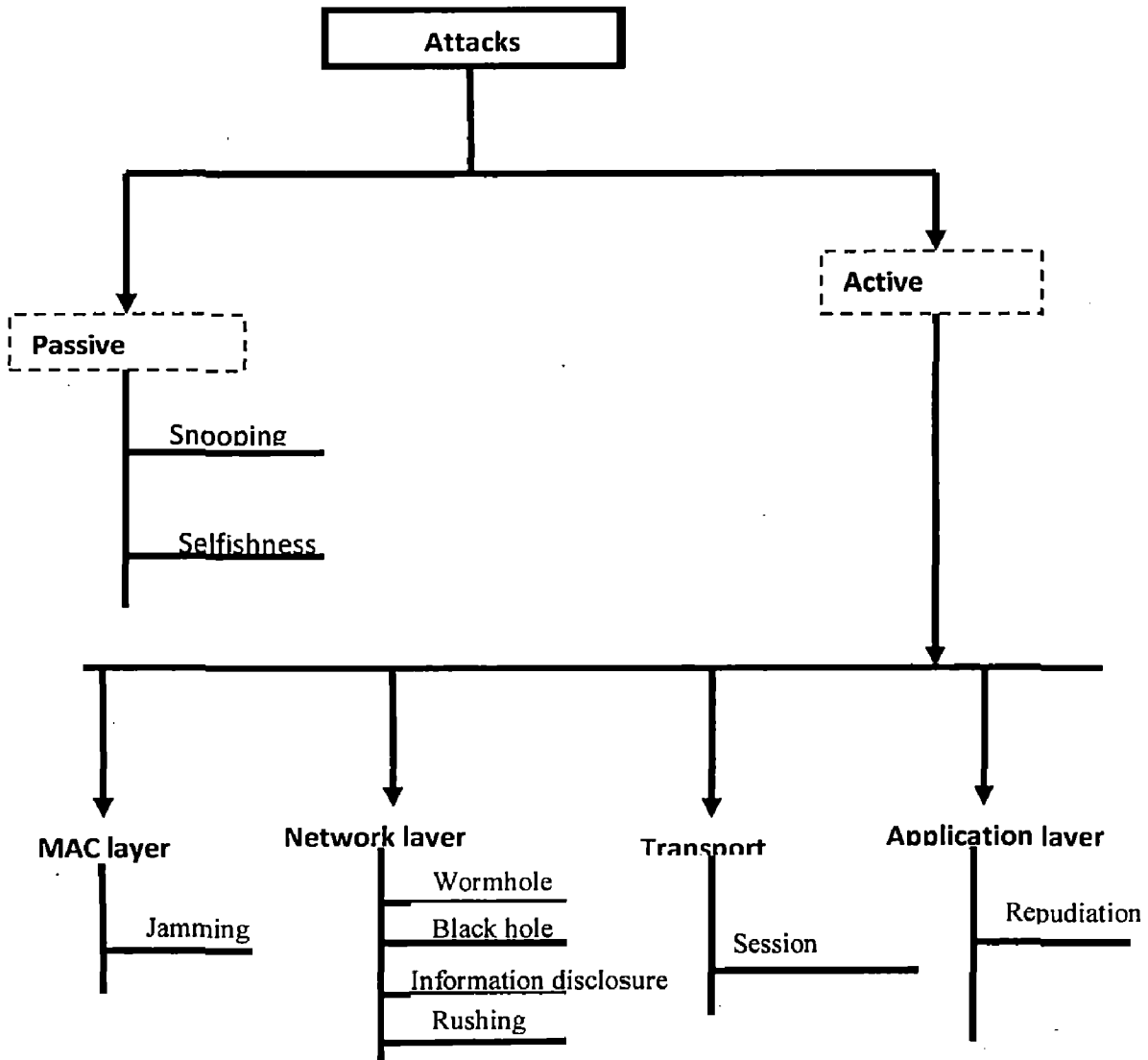


Figure 2.3: Possible attacks in MANETs

3. WORMHOLE ATTACKS

3.1 Overview

Among the different attacks on MANETs, colluding attacks, where two or more nodes cooperate to execute the attack, are more difficult to detect. Among colluding attacks, a wormhole attack is a particularly severe attack on MANET routing, where two or more attackers, connected by a high-speed off-channel link or logical tunnel through other nodes [6]. These attackers then record the wireless data they overhear, forward it to each other, and replay packets at the other end of the network. By replaying valid network messages at improper places, wormhole attackers can make far apart nodes believe they are immediate neighbors, and force all communications between affected nodes to go through them.

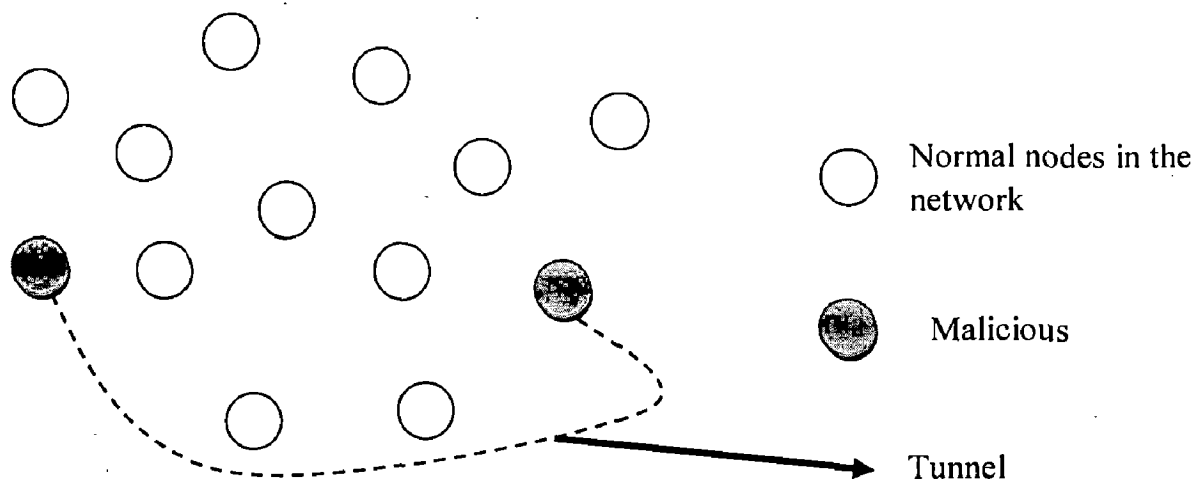


Figure 3.1: MANET with two malicious nodes forming a tunnel

Figure 3.1 shows a MANET with two attacker nodes. If the attacker nodes are placed well in the network then they can attract more traffic. If used well, an off-link tunnel may help other nodes in routing the information easily by saving power consumption and delay. But this keeps the attacker in an excellent position in the network where he can hear the traffic and analyze it or drop critical information to distract the other nodes in the network.

3.2 Types of wormhole attacks

Based on the wormhole tunnel, two different types of wormhole attacks are possible.

1. In-band wormhole attacks
2. Out-band wormhole attacks

In in-band wormhole attack, the wormhole tunnel is established through other nodes in the network.

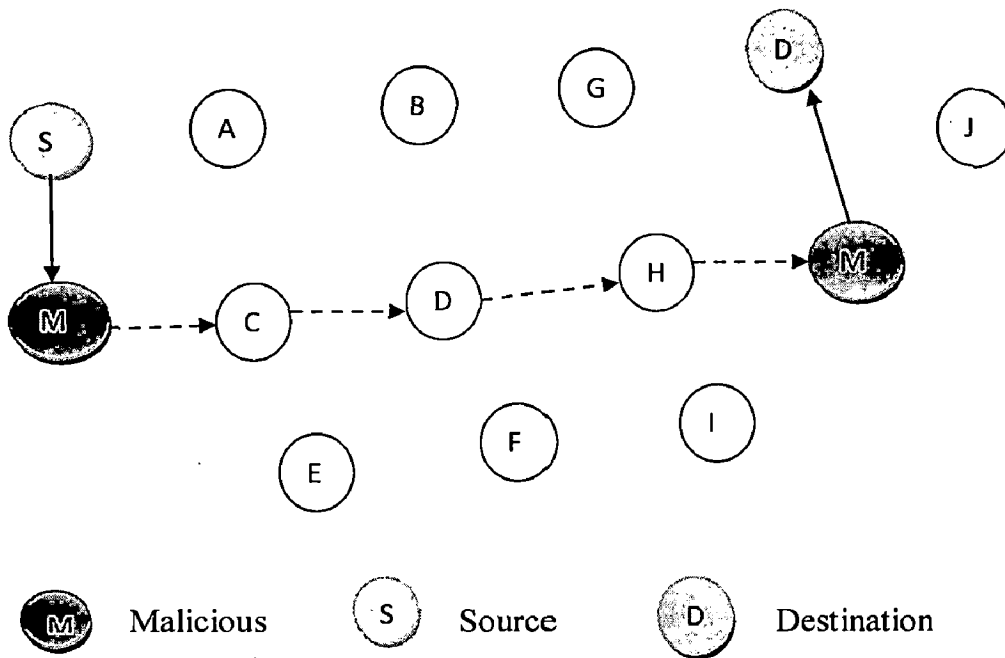


Figure 3.2: In-band wormhole attack

In Figure 3.2, S sends the data packets to D through the malicious nodes M1 and M2. But nodes S and D assume that M1 and M2 are direct neighbors. In fact, the transmission between M1 and M2 goes through some other nodes. This is possible in DSR if node M1 tunnels all the RREQ packets to node M2 and node M2 rebroadcasts the packets. If any RREPLY comes to node M2, it tunnels it back to M1 and M1 sends it to source node S. Node S assumes the path length to be 2 hops. So it selects this path S-M1-M2-D node sends the data packets. M1 can analyze this traffic or it can drop all the data packets.

In out-band wormhole attack; the tunnel is established using a dedicated wired connection between the attackers or using a high frequency wireless transmission. In out-band wormhole attacks the attackers are actual neighbors. They do not need to depend on other nodes for their communication.

In both in-band and out-band wormhole attacks, it is possible to hide or expose the malicious nodes.

In hidden wormhole attack, after receiving the RREQ packet, the malicious node simply tunnels it without altering the packet. So the source node does not know that M1 is in the path. In Figure 3.2 with hidden wormhole attack, node S assumes that node D is its direct neighbor.

In exposed wormhole attack after receiving the RREQ packet the malicious nodes adds itself to the end of the list and tunnels the packet to the other malicious node. The other node also appends itself to the list and broadcasts the packet. In Figure 3.2, with exposed wormhole attack, node S assumes the path to D as S-M1-M2-D.

3.3 Wormhole attack effects

To form a wormhole tunnel, an attacker places the first malicious node near the destination node and the second malicious node near the source node. This placement of wormhole attack nodes has severe effects on route discovery process.

In case of out-band wormhole attacks due to the use of high frequency bands for communication; assist the wormhole nodes to propagate the RREQ speedily through the wormhole tunnel to destination, resulting RREQ through other legitimate paths to be discarded. After receipt of RREQ, destination processes RREQ and replies with RREPLY message.

Now the type of attack formed depends on the processing of received RREPLY from destination node at wormhole attacker node near destination. Following attacks could be possible.

3.3.1 DoS attack

If the received RREPLY from destination node is discarded at wormhole node, then routes are not discovered at the source node and resulting in repeated route discovery process at source node, forming a Denial of service attack [3].

3.3.2 Cache poisoning

If the received RREPLY from destination node are tunnelled back to source node through formed wormhole tunnel, then a shorter path through the wormhole tunnel is recorded by source node and nodes surrounding source node. This shorter path between source and destination nodes may be communicated to other nodes of network using an optimisation of DSR called Automatic Route Shortening. Resulting in an attack called cache poisoning [4].

3.3.3 Sinkhole attack

After successful formation of a path through the wormhole tunnel, between source and destination nodes, the attacker node may selectively drop the data packets, resulting in an indirect sinkhole attack [7].

3.3.4 Traffic analysis

After successful formation of a path through the wormhole tunnel, between source and destination nodes, the attacker node gets to see every packet destined to other nodes. Hence they can analyse the traffic [7].

In our thesis work, we will address all these different effects of wormhole attacks.

3.4 Difficulties in detection

Wormhole attack is a colluding attack, which means two or more malicious nodes cooperate to execute the attack. So it is not possible to detect this attack with ordinary intrusion detection systems (IDS) like pathrator or ex-pathrator [8]. Pathrator requires all the nodes to be in promiscuous mode, when node A propagates a packet to node B, it listens to B to know whether it is forwarding the packet or not. If B is not forwarding the

packet, it is treated as a malicious node. While analyzing the traffic, wormhole attacks do not drop packets. Thus IDS cannot detect the attack. Furthermore even it is possible to drop packets, the attacker need to drop packets at the malicious node which is close to the destination.

It is possible to execute the attack even when all the information, including routing information, is encrypted or digitally signed. This can be achieved because even outsider nodes can create out-band wormhole attack without altering the packets, simply by tunneling the control packets, If it is difficult to identify control packets they tunnel all packets. This is possible because MANETs operate in an open medium, where any node can hear the transmission, if the transmitting node is within its range.

It is practically not possible to propose a completely software based solution for detecting out-band wormhole attacks [2]. This is because, in out-band worm hole attacks, the *attackers are actual neighbors, and they use other dedicated medium for communication* between themselves. It is not possible for other nodes to detect this communication.

3.5 Existing prevention techniques

3.5.1 Packet leashes:

A leash is any information that is added to a packet designed to restrict the packet's maximum allowed transmission distance. In [2] [3], authors proposed two different types of leashes geographical leashes and temporal leashes. A geographical leash ensures that the recipient of the packet is within a certain distance from the sender. A temporal leash ensures that the packet has an upper bound on its lifetime, which restricts the maximum travel distance, since the packet can travel at most at the speed of light. Either type of leash can prevent a wormhole attack, because it allows the receiver of a packet to detect if the packet traveled further than the leash allows.

To construct a geographical leash, in general, each node must know its own location and all nodes must have loosely synchronized clocks. When sending a packet, the sending node includes in the packet its own location, p_s , and the time at which it sent the packet t_s . If the clocks of the sender and receiver are synchronized to within $\pm\Delta$, and v is an upper

bound on the velocity of any node, then the receiver can compute an upper bound d_{sr} on the distance between the sender and itself.

To construct a temporal leash, in general, all nodes must have tightly synchronized clocks, such that the maximum difference between any two nodes' clocks is Δ . The value of the parameter Δ must be known by all nodes in the network, and for temporal leashes, generally must be on the order of a few microseconds or even hundreds of nanoseconds. To use temporal leashes, when sending a packet, the sending node includes in the packet, the time at which it sent the packet, t_s ; when receiving a packet, the receiving node compares this value to the time at which it received the packet, t_r . The receiver is able to detect if the packet traveled too far, based on the claimed transmission time and the speed of light.

3.5.2 Delay based:

In [5], the authors proposed a mechanism which uses link delays to identify wormholes. This detection mechanism works for proactive routing protocols like OLSR. To maintain network topology, all the nodes in the network broadcast periodical HELLO messages to their neighbours. After receiving the neighbour's reply for HELLO messages, attackers tunnel this HELLO messages to construct wormholes. After receiving all reply messages, a node calculates the delay. If the delay is longer this link is identified as a suspicious link. For all suspicious links, a node sends a Probe packet. Once a node receives a probe packet, it responds to it by stopping all other transmissions. Using this information a node finds wormholes.

In [9], the authors proposed a per hop delay based technique to identify a wormhole attack. Where, each node sends a RREQ message and waits for RREPLY message before sending the actual data. All the intermediate nodes attach a time stamp to it; the destination copies the path and time stamps and sends back to source. The source calculates the delays and suspects the links with large delays.

3.5.3 Statistical methods:

In [10], the authors proposed a statistical mechanism for detecting wormholes. This mechanism works for reactive routing protocols in which each node sends RREQ before sending the data to a node if it does not have the route to it, and the destination sends route reply messages. After receiving all the route reply messages, the source calculates the frequency of each link. If the difference between first and second frequency is very high, then the first link is a wormhole link.

In [6], each node collects information from 3-hop neighbors and sends it to the coordinator. The coordinator analyses and takes decisions based on this information.

3.5.4 Neighbor List:

Khalil et al [11] [12] propose a protocol for wormhole attack discovery in static networks, called LiteWorp. In LiteWorp, nodes obtain full two-hop routing information from their neighbors. While in a standard ad hoc routing protocol, the nodes usually keep track of who their neighbors are; in LiteWorp they also know who the neighbor's neighbors are. They can take advantage of two-hop, rather than one-hop, neighbor information. This information can be exploited to detect wormhole attacks.

After authentication, nodes do not accept messages from those they did not originally register as neighbors. Also, nodes observe their neighbor's behavior to determine whether data packets are being properly forwarded by the neighbor, - a so-called 'watchdog' approach. LiteWorp adds an interesting wormhole-specific twist to the standard watchdog behavior: nodes not only verify that all packets are forwarded properly, but also make sure that no node is sending packets it did not receive (as would be the case with a wormhole)

3.5.5 Wormhole detection based on packet dropping:

Several researchers worked on the wormhole attack problem by treating a wormhole as a misbehaving link. In such approaches, a wormhole attack is not specifically identified. Rather, the wormhole's destructive behaviour is mitigated.

Baruch [13] and Chigan [8] use link rating schemes to prevent black hole and wormhole attacks. They rely on authenticated acknowledgements of data packets to rate links: if a link drops packets, the acknowledgements do not get through; such a link is rated low and avoided in future.

3.5.6 Network visualization:

Wang and Bhargava [14] introduce an approach in which network visualization is used for discovery of wormhole attacks in stationary sensor networks. In their approach, each sensor estimates the distance to its neighbors using the received signal strength. During the initial sensor deployment, all sensors send this distance information to the central controller, which calculates the network's physical topology based on individual sensor distance measurements. With no wormholes present, the network topology should be more or less flat, while a wormhole would be seen as a 'string' pulling different ends of the network together.

3.5.7 Directional Antennas:

Directional antennas have been extensively studied in the general literature [16]. When directional antennas are used, nodes use specific 'sectors' of their antennas to communicate with each other. Therefore, a node receiving a message from its neighbor has some information about the location of that neighbor. It knows the relative orientation of the neighbor with respect to itself. This extra bit of information makes wormhole discovery much easier than in networks with exclusively Omni directional antennas.

In [16], Hu and Evans propose a solution to wormhole attacks for ad hoc networks in which all nodes are equipped with directional antennas. Wormholes introduce substantial inconsistencies in the network, and can easily be detected.

In SERLOC [7], Lazos et al use a slightly different approach. Here, only a few nodes need to be equipped with directional antennas, but these nodes also have to be location-aware. These nodes then send out localization beacons, based on which, regular network nodes determine their own relative locations.

3.6 Limitations of existing techniques

The various existing techniques studied in the previous section have many limitations, these are discussed below

- The level of time synchronization required for temporal leases [3] (on the order of Nanoseconds) entails the use of specialized hardware not currently practical in wireless ad hoc networks. In sensor networks, such level of synchronization is impossible [13] at this time. Temporal packet leases thus offer an elegant but not practical solution to Wormhole attacks.
- Geographical leases work fine when GPS coordinates are practical and available. However, modern GPS technology has significant limitations that should not be overlooked. While the price of GPS devices is going down, it remains substantial. Besides, GPS is somewhat of a nuisance for personal laptops. Also, while it is possible to achieve GPS precision of about 3m with state-of-the-art GPS devices [13], consumer-level devices do not get (and do not require) this level of resolution. Finally, GPS systems are not versatile, as GPS devices do not function well inside buildings, under water, in the presence of strong magnetic radiation, etc.
- RTT-based approaches [5][10][17] are incompatible with the standard 802.11 MAC protocol. Thus, on top of possibly requiring specialized hardware, these approaches also prohibit the use of the standard MAC protocol, and, overall, do not seem practical.
- Some approaches [13][8] are geared towards discovery and prevention of only one kind of wormhole behavior: packet loss. Wormholes can do much more than that. They can send packets out of order, confuse location-based schemes, or simply aggregate packets for traffic analysis. Even the distortion of topology information that a wormhole introduces can be a significant problem in particular networks.

- The methods proposed by [4] and [7] are both viable, and could be easily applied to networks that use directional antennas. Currently, such networks are mostly in research stage, and their future prominence is not clear [14].
- Methods given in [11][12] are interesting, but would not work at all in a scenario where node mobility is a factor. Since a node's neighbors are determined and detected only once [18], and the packets from non-neighboring nodes are rejected, no node movement is allowable. Therefore, these are applicable to static networks only.
- Methods given in [10] [17] are somewhat limited in scope as they apply only to routing protocols that are both on-demand and multipath. Non-multipath on-demand protocols do not provide enough information for the determination of link frequencies. While on-demand routing protocols keep complete information about routes they discover, proactive ones rely on next-hop information, which does not allow the calculation of link frequencies

Overall, while a number of techniques have been proposed to combat wormhole attacks, an easy lightweight solution is still lacking. The following are some of the issues we have addressed in this work.

1. No solution addressed DOS attack created by out-band wormhole in reactive protocols.
2. All the solutions that are proposed are applicable for either proactive or reactive protocols. We need a solution works for both proactive and reactive protocols.
3. All the solutions work for either multi-hop, single path or multi-hop, multipath protocols. We need a solution works for both types of protocols.
4. Most of the solutions use additional hardware for maintaining time synchronization or to know the exact position of the node. Our solution should do not require any additional hardware.
5. In this work we do not assume any restrictions on node mobility and on the topology of the network.
6. We need a solution that works for both hidden and exposed wormhole attacks.

4. MULTIPATH DSR PROTOCOL

4.1 Introduction

In this chapter we present our proposed multi-path routing protocol based on DSR, called Multipath DSR (MDSR). The main objective of the proposed MDSR is to minimize the effects of in-band and out-band wormhole attacks. In the next chapter, we propose a security extension to MDSR to detect and isolate hidden and exposed in-band wormhole attacker nodes.

Multipath routing protocols, due to the existence of multiple routes between communicating nodes, have the following advantages over unicast protocols [19].

1. Increases the aggregated throughput of the network
2. Achieves load balancing and resource preservation.
3. Good resistance against attacks like repudiation, eavesdropping and traffic analysis.

Our proposed MDSR, like any other existing multi-path routing protocol, based on reactive routing protocols like DSR and AODV has following three phases.

1. Finding the routes to destination
2. Selecting the route to use
3. Maintaining the routes

Before stating the actual changes needed to DSR, let us closely observe the RREQ propagation in DSR, to point out the loopholes that are exploited by wormhole attacker to form a wormhole tunnel with the help of Figure 4.1.

Suppose node 0 broadcasts a RREQ packet to node 30 with RREQ ID=1. All the nodes in the transmission range of 0 will receive this RREQ. From Figure 4.1 nodes 1, 8 and 9 receives the RREQ. Since these nodes are not the intended recipients and this is the first RREQ they are seeing from node 0 with this RREQ ID, they will forward this RREQ to

their neighbor nodes 2, 10, 18, 16, 17. This process continues until the RREQ reaches the destination node 30.

Out-band wormhole attack uses the fact that all intermediate nodes forward a RREQ packet only once if it is not seen earlier. If some parts of the network are congested or highly mobile or out-band wormhole attack is launched at the time of RREQ propagation, we observed two problems. They are described as follows

1. RREQ from the non congested paths arrive quickly compared to the paths with congested or highly mobile or out-band wormhole formed areas of network. This results no paths through congested or highly mobile area. It is ok, if the area is congested or under high mobility for long time. But if the area recovered quickly and if through that area, a shorter path exists, and then the shorter path may be not utilized.
2. The 1-hop neighbors of destination after receiving first RREQ propagate to destination and also among them. Then this results in discarding the RREQ packets from most of the other paths to the destination node. This problem can be visualized using Figure 4.1. Suppose among the 1-hop neighbors of node 30, node 22 receive RREQ early, and it forwards this RREQ to nodes 21, 23, 30, 29 and 31. All the nodes receive this RREQ and drop all the consequent RREQ packets. This will result formation of only 1 path between source and destination.

In our proposed MDSR, we took care to overcome above problems in forwarding RREQ in the process of finding the routes to destination, as these problems are exploited by wormhole attacker to form wormhole tunnels.

Selecting the route to use depends on the needs of the application. If the application requires robustness, it can send same packet through multiple paths. If the application requires load balancing among the network it can choose different paths to send data packets.

Maintaining routes is same as normal DSR.

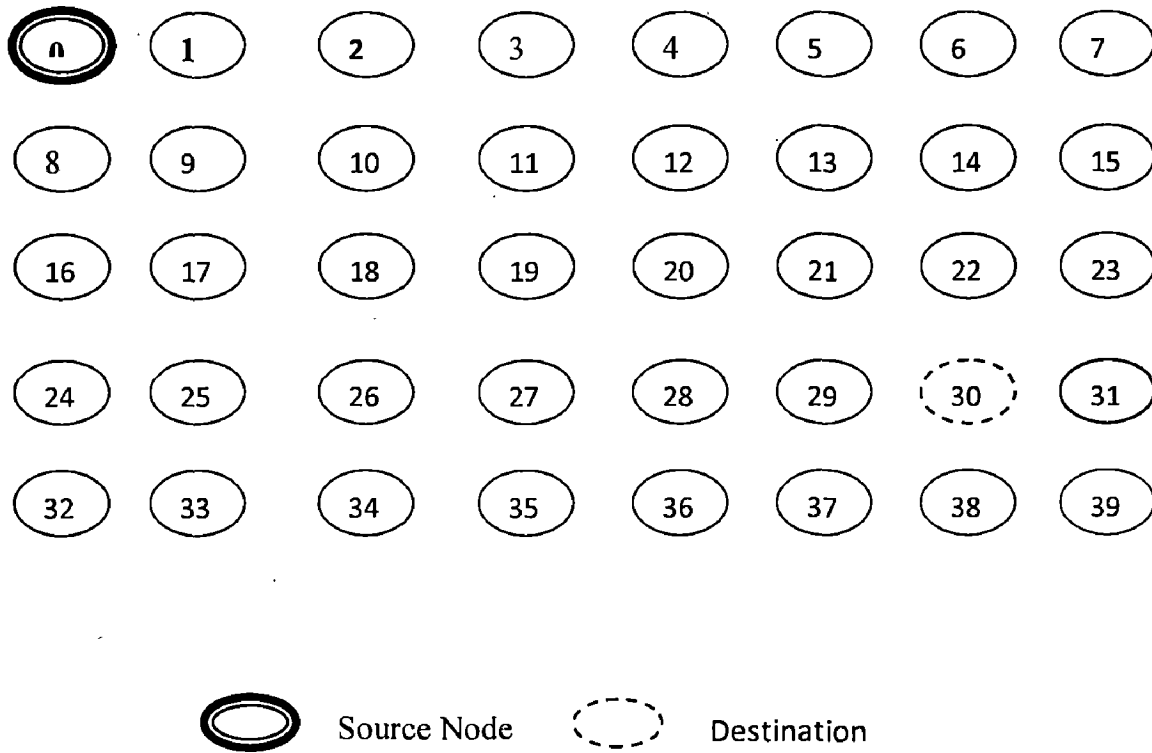


Figure 4.1 RREQ propagation in DSR

4.2 Modifications to DSR

To address the above problems, we proposed following modifications.

1. Route discovery at source node.
2. Processing and forwarding of RREQ at the intermediate node.
3. RREPLY at destination and intermediate nodes.

4.2.1 Route Discovery at source node

- When a source node wants to transmit a data packet to a destination node, to which it does not have a known path, it initiates the route discovery process by broadcasting a RREQ packet.
- After broadcasting the RREQ packet, the source node sets a timer whose time period T is determined by using formula given below.

$$T = \frac{2 * R}{V} + C$$

Where R = Maximum Transmission range.

V= Speed of the wireless signal.

C= Constant value, $R/2*V$ as used in our simulations.

The time value of timer indicates the time needed to receive a RREPLY from 1-hop neighbor.

- Acceptance of RREPLY depends on the arrival time and the path length between source and destination node.
- The possible arrivals for RREPLY packets could be as follows.
 - case i. Arrival of RREPLY before timer expires and
 - a. if path length is equal to 1, then accept the RREPLY.
 - b. else reject RREPLY, as RREPLY received may be a forged reply from a malicious node.
 - case ii. Arrival of RREPLY after the timer expires and
 - a. if path length is equal to 1, then reject the RREPLY, because it may be a RREPLY received from a malicious wormhole tunnel.
 - b. else path length greater than 1, accept the RREPLY, RREPLY has been travelled along the path containing only legitimate nodes from destination to source.

This is shown in Table 4.1

	With in time period of Timer	After the Timer expires
Path length = 1	Accept, as RREPLY received satisfies the 1-hop Roundtrip time.	Reject, May be from WORMHOLE TUNNEL.
Path length > 1	Reject. May be a forged reply.	Accept (path having only legitimate nodes).

Table 4.1 RREQ at source node

4.2.2 RREQ processing at intermediate nodes

The main purpose of this modified RREQ processing at intermediate nodes is to make each intermediate node forward the RREQ more than once if it is from a congested area. And also make all 1-hop neighbors of destination forward the RREQ packets received

through different paths by discarding the RREQ coming from other 1-hop neighbors of destination. In order to achieve this each node in MANET maintains two Tables, Seen-RREQ and the other is Neighborhood table.

Seen-RREQ Table is the same as RREQ Table in DSR, which consists of source node, RREQ ID, TTL value. For example Table 4.2 shows the Seen-RREQ Table at node 1 in Figure 4.1, after node 1 broadcast RREQ packet that is received from node 0.

Source Node	RREQ ID	TTL
0	1	225

Table 4.2 Seen-RREQ Table

Table 4.3 shows a sample Neighborhood Table at node 22 in Figure4.1.

Destination ID	1-hop neighbors
30	29
30	21

Table 4.3 Neighborhood Table

The procedure for filling the entries in both tables and taking decision about forwarding or discarding RREQ packet at intermediate nodes is given in detail in the following steps.

- When RREQ is received at an intermediate node, it needs to know whether destination is within 1-hop neighborhood.
- To confirm the presence of the destination in 1-hop neighborhood, the following procedure is followed.
 1. Each intermediate node after receiving the RREQ packet, delays the forwarding of RREQ by time equal to 1-way propagation delay calculated using formula given below.

$$T = \frac{R}{V} + C$$

Where, R = Maximum Transmission range.

V= Speed of the wireless signal.

C= constant value, as used R/2*V in our simulations.

2. If the intermediate node overhears a RREPLY with Hop count equal to 1, before timer expires, the intermediate node and the node that forwarded the RREQ are in 1-hop neighborhood destination.
3. If the node that forwarded the RREQ is not in the 1-hop neighborhood of the destination, the intermediate node forwards RREQ's as follows
 - a. If the path in the RREQ except source contains a node in the neighborhood table with destination ID equal to destination of RREQ, then discard the RREQ, as it is already forwarded by one of the 1-hop neighbors of destination.
 - b. Else if the received RREQ has no entry in the table, then the node follows the source node RREQ forwarding procedure described in section 4.2.1 and adds an entry to the Seen-RREQ table.
 - c. Else if received RREQ has higher TTL value than in the stored entry, then it updates the TTL of store entry and forward as described in section 4.2.1
 - d. Else discards the RREQ, as it is already seen.
4. If the intermediate node and node that forwarded the RREQ are in the 1-hop neighborhood of the destination node, the intermediate node forwards RREQ's as follows
 - a. If the received RREQ has no entry in the Neighborhood table, add an entry into the table with destination ID equal to destination node of RREQ and 1-hop neighbor as the last hop node from which it received this RREQ. Discard the RREQ.
 - b. Else discard the RREQ.

Figure 4.2 gives the flow chart for processing RREQ at intermediate nodes. By delaying RREQ propagation by R/V time, 1-hop neighbors of the destination learn about other 1-hop neighbors of the destination and add them to neighborhood table. With the help of this neighborhood table, 1-hop neighbors of destination discard the RREQ packets from other 1-hop neighbors of destination. With the help of TTL value and RREQ ID in the seen-RREQ table, each intermediate node forwards RREQ packet more than once.

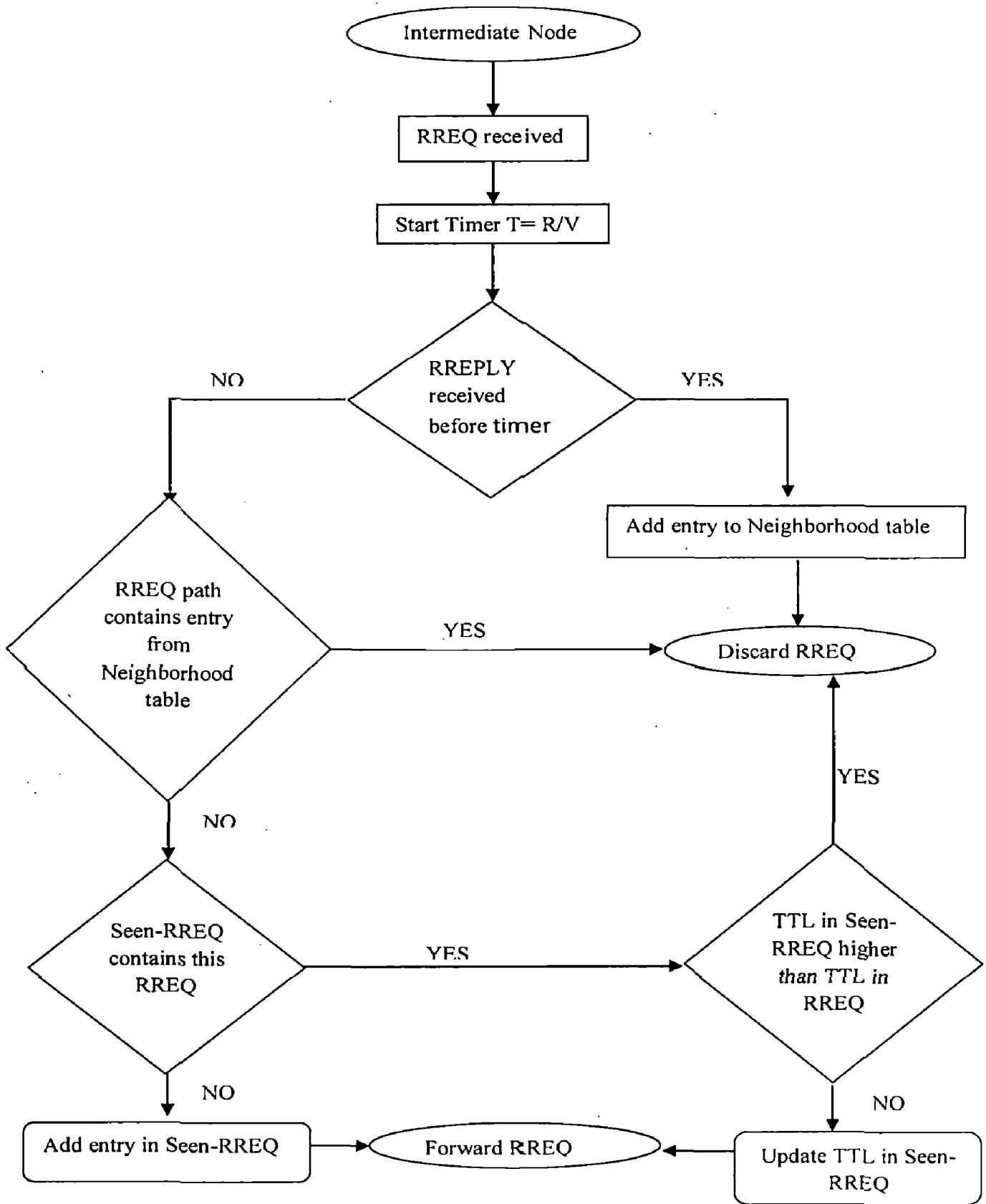


Figure 4.2 Flow chart of RREQ processing at intermediate nodes

To understand the process of updating Neighborhood table at intermediate nodes, the following section presents three scenarios with examples.

4.2.3 Example Neighborhood table updating scenarios

We have presented three different scenarios to understand the neighborhood table updating process by intermediate nodes which are 1-hop neighbors of destination.

4.2.3.1 Scenario 1

This scenario is shown in Figure 4.3. In which there are two intermediate nodes 1 and 2, both are 1-hop neighbors of each other. And only one of them node 2 is in 1-hop neighborhood of destination.

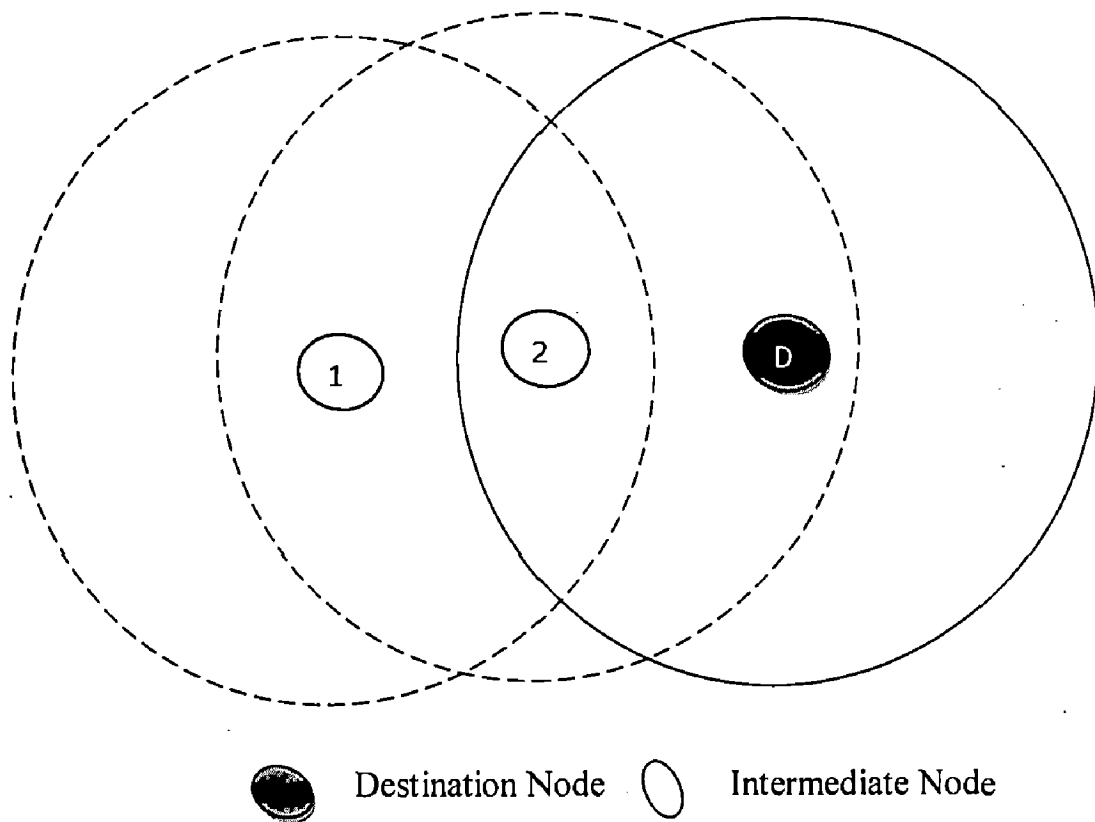


Figure 4.3 Scenario 1

Suppose node 1 forwarded the RREQ sent by some other node. Node 2 receives this RREQ and waits 1-hop propagation time. In Figure 4.3 node D is not in the radio propagation range of 1, so node 2 will not receive RREPLY. After 1-hop propagation

time expires, node 2 forwards this RREQ by adding an entry into its Seen-RREQ table. In this example scenario no entry will be added to Neighborhood table.

4.2.3.2 Scenario 2

This scenario is shown in Figure 4.4, in which there are two intermediate nodes 1 and 2. Both are 1-hop neighbors of each other. Also, both nodes are 1-hop neighbors of destination.

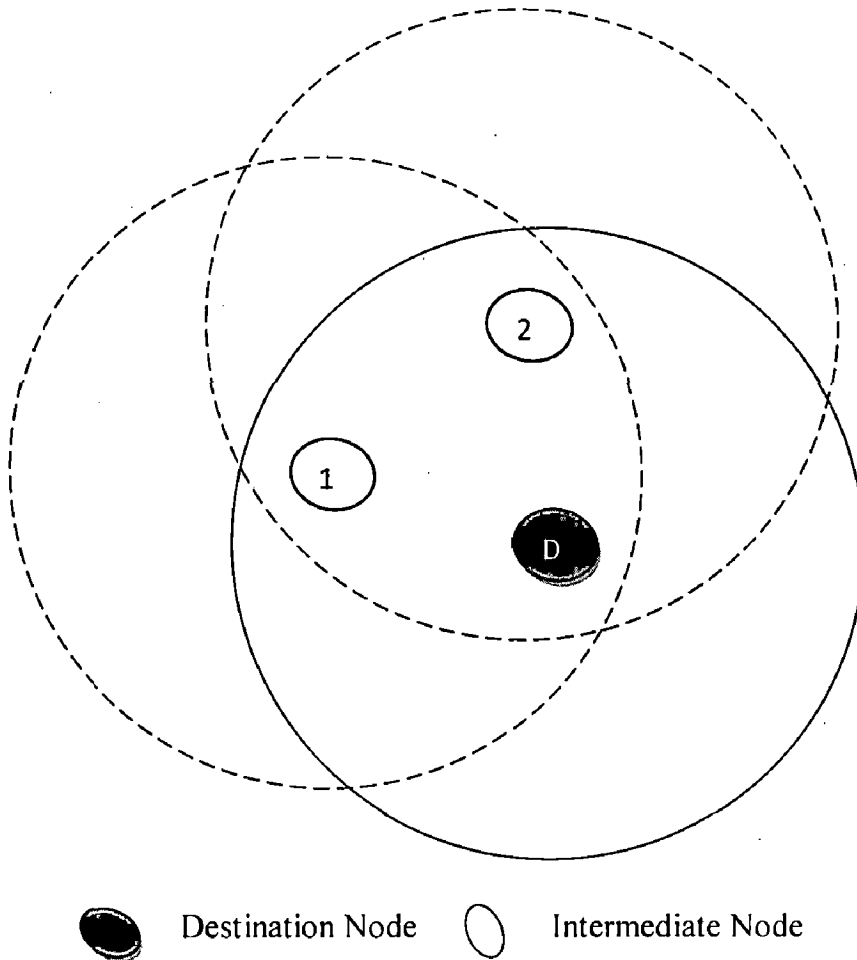


Figure 4.4: Scenario2

Suppose node 1 forwards RREQ packet sent by some other node. Both destination and node 2 receive this RREQ. Destination sends RREPLY immediately. Node 2 waits for 1-hop propagation time. Before timer expires node 2 receives the RREPLY from node D which is 1-hop distance and also originated at D. So node 2 by observing RREPLY finds

that this RREPLY is sent to the source node through node 1. It adds node 1 to its Neighborhood table and discards this RREQ, and consequent RREQ with same ID containing node 1 in its path.

4.2.3.3 Scenario 3

This scenario is shown in Figure 4.5, in which there are two intermediate nodes 1 and 2. Both are 1-hop neighbors of destination, but not each other.

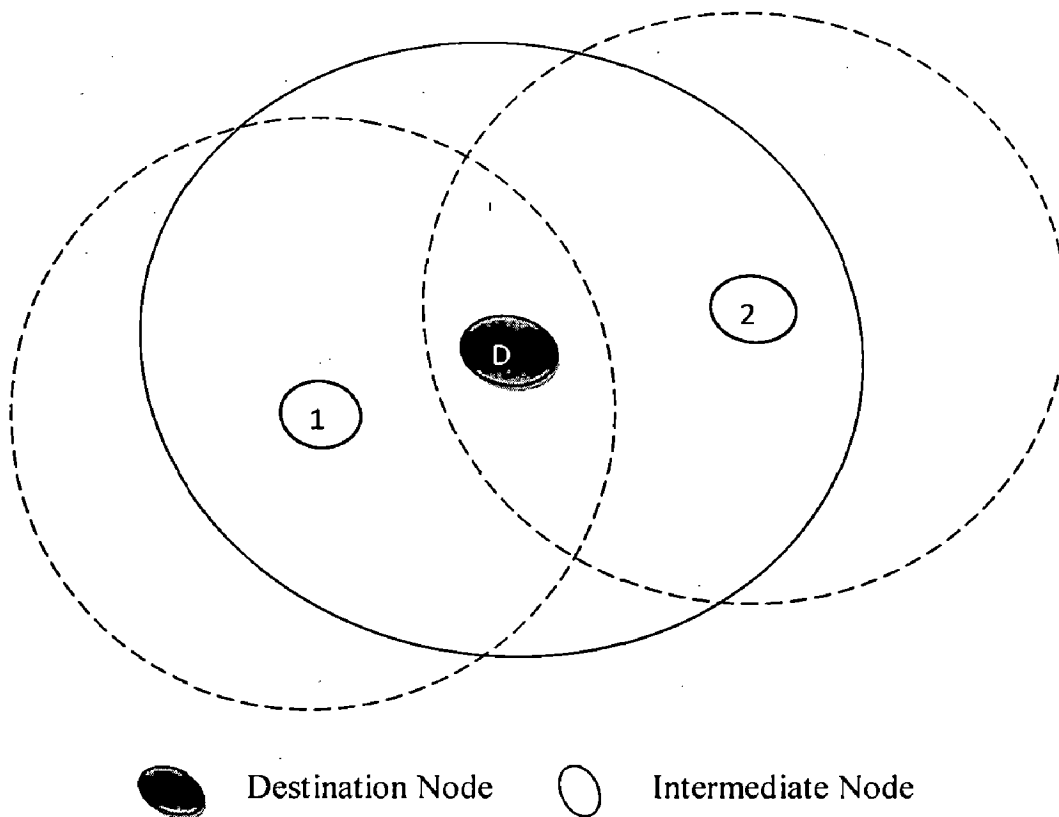


Figure 4.5: Scenario3

Suppose node 1 forwards RREQ packet sent by some other node. Destination receives this RREQ. It sends RREPLY immediately. Node 2 receives a RREPLY from D which is 1-hop distance, and also originated at D. So node 2 by observing the RREPLY finds that this RREPLY is sent to the source node through node 1. It adds node 1 to its

Neighborhood table and discards this RREQ, and consequent RREQs with same ID containing node 1 in its path.

4.2.4 RREPLY at destination and intermediate nodes

- As and when the destination node receives the RREQ, it immediately sends RREPLY. This can be achieved by giving the processing and sending of RREQ and RREPLY packets highest priority in the network.
- If an intermediate node receives a RREPLY, it checks if it is the intended next recipient. If yes, it forwards the RREPLY back to source along the route given in RREPLY.
- If an intermediate node overhears a RREPLY and if it is not the intended next recipient, then it adds the first node in the path from destination to source to Neighborhood table. The first node in the path can be found from the RREPLY message itself because RREPLY messages carry the entire path from source to destination.

Suppose in Figure 4.1, destination node 30 is sending a RREPLY with the following route 30-21-20-11-2-1-0 to source node 0. All the 1-hop neighbors of 30 overhear this RREPLY. They add node 21 to their neighbor list and discards all the RREQ with same ID, coming to destination 30 through node 21 from the source node 0.

4.3 Mitigation of wormhole attack effects

4.3.1 Dos attack

We described the formation of Dos attack in section 3.3. Our proposed modifications allow forming of the paths through legitimate nodes.

This problem solution can be visualized using Figure 4.1. Suppose among the 1-hop neighbors of node 30 (destination), node 22 receives RREQ early, and it forwards this RREQ to nodes 21, 23, 30, 29 and 31. After receiving RREQ, each node except node 30 (destination) waits for R/V time to allow overhearing of RREPLY, to know the possibility of presence of destination in their 1-hop neighborhood.

After reception of RREQ at destination, it sends RREPLY, which is overheard by nodes 21, 22, 29 and 31, and they will add node 22 to their neighborhood table, stating node 22 as 1-hop neighbor of node 30 (destination). All 1-hop neighbors of node 30 (destination) will not forward RREQ with the seen RREQ ID coming from node 22 but forward RREQs from other paths, forming a path only through legitimate nodes. Hence Dos attack can be mitigated.

4.3.2 Cache poisoning, indirect Sinkhole attack and Traffic analysis

The use of timer for accepting RREPLY at source and intermediate nodes, results in no tunneling of RREPLY through the in-band wormhole tunnel, resulting in no path through the wormhole tunnel. Hence all these attacks are mitigated.

5. SECURITY EXTENSION

In this chapter we propose a security extension to the multi-path DSR (MDSR) protocol to pinpoint and isolate the in-band wormhole attacker nodes. This security extension is applicable to all reactive and proactive routing protocols.

We considered following assumptions in the design of this security extension.

1. All the RREPLY messages are authenticated by using digital signature.
2. The attacker can only encapsulate the original packet and transfer it to other attacker.
3. The attacker cannot change the contents of the packet.
4. All the nodes know the maximum transfer unit (MTU) of the network. This information is also available at the network layer.
5. Attacker nodes cannot fragment and reassemble the packets, without losing the digital sign.
6. All links are bi-directional.

5.1 Extension to proposed MDSR

The extensions for the proposed MDSR are as follows.

- Use of fixed RREPLY message size.
- A table called suspicious node table containing entries listing malicious nodes is maintained at each node.

5.1.1 Fixed RREPLY message size

As we know the MTU of the network, we fix the size of RREPLY message to be 40 bytes smaller than MTU of the network. All other messages/ packets are of size less than RREPLY size. This can be seen in Figure 5.1:

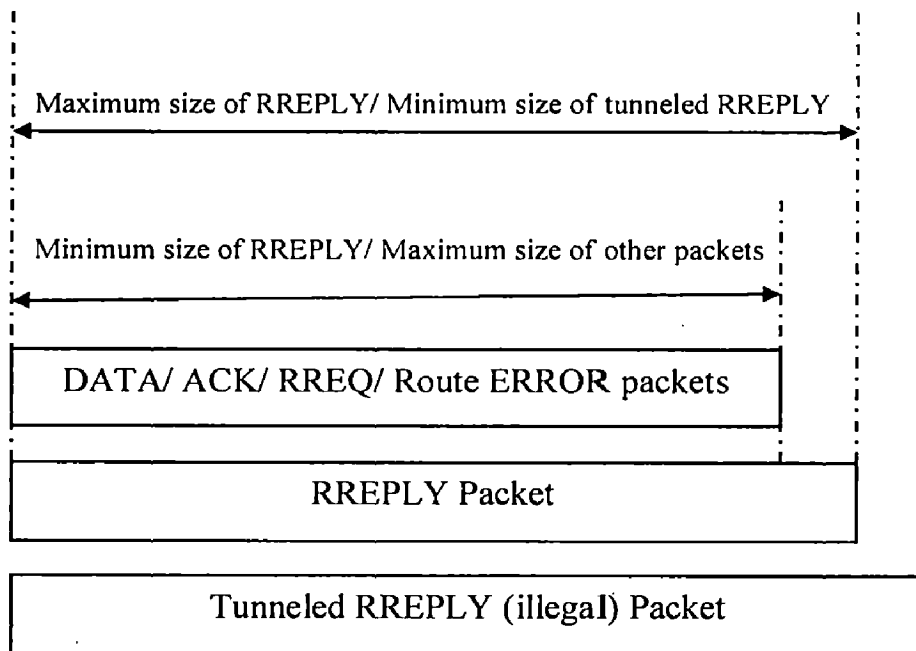


Figure 5.1 Restricted packet sizes in the network

5.1.2 Suspicious table

Each node in network maintains a table listing the malicious nodes found during operation of the network.

5.2 Mechanism

- After receiving the RREQ, the destination node forms a RREPLY message.
- Before sending the RREPLY, the destination calculates the size of the RREPLY packet. If the size is less than some predefined maximum SIZE, it pads the RREPLY. Then it digitally signs the RREPLY packet and sends it.
- Figure 5.1 shows the mechanism In the proposed extension,. Any intermediate node tunnels the RREPLY, and then the new packet will exceed allowable maximum packet size of the network.
- Any intermediate node that sees the packet with more than the allowable maximum size simply discards that packet and makes a note of the node which tunnels the packet as malicious node in Suspicious table

- The nodes discard all the packets coming from or destined to the nodes in suspicious table.
- If RREPLY messages are not tunneled, it is not possible to establish a connection through the tunnel. This will avoid the formation of wormhole tunnel.

5.3 Analysis

With this security extension we can detect the wormhole attacker nodes at the first place, so no data packets go through the wormhole tunnel. Our assumptions make sure that the sending of packets of size greater than the allowed size is impossible. Hence this mechanism does not require any threshold value as is required by some mechanisms. Once an illegal packet is received, the source of the received packet is considered as malicious and added to the Suspicious table to avoid formation of wormhole tunnel and isolate the node from the network.

Applications like Mobile IP need packets to be encapsulated for their operation. In such applications, due to the use of our security extension, nodes of the network add agent nodes which use encapsulation to their suspicious table. To avoid this, an exception can be added to allow encapsulation for agent nodes.

For proactive routing protocols, instead of RREPLY messages, periodic hello messages can be set to maximum size.

6. Simulation

6.1 JiST-swans

Java in Simulation Time (JiST): JiST is a new Java-based discrete-event simulation engine, with a number of novel and unique design features [20]. It is a prototype of a new general-purpose approach to building discrete event simulators, called *virtual machine-based simulation that* unifies the traditional systems and language-based simulator designs. The resulting simulation platform is more efficient. It out-performs existing highly optimized simulators both in time and memory consumption.

The JiST system architecture, depicted in Figure 6.1, consists of four distinct components: a compiler, a byte code rewriter, a simulation kernel and a virtual machine. JiST simulation programs are written in plain, unmodified Java and compiled to byte code using a regular Java language compiler. These compiled classes are then modified, via a byte code-level rewriter, to run over a simulation kernel and to support the simulation time semantics described shortly. The simulation program, the rewriter and the JiST kernel are all written in pure Java. Thus, this entire process occurs within a standard, unmodified Java virtual machine (JVM). The benefits of this approach to simulator construction over traditional systems and languages approaches are numerous [20].

Embedding the simulation semantics within the Java language allows reuse of a large body of work, including the Java language itself, its standard libraries and existing compilers. JiST benefits from the automatic garbage collection, type-safety, reflection and many other properties of the Java language. This approach also lowers the learning curve for users and facilitates the reuse of code for building simulations. The use of a standard virtual machine provides an efficient, highly-optimized and portable execution platform and allows for important cross-layer optimization between the simulation kernel and running simulation. Furthermore, since the kernel and the simulation are both running within the same process space it reduces serialization and context switching

overheads. In summary, a key benefit of the JiST approach is that it allows for the efficient execution of simulation programs within the context of a modern and popular language. JiST combines simulation semantics, found in custom simulation languages and simulation libraries, with modern language capabilities. This design results in a system that is convenient to use, robust and efficient.

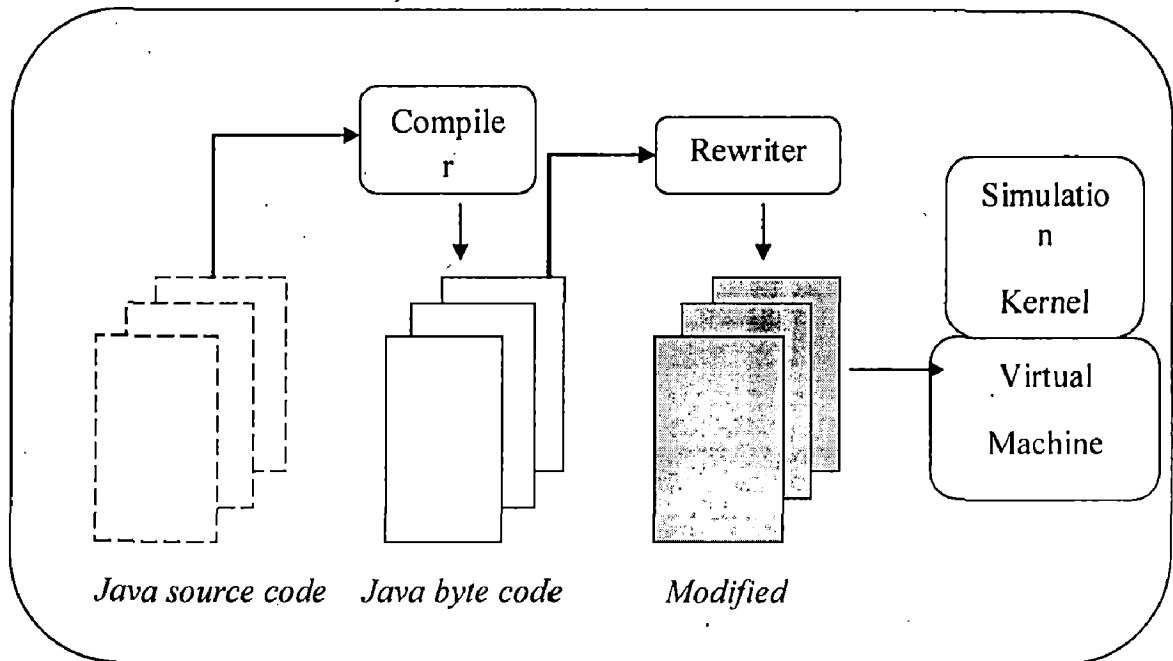


Figure 6.1: JiST system architecture

Scalable wireless network simulator (SWANS) [20] is a scalable wireless network simulator built atop the JiST platform. It was created primarily because existing network simulation tools are not sufficient for current research needs, and its performance serves as a validation of the virtual machine-based approach to simulator construction. SWANS are organized as independent software components that can be composed to form complete wireless network or sensor network configurations. Its capabilities are similar to ns2 and GloMoSim, but are able to simulate much larger networks. SWANS leverages the JiST design to achieve high simulation throughput, save memory, and run standard Java network applications over simulated networks. In addition, SWANS implements a data structure, called hierarchical binning, for efficient computation of signal propagation.

Every SWANS component is encapsulated as a JiST entity: it stores its own local state and interacts with other components via exposed event-based interfaces. A SWAN contains

components for constructing a node stack as shown in Figure 4.2, as well components for a variety of mobility models and field configurations. It allows components to be readily interchanged with suitable alternate implementations of the common interfaces and for each simulated node to be independently configured. Finally, it also confines the simulation communication pattern. For example, Application or Routing components of different nodes cannot communicate directly. They can only pass messages along their own node stacks. Consequently, the elements of the simulated node stack above the Radio layer become trivially parallelizable, and may be distributed with low synchronization cost. In contrast, different Radios do contend (in simulation time) over the shared Field entity and raise the synchronization cost of a concurrent simulation execution. To reduce this contention in a distributed simulation, the simulated field may be partitioned into non-overlapping, cooperating Field entities along a grid.

6.2 Simulation parameters

In this section we list the various simulation parameters we used in our simulation scenarios. These are given in Table 6.1, Table 6.2 and Table 6.3 for field parameters, physical layer parameters and protocols used respectively.

Field parameters

Parameter	Value
Field width	1100mt
Field height	1100mt
Signal Propagation Model	Hierarchical binning
Signal Interference Model	RadioNoiseAdditive
Path Loss	TwoRay
Fading	None
Placement	Scenario specific
Mobility	Static/ Random Walk

Table 6.1 Field Parameters

Physical layer parameters

Parameter	Value
Transmit Power	15.0 db
Band Width	2Mb/s
Transmission Range	130mt
Temperature	290 c

Table 6.2 Physical layer parameters

Protocols

Parameter	Value
Mac Protocol	IEEE 802.11
Routing Protocol	Dynamic Source Routing (DSR)
Transport protocol	User datagram protocol (UDP)
Application	Constant Bit Rate (CBR)

Table 6.3 Protocols used

6.3 Simulation network

In order to simulate the proposed protocols, we created the network shown in Figure 6.2. In this network there are two attacker nodes which are placed near the source and destination initially and moved away as the simulation progress. We have created two different types of attacker nodes.

Type1 attacker nodes create an in-band hidden wormhole attack. This can be done by tunneling all the RREQ to the partner attacking node. The other attacker node broadcast the RREQ to places of the network near the destination. After it receives a RREPLY it tunnels back the RREPLY to the first attacker, which sends this RREPLY to the source. After the successful creation of wormhole, successive data packets will be dropped causing the throughput of the network to decrease.

Type2 attacker nodes create an out-band hidden wormhole attack. This is achieved by placing one attacker near the source and other attacker near the destination. If any RREQ packet is received by the attacker near the source, it gives this packet to other attacker. In

this type of attack attackers introduce only propagation delay on the packet, no MAC layer delays.

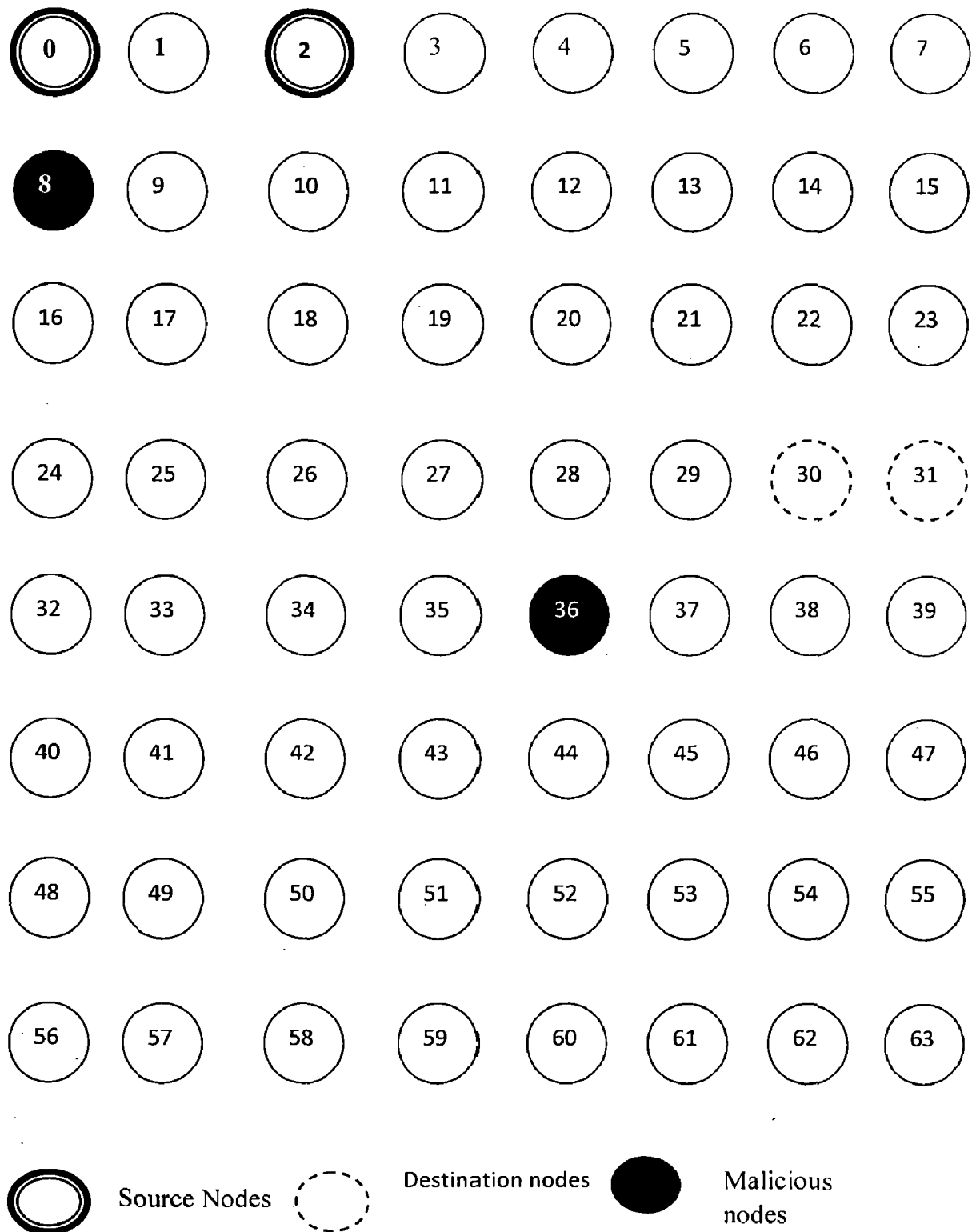


Figure 6.2 Simulation network with 64 nodes of which 2 are malicious

6.4 Metrics for Evaluation

In order to evaluate the simulation results the following parameters are considered.

1. **Overhead:** In a proactive routing protocol like DSR the main overhead comes while transmitting the RREQ packet. The number RREQ packets transferred during the simulation gives the overhead of the network.
2. **Throughput:** Throughput is calculated as the ratio to the number of data packets transmitted to the number of data packets successfully received at the destination. In order to eliminate duplicate packets, we have modified DSR to check for duplicates and remove them.
3. **Connectivity among attackers:** This is defined as the number of RREPLY packets successfully tunneled by the attackers. This is an important parameter in in-band wormhole attacks. A decrease in the number of RREPLYs tunneled shows the isolation of attackers from network.

6.5 Analysis of results

Figure 6.3 shows the number of RREQ packets forwarded by traditional DSR and proposed Multipath DSR (MDSR). From Figure 6.3 we can observe that the proposed MDSR introduces very less additional overhead on traditional DSR in the network. This is as we have discussed in section 4.2.

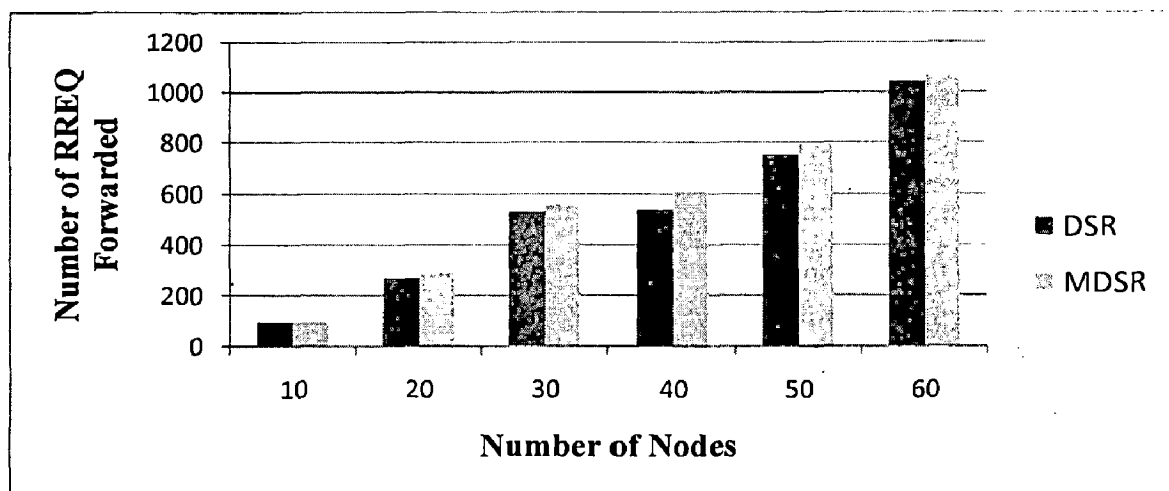


Figure 6.3 No. of RREQ forwarded by intermediate nodes.

Next simulated different static network topologies, uniformly distributed, clustered and linear by varying number of nodes and their positions in the simulation field. With both DSR and MDSR obtained the throughput Figure 6.4 shows the results. As we can see in Figure 6.4 through put of MDSR is high compared to traditional DSR. Especially when we have simulated with linear network like in case 2 the throughput increased well.

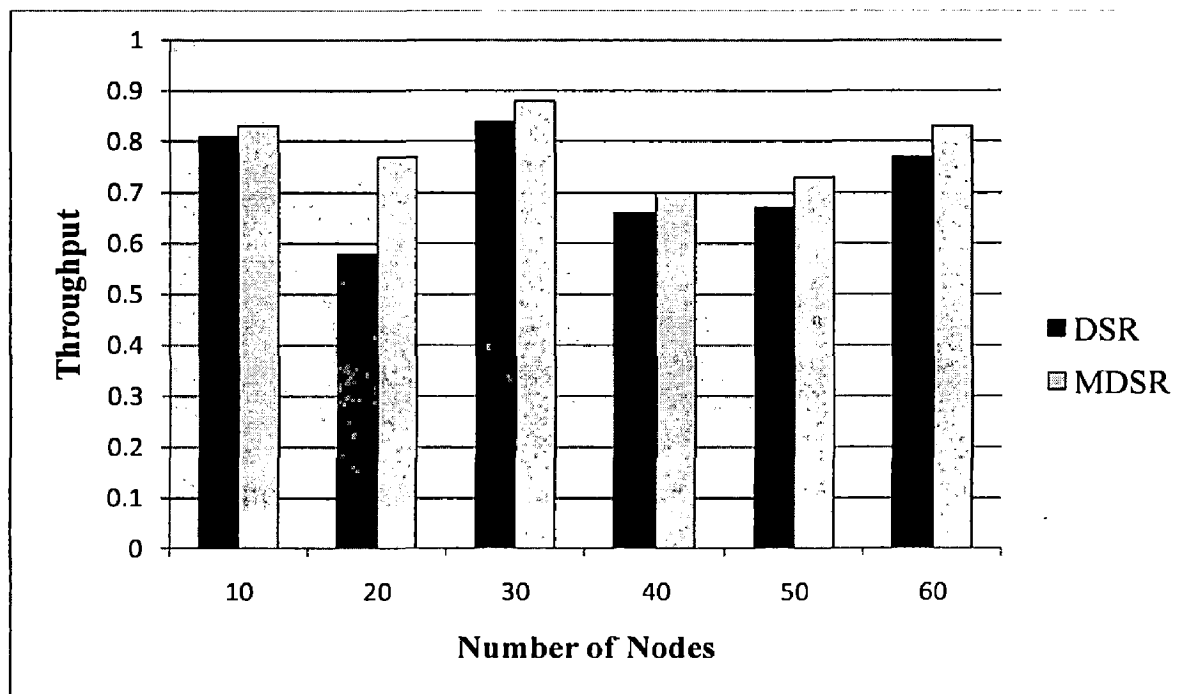


Figure 6.4 Throughput comparisons, DSR vs. MDSR under no attack

Then we have introduced two attackers of type2 in to the network as shown in the Figure 6.2. The attackers establish out-band hidden worm hole and performs DOS attack as described in section 3.3.

As we can observe from the graph shown in Figure 6.5, the throughput of the traditional DSR is very low (equal to 0) when the attackers are near the source and destination. This is because in traditional DSR once a node sees a RREQ packet with particular ID from a node it doesn't forward any subsequent packets. The attackers are exploiting this. That is why the throughput is nearly zero when the attackers are close. But with MDSR the RREPLY is coming through alternate legitimate paths so there is through put even if the attackers are at 1-hop neighbors of both source and destination.

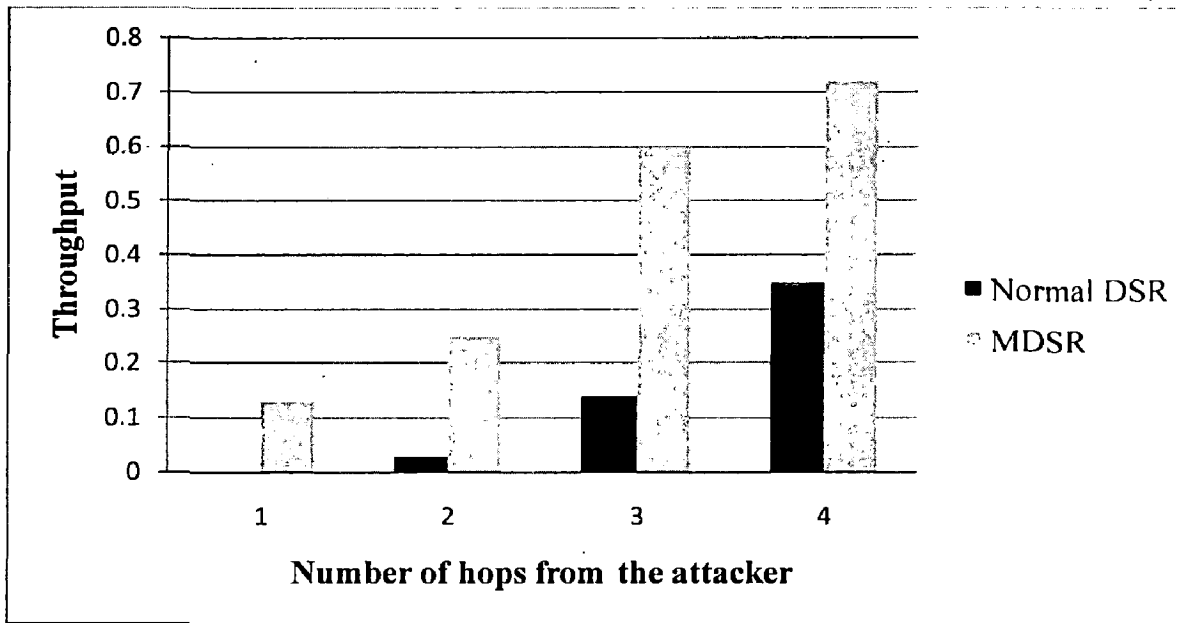


Figure 6.5 Throughput comparisons under out-band hidden wormhole attack.

Next we added the security extension to the MDSR (MDSR-Se). We removed type2 attackers from the network and added two type1 attackers. Type1 attackers perform traffic analysis and packet dropping attacks. Figure 6.6 shows the throughput of the network with DSR, MDSR and MDSR-Se. MDSR-Se has very good throughput even when the attackers are too close. This is because the security extension isolates the attackers from the network and completely eliminates the wormhole tunnel.

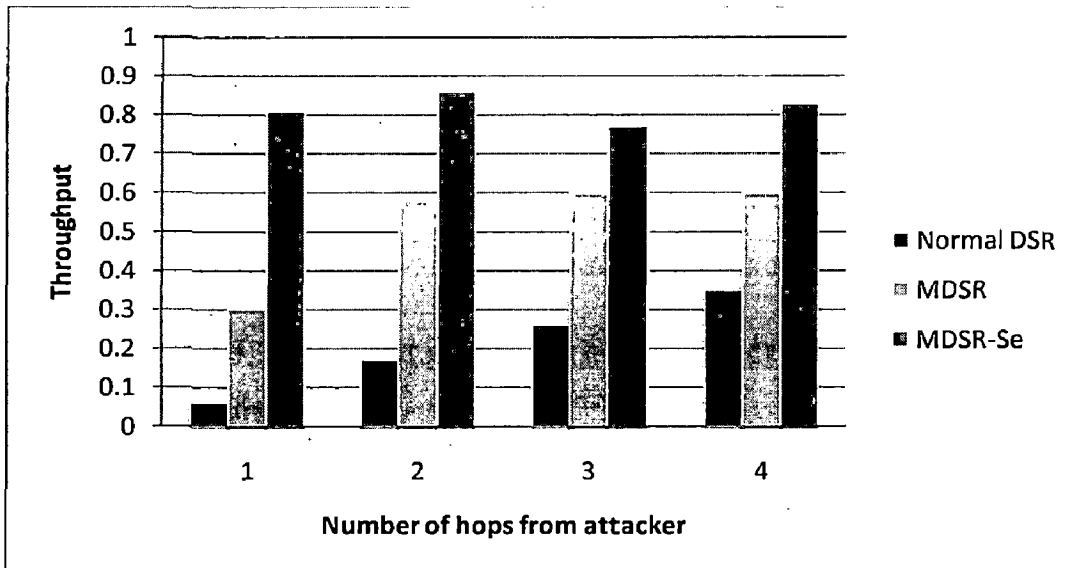


Figure 6.6 DSR vs. MDSR vs. MDSR-Se throughput comparison under in-band wormhole attack

To show the connectivity we have simulate the network with in-band wormhole. Figure 6.7 shows the percentage of RREPLYs tunneled with DSR and MDSR with security extension. The higher rate of percentage RREPLYs tunneled the higher connectivity between attackers. In Figure 6.7 with MDSR-Se the percentage of RREPLYs tunneled greatly reduced.

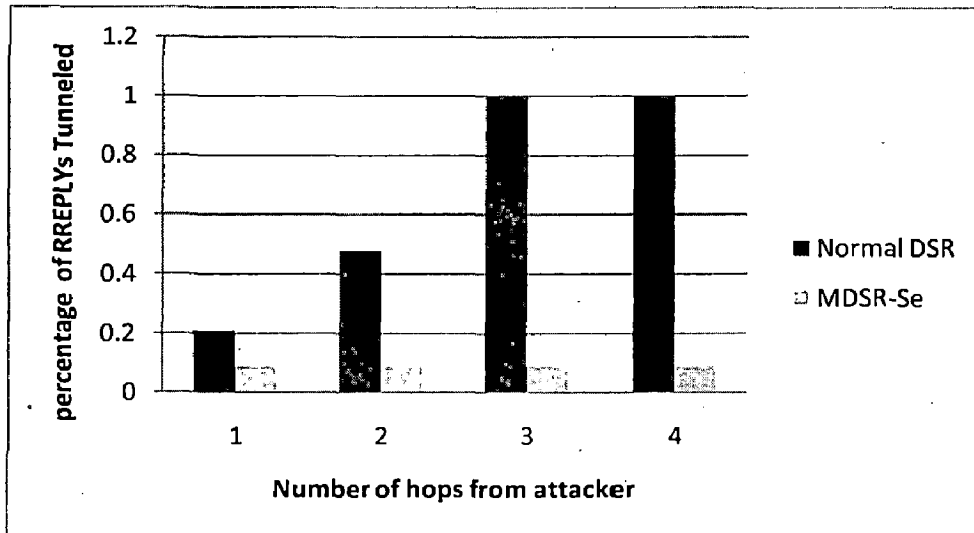


Figure 6.7 Percentage of RREPLYs tunneled with DSR and with MDSR-Se

Isolation of attackers from network not only means reducing the percentage of RREPLYs tunneled. But also the communication between the attackers must be broken. It is not possible to intercept the communication of attackers when they have out-banded communication. If the attackers are using in-band to communicate, it is possible to isolate the attackers by stopping most of the packets flowing between them. The packets include tunneled RREQ. Figure 6.8 shows the number of RREQ packets successfully tunneled with DSR and with MDSR-Se.

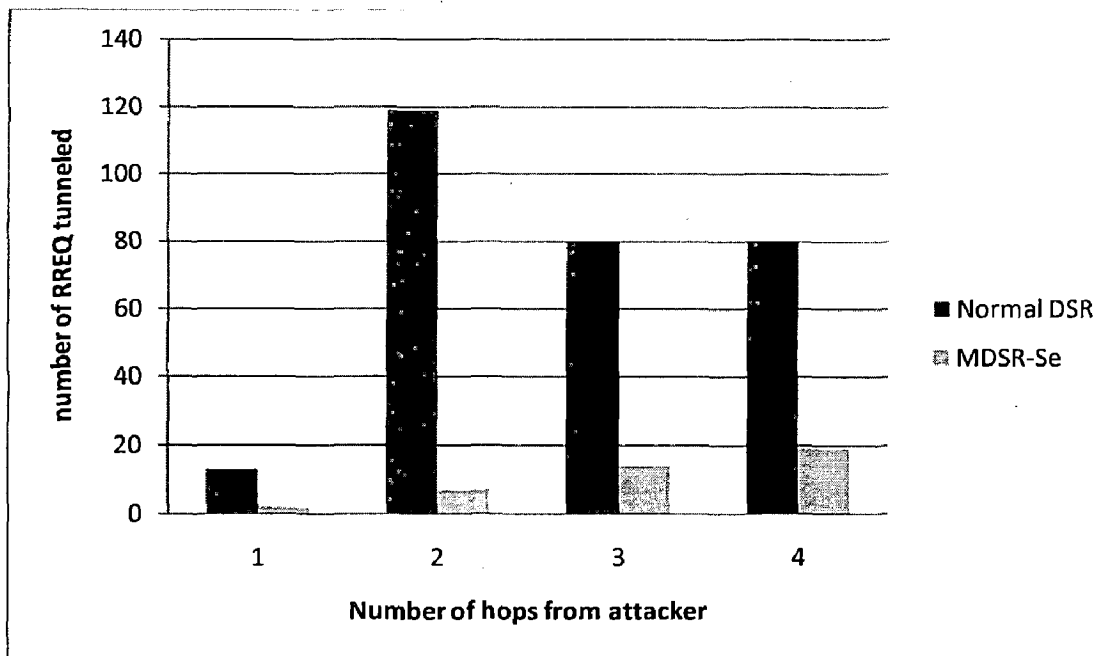


Figure 6.8 number of RREQ successfully tunneled under DSR and MDSR-Se

From Figures 6.7 and Figure 6.8 it is clear that with the MDSR-Se it is possible to isolate the in-band wormhole attackers from the network.

7. CONCLUSION

7.1 Summary of work done

Secure routing protocols for ad-hoc networks that are designed to minimize the route falsification attacks, fail against in-band worm-hole attacks created by the insider nodes without any special hardware. Even though these attacks are less powerful compared to traditional out-band worm-hole attacks, they can be launched very easily. In multipath routing protocols there is a higher chance that these attackers succeed in gaining transmission.

Our proposed mechanism, with fixed size RREPLY messages to complement the existing source routing protocols like (DSR), resist the creation of in-band worm-hole tunnels with very less additional overhead. We have investigated the effectiveness of our proposal using simulation. Simulation results confirm that in-band wormhole attacks can be detected and isolated completely from the network.

Even though it is practically not possible to pinpoint and isolate hidden out-band wormhole attacks, our proposed MDSR with Neighborhood table and timer at source and intermediate nodes, minimizes DOS attacks of hidden out-band wormhole, by allowing legitimate RREQ to reach the destination through alternate paths.

7.2 Suggestions for further Work

- The MDSR-Se can be simulated with proactive routing protocols like DSDV and OLSR. In order to simulate MDSR-Se, instead of RREPLY, periodic HELLO messages should be used.
- In order to further minimize the sinkhole attack with out-band wormholes; one can combine our proposed MDSR with path rating mechanisms. In the Path rating mechanisms, destination send periodic messages to source node about the quality of different paths. This can be used to select paths to communicate. In order to do

this, one needs to use our MDSR to find different paths between source and destination and rate the paths for communication.

- One can also place the time stamps in RREQ and RREPLY messages at intermediate nodes. This time stamps include time difference between two successive RREQ of same ID received at each intermediate node, and time gap between RREQ forwarded and RREPLY received along the path. By simulating with various wormhole lengths, one can obtain different values. After obtaining these values one can use Data mining techniques to eliminate the possible tunneled paths at first place.

7.3 Contributions

Papers selected

[1] Tirumalesh .C, kumkum Garg, "Secure Multipath Routing Protocol for detecting and avoiding worm-hole attacks", ICCNT 2009, Volume 23, May 2009. *International Conference on Computer and Network Technology, Chennai.*

[2] Tirumalesh.C , kumkum Garg. "Secure Multipath Routing Protocol for detecting and mitigating worm-hole attacks", CICSyN2009, May 2009. *First International Conference on Computational Intelligence, Communication Systems and Networking, Indore.*
Contribution to Jist-Swans

1. Integrated Inspect animation support into JiST-Swans code.
2. Added statistical collection to DSR implementation.
3. Modified DSR to eliminate duplicate messages.

References

- [1] S. Corson, J. Macker “Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations (RFC 2501)” [Online at <http://www.ietf.org/rfc/rfc2501.txt>] [last Accessed, june 2009], 1999.
- [2] D. Johnson, Y. Hu and D. Maltz, “Dynamic Source Routing Protocol (DSR)”. [Online at <http://www.ietf.org/rfc/rfc4728.txt>] [Accessed, Jun 2008], 2007.
- [3] Hu.Y.-C, Perrig. A, Johnson. D.B, “Packet leashes: a defense against wormhole attacks in wireless networks”, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, Volume: 3, pp: 1976-1986, April 2003.
- [4] Y. C. Hu, A. Perrig, and D.B. Johnson, “Wormhole Attacks in Wireless Networks”, IEEE JSAC Volume: 24, pp.370-80, Feb. 2006.
- [5] Nait-Abdesselam. F, “Detecting and avoiding wormhole attacks in wireless ad hoc networks”, Communications Magazine, IEEE, Volume: 46, pp: 127-133, April 2008.
- [6] S. Zheng, T. Jiang, J. S. Baras, “Intrusion Detection of In-Band Wormholes in MANETS Using Advanced Statistical Methods”, Milcom: 08 Assuring Mission Succes, November 2008.
- [7] L. Lazos, R. Poovendran, “SeRLoc: secure range-independent localization for wireless sensor networks”, ACM workshop on wireless security, pp: 21-30, October 2004.
- [8] Chunxiao Chigan, Bandaru. R, “Secure node misbehaviors in mobile ad hoc networks”, IEEE 60th VTC2004-Fall, Volume: 7, pp: 4730-4734, September 2004.
- [9] Hon Sun Chiu, King-Shan Lui, “DelPHI: wormhole detection mechanism for ad hoc wireless networks”, IEEE Wireless Pervasive Computing, pp: 6-12, January 2006.
- [10] Song. N, Qian, L.Li, X., “Wormhole attacks detection in wireless ad hoc networks: a statistical analysis approach”, 19th IEEE International conference on Parallel and Distributed Processing, pp: 8-16, April 2005.
- [11] Znaidi, Wassim Minier, Marine Babau, Jean-Philippe, “Detecting Wormhole Attacks in Wireless Networks Using Local Neighborhood Information”, Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International symposium, pn : 1- 5, 15-18 Sept. 2008.

- [12] Gunhee Lee Dong-kyoo Kim Jungtaek Seo, "An Approach to Mitigate Wormhole Attack in Wireless Ad Hoc Networks", Information Security and Assurance, 2008. ISA 2008. International Conference , pn: 220 - 225 , 24-26 April 2008
- [13] Baruch Awerbuch, Reza Curtmola, Herbert Rubens, David Holmer, and Cristina Nita-Rotaru, "On the Survivability of Routing Protocols in Ad Hoc Wireless Networks", IEEE SecureComm, September 2005.
- [14] Weichao Wang, Bharat Bhargava, "Visualization of wormholes in sensor networks", ACM workshop on wireless security, pp: 51-60, 2004.
- [15] R. Barr, Z.J. Haas, and R. van Renesse, "Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks", chapter 19 – Scalable Wireless Ad Hoc Network Simulation, Auerbach, 2005.
- [16] L. Hu, D. Evans , "Using Directional Antennas to Prevent Wormhole Attacks", Proceedings of the 11th Network and Distributed System Security Symposium, pp.131-141, 2003.
- [17] Gorlatova Maria A, Kelly, Marc Liscano, Ramiro Mason, Peter C. "Enhancing frequency-based wormhole attack detection with novel jitter waveforms", IEEE Securecomm 2007, pp: 304-309, September 2007.
- [18] Maheshwari. R, Jie Gao, Das. S .R, "Detecting Wormhole Attacks in Wireless Networks Using Connectivity Information", IEEE PIMRC2008, pp: 1-5, September 2008.
- [19]] Sebastien Berton, Hao Yin , Chuang Lin Geyong Min, "Secure, Disjoint, Multipath Source Routing Protocol(SDMSR) for Mobile Ad-Hoc Networks", GCC 2006, pp. 387-394, oct 2006.
- [20] R. Barr, Z.J. Haas, and R. van Renesse, "Jist: An efficient approach to simulation using virtual machines," Software Practice & Experience, vol. 35, no. 6, pp. 539–576, 2005.
- [21] S. Kurkowski, T. Camp, M. Colagrosso, "A Visualization and Analysis Tool for Wireless Simulations: iNSpect" , Technical Report MCS 06-01, Colorado School of Mines, January 2006.
- [22] S. Kurkowski, T. Camp, N. Mushell, and M. Colagrosso, "A Visualization and Analysis Tool for NS-2 Wireless Simulations: iNSpect" , Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 503-506, 2005.

```

/**
 *
 *The following are some of the main functions to create out-band hidden wormhole attacks....
 **/

public void takepack(Ip send) {
    System.out.println(localAddr+"One RREQ received from partner out-band.....");
    JstAPI.sleep((long)(Math.random() * BROADCAST_JITTER));
    netEntity.send(send, Constants.NET_INTERFACE_DEFAULT, MacAddress.ANY);
}

private void ProcessOptions(RouteDsrMsg msg, NetAddress src, NetAddress dst,
    short protocol, byte priority, byte ttl,
    short id, short fragOffset)
{
    Iterator iter = msg.getOptions().iterator();
    RouteDsrMsg.OptionAckRequest ackRequest = null;
    RouteDsrMsg.OptionSourceRoute sourceRoute = null;
    while (iter.hasNext())
    {
        byte[] optBuf = (byte[])iter.next();
        RouteDsrMsg.Option opt = RouteDsrMsg.Option.create(optBuf, 0);
        if (opt == null)
        {
            // This should never happen in the simulation
            throw new RuntimeException("Unrecognized DSR Option");
        }
        switch (opt.getType())
        {

```

```

    case RouteDsrMsg.OPT_ROUTE_REQUEST:
        HandleRequest(msg, (RouteDsrMsg.OptionRouteRequest)opt, optBuf, src,
            dst, protocol, priority, ttl, id, fragOffset);
        break;
    }
}

private void ForwardRequest(RouteDsrMsg msg, RouteDsrMsg.OptionRouteRequest opt,
    byte[] optBuf, NetAddress src, NetAddress dst,
    short protocol, byte priority, byte ttl,
    short id, short fragOffset)
{
    // If I've already forwarded this request, ignore it
    for (int i = 0; i < opt.getNumAddresses(); i++)
    {
        if (localAddr.equals(opt.getAddress(i))) return;
    }

    RouteDsrMsg newRequest = (RouteDsrMsg)msg.clone();
    List newOptions = newRequest.getOptions();
    newOptions.remove(optBuf);
    NetAddress[] newAddresses = new NetAddress[opt.getNumAddresses()];
    for (int i = 0; i < newAddresses.length; i++)
    {
        newAddresses[i] = opt.getAddress(i);
    }

    newRequest.addOption(RouteDsrMsg.OptionRouteRequest.create(opt.getId(),
        opt.getTargetAddress(), newAddresses));
    NetMessage.Ip newRequestIp = new NetMessage.Ip(newRequest, src, dst,
        protocol, priority, (byte)(ttl - 1), id, fragOffset);
}

```

```

System.out.println(localAddr+"One RREQ tunneled to partner out-band.....from "+src);

partner.takepack(newRequestIp);

}

```

.....
The following are some of main functions in in-band wormhole attack
.....

```

private void ProcessOptions(RouteMDSRMsg msg, NetAddress src,
                          NetAddress dst, short protocol, byte priority, byte ttl, short id,
                          short fragOffset) {

    Iterator iter = msg.getOptions().iterator();
    RouteMDSRMsg.OptionAckRequest ackRequest = null;
    RouteMDSRMsg.OptionSourceRoute sourceRoute = null;
    while (iter.hasNext())
    {
        byte[] optBuf = (byte[])iter.next();
        RouteMDSRMsg.Option opt = RouteMDSRMsg.Option.create(optBuf, 0);
        if (opt == null)
        {
            // This should never happen in the simulation
            throw new RuntimeException("Unrecognized MDSR Option");
        }

        switch (opt.getType())
        {
            case RouteMDSRMsg.OPT_ROUTE_REQUEST:
                HandleRequest(msg, (RouteMDSRMsg.OptionRouteRequest)opt, optBuf, src,
                              dst, protocol, priority, ttl, id, fragOffset);

                break;

            case RouteMDSRMsg.OPT_ROUTE_REPLY:
                HandleReply(msg, (RouteMDSRMsg.OptionRouteReply)opt);
                break;

            case RouteMDSRMsg.OPT_SOURCE_ROUTE:
                sourceRoute = (RouteMDSRMsg.OptionSourceRoute)opt;

                if (localAddr.equals(NextRecipient(sourceRoute, dst)))
                {
                    ForwardPacket(msg, sourceRoute, optBuf, src, dst, protocol,
                                   priority, ttl, id, fragOffset);
                }
                else
                {
                    PerformRouteShortening(sourceRoute, src, dst);
                }

                break;

            case RouteMDSRMsg.OPT_ACK_REQUEST:

```

```

ackRequest = (RouteMDSRMsg.OptionAckRequest)opt;
break;

case RouteMDSRMsg.OPT_ACK:
    HandleAck((RouteMDSRMsg.OptionAck)opt, dst);
    break;

case RouteMDSRMsg.OPT_ROUTE_ERROR:
    HandleError((RouteMDSRMsg.OptionRouteError)opt);
    break;

case RouteMDSRMsg.OPT_PAD1:
case RouteMDSRMsg.OPT_PADN:
    break;

default:
    // Possible problem: The processing of unrecognized options should
    // probably occur before the processing of any other options.
    // This will never arise in the simulation, though.
    switch ((opt.getType() & 0x60) >> 5)
    {
        case RouteMDSRMsg.UNRECOGNIZED_OPT_IGNORE:
            // Ignore this option
            break;

        case RouteMDSRMsg.UNRECOGNIZED_OPT_REMOVE:
            {
                // Remove this option from the packet
                RouteMDSRMsg newMsg = (RouteMDSRMsg)msg.clone();
                List options = newMsg.getOptions();
                options.remove(optBuf);
                msg = newMsg;
                break;
            }

        case RouteMDSRMsg.UNRECOGNIZED_OPT_MARK:
            {
                // Set a particular bit inside the option
                RouteMDSRMsg newMsg = (RouteMDSRMsg)msg.clone();
                byte[] newOptBuf = new byte[optBuf.length];
                System.arraycopy(optBuf, 0, newOptBuf, 0, optBuf.length);
                newOptBuf[2] |= 0x80;

                List options = newMsg.getOptions();
                options.remove(optBuf);
                options.add(newOptBuf);
                msg = newMsg;
                break;
            }

        case RouteMDSRMsg.UNRECOGNIZED_OPT_DROP:
            // Drop the packet
            return;

        default:
            throw new RuntimeException("Should never reach this point");
    }

```

```

        }
        break;    }
    }

    if (ackRequest != null)
    {
        HandleAckRequest(msg, ackRequest, src, dst, sourceRoute);
    }
}

NetAddress partners;
int type;
/**
 * Creates a new RouteMMDSR object.
 *
 * @param localAddr local node address
 */
public RouteMMDSR(NetAddress localAddr, NetAddress partners, int type)
{
    this.localAddr = localAddr;
    this.partners = partners;
    this.type = type;
    InitRouteCache();
    InitBuffer();
    InitRequestTable();
    InitRouteReplyTable();
    InitMaintenanceBuffer();
    initializeStatisticCollector();
    nextRequestId = 0;
    nextAckId = 0;
    activeRequests = new HashSet();
    activeAcks = new HashSet();

    self = (RouteInterface.Dsr)JstAPI.proxy(this, RouteInterface.Dsr.class);
    //*****Logger place
    stats.logger.printf("%s Node Successfully intilized..\n", localAddr);
    stats.logger.flush();
}

```

.....

.....

The following are some of the main methods in proposed MDSR

```

private void ForwardRequest(RouteDsrMsg msg, RouteDsrMsg.OptionRouteRequest opt,
    byte[] optBuf, NetAddress src, NetAddress dst,
    short protocol, byte priority, byte ttl,
    short id, short fragOffset)

```

```

{
    // If I've already forwarded this request, ignore it
    for (int i = 0; i < opt.getNumAddresses(); i++)
    {
        if (localAddr.equals(opt.getAddress(i))) return;
    }

    // To do in future: Check the Route Cache to see if we know a route to the
    // destination

    // Clone the message, add this node's address to the Source Route option,
    // and retransmit it.
    RouteDsrMsg newRequest = (RouteDsrMsg)msg.clone();
    List newOptions = newRequest.getOptions();
    newOptions.remove(optBuf);
    NetAddress[] newAddresses = new NetAddress[opt.getNumAddresses() + 1];
    for (int i = 0; i < newAddresses.length-1; i++)
    {
        newAddresses[i] = opt.getAddress(i);
    }
    newAddresses[newAddresses.length - 1] = localAddr;
    newRequest.addOption(RouteDsrMsg.OptionRouteRequest.create(opt.getId(),
        opt.getTargetAddress(), newAddresses));

    maintenancebuffer.add(newRequest);
    JistAPI.sleep((long)(Math.random() * BROADCAST_JITTER));
    self.clearMBuf(src,dest,opt);
}

```

```

Public void clearMBuf(NetAddress src,NetAddress dest, OptionRouteRequest opt)
{
    If( maintainencebuffer.exists(src,dest,opt))
    {
        RouteDsrMsg newReques= Maintainencebuffer.get(src,dest,opt);
        NetMessage.Ip newRequestIp = new NetMessage.Ip(newRequest, src, dst,
protocol, priority, (byte)(ttl - 1), id, fragOffset);
        if(neighbourhoodtable.checkthis(newReques))
            netentity.send(newRequestIp);
    }
    else
    {
        //nothing to doooooooooooooooooooooo
    }
}

```

```

private void HandleReply(RouteDsrMsg msg, RouteDsrMsg.OptionRouteReply reply,int ttl)
{
if(isforMe(msg))
{ NetAddress dest;
RouteRequestTableEntry entry;
OutbandHiddenStatistics.noofrreplyreceived++;
System.out.println(localAddr+" received one route reply.....");
// Update the Route Request Table
dest = reply.getAddress(reply.getNumAddresses() - 1);
entry = (RouteRequestTableEntry)routeRequestTable.get(dest);
if (entry != null) entry.numRequestsSinceLastReply = 0;
}
}

```



```

activeRequests.remove(dest);

// Add the route to our Route Cache

for (int i = 0; i < reply.getNumAddresses()-1; i++)
{
    if (localAddr.equals(reply.getAddress(i)))
    {
        NetAddress[] route = new NetAddress[reply.getNumAddresses() - 2 - i];

        for (int j = i; j < i + route.length; j++)
        {
            route[j - i] = reply.getAddress(j+1);

            System.out.println(route[j-i]);
        }

        InsertRouteCache(dest, route);

        break;
    }
} else{
    If(ttl==Constants.TTL_DEFAULT)
    {
        //iam at one hop neighborhood and iam not intended next recipient add entry to
neighborhood table

        ntable.addfrom(msg);
    }
} }

private void SendRouteReply(OptionRouteRequest opt, NetAddress src) {
    NetAddress[] routeToHere = new NetAddress[opt.getNumAddresses() + 2];

    routeToHere[0] = src;

    for (int i = 1; i < routeToHere.length-1; i++)
    {

```

```

    routeToHere[i] = opt.getAddress(i-1);
}

routeToHere[routeToHere.length - 1] = localAddr;

NetAddress[] routeFromHere = new NetAddress[routeToHere.length - 2];
for (int i = 0; i < routeFromHere.length; i++)
{
    routeFromHere[i] = routeToHere[routeToHere.length - i - 2];
}

// Add a Route Reply option indicating how to get here from the
// source and a Source Route option indicating how to get to the
// source from here.
RouteMDSRMsg reply = new RouteMDSRMsg(null);
reply.addOption(RouteMDSRMsg.OptionRouteReply.create(routeToHere));

if (routeFromHere.length > 0)
{
    reply.addOption(RouteMDSRMsg.OptionSourceRoute.create(0,
        routeFromHere.length, routeFromHere));
}

int nsize=reply.getSize();
byte[] data = new byte[1000-nsize];
Message payload = new MessageBytes(data);
RouteMDSRMsg replyreal=new RouteMDSRMsg(payload);
replyreal.addOption(RouteMDSRMsg.OptionRouteReply.create(routeToHere));
if (routeFromHere.length > 0)

```

```

    {
        replyreal.addOption(RouteMDSRMsg.OptionSourceRoute.create(0,
            routeFromHere.length, routeFromHere));
    }

    // RouteMDSRMsg.OptionPadN.create(bb);

System.out.println("SIZE OF ROUTE REPLY IS :::::::::::" + replyreal.getSize());

   NetMessage.Ip replyMsg = new NetMessage.Ip(replyreal, localAddr,
        src, Constants.NET_PROTOCOL_DSR, Constants.NET_PRIORITY_NORMAL,
        Constants.TTL_DEFAULT);

    JstAPI.sleep((long)(Math.random() * BROADCAST_JITTER));

    Transmit(replyMsg);

/** .....logger place..... */
    // stats.logger.printf("RREPLY : to : %s \t lasthop : %s\n",src,opt.getLastHopAddress());
    // stats.logger.flush();
/** ..... */
}

private void AddRequestId(NetAddress src, short id,int length)
{
    // Do nothing if it's already in the table
    if (SeenRequestLately(src, id,length)) return;

    // Otherwise add this id to the table
    RouteRequestTableEntry entry = (RouteRequestTableEntry)routeRequestTable.get(src);

    if (entry == null)

```

```

{
    entry = new RouteRequestTableEntry();
    routeRequestTable.put(src, entry);
}

entry.ids.addFirst(new RID(id,length));
if (entry.ids.size() > MAX_REQUEST_TABLE_IDS)
{
    // Make sure the list doesn't grow too large by removing the least
    // recently seen id number
    entry.ids.removeLast();
}

/**
 *
 * Route request ID with length
 *
 */
public class RID
{
    short id;
    int length;
    public RID(short id,int length)
    {
        this.id=id;
        this.length=length;
    }
}

```

```

public void receive(Message msg, NetAddress src, MacAddress lastHop,
                    byte macId, NetAddress dst, byte priority, byte ttl) {
    if (!(msg instanceof RouteMDSRMsg))
    {
        throw new RuntimeException("Non-DSR message received by DSR");
    } // Don't process any options here -- that's all done by peek. Just forward
        // any content on to the transport layer (or whatever).

        RouteMDSRMsg dsrMsg = (RouteMDSRMsg)msg;
RouteMDSRMsg.OptionSourceRoute sourceRoute = GetSourceRoute(dsrMsg);

        if (sourceRoute != null)
        {
            // Strange as it may seem, we will discard this packet, which is
            // in fact intended for us, if it arrives here before traversing
            // the other links in the intended route. (Route shortening should
            // prevent this from happening too often.)

            if (!localAddr.equals(NextRecipient(sourceRoute, dst))) return;
        }

        if (dsrMsg.getContent() != null )
        {
            boolean ok=false;

            Iterator iter1 = dsrMsg.getOptions().iterator();
            while(iter1.hasNext())
            {

                byte[] optBuf1 = (byte[])iter1.next();
                RouteMDSRMsg.Option opt1 = RouteMDSRMsg.Option.create(optBuf1,
0);

```

```

        if (opt1 == null)
        {
            // This should never happen in the simulation
            throw new RuntimeException("Unrecognized DSR Option");
        }

        if(opt1.getType()==RouteMDSRMsg.OPT_ROUTE_REPLY)
        {
            ok=true;
            break;
        }
    }if(!ok){
// Now go through some strange contortions to get this message received by
// the proper protocol handler
NetMessage.Ip newIp = new NetMessage.Ip(dsrMsg.getContent(), src, dst,
    dsrMsg.getNextHeaderType(), priority, ttl);

netEntity.receive(newIp, lastHop, macId, false);

/**.....logger place.....*/
stats.logger.printf("RECV : from : %s \n",src );
stats.logger.flush();
stats.packetReceivied();
/**.....*/
    }
}}

```

```

/**
 *this is driver file to create field and nodes for simulation.
 *
 **/
package driver;

public class MainDriver {
    private Location.Location2D location;;
    private Random rand;
    @SuppressWarnings("unchecked")
    private Vector sources;
    private Vector malnodes;
    private Mobility rmobility;
    private PathLoss pl;
    private Field field;
    private RadioInfo.RadioInfoShared radioInfo;
    private Placement place;
    private Mapper protMap;
    private PacketLoss outloss,inloss;
    private Vector nodes;

    public MainDriver()
    {
        sources=new Vector();
        malnodes=new Vector();
        nodes=new Vector();
    }
    private void addNode(int i, int x,int y) {
        RadioNoise radio = new RadioNoiseAdditive(i, radioInfo);
        Location location = new Location.Location2D(x,y);
        if(location==null)
        {
            System.out.println("Location is NULL.....");
            return;
        }
        MacInterface macProxy = null;
        Mac802_11 mac = new Mac802_11(new MacAddress(i), radio.getRadioInfo());
        mac.setRadioEntity(radio.getProxy());
        macProxy = mac.getProxy();
        // network
        final NetAddress address = new NetAddress(i);
        NetIp net = new NetIp(address, protMap, inloss, outloss /*, ipStats*/);

        // transport
        TransUdp udp = new TransUdp();

        System.out.println("Node "+i+" is added at locatio "+x+"-"+y);

        field.addRadio(radio.getRadioInfo(), radio.getProxy(), location);
        field.startMobility(radio.getRadioInfo().getUnique().getID());

        // node entity hookup
        radio.setFieldEntity(field.getProxy());
    }
}

```

```

radio.setMacEntity(mac.getProxy());

mac.setRadioEntity(radio.getProxy());
byte intId = net.addInterface(mac.getProxy(),new
MessageQueue.NoDropMessageQueue(Constants.NET_PRIORITY_NUM,
(byte)200));

mac.setNetEntity(net.getProxy(),
intId);
net.setProtocolHandler(Constants.NET_PROTOCOL_UDP, udp.getProxy());
udp.setNetEntity(net.getProxy());

RouteInterface route = null;

// routing
if(i!=8 && i!=37){
RouteMDSR dsr = new RouteMDSR(address);
dsr.setNetEntity(net.getProxy());
//dsr.getProxy().start();
route = dsr.getProxy();

net.setProtocolHandler(Constants.NET_PROTOCOL_DSR, route);
net.setRouting(route);
nodes.add(dsr);
}
else if (i==8)
{
RouteMMDSR dsr=new RouteMMDSR(new NetAddress(8),new NetAddress(37),1);
dsr.setNetEntity(net.getProxy());
//dsr.getProxy().start();
route = dsr.getProxy();

net.setProtocolHandler(Constants.NET_PROTOCOL_DSR, route);
net.setRouting(route);
}
else if(i==37)
{
RouteMMDSR dsr=new RouteMMDSR(new NetAddress(37),new NetAddress(8),2);
dsr.setNetEntity(net.getProxy());
//dsr.getProxy().start();
route = dsr.getProxy();

net.setProtocolHandler(Constants.NET_PROTOCOL_DSR, route);
net.setRouting(route);
}
}

private void generateCBRTraffic() {
int etime = Integer.parseInt(ss.endtime);

```



```
long delayInterval = (long) (((double) 1024 / 512) * 1 * Constants.SECOND); //delay interval between packets.
```

```
long iterations = (long) Math.ceil(((double) etime * (double) Constants.SECOND) / delayInterval);
```

```
int nocon = Integer.parseInt(ss.nooftransmissions);
```

```
System.out.println("!!!!!!!!!!!!!!!!!!!!!!3.Here~generating traffic!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
```

```
System.out.println("No. of iterations of transfer :" + iterations);
```

```
StatisticCollector.total=(int) iterations;
```

```
byte[] data = new byte[512];
```

```
Message payload = new MessageBytes(data);
```

```
long currentTime = 0;
```

```
Vector desti=new Vector ();
```

```
int kh=31;
```

```
desti.add(kh);
```

```
kh=30;
```

```
desti.add(kh);
```

```
System.out.print("Chosen " + nocon + " Sources are: ");
```

```
for (int i = 0; i < sources.size(); i++) {
```

```
    System.out.print(sources.get(i) + " ");
```

```
}
```

```
System.out.println();
```

```
System.out.print("Chosen " + nocon + " destinations are: ");
```

```
for (int i = 0; i < desti.size(); i++) {
```

```
    System.out.print(desti.get(i) + " ");
```

```
}
```

```
System.out.println();
```

```
for (int i = 0; i < iterations; i++) {
```

```
    for (int j = 0; j < 2; j++) {
```

```
       NetMessage msg;
```

```
        TransUdp.UdpMessage udpMsg = new TransUdp.UdpMessage(4010, 4010, payload);
```

```
        int src = ((Integer) sources.get(j)).intValue();
```

```
        int dest = (Integer) desti.get(j);
```

```
        RouteInterface srcRoute = (RouteInterface) nodes.elementAt(src);
```

```
//System.out.println("Sending....."+src);
```

```
        msg = new NetMessage.Ip(udpMsg,
```

```
            new NetAddress(src),
```

```
            new NetAddress(dest),
```

```
            Constants.NET_PROTOCOL_UDP,
```

```
            Constants.NET_PRIORITY_NORMAL,
```

```
            (byte) Constants.TTL_DEFAULT);
```

```
        srcRoute.send(msg);
```

```
    }
```

```
    JistAPI.sleep(delayInterval); //packet delay for each source.
```

```

        currentTime += delayInterval;
    }
}

/**
 *
 * builds the simulation field
 *
 */

public void buildField()
{
    rand=new Random(Long.parseLong(ss.seed));
    makeSources();
    makemaliciousNodes();

    rmobility=new Mobility.Static();

    int x=1100;
    int y=1100;
    Location.Location2D[] corners = new Location.Location2D[4];
    corners[0] = new Location.Location2D(0, 0);
    corners[1] = new Location.Location2D(1100, 0);
    corners[2] = new Location.Location2D(0, 1100);
    corners[3] = new Location.Location2D(1100,
        1100);
    Spatial spatial = spatial = new Spatial.LinearList(corners[0], corners[1],
        corners[2], corners[3]);

    pl=new PathLoss.TwoRay();
    /// rmobility.
    field = new Field(spatial, new Fading.None(), pl,
rmobility,Constants.PROPAGATION_LIMIT_DEFAULT);

    radioInfo = RadioInfo.createShared(Constants.FREQUENCY_DEFAULT,
Constants.BANDWIDTH_DEFAULT, Constants.TRANSMIT_DEFAULT,
    Constants.GAIN_DEFAULT, Util.fromDB(Constants.SENSITIVITY_DEFAULT),
Util.fromDB(Constants.THRESHOLD_DEFAULT), Constants.TEMPERATURE_DEFAULT,
Constants.TEMPERATURE_FACTOR_DEFAULT,
    Constants.AMBIENT_NOISE_DEFAULT);
    protMap = new Mapper(new int[]{
    Constants.NET_PROTOCOL_UDP,
    Constants.NET_PROTOCOL_DSR,

});

    outloss = new PacketLoss.Zero();
    inloss = new PacketLoss.Zero();
    //inloss=outloss;
    place = new Placement.Random(location);
}

```

```

        int tonodes=Integer.parseInt(ss.noofnodes);

int xx=25;
int m=0;
int yy=25;
    for (int i = 0; i <8 ; i++) {
        xx=25;
        for(int j=0;j<8;j++){

            addNode(m,xx,yy);
            m++;
            xx+=150;
        }
        yy+=150;
    }
    generateCBRTraffic();
}

/**
 * Creates location object
 */
private void createfield()
{
    float x=1100;
    float y=1100;
    location=new Location.Location2D(x,y);
}
/**
 *
 * get Sources for transmission
 *
 */
private void makeSources()
{
    int i=0;
    sources.add(i);
    i=2;
    sources.add(i);
}
/**
 *
 * get malicious nodes which are not original transmission sources and destinations
 *
 */
private void makemaliciousNodes()
{
    int i=9;
    malnodes.add(i);
    i=36;
    malnodes.add(i);
}

public static void main(String args[])
{

```



```

private int numberofpacketsreceived;
/**number of route requests send by this node*/
private int noofrouterequestssend;
public static int total;
public static int totalmal=0;
public StatisticCollector(NetAddress localAddr)
{
try {
File sfile=new File("logs/"+localAddr);
sfile.createNewFile();
logger=new PrintWriter("logs/"+localAddr);
numberofpacketsssend=0;
numberofpacketsreceived=0;
} catch (Exception ex) {
System.err.println("Statistic Collector :"+ex);
}
}
Vector nodes;
public StatisticCollector(Vector nodes)
{
this.nodes=nodes;
ix
}
public void packetReceved()
{
numberofpacketsreceived++;
}
public void packetSend()
{
numberofpacketssend++;
}
public PrintWriter getLogger() {
return logger;
}
public void setLogger(PrintWriter logger) {
this.logger = logger;
}
public void sendRREQ()
{
noofrouterequestssend++;
}
@Override
public void run() {
int totalRREQ=0;
int totalDataSEND=0;
int totalDATARECEIVE=0;
for(int i=0;i<nodes.size();i++)
{
if(nodes.get(i) instanceof RouteMDSR)

```

