Implementation and Performance Analysis of Black Hole Attacks on AODV in MANETs

Temitope Abiodun Ayoku

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
July 2015
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Serhan Çiftçioğlu
Acting Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asst. Prof. Dr. Gürcü Öz
Co-Supervisor

Assoc. Prof. Dr. Ali Hakan Ulusoy
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Alexander Chefranov _____

2. Assoc. Prof. Dr. Ahmet Rizaner _____

3. Assoc. Prof. Dr. Muhammed Salamah _____

4. Assoc. Prof. Dr. Ali Hakan Ulusoy _____

5. Asst. Prof. Dr. Gürcü Öz _____

ABSTRACT

A Mobile Ad-hoc Network (MANET) is an infrastructureless network which consists of different number of mobile nodes that are active to establish a temporary network for the transmission of data from source to destination. MANETs are extensively used, but security issues have become one of the main concerns in networking environment. Black hole attack is one of the majority threats in network security. Black hole utilizes the routing protocol to declare itself of having the shortest path to the destination node, and drops the routing packets meant for the destination node, instead of forwarding this packet to its appropriate neighbors.

In this thesis, we attempt to investigate the existing work about the black hole attacks in MANETs with Ad-hoc On-demand Distance Vector (AODV) routing protocol. For the purpose of study, firstly original AODV protocol with one and two black holes is implemented in Network Simulator version 2 (NS-2). Then a modified AODV protocol as a simple solution to the black hole attacks is implemented with one back hole. Simulations are performed on the basis of Packet Delivery Ratio (PDR) and End-to-End Delay (EED) and the effect is analyzed for the original and modified AODV after adding the black hole nodes in the network. The simulation results show that PDR drops from 98.31% to 19.30% and 8.77% with one and two black holes respectively. However, by the use of modified AODV protocol PDR is achieved as 39.19%. The results also show that there is a marginal increase in the EED when black holes are included in the original AODV. By the use of modified AODV, EED is dropped from 0.28 ms to 0.12 ms.

Keywords: MANET, AODV, Black Hole, Packet Delivery Ratio, End-to-End Delay

ÖZ

Gezgin alt yapısız ağlar (MANETs) çok sayıda gezgin cihazdan oluşan ve bir kaynaktan bir hedefe veri iletimi için alt yapıya ihtiyaç olmadan oluşturulan geçici ağlardır. Günümüzde MANET yaygın olarak kullanılmakta olup güvenlik sorunları başlıca problemlerden biri haline gelmiştir. Kara delik saldırıları, ağ güvenliğini tehdit eden en önemli tehlikelerden bir tanesidir. Kara delik düğümleri yönlendirme protokolünü kullanarak hedef düğüme en kısa yolun kendisi üzerinden geçtiğini ilan eder ve hedefe gönderilmiş olan paketlerin kendisi üzerinden geçmesini sağlayarak gelen paketleri hedefe yönlendirmek yerine imha eder.

Bu tez çalışmasında, kara delik saldırılarının AODV yönlendirme protokolü kullanan MANET üzerindeki etkisi araştırılmış olup var olan çalışmalar incelemiştir. Tez çalışmasının amacı doğrultusunda ilk önce orijinal AODV yönlendirme protokolü bir ve iki kara delik içerecek şekilde Ağ Simülatörü NS-2 de uygulanmıştır. Daha sonra kara delik saldırılarına çözüm olarak önerilmiş olan geliştirilmiş AODV yönlendirme protokolü bir kara delik olduğu durumda uygulanmıştır. Orijinal ve geliştirilmiş AODV yönlendirme protokolünü içeren benzetim çalışmaları, ağa kara delik düğümleri eklenerek Paket Teslim Oranları (PDR) ve Noktalar Arası Gecikme (EED) süreleri ile incelenmiştir. Benzetim çalışmaları kara delik bulunmadığı ortamda %98.31 olan PDR'nin bir kara delik olduğu durumda %19.30, iki kara delik olduğu durumda %8.77'ye düştüğünü göstermiştir. Geliştirilmiş AODV yönlendirme protokolünün kullanılması ile PDR %39.19 olarak elde edilmiştir. Sonuçlar ayrıca kara deliklerin mevcut olduğu durumda EED de çok az bir artış olduğunu göstermiştir. Geliştirilmiş AODV yönlendirme protokolünün kullanılması ile EED 0.28 ms'den 0.12 ms'ye düşmüştür.

Anahtar kelimeler: MANET, AODV, Kara Delik, Paket Teslim Oranı, Noktalar Arası Gecikme

DEDICATION

We strive hard to earn a degree, struggle to pay for knowledge, but what happens, if we have acquire the best knowledge, then we realize university has no value. People say education is the best legacy, but people turn it to a competition between one another. They forgot that, this world was discovered through a text book, the knowledge we acquire has made us cynical. I hope could help everyone at once, but greed has contaminate men's consciousness and has barricaded the world with hatred. "We want to live by each other's happiness not by each other's misery". Education becomes unaffordable.

I dedicated this research work to Almighty GOD who has given me the knowledge and power to do everything within me, to my lovely mother Mrs**.** C. B. Ayoku, the less privileged students' who dropped out of school or have no money to register to a standard university, to my brother and sisters (Abidemi Ayoku, Oyebanji Ayoku & Temitayo Ayoku), and in memory of my late father, Muritala Adeleke Ayoku.

ACKNOWLEDGEMENT

Praise be to Almighty GOD, earning a M.S. degree would be an impossible journey without the support of my course professors and my loved ones. Special thanks to two hard working professors, Assoc. Prof. Dr. Ali Hakan Ulusoy and Asst. Prof. Dr. Gürcü Öz; for their collaborative effort, for believing in me throughout the research work and their support for me academically. They are such an inspiration all rounds of life, and I learn so much from them. They never yell or get angry at me whenever I commit an error or never understood some topic, instead they bring light into the topic. I would also like to extend my gratitude to teaching and non-teaching staff of Department of Computer Engineering, Eastern Mediterranean University.

Secondly, my deepest gratitude goes to my lovely mother. For her humbleness, patience, dedication and financial support. Love you mama bidemi. To my sisters and brother (Abidemi Ayoku, Oyebanji Ayoku and Temitayo Ayoku). They are real definition of “true blood” family, thanks for their tremendous love and support over the years. To Odunukan families (Mr. Odunlami, Mr. Oniyide, Mr Babatunde Odunukan, Iya Muyiwa, Aunty Dayo, Mummy Tunde, Aunty Ronke, Mr. Babatunde Hassan Ambali etc.). Thanks for their prayers. Without forgetting one person, Samira Rahmati. One advice for her, “life is a journey we need to fulfil; I understand your pains and troubles, but never give up in what you do, because worst thing comes free to us in life. You have been always supportive during this thesis work”.

Thirdly, to my longest serving friends in Cyprus, from Turkey (Selman and Özge Özcan). I owe you guy from the bottom of my heart. You took me like your family, thanks for your support. I extend my thanks to all friends and colleagues.

Lastly, I know he is no more with us today, but we feel his presence. It has been nine years I started this journey. He has been most supportive in all way before I left Nigeria for study, but it is sad that he never witnesses today. I know he will be in heaven smiling down. He has been my mentor from day one, and he wanted me to achieve and go farther than him. May Almighty GOD, forgive his sins on earth and grant him a place in his kingdom. Continue to rest in peace (Late Muritala Adeleke Ayoku).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABR	Associativity-Based Routing
AGT	Agent
AODV	Ad-hoc On-demand Distance Vector
AWK	Alfred Weinberger Kernighan
CAODV	Credit based Ad-hoc On-demand Distance Vector
CBR	Constant Bit Rate
CBRP	Cluster Based Routing Protocol
CDMA	Code Division Multiple Access
CGSR	Clusterhead Gateway Switch Routing
CRRT	Collect Route Reply Table
DiffServ	Differentiated Services
DRT	Data Routing Table
DSDV	Destination-Sequenced Distance-Vector
DSR	Dynamic Source Routing
DoS	Denial of Service
EED	End-to-End Delay
FREP	Further Reply
FREQ	Further Request
FSR	Fisheye State Routing
GSR	Global State Routing
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IfQ	Interface Queue

IntServ	Integrated Services
LL	Link Layer
MANET	Mobile Ad-hoc Network
MAC	Media Access Control
ms	Milliseconds
NAM	Network Animator
NS	Network Simulator
OTCL	Object Oriented Tool Command Language
PDR	Packet Delivery Ratio
PHY	Physical
PR	Packets Received
PS	Packets Sent
QoS	Quality of Service
RERR	Route Error
RREP	Route Replay
RREQ	Route Request
RSVP	Resource Reservation Protocol
RPNS	Reply Neighbor Set
RQNS	Request Neighbor Set
s	Seconds
SSR	Scalable Source Routing
SAODV	Secure Ad-hoc On-demand Distance Vector
TAT	Total Arrival Time
TC	Total Connection
TCL	Tool Command Language

TCP	Transmission Control Protocol
TTL	Time To Live
TORA	Temporally Ordered Routing Algorithm
TST	Total Sent Time
UDP	User Datagram Protocol
WRP	Wireless Routing Protocol
ZRP	Zone Routing Protocol

Chapter 1

INTRODUCTION

1.1 Problem Definition and Motivation

A group of devices or stations that are connected without a wire are referred as a wireless network. The problem facing such connections are limited to open medium, speed and bandwidth. There are two types of this system model which are wireless Mobile Ad-hoc Networks (MANETs) and Fixed Backbone Wireless system [1]. In MANETs which are also refer as infrastructureless or self-configuring networks, nodes or mobile devices can establish a dynamic connection among themselves without an access point. Each node can act as a router. Routing protocols transmit packets from the source to its destination by using most efficient path and many routing protocols have been standardized by Internet Engineering Task Force (IETF). The security issues become one of the main concerns in MANETs since dynamic connections make it vulnerable to many attacks such as spoofing, eavesdropping, Denial of Service (DoS), black hole, etc. [2].

1.2 Thesis Objectives

In this thesis we will focus on Ad-hoc On-demand Distance Vector (AODV) routing protocol with black hole attacks in MANETs. The objective of the thesis work is to study the effect of black hole attacks on AODV protocol. The thesis also presents a modified AODV protocol as a solution to the black hole problem to improve the performance. The thesis also aims to perform accurate simulations with different

positioning and movement of nodes, to design and develop a suitable simulation standard, analysis the result and evaluate the performance metric of the network.

1.3 Thesis Contributions

The research in this thesis depends mainly on the study in [3] and it presents:

- Implementation of AODV routing protocol in Network Simulator version 2 (NS-2),
- Implementation of the black hole attack in AODV routing protocol in NS-2,
- Implementation of the modified AODV routing protocol proposed in [3] in NS-2 in the presence of black hole attack,
- Analyzing the performance metrics presented in [3] such as the Packet Delivery Ratio (PDR) with and without the black hole attack, and loss percentage at the black hole,
- Analyzing the End-to-End Delay (EED) performance with and without the black hole attack.

1.4 Thesis Organization

The thesis is divided into six chapters. Chapter 1 focuses on the problem definition, thesis objectives and thesis contributions. In Chapter 2, the overview of related work done is explained with different methodology and performance metrics. We furthermore discuss categories of the routing protocols, and explain in details, the AODV and black hole attack in Chapter 2. Chapter 3 presents the details about NS-2, the TCL file for designing the network topology and the AWK script file for collecting results from the data generated from the trace file. Chapter 4 discusses the implementation of a black hole for AODV routing protocol in NS-2. In Chapter 5, the simulation results are presented. Finally, a conclusion is presented along with the feature work in Chapter 6.

Chapter 2

BACKGROUND AND RELATED WORK

2.1 Literature Review about Black Hole Attacks in MANETs

In black hole attacks, the malicious node utilizes the routing protocol to declare itself as having the shortest path to the destination node, and drops the routing packets meant for the destination node instead of forwarding the packets to its appropriate neighbors.

In MANETs, different machanisms are proposed to tackle black hole attacks in current years. Sanjay Ramaswamy et al. in [4] invented a technique to identify multiple black hole nodes. These nodes work collaboratively as a unit in order to perform the black hole attack. They propose a modified AODV routing protocol by setting up a Data Routing Table (DRT), whereby every node entry is cross checked. It is shown that the obtained results are better than earlier solutions presented.

Nital Mistry et al. in [2] modify the AODV routing protocol by adding a new table that include MOS_WAIT_TIME, Cmg_RREP_Tab, and Mail_node entries. All new Route Replies (RREPs) are stored in the new table. The MOS_WAIT_TIME is adjusted to half of RREP_WAIT_TIME value – this identifies source node waiting time value for RREP control message, before regenerating Route Request (RREQ) control message. Stored RREPs in the Cmg_RREP_Tab table go through a checking procedure by the source node. It removes any RREP with high destination sequence

number immediately. The Mail_node is used to observe and record malicious node, then delete any of such record once identified. Any RREQ message generated from this black hole is not forwarded. The performance metric, in terms of PDR, in a normal AODV with black hole drop by 81.81% and the same result is presented for their solution. However there is an increment of 13.28% in EED performance.

In [5], the authors explain an approach on how the source node waits for a request from the neighboring nodes. This includes the next hop information from other intermediate nodes for a fixed time. When this time expires, the source node checks the Collect Route Reply Table (CRRT) for redundancies in the next hop node or not. If the redundancies exist in next hop, it is assumed that these routes are ok and free from any malicious attacks. The performance metrics such as PDR, EED and overhead are calculated and it is shown that PDR is achieved as 90 – 100 % with a black hole.

In [6], the authors explain protocol that allows all intermediate nodes to attach the next hop information when sending the RREP message. While the source node receives the RREP message, it investigates the reliabilty of the next hop if such path exists between the intermdeiary node and destionation node by sending an RREQ message. When such path is authenticated, the intermediary node receives the Further Request (FREQ) and sends back a Further Reply (FREP) with the information of source node that contains the check result. The source node receives this information (i.e. FREP) and verifies the originality of the route. In this work, the RREP control packet is modified to contain details of the next hop and the intermediate node which have to send RREP message, two times with one route request. The proposed solution increases delay and overhead.

Pooja Jaiswal et al. in [7] propose a solution to tackle black hole attack. They created a mechanism to receive and record destination sequence number from the neighboring node. Their idea is to compare the huge difference in the sequence numbers between source node and neighboring node. The neighboring node is declared malicious if there is a large difference, and its entry is discarded. In fact, their results show a better performance metric in terms of PDR and EED.

Hesiri Weerasinghe et al. in [8] also proposes a solution to black hole attack. They discover that the source node can establish a secure path to the destination node by discerning and separating one or more malicious node, so packets are delivered securely. They modify some methods suggested by Sanjay Ramaswamy et al. in [4]. A new DRT is implemented using the FREQ and FREP to examine the incoming packets.

Bo Sun et al. in [9] propose a neighborhood-based method. This method is used to detect whether black hole attacks exist or not. They present a routing recovery protocol to create a reliable path that is connected to the destination node, to collect the neighboring nodes information. They suggest two control packets as Reply Neighbor Set (RPNS) and Request Neighbor Set (RQNS). The RQNS consists of {Src_Addr, Des_Addr, Request_Neighbor_Seq#, Next_Hop} where Src_Addr is the delivery address for the source node, Dest_Addr is the delivery address for destination node, and Request_Neighbor_Seq# is the sequence number of RQNS. They explain further that an increment takes place in the Request_Neighbor_Seq# if a node forwards an RQNS. The RPNS consists of {Src_Addr, Dest_Add, Request_Neighbor_Seq#, Neighbor_Set} where the first three object corresponds to the ones explained for RQNS, while the last object contains the current neighbor set

of the malicious node. This is used when a malicious node recieves a RQNS message, and replies a RPNS message. The study shows that black hole attacks are lessened by this method and a higher throughput, a lower detection time and a detection probability can be obtained.

The comparisons of different researches about black hole attacks are presented in Table 2.1.

Table 2.1: Modified routing protocols with black hole

Routing Protocol	**Simulator**	**Detection Type**	**Single Path / Multi Path**	**Summary of Results**
Neighborhood based and Routing Recovery [9]				
AODV	NS-2	Single detection	Single Path	The chance of one attacker identification is 93%.
Redundant Route and Unique Sequence Number Scheme [10]				
AODV	NS-2	Single detection	Single Path	Verify 75% to 98% of the routes.
DPRAODV [11]				
AODV	NS-2	Single detection	Single Path	PDR is improved to 80 - 85% for AODV with black hole attack.
Next Hop Information Scheme [12]				
AODV	NS-2	Single detection	Single Path	An improvement of 40-50% in PDR and packet dropped ratio is reduced to 75-80%.
Nital Mistry et al.'s Method [2]				
AODV	NS-2	Single detection	Single Path	PDR is improved to 81.811% when network size varies.
Deng's solution [6]				
AODV	---------	Single detection	Single Path	Cooperative black hole attack cannot be detected.
Distributed Cooperative Mechanism (DCM) [13]				
AODV	NS-2	Cooperative detection	Single Path	PDR is improved from 64.14% to 92.93%.
DRT and Cross Checking Scheme [4,9]				

AODV	Qualnet	Cooperative detection	Single Path	Increase in throughput compare to original AODV.
Flow Conservation based approach [14]				
AODV	NS-2	Cooperative detection	Single Path	This method does not need different nodes to overhead on each other's packet.
SAODV Protocol [15]				
Secure AODV (SAODV)	NS-2	Single detection	Single Path	Improvement in PDR compared to original AODV.
Mechanism Based on Judgment Process [16]				
AODV	NS-2	Cooperative detection	Single Path	Time delay is not experienced and very easy.
Algorithm based on Preprocessor [17]				
AODV	----------	Single detection	Single Path	It fails when two malicious nodes collaborate.
Mechanism using recvReply() function[18]				
AODV	NS-2	Single detection	Single Path	This improvement only involves a minimum modification.
Credit based on AODV (CAODV) [19]				
Credit based AODV (CAODV)	NS-2	Single detection	Single Path	Improvement of 40% is observed in throughput compared to original AODV.
Neha Kaushik et al.'s Method [26]				
AODV	NS-2	Single detection	Single Path	Improvement in PDR, throughput and EED
IDSAODV [3]				
AODV	NS-2	Single and Cooperative detection	Single Path	Propose a solution to detect malicious node. Increase in PDR of 20% with two black holes compare to normal AODV with black hole.

Based on the survey, it is observed that black hole attacks are still a problem in MANETs and they should be further investigated. After analyzing the results

presented in the above research studies, [3] is selected to be implemented in this thesis since parameters used in the simulations are clearly presented, the obtained results are reliable and it includes the detection of cooperative black hole attacks. Some of the above closely related research papers such as [26] do not include the detection of cooperative black hole attacks and the presented results are not found reliable. That is why those studies are not implemented in this thesis.

2.2 Routing Protocols in MANETs

Routing can be defined as an exchange of data between two hosts in a network. The method of routing is to forward the packet towards its destination node using the optimal path. This route is measured in different metrics like traffic, security and so on. Most protocols in MANET function efficiently over a broad range of networking context from a medium of wireless ad-hoc group to a large mobile networks session.

2.2.1 Categories of Routing Protocol in MANET

As shown in Figure 2.1, routing protocols are categorized into three as proactive, reactive, and hybrid.

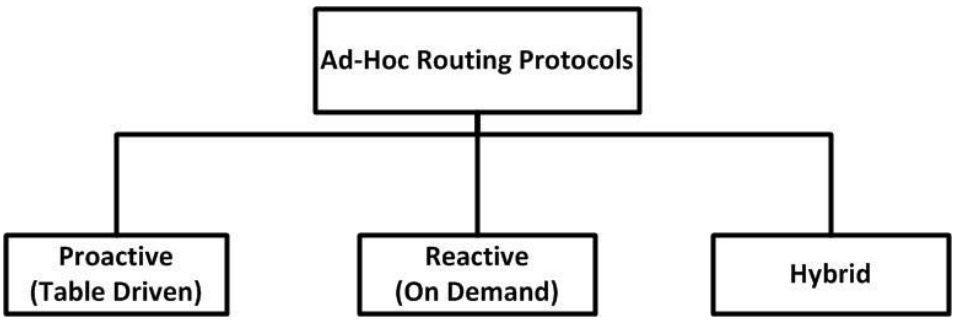

Figure 2.1: Routing protocols levels.

Proactive protocols are also referred as table driven protocols [20]. Each node broadcasts their routing data to the neighboring node to update its routing table periodically. Every node has a routing table that consists of next hop node,

destination node and number of hops information about the network topology. Table driven or proactive protocols have several complications which are repetition of path entries to a particular destination node is unnecessarily taking place in the routing tables, and when the routing tables are periodically updated, they keep the nodes active and this exhausts each nodes batteries. Destination-Sequenced Distance-Vector (DSDV), Global State Routing (GSR), Wireless Routing Protocol (WRP), Fisheye State Routing (FSR) and Clusterhead Gateway Switch Routing (CGSR) protocols are all examples of table driven routing protocols.

A reactive protocol is also referred as on demand protocol. This protocol is classified as a lazy type of routing. The source node only sets up a path if there is a need to forward the packet to the destination node [21]. It broadcasts the RREQ packet towards the intermediate node. When the intermediate node receives this RREQ it keeps forwarding this packet until it reaches to the destination node. Afterward, a reply packet is sent to the target node in the shortest path. Each node's routing tables are not updated periodically. Scalable Source Routing (SSR), Cluster Based Routing Protocol (CBRP), Temporally Ordered Routing Algorithm (TORA), AODV, Dynamic Source Routing (DSR), Associativity-Based Routing (ABR) are the examples of on demand routing protocols.

Hybrid protocols are the combination of both reactive and proactive approaches. The main idea behind these protocols is to reduce the control overhead of proactive routing protocol and decrease the latency caused by route discovery in reactive routing protocols. Zone Routing Protocol (ZRP) is an example of hybrid routing protocols [22, 23].

2.3 Ad-hoc On-demand Distance Vector

In AODV, routes are initiated by node for route discovery to the destination node only during the data transmission session and disconnect when is not active [27, 28]. For a reactive routing protocol, there are two modules as route discovery and route maintenance. AODV routing includes the following steps 1-6 for route discovery module and step 7 for route maintenance module:

Step 1: The source node floods the network with the RREQ control messages which contain the source address, request ID, source sequence number, destination address, destination sequence no, and hop count, in order to seek a route to its target node for the transmission of packets. The initialization of the variables and parameters is given in the “aodv.h” script provided in Appendix A.1.

Step 2: The neighboring nodes receive the RREQ control messages, check from the routing tables, and if no such route is found in the routing table, forward the RREQ control to appropriate path to reach the destination.

Step 3: The destination node receives the RREQ control message, and generates the RREP control message which consists of the source address, destination address, destination sequence number, hop count, and life-time.

Step 4: The destination node then unicasts the RREP control back to the same route used for the RREQ message. Source node receives the RREP control from all intermediary nodes.

Step 5: Source node then compares the sequence numbers in the RREP control messages to decide the fresh route which has the highest sequence number.

Step 6: A secure route is determined and packets are transmitted through this route.

Step 7: Both source node and destination node sends each other HELLO messages to know the activeness of this link. If the link is broken, an error message is sent.

The route discovery and route maintenance modules are explined below with an example.

Route Discovery Module: As shown in Figure 2.2, when node S wants to transmit a packet to node D, it checks in its routing table if there is a valid entry available for this path. But in case any path is not found in the routing table, node S broadcasts a route discovery to the neighboring nodes. This broadcast message consists of the address of node S and RREQ message. Any neighboring node which receives this message checks its routing table for a fresh path to node D. The fresh route is determined by the sequence number with the highest number. Thus when this route is selected, node S receives an RREP control message which includes the route. But if no such route is found in the routing table, the intermediate node updates its routing table and then transmit a RREQ to neighboring nodes until the RREQ gets to node D. When node D receives this message, it replies with a unicast RREP message via same route it uses to receive the RREQ to node S. Node S receives so many RREP from all the neighboring nodes, but it only picks the one which has the highest sequence number.

Route Maintenance Module: HELLO messages are used to check if the source node and destination node are active in connection-wise. From time to time, each node transmits the HELLO message to its neighbor and waits for a reply. If these messages are received in two ways (i.e. different direction), a symmetrical link is maintained through this way. But if there is an interruption and cannot be repaired, node S sends a Route Error (RERR) message.

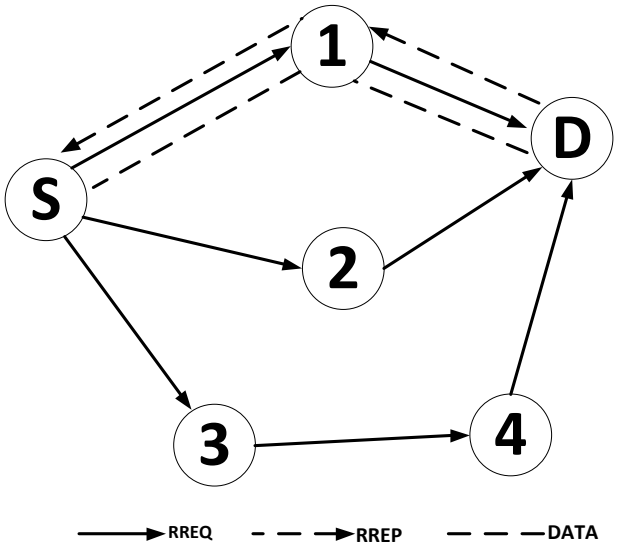

Figure 2.2: Route discovery in AODV [26].

2.4 Black Hole Attacks in AODV Routing Protocol

Routing protocols for MANETs are vulnerable to a variety of attacks. Black hole attack is one of these attacks. In [2, 27], black hole attacks are categorized as DoS attack, which makes a malicious node take advantage of susceptible route discovery packet of the routing protocol so as to broadcast itself of having the closest path to a destination node. Since our study is focused on AODV protocol, we will simulate and show how a malicious node can affect the performance on this protocol. The black hole in AODV routing protocol is distinguished as internal or external attack. Internal attack occurs when the source node forwards a packet to the destination node

but the malicious node is in between these two nodes. It is difficult to manage this attack because it becomes an active data route. For the external black hole attack, the malicious nodes do not have previous route but seek to fit in the effective route between source node and destination node. According to [5], there are no security mechanisms in AODV protocol. Since malicious node uses the AODV protocol, it can perform various attacks.

A black hole attack can work as a misbehaved node as well as a multiple misbehave nodes. In [7], malicious node does not need to consult the routing table, its primary function is to modify the routing protocol so that traffic flows through it. AODV routing includes the following steps when there is a black hole:

Step 1: The source node floods the network with the RREQ control messages which contain the source address, request ID, source sequence number, destination address, destination sequence no, and hop count, in order to seek a route to its target node for the transmission of packets. The initialization of the variables and parameters is given in the “baodv.h” script provided in Appendix A.2.1

Step 2: The neighboring nodes receive the RREQ control messages, check from the routing tables, and if no such route is found in the routing table, forward the RREQ control to appropriate path to reach the destination.

Step 3: Once the malicious node receives RREQ control message, it immediately generates a false RREP control message since it does not waste time to check the routing table with the highest sequence number.

Step 4: Malicious node forwards the false RREQ control message to the source node and claims to have the fresh route to the destination node.

Step 5: Source node then compares the sequence numbers in the RREP control messages to selects the fresh route with the highest sequence number coming from the malicious node. Source node trusts this route and believes it will forward the packets to the destination node.

Step 6: Packets are transmitted through the malicious route.

Step 7: Malicious node drops these packets and never forwards the packets to the destination.

Assume that node 1 in Figure 2.2 becomes a malicious node and named as node X, node S is the source node, node D is the destination node and nodes 2, 3, 4 are the intermediate nodes as shown in Figure 2.3. The source node transmits an RREQ message to the closest nodes to get the fresher path to reach destination node D.

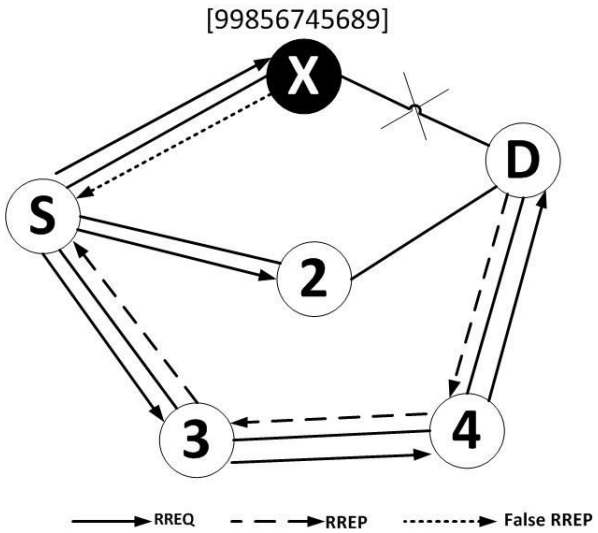

Figure 2.3: Black hole attack.

Immediately, the malicious node X responds to the source node. Since source node S does not consult the routing table, it ignores every RREP message from its neighboring node even they are reliable. With the assumption that route discovery process is finished, it accepts the malicious route as a priority so as to send a data packet through this route. How the malicious node X processes and responds to the RREP control message are explained in the next session. Data packets are received via the malicious node X and then dropped instead of forwarded to the destination node D as the protocol requires.

In the implementation section, the sequence number is used. Figure 2.4 shows the sequence number of each nodes (node X = [99856745689], node 2 = [24] and node D = [34]). These destination sequence numbers are classified as a 32-bit integer related to every route and are used to decide the clarity of a particular path. To determine the fresher route depends on the larger the sequence number is [28]. The destination sequence numbers are shown in square brackets in Figure 2.4. Node S broadcast an RREQ control message to all neighboring nodes in network, in order to discover a path to node D. Node 3, node X and node 2 do not have a direct route to node D. Node 3 then re-broadcasts it RREQ control message in order to identity the path to node D. Malicious node X receives the broadcast RREQ control message and then creates a deceptive RREP control message and sends out to the node S by an abnormal expected destination sequence number. Since, this is the highest destination sequence number, node S considers it to be fresher, and sends a packet to node X. An RREQ control message send by node 3 arrives at node D which creates a RREP control message and routes it back. Since node S receives an RREP control message from both malicious node X and intermediate node 3, source node S first checks if there is an existing entry for destination node D in the table or not. If there

is an entry in table, a cross check of sequence number is done by source node S. It reviews and selects the RREP message with the higher sequence number. Immediately, a new RREP control message is updated by source node S. An RERR control message is generated if there is any disconnection between the nodes during packets transfer.

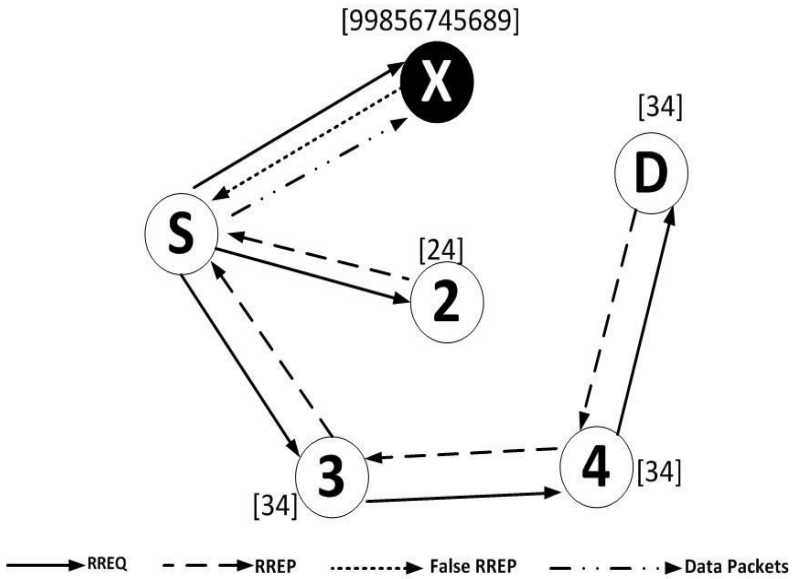

Figure 2.4: Transmission of destination sequence numbers.

2.5 Black Hole Attacks in Modified AODV Routing Protocol

In [3], the author observes that an early RREP control message arrives to source from the malicious node, since malicious does not consult the routing table, while the second RREP control message arrives from the original destination node. The following algorithm presents the proposed solution (modified AODV) for the black hole attacks.

Step 1: The source node floods the network with the RREQ control messages which contain the source address, request ID, source sequence number, destination address, destination sequence no, and hop count, in order to seek a route to its target node for

the transmission of packets. The initialization of the variables and parameters is given in the “idsadov.h” script provided in Appendix A.3.1.

Step 2: The neighboring nodes receive the RREQ control messages, check from the routing tables, and if no such route is found in the routing table, forward the RREQ control to appropriate path to reach the destination.

Step 3: The destination node receives the RREQ control message, and generates the RREP control message which consists of the source address, destination address, destination sequence number, hop count, and life-time.

Step 4: The destination node then unicasts the RREP control back to the same route used for the RREQ message. Source node receives the RREP control from all intermediary nodes.

Step 5: The source uses the first RREP message to initiate the data transfer.

Step 6: If a second RREP message arrives then switches to the new route since it is assume that the black hole does not waste time to check the routing table and RREP message arrives from the black hole earlier than the actual destination.

Chapter 3

NETWORK SIMULATOR AND TOOLS

In this chapter, how NS-2 packages and programming language are used together with Tool Command Language (TCL) and AWK script files used for writing a network scenario and data collection are presented.

3.1 NS-2 Network Simulator

NS-2 is a freeware network simulation software, developed by the University of California, Berkeley. The main idea is to analyze the improvement of wide-ranging networks and present interactive behavior of network protocols. It is used to simulate both wired and wireless networks, Transmission Control Protocol / Internet Protocol (TCP/IP), routing, and multicast protocols studies. Researchers can find, examine and modify its main source code easily. Additionally, researchers all over the world expand and update on its features, and add new protocols. It is currently one of the most widely used simulation software in the field of network. Since 1995, NS-2 has been supported by Xerox Corporation which has joined VINT project [24]. NS-2 as a discrete event-driven, and object-oriented network simulator can fully simulate the entire network environment. As shown in Figure 3.1, as a first step for a NS-2 process, the network topology is declared, and component configurations are designed in TCL script file that is programmed in object orientated language C++.

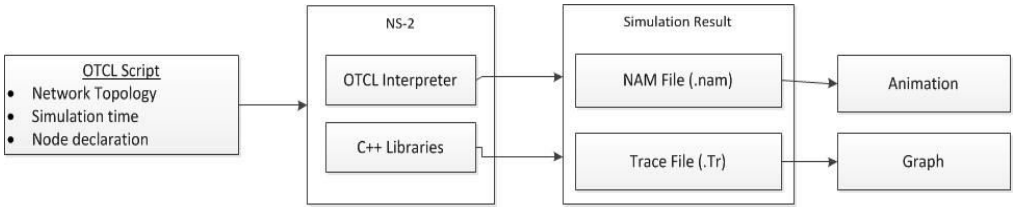

Figure 3.1: NS-2 processes.

In the second step, a group of C++ class library is used for the vast majority of common network protocols, and model the link layer. The Object Oriented Tool Command Language (OTCL) interpreter is used as an intermediary between the TCL script file and the C++ language. It has a high degree of simulation events, network element object libraries and event scheduler. The event scheduler triggers an event in the event queue and executes the event.

During the third step of interpretation process, where the TCL script file is interpreted and simulated, NS generates two scripts which are used to analysis the experiment results. Those two files are the trace file with an extension .tr which is shown in Appendix B.3 and Network Animator (NAM) with extension .nam. NAM displays the animated behavior of nodes in the simulation and the trace file shows the digital information and behavior of all nodes in the simulation. For the purpose of our studies, we used NS 2.29 version and installed on Oracle virtual box Windows environment. In our studies, after identifying the number of nodes and parameters to use, we design our network with C++ code but in TCL format as seen in Appendix B.1. Then we analysis the trace file and obtain the network performance metric with AWK script file as shown in Appendix B.5. To execute NS-2 for a particular case such as AODV protocol with black hole, the steps given below are followed:

Step 1: Configure the network topology, by declaring and initializing the parameters and variables using the TCL script file, and save this script file as BlackHoleAODV.tcl shown in Appendix B.1 in a folder (exper1) on desktop.

Step 2: From the terminal, type “cd Desktop/exper1” and press enter. “Desktop” is the root directory while “exper1” is the folder name.

Step 3: Type “ns BlackHoleAODV.tcl” and press enter to execute the file in NS-2. After a while, the terminal shows the simulation is finished.

Step 4: Two files as trace file (.tr) and animation file (.nam) are generated in “exper1” folder.

Step 5: Type nam followed by the name of the nam file (e.g. nam BlackHoleAODV.nam) to view the animation and press enter.

3.2 TCL Language Script in NS-2

TCL was created by John Ousterhout in 1988 at University of California, Berkeley. According to [29], TCL is classified to be very powerful C programming language. It is used mostly by the programmer and computer related sector that includes the web, networking, administration and desktop applications. TCL has a compatibility with C programming language which can be embedded in a system platform. Appendix B.1 shows the TCL code that is used to design the network topology and the black hole attack in the network. To run the TCL file in the terminal, we type “ns BlackHoleAODV.tcl” command in the root folder “ns-allinone/ns-2.29”. NS indicates the simulation software and BlackHoleAODV.tcl indicates the TCL file.

3.3 AWK Script File

AWK is another powerful interpreted programming language tool, mainly designed for text processing, but its primary function is for data extraction. AWK, the acronym that stands for Alfred Weinberger Kernighan, defines the last names of the developers. AWK was developed at Bell Labs by 1970 with the help of Alfred Aho, Peter Weinberger, and Brian Kernighan [25]. In our studies, a large volume of data spaces are required with the generated trace files after the simulation. For collecting these data from the trace files, we write a C code in AWK environment to process this function. To execute the AWK script file, we open the terminal and then type "AWK -f calculation.awk out.tr", where out.tr is the trace file as shown in Appendix B.3 and calculation.awk is the C++ code for calculating the performance metric as shown in Appendix B.5. To save the results in a text file, the greater than (>) symbol must be used followed by name of the file such as "AWK -f calculate.awk out.tr > result".

Chapter 4

METHODOLOGY

4.1 Implementing a New Routing Protocol with Black Hole

In [30], the author discusses how to implement AODV routing protocol in NS-2. In our study, we follow the same steps as mentioned in that paper. We have created a node as a black hole in MANET. This malicious node uses our implemented AODV routing protocol algorithm.

To proceed with our studies, we install the protocol in a directory named as "ns-2.29". We paste AODV protocol in the same directory in "ns-2.29" folder, and rename the duplicated AODV folder as "bhaodv". We now have the "bhaodv" folder, and then we alter the name of all files with "aodv" in this particular folder to "bhaodv" such as "aodv.cc" as "bhaodv.cc", "aodv.h" as "bhaodv.h" and so on. Furthermore, we rename all the functions and classes, structs, variable and constants in the "bhaodv" directory. In our studies, we do not modify the "aodv_packet.h". The reason is that "AODV" and "bhaodv" exchange same AODV packets. So there is no need for this modification in the "bhaodv" directory. After making all changes as mentioned, we then start with the implementation by adding the name of the protocol "bAODV" in the "\tcl\lib\ ns-lib.tcl" file directory as shown in Figure 4.1. This modification is meant for the protocol agent. When any node behaves like a malicious node, this code is scheduled and assigned from the start of the simulation to this node.

```
bAODV {
set ragent [$self create-baodv-agent $node]
}
Simulator instproc create-baodv-agent { node } {
set ragent [new Agent/bAODV [$node node-addr]]
$self at 0.0 "$ragent start" # start BEACON/HELLO
Messages
$node set ragent_ $ragent
return $ragent
}
```

Figure 4.1: bAODV protocol agent.

The next step after modifying the above file is to modify "\makefile" in the "ns-2.29" directory. As shown in Figure 4.2, we need to make changes by copy and paste the text below under the "aodv" protocol in the "makefile". At this point, we have implement a new protocol. In the next step we implement a black hole.

```
baodv/baodv_logs.o baodv/baodv.o \
baodv/baodv_rtable.o baodv/baodv_rqueue.o \
```

Figure 4.2: bAODV code in the makefile.

Figure 4.3 shows a code which we have to add in the "baodv/baodv.cc" file. With this code, whenever a packet is sent to "recv" method in the "aodv/aodv.cc", it receives this packet and processes such packet based on its categories. If this packet is AODV type, it forwards such packet to the "recAODV" method. In normal AODV without any modification, it forwards any data packet to its destination, but if such protocol is a black hole, this data packet is dropped. Figure 4.3 displays a C++ code which is given in Appendix A.2.2, used by the malicious node, where in the "if statement" it tells the node to receive the packets if it is the destination otherwise drop them.

```
if ( (u_int32_t)ih->saddr() ==index)
forward((baodv_rt_entry*) 0, p, NO_DELAY);
else
drop(p, DROP_RTR_ROUTE_LOOP);
```

Figure 4.3: C++ code used by the malicious node to drop packet.

As explained in Chapter 2, when the black hole node receives the RREQ control message, it replies with a false RREP without delay as a fresh route to the destination node. For a node to act in this way, Figure 4.4 shows the code we modify presented in Appendix A.2.2. We change the sequence number in the "recvRequest" message to an unexpected number such as 99856745689 and we set the hop count to 1. After all the steps explained are finished, we need to recompile all the files in NS-2 by opening the terminal and typing the following commands followed by the enter button.

Command 1: "cd/ns-allinone/ns-2.29"

Command 2: "make clean"

Command 3: "make"

If there is no error message after entering all these commands, the black hole protocol is installed successfully.

```
sendReply(rq->rq_src,          // IP Destination
1,                             // Hop Count
index,                        // Dest IP Address
99856745689,                 // Highest Dest Sequence Num
MY_ROUTE_TIMEOUT,             // Lifetime
rq->rq_timestamp);           // timestamp
```

Figure 4.4: C++ code format to modify the sequence number.

4.2 System Requirement

In our studies, we use a Toshiba laptop and installed a virtual machine box which runs in a Windows 7 environment like a Linux operating system (Ubuntu 10.0 version). The laptop specifications are as follows: Processor: Intel(R) Core(TM) i7 CPU Q 720 @ 1.60GHz , RAM: 8.00 GB, system type: 32-bit. Due to large volume of the trace file during each simulation, our system requires a 2 GB RAM of memory and a large storage of hard disk in Windows operating system.

4.3 Basic Simulation Process

NS-2 has basic operation processes as shown in Figure 4.5. Throughout the simulation, there are three main phases: The first phase is modifying the source code. This is only possible if an implementation is applied to the protocol. The second one is writing the TCL simulation script. This is the most important part and essential component of the simulation. As described in Chapter 3, TCL script is written in C++ code, in which we describe the network topology, properties, network components, simulation event scheduling start and stop of the simulation. The third one is the analysis of the trace file. It uses the AWK script file to collect results and corresponding data that can be used to plot respective graphs.

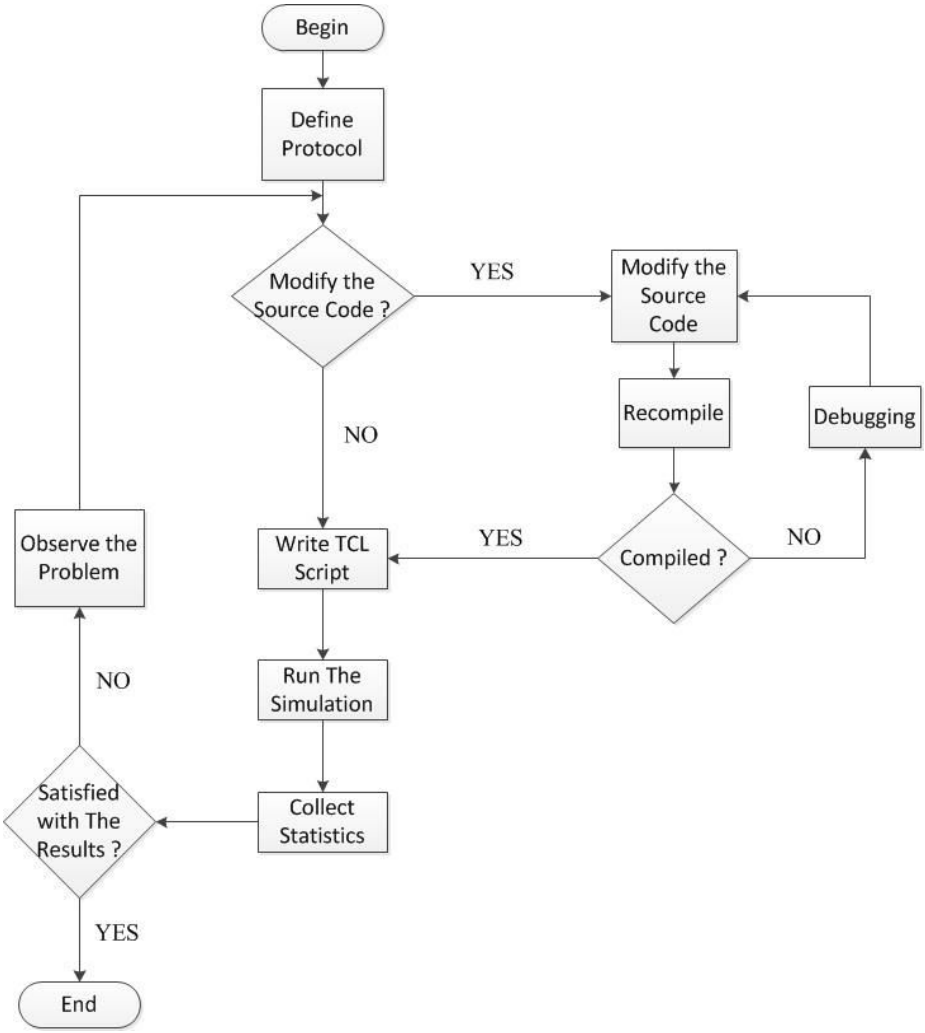

Figure 4.5: A flow chart on how to implement and execute simulation in NS-2.

4.4 Simulation Parameters

In this session, we explain the parameter setup of the simulation we use in our studies. To conduct a simulation in wireless network mode, some information about the node components are required. NS-2 uses the default values if any component are not declared or assigned. The mobile nodes in NS-2 components consist of Interface Queue (IfQ), Media Access Control (MAC) layer, Link Layer (LL) and Physical (PHY) layer. These components are used by the nodes to transmit and receive signals in the wireless channel. From these components, each has a definition of different

parameter like the routing protocol, antenna type and the radio propagation model. Examples of wireless parameters are as follows:

Routing protocol: AODV, DSR and TORA.

MAC: Code Division Multiple Access (CDMA), Institute of Electrical and Electronics Engineers (IEEE) 802.xx.

PHY layer: Omnidirectional antenna and directional antenna.

Quality of Service (QoS): Differentiated Services (DiffServ), Integrated Services (IntServ) and Resource Reservation Protocol (RSVP).

Radio propagation model: Two ray ground radio propagation channel.

Mobility model: Random mobility model.

For the purpose of this study, we used AODV routing protocol, IEEE 802.11 MAC and omnidirectional antenna for PHY layer, DiffServ as QoS, two ray ground radio propagation channel model and a random mobility model. Finally, we used User Datagram Protocol (UDP) at the transport layer and all data packet are in Constant Bit Rate (CBR). As shown in Appendix B.1, a UDP connection is integrated between the even nodes and the odd nodes. So that 0, 2, 4, 6, 8, 10, 12, 14 and 16 are source nodes while 1, 3, 5, 7, 9, 11, 13, 15 and 17 are destination nodes. The communicating pairs are defined as 0 =>1, 2=>3, etc. in a total of 9 pairs. Node 18 and node 19 are made standby without no connection with each other or to another node. They act as an intermediate node. We attach UDP agent to the sending nodes while a NULL

agent are attached to the receiving nodes. We use a for loop statement to create the communication between the nodes. But for the coordinates of each node we use "./setdest" to get an accurate result. The third party application of NS to generate a random movement and positions for each node we save each movement and position of a particular simulation in the simulation directory root as "./move1". A sample of the movement and position file is shown in Appendix B.2. As shown in Table 4.1, we configure our network topology with the following parameters and values. An area of 750×750 m^2, 20 nodes is generated using a "for loop" statement. A simulation time of 500 seconds (s), which starts from begin to the end of the simulation. In each scenario of our study, we configure the source node and destination node to start packet transmission at the first seconds of each scenario. The CBR parameter is set as a packet size of 512 bytes and the data rate of 10 Kbits/s. The UDP connection stops at 450 seconds of the simulation time.

Table 4.1: Parameters for simulation.

Parameter	Value	
Routing Protocol	AODV	
Packet Type	UDP	
Traffic Application	CBR	
Pause Time	1 s	
Simulation Time	500 s	
Network Area	750×750 m^2	
Packet Size	512 Bytes	
Data Rates	10 Kbits/s	
Numbers of Malicious Node	1 (Node 19)	2 (Node 18 & 19)
Total Nodes	20	

4.4.1 Performance Metrics

Analyzing the performance metric of our simulation is to measure the network protocol and performance, which is divided into two: the qualitative and quantitative

metrics. Qualitative metrics describe the overall performance of the network in one area such as security and distribution operation. For quantitative metric, it is described in certain aspects of the performance of network such as PDR, EED, packet loss rate, normalized routing overhead, network jitter and so on. Table 4.2 shows the parameters, performance metric and computations including the definitions used in the analysis.

Table 4.2: Formula and calculation of performance metrics.

Parameters / Metric	**Definition**	**Computation**
Packets Sent (PS)	Total number of packets sent by source node	Extracted from trace file
Packets Received (PR)	Total number of packets received by destination node	Extracted from trace file
Packet Delivery Ratio (PDR)	Ratio of packets received to packets sent	(PR/PS)×100
Loss %	Total loss in the system	100 - PDR
Black hole loss %	Total loss at black hole	(Black hole dropped/ PS) × 100
Total Sent Time (TST)	Total time needed to send packets to destination nodes	Extracted from trace file
Total Arrival Time (TAT)	Total time it takes destination node to receive packet	Extracted from trace file
Total Connection (TC)	Total number of connection	Extracted from trace file
End-to-End Delay (EED)	Time spent on a packet to deliver	(TAT - TST) / TC

EED is the average time taken for a packet to be transmitted from the source to destination. It is in milliseconds (ms) and it includes data waiting on queue for transimission and route discovery.

For the purpose of our studies, we use PDR and EED as performance metric. In Table 4.2 , PS is the total number of packets sent by the source node, while PR is the total number of packets received by destination node. To identify if a packet is sent or received, depends first figure on column in the trace file as shown in the following example.

s -t 1.000000000 -Hs 0 -Hd -2 -Ni 0 -Nx 232.25 -Ny 314.29 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 1.0 -It cbr -Il 512 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

r -t 1.000000000 -Hs 0 -Hd -2 -Ni 0 -Nx 232.25 -Ny 314.29 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 1.0 -It cbr -Il 512 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

The above example shows that packet is sent at 1.000000000 (-t) second by node 0 (–Hs) to node 2 (–Hd), node id (-Ni) = 0 is at *x* postion of 232.25 (-Nx) and *y* position of 314.29 (-Ny) and *z* position of 0 (-Nz), the energy level is 1.000000 (-Ne), trace level (-Nl) is Agent (AGT), no reason for this event (-Nw), duration (-Ma) = 0, destination ethernet address (–Md) = 0, source ethernet address (–Ms) = 0, ethernet type (-Mt) = 0, source address and port (-Is) = 0.0, destination adress and port number (–Id) = 0.0, packet type (–It) = CBR, packet size (–Il) = 512, flow id (–If) = 0, unique id (–Ii) = 0 Ii), Time To Live (TTL) (-Iv) value = 32, how many nodes

traversed (–Pn) = CBR, reply length (-Pi) = 0, numbers of times this packet was forwarded (–Pf) = 0 and optimal number of forwards (-Po) = 16777215.

The total number of packets sent denoted by *s* on first column of the trace file is assigned to PS, while total number of packets recieved denoted by *r* on first column of the trace file is assigned to PR.

PDR is calculated as PR divded by PS multiply by 100. For ad-hoc network, the higher the PDR, the better the result is. To get more accurate results we take the average of 100 simulations for a particular event.

4.5 Testing and Evaluation Performance

To define the duration of the simulation time, several simulation tests on the system are carried on for 1,000 seconds without a black hole. We use the parameters shown in Table 4.1 with varying simulation time. Figures 4.6 shows a graph of PDR plotted against simulation time of 1,000 seconds without black hole. Figure presents that PDR fluctuates up to 20 seconds, and then remains almost constant at 100% from 20 seconds to the end of the simulation time. The experiment is repeated a number of times and similar results are observed that the system becomes stable around 20 seconds. Therefore, in our simulations we use simulation time as 500 seconds and repeat each set 100 times and use average values in our graph as it is done in [3].

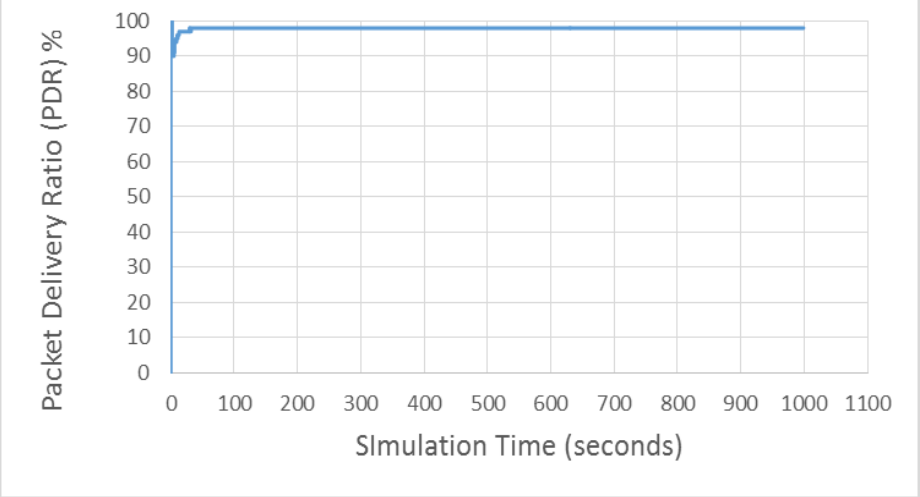

Figure 4.6: PDR versus 1,000 s simulation time in AODV without black hole.

4.6 Analyzing the Trace File and Performance Metric Results

After implementing and execution of simulation parameter as in Chapter 3, the output of the TCL script file produces an extension file that has .tr. In this trace file, all data is available to analysis the performance of the network we simulate. In our simulation output, we use the new trace file format which includes more details of the event. A sample of the new trace files is shown in Appendix B.3. As shown in Appendix B.4, we analysis each field in the trace file and explain each function.

Chapter 5

SIMULATION RESULTS

5.1 Simulation of Original AODV

We use the original code that is installed with the network simulation tool. However, in this section we test and compare the black hole implemented protocol against the original.

5.1.1 Simulation of Original AODV without Black Hole

With the absence of the malicious node, in this particular simulation 18 out 20 nodes communicate with each other, leaving nodes 18 and 19 as free agents that act as any intermediate node. The connections are done even against odd (node 0 is sender while node 1 is the receiver etc.) as explained in Chapter 4. In the TCL script file, there are two steps to configure an AODV routing protocol:

Step 1: Declare the routing protocol as variable “set rp AODV ;”

Step 2: Call this variable by reference “$ns_ node-config -adhocRouting $val (rp) \”.

We use a for loop statement to create nodes under this line. Nodes adapt to this protocol without delay. We conducted 100 scenarios (runs) for original AODV without a black hole. In each scenario, the coordinates and mobility of nodes change.

We explained on how to collect data from the trace file in Chapter 3. Table 5.1 shows an average of 98.31% as PDR. The table also displays the number of packets sent and received for each node.

Table 5.1: Average simulation results for AODV without black hole.

Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1036.40	1019.71	1.61
Node 2 -> Node 3	1042.30	1027.16	1.45
Node 4 -> Node 5	1039.67	1014.30	2.44
Node 6 -> Node 7	1023.32	1010.82	1.22
Node 8 -> Node 9	1035.69	1021.93	1.33
Node 10 -> Node 11	1001.24	982.07	1.91
Node 12 -> Node 13	1013.53	998.68	1.47
Node 14 -> Node 15	1032.13	1010.47	2.10
Node 16 -> Node 17	1031.62	1014.70	1.64
TOTAL	**9255.90**	**9099.84**	1.69

5.1.2 Simulation of AODV with Single Black Hole

In the section, we include a malicious node in the network. It sends a false RREP, absorbs the packet and never forwards this packet to the destination node. Figure 5.1 shows how we create a malicious node in the network by modifying the TCL script file. All nodes assigned with "$ns_ node-config -adhocRouting blackholeAODV \" code act as malicious nodes. For loop statement is used to create the node, which counts from 18 to 19. By that way node 18 is made to behave as a malicious node. We modify the properties of the malicious node by assigning a red color to identify it during the animation. The simulation is conducted 100 times.

After executing the simulation and collecting data with the AWK script file we evaluate the results on average of PDR. Appendix C.2 shows the first five rounds of simulations. Table 5.2 shows that the average PDR is very low, which is 19.30%.The results also show that most of the packets are absorbed by the malicious node. The lost at the malicious node is 51.11% which is very high.

```
# Creating mobile AODV nodes for simulation

puts "Creating nodes..."
# create nodes
for {set i 0} {$i < 19} {incr i} {
  set node_($i) [$ns_ node]
   $node_($i) random-motion 1;
}
$ns_ node-config   -adhocRouting blackholeAODV
for {set i 19} {$i<20 } {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 1;
    $ns_ at 0.01 "$node_($i) label \"blackhole node\""
    $node_($i) color red
    $ns_ at 0.01 "$node_($i) color red"

}
```

Figure 5.1: Code for creating both AODV and black hole nodes in TCL file script.

Table 5.2: Average simulation results of original AODV with one black hole (node 18).

Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1061.53	185.26	545.83	82.55	51.42
Node 2 -> Node 3	1076.84	165.57	622.21	84.62	57.78
Node 4 -> Node 5	1056.66	247.85	509.49	76.54	48.22
Node 6 -> Node 7	1084.17	221.61	493.47	79.56	45.52
Node 8 -> Node 9	1054.57	265.77	509.22	74.80	48.29
Node 10 -> Node 11	1067.40	122.22	547.63	88.55	51.31
Node 12 -> Node 13	1081.49	208.64	548.34	80.71	50.70
Node 14 -> Node 15	1078.20	222.50	534.99	79.36	49.62
Node 16 -> Node 17	1054.15	216.74	602.43	79.44	57.15
TOTAL	**9615.01**	**1856.16**	**4913.61**	**80.70**	**51.11**

5.1.3 Simulation of Original AODV with Two Black Holes

In this section, we include two malicious nodes (node 18 and 19) in original AODV. Appendix C.1 shows the outcomes of the simulation. Table 5.3 displays the

effectiveness of two malicious nodes on PDR. The total number of absorbed packets by both malicious nodes is 65.4% which is very high, and PDR displays low result of 8.77%. These results show that malicious nodes are threat to the system. In the next section, we implement a solution to increase the PDR in the presence of malicious node.

Table 5.3: Average results of original AODV with two black holes (nodes 18 and 19).

Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop (18)**	**Black Hole Drop (19)**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1055.29	123.19	365.87	293.26	88.33	62.46
Node 2 -> Node 3	1073.79	59.52	333.29	426.14	94.46	70.72
Node 4 -> Node 5	1038.91	78.55	342.77	354.61	92.44	67.13
Node 6 -> Node 7	1061.27	88.24	296.15	422.79	91.69	67.74
Node 8 -> Node 9	1064.62	133.02	325.80	363.43	87.51	64.74
Node 10-> Node 11	1057.22	91.87	402.17	288.52	91.31	65.33
Node 12-> Node 13	1077.22	122.25	255.95	367.13	88.65	57.84
Node 14-> Node 15	1049.74	60.02	323.93	401.81	94.28	69.14
Node 16-> Node 17	1084.71	81.98	343.80	346.91	92.44	63.68
TOTAL	**9562.77**	**838.64**	**2989.73**	**3264.60**	**91.23**	**65.40**

Table 5.4: Comparison of performance metrics for original AODV.

Metric	**Without Black Hole**	**With One Black Hole (Node 19)**	**With Two Black Hole (Nodes 18 & 19**
PDR (%)	98.31	19.30	8.77
Black Hole Lost %	-	51.11	65.40
EED (ms)	0.15	0.28	0.31

In Table 5.4, PDR, EED and lost percentage at the black hole results are presented for the original AODV without and with one and two black holes. The results show

that the loss percentage increases as the number of black holes increase in the system while there is a marginal increase in the EED.

5.2 Simulation of Modified AODV

In the previous section and Chapter, we explained how to simulate the original AODV in NS-2, and then we obtained results from the trace file. We implemented the original AODV in NS-2 with one and cooperative black hole. In this section, we present the modifications on the original AODV to apply the solution (modified AODV) for the black hole attacks.

To implement modified AODV, the steps given below are followed. Firstly, we copy the entire folder of original AODV, and rename it as “idsaodv”, then follow the explanations in Chapter 4 for implementation of new routing protocol. Before executing the “make command” from the terminal, we implement the following solution first, then execute the commands. In Chapter 4 to implement a black hole in the system, the RREP method (recvRequest) is modified in “baodv.cc” file. For the required solution implementation, we need to modify the RREP method (recvReply) in the “idsaodv/idsaodv.cc” file.

Secondly, we need to create and implement a caching mechanism used by RREP message control, which assigns an identification number on each RREP control message, then focus on the second RREP message. Appendix A.3.2 shows the caching mechanism used to capture the second message. Explanation for all methods include in the caching mechanism for RREP control message (recvReply) as follows:

“rrep_insert”: Is used for adding incoming RREP message.

"rrep_lookup": Is used for checking the table if there is any duplicated or existing RREP control message.

"rrep_remove": Is used to delete any RREP control message recieved from a defined node.

"rrep_purge": Is used periodically to delete expired time from the list.

```
idsAODV::recvReply(Packet *p) {
idsBroadcastRREP * r = rrep_lookup(rp->rp_dst);
if(ih->daddr() == index) {
if (r == NULL) {
count = 0;
rrep_insert(rp->rp_dst);
} else {
r->count ++;
count = r->count;
}
UPDATE ROUTE TABLE
} else {
Forward(p);
}
}
```

Figure 5.2: RREP method used in the "idsaodv.cc" file.

Figure 5.2 displays the RREP control message used in the "idsaodv.cc" presented in Appendix A.3 that consists of seven steps. We discuss these steps as follows:

Step 1: Triggers the "rrep_lookup" function to check if an RREP control already exists, then assigns the result to variable *r*.

Step 2: Checks if the RREP control message is received by the same destination node (YES or NO).

Step 3: If Step 2 is NO forwards to approiate node.

Step 4: If Step 2 is YES checks the variable in Step 1 if it is empty (YES or NO).

Step 5: If Step 4 is NO assigns the variable in Step 1 to counter.

Step 6: If Step 4 is YES sets counter to zero, then triggers the “rrep_insert” function to add incoming RREP control message.

Step 7: Updates the routing table.

5.2.1 Simulation of Modified AODV without Black Hole

In the TCL script, we configure the routing protocol to “idsAODV” by typing “$ns node-config -adhocRouting idsAODV”. All nodes declared under the configuration adapt to idsAODV routing algorithm as mentioned in previous chapter. The coordinates and movements of all nodes are generated with a third party software as mentioned in Chapter 4. Appendix B.2 shows an example of mobility and coordinates of each node. Table 5.5 shows the average results of 100 simulations for the modified AODV protocol without a black hole. The average PDR is achieved as 94.85%. Appendix C.4 only displays the first five simulation results.

Table 5.5: Average simulation results of modified AODV without black hole.

Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1006.15	982.06	2.39
Node 2 -> Node 3	1052.31	992.33	5.70
Node 4 -> Node 5	1031.84	980.65	4.96
Node 6 -> Node 7	1006.64	962.92	4.34
Node 8 -> Node 9	1059.61	992.50	6.33
Node 10 -> Node 11	1040.05	958.82	7.81
Node 12 -> Node 13	1051.34	1000.15	4.87
Node 14 -> Node 15	1068.43	1026.63	3.91
Node 16 -> Node 17	1025.72	965.13	5.91
TOTAL	**9342.09**	**8861.19**	**5.15**

5.2.2 Simulation of Modified AODV with One Black Hole

In this section, we include a malicious node and observe the performance of modified AODV protocol. The simulation is conducted 100 times. Appendix C.5

shows the first five of the simulation results which include the number of packets sent and received from source and destination node respectively. Table 5.6 displays the average PDR results for the modified AODV with a black hole node. PDR is 39.19% which shows improvement compare to results of original AODV presented in Table 5.2 where PDR is 19.3%. We also notice in Table 5.6 that the number of packets absorbed and dropped by the malicious node is very low 34.95%.

Table 5.6: Average simulation results of modified AODV with one black hole (node 19).

Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1022.02	456.47	327.57	55.34	32.05
Node 2 -> Node 3	1078.91	427.37	390.58	60.39	36.20
Node 4 -> Node 5	1064.88	391.67	341.63	63.22	32.08
Node 6 -> Node 7	1072.08	426.82	357.18	60.19	33.32
Node 8 -> Node 9	1084.34	480.00	316.27	55.73	29.17
Node 10 -> Node 11	1065.37	444.54	408.54	58.27	38.35
Node 12 -> Node 13	1056.82	361.89	410.27	65.76	38.82
Node 14 -> Node 15	1051.21	340.51	416.24	67.61	39.60
Node 16 -> Node 17	1067.79	418.18	373.95	60.84	35.02
TOTAL	**9563.42**	**3747.45**	**3342.23**	**60.81**	**34.95**

Table 5.7 shows the comparison of two different performance metrics (PDR and EED) with and without black hole for modified AODV.

Table 5.7: Comparison of performance metrics results for modified AODV

Metric	**Without Black Hole**	**With One Black Hole (Node 19)**
PDR (%)	94.85	39.19
Black Hole Lost %	-	34.95
EED (ms)	0.15	0.12

Finally, we investigate the same proposed work in [3] to detect black hole attacks comparing the original AODV with one and two black hole respectively, and modified AODV with one black. We observe that PDR of modified AODV with one black hole that is 39.19%, quadruple the result of original AODV with one black hole that is 8.77%.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Due to security issues in MANETs, various implementations and propose mechanisms have been conducted by many researchers to solve black hole attack problem. In this thesis work, we summarized the categories of routing protocol, AODV routing protocol and black hole attack. The results are presented for the original AODV without black hole and with one and two black holes, the modified AODV without black hole and with a black hole.

In our study, firstly PDR and EED results of original AODV without a black hole attack are obtained. Then AODV protocol with one and two black holes is implemented in NS-2 and results of PDR, EED and loss percentage at the black hole(s) are obtained. It is concluded from the obtained results that PDR without a black hole drops from 98.31% to 19.30% and 8.77% with one and two black holes respectively. Additionally it is noticed that the black hole lost is 51.11% and 65.40% for one and two black holes respectively. The results also show that there is a marginal increase in the EED when black holes are included. Finally the proposed AODV protocol in [3] is implemented with one black hole to improve PDR in NS-2. The modified AODV protocol in [3] assumes that the black hole does not waste time to check the routing table and returns the reply quickly to the source. Because of this in the modified AODV protocol the source initially uses the first reply that is with a

high probability coming from the black hole to transfer the data but when a second reply come it switches to the path defined by that reply. The results of the modified AODV protocol show that PDR with one black hole is improved from 19.30% to 39.19% compared to the results of original AODV. On the other hand, PDR of modified AODV drops slightly to 94.85% compared to the PDR of original AODV which is 98.31% when there is no black hole attack. The results also show that by the modified AODV protocol the loss at the black hole is dropped from 51.11% to 34.95% and EED is dropped from 0.28 ms to 0.12 ms.

In conclusion, we detect the effect of one and two black hole attack in MANETs by analyzing the performance metrics in term of PDR and EED. Based on the existing work and current survey, we see that the black hole problem is still a significant issue in MANETs and more algorithms should be investigated to solve the black hole problem.

6.2 Future Work

In our future work, we plan to analysis more performance metric in terms of throughput and routing overhead. Then we can further implement different methods to deal with the black hole attacks.

REFERENCES

[1] Kushwah, V. S., & Sharma, G. (2010). Implementation of New Routing Protocol for Node Security in a Mobile Ad Hoc Network. *International Journal of Computer Science and Security*, 8(9), pp. 31-36.

[2] Mistry, N., Jinwala, D. C., & Zaveri, M. (2010). Improving AODV Protocol against Blackhole Attacks. *Proceedings of the International Multi Conference of Engineers and Computer Scientists,* 2.

[3] Dokurer, S., Erten, Y. M., & Acar, C. E. (2007). Performance Analysis of Ad-hoc Networks under Black Hole Attacks. *Proceedings of IEEE SoutheastCon,* pp. 148-153.

[4] Ramaswamy, S., Fu, H., Sreekantaradhya, M., Dixon, J., & Nygard, K. E. (2003). Prevention of Cooperative Black Hole Attack in Wireless Ad Hoc Networks. In *International Conference on Wireless Networks,* pp. 570-575.

[5] Tamilselvan, L., & Sankaranarayanan, V. (2007). Prevention of Blackhole Attack in MANET. *2nd International Conference on Wireless Broadband and Ultra Wideband Communications,* pp. 21-21.

[6] Lundberg, J. (2000). Routing Security in Ad Hoc Networks. *Helsinki University of Technology,* available http://web.informatik.uni-bonn.de/IV/

Mitarbeiter/mp/paper/secure_routing/routing%20security%20in%20ad%20hoc%20networks%20-%20lundberg.pdf (last accessed on July 2015).

[7] Jaiswal, P., & Kumar, D. R. (2012). Prevention of Black Hole Attack in MANET. *International Journal of Computer Networks and Wireless Communications,* pp. 599-606.

[8] Weerasinghe, H., & Fu, H. (2007). Preventing Cooperative Black Hole Attacks in Mobile Ad Hoc Networks: Simulation, Implementation and Evaluation. *Proceedings of the Future Generation Communication and Networking,* 2, pp. 362-367.

[9] Sun, B., Guan, Y., Chen, J., & Pooch, U. W. (2003). Detecting Black-hole Attack in Mobile Ad Hoc Networks. *5th European Personal Mobile Communications Conference*, pp. 490- 495.

[10] Al-Shurman, M., Yoo, S. M., & Park, S. (2004). Black Hole Attack in Mobile Ad Hoc Networks. *Proceedings of the 42nd Annual Southeast Regional Conference*, pp. 96-97.

[11] Raj, P. N., & Swadas, P. B. (2009). DPRAODV: A Dynamic Learning System against Blackhole Attack in AODV based MANET. *International Journal of Computer Sciences Issues*, 2, pp. 54-59

[12] Jaisankar, N., Saravanan, R., & Swamy, K. D. (2010). A Novel Security Approach for Detecting Black Hole Attack in MANET. *Information Processing and Management Communications in Computer and Information Sciecnce,* 70, pp. 217-223.

[13] Yu, C. W., Wu, T. K., Cheng, R. H., & Chang, S. C. (2007). A Distributed and Cooperative Black Hole Node Detection and Elimination Mechanism for Ad Hoc Networks. *Emerging Technologies in Knowledge Discovery and Data Mining Conference,* pp. 538-549.

[14] Gonzalez, O. F., Howarth, M., & Pavlou, G. (2007). Detection of Packet Forwarding Misbehavior in Mobile Ad-hoc Networks. *Wired/Wireless Internet Communications,* 4517, pp. 302-314.

[15] Murthy, S., & Garcia-Luna-Aceves, J. J. (1996). An Efficient Routing Protocol for Wireless Networks. *Mobile Networks and Applications*, 1(2), pp. 183-197.

[16] Medadian, M., Mebadi, A., & Shahri, E. (2009). Combat with Black Hole Attack in AODV Routing Protocol. *9th Malaysia International Conference on Communications,* pp. 530-535.

[17] Mandhata, S. C., & Patro, S. N. (2011). A Counter Measure to Black Hole Attack on AODV-based Mobile Ad-hoc Networks. *International Journal of Computer & Communication Technology*, 2(6), pp. 37-42.

[18] Jali, K. A., Ahmad, Z., & Ab Manan, J. L. (2011). Mitigation of Black Hole Attacks for AODV Routing Protocol. *International Journal of New Computer Architectures and their Applications*, 1(2), pp. 336-343.

[19] Saetang, W., & Charoenpanyasak, S. (2012). CAODV Free Blackhole Attack in Ad Hoc Networks. *International Conference on Computer Networks and Communication Systems,* pp. 58-63.

[20] Tseng, F. H., Chou, L. D., & Chao, H. C. (2011). A Survey of Black Hole Attacks in Wireless Mobile Ad Hoc Networks. *Human-centric Computing and Information Sciences*, 1(4), pp. 1-16.

[21] Misra, P. (1999). Routing Protocols for Ad Hoc Mobile Wireless Networks. *Courses Notes,* available at http://www.cse.wustl.edu/~jain/cis788-99/ftp/adhoc_routing/ (last accessed on July 2015).

[22] Bilandi, N., & Verma, H. K. (2012). Comparative Analysis of Reactive, Proactive and Hybrid Routing Protocols in MANET. *International Journal of Electronics and Computer Science Engineering*, 1(3), pp. 1660-1667.

[23] Jhaveri, R. H., Patel, S. J., & Jinwala, D. C. (2012). DoS Attacks in Mobile Ad Hoc Networks: A Survey. In *Second International Conference on Advanced Computing & Communication Technologies,* pp. 535-541.

[24] Virtual InterNetwork Testbed, available at http://www.isi.edu/nsnam/vint (last accessed on July 2015).

[25] Aho, A. V., Kernighan, B. W., & Weinberger, P. J. (1979). AWK - A Pattern Scanning and Processing Language. *Software: Practice and Experience*, 9(4), pp. 267-279.

[26] Kaushik, N., & Dureja, A. (2013). Performance Evaluation of Modified AODV against Black Hole Attack in MANET. *European Scientific Journal*, 9(18), pp. 182-193.

[27] Perkins, C., Belding-Royer, E., & Das, S. (2003). Ad Hoc On-demand Distance Vector (AODV) Routing, No. RFC 3561.

[28] Kurosawa, S., Nakayama, H., Kato, N., Jamalipour, A., & Nemoto, Y. (2007). Detecting Blackhole Attack on AODV-based Mobile Ad Hoc Networks by Dynamic Learning Method. *International Journal of Network Security*, 5(3), pp. 338-346.

[29] Tcl, *Webopedia an Internet Dictionary*, available at http://www.webopedia. com/TERM/T/Tcl.html (last accessed on July 2015).

[30] Roopak, M., & Reddy, B. (2013). Black Hole Attack Implementation in AODV Routing Protocol. *International Journal of Scientific & Engineering Research*, 4(5), pp. 402-406.

APPENDICES

Appendix A: Script Files

Appendix A.1: Original AODV Script (aodv.h)

```
#ifndef __aodv_h__
#define __aodv_h__
/#include <agent.h>
//#include <packet.h>
//#include <sys/types.h>
//#include <cmu/list.h>
//#include <scheduler.h>
#include <cmu-trace.h>
#include <priqueue.h>
#include <aodv/aodv_rtable.h>
#include <aodv/aodv_rqueue.h>
#include <classifier/classifier-port.h>
/*
  Allows local repair of routes
*/
#define AODV_LOCAL_REPAIR
/*
  Allows AODV to use link-layer (802.11) feedback in determining when
  links are up/down.
*/
#define AODV_LINK_LAYER_DETECTION
/*
Causes AODV to apply a "smoothing" function to the link layer feedback
 that is generated by 802.11.  In essence, it requires that RT_MAX_ERROR
 errors occurs within a window of RT_MAX_ERROR_TIME before the link is
considered bad.
*/
#define AODV_USE_LL_METRIC
/*
Only applies if AODV_USE_LL_METRIC is defined.Causes AODV to apply
omniscient knowledge to the feedback received from 802.11.  This may be flawed,
because it does not account for congestion.
*///#define AODV_USE_GOD_FEEDBACK
class AODV;
#define MY_ROUTE_TIMEOUT        10                      // 100 seconds
#define ACTIVE_ROUTE_TIMEOUT    10                              // 50 seconds
#define REV_ROUTE_LIFE          6                       // 5  seconds
#define BCAST_ID_SAVE           6                       // 3 seconds
// No. of times to do network-wide search before timing out for
// MAX_RREQ_TIMEOUT sec.
```

```
#define RREQ_RETRIES            3
// timeout after doing network-wide search RREQ_RETRIES times
#define MAX_RREQ_TIMEOUT     10.0 //sec
/* Various constants used for the expanding ring search */
#define TTL_START     5
#define TTL_THRESHOLD 7
#define TTL_INCREMENT 2
// This should be somewhat related to arp timeout
#define NODE_TRAVERSAL_TIME     0.03            // 30 ms
#define LOCAL_REPAIR_WAIT_TIME  0.15 //sec
// Should be set by the user using best guess (conservative)
#define NETWORK_DIAMETER        30              // 30 hops
// Must be larger than the time difference between a node propagates a route
// request and gets the route reply back.
//#define RREP_WAIT_TIME     (3 * NODE_TRAVERSAL_TIME *
NETWORK_DIAMETER) // ms
//#define RREP_WAIT_TIME     (2 * REV_ROUTE_LIFE)  // seconds
#define RREP_WAIT_TIME        1.0  // sec
#define ID_NOT_FOUND    0x00
#define ID_FOUND        0x01
//#define INFINITY        0xff
// The followings are used for the forward() function. Controls pacing.
#define DELAY 1.0           // random delay
#define NO_DELAY -1.0       // no delay
// think it should be 30 ms
#define ARP_DELAY 0.01      // fixed delay to keep arp happy
#define HELLO_INTERVAL          1               // 1000 ms
#define ALLOWED_HELLO_LOSS      3               // packets
#define BAD_LINK_LIFETIME       3               // 3000 ms
#define MaxHelloInterval        (1.25 * HELLO_INTERVAL)
#define MinHelloInterval        (0.75 * HELLO_INTERVAL)
/*
Timers (Broadcast ID, Hello, Neighbor Cache, Route Cache)
*/
class BroadcastTimer : public Handler {
public:
        BroadcastTimer(AODV* a) : agent(a) {}
        void        handle(Event*);
private:
        AODV    *agent;
           Event   intr;
};
class HelloTimer : public Handler {
public:
        HelloTimer(AODV* a) : agent(a) {}
        void        handle(Event*);
private:
```

```
        AODV    *agent;
           Event   intr;
};
class NeighborTimer : public Handler {
public:
        NeighborTimer(AODV* a) : agent(a) {}
        void        handle(Event*);
private:
        AODV    *agent;
           Event   intr;
};
class RouteCacheTimer : public Handler {
public:
        RouteCacheTimer(AODV* a) : agent(a) {}
        void        handle(Event*);
private:
        AODV    *agent;
           Event   intr;
};
class LocalRepairTimer : public Handler {
public:
        LocalRepairTimer(AODV* a) : agent(a) {}
        void        handle(Event*);
private:
        AODV    *agent;
           Event   intr;
};
/*
  Broadcast ID Cache
*/
class BroadcastID {
        friend class AODV;
 public:
        BroadcastID(nsaddr_t i, u_int32_t b) { src = i; id = b;  }
 protected:
        LIST_ENTRY(BroadcastID) link;
        nsaddr_t        src;
        u_int32_t       id;
        double          expire;         // now + BCAST_ID_SAVE s
};
LIST_HEAD(aodv_bcache, BroadcastID);
/*
  The Routing Agent
*/
class AODV: public Agent {
  /*
   * make some friends first
```

```
 */
    friend class aodv_rt_entry;
    friend class BroadcastTimer;
    friend class HelloTimer;
    friend class NeighborTimer;
    friend class RouteCacheTimer;
    friend class LocalRepairTimer;
public:
    AODV(nsaddr_t id);
    void                recv(Packet *p, Handler *);
protected:
    int         command(int, const char *const *);
    int         initialized() { return 1 && target_; }
    /*
     * Route Table Management
     */
    void        rt_resolve(Packet *p);
    void        rt_update(aodv_rt_entry *rt, u_int32_t seqnum,
                          u_int16_t metric, nsaddr_t nexthop,
                          double expire_time);
    void        rt_down(aodv_rt_entry *rt);
    void        local_rt_repair(aodv_rt_entry *rt, Packet *p);
public:
    void        rt_ll_failed(Packet *p);
    void        handle_link_failure(nsaddr_t id);
protected:
    void        rt_purge(void);
    void        enque(aodv_rt_entry *rt, Packet *p);
    Packet*     deque(aodv_rt_entry *rt);
    /*
     * Neighbor Management
     */
    void        nb_insert(nsaddr_t id);
    AODV_Neighbor*      nb_lookup(nsaddr_t id);
    void        nb_delete(nsaddr_t id);
    void        nb_purge(void);
    /*
     * Broadcast ID Management
     */
    void        id_insert(nsaddr_t id, u_int32_t bid);
    bool            id_lookup(nsaddr_t id, u_int32_t bid);
    void        id_purge(void);
    /*
     * Packet TX Routines
     */
    void        forward(aodv_rt_entry *rt, Packet *p, double delay);
    void        sendHello(void);
```

```
 void            sendRequest(nsaddr_t dst);
void            sendReply(nsaddr_t ipdst, u_int32_t hop_count,
                        nsaddr_t rpdst, u_int32_t rpseq,
                        u_int32_t lifetime, double timestamp);
 void           sendError(Packet *p, bool jitter = true);
 /*
  * Packet RX Routines
  */
 void           recvAODV(Packet *p);
 void           recvHello(Packet *p);
 void           recvRequest(Packet *p);
 void           recvReply(Packet *p);
 void           recvError(Packet *p);
    /*
     * History management
     */
    double              PerHopTime(aodv_rt_entry *rt);
 nsaddr_t        index;                  // IP Address of this node
 u_int32_t       seqno;                   // Sequence Number
 int             bid;                  // Broadcast ID
 aodv_rtable         rthead;                 // routing table
 aodv_ncache         nbhead;                 // Neighbor Cache
 aodv_bcache          bihead;                // Broadcast ID Cache
 /*
 * Timers
  */
 BroadcastTimer  btimer;
 HelloTimer      htimer;
 NeighborTimer   ntimer;
 RouteCacheTimer rtimer;
 LocalRepairTimer lrtimer;
 /*
  * Routing Table
  */
 aodv_rtable          rtable;
 /*
  *  A "drop-front" queue used by the routing layer to buffer
  *  packets to which it does not have a route.
  */
 aodv_rqueue         rqueue;
 /*
  * A mechanism for logging the contents of the routing
  * table.
  */
 Trace           *logtarget;
 /*
  * A pointer to the network interface queue that sits
```

```
        * between the "classifier" and the "link layer".
        */
       PriQueue       *ifqueue;
       /*
        * Logging stuff
        */
       void           log_link_del(nsaddr_t dst);
       void           log_link_broke(Packet *p);
       void           log_link_kept(nsaddr_t dst);
          /* for passing packets up to agents */
          PortClassifier *dmux_;
};
#endif /* __aodv_h__ */
```

Appendix A.2: Original AODV Script with Black Hole

Appendix A.2.1: baodv.h

```
#ifndef __blackholeaodv_h__
#define __blackholeaodv_h__
//#include <agent.h>
//#include <packet.h>
//#include <sys/types.h>
//#include <cmu/list.h>
//#include <scheduler.h>
#include <cmu-trace.h>
#include <priqueue.h>
#include <blackholeaodv/blackholeaodv_rtable.h>
#include <blackholeaodv/blackholeaodv_rqueue.h>
#include <classifier/classifier-port.h>
/*
Allows local repair of routes
*/
#define blackholeAODV_LOCAL_REPAIR
/*
  Allows AODV to use link-layer (802.11) feedback in determining when
  links are up/down.
*/
#define blackholeAODV_LINK_LAYER_DETECTION
/*Causes AODV to apply a "smoothing" function to the link layer feedback that is
generated by 802.11.  In essence, it requires that RT_MAX_ERROR errors occurs
within a window of RT_MAX_ERROR_TIME before the link is considered bad.
*/
#define blackholeAODV_USE_LL_METRIC
```

```
/*
  Only applies if AODV_USE_LL_METRIC is defined. Causes AODV to apply
omniscient knowledge to the feedback received from 802.11.  This may be flawed,
because it does not account for congestion.
*/
//#define AODV_USE_GOD_FEEDBACK
class blackholeAODV;
#define MY_ROUTE_TIMEOUT        10                      // 100 seconds
#define ACTIVE_ROUTE_TIMEOUT    10                              // 50 seconds
#define REV_ROUTE_LIFE          6                       // 5  seconds
#define BCAST_ID_SAVE           6                       // 3 seconds
// No. of times to do network-wide search before timing out for
// MAX_RREQ_TIMEOUT sec.
#define RREQ_RETRIES            3
// timeout after doing network-wide search RREQ_RETRIES times
#define MAX_RREQ_TIMEOUT     10.0 //sec
/* Various constants used for the expanding ring search */
#define TTL_START     5
#define TTL_THRESHOLD 7
#define TTL_INCREMENT 2
// This should be somewhat related to arp timeout
#define NODE_TRAVERSAL_TIME     0.03            // 30 ms
#define LOCAL_REPAIR_WAIT_TIME  0.15 //sec
// Should be set by the user using best guess (conservative)
#define NETWORK_DIAMETER        30              // 30 hops
// Must be larger than the time difference between a node propagates a route
// request and gets the route reply back.
//#define RREP_WAIT_TIME     (3 * NODE_TRAVERSAL_TIME *
NETWORK_DIAMETER) // ms
//#define RREP_WAIT_TIME     (2 * REV_ROUTE_LIFE)  // seconds
#define RREP_WAIT_TIME        1.0  // sec
#define ID_NOT_FOUND    0x00
#define ID_FOUND        0x01
//#define INFINITY        0xff
// The followings are used for the forward() function. Controls pacing.
#define DELAY 1.0           // random delay
#define NO_DELAY -1.0       // no delay
// think it should be 30 ms
#define ARP_DELAY 0.01      // fixed delay to keep arp happy
#define HELLO_INTERVAL          1               // 1000 ms
#define ALLOWED_HELLO_LOSS      3               // packets
#define BAD_LINK_LIFETIME       3               // 3000 ms
#define MaxHelloInterval        (1.25 * HELLO_INTERVAL)
#define MinHelloInterval        (0.75 * HELLO_INTERVAL)
/*
Timers (Broadcast ID, Hello, Neighbor Cache, Route Cache)
*/
```

```
class BlackHoleBroadcastTimer : public Handler {
public: BlackHoleBroadcastTimer(blackholeAODV* a) : agent(a) {}
        void        handle(Event*);
private:
        blackholeAODV    *agent;
           Event   intr;
};
class BlackHoleHelloTimer : public Handler {
public:
  BlackHoleHelloTimer(blackholeAODV* a) : agent(a) {}
    void handle(Event*);
private:
        blackholeAODV    *agent;
           Event   intr;
};
class BlackHoleNeighborTimer : public Handler {
public:
        BlackHoleNeighborTimer(blackholeAODV* a) : agent(a) {}
        void        handle(Event*);
private:
        blackholeAODV    *agent;
           Event   intr;
};
class BlackHoleRouteCacheTimer : public Handler {
public:
        BlackHoleRouteCacheTimer(blackholeAODV* a) : agent(a) {}
        void        handle(Event*);
private:
        blackholeAODV    *agent;
           Event   intr;
};
class BlackHoleLocalRepairTimer : public Handler {
public:
        BlackHoleLocalRepairTimer(blackholeAODV* a) : agent(a) {}
        void        handle(Event*);
private:
        blackholeAODV    *agent;
           Event   intr;
};
/*
  Broadcast ID Cache
*/
class BlackHoleBroadcastID {
        friend class blackholeAODV;
 public:
        BlackHoleBroadcastID(nsaddr_t i, u_int32_t b) { src = i; id = b;  }
 protected:
```

```
        LIST_ENTRY(BlackHoleBroadcastID) link;
        nsaddr_t        src;
        u_int32_t       id;
        double          expire;         // now + BCAST_ID_SAVE s
};
LIST_HEAD(blackholeaodv_bcache, BlackHoleBroadcastID);
/*
  The Routing Agent
*/
class blackholeAODV: public Agent {
  /*
   * make some friends first
   */
        friend class blackholeaodv_rt_entry;
        friend class BlackHoleBroadcastTimer;
        friend class BlackHoleHelloTimer;
        friend class BlackHoleNeighborTimer;
        friend class BlackHoleRouteCacheTimer;
        friend class BlackHoleLocalRepairTimer;
 public:
      blackholeAODV(nsaddr_t id);
        void                    recv(Packet *p, Handler *);
 protected:
        int             command(int, const char *const *);
        int             initialized() { return 1 && target_; }
        /*
         * Route Table Management
         */
        void            rt_resolve(Packet *p);
        void            rt_update(blackholeaodv_rt_entry *rt, u_int32_t seqnum,
                                  u_int16_t metric, nsaddr_t nexthop,
                                  double expire_time);
        void            rt_down(blackholeaodv_rt_entry *rt);
        void            local_rt_repair(blackholeaodv_rt_entry *rt, Packet *p);
 public:
        void            rt_ll_failed(Packet *p);
        void            handle_link_failure(nsaddr_t id);
 protected:
        void            rt_purge(void);
        void            enque(blackholeaodv_rt_entry *rt, Packet *p);
        Packet*         deque(blackholeaodv_rt_entry *rt);
       /*
        * Neighbor Management
         */
        void            nb_insert(nsaddr_t id);
        blackholeAODV_Neighbor*       nb_lookup(nsaddr_t id);
        void            nb_delete(nsaddr_t id);
```

```
void        nb_purge(void);
/*
 * Broadcast ID Management
 */
void        id_insert(nsaddr_t id, u_int32_t bid);
bool            id_lookup(nsaddr_t id, u_int32_t bid);
void        id_purge(void);
/*
 * Packet TX Routines
 */
void        forward(blackholeaodv_rt_entry *rt, Packet *p, double delay);
void        sendHello(void);
void        sendRequest(nsaddr_t dst);
void        sendReply(nsaddr_t ipdst, u_int32_t hop_count,
                  nsaddr_t rpdst, u_int32_t rpseq,
                  u_int32_t lifetime, double timestamp);
void        sendError(Packet *p, bool jitter = true);
/*
 * Packet RX Routines
 */
void        recvblackholeAODV(Packet *p);
void        recvHello(Packet *p);
void        recvRequest(Packet *p);
void        recvReply(Packet *p);
void        recvError(Packet *p);
   /*
    * History management
    */
   double           PerHopTime(blackholeaodv_rt_entry *rt);
nsaddr_t       index;                // IP Address of this node
u_int32_t      seqno;                // Sequence Number
int            bid;                 // Broadcast ID
blackholeaodv_rtable       rthead;              // routing table
blackholeaodv_ncache        nbhead;             // Neighbor Cache
blackholeaodv_bcache        bihead;             // Broadcast ID Cache
/*
 * Timers
 */
BlackHoleBroadcastTimer  btimer;
BlackHoleHelloTimer      htimer;
BlackHoleNeighborTimer   ntimer;
BlackHoleRouteCacheTimer rtimer;
BlackHoleLocalRepairTimer lrtimer;
/*
 * Routing Table
 */
blackholeaodv_rtable        rtable;
```

```
        /*
         *  A "drop-front" queue used by the routing layer to buffer
         *  packets to which it does not have a route.
         */
        blackholeaodv_rqueue        rqueue;
        /*
         * A mechanism for logging the contents of the routing
         * table.
         */
        Trace           *logtarget;
        /*
          A pointer to the network interface queue that sits
         * between the "classifier" and the "link layer".
         */
        PriQueue        *ifqueue;
    /*
         * Logging stuff
         */
        void            log_link_del(nsaddr_t dst);
        void            log_link_broke(Packet *p);
        void            log_link_kept(nsaddr_t dst);
           /* for passing packets up to agents */
           PortClassifier *dmux_;
};
#endif /* __blackholeaodv_h__ */
```

Appendix A.2.2: baodv.cc

```
void
blackholeAODV::recvRequest(Packet *p) {
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
blackholeaodv_rt_entry *rt;
  /*
  * Drop if:
   *      - I'm the source
   *      - I recently heard this request.
   */
  if(rq->rq_src == index) {
#ifdef DEBUG
    fprintf(stderr, "%s: got my own REQUEST\n", __FUNCTION__);
#endif // DEBUG
    Packet::free(p);
    return;
  }
 if (id_lookup(rq->rq_src, rq->rq_bcast_id)) {
```

```
#ifdef DEBUG
  fprintf(stderr, "%s: discarding request\n", __FUNCTION__);
#endif // DEBUG
  Packet::free(p);
   return;
 }
 /*
  * Cache the broadcast ID
  */
 id_insert(rq->rq_src, rq->rq_bcast_id);
/*
  * We re either going to forward the REQUEST or generate a
  * REPLY. Before we do anything, we make sure that the REVERSE
  * route is in the route table.
  */
 blackholeaodv_rt_entry *rt0; // rt0 is the reverse route
   rt0 = rtable.rt_lookup(rq->rq_src);
   if(rt0 == 0) { /* if not in the route table */
   // create an entry for the reverse route.
     rt0 = rtable.rt_add(rq->rq_src);
   }
   rt0->rt_expire = max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE));
   if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
           ((rq->rq_src_seqno == rt0->rt_seqno) &&
            (rq->rq_hop_count < rt0->rt_hops)) ) {
   // If we have a fresher seq no. or lesser #hops for the
   // same seq no., update the rt entry. Else don't bother.
rt_update(rt0, rq->rq_src_seqno, rq->rq_hop_count, ih->saddr(),
                max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE)) );
     if (rt0->rt_req_timeout > 0.0) {
     // Reset the soft state and
     // Set expiry time to CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT
     // This is because route is used in the forward direction,
     // but only sources get benefited by this change
       rt0->rt_req_cnt = 0;
       rt0->rt_req_timeout = 0.0;
       rt0->rt_req_last_ttl = rq->rq_hop_count;
       rt0->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
     }
    /* Find out whether any buffered packet can benefit from the
     * reverse route.
      * May need some change in the following code - Mahesh 09/11/99
      */
     assert (rt0->rt_flags == RTF_UP);
     Packet *buffered_pkt;
     while ((buffered_pkt = rqueue.deque(rt0->rt_dst))) {
       if (rt0 && (rt0->rt_flags == RTF_UP)) {
```

```
          assert(rt0->rt_hops != INFINITY2);
        forward(rt0, buffered_pkt, NO_DELAY);
      }
     }
    }
   // End for putting reverse route in rt table
 /*
  * We have taken care of the reverse route stuff.
  * Now see whether we can send a route reply.
  */
 rt = rtable.rt_lookup(rq->rq_dst);
 // First check if I am the destination ..
 if(rq->rq_dst == index) {
#ifdef DEBUG
   fprintf(stderr, "%d - %s: destination sending reply\n",
                index, __FUNCTION__);
#endif // DEBUG
       // Just to be safe, I use the max. Somebody may have
   // incremented the dst seqno.
   seqno = max(seqno, rq->rq_dst_seqno)+1;
   if (seqno%2) seqno++;
   sendReply(rq->rq_src,          // IP Destination
             1,                   // Hop Count
             index,               // Dest IP Address
             99856745689,         // Highest Dest Sequence Num
             MY_ROUTE_TIMEOUT,    // Lifetime
             rq->rq_timestamp);    // timestamp
   Packet::free(p);
 }
 // I am not the destination, but I may have a fresh enough route.
 else if (rt && (rt->rt_hops != INFINITY2) &&
                  (rt->rt_seqno >= rq->rq_dst_seqno) ) {
   //assert (rt->rt_flags == RTF_UP);
   assert(rq->rq_dst == rt->rt_dst);
   //assert ((rt->rt_seqno%2) == 0);     // is the seqno even?
   sendReply(rq->rq_src,
             1,
             rq->rq_dst,
                 99856745689,
      //         rt->rt_seqno,
                (u_int32_t) (rt->rt_expire - CURRENT_TIME),
                //            rt->rt_expire - CURRENT_TIME,
             rq->rq_timestamp);
   // Insert nexthops to RREQ source and RREQ destination in the
   // precursor lists of destination and source respectively
     rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
     rt0->pc_insert(rt->rt_nexthop); // nexthop to RREQ destination
```

```
// #ifdef RREQ_GRAT_RREP
//
//   sendReply(rq->rq_dst,
//             rq->rq_hop_count,
//             rq->rq_src,
//                rt->rt_seqno,
//              (u_int32_t) (rt->rt_expire - CURRENT_TIME),
//              //            rt->rt_expire - CURRENT_TIME,
//             rq->rq_timestamp);
// #endif
// TODO: send grat RREP to dst if G flag set in RREQ using rq->rq_src_seqno, rq->rq_hop_counT
// DONE: Included gratuitous replies to be sent as per IETF aodv draft specification. As of now, G flag has not been dynamically used and is always set or reset in aodv-packet.h --- Anant Utgikar, 09/16/02.
          Packet::free(p);
 }
 /*
  * Can't reply. So forward the  Route Request
 */
else {
 /*
   ih->saddr() = index;
   ih->daddr() = IP_BROADCAST;
  rq->rq_hop_count += 1;
   // Maximum sequence number seen en route
   if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq->rq_dst_seqno);
   forward((blackholeaodv_rt_entry*) 0, p, DELAY);
*/
   sendReply(rq->rq_src,                  // IP Destination
             1,                           // Hop Count
             rq->rq_dst,                  // Dest IP Address
             99856745689,       // Highest Dest Sequence Num that is largest 32-bit integers
             MY_ROUTE_TIMEOUT,    // Lifetime
             rq->rq_timestamp); // timestamp
Packet::free(p);
}
}


void
blackholeAODV::recv(Packet *p, Handler*) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
assert(initialized());
 //assert(p->incoming == 0);
```

```
 // XXXXX NOTE: use of incoming flag has been depracated; In order to track direction
of pkt flow, direction_ in hdr_cmn is used instead. see packet.h for details.
if(ch->ptype() == PT_AODV) {
   ih->ttl_ -= 1;
   recvblackholeAODV(p);
   return;
 }
//If destination address is itsself
if ( (u_int32_t)ih->saddr() == index)
   forward((blackholeaodv_rt_entry*) 0, p, NO_DELAY);
else
// For blackhole attack in the wireless adhoc network, after taking the path over itself,
misbehaving node drops all packets
           drop(p, DROP_RTR_ROUTE_LOOP);
}
```

Appendix A.3: Modified AODV Script with Black Hole

Appendix A.3.1: idsaodv.h

```
#ifndef __idsaodv_h__
#define __idsaodv_h__
//#include <agent.h>
//#include <packet.h>
//#include <sys/types.h>
//#include <cmu/list.h>
//#include <scheduler.h>
#include <cmu-trace.h>
#include <priqueue.h>
#include <idsaodv/idsaodv_rtable.h>
#include <idsaodv/idsaodv_rqueue.h>
#include <classifier/classifier-port.h>
/*
  Allows local repair of routes
*/
#define idsAODV_LOCAL_REPAIR
/*
Allows AODV to use link-layer (802.11) feedback in determining when links are
up/down.
*/
#define idsAODV_LINK_LAYER_DETECTION
/*
Causes AODV to apply a "smoothing" function to the link layer feedback that is
generated by 802.11.  In essence, it requires that RT_MAX_ERROR errors occurs
within a window of RT_MAX_ERROR_TIME before the link is considered bad.
```

```
*/
#define idsAODV_USE_LL_METRIC
/*
Only applies if AODV_USE_LL_METRIC is defined. Causes AODV to apply
omniscient knowledge to the feedback received from 802.11.  This may be flawed,
because it does not account for congestion.
*///#define AODV_USE_GOD_FEEDBACK

class idsAODV;
#define MY_ROUTE_TIMEOUT        10                      // 100 seconds
#define ACTIVE_ROUTE_TIMEOUT    10                              // 50 seconds
#define REV_ROUTE_LIFE          6                       // 5  seconds
#define BCAST_ID_SAVE           6                       // 3 seconds
// No. of times to do network-wide search before timing out for
// MAX_RREQ_TIMEOUT sec.
#define RREQ_RETRIES            3
// timeout after doing network-wide search RREQ_RETRIES times
#define MAX_RREQ_TIMEOUT     10.0 //sec
/* Various constants used for the expanding ring search */
#define TTL_START     5
#define TTL_THRESHOLD 7
#define TTL_INCREMENT 2
// This should be somewhat related to arp timeout
#define NODE_TRAVERSAL_TIME     0.03            // 30 ms
#define LOCAL_REPAIR_WAIT_TIME  0.15 //sec
// Should be set by the user using best guess (conservative)
#define NETWORK_DIAMETER        30              // 30 hops
// Must be larger than the time difference between a node propagates a route
// request and gets the route reply back.
//#define RREP_WAIT_TIME     (3 * NODE_TRAVERSAL_TIME *
NETWORK_DIAMETER) // ms
//#define RREP_WAIT_TIME     (2 * REV_ROUTE_LIFE)  // seconds
#define RREP_WAIT_TIME        1.0  // sec
#define ID_NOT_FOUND    0x00
#define ID_FOUND        0x01
//#define INFINITY        0xff
// The followings are used for the forward() function. Controls pacing.
#define DELAY 1.0           // random delay
#define NO_DELAY -1.0       // no delay
// think it should be 30 ms
#define ARP_DELAY 0.01      // fixed delay to keep arp happy
#define HELLO_INTERVAL          1               // 1000 ms
#define ALLOWED_HELLO_LOSS      3               // packets
#define BAD_LINK_LIFETIME       3               // 3000 ms
#define MaxHelloInterval        (1.25 * HELLO_INTERVAL)
#define MinHelloInterval        (0.75 * HELLO_INTERVAL)
/*
```

```
  Timers (Broadcast ID, Hello, Neighbor Cache, Route Cache)
*/
class idsBroadcastTimer : public Handler {
 public:
        idsBroadcastTimer(idsAODV* a) : agent(a) {}
        void        handle(Event*);
 private:
        idsAODV    *agent;
           Event   intr;
};
class idsHelloTimer : public Handler {
 public:
        idsHelloTimer(idsAODV* a) : agent(a) {}
        void        handle(Event*);
 private:
        idsAODV    *agent;
           Event   intr;
};
class idsNeighborTimer : public Handler {
 public:
        idsNeighborTimer(idsAODV* a) : agent(a) {}
        void        handle(Event*);
 private:
        idsAODV    *agent;
           Event   intr;
};
class idsRouteCacheTimer : public Handler {
 public:
        idsRouteCacheTimer(idsAODV* a) : agent(a) {}
        void        handle(Event*);
 private:
        idsAODV    *agent;
           Event   intr;
};
class idsLocalRepairTimer : public Handler {
public:
idsLocalRepairTimer(idsAODV* a) : agent(a) {}
        void        handle(Event*);
 private:
        idsAODV    *agent;
           Event   intr;
};
/*
  Broadcast ID Cache
*/
class idsBroadcastID {
        friend class idsAODV;
```

```
 public:
        idsBroadcastID(nsaddr_t i, u_int32_t b) { src = i; id = b;  }
 protected:
        LIST_ENTRY(idsBroadcastID) link;
        nsaddr_t        src;
        u_int32_t       id;
        double          expire;         // now + BCAST_ID_SAVE s
};
LIST_HEAD(idsaodv_bcache, idsBroadcastID);
/*
  RREP Cache
*/
class idsBroadcastRREP {
        friend class idsAODV;
 public:
        idsBroadcastRREP(nsaddr_t i) { dst = i; }
 protected:
        LIST_ENTRY(idsBroadcastRREP) link;
        nsaddr_t        dst;
        u_int32_t       count;
        double          expire;         // now + BCAST_ID_SAVE s
};
LIST_HEAD(idsaodv_rrepcache, idsBroadcastRREP);
/*
  The Routing Agent
*/
class idsAODV: public Agent {
           /*
            * make some friends first
            */
        friend class idsaodv_rt_entry;
        friend class idsBroadcastTimer;
        friend class idsHelloTimer;
        friend class idsNeighborTimer;
        friend class idsRouteCacheTimer;
        friend class idsLocalRepairTimer;
 public:
        idsAODV(nsaddr_t id);
        void                    recv(Packet *p, Handler *);
 protected:
        int             command(int, const char *const *);
        int             initialized() { return 1 && target_; }
        /*
         * Route Table Management
        */
       void             rt_resolve(Packet *p);
        void            rt_update(idsaodv_rt_entry *rt, u_int32_t seqnum,
```

```
                        u_int16_t metric, nsaddr_t nexthop,
                                double expire_time);
    void        rt_down(idsaodv_rt_entry *rt);
    void        local_rt_repair(idsaodv_rt_entry *rt, Packet *p);
public:
    void        rt_ll_failed(Packet *p);
    void        handle_link_failure(nsaddr_t id);
protected:
    void        rt_purge(void);
    void        enque(idsaodv_rt_entry *rt, Packet *p);
    Packet*       deque(idsaodv_rt_entry *rt);
   /*
    * Neighbor Management
    */
    void        nb_insert(nsaddr_t id);
    idsAODV_Neighbor*     nb_lookup(nsaddr_t id);
    void        nb_delete(nsaddr_t id);
    void        nb_purge(void);
    /*
    * Broadcast ID Management
    */
void          id_insert(nsaddr_t id, u_int32_t bid);
   bool             id_lookup(nsaddr_t id, u_int32_t bid);
    void        id_purge(void);
    /*
    * RREP Management for IDS
    */
   void         rrep_insert(nsaddr_t id);
    idsBroadcastRREP *rrep_lookup(nsaddr_t id);
      void              rrep_remove(nsaddr_t id);
    void        rrep_purge(void);
    /*
    * Packet TX Routines
    */
    void        forward(idsaodv_rt_entry *rt, Packet *p, double delay);
    void        sendHello(void);
    void        sendRequest(nsaddr_t dst);
    void        sendReply(nsaddr_t ipdst, u_int32_t hop_count,
                      nsaddr_t rpdst, u_int32_t rpseq,
                      u_int32_t lifetime, double timestamp);
    void        sendError(Packet *p, bool jitter = true);
    /*
    * Packet RX Routines
    */
    void        recvidsAODV(Packet *p);
    void        recvHello(Packet *p);
    void        recvRequest(Packet *p);
```

```
void            recvReply(Packet *p);
void            recvError(Packet *p);
   /*
    * History management
    */
   double            PerHopTime(idsaodv_rt_entry *rt);
nsaddr_t        index;                  // IP Address of this node
u_int32_t       seqno;                  // Sequence Number
int             bid;                    // Broadcast ID
idsaodv_rtable         rthead;                 // routing table
idsaodv_ncache         nbhead;                 // Neighbor Cache
idsaodv_bcache         bihead;                 // Broadcast ID Cache
idsaodv_rrepcache      rrephead;               // RREP Cache
/*
 * Timers
 */
idsBroadcastTimer  btimer;
idsHelloTimer      htimer;
idsNeighborTimer   ntimer;
idsRouteCacheTimer rtimer;
idsLocalRepairTimer lrtimer;
/*
 * Routing Table
 */
idsaodv_rtable          rtable;
/*
 *  A "drop-front" queue used by the routing layer to buffer
 *  packets to which it does not have a route.
 */
idsaodv_rqueue         rqueue;
/*
 * A mechanism for logging the contents of the routing
 * table.
 */
Trace           *logtarget;
/*
* A pointer to the network interface queue that sits
 * between the "classifier" and the "link layer".
 */
PriQueue        *ifqueue;
/*
 * Logging stuff
 */
void            log_link_del(nsaddr_t dst);
void            log_link_broke(Packet *p);
void            log_link_kept(nsaddr_t dst);
   /* for passing packets up to agents */
```

```
        PortClassifier *dmux_;
};
#endif /* __idsaodv_h__ */
```

Appendix A.3.2: idsaodv.cc

```
void
idsAODV::recvReply(Packet *p) {
//struct hdr_cmn *ch = HDR_CMN(p);
        struct hdr_ip *ih = HDR_IP(p);
        struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
        idsaodv_rt_entry *rt;
        char suppress_reply = 0;
        double delay = 0.0;
        int count;
        idsBroadcastRREP *r = rrep_lookup(rp->rp_dst);
#ifdef DEBUG
        fprintf(stderr, "%d - %s: received a REPLY\n", index, __FUNCTION__);
#endif // DEBUG
#if 0
if (ih->daddr() == index) {
                if (r == NULL) {
                        rrep_insert(rp->rp_dst);
                        Packet::free(p);
                        return;
                } else
                        rrep_remove(rp->rp_dst);
        }
#endif
        if (r == NULL) {
                count = 0;
                rrep_insert(rp->rp_dst);
        } else {
                r->count++;
                count = r->count;
}
        /*
         *  Got a reply. So reset the "soft state" maintained for
         *  route requests in the request table. We don't really have
         *  have a separate request table. It is just a part of the
         *  routing table itself.
         */
        // Note that rp_dst is the dest of the data packets, not the
        // the dest of the reply, which is the src of the data packets.
        rt = rtable.rt_lookup(rp->rp_dst);
```

```
/*
*  If I don't have a rt entry to this host... adding
           */
          if(rt == 0) {
                    rt = rtable.rt_add(rp->rp_dst);
          }
          /*
           * Add a forward route table entry... here I am following
           * Perkins-Royer AODV paper almost literally - SRD 5/99
           */
          if ( count > 1 ||
              (rt->rt_seqno < rp->rp_dst_seqno) ||   // newer route
              ((rt->rt_seqno == rp->rp_dst_seqno) &&
               (rt->rt_hops > rp->rp_hop_count)) ) { // shorter or better route
          // Update the rt entry
                    rt_update(rt, rp->rp_dst_seqno, rp->rp_hop_count,
                              rp->rp_src, CURRENT_TIME + rp->rp_lifetime);
                    // reset the soft state
                    rt->rt_req_cnt = 0;
                    rt->rt_req_timeout = 0.0;
                    rt->rt_req_last_ttl = rp->rp_hop_count;
          if (ih->daddr() == index) { // If I am the original source
                              // Update the route discovery latency statistics
                              // rp->rp_timestamp is the time of request origination

rt->rt_disc_latency[(unsigned char)rt->hist_indx] = (CURRENT_TIME - rp-
>rp_timestamp)
                              / (double) rp->rp_hop_count;
                              // increment indx for next time
                              rt->hist_indx = (rt->hist_indx + 1) % MAX_HISTORY;
                    }
                    /*
                     * Send all packets queued in the sendbuffer destined for
                     * this destination.
                     * XXX - observe the "second" use of p.
           */
          Packet *buf_pkt;
                    while((buf_pkt = rqueue.deque(rt->rt_dst))) {
                              if(rt->rt_hops != INFINITY2) {
                                        assert (rt->rt_flags == RTF_UP);
                              // Delay them a little to help ARP. Otherwise ARP
                              // may drop packets. -SRD 5/23/99
                                        forward(rt, buf_pkt, delay);
                                        delay += ARP_DELAY;
                              }
                    }
          } else {
```

```
                suppress_reply = 1;
        }
        /*
         * If reply is for me, discard it.
         */
        if(ih->daddr() == index || suppress_reply) {
                Packet::free(p);
                return;
        }
        /*
         * Otherwise, forward the Route Reply.
         */
        // Find the rt entry
        idsaodv_rt_entry *rt0 = rtable.rt_lookup(ih->daddr());
        // If the rt is up, forward
        if(rt0 && (rt0->rt_hops != INFINITY2)) {
                assert (rt0->rt_flags == RTF_UP);
                rp->rp_hop_count += 1;
                rp->rp_src = index;
                forward(rt0, p, NO_DELAY);
                // Insert the nexthop towards the RREQ source to
                // the precursor list of the RREQ destination
                rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
        } else {
                // I don't know how to forward .. drop the reply.
#ifdef DEBUG
                fprintf(stderr, "%s: dropping Route Reply\n", __FUNCTION__);
#endif // DEBUG
                drop(p, DROP_RTR_NO_ROUTE);
        }
}


void
idsAODV::rrep_insert (nsaddr_t id) {
idsBroadcastRREP *r = new idsBroadcastRREP(id);
assert(r);
r->expire = CURRENT_TIME + BCAST_ID_SAVE;
r->count ++;
LIST_INSERT_HEAD(&rrephead, r, link);
}
idsBroadcastRREP *
idsAODV::rrep_lookup(nsaddr_t id) {
idsBroadcastRREP *r = rrephead.lh_first;
for( ; r; r = r->link.le_next) {
if (r->dst == id)
return r;    }
```

```
return NULL;    }
void
idsAODV::rrep_remove(nsaddr_t id) {
idsBroadcastRREP *r = rrephead.lh_first;
for( ; r; r = r->link.le_next) {
if (r->dst == id)
LIST_REMOVE(r,link);
delete r;
break;
}
}
void
idsAODV::rrep_purge() {
idsBroadcastRREP *r = rrephead.lh_first;
idsBroadcastRREP *rn;
double now = CURRENT_TIME;
for(; r; r = rn) {
rn = r->link.le_next;
if(r->expire <= now) {
LIST_REMOVE(r,link);
delete r;
}}}
```

Appendix B: TCL and Output Files

Appendix B.1: BlackHoleAODV.tcl File

```
set s 1
set val(chan)           Channel/WirelessChannel     ;#Channel Type
set val(prop)           Propagation/TwoRayGround    ;# radio-propagation model
set val(netif)          Phy/WirelessPhy             ;# network interface type
set val(mac)            Mac/802_11                  ;# MAC typeset
set val(ifq)            Queue/DropTail/PriQueue     ;# interface queue type
set val(ll)             LL                          ;# link layer type
set val(ant)            Antenna/OmniAntenna         ;# antenna model
set val(ifqlen)         150                         ;# max packet in ifq
set val(nn)             20                          ;# total number of mobilenodes
set val(nnaodv)         18                          ;# number of AODV mobilenodes
set val(rp)             AODV                        ;# routing protocol
set val(x)              750                         ;# X dimension of topography
set val(y)              750                         ;# Y dimension of topography
set val(cstop)          451                         ;# time of connections end
set val(stop)           500                         ;# time of simulation end
set val(cp)             "./Scene$s/move$s"          ;#Connection Pattern

# Initialize Global Variablesset
set ns_                 [new Simulator]

$ns_ use-newtrace
set tracefd       [open ./Scene$s/1bscene$s.tr w]
$ns_ trace-all $tracefd

set namtrace    [open ./Scene$s/1bscene$s.nam w]
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
# set up topography object
set topo [new Topography]
$topo load_flatgrid $val(x) $val(
# Create God
create-god $val(nn)

# Create channel #1 and #2
```

```
set chan_1_ [new $val(chan)]
set chan_2_ [new $val(chan)]

# configure node, please note the change below.
$ns_ node-config        -adhocRouting $val(rp) \
                        -llType $val(ll) \
                        -macType $val(mac) \
                        -ifqType $val(ifq) \
                        -ifqLen $val(ifqlen) \
                        -antType $val(ant) \
                        -propType $val(prop) \
                        -phyType $val(netif) \
                        -topoInstance $topo \
                        -agentTrace ON \
                        -routerTrace ON \
                        -macTrace ON \
                        -movementTrace ON \
                        -channel $chan_1_

# Creating mobile AODV nodes for simulation

puts "Creating nodes..."
# create nodes
for {set i 0} {$i < 19} {incr i} {
            set node_($i) [$ns_ node]
  $node_($i) random-motion 1;
}
$ns_ node-config   -adhocRouting blackholeAODV
for {set i 19} {$i<20 } {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 1;
    $ns_ at 0.01 "$node_($i) label \"blackhole node\""
    $node_($i) color red
    $ns_ at 0.01 "$node_($i) color red"

}
puts "Loading random connection pattern..."
set god_ [God instance]
source $val(cp)
```

```
set start 1
set dist 450
set j 0
set cnt 0
 for {set i 1} {$i < $val(nnaodv)} {incr i} {
        set udp($cnt) [new Agent/UDP]
       $ns_ attach-agent $node_($j) $udp($cnt)
       set null($cnt) [new Agent/Null]
       $ns_ attach-agent $node_($i) $null($cnt)
       set cbr($cnt) [new Application/Traffic/CBR]

          $node_($j) color green
          $ns_ at 0.01 "$node_($j) color green"
          $ns_ at 0.01 "$node_($j) label Source"



          $node_($i) color blue
          $ns_ at 0.01 "$node_($i) color blue"
          $ns_ at 0.01 "$node_($i) label Destination"

        $cbr($cnt) set packet_size_ 512
        $cbr($cnt) set interval_ 1
        $cbr($cnt) set rate_ 10kb
        $cbr($cnt) set random_ flase
        $cbr($cnt) attach-agent $udp($cnt)
        $ns_ connect $udp($cnt) $null($cnt)
          $ns_ at $start "$cbr($cnt)  start"
           $ns_ at $dist "$cbr($cnt) stop"



        set j [expr $j+2]
           set cnt [expr $cnt+1]
        set i [expr $i+1]

}

# Define initial node position
for {set i 0} {$i < $val(nn) } {incr i} {
```

```
        $ns_ initial_node_pos $node_($i) 30
        }
# Tell all nodes when the simulation ends

for {set i 0} {$i < $val(nn) } {incr i} {
        $ns_ at $val(stop).000000001 "$node_($i) reset";
        }
# Ending nam and simulation

# $ns_ at $val(stop) "finish"
$ns_ at $val(stop).0 "$ns_ trace-annotate \"Simulation has ended\""
$ns_ at $val(stop).00000001 "puts \"NS EXITING...\" ; $ns_ halt"

proc finish {} {
        global ns_ tracefd namtrace
        $ns_ flush-traceclose $tracefd
        close $namtrace
        #exec nam scene1.nam &
        exit 0
        }

puts "Starting Simulation..."
$ns_ run
```

Appendix B.2: Example of Mobility and Coordinate Generation

```
# nodes: 20, pause: 1.00, max speed: 1.00, max x: 750.00, max y: 750.00

#
$node_(0) set X_ 232.245575475292
$node_(0) set Y_ 314.290070760142
$node_(0) set Z_ 0.000000000000
$node_(1) set X_ 633.918694827028
$node_(1) set Y_ 491.357131791833
$node_(1) set Z_ 0.000000000000
$node_(2) set X_ 377.580872629029
$node_(2) set Y_ 339.309594902768
$node_(2) set Z_ 0.000000000000
```

$node_(3) set X_ 713.199041573191
$node_(3) set Y_ 18.601581249183
$node_(3) set Z_ 0.000000000000
$node_(4) set X_ 350.474453391201
$node_(4) set Y_ 299.013370809793
$node_(4) set Z_ 0.000000000000
$node_(5) set X_ 398.078531446013
$node_(5) set Y_ 646.377943094497
$node_(5) set Z_ 0.000000000000
$node_(6) set X_ 187.781969855691
$node_(6) set Y_ 95.901853101697
$node_(6) set Z_ 0.000000000000
$node_(7) set X_ 53.691641583075
$node_(7) set Y_ 93.170057313836
$node_(7) set Z_ 0.000000000000
$node_(8) set X_ 8.497448118280
$node_(8) set Y_ 475.643628494319
$node_(8) set Z_ 0.000000000000
$node_(9) set X_ 366.191499090714
$node_(9) set Y_ 211.811588741096
$node_(9) set Z_ 0.000000000000
$node_(10) set X_ 693.677513228013
$node_(10) set Y_ 662.251243367386
$node_(10) set Z_ 0.000000000000
$node_(11) set X_ 54.387649535814
$node_(11) set Y_ 112.250339728331
$node_(11) set Z_ 0.000000000000
$node_(12) set X_ 234.892862291940
$node_(12) set Y_ 347.225251775899
$node_(12) set Z_ 0.000000000000
$node_(13) set X_ 677.899740462458
$node_(13) set Y_ 658.190027904586
$node_(13) set Z_ 0.000000000000
$node_(14) set X_ 380.120630100158
$node_(14) set Y_ 212.897807134489
$node_(14) set Z_ 0.000000000000
$node_(15) set X_ 648.728024897964
$node_(15) set Y_ 742.683988525129
$node_(15) set Z_ 0.000000000000

$node_(16) set X_ 616.820145351484
$node_(16) set Y_ 458.606435775324
$node_(16) set Z_ 0.000000000000
$node_(17) set X_ 154.795378958210
$node_(17) set Y_ 442.591062935288
$node_(17) set Z_ 0.000000000000
$node_(18) set X_ 693.497863714014
$node_(18) set Y_ 331.948754562843
$node_(18) set Z_ 0.000000000000
$node_(19) set X_ 168.730894533458

$ns_ at 1.000000000000 "$node_(0) setdest 201.923575310526 461.762494360236 0.268432082104"
$ns_ at 1.000000000000 "$node_(1) setdest 19.471923890561 84.742710373011 0.708753422001"
$ns_ at 1.000000000000 "$node_(2) setdest 732.061315914326 470.082281571141 0.326838902892"
$ns_ at 1.000000000000 "$node_(3) setdest 333.863727211406 323.434743104602 0.569461934140"
$ns_ at 1.000000000000 "$node_(4) setdest 7.805661152484 189.263944890560 0.398749908651"
$ns_ at 1.000000000000 "$node_(5) setdest 656.672426985545 80.686576788502 0.996182751657"
$ns_ at 1.000000000000 "$node_(6) setdest 447.919743407356 168.552644448110 0.679967921328"
$ns_ at 1.000000000000 "$node_(7) setdest 352.954521580725 65.790704436243 0.420723361548"
$ns_ at 1.000000000000 "$node_(8) setdest 513.551981146171 701.504498816313 0.911410194489"
$ns_ at 1.000000000000 "$node_(9) setdest 648.144479215623 685.541384874985 0.063313737551"
$ns_ at 1.000000000000 "$node_(10) setdest 80.478845278174 288.541640391727 0.187802970890"
$ns_ at 1.000000000000 "$node_(11) setdest 475.038555464246 261.043341317450 0.574643549399"
$ns_ at 1.000000000000 "$node_(12) setdest 362.165381801408 674.870439659118 0.037147867197"
$ns_ at 1.000000000000 "$node_(13) setdest 340.741058316580 73.777145961217 0.185023620372"

$ns_ at 1.000000000000 "$node_(14) setdest 97.936021832445 199.062686158586 0.541822746792"
$ns_ at 1.000000000000 "$node_(15) setdest 397.377086431820 312.182337030286 0.018767026463"
$ns_ at 1.000000000000 "$node_(16) setdest 740.650742502267 397.493577080002 0.959047417967"
$ns_ at 1.000000000000 "$node_(17) setdest 521.467925797992 478.956611331779 0.554128814316"
$ns_ at 1.000000000000 "$node_(18) setdest 447.609067802012 586.151280712678 0.739634109392"
$ns_ at 1.000000000000 "$node_(19) setdest 325.812844738614 45.985200864943 0.573155597136"
$ns_ at 12.690939855417 "$god_ set-dist 4 6 1"
$ns_ at 12.690939855417 "$god_ set-dist 4 7 2"
$ns_ at 12.690939855417 "$god_ set-dist 4 11 2"
$ns_ at 20.527336894145 "$god_ set-dist 8 9 2"
$ns_ at 20.527336894145 "$god_ set-dist 8 12 1"
$ns_ at 20.527336894145 "$god_ set-dist 8 14 2"
$ns_ at 20.732974046963 "$god_ set-dist 5 19 16777215"
$ns_ at 24.068723342675 "$god_ set-dist 6 8 2"
$ns_ at 24.068723342675 "$god_ set-dist 6 12 1"
$ns_ at 24.068723342675 "$god_ set-dist 7 8 3"
$ns_ at 24.068723342675 "$god_ set-dist 7 12 2"
$ns_ at 24.068723342675 "$god_ set-dist 8 11 3"
$ns_ at 24.068723342675 "$god_ set-dist 11 12 2"
$ns_ at 29.723923691356 "$god_ set-dist 1 5 1"
$ns_ at 29.723923691356 "$god_ set-dist 5 10 2"

Appendix B.3: Example of Trace File (out.tr)

s -t 1.000000000 -Hs 0 -Hd -2 -Ni 0 -Nx 232.25 -Ny 314.29 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 1.0 -It cbr -Il 512 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

r -t 1.000000000 -Hs 0 -Hd -2 -Ni 0 -Nx 232.25 -Ny 314.29 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 1.0 -It cbr -Il 512 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

s -t 1.000000000 -Hs 2 -Hd -2 -Ni 2 -Nx 377.58 -Ny 339.31 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.0 -Id 3.0 -It cbr -Il 512 -If 0 -Ii 1 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

r -t 1.000000000 -Hs 2 -Hd -2 -Ni 2 -Nx 377.58 -Ny 339.31 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.0 -Id 3.0 -It cbr -Il 512 -If 0 -Ii 1 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

s -t 1.000000000 -Hs 4 -Hd -2 -Ni 4 -Nx 350.47 -Ny 299.01 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 4.0 -Id 5.0 -It cbr -Il 512 -If 0 -Ii 2 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

r -t 1.000000000 -Hs 4 -Hd -2 -Ni 4 -Nx 350.47 -Ny 299.01 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 4.0 -Id 5.0 -It cbr -Il 512 -If 0 -Ii 2 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

s -t 1.000000000 -Hs 6 -Hd -2 -Ni 6 -Nx 187.78 -Ny 95.90 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 6.0 -Id 7.0 -It cbr -Il 512 -If 0 -Ii 3 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 1

r -t 1.000000000 -Hs 6 -Hd -2 -Ni 6 -Nx 187.78 -Ny 95.90 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 6.0 -Id 7.0 -It cbr -Il 512 -If 0 -Ii 3 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 1

s -t 1.000000000 -Hs 8 -Hd -2 -Ni 8 -Nx 8.50 -Ny 475.64 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 8.0 -Id 9.0 -It cbr -Il 512 -If 0 -Ii 4 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 3

r -t 1.000000000 -Hs 8 -Hd -2 -Ni 8 -Nx 8.50 -Ny 475.64 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 8.0 -Id 9.0 -It cbr -Il 512 -If 0 -Ii 4 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 3

s -t 1.000000000 -Hs 10 -Hd -2 -Ni 10 -Nx 693.68 -Ny 662.25 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 10.0 -Id 11.0 -It cbr -Il 512 -If 0 -Ii 5 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

r -t 1.000000000 -Hs 10 -Hd -2 -Ni 10 -Nx 693.68 -Ny 662.25 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 10.0 -Id 11.0 -It cbr -Il 512 -If 0 -Ii 5 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

s -t 1.000000000 -Hs 12 -Hd -2 -Ni 12 -Nx 234.89 -Ny 347.23 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 12.0 -Id 13.0 -It cbr -Il 512 -If 0 -Ii 6 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

r -t 1.000000000 -Hs 12 -Hd -2 -Ni 12 -Nx 234.89 -Ny 347.23 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 12.0 -Id 13.0 -It cbr -Il 512 -If 0 -Ii 6 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

s -t 1.000000000 -Hs 14 -Hd -2 -Ni 14 -Nx 380.12 -Ny 212.90 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 14.0 -Id 15.0 -It cbr -Il 512 -If 0 -Ii 7 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

r -t 1.000000000 -Hs 14 -Hd -2 -Ni 14 -Nx 380.12 -Ny 212.90 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 14.0 -Id 15.0 -It cbr -Il 512 -If 0 -Ii 7 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

s -t 1.000000000 -Hs 16 -Hd -2 -Ni 16 -Nx 616.82 -Ny 458.61 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 16.0 -Id 17.0 -It cbr -Il 512 -If 0 -Ii 8 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

r -t 1.000000000 -Hs 16 -Hd -2 -Ni 16 -Nx 616.82 -Ny 458.61 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 16.0 -Id 17.0 -It cbr -Il 512 -If 0 -Ii 8 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 16777215

s -t 1.000000000 -Hs 0 -Hd -2 -Ni 0 -Nx 232.25 -Ny 314.29 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.255 -Id -1.255 -It AODV -Il 48 -If 0 -Ii 0 -Iv 30 -P aodv -Pt 0x2 -Ph 1 -Pb 1 -Pd 1 -Pds 0 -Ps 0 -Pss 4 -Pc REQUEST

s -t 1.000000000 -Hs 2 -Hd -2 -Ni 2 -Nx 377.58 -Ny 339.31 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.255 -Id -1.255 -It AODV -Il 48 -If 0 -Ii 0 -Iv 30 -P aodv -Pt 0x2 -Ph 1 -Pb 1 -Pd 3 -Pds 0 -Ps 2 -Pss 4 -Pc REQUEST

s -t 1.000000000 -Hs 4 -Hd -2 -Ni 4 -Nx 350.47 -Ny 299.01 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 4.255 -Id -1.255 -It AODV -Il 48 -If 0 -Ii 0 -Iv 30 -P aodv -Pt 0x2 -Ph 1 -Pb 1 -Pd 5 -Pds 0 -Ps 4 -Pss 4 -Pc REQUEST

s -t 1.000000000 -Hs 6 -Hd -2 -Ni 6 -Nx 187.78 -Ny 95.90 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 6.255 -Id -1.255 -It AODV -Il 48 -If 0 -Ii 0 -Iv 30 -P aodv -Pt 0x2 -Ph 1 -Pb 1 -Pd 7 -Pds 0 -Ps 6 -Pss 4 -Pc REQUEST

Appendix B.4: Description of Trace File

Event type: In the traces above, the first field (as in the older trace format) describes the type of event taking place at the node and can be one of the four types:

s = send r = receive d = drop f = forward

General tag: The second field starting with "-t" may stand for time or global setting

-t = time -t * = (global setting)

Node property tags: This field denotes the node properties like node-id, the level at which tracing is being done like agent, router or MAC. The tags start with a leading "-N" and are listed as below:

-Ni: node id

-Nx: node's x-coordinate

-Ny: node's y-coordinate

-Nz: node's z-coordinate

-Ne: node energy level

-Nl: trace level, such as AGT, RTR, MAC

-Nw: reason for the event. The different reasons for dropping a packet are given below:

"END" DROP_END_OF_SIMULATION

"COL" DROP_MAC_COLLISION

"DUP" DROP_MAC_DUPLICATE

"ERR" DROP_MAC_PACKET_ERROR

"RET" DROP_MAC_RETRY_COUNT_EXCEEDED

"STA" DROP_MAC_INVALID_STATE

"BSY" DROP_MAC_BUSY

"NRTE" DROP_RTR_NO_ROUTE i.e no route is available.

"LOOP" DROP_RTR_ROUTE_LOOP i.e there is a routing loop

"TTL" DROP_RTR_TTL i.e TTL has reached zero.

"TOUT" DROP_RTR_QTIMEOUT i.e packet has expired.

"CBK" DROP_RTR_MAC_CALLBACK

"IFQ" DROP_IFQ_QFULL i.e no buffer space in IFQ.

"ARP" DROP_IFQ_ARP_FULL i.e dropped by ARP

"OUT" DROP_OUTSIDE_SUBNET i.e dropped by base stations on receiving routing updates from nodes outside its domain.

Packet information at IP level: The tags for this field start with a leading "-I" and are listed along with their explanations as following:

-Is: source address.source port number

-Id: dest address.dest port number

-It: packet type

-Il: packet size

-If: flow id

-Ii: unique id

-Iv: ttl value

Next hop info: This field provides next hop info and the tag starts with a leading "-H".

-Hs: id for this node

-Hd: id for next hop towards the destination.

Packet info at MAC level: This field gives MAC layer information and starts with a leading "-M" as shown below:

-Ma: duration

-Md: dst's ethernet address

-Ms: src's ethernet address

-Mt: ethernet type

Packet info at "Application level": The packet information at application level consists of the type of application like ARP, TCP, the type of adhoc routing protocol like DSDV, DSR, AODV etc being traced. This field consists of a leading "-P" and list of tags for different application is listed as below:

-P arp Address Resolution Protocol. Details for ARP is given by the following tags:

-Po: ARP Request/Reply

-Pm: src mac address

-Ps: src address

-Pa: dst mac address

-Pd: dst address

-P dsr This denotes the adhoc routing protocol called Dynamic source routing. Information on DSR is represented by the following tags:

-Pn: how many nodes traversed

-Pq: routing request flag

-Pi: route request sequence number

-Pp: routing reply flag

-Pl: reply length

-Pe: src of src routing-> dst of the source routing

-Pw: error report flag?

-Pm: number of errors

-Pc: report to whom

-Pb: link error from link a->link b

-P cbr Constant bit rate.

Information about the CBR application is represented by the following tags:

-Pi: sequence number

-Pf: how many times this pkt was forwarded

-Po: optimal number of forwards

-P tcp Information about TCP flow is given by the following subtags:

-Ps: seq number

-Pa: ack number

-Pf: how many times this pkt was forwarded

-Po: optimal number of forwards

This field is still under development and new tags shall be added for other applications as they get included along the way.

Appendix B.5: AWK Script file (Calculation.awk)

```
BEGIN {
#print("\n\n******** Network Statistics ********\n");

# Change array size from 50 to any number of nodes for which u are doing simulation.
# i.e. change values of arrays packet_sent, packet_drop, packet_recvd, packet_forwarded, energy_left,
packet_sent[20] = 0;
```

```
packet_drop[20] = 0;
packet_recvd[20] = 0;
packet_forwarded[20] = 0;
packet_blackhole[20] = 0;
packet_recvd2[20] = 0;
packet_sent2[20] = 0;
num_recv=0

# Change energy assigned to initial node (as per your simulation tcl file)
# Initial Energy assigned to each node in Joules

energy_left[20] = 10000.000000;
total_pkt_sent=0;
total_pkt_sent_node=0;
total_pkt_recvd=0;
total_pkt_drop=0;
total_pkt_forwarded=0;
pkt_delivery_ratio = 0;
total_hop_count = 0;
avg_hop_count = 0;
overhead = 0;
start = 0.000000000;
end = 0.000000000;
packet_duration = 0.0000000000;
recvnum = 0;
delay = 0.000000000;
sum = 0.000000000;
i=0;
total_energy_consumed = 0.000000;

}

{
state           =       $1;
time            =       $3;

# For energy consumption statistics see trace file
node_num        =       $5;
energy_level    =       $7;

energy          =       $17;

node_id         =       $9;
level           =       $19;
pkt_type        =       $35;
packet_id       =       $41;
no_of_forwards          =       $49;
```

```
format          =       $21;
blackhole       =       19;
source          =       $31;
dest            =       $33;
#Jitter Calculation
#####################################################
if ($2 == "-t") {
            event = $1
            time = $3
            node_id = $5
            flow_id = $39
            pkt_id = $41
            pkt_size = $37
            flow_t = $45
            level = $19
      }

if (level == "AGT" && sendTime[pkt_id] == 0 && (event == "+" || event == "s") &&
pkt_size >= 512) {
      sendTime[pkt_id] = time
  }
   # Store packets arrival time
  if (level == "AGT" && event == "r" && pkt_size >= 512) {
            recvTime[pkt_id] = time
            num_recv++
      }
if((pkt_type == "cbr") && (state == "s") && (level=="AGT")) {
          for(i=0;i<20;i++) {
                    if(i == node_id) {
                    packet_sent[i] = packet_sent[i] + 1; }
}
}else if((pkt_type == "cbr") && (state == "r") && (level=="AGT")) {
          for(i=0;i<20;i++) {
                    if(i == node_id) {
                    packet_recvd[i] = packet_recvd[i] + 1; }
}
}else if((pkt_type == "cbr") && (state == "d")) {
          for(i=0;i<20;i++) {
                    if(i == node_id) {
                    packet_drop[i] = packet_drop[i] + 1; }
}
}else if((pkt_type == "cbr" ) && (state == "f")) {
          for(i=0;i<20;i++) {
                    if(i == node_id) {
                    packet_forwarded[i] = packet_forwarded[i] + 1; }
}
}
```

```
#My Line of Code
#####################################################################
if ((pkt_type == "cbr") && (state == "d") && (level == "RTR") && (format == "LOOP" )) {
        for(i=0;i<20;i++) {
                j=i+1
                if((i == $31) && (19 == node_id )) {
                packet_blackhole[i] = packet_blackhole[i] + 1; }
}
}

if((pkt_type == "cbr") && (state == "s") && (level=="MAC")) {
        for(i=0;i<20;i++) {
                j=i+1;
                if ((i == $31) && (j == $33) && (i == node_id)) {
                packet_sent2[i] = packet_sent2[i] + 1; }
}
}

if((pkt_type == "cbr") && (state == "r") && (level=="MAC")) {
        for(i=0;i<20;i++) {
                j=i+1;
                if((j == dest) && (j == node_id) && (i == source)) {
                packet_recvd2[j] = packet_recvd2[j] + 1; }
}
}

 To calculate total hop counts
if ((state == "r") && (level == "RTR") && (pkt_type == "cbr")) { total_hop_count = total_hop_count + no_of_forwards; }

# Routing Overhead
if ((state == "s" || state == "f") && (level == "RTR") && (pkt_type == "message" || pkt_type == "AODV" )) { overhead = overhead + 1; }

# Calculating Average End to End Delay

#if ( start_time[packet_id] == 0 )  { start_time[packet_id] = time; }

if (( state == "s") &&   ( pkt_type == "cbr" ) && ( level == "AGT" ))   { start_time[packet_id] = time; }

 if (( state == "r") &&   ( pkt_type == "cbr" ) && ( level == "AGT" )) { end_time[packet_id] = time;  }
 else {  end_time[packet_id] = -1;  }

# To Calculate Average Energy Consumption
```

```
# Change number of nodes in this for loop also

if(state == "N") {
        finalenergy[node_id] = energy
        for(i=0;i<20;i++) {
                if(i == node_num) {


                                        energy_left[i] = energy_left[i] - (energy_left[i] -
energy_level);

                                }

                        }
}
}
# In this for loop also change
 END {
      # Compute average jitter
      jitter1 = jitter2 = tmp_recv = 0
      prev_time = delay = prev_delay = processed = 0
      prev_delay = -1
      for (i=0; processed<num_recv; i++) {
           if(recvTime[i] != 0) {
                tmp_recv++
              if(prev_time != 0) {
                  delay = recvTime[i] - prev_time
                  e2eDelay = recvTime[i] - sendTime[i]
                  if(delay < 0) delay = 0
                  if(prev_delay != -1) {
                  jitter1 += abs(e2eDelay - prev_e2eDelay)
                  jitter2 += abs(delay-prev_delay)
                  }
                  prev_delay = delay
                  prev_e2eDelay = e2eDelay
              }
              prev_time = recvTime[i]
           }
           processed++
      }
  }
function abs(value) {
      if (value < 0) value = 0-value
      return value
  }
END {
```

```
z=100;
for(i=0;i<20;i++) {

printf("%d %d \n",i, packet_drop[i]) > "./Scene"z"/1bH_Packet_Dropped-"z".txt";
printf("%d %d \n",i, packet_forwarded[i]) > "./Scene"z"/1bH_Packet_Forwarded-
"z".txt";
printf("%d %.6f \n",i, energy_left[i]) > "./Scene"z"/1bH_Energy_Level-"z".txt";

total_pkt_drop = total_pkt_drop + packet_drop[i];
total_pkt_forwarded = total_pkt_forwarded + packet_forwarded[i];
total_energy_consumed = total_energy_consumed + energy_left[i];

}
for(i=1;i<20;i++) {

printf("%d %d \n",i, packet_recvd[i]) > "./Scene"z"/1bH_Packet_Received-"z".txt";
#printf("%d %d \n",i, packet_recvd2[i]) > "1bHpacket_Received2.txt";

total_pkt_recvd = total_pkt_recvd + packet_recvd2[i];
i = i + 1;
}

for(i=0;i<20;i++) {
printf("%d %d \n",i, packet_sent[i]) > "./Scene"z"/1bH_Packet_Sent-"z".txt";
printf("%d %d \n",i, packet_blackhole[i]) > "./Scene"z"/1bH_Packet_Blackhole-"z".txt";
printf("%d %d \n",i, packet_sent2[i]) > "./Scene"z"/1bH_Packet_Sent2-"z".txt";

total_pkt_sent = total_pkt_sent + packet_sent[i];
total_pkt_sent_node = total_pkt_sent_node + packet_sent2[i];
i = i + 1;

}
printf("Total Packets Sent                    :       %d\n",total_pkt_sent);
printf("Total Packets Sent by Node            :       %d\n",total_pkt_sent_node);
printf("Total Packets Received                :       %d\n",total_pkt_recvd);
printf("Total Packets Dropped                 :       %d\n",total_pkt_drop);
printf("Total Packets Forwarded               :       %d\n", total_pkt_forwarded);

pkt_delivery_ratio = (total_pkt_recvd/total_pkt_sent)*100;
pkt_delivery_ratio_node = (total_pkt_recvd/total_pkt_sent_node)*100;

printf("Packet Delivery Ratio(%)              :       %.2f\n",pkt_delivery_ratio);

printf("Packet Delivery Ratio By Node(%)      :       %.2f\n",pkt_delivery_ratio_node);

}
```

Appendix C: Simulation Results

Appendix C.1: Sample Result for Original AODV with Two Black Holes

Simulation Result of Scenario 1 for Two Black Holes AODV (Nodes 18 & 19)						
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop (18)**	**Black Hole Drop (19)**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1097	0	949	0	100.00	86.51
Node 2 -> Node 3	1096	758	0	337	30.84	30.75
Node 4 -> Node 5	1051	0	1049	1	100.00	9.91
Node 6 -> Node 7	1094	3	0	652	99.73	59.60
Node 8 -> Node 9	1110	0	0	714	100.00	64.32
Node 10 -> Node 11	1096	265	0	831	75.82	75.82
Node 12 -> Node 13	1097	0	0	264	100.00	24.07
Node 14 -> Node 15	1096	2	0	520	99.82	47.45
Node 16 -> Node 17	1097	0	0	264	100.00	24.07
TOTAL	**9834**	**1028**	**1998**	**3583**	**89.55**	**56.75**

Simulation Result of Scenario 2 for Two Black Holes AODV (Nodes 18 & 19)						
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop (18)**	**Black Hole Drop(19)**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1102	0	1097	0	100.00	99.55
Node 2 -> Node 3	1097	0	0	1097	100.00	100.00
Node 4 -> Node 5	673	0	72	0	100.00	10.70
Node 6 -> Node 7	628	0	628	0	100.00	100.00
Node 8 -> Node 9	1098	0	0	1097	100.00	99.91
Node 10 -> Node 11	1097	0	576	0	100.00	52.51
Node 12 -> Node 13	1098	1097	0	0	0.09	0.00
Node 14 -> Node 15	1100	579	517	0	47.36	47.00
Node 16 -> Node 17	1139	1	0	729	99.91	64.00
TOTAL	**9032**	**1677**	**2890**	**2923**	**81.43**	**64.36**

Simulation Result of Scenario 3 for Two Black Holes AODV (Nodes 18 & 19)						
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop (18)**	**Black Hole Drop(19)**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1093	1	1043	49	99.91	99.91
Node 2 -> Node 3	1097	0	1097	0	100.00	100.00
Node 4 -> Node 5	1096	1	164	931	99.91	99.91
Node 6 -> Node 7	1096	0	894	202	100.00	100.00
Node 8 -> Node 9	1136	2	986	14	99.82	88.03
Node 10 -> Node 11	1097	0	1097	0	100.00	100.00
Node 12 -> Node 13	1096	5	1090	1	99.54	99.54
Node 14 -> Node 15	1095	1	24	1070	99.91	99.91
Node 16 -> Node 17	1049	0	0	883	100.00	84.18
TOTAL	**9855**	**10**	**6395**	**3150**	**99.90**	**96.85**

Simulation Result of Scenario 4 for Two Black Holes AODV (Nodes 18 & 19)						
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop (18)**	**Black Hole Drop(19)**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1200	0	967	0	100.00	80.58
Node 2 -> Node 3	1100	0	139	957	100.00	99.64
Node 4 -> Node 5	1105	774	320	0	29.95	28.96
Node 6 -> Node 7	1128	0	11	1078	100.00	96.54
Node 8 -> Node 9	1096	652	444	0	40.51	40.51
Node 10 -> Node 11	939	0	869	0	100.00	92.55
Node 12 -> Node 13	1099	1	604	491	99.91	99.64
Node 14 -> Node 15	989	1	974	0	99.90	98.48
Node 16 -> Node 17	1096	0	459	139	100.00	54.56
TOTAL	**9752**	**1428**	**4787**	**2665**	**85.36**	**76.42**

Simulation Result of Scenario 5 for Two Black Holes AODV (Nodes 18 & 19)						
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop (18)**	**Black Hole Drop(19)**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1096	706	0	390	35.58	35.58
Node 2 -> Node 3	1097	2	185	0	99.82	16.86
Node 4 -> Node 5	1097	0	1097	0	100.00	100.00
Node 6 -> Node 7	1127	1	477	617	99.91	97.07
Node 8 -> Node 9	1095	184	0	581	83.20	53.06
Node 10 -> Node 11	1101	1	166	18	99.91	16.71
Node 12 -> Node 13	1097	1	1096	0	99.91	99.91
Node 14 -> Node 15	559	0	419	0	100.00	74.96
Node 16 -> Node 17	1095	0	1095	0	100.00	100.00
TOTAL	**9364**	**895**	**4535**	**1606**	**90.44**	**65.58**

Appendix C.2: Sample Result for Original AODV with One Black Hole

Simulation Result of Scenario 1 for One Black Hole AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	978	0	128	100.00	13.09
Node 2 -> Node 3	979	0	979	100.00	100.00
Node 4 -> Node 5	1004	38	939	96.22	93.53
Node 6 -> Node 7	1097	1097	0	0.00	0.00
Node 8 -> Node 9	1096	143	953	86.95	86.95
Node 10 -> Node 11	979	0	43	100.00	4.39
Node 12 -> Node 13	979	0	979	100.00	100.00
Node 14 -> Node 15	978	0	978	100.00	100.00
Node 16 -> Node 17	998	1	143	99.90	14.33
TOTAL	**9088**	**1279**	**5142**	**85.93**	**56.58**

Simulation Result of Scenario 2 for One Black Hole AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1112	623	114	43.98	10.25
Node 2 -> Node 3	1097	0	1086	100.00	99.00
Node 4 -> Node 5	1100	147	452	86.64	41.09
Node 6 -> Node 7	1084	168	132	84.50	12.18
Node 8 -> Node 9	944	879	0	6.89	0.00
Node 10 -> Node 11	1106	653	39	40.96	3.53
Node 12 -> Node 13	181	181	0	0.00	0.00
Node 14 -> Node 15	983	1	7	99.90	0.71
Node 16 -> Node 17	966	0	12	100.00	1.24
TOTAL	**7461**	**2652**	**1842**	**64.46**	**24.69**

Simulation Result of Scenario 3 for One Black Hole AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1117	0	231	100.00	20.68
Node 2 -> Node 3	1094	0	102	100.00	9.32
Node 4 -> Node 5	1097	0	1097	100.00	100.00
Node 6 -> Node 7	1096	85	1008	92.25	91.97
Node 8 -> Node 9	1099	595	501	45.86	45.59
Node 10 -> Node 11	1096	82	220	92.52	20.07
Node 12 -> Node 13	1097	270	826	75.39	75.30
Node 14 -> Node 15	1109	391	629	64.74	56.72
Node 16 -> Node 17	1099	1	1091	99.91	99.27
TOTAL	**9904**	**1424**	**5705**	**85.62**	**57.60**

Simulation Result of Scenario 4 for One Black Hole AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1097	1097	0	0.00	0.00
Node 2 -> Node 3	1145	679	413	40.70	36.07
Node 4 -> Node 5	1090	130	580	88.07	53.21
Node 6 -> Node 7	1105	41	200	96.29	18.10
Node 8 -> Node 9	1096	475	7	56.66	0.64
Node 10 -> Node 11	1097	1097	0	0.00	0.00
Node 12 -> Node 13	1083	302	775	72.12	71.56
Node 14 -> Node 15	1096	1	445	99.91	40.60
Node 16 -> Node 17	1097	1097	0	0.00	0.00
TOTAL	**9906**	**4919**	**2420**	50.34	**24.43**

Simulation Result of Scenario 5 for One Black Hole AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1097	1	894	99.91	81.49
Node 2 -> Node 3	1096	0	1096	100.00	100.00
Node 4 -> Node 5	1097	0	1097	100.00	100.00
Node 6 -> Node 7	1098	0	1070	100.00	97.45
Node 8 -> Node 9	777	1	419	99.87	53.93
Node 10 -> Node 11	1096	0	276	100.00	25.18
Node 12 -> Node 13	1097	0	894	100.00	81.49
Node 14 -> Node 15	1097	0	575	100.00	52.42
Node 16 -> Node 17	1099	0	990	100.00	90.08
TOTAL	**9554**	**2**	**7311**	**99.98**	**76.52**

Appendix C.3: Sample Result for Original AODV without Black Hole

Simulation Result of Scenario 1 for AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1100	1089	1.00
Node 2 -> Node 3	1096	1095	0.09
Node 4 -> Node 5	1096	1094	0.18
Node 6 -> Node 7	1097	1097	0.00
Node 8 -> Node 9	1097	1095	0.18
Node 10 -> Node 11	1089	1086	0.28
Node 12 -> Node 13	1096	1096	0.00
Node 14 -> Node 15	1097	1096	0.09
Node 16 -> Node 17	1096	1096	0.00
TOTAL	**9864**	**9844**	**0.20**

Simulation Result of Scenario 2 for AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1117	1089	2.51
Node 2 -> Node 3	1095	1093	0.18
Node 4 -> Node 5	1096	1092	0.37
Node 6 -> Node 7	1097	1096	0.09
Node 8 -> Node 9	845	834	1.30
Node 10 -> Node 11	1062	1044	1.70
Node 12 -> Node 13	1156	1092	5.54
Node 14 -> Node 15	1067	918	13.96
Node 16 -> Node 17	1118	1096	1.97
TOTAL	**9653**	**9354**	**3.10**

Simulation Result of Scenario 3 for AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1097	1094	0.27
Node 2 -> Node 3	1096	1094	0.18
Node 4 -> Node 5	1095	1095	0.00
Node 6 -> Node 7	1097	1094	0.27
Node 8 -> Node 9	1096	1096	0.00
Node 10 -> Node 11	1095	1093	0.18
Node 12 -> Node 13	847	780	7.91
Node 14 -> Node 15	1097	1097	0.00
Node 16 -> Node 17	1097	1095	0.18
TOTAL	**9617**	**9538**	**0.82**

Simulation Result of Scenario 4 for AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1097	1091	0.55
Node 2 -> Node 3	1095	1095	0.00
Node 4 -> Node 5	1066	1042	2.25
Node 6 -> Node 7	1127	1093	3.02
Node 8 -> Node 9	1097	1097	0.00
Node 10 -> Node 11	1097	1097	0.00
Node 12 -> Node 13	1097	1096	0.09
Node 14 -> Node 15	1097	1097	0.00
Node 16 -> Node 17	1095	1095	0.00
TOTAL	**9868**	**9803**	**0.66**

Simulation Result of Scenario 5 for AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1097	1097	0.00
Node 2 -> Node 3	147	147	0.00
Node 4 -> Node 5	1095	1091	0.37
Node 6 -> Node 7	1095	1093	0.18
Node 8 -> Node 9	625	577	7.68
Node 10 -> Node 11	1111	1084	2.43
Node 12 -> Node 13	1104	1094	0.91
Node 14 -> Node 15	1103	1088	1.36
Node 16 -> Node 17	1097	1097	0.00
TOTAL	**8474**	**8368**	**1.25**

Appendix C.4: Sample Result for Modified AODV without Black Hole

Simulation Result of Scenario 1 for Modified AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1089	1069	1.84
Node 2 -> Node 3	1078	632	41.37
Node 4 -> Node 5	1110	1079	2.79
Node 6 -> Node 7	1083	1082	0.09
Node 8 -> Node 9	964	747	22.51
Node 10 -> Node 11	1131	1081	4.42
Node 12 -> Node 13	813	598	26.45
Node 14 -> Node 15	1097	1097	0.00
Node 16 -> Node 17	1081	1025	5.18
TOTAL	**9446**	**8410**	**10.97**

Simulation Result of Scenario 2 for Modified AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1097	847	22.79
Node 2 -> Node 3	978	894	8.59
Node 4 -> Node 5	1094	1088	0.55
Node 6 -> Node 7	1081	1041	3.70
Node 8 -> Node 9	996	796	20.08
Node 10 -> Node 11	1097	1092	0.46
Node 12 -> Node 13	1088	1085	0.28
Node 14 -> Node 15	1096	929	15.24
Node 16 -> Node 17	1089	862	20.84
TOTAL	**9616**	**8634**	**10.21**

Simulation Result of Scenario 3 for Modified AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1097	1096	0.09
Node 2 -> Node 3	1097	1096	0.09
Node 4 -> Node 5	1097	1095	0.18
Node 6 -> Node 7	1029	1028	0.10
Node 8 -> Node 9	1064	1009	5.17
Node 10 -> Node 11	1094	1073	1.92
Node 12 -> Node 13	1085	1064	1.94
Node 14 -> Node 15	1097	1094	0.27
Node 16 -> Node 17	1096	1082	1.28
TOTAL	**9756**	**9637**	**1.22**

Simulation Result of Scenario 4 for Modified AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1097	1096	0.09
Node 2 -> Node 3	1097	1096	0.09
Node 4 -> Node 5	1097	1095	0.18
Node 6 -> Node 7	1029	1028	0.10
Node 8 -> Node 9	1064	1009	5.17
Node 10 -> Node 11	1094	1073	1.92
Node 12 -> Node 13	1085	1064	1.94
Node 14 -> Node 15	1097	1094	0.27
Node 16 -> Node 17	1096	1082	1.28
TOTAL	**9756**	**9637**	**1.22**

Simulation Result of Scenario 5 for Modified AODV			
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Loss %**
Node 0 -> Node 1	1102	1018	7.62
Node 2 -> Node 3	1090	1088	0.18
Node 4 -> Node 5	1096	1095	0.09
Node 6 -> Node 7	1022	1020	0.20
Node 8 -> Node 9	1095	1034	5.57
Node 10 -> Node 11	169	141	16.57
Node 12 -> Node 13	962	753	21.73
Node 14 -> Node 15	1091	1052	3.57
Node 16 -> Node 17	1088	1068	1.84
TOTAL	**8715**	**8269**	**5.12**

Appendix C.5: Sample Result for Modified AODV with one Black Hole

Simulation Result of Scenario 1 for One Black Hole Modified AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	978	160	434	83.64	44.38
Node 2 -> Node 3	1088	229	303	78.95	27.85
Node 4 -> Node 5	1052	0	296	100.00	28.14
Node 6 -> Node 7	1091	741	106	32.08	9.72
Node 8 -> Node 9	1096	997	97	9.03	8.85
Node 10 -> Node 11	860	1	168	99.88	19.53
Node 12 -> Node 13	1094	0	1023	100.00	93.51
Node 14 -> Node 15	628	0	628	100.00	100.00
Node 16 -> Node 17	1089	39	588	96.42	53.99
TOTAL	**8976**	**2167**	**3643**	**75.86**	**40.59**

Simulation Result of Scenario 2 for One Black Hole Modified AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1094	1090	4	0.37	0.37
Node 2 -> Node 3	1097	1096	1	0.09	0.09
Node 4 -> Node 5	1098	0	1097	100.00	99.91
Node 6 -> Node 7	1093	716	8	34.49	0.73
Node 8 -> Node 9	1044	6	135	99.43	12.93
Node 10 -> Node 11	1102	79	1015	92.83	92.11
Node 12 -> Node 13	1100	19	1075	98.27	97.73
Node 14 -> Node 15	1097	0	1097	100.00	100.00
Node 16 -> Node 17	1121	408	7	63.6	0.62
TOTAL	**9846**	**3414**	**4439**	**65.33**	**45.08**

Simulation Result of Scenario 3 for One Black Hole Modified AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1099	27	2	97.54	0.18
Node 2 -> Node 3	1114	465	76	58.26	6.82
Node 4 -> Node 5	980	973	6	0.71	0.61
Node 6 -> Node 7	1131	226	36	80.02	3.18
Node 8 -> Node 9	1096	1089	7	0.64	0.64
Node 10 -> Node 11	1097	1096	1	0.09	0.09
Node 12 -> Node 13	1097	7	300	99.36	27.35
Node 14 -> Node 15	993	719	151	27.59	15.21
Node 16 -> Node 17	1096	29	1	97.35	0.09
TOTAL	**9703**	**4631**	**580**	**52.27**	**5.98**

Simulation Result of Scenario 4 for One Black Hole Modified AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1097	1097	0	0.00	0.00
Node 2 -> Node 3	1145	679	413	40.7	36.07
Node 4 -> Node 5	1090	130	580	88.07	53.21
Node 6 -> Node 7	1105	41	200	96.29	18.10
Node 8 -> Node 9	1096	475	7	56.66	0.64
Node 10 -> Node 11	1097	1097	0	0.00	0.00
Node 12 -> Node 13	1083	302	775	72.11	71.56
Node 14 -> Node 15	1096	1	445	99.91	40.60
Node 16 -> Node 17	1097	1097	0	0.00	0.00
TOTAL	**9906**	**4919**	**2420**	**50.34**	**24.43**

Simulation Result of Scenario 5 for One Black Hole Modified AODV (Node 19)					
Sending Node -> Receiving Node	**Sent Packets**	**Received Packets**	**Black Hole Drop**	**Loss %**	**Black Hole Loss %**
Node 0 -> Node 1	1097	1	894	99.91	81.49
Node 2 -> Node 3	1096	0	1096	100.00	100.00
Node 4 -> Node 5	1097	0	1097	100.00	100.00
Node 6 -> Node 7	1098	0	1070	100.00	97.45
Node 8 -> Node 9	777	1	419	99.87	53.93
Node 10 -> Node 11	1096	0	276	100.00	25.18
Node 12 -> Node 13	1097	0	894	100.00	81.49
Node 14 -> Node 15	1097	0	575	100.00	52.42
Node 16 -> Node 17	1099	0	990	100.00	90.08
TOTAL	**9554**	**2**	**7311**	**99.98**	**76.52**