ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Preventing Flooding Attack in MANETs using the reserved bits of AODV messages

By

Teklay Gebremichael

March 2014

ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Preventing Flooding Attack in MANETs using the reserved bits of AODV messages

By: Teklay Gebremichael

Advisor: Dr. Yalemzewd Negash

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer Engineering

March 2014

ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Preventing Flooding Attack in MANETs using the reserved bits of AODV messages

By: Teklay Gebremichael

Approval by Board of Examiners

_____Prof. H.Y.Kwon_____ _____

Head, School of Electrical and Computer Engineering Signature

____Dr. Yalemzewd Negash_____ _____

Advisor Signature

____Mr. Menore Tekeba_____ _____

Internal Examiner Signature

____Dr. Nune Sreenavas_____ _____

External Examiner Signature

DECLARATION

I, the undersigned, hereby declare that this thesis is my original work performed under the supervision of Dr. Yalemzewd Negash, has not been presented as a thesis for a degree program in any other university and all sources of materials used for the thesis are duly acknowledged.

Name: Teklay Gebremichael

Signature: _____

Date of submission:_____

This thesis has been submitted for examination with my approval as university advisor.

Dr. Yalemzewd Negash _____

Advisor Signature

ACKNOWLEDGMENT

I would like to thank my advisor, Dr. Yalemzewd Negash, for his guidance and advice during each stage of this research study. I want also to express my sincere appreciation to my friends for their encouragement and concern.

Finally, my heartfelt appreciation goes to my family for their love, support, patience, and encouragement throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AODV	Ad-hoc On-demand Distance Vector
ASR	Anonymous Secure Routing
AWK	Aho, Weinberger, Kernigham
CBR	Constant Bit Rate
DoS	Denial of Service
DPQ	Data Processing Queue
DSDV	Destination-Sequenced Distance Vector
DSR	Dynamic Source Routing
FAP	Flooding Attack Prevention
FSR	Fish-eye State Routing
FT-AODV	Fault Tolerant Ad-hoc On-demand Distance Vector
IPQ	Immediate Processing Queue
IEEE	Institute of Electrical and Electronics Engineers
MAC	Medium Access Control
MANET	Mobile Ad hoc Network
NAM	Network Animator
NS-2	Network Simulator-2
OLSR	Optimized Link State Routing
OTCL	Object-oriented Tool Command Language
PDR	Packet Delivery Ratio
PDM	Period-based Defense Mechanism
QoS	Quality of Service
RERR	Route Error

RREP	Route Reply
RREQ	Route Request
RREQQ	Route Request Processing Queue
RRFA	Route Request Flooding Attack
RRFD	Route Request Flooding Defense
SADOV	Secured Ad-hoc On-demand Distance Vector
SLSP	Secured Link State Protocol
TCL	Tool Command Language
TCP	Transmission Control Protocol
TORA	Temporally Ordered Routing Algorithm
WMN	Wireless Mesh Network
WSN	Wireless Sensor Network
ZRP	Zone Routing Protocol

ABSTRACT

The flexibility and other advantages of mobile ad-hoc networks come with different security flaws, and flooding attack is one of the common attacks in reactive routing protocols. In ad-hoc flooding attack, the attacker either broadcasts a lot of route request packets or sends a lot of attacking data packets to exhaust the nodes' resources. Most of the existing prevention mechanisms are based on the concept of rate limit or trust values. In the former mechanisms, flooding attack below the rate limit cannot be prevented and the later solutions are computationally intensive in high mobility networks.

The new prevention mechanism uses the reserved bits of the route request message and one of the existing solutions called filtering to prevent flooding attacks from external and internal attackers respectively. While sending route request packets, some information identifying the originator is embedded into the reserved bits of the route request message, and the receiver verifies the request by checking the value of the reserved bits or the rate limitation and blacklists the sender if it is an attacker. Three scenarios are designed and four performance metrics are used to study the effect of the flooding attack and the efficiency of the prevention mechanism.

The results of the scenarios show that the effect of flooding attack is severe for higher route request flooding rates and increasing rate of attacking data packets. In addition, multiple attackers which do not violate the rate limit can also significantly flood a given network. The new prevention solution improves the network performance in all cases especially if the majority of the attackers are external. For example, the packet delivery ratio for 16 attackers, abiding by a rate limit, is enhanced to about 99% by the new prevention when all the attackers are external and 73% when half of them are internal compared to 13.3% in the case of the route request flooding attack. In data flooding too, the improvement of the new solution is significant that there is 8.6% of packet delivery ratio increase when all the attackers are external over the data flooding attack with attack rate of 50 packets/s.

The simulation, in general, illustrates that if some proportions of the attackers are external, then there is a considerable improvement in network performance using the new mechanism to prevent both route request and data flooding attacks as external attackers will be isolated the moment they start sending route request packets.

Keyword: Preventing flooding attack, Data flooding attack, Route request flooding attack, AODV

CHAPTER ONE

INTRODUCTION

1.1. OVERVIEW

The proliferation of mobile computing and communication devices (e.g., cell phones, laptops, hand held digital devices, personal digital assistants, or wearable computers) is driving a revolutionary change in our information society. We are moving from the Personal Computer age (i.e., a one computing device per person) to the Ubiquitous Computing age in which a users utilize, at the same time, several electronic platforms through which they can access all the required information whenever and wherever needed [1].

With the advent of modern technology, everyone desires to stay connected, everywhere, all the time. The ever increasing demand for keeping in touch with the outside world has not only triggered a never ending technological development, but is also affecting the social life of an ordinary person in various ways. Telephonic conversation is no longer the only requirement for telecommunication users, now they like to play interactive games, share videos and files, browse the Internet, use social networking sites, check emails, hold video conferences and use other multimedia applications. Moreover, the users own more than one communication device and there is a requirement for interconnectivity [2].

The IEEE 802.11 standards specify two operating modes: infrastructure mode and ad hoc mode. Infrastructure mode is used to connect clients with wireless network adapters, to an existing wired network with the help from wireless router or access point. Ad hoc mode is used to connect wireless clients directly together, without the need for a wireless router or access point [3].

While infrastructure-based network provides a great way for mobile devices to get network services, it takes time and potentially high cost to set up the necessary infrastructure. More recently, new alternative ways to deliver the services have been emerging. These are focused around having the mobile devices connect to each other in the transmission range through automatic configuration, setting up an ad hoc mobile network that is both flexible and powerful. In this way, not only can

mobile nodes communicate with each other, but can also receive Internet services through Internet gateway node, effectively extending Internet services to the non-infrastructure area [1].

A Mobile Ad Hoc Network (MANET) represents an infrastructure-less distributed system that comprises wireless mobile nodes that can freely and dynamically self organize into arbitrary and temporary “ad-hoc” network topologies, allowing devices to seamlessly inter-network with no pre-existing communication infrastructure. Infrastructure wired or wireless networks refer to networks that possess communication infrastructures (e.g. Routers, gateways, base-stations, etc). The internet and traditional cellular wireless networks are typical examples of infrastructure networks [21]. The infrastructure and infrastructure-less networks are shown in figures 1.1 and 1.2 respectively.

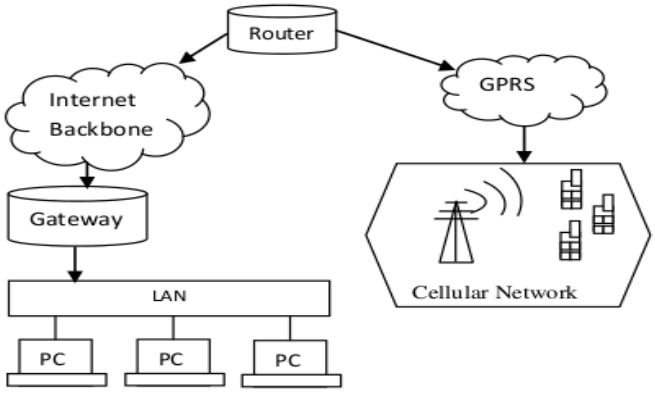

Figure 1.1:: Infrastructure networks [21]

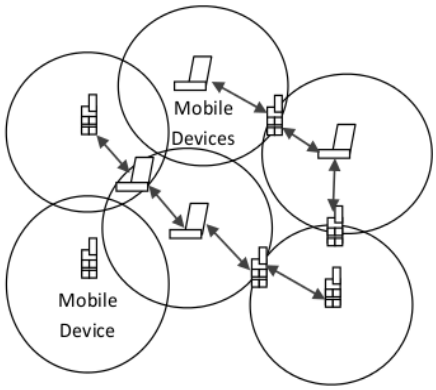

Figure 1.2: Infrastructure-less networks [21]

The field of ad-hoc networking itself contains several subfields such as Mobile Ad-Hoc Networks (MANETs) where all nodes are assumed to be mobile, Wireless Mesh Networks (WMNs) a

combination of ad-hoc and infrastructure network, Wireless Sensor Networks (WSNs) ad-hoc networks made up of small sensor devices, and Vehicular Ad-hoc Networks (VANETs) [4].

Mobile Ad Hoc Networks are useful in situations like Military Communication automated Battle fields, Emergency Services (Disaster recovery and earthquakes), Educational Applications (Setup virtual class & conference rooms) and Entertainment (Multi-user games) [5]. At home ad-hoc networks can also provide connectivity to different personal devices such as notebooks, PDAs, TV, cell phones and even appliances which include air conditioners, microwave oven etc. Since no cost is required to establish the infrastructure, the services offered are quite cheap and are attractive for masses [2].

Ad-hoc networks are typically decentralized and do not feature dedicated devices with defined roles such as routers or switches. Instead all participating nodes act as both routers and end-users. As devices are limited by their radio range ad-hoc networks typically employ a strategy known as multi-hopping in which a source node will send a message to the destination by passing it to a series of intermediate nodes. This enables geographically disparate nodes to communicate wirelessly. Multi-hopping is typical of the distributed architecture of ad-hoc networks [4].

The highly dynamic nature of a mobile ad hoc network results in frequent and unpredictable changes of network topology, adding difficulty and complexity to routing among the mobile nodes. The challenges and complexities, coupled with the critical importance of routing protocol in establishing communications among mobile nodes, make routing area the most active research area within the MANET domain. Numerous routing protocols and algorithms have been proposed, and their performance under various network environments and traffic conditions has been studied and compared [6].

Routing mechanisms are more vulnerable in ad hoc networks than in conventional networks because in ad hoc networks, each device acts as a relay. This means, for example, that an adversary who hijacks an ad hoc node could paralyze the entire network by disseminating false routing information. Moreover, weaknesses in the protocols can be exploited to perform malicious neighbor discovery [7]. And some of the most common attacks in mobile ad hoc networks are surveyed in [9, 23, 33, 39-45].

The strict resource constraints in MANETs constitute nontrivial challenge to security design. The wireless channel is bandwidth-constrained and shared among multiple networking entities. The computation capability of a mobile node is also constrained. The wireless medium and node mobility poses far more dynamics in MANETs compared to the wireline networks. The network topology is highly dynamic as nodes frequently join or leave the network, and roam in the network on their own will. Despite such dynamics, mobile users may request for anytime, anywhere security services as they move from one place to another [8].

It has been observed that although active research is being carried out in this area, the proposed solutions are not complete in terms of effective and efficient routing security. There are limitations on all solutions. They may be of high computational or communication overhead (in case of cryptography and key management based solutions) which is detrimental in case of resource constrained MANETS, or of the ability to cope with only single malicious node and ineffectiveness in case of multiple colluding attackers. Furthermore, most of the proposed solutions can work only with one or two specific attacks and are still vulnerable to unexpected attacks [9].

Mobile ad hoc networks should posses a better and effective security as various upcoming applications based on MANET are on their way in future. Mitigating the routing issues will create a better, efficient, and secure application in mobile ad hoc networks [10].

1.2. STATEMENT OF THE PROBLEM

Attackers can exploit on-demand routing protocols to initiate different attacks. As described in [11], in ad hoc flooding attack, the attacker either broadcasts a lot of Route Request (RREQ) packets for a node which is not in the network or sends a lot of attacking data packets to exhaust the communication bandwidth and nodes' resources. The RREQ messages will be received by all the neighbors of the originator since RREQ packets are broadcasted to discover a route. The intermediate nodes will look up their routing tables for possible route to the destination, but none of the nodes will have a route to the destination node specified in the RREQ message and the intermediate nodes will flood the RREQ packet to their respective neighbors. The malicious node may worsen the situation by repeating this procedure frequently. As a result, the requests will be routed through all nodes there by creating a congested network and consuming the scarce network resources like bandwidth and battery life. By doing so, the route request flooding attack may lead to denial of service attack as there will be more RREQ packets in the network and the nodes will be busy in processing and forwarding these fake requests rather than relaying genuine packets. This is due to the fact that control packets like RREQ have higher priority than data packets. The attacker can also send useless data packets to overuse the available bandwidth. Different literatures, [11, 12, 19, 46, 47], have shown that the presence of malicious flooding nodes in MANET can affect the performance of the overall wireless network and flooding attack can act as one of the major security threats in MANETs.

1.3. PROPOSED SOLUTION

A new prevention mechanism is proposed that uses the reserved bits of the messages of AODV routing protocol to prevent flooding attacks that can be imposed by external attackers. The genuine nodes maintain common keys initially configured. When a node sends a request, it first manipulates its address using the two keys and sends this value embedded in the reserved bits of the RREQ message. The receiving node, on the other hand, verifies the RREQ by extracting the value in the reserved bits to determine whether the sender is an intruder or not. This is done by first manipulating the source address of the originator using the two keys and comparing it with the value extracted from the reserved bits of the received RREQ message. If the receiver detects a malicious node that doesn't have a right key, then it blocks the sender from using the network by dropping its requests. And further RREQ messages from the attacker are prevented by blacklisting the misbehaving node. Since

the malicious node is unable to establish link with its neighbors without having the keys, the data flooding is also effectively avoided as data flooding attack occurs after successfully establishing active link. So, by using the reserved bits, both route request and data flooding attacks can be completely prevented from external attackers and the attackers are isolated from the network by their respective neighbors. In addition, to prevent flooding attacks from internal attackers, those malicious nodes which have the keys, one of the existing preventions can be integrated into this solution. The idea of using the reserved bits is taken from [62] which used some of the reserved bits to transmit energy information for building energy efficient routing algorithm.

1.4. OBJECTIVE

1.4.1. GENERAL OBJECTIVE

The general objective of this thesis work is to develop a mechanism to detect and prevent both RREQ and data flooding attacks in AODV routing protocol from external attackers by using the reserved bits of the RREQ messages.

1.4.2. SPECIFIC OBJECTIVES

- To study literatures to understand and examine the effect of flooding attack on network performance
- To find a mechanism for nodes to identify external attackers while discovering routes
- To model the flooding attack and design the flooding attack prevention mechanism in AODV routing protocol
- To modify the existing C++ code of AODV routing protocol to incorporate the flooding attack and the proposed prevention mechanism into the protocol
- To design simulation scenarios that will be used to study the effect of both flooding attacks on network performance and the effectiveness of the prevention algorithm
- To simulate the designed scenarios using NS-2 simulator
- To discuss and interpret the simulation results

1.5. METHODOLOGY

Different literatures have been reviewed to study how AODV works and the different types of attacks that mobile ad hoc networks face and particularly the effect of flooding attack in affecting network performance. The literatures developed for preventing flooding attacks are also discussed along with their limitations and are presented in this thesis research. First, the flooding attack is modeled and the prevention algorithm is also designed.

Then, the attack and the prevention mechanism are implemented by modifying the existing C++ code of the AODV routing protocol. NS-2 is used as a simulation platform for this thesis work and the implementation of AODV in NS-2 is also studied. To study the effect of the flooding attack, the modified routing protocol is simulated in the absence of any prevention mechanism. The effectiveness of the prevention mechanism is evaluated by using the modified version of the AODV protocol and one of the existing solutions, [15], is used for comparison.

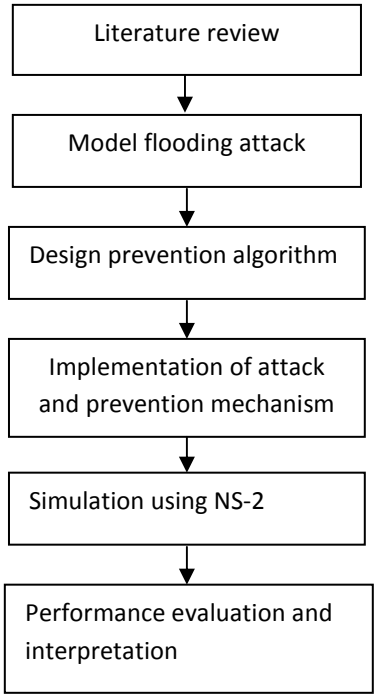

Figure 1.3: Methodology used

Three simulation scenarios are designed to study the effect of the attack and the effectiveness of the prevention mechanism. The output of the NS-2 simulation, trace file, is studied to measure the performance by using packet delivery ratio, average end-to-end delay, routing load, and the average

energy remaining at the end of the simulation as performance metrics. The trace file is analyzed using the AWK script and the results are presented in graphs using gnuplot. The simulation scenarios are run five times and the average of the results is taken. The results are then presented and interpreted as described in chapter 6. The summary of the methodology used in this thesis work is summarized in figure 1.3 diagrammatically.

1.6. CONTRIBUTION

The major contribution of this work is that it excludes external unauthorized nodes from participating in the process of routing and sending data and hence preventing multiple attacks. Being a lightweight prevention mechanism, the incorporation of this solution does not change the protocol's message formats or sizes. The effect of flooding attack on energy consumption of the nodes is also studied here which does not exist in the previous studies.

The following are the main contributions of the proposed prevention mechanism.

- External attackers are prevented from imposing both route request and data flooding attacks by completely isolating them. Since, external attackers do not embed the appropriate keys in the reserved bits of the AODV message; they can be identified as malicious even with a single RREQ message before discovering a route.
- Route request flooding attack occurring below the rate limitation can be prevented which is the limitation of the solutions that use the concept of threshold. Hence, the severe effect of flooding by many external attackers sending RREQ messages abiding by the threshold limitation to flood a network cooperatively is avoided.

1.7. SCOPE

This thesis work focuses on a simulation study to propose and implement a prevention mechanism for both route request and data flooding attacks imposed by external attackers in AODV routing protocol by making use of the reserved bits of the RREQ message. Flooding attacks by other control packets other than RREQ and attacks in other reactive routing protocols are not studied.

1.8. THESIS ORGANIZATION

The rest of the document is organized as follows. Chapter two discusses the works related to prevention of flooding attack in MANETs. In chapter three, routing protocols in MANETs are reviewed and specifically the AODV routing protocol is studied in detail regarding its operation including route discovery and route maintenance. The vulnerabilities of MANETs are discussed in chapter four. The sources of vulnerabilities and the types of attacks in MANETs including the flooding attack are also studied in this section. Chapter five introduces the NS-2 simulator and its installation, implementation of AODV in NS-2, modeling flooding attack, design of the prevention algorithm and their implementations. The prevention algorithm is described in flowcharts in this chapter. The simulation study is discussed in chapter six. Here, the simulation scenarios used in this study, the performance metrics, the results of the simulations and the discussions of the results are presented. Chapter seven presents the conclusions and recommendations. Then the reference materials used to develop this thesis work are presented next. Finally, in the appendix, sample of TCL codes, AWK scripts, mobility files and traffic scenario files are presented in addition to the C++ codes of the flooding attack and the prevention mechanism that are implemented in NS-2 simulator.

CHAPTER TWO

RELATED WORKS

In this section, previous research works which are related to this thesis work, that is, preventing flooding attacks, are examined. The area of focus and the limitations of these works are also discussed.

The authors in [11] are the first to present a new DoS attack, called ad hoc flooding attack, and its defense in ad hoc networks. The attack can result in denial-of-service when used against on-demand routing protocols such as AODV and DSR. The intruder broadcasts mass Route Request packets or sends a lot of attacking DATA packets to exhaust the communication bandwidth and nodes' resources. After analyzing ad hoc flooding attack, the authors developed flooding attack prevention (FAP) to prevent the ad-hoc flooding attack. The FAP is composed of neighbor suppression and path cutoff. When the intruder broadcasts exceeding packets of route request, the immediate neighbors of the intruder observe a high rate of route request and then they lower the corresponding priority according to the rate of incoming queries and low priority queries are eventually discarded. The main idea of neighbor suppression is that each neighbor calculates the rate of RREQ originated by intruder. If the rate exceeds some threshold, in this case 15 RREQs/s, the node is added to black list as intruder. When the intruder sends many attacking DATA packets to the victim node, the node may cut off the path and does not set up a path with the intruder any more. If all neighbor nodes around the node refuse to forward its packets, the node cannot communicate with the other nodes in mobile ad hoc networks. The node has been isolated from the network in practice even if it is still in the networks in location. They studied for attacking packets of 10packets/s, 20packets/s, 30packets/s and 40packets/s and found that the prevention is more effective for higher flooding rates. The threshold limit used in FAP is 15 RREQ/s which is too large to prevent flooding attacks that occur below the rate of 15 RREQs/s. Very few attackers can also significantly flood a given network with fake RREQs without violating the threshold limit of FAP.

[12] Proposes a scheme to mitigate Malicious Control Packet Floods in Ad Hoc Networks and evaluates the effect of control packet flooding by malicious nodes on a MANET's performance for AODV protocol. The proposed technique uses statistical analysis to detect misbehaving nodes and

reduces their impact on network performance. Each node monitors the route requests it receives. Each node maintains a count of RREQs received for each RREQ sender during a preset time period ($\delta\tau$). At the end of the time period, the node computes the rate at which it has been receiving route requests from each sender and also computes the smoothed average and smoothed average deviation of all RREQ sources at the end of the time period repeatedly for each RREQ sender. To distinguish between malicious RREQ floods and those by normal nodes, a cut-off rate is calculated based on the smoothed average and deviations. The RREQs from a sender whose smoothed average rate is above the CutOffRate will be dropped without forwarding. The paper showed that, prior to saturation, good nodes drop over 90% of the malicious RREQs received by them. At and beyond saturation, normal nodes that need to establish long routes also send abnormally high RREQs, which tends to increase the overall node average. This in turn reduces the fraction of malicious RREQs dropped in low-rate attacks. Very few RREQs are dropped for loads below saturation and the solution is effective for high route request rates.

An obligation-based model called Followship is proposed in [13] to mitigate the flooding and packet dropping attacks. The architecture of the scheme comprises three operational components namely rate-limitation component, enforcement component and restoration component. Rate-limitation minimizes the flooding attacks while the enforcement component reduces the packet drop attacks. And the restoration component is used to resolve the ambiguity between the intentional and accidental packet drops. Followship model is enforced at every node where each incoming packet has to pass through the three components before being sent out. The rate-limitation component monitors each requesting node's channel usage at regular intervals of time. If the packets generated by the neighbor exceed the transmission-threshold within the given interval, then the neighbor is anticipated to be malicious or selfish and the packet is discarded. And this node is penalized for this by decrementing its contribution-count, the node's committed resource to the network. But, the proposed solution is not verified mathematically or using simulation. Although the solution is proposed to prevent flooding and dropping attacks, it uses rate limitation to prevent flooding attack which is ineffective at lower attack rates.

Another prevention mechanism is presented in [14]. According to this paper, route request flooding attack can be classified in to two: breadth-RRFA and depth-RRFA. In breadth-RRFA, an attacker would initiate route discoveries to a large number of unreachable destinations. In depth-RRFA, the

attacker will send out a large number of RREQs repeatedly to one unreachable destination. The proposed scheme is called Route Request Flooding Defense (RRFD) and uses the binary exponential back off algorithm to delay repeated route requests to punish the node sending many route requests per unit time. In RREQ binary exponential back off, each node will ensure that its neighbor follows a binary exponential back off algorithm when sending RREQs in a Route Discovery Cycle. If RREQs are sent faster than what is allowed, excess RREQs are dropped. This ensures that the generation of RREQs in a Route Discovery Cycle follows a binary exponential back off as stated in the AODV specifications. They have proven that RRFD is at least 50% effective against a breadth-RRFA and RRFD is found to be very effective against depth-RRFA as the success rate can approach 100% if the attacker is very aggressive in sending large number of RREQs. The repeated delays of route request reduces throughput due to increased waiting and the solution also considers only the route request flooding attack. It is not effective in preventing breadth-RRFA and the solution cannot also isolate the malicious node.

[15] Proposed a technique that uses a filter to detect misbehaving nodes and reduces their impact on network performance. The aim of the filter is to limit the rate of RREQ packets. Each node maintains two threshold values. The threshold values are the criterion for each node's decision of how to react to a RREQ message. The RATE_LIMIT parameter denotes the number of RREQs that can be accepted and processed as normal per unit of time. If the rate of this RREQ originator is below the RATE_LIMIT, the RREQ packet is processed as normal. The BLACKLIST_LIMIT parameter is used to specify a value that aids in determining whether a node is acting as malicious or not. If the number of RREQs originated by a node per unit time exceeds the value of BLACKLIST_LIMIT, the sender node is identified as malicious and it will be blacklisted. This will prevent further flooding of the fake RREQs in the network. By blacklisting a malicious node, all neighbors of the malicious node restrict the RREQ flooding. Also the malicious node is isolated due to this distributed defense and so cannot hog its neighbor's resources. If the rate of RREQs originated by a node is between the RATE_LIMIT and the BLACKLIST_LIMIT, the RREQ packet is added to a "delay queue" waiting to be processed. Then the malicious node with a high attack rate will thus be severely delayed. The paper used a rate of 5 requests per second for RATE_LIMIT and 10 requests per second for BLACKLIST_LIMIT. The paper claims enhancement in the performance of the network in presence of compromised nodes by analyzing the Packet Delivery Ratio and End-to-End Delay as performance metrics. The limitation of

this solution is flooding attack below the rate limit, 5 RREQs/s, cannot be avoided and many attackers can send route requests to congest a given network without violating the rate limitation.

A prevention mechanism based on rate-limitation is also stated in [16]. In this solution, each incoming packet passes through the rate-limitation component before being forwarded (transmitted) to the next-hop neighbor which is is developed for Anonymous Secure Routing Protocol. The ASR protocol neither contains the source node's address nor the destination node's address in it route request message. In addition, the intermediate nodes do not append their identity to the route request. To achieve this, the rate-limitation at every node uses a threshold-tuple (α, β , γ). 'α' gives the maximum number of packets a node can transmit in an interval. It is derived from the empirical results taken over the average number of packets transmitted in an interval by the node and the average number of neighbors in its environment. 'β' permits the maximum number of times a flooding node can exceed 'α' before being blacklisted. 'γ' denotes the number of consecutive intervals for which the flooding node has to abide within 'α', in order for it to be white-listed or redeemed. If the packets transmitted by the neighbor exceed the predetermined transmission-threshold (α) within a given interval, then the subsequent packets are dropped. If the same neighbor exceeds the transmission-threshold (α) by blacklist-threshold (β) intervals, then the neighbor is blacklisted and all the packets received from this neighbor are discarded in the future intervals.. However the node continues to monitor the behavior of the blacklisted node in the successive intervals. In order to be white-listed or redeemed, the blacklisted node has to exhibit benign behavior for 'γ' intervals, which is known as whitelist-threshold. The authors analyzed their solution with varying the thresholds and measure the number of packets flooded with time and claimed their solution to be promising in preventing flooding attack. This solution also uses the concept of threshold where it is possible malicious nodes can still flood the network by abiding the transmission-threshold (α), besides it is developed for ASR protocol.

[17] Uses a trust estimator in each node to evaluate the trust level of its neighboring nodes to prevent route request flooding attack in DSR. In this paper, the neighbors are categorized into friends (most trusted), acquaintances (trusted) and strangers (not trusted) based on their trust values. During network initiation all nodes will be strangers to each other. A stranger node is a node with minimum trust level. A node is acquaintance to its neighbor means it has received some packets through that node and has a trust level between the friends and strangers. Friends are most trusted nodes or the

nodes with highest trust level. Here the higher trust level means neighbors had received or transferred many packets successfully through this particular node. The result of trust estimation function is the relationship status of all of neighbors as friend, acquaintance or stranger. Different threshold values are defined for different types of neighbors to become friend, Acquaintance and stranger. *Tracq* and *Trfri* are the threshold values for the acquaintance and the friend respectively. The RREQ count is checked together with the friendship table. The thresholds are selected in such way that friends will have higher rates and strangers lower rates. If a neighbor is stranger, then its packet is discarded if the rate exceeds the threshold for stranger. If the neighbor is acquaintance, and its rate exceeds the limit then its packets are added to delay queue. The trust function has to be recalculated every time a node re-connects and thus this solution is computationally expensive and not effective for high mobility networks. This trust-based mechanism is also developed for another reactive protocol, DSR.

[18] Introduces the Period-based Defense Mechanism to prevent data flooding attack. The proposed PDM scheme sets up *w* periods for the data transmission. The PDM scheme checks data packet floods at the end of each period in order to enhance the throughput of burst traffic. The existing schemes may not guarantee the Quality of Service (QoS) of burst traffic since multimedia data are usually burst. The prevention mechanism can guarantee the Quality of Service (QoS) of burst traffic. At the end of the period, it compares the variance of received data packets with the variance limit. When variance is greater than limit, it checks whether data packets for source-destination pairs are in the blacklist or not. The blacklist is maintained by each mobile node, which is initially empty. The maximum number of received data packets for a certain source node – destination node pair is listed in the blacklist. It updates the blacklist by the greatest number of received data packets in the period. Hence, the proposed scheme uses a blacklist, considers the data type, and processes packets according to the priority so as to defend against data flooding attacks; since the attacker forwards many data packets at a high rate for the whole session. The solution is developed to differentiate attack traffic and burst traffic for preventing data flooding attack only to guarantee QoS.

[19] Provides a method to mitigate route query floods in MANETs. Each node sets an upper bound number for the storage of the resulting (partial) routes with respect to RREQs received from one source node. If the cached resulting (partial) routes for the source node exceed the bound, further RREQs from the source will be denied. In AODV, when an intermediate node receives a RREQ and decides to forward the RREQ, it will cache a reverse route entry for the originator node, and wait for

RREP (or other control packets). Each node sets an upper bound for the number of cached reverse route entries with respect to RREQs from one source node. The RREQ_threshold may be adjusted accordingly when the network situation changes. If a node sends RREQ much frequently than normal nodes or a node sends more RREQs, the RREQ_threshold for this node may be reduced as a punishment. When a malicious node broadcasts exceeding packets of Route Request and consequently the number of reverse route entries exceeds RREQ_threshold, nodes deny any future RREQ packets from the malicious node. If the count of route request is less than the threshold value, the RREQ packet is processed as normal. Otherwise, the RREQ is simply dropped or recorded for further process. This solution also uses the threshold approach and is not effective for collaborative attackers sending RREQ below the rate limit.

A prevention mechanism, similar to [17] that is developed for another reactive protocol DSR, is described in [20]. The solution scheme classifies all the nodes in an ad hoc network to be either member or guest according to the trust value and their relationships with their neighboring nodes. During network initiation all nodes will be guest. A trust function is used in each node to evaluate the trust level of its neighboring nodes. The trust level is a function of various parameters like length of the association, ratio of the number of packets forwarded successfully by the neighbor to the total number of packets sent to that neighbor, ratio of number of packets received intact from the neighbor to the total number of received packets from that node, average time taken to respond to a route request etc. Accordingly, the neighbors are categorized into guest or members. The guest node may have high chances of being a malicious node. Any new node entering ad hoc network will be a guest to all its neighbors and become member after gaining a certain level of trust, which is computed by a separate trust function. During route discovery phase of the DSR protocol, the extended system also computes the aggregate trust along different paths to the destination and the most trusted path between the source and the destination is found out before establishing the data transfer. If the count for forwarded packets for guest is less than the threshold the packet is forwarded and the count is incremented else the packet is dropped. After applying the trust function, the number of RREQ packets forwarded by an intermediate node has decreased but as can be seen from its simulation results the decrement is insignificant to prevent flooding attack. In addition, the trust function has to be recalculated every time a node associates and thus this solution is computationally expensive and not effective for high mobility networks.

[21] Proposed a scheme called Flood Tolerant AODV Protocol (FT-AODV) to prevent flooding attack. The basic idea of the scheme is that it gives packet processing priority to the nodes which have a longer history of stable behavior. It divides the single packet buffer into three different queues: Immediate Processing Queue [IPQ], RREQ Request Processing Queue [RREQQ], and Data Processing Queue [DPQ]. Any RERR or RREP packet arriving at the node enters the IPQ, any RREQ packet arriving at the node enters RREQQ and any Data packet arriving at the node enters the DPQ queue. The method makes use of the RREQ_RATE_LIMIT and a value called $\text{Threshold}_{\text{Reliable}}$ which denotes the minimum transfer of useful data packets from any node in the network for that node to be considered reliable. They assumed that a node which has stayed in the network for a longer time is more reliable than a new comer. IPQ has higher priority because forwarding RREP or RERR fast makes the route discovery quick. The RREQQ gets the next higher priority and packets in DPQ get the least priority. Depending on priorities these packets get preference of memory in the queue and dispatcher schedules. This solution also uses the threshold concept but applied to the queue and is ineffective for low rate attackers. Besides, it does not present any analysis on how well it prevents the flooding attack.

A solution for prevention of HELLO flooding attack, another flooding attack, is presented in [22]. As the hello packets are continuously flooded by the malicious node, the neighbor node is not able to process other packets. Absence of hello packet during the periodical hello interval may lead to wrong assumption that one of the intermediate neighbor nodes sends Route Error (RERR) message and the source node re-initiates the route discovery process. The hello interval is changed in a random fashion but it is limited between the maximum and minimum hello interval values. This method assumes hello interval values are changed in a random manner. This value is encrypted and attached in the header part of the data packet. Nodes that are located in the coverage area are able to process the header part of the packet and update this hello interval value and changing the time of sending hello packets its neighbor. But the malicious won't concentrate the processing of other packets; it continuously sends large number hello packets to its neighbor. It is unaware of these changes of hello interval. Malicious nodes are not aware of this change of hello interval, so they do not change the interval and continuously send the packet to their neighbor. This behavior exhibits the confirmation of malicious activity and the neighbor node ignores the processing of packets. This solution is developed for preventing another type of flooding attack, HELLO packets flooding attack.

In general, most of the prevention mechanisms are based on the concept of threshold limitation. The threshold limitation can be applied to prevent flooding attacks in different forms. But the limitation with such solutions is that they cannot prevent flooding attack below the rate limitation specified and the limitation can also be exploited to further flood a given network. This could happen by many attackers which send RREQs below the rate limit but the combined effect could be equivalent to one or more attackers flooding at high rate. The other set of solutions are based on the trust value relationships but these solutions are ineffective in high mobility networks requiring the trust functions to be re-computed when ever nodes rejoin the network. Thus such solutions are computationally expensive to be prominent solutions in the MANET networks.

It has been pointed out in [9] that the proposed solutions in MANETs are not complete in terms of effective and efficient routing security. There are limitations on all solutions. They may be of high computational or communication overhead which is detrimental in case of resource constrained MANETS, or most of the proposed solutions can work only with one or two specific attacks and are still vulnerable to unexpected attacks. So, in this case, many of the prevention solutions are developed to prevent either RREQ flooding attacks or data flooding attacks only. This is due to the fact that the nature of both flooding attacks is entirely different. So, preventing both types of flooding attacks with one prevention algorithm is an advantage as the network resources in MANET networks hinder for computationally expensive and efficient mechanisms.

CHAPTER THREE

ROUTING PROTOCOLS IN MANETS

This section discusses the overview of routing protocols in MANETs and the working principle of AODV routing protocol is also discussed in detail.

3.1. OVERVIEW OF ROUTING PROTOCOLS

The limitation on wireless transmission range requires routing in multiple hops. So the nodes depend on one another for transmission of packets from source nodes to destination nodes via the routing nodes [9]. The goal of routing in a MANET is to discover the most recent topology of a continuously changing network to find a correct route to a specific node [23].

Routing protocols for ad hoc wireless networks can be classified into three types based on the underlying routing information update mechanism employed. An ad hoc routing protocol could be reactive (on demand), proactive (table driven) or hybrid [9, 24]. Figure 3.1, shows the three types of ad hoc routing protocols and some list of the available routing protocols for that category.

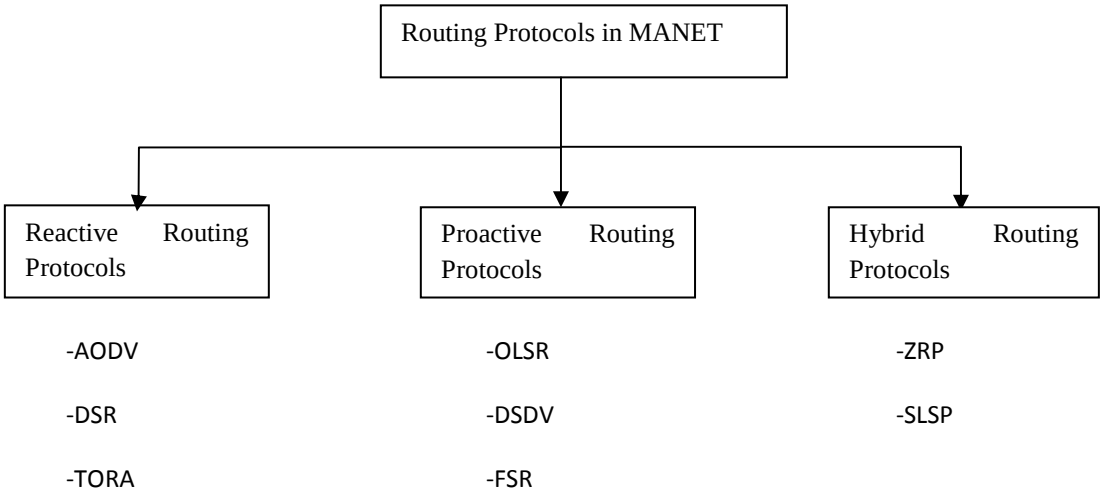

Figure 3.1: Routing protocols in MANET

Reactive MANET protocols only find a route to the destination node when there is a need to send data. The source node will start by transmitting route requests throughout the network. The sender will then wait for the destination node or an intermediate node to respond with a list of intermediate

nodes between the source and destination. Thus, reactive MANET protocols are most suited for networks with high node mobility or where the nodes transmit data infrequently. Examples of reactive MANET protocols include Ad Hoc On-Demand Distance Vector (AODV), Dynamic Source Routing (DSR), and Temporally Ordered Routing Algorithm (TORA) [9].

In proactive routing protocols, on the other hand, every node maintains the network topology information in the form of routing tables by periodically exchanging routing information. Routing information is generally flooded in the whole network. Whenever a node requires a path to a destination, it runs an appropriate path finding algorithm on the topology information it maintains. Thus, proactive MANET protocols work best in networks that have low node mobility or where the nodes transmit data frequently. Examples of proactive MANET protocols include Optimized Link State Routing (OLSR), Fish-eye State Routing (FSR), Destination-Sequenced Distance Vector (DSDV) [9, 24].

Hybrid routing protocols such as Zone Routing Protocol (ZRP) and Secured Link State Protocol (SLSP) combine the best features of both reactive and proactive routing protocols. For example, a node communicates with its neighbors using a proactive routing protocol, and uses a reactive protocol to communicate with nodes farther away. In other words, for each node, nodes within certain geographical are reached using proactive routing protocols. Outside the geographical area, reactive routing protocols will be used [24].

3.2. AD-HOC ON-DEMAND DISTANCE VECTOR ROUTING (AODV) PROTOCOL

The Ad hoc On-Demand Distance Vector (AODV) [25, 26] algorithm enables dynamic, self-starting, multihop routing between participating mobile nodes wishing to establish and maintain an ad hoc network. AODV allows mobile nodes to obtain routes quickly for new destinations, and does not require nodes to maintain routes to destinations that are not in active communication. One distinguishing feature of AODV is its use of a destination sequence number for each route entry. The destination sequence number is created by the destination to be included along with any route information it sends to requesting nodes. Using destination sequence numbers ensures loop freedom. A sequence number is a monotonically increasing number maintained by each originating node and is used by other nodes to determine the freshness of the information contained from the originating node. Route Requests (RREQs), Route Replies (RREPs), and Route Errors (RERRs) are the message

types defined by AODV. These packet types are received via UDP, and normal IP header processing applies.

The message formats for RREQ, RREP and RERR messages are defined in the draft of AODV, RFC 3561. The message format for RREQ is shown is figure 3.2 and it includes the following fields and different flags.

RREQ ID: A sequence number uniquely identifying the particular RREQ when taken in conjunction with the originating node's IP address.

Destination IP Address: The IP address of the destination for which a route is desired.

Destination Sequence Number: The latest sequence number received in the past by the originator for any route towards the destination.

Originator IP Address: The IP address of the node which originated the Route Request.

Originator Sequence Number: The current sequence number to be used in the route entry pointing towards the originator of the route request.

Hop Count: The number of hops from the Originator IP Address to the node handling the request.

Reserved: Sent as 0; ignored on reception.

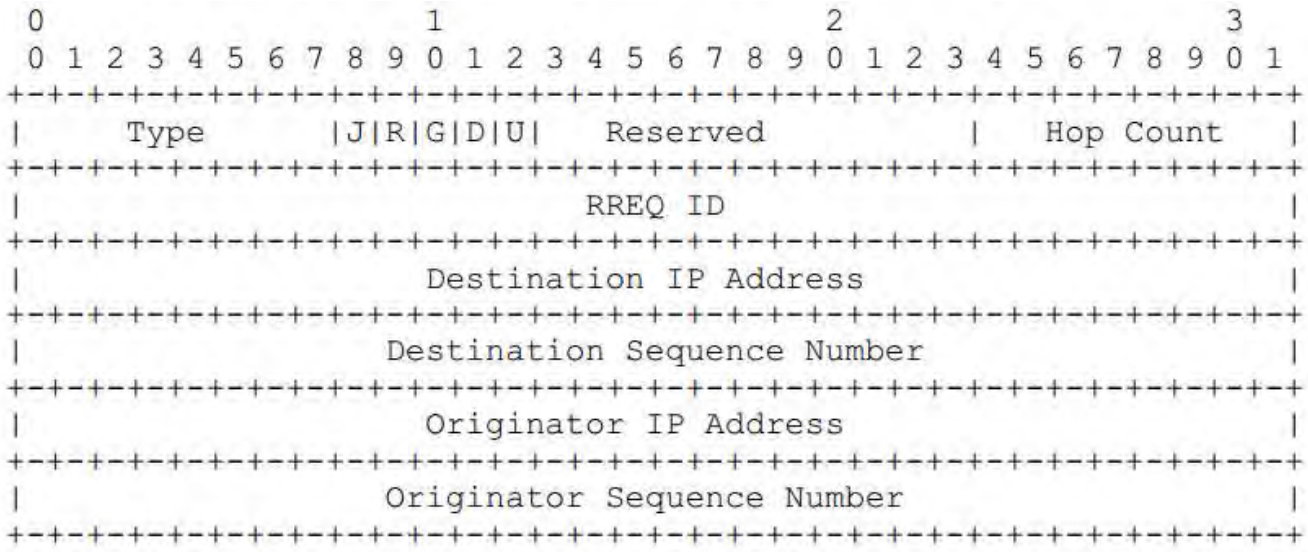

Figure 3.2: Route request (RREQ) message format [25]

The Route Reply (RREP) message format also includes Destination IP Address, which is the IP address of the destination for which a route is supplied, Destination Sequence Number, which is the destination sequence number associated to the route, and Originator IP Address is the IP address of the node which originated the RREQ for which the route is supplied.

AODV uses RREQ and RREP to find routes for a destination and RERR is used for route maintenance during failure. Route table information must be kept even for short-lived routes. Some of the fields AODV uses with each route table entry include Destination IP Address, Destination Sequence Number, Network Interface, Hop Count (number of hops needed to reach destination), Next Hop, Lifetime (expiration or deletion time of the route) and other state and routing flags (e.g., valid, invalid, repairable, being repaired).

3.2.1. AODV OPERATION

As described in [25, 26], every route table entry at every node must include the latest information available about the sequence number for the IP address of the destination node for which the route table entry is maintained. This sequence number is called the "destination sequence number". It is updated whenever a node receives new (i.e., not stale) information about the sequence number from RREQ, RREP, or RERR messages that may be received related to that destination. AODV depends on each node in the network to own and maintain its destination sequence number to guarantee the loop-freedom of all routes towards that node.

3.2.1.1. UPDATING ROUTING TABLE IN AODV

When a node receives an AODV control packet from a neighbor, it checks its route table for an entry for the destination. In the event that there is no corresponding entry for that destination, an entry is created. The sequence number is either determined from the information contained in the control packet, or else the valid sequence number field is set to false [25]. The route is only updated according to the following rule.

A node *i* applies the rules shown in figure 3.3 whenever it receives a route advertisement for destination *d* from a neighbor *j*. The variables $seqnum^{d}_{i}$, $hopcount^{d}_{i}$, and $nexthop^{d}_{i}$ denote the destination sequence number, the hop count and the next hop, respectively, for destination *d* at node *i* and $seqnum^{d}_{j}$, $hopcount^{d}_{j}$, and $nexthop^{d}_{j}$ denote the destination sequence number, the hop count and

the next hop, respectively, for destination *d* at node *j*. This means that an entry will be updated if the new information has greater or equal sequence number than the receiving nodes has [27].

1: **if** $(seq\_num_i^d < seq\_num_j^d)$ **or** $((seq\_num_i^d = seq\_num_j^d)$ **and** $(hop\_count_i^d > hop\_count_j^d))$ **then**
2: $seq\_num_i^d := seq\_num_j^d$;
3: $hop\_count_i^d := hop\_count_j^d + 1$;
4: $next\_hop_i^d := j$;
5: **end if**

Figure 3.3: AODV route update rules [27]

For each valid route maintained by a node as a routing table entry, the node also maintains a list of precursors that may be forwarding packets on this route. These precursors will receive notifications from the node in the event of detection of the loss of the next hop link. The list of precursors in a routing table entry contains those neighboring nodes to which a route reply was generated or forwarded [25]. Whenever a node wants to send data to a given destination node and does not have any route to that node in its route table, it initiates a process called a route discovery procedure which is discussed in the following subtopic.

3.2.1.2. ROUTE DISCOVERY PROCEDURE IN AODV

Route discovery [25, 26, 28] begins when a source node needs a route to a destination and does not have one available in its routing table. It first places the destination IP address and last known sequence number for that destination, as well as its own IP address and current sequence number, into a Route Request (RREQ) message. The RREQ ID field is incremented by one from the last RREQ ID used by the current node. The Hop Count field is set to zero. Before broadcasting the RREQ, the originating node buffers the RREQ ID and the Originator IP address (its own address) of the RREQ for PATH_DISCOVERY_TIME. In this way, when the node receives the packet again from its neighbors, it will not reprocess and re-forward the packet. A node should not originate more than RREQ_RATELIMIT RREQ messages per second, which in this case is set to 10. After broadcasting a RREQ, a node waits for a RREP. If a reply is not received within NET_TRAVERSAL_TIME milliseconds, the node may try again to discover a route by broadcasting another RREQ, up to a maximum of RREQ_RETRIES times at the maximum TTL value. Each new attempt must increment and update the RREQ ID.

When a node receives the RREQ, it first determines whether it has received a RREQ with the same Originator IP Address and RREQ ID within at least the last PATH_DISCOVERY_TIME. If such a RREQ has been received, the node silently discards the newly received RREQ. If not, it creates a reverse route entry for the source node in its route table. It then checks whether it has a fresh enough route to the destination node. In order to respond to the RREQ, the node must either be the destination itself, or intermediate node with an unexpired route to the destination whose corresponding sequence number is at least as great as that contained in the RREQ. If neither of these conditions is met, the node rebroadcasts the RREQ to its respective neighbors by incrementing the hop count value in the RREQ by one, to account for the new hop through the intermediate node.

Either the destination itself or an intermediate node with a fresh enough route will respond for the route request message using a Route Reply (RREP) message. Once created, the RREP is unicast to the next hop toward the originator of the RREQ as indicated by the route table entry for that originator. As the RREP is forwarded back towards the node which originated the RREQ message, the Hop Count field is incremented by one at each hop. Thus, when the RREP reaches the originator, the Hop Count represents the distance, in hops, of the destination from the originator. When an intermediate node receives the RREP, it creates a forward route entry for the destination node in its route table, and then forwards the RREP to the source node. If the current node is not the node indicated by the Originator IP Address in the RREP message and a forward route has been created or updated, the node consults its route table entry for the originating node to determine the next hop for the RREP packet, and then forwards the RREP towards the originator using the information in that route table entry. Once the source node receives the RREP, it can begin using the route to transmit data packets to the destination. If it later receives a RREP with a greater destination sequence number or equivalent sequence number and smaller hop count, it updates its route table entry and begins using the new route.

The route discovery procedure can be explained diagrammatically as shown in figure 3.4. The diagram is taken from [63] with modification to include the reverse and forward routes setup. If node *A* wants to send data to a destination *G* and does not have valid route to *G*, it floods RREQ messages (lines seen in solid red color) to its neighbors (*B* and *C*). The intermediate nodes then cache the received message and flood the request to their respective neighbors if they don't have fresh enough route to *G*. while receiving requests, the intermediate nodes save the route back to the originator that

will be used to forward reply (lines seen in dotted black color). When the RREQ reaches *G*, node *G* prepares a RREP (lines seen in dotted green color) and this reply is unicasted to the originator using the partial route established during the propagation of RREQ messages. The intermediate nodes then forward the RREP to the originator by adding forward path (lines seen in solid blue color) to their table. When node *A* receives a reply, it immediately starts to forward data to *G* using the established route.

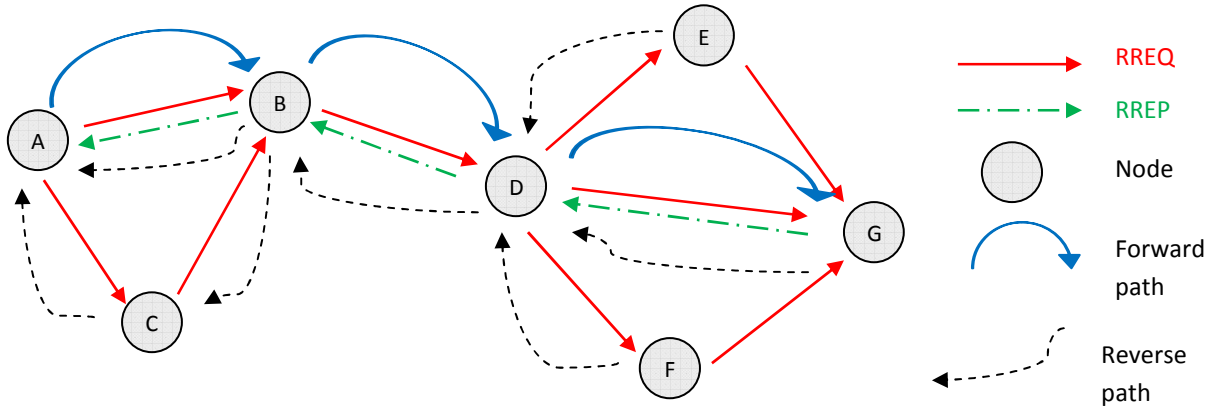

Figure 3.4: Route discovery procedure in AODV

3.2.1.3. ROUTE MAINTENANCE IN AODV

On-Demand protocols also employ a route maintenance procedure [25, 26, 28], where nodes monitor the operation of the route and inform the sender of any routing error. A node may offer connectivity information by broadcasting local Hello messages. A node should only use hello messages if it is part of an active route. Every HELLO_INTERVAL milliseconds, the node checks whether it has sent a broadcast (e.g., a RREQ or an appropriate layer 2 message) within the last HELLO_INTERVAL. If it has not, it may broadcast a RREP with TTL = 1, called a Hello message. Local Hello messages are used to determine local connectivity, which can reduce response time to routing requests and can trigger updates when necessary.

Link breaks in non-active links do not trigger any protocol action. However, when a link break in an active route occurs, the node upstream of the break determines whether any of its neighbors use that link to reach the destination. If so, it creates a Route Error (RERR) packet and the RERR contains the IP address of each destination that is now unreachable due to the link break. The RERR also contains the sequence number of each such destination, incremented by one. The node then broadcasts the packet and invalidates those routes in its route table. When a neighboring node receives the RERR, it

in turn invalidates each of the routes listed in the packet, if that route used the source of the RERR as a next hop. If one or more routes are deleted, the node then goes through the same process, whereby it checks whether any of its neighbors route through it to reach the destinations. If so, it creates and broadcasts its own RERR message.

A Route Error (RERR) message may be either broadcast (if there are many precursors), unicast (if there is only 1 precursor), or iteratively unicast to all precursors (if broadcast is inappropriate).

Route Error and link breakage processing requires the following steps:

- Invalidating existing routes
- Listing affected destinations
- Determining which, if any, neighbors may be affected
- Delivering an appropriate RERR to such neighbors

Table 3-1: AODV message types

No	Message type	Purpose	Used in stage
1	RREQ	Used to find routes and is initiated by a source node	Route discovery
2	RREP	Response to RREQ message	Route discovery
3	RERR	Notifies link failures	Route maintenance
4	HELLO	Provides connectivity information	Local connectivity, may also trigger route update

Once a source node receives the RERR, it invalidates the listed routes as described. If it determines it still needs any of the invalidated routes, it reinitiates route discovery for that route. The purpose of the AODV messages namely RREQ, RREQ, RERR and HELLO messages is summarized in table 3-1.

The flow chart shown in figure 3.5, a simplified version, summarizes the action of an AODV protocol when a node is processing an incoming AODV message. Both RREQ and RREP are processed during route discovery procedure and RRER is generated and processed when an active link fails.

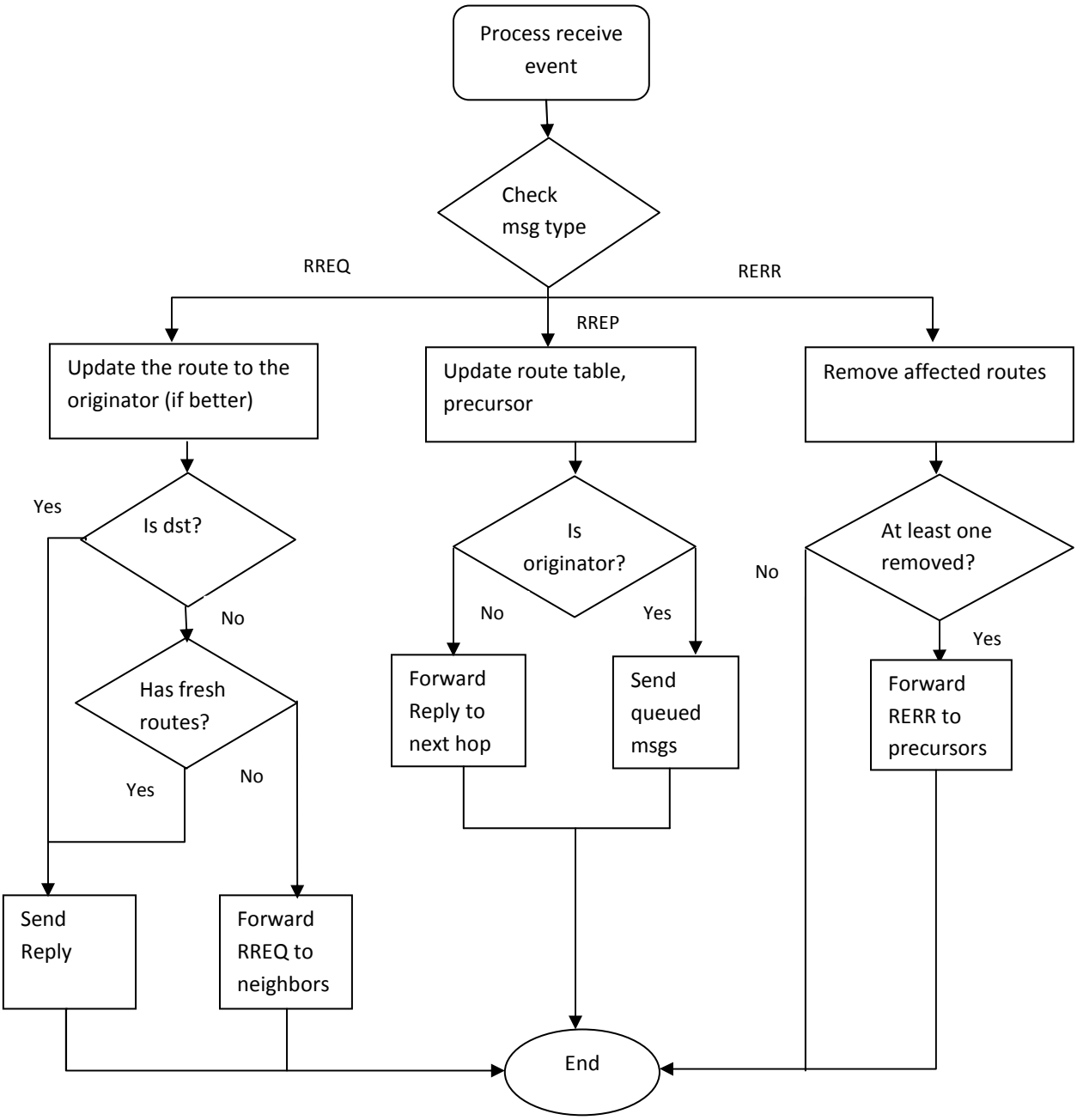

Figure 3.5: Processing an incoming message in AODV [29]

3.2.2. AODV SECURITY

Unfortunately, majority of the routing protocols in mobile ad hoc networks are witnessed for declined performance at the time of communicating with large scale of misbehaving nodes, which definitely disrupts the course of data rendering the routing protocol to resume again the route exploration

procedure or to choose an unconventional route in case it is available. Moreover the newly opted route has the feasibility of possessing a few malicious nodes, resulting in failure of new route too. The fundamental issue with frequently used routing protocols is that they rely on all mobile nodes of network and based on the assumption that nodes will behave or cooperate properly but there might be a condition where some nodes are not behaving properly. Majority of the adhoc network routing protocols become inefficient and show reduced performance while mitigating with big number of misbehaving nodes [10].

AODV is designed for use in networks where the nodes can all trust each other, either by use of preconfigured keys, or because it is known that there are no malicious intruder nodes. AODV has been designed to reduce the dissemination of control traffic and eliminate overhead on data traffic, in order to improve scalability and performance [25].

In [30], the authors extend the AODV routing protocol, Secured AODV (SAODV), to guarantee security based on the approach of key management scheme in which each node must have certificated public keys of all nodes in the network. This work uses two mechanisms to secure the AODV messages: digital signature to authenticate the fixed fields of the messages and hash chains to secure the hop count field. This protocol uses public key distribution approach in the ad hoc network; therefore, it is difficult to deploy and computationally heavy since it requires both asymmetric cryptography and hash chains in exchanging messages [31].

CHAPTER FOUR

MANET SECURITY FLAWS

MANETs are susceptible to many security issues due to their natural characteristics and properties. MANET operations are performed by the cooperation of nodes working as both hosts and routers, which do not have any knowledge about the trustworthiness of other nodes [32].

The ad hoc networks' flexibility and convenience come at a price. Ad hoc wireless networks inherit the traditional problems of wireless communications and wireless networking like the wireless medium has neither absolute, nor readily observable boundaries outside of which stations are known to be unable to receive network frames; the channel is unprotected from outside signals; and the wireless medium is significantly less reliable than wired media [6].

Expensive and cumbersome security mechanisms can delay or prevent exchanges of routing information, leading to reduced routing effectiveness, and may consume excessive network or node resources leading to many new opportunities for possible Denial-of-Service (DoS) attacks through the routing protocol [33]. Surveys of solutions to different security issues of MANETs have been reviewed in [23, 35-38].

4.1. VULNERABILITY OF MANETS

The following features make MANETs more vulnerable to different attacks than traditional networks [34, 35, 39 and 40].

Lack of centralized management: MANETs do not have a centralized monitor server. The absence of management makes the detection of attacks difficult because it is not easy to monitor the traffic in a highly dynamic and large scale adhoc network. Lack of centralized management will impede trust management for nodes [40].

Dynamic topology: Mobile nodes are generally autonomous units that are capable of roaming independently. This means that tracking down a particular mobile node in a large-scale ad hoc network cannot be done easily. Node mobility and wireless connectivity allow nodes to enter and

leave the network spontaneously to form and break links unintentionally. Therefore, the network topology has no fixed form regarding both its size and shape [35].

Cooperativeness: Routing algorithm for MANETs usually assume that nodes are cooperative and non-malicious. As a result a malicious attacker can easily become an important routing agent and disrupt network operation by disobeying the protocol specifications [40].

Power limitation: Ad hoc enabled mobile hosts are small and lightweight, and they are often supplied with limited power resources, such as small batteries. This limitation causes vulnerability for attackers targeting to consume some nodes' batteries. This feature also represents a challenging constraint when designing security solutions for MANETs [35].

Multi-hop: Because of the lack of central routers and gateways, hosts are themselves routers. Thus, packets follow multihop routes and pass through different mobile nodes before arriving at their final destination. Due to the possible untrustworthiness of such nodes, this feature presents a serious vulnerability [35].

Insecure operational environment: the operational environment in which MANET's are generally used may not be always securing, for example, a battle field. In such environment, nodes may be moving in and out of hostile and insecure enemy territory, where they would be highly vulnerable to security attacks [39].

Memory and computation power limitation: Ad hoc enabled mobile nodes have limited storage devices and weak computational capabilities. Consequently, high complexity security solutions, such as symmetric or asymmetric data encryption, are difficult to implement [35].

4.2. TYPES OF ATTACKS

Attacks include any action that intentionally aims to cause any damage to the network. [35, 41, 42] classify attacks into different categories. Attacks can be divided according to their origin or their nature. An origin-based classification splits attacks into two categories, external and internal, whereas a nature-based classification splits them into passive attacks and active attacks. The classification is summarized in table 4-1.

Table 4-1: Classification of attacks

No	Classification of attacks	
	Nature-based	**Origin-based**
1	Passive attacks	Internal attacks
2	Active attacks	External attacks

External attacks include attacks launched by a node that does not belong to the logical network, or is not allowed to access to it where as internal attacks include attacks launched by an internal compromised or malicious node [35].

i) Passive Attacks

A passive attack [35, 41, 42] does not disrupt the normal operation of the network; the attacker snoops the data exchanged in the network without altering it. Here the requirement of confidentiality gets violated. For that, the attacker analyzes the packets to pick up required information. Due to the nature of the wireless communication medium which is widely shared, it is easier for an attacker to launch such an attack in this environment than in traditional wired environments. The security attribute that must be provided here is information confidentiality. Attacks of such type include Eavesdropping and Traffic Analysis & Monitoring. During eavesdropping, the information the attacker may target include the location, public key, private key or even passwords of the nodes. By traffic analysis and monitoring, packet transmission is analyzed to infer important information such as a source, destination, and source-destination pair. The information collected could be used to initiate other active attacks.

ii) Active Attacks

An active attack attempts to alter or destroy the data being exchanged in the network there by disrupting the normal functioning of the network which are more severe and significantly affect the performance of the network. This type of attacks can be done from MAC layer, network layer, transport and application layers. There are many attacks in this category. Some of the active attacks surveyed in [9, 23, 33, 39-45] are discussed as follows.

- *Jamming Attack**:*** In this form of attack, the attacker initially keeps monitoring the wireless medium in order to determine the frequency at which the destination node is receiving signals from the sender. It then transmits signals on that frequency so that error free reception at the receiver is hindered [45].

- *Blackhole attack**:*** In a black hole attack, a malicious node sends fake routing information, claiming that it has an optimum route and causes other good nodes to route data packets through the malicious one. For example, in AODV, the attacker can send a fake RREP (including a fake destination sequence number that is fabricated to be equal or higher than the one contained in the RREQ) to the source node, claiming that it has a sufficiently fresh route to the destination node. This causes the source node to select the route that passes through the attacker. Therefore, all traffic will be routed through the attacker, and therefore, the attacker can misuse or discard the traffic [23].

- *Modification:* In a message modification attack, adversaries make some changes to the routing messages, and thus endanger the integrity of the packets in the networks. Since nodes in the ad hoc networks are free to move and self-organize, relationships among nodes at some times might include the malicious nodes [41].

- *Denial of Service attack:* In a denial-of-service (DoS) attack, an attacker attempts to prevent legitimate users from accessing information or services. A denial of service (DoS) attack is an attack that clogs up so much memory on the target system that it cannot serve its users, or it causes the target system to crash, reboot, or otherwise deny services to legitimate users [44].

- *Flooding attack:* In flooding attack, attacker exhausts the network resources, such as bandwidth and a node's resources, like computational and battery power or to disrupt the routing operation to cause severe degradation in network performance. For example, in AODV protocol, a malicious node can send a large number of RREQs in a short period to a destination node that does not exist in the network. Because no one will reply to the RREQs, these RREQs will flood the whole network. As a result, all of the node battery power, as well as network bandwidth will be consumed and could lead to denial-of-service attack [41].

- *Man-in-the-middle:* An attacker sits between the sender and receiver and sniffs any information being sent between two nodes. In some cases, attacker may impersonate the sender to communicate with receiver or impersonate the receiver to reply to the sender [39].

- *Impersonation or Spoofing attack:* Spoofing is a special case of integrity attacks whereby a compromised node impersonates a legitimate one due to the lack of authentication in the current ad hoc routing protocols. The main result of the spoofing attack is the misrepresentation of the network topology that may cause network loops or partitioning. Lack of integrity and authentication in routing protocols creates fabrication attacks that result in erroneous and bogus routing messages [44].

- *Gray hole attack:* The gray hole attack has two phases. In the first phase, a malicious node exploits the AODV protocol to advertise itself as having a valid route to a destination node. In the second phase, the node drops the intercepted packets with a certain probability. This attack is more difficult to detect than the black hole attack. A gray hole may exhibit its malicious behavior in different ways. It may drop packets coming from (or destined to) certain specific node(s) in the network while forwarding all the packets for other nodes. Another type of gray hole node may behave maliciously for some time duration by dropping packets but may switch to normal behavior later. [41].

- *Packet dropping attacks:* Direct interruption to the routing messages could be done by using the packet dropping attacks. In a standard packet dropping attack, an adversary collaborates as usual in the route discovery process and launches the constant packet dropping attacks if it is included as one of the intermediate nodes. In addition, instead of constantly dropping all the packets, adversaries might vary their techniques using random, selective, or periodic packet dropping attacks to help their interrupting behavior remain concealed [41].

- *Selfish Nodes:* In this, a node is not serving as a relay to other nodes which are participating in the network. This malicious node which is not participating in network operations, uses the network for its advantage to save its own resources such as power [40].

4.3. FLOODING ATTACKS IN AODV

The Ad Hoc Flooding Attack, described in [11], results in denial of service when used against all on on-demand ad hoc networks routing protocols, including protocols that were designed to be secure. In this attack, the attacker either broadcasts a lot of route request packets for node ID which is not in the network or sends a lot of attacking data packets to exhaust the communication bandwidth and nodes' resource so that the valid communication cannot be kept. The attack becomes significant when the fake RREQ packet or the useless data is sent frequently.

On-demand routing protocols use the route discovery process to obtain the route between two nodes. In the route discovery, the source node floods the network by sending the RREQ packets. Because the priority of the RREQ control packet is higher than data packet, then at the high load also RREQ packets are transmitted. A malicious node exploits this feature of on demand routing to launch the attack [31].

4.3.1. TYPES OF FLOODING ATTACKS

Ad-hoc Flooding Attack can be divided into Route Request (RREQ) Flooding Attack and Data Flooding Attack [17, 31 and 46]. Another similar classification in [22] divides flooding attack in to control packet flooding attack and data packet flooding attack. The control packet flooding can include flooding RREQ or other control packets like HELLO packets. In this thesis work, the first classification is discussed in detail.

i) RREQ Flooding Attack

In AODV, if a node wants to send data packets to a destination that is not in its routing table, it will buffer the data packets and broadcast a Route Request (RREQ) into the network. The RREQ packet will be forwarded by other intermediate nodes to the intended destination node. The destination node, upon receiving the RREQ, will send a Route Reply (RREP) on the reverse route back to the source node. If no RREP is received after a fixed number of attempts, the data packets from the buffer will be dropped. If more data packets are received from the application layer, a new route discovery process will be initiated [14].

The RREQ Flooding Attack [46] is a denial-of-service attack in which malicious nodes take advantage of the route discovery process of the reactive routing protocols (e.g. AODV, DSR) in

MANET. In this attack, a compromised node aims to flood the network with a large number of RREQs to non-existent destinations in the network. Since a node with such invalid destination node-id does not exist in the network, a reply packet cannot be generated by any node in the network and they keep on flooding the RREQ packet. When such fake RREQ packets are broadcasted into the network in high numbers, the network gets saturated with RREQs and is unable to transmit data packets. Thus, it leads to congestion in the network. The RREQ Flooding Attack also results in overflow of route table in the intermediate nodes so that the nodes cannot receive new RREQ packet, resulting in a denial- of-service attack. Moreover, unnecessarily forwarding these fake route request packets cause wastage of precious node resources such as energy and bandwidth.

This type of attack is hard to detect since any normal node with frequently broken routes could legitimately initiate frequent route discoveries. One or more malicious nodes flooding the MANET with RREQ control packets related to bogus route discoveries can cause a sharp drop in network throughout [15]**.**

ii) Data Flooding Attack

In the data flooding attack, malicious node floods the network by sending useless data packets. To launch the data flooding, first the malicious node builds a path to all the nodes then sends the large amount of bogus data packets. These useless data packets exhaust the network resources and hence legitimated user can not able to use the resources for valid communication [17].

4.3.2. Effects of Flooding Attacks

Some of the effects of flooding attack are listed in [46] as follows.

- Wastage of bandwidth
- Wastage of nodes' processing time, thus increasing the overhead
- Overflow of the routing table entries, causing exhaustion of an important network resource like memory
- Exhaustion of the nodes' battery power
- Degraded throughput

Due to flooding attack, a genuine node cannot fairly serve other nodes due to the network-load imposed by the fake RREQs and useless data packets. Most of the network resources are wasted in trying to generate routes to destinations that do not exist or routes that are not going to be used for any communication [46].

Since control packets are given higher priority over data packets in transmitting, then at high loads, the wireless channel usage can be completely dominated by the control packets used for route discoveries. In this situation, valid communication can't be kept and normal network nodes cannot be served, then it leads to a kind of denial-of-service attack [15]. Different literatures like [11, 12, 19, 46 and 47] have shown that the presence of malicious flooding nodes in MANET can affect the performance of the overall wireless network.

[46], for example, showed that due to the extensive flooding in the network, average percentage of packet loss, average routing overhead and average bandwidth requirement− all increase, thus decreasing the overall network throughput. In [19], the effect of flooding attack lowered the PDR below 70% even when there is only one malicious node generating 5 RREQ per second with 50 nodes MANET network. It is also shown in [11] that for 50 wireless nodes the PDR has sharply dropped to around 1.2%, from 96% in the absence of attack, with one malicious node sending 50 RREQs/s. [12] in its simulation also claimed a significant drop in throughput by one malicious node generating 1 RREQ packet/s for scenario of 100 nodes. Hence, the presence of malicious flooding nodes in MANET can affect the performance of the overall wireless network and can act as one of the major security threats. This effect could be more severe if many attackers combine both RREQ and data flooding attacks which may completely clog a given network.

CHAPTER FIVE

PREVENTION OF FLOODING ATTACK

The proposed prevention mechanism is discussed in detail in this section. Before that, the network simulator and the implementation of AODV in NS-2 are introduced.

5.1. NETWORK SIMULATOR 2 (NS2)

As surveyed in [48], there are many network simulators with different features and NS-2 is one of the most popular open source network simulators.

Network Simulator (Version 2) [49, 50], widely known as NS2, is simply an event-driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community.

NS2 is a discrete-event simulator, where actions are associated with events rather than time. An event in a discrete-event simulator consists of execution time, a set of actions, and a reference to the next event. These events connect to each other and form a chain of events on the simulation timeline. Unlike a time-driven simulator, in an event-driven simulator, time between a pair of events does not need to be constant.

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e.,a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend). NS2 uses OTcl to create and configure a network, and uses C++ to run simulation. All C++ codes need to be compiled and linked to create an executable file. Since the body of NS2 is fairly large, the compilation time is not negligible. OTcl, on the other hand, is an interpreter, not a compiler. Any change in a OTcl file does not need compilation. Nevertheless, since OTcl does not convert all the codes into machine language, each line needs more execution time. In summary, C++ is fast to run but slow to change. OTcl, on the other hand, is slow to run but fast to change. It is therefore suitable

to run a small simulation over several repetitions (each may have different parameters). NS2 is constructed by combining the advantages of these two languages.

NS2 provides users with an executable command ***ns*** which takes on input argument, the name of a Tcl simulation scripting file. In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used. To analyze a particular behavior of the network, users can extract a relevant subset of text-based data and transform it to a more conceivable presentation. The NS-2 simulation flow is shown in figure 5.1.

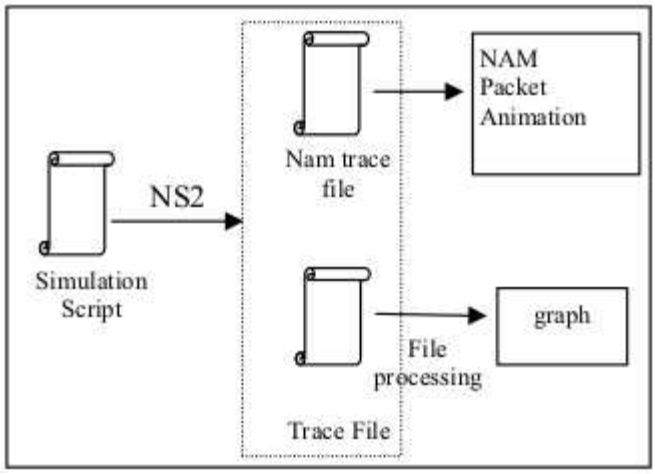

Figure 5.1: NS-2 simulation process flow [51]

NS2 Wireless Packet Trace Format is just an extension from the "Normal Packet Trace Format". The trace format can be old trace format or new trace format. The new wireless trace format is described in [52] and the new trace format is used for this particular simulation study.

5.1.1. INSTALLATION OF NS2

NS2 is installed in Linux Ubuntu 12.04 using the installation procedure in [53]. The installation of NS2 is described in the following few steps:

Step 1: Install some software required by NS2.

```
sudo apt-get install tcl8.5-dev tk8.5-dev
sudo apt-get install build-essential autoconf automake
```

```
sudo apt-get install perl xgraph libxt-dev libx11-dev libxmu-dev
```

Step 2: Download the source file for NS2, version 2.35. The following link can used to download NS2.

http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.35/ns-allinone-2.35.tar.gz/download. And the downloaded file is named as *ns-allinone-2.35.tar.gz.*

Step 3: Extract the file. In my case I have extracted the file to my home directory as follows.

```
tar -zxvf ns-allinone-2.35.tar.gz -C /home/tek
```

Step 4: Modify the makefile file. This is to add gcc-4.5 to this version of NS2 because gcc-4.5 is the default gcc compiler for the current Ubuntu version.

Step 5: Install NS2. For installation, do the following.

```
cd /home/tek/ns-allinone-2.35
```

```
sudo ./install
```

Step 6: Add the path setting of NS2. Edit the */home/tek/.bashrc* file and add the following paths at the end of the file.

```
Export PATH=$PATH:/home/tek/ns-allinone-2.35/bin:/home/tek/ns-allinone-2.35/tcl8.5.10/unix:/home/tek/ns-allinone-2.35/tk8.5.10/unix
```

```
Export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/tek/ns-allinone-2.35/otcl-1.14:/home/tek/ns-allinone-2.35/lib
```

```
Export TCL_LIBRARY=$TCL_LIBRARY:/home/tek/ns-allinone-2.35/tcl8.5.10/library
```

And then enable the path setting by running the `source .bashrc` command from the home directory.

Step 7: Verifying

Check this by running `which ns` command and prints out the path of “ns” command. If it prints out */home/tek/ns-allinone-2.35/bin/ns*, then it works.

5.1.2. OVERVIEW OF AODV IMPLEMENTATION IN NS2

NS2 implements, as described in [54], a routing protocol using routing agents which are used to create, transmit, receive, and destroy routing packets. NS2 declares a C++ class *AODV* and class *AODV* derives from class *Agent* and inherits three important attributes and behaviors: (1): a pointer "*target_*" which points to a link layer object, (2) a function *allocpkt()* which can be used to create packets and (3) a packet reception function *recv(p,h*). Among these attributes and behaviors, only packet reception is overridden by class AODV.

Main C++ files for AODV are stored in the directory of *~ns/aodv/* and include the following.

- *Aodv.cc and aodv.h*: These files include the main definitions of AODV routing agents.
- *Aodv_packet*.h: this file defines the packet header of AODV protocol including RREQ, RREP and RERR messages.
- *Aodv_rtable*.cc *and aodv_rtbale.h*: these files handle the routing entries and tables of the AODV routing protocol.
- *Aodv_rqueue*.cc *and aodv_rqueue.h*: define the buffer which stores data packets during a routing discovery procedure.

Other AODV related classes include Agent, Timers and routing information. The *Agent* class is responsible for creating, sending, receiving, processing, and destroying routing packets and AODV uses class *AODV* for this purpose. The timers take care of the time-driven actions and include classes like *BroadcastTimer, HelloTimer, NeighbourTimer* and *RoutecacheTimer*. Routing information is handled using two classes: *aodv_rt_entry* and *aodv_rt_table* classes.

For the purpose of route discovery, AODV uses *SendRequest (nsaddr_t dst), SendReply (nsaddr t ipdst,…), recvRequest (Packet *p)* and *recvReply (Packet *p)* functions to send RREQ, send RREP, receive RREQ and receive RREP respectively. In the case of the send functions, the RREQ or RREP is first build and the respective fields are filled with the right values before they are sent to neighbors. The implementation of AODV routing protocol in NS-2 is also briefly described in [55].

5.2. MODELING OF FLOODING ATTACK

In this case the attacker selects a destination that does not exist in the network. Then it builds the RREQ message and simply floods the request without even checking the routing table for a route. If the attacker is an internal one, it embeds the right key in the reserved bits like the genuine nodes, but for external attacker the default value of the reserved bits is maintained. After receiving the RREQ from the attacker, the intermediate nodes try to look the route for the destination in their routing table and finally flood the request to their respective neighbors as none of the nodes has a route to the destination. The attacker waits for some interval, the interval which controls the rate of flooding, and starts the process again. This flooding interval is varied during the simulation. The process of RREQ flooding attack can be summarized as in figure 5.2.

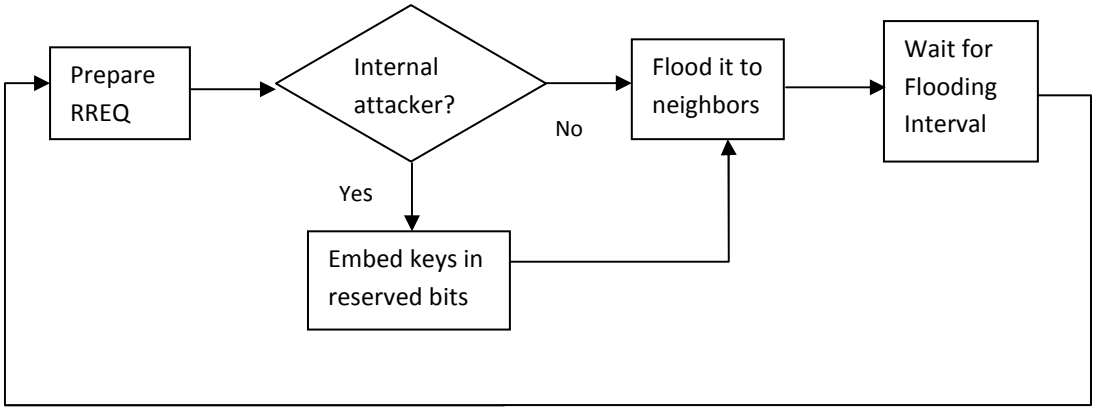

Figure 5.2: Modeling RREQ flooding attack

If the attack is DATA Flooding Attack, the attacker simply sends excessive useless data with varying packet rate to a destination there by making other intermediate nodes busy in transmitting that useless data to overuse the available resources like bandwidth, computational power and battery life.

5.3. DESIGN AND IMPLEMENTATION OF THE PREVENTION MECHANISM

The RREQ flooding attack occurs at the route discovery stage. So in this case the existing protocol is modified to include the proposed scheme. The route request sending and receiving procedures are modified and are described as follows.

5.3.1. SENDING ROUTE REQUEST

The simplified steps of sending request process is shown in figure 5.3. The flowchart in route discovery of [56] is slightly modified before it is incorporated in to this work. When a source node S wants to send data to destination D, it first checks in its routing table if a route to D is available. If there is an active route, there is no need for route discovery it simply forwards its data to D using the information in the routing table. If there is no any route to D, then S initiates route discovery by first caching the message in its queue and prepares the RREQ. It fills the fields like the source address, destination address, hop count, sequence numbers and others. Among other things the reserved bits are sent as zero and ignored at reception by default.

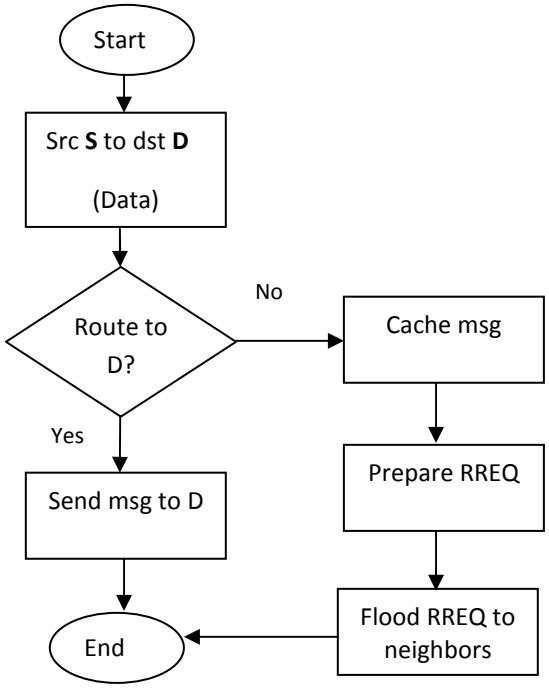

Figure 5.3: Sending route request

For the prevention scheme, the RREQ sending procedure is modified by embedding some information in the reserved bits to identify the attacker. We can simply embed the key in the reserved bits and confirm up on reception. But if the attacker is aware of the embedded key, it can learn it. To prevent this, two keys (key1 and key2) and two operations (XOR and modulo) are used. Key1 is used to flip one bit of the address of the sending node. For example, if key1=10, then the 10th bit of the address is flipped. Then this value is divided by key2. Then the remainder is embedded in the reserved bits as shown in the marked fields in figure 5.4. By doing so, even if the attacker is aware of the presence of

embedded information, unless it has the two keys, it cannot guess the value easily. So every legitimate node sending RREQ embeds some information that identifies itself.

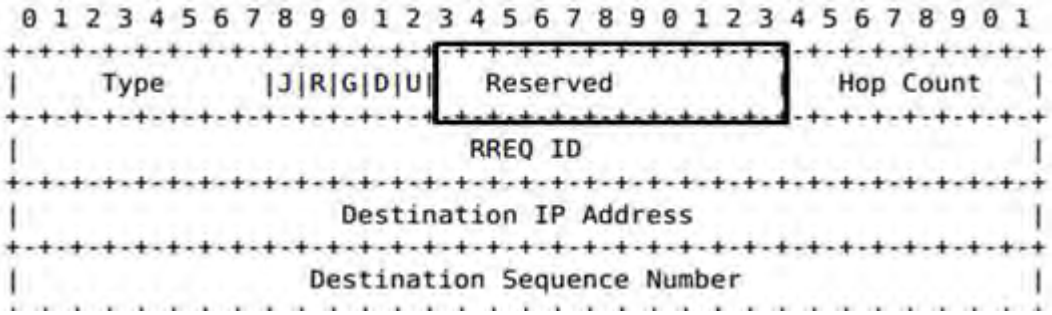

Figure 5.4: Reserved bits of RREQ message

5.3.2. RECEIVING ROUTE REQUEST

When an intermediate node receives route request, it checks for the freshness of the information and caches the request and sets up the reverse path back to the originator. if the receiving node is an intermediate node with fresh route to the destination or if the receiving node is the destination itself, then it sends the reply to the originator of the RREQ message. Otherwise the request is forwarded to its neighbors. This is the normal operation of the protocol and the simplified process is shown in figure 5.5 digramatically which is extracted from figure 3.5.

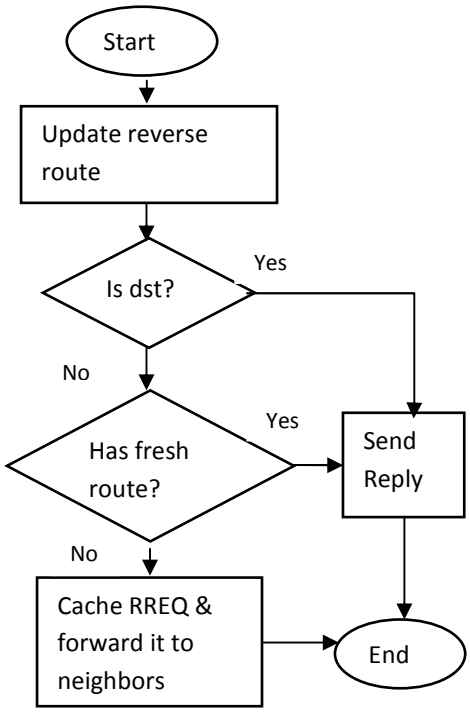

Figure 5.5: Receiving route request

The normal procedure of receivng RREQ is modified to prevent the flooding attacks. As a prevention mechanism, one of the existing solutions, the Filter method [15], is incorporated into the proposed solution in this work. Using the reserved bits, external attackers can be prevented form performing flooding attacks. To prevent internal attackers, attackers that know the keys, the filter method is used.

In Filtering method, each node maintains two threshold values. The threshold values are the criterias for each node's decision of how to react to a RREQ message. The RATE_LIMIT parameter denotes the number of RREQs that can be accepted and processed as normal per unit time by a node. The BLACKLIST_LIMIT parameter is used to specify a value that aids in determining whether a node is acting malicious or not. If the number of RREQs originated by a node per unit time exceeds the value of BLACKLIST_LIMIT, then the correponding node is regarded as an attacker and will be blacklisted. This will prevent further flooding of the fake RREQs in the network. If the rate of RREQs originated by a node is between the RATE_LIMITand the BLACKLIST_LIMIT, the RREQ packet is added to a "delay queue" waiting to be processed where malicious node that has a high attack rate will thus be severely delayed. If the rate of this RREQ originator is below the RATE_LIMIT, the RREQ packet is processed as normal. Hence, internal attackers will be prevented using this method from sending fake RREQ to flood a given network though it is also possible to use other existing solutions instead of the filtering method.

The algorithm of the prevention mechanism works as follows. Upon reception, the receiving node extracts the source address from the RREQ message and if the source address of the sender is in the black list, then its request is dropped. If not, then the values of the reserved bits is checked for the right information. The source address of the originator is processed using the two keys and compared with the value stored in the reserved bits of the incoming RREQ message. If they don't much, the sending node is an attacker, external, then it is added to the black list and its request is dropped. If equal, the rate of the RREQ is compared with the RATE_LIMIT. If the rate is below the RATE_LIMIT, it is not an attcker and its request is processed as normal. If the rate is is greater than the BLACKLIST_LIMIT, the node is labeled as an attacker by adding its address to the black list and its RREQ message is dropped. If the rate of route request is between the two thresholds, then the RREQ messages are delayed by adding them to the delay queue. This procedure is done by a neighboring node for every reception of RREQ. The complete prevntion algorith is shown in figure 5.6.

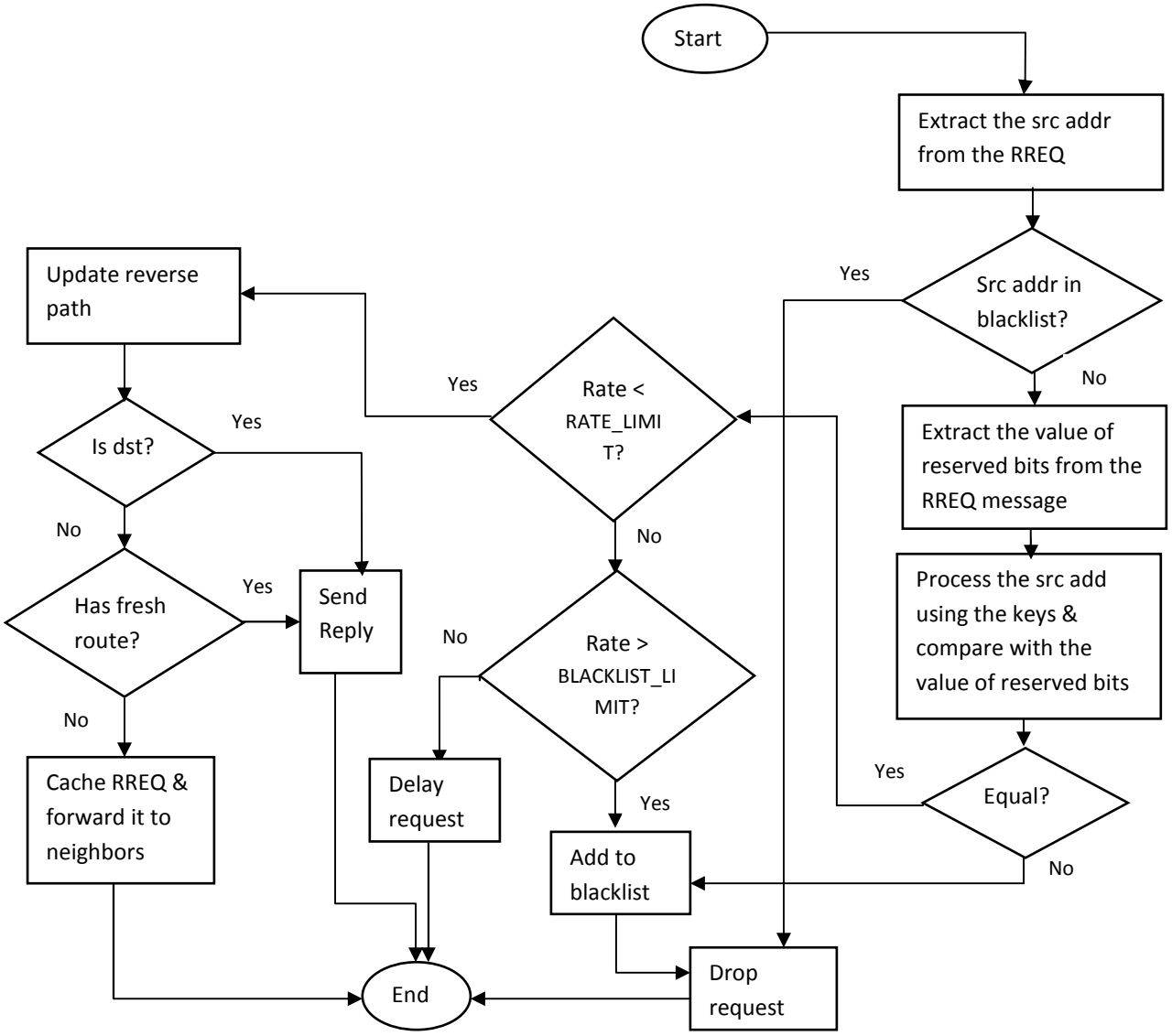

Figure 5.6: Modified RREQ receiving procedure to prevent flooding attack

5.3.3. IMPLEMENTATION

To implement this prevention, the algorithms of the flooding attack and the prevention mechanism are incorporated to the existing AODV routing protocol by modifying the C++ code of the protocol. In implementing the flooding attack, *aodv.cc* and *aodv.h* files are modified to include the malicious behavior to the protocol. Class *FloodTimer* is used to implement a timer that will be used to schedule the flooding attack. Then if the respective node is an attacker, internal or external, then a function (*RREQ_flood()*) is run every some interval (or *flooding_interval*) as scheduled by the timer. If the

attacker is internal (knows the keys) or the node is genuine, then it has to manipulate its own address using the two keys and embed the value in the reserved bits before flooding any RREQ to its neighbors. Some of the modifications (C++ codes) made to implement the flooding attack are attached in appendix D.

To implement the filtering method, a source code in [57] is used with some modification. In implementing the new solution (including the filtering method), *aodv.cc, aodv.h* and *aodv_rtable.h* are modified. In *apdv_rtable.h* two classes, *RREQ_count* and *Blacklist*, are defined. The *RREQ_count* class is used to hold the number of RREQs a particular node sends per second which is used by the filter method. The *Blacklis*t class is used to hold the nodes that are identified as attackers by violating the limitation and it is used in holding internal or external attackers that are blacklisted by the prevention mechanism. The member variables and methods used by these classes are defined in *aodv.h* file. While sending RREQs, a node uses the reserved bits to embed the manipulated values and at the same time it calculates the rate of RREQs sent by a given neighbor for internal nodes. Upon reception, on the other hand, the values of the reserved bits are checked and the rates of RREQs are compared for internal attackers according to the prevention shown in figure 5.6.

Basically, the SendRequest and recvRequest functions of AODV, which are found in *aodv.cc* file, are modified to implement the prevention mechanism. Other supplementary functions are also used. The C++ codes of this prevention mechanism are shown in appendix E. In addition to this, Sample TCL code, mobility file, traffic scenario file and AWK scripts are shown in appendix A, B, C and F respectively.

CHAPTER SIX

SIMULATION

6.1. SIMULATION EXPERIMENT SETUP

To study the feasibility of the prevention mechanism, network simulator 2 (NS2) [49, 50] is used to conduct series of experiments to evaluate its effectiveness on preventing flooding attack in AODV. As explained in [54], the process of creating Mobile Nodes consists of Mobile Node Configuration (for specifying configuration information) and Mobile Node Construction (to create the mobile nodes) steps. Each node is assumed to be equipped with a wireless transceiver operating on 802.11 wireless standards. The physical radio frequency characteristics of each wireless transceiver are with a bit rate of 2Mb/sec and a transmission range of 250 meters. Moreover, Each Mobile Node has a buffer which can hold up to 50 packets. The service discipline is based on prioritized queue which gives priority to routing packets. The type of antenna used is also omni-directional.

The Random Waypoint Model [58] is used as a mobility model. A node in this model randomly chooses a destination point ('waypoint') in the area and moves with constant speed on a straight line to this point. After waiting a certain pause time, it chooses a new destination and speed, moves with constant speed to this destination, and so on. The destination points are uniformly randomly distributed on the system area. NS2 is shipped with a script, `setdest`, to generate mobility file based on this model. For generating traffic scenario, Constant Bit Rate (CBR) traffic sources are used for generating data packets. The source-destination pairs are spread randomly over the network. To generate the traffic file randomly, NS2's `cbrgen.tcl` file is used.

The energy consumption model used is described in [59]. A wireless network interface can be in one of the following four states: Transmit, Receive, Idle or Sleep states. Each state represents a different level of energy consumption. Transmit is a state when the node is transmitting a frame with transmission power and Receive is when node is receiving a frame with reception power. Idle (listening) is when no messages are being transmitted over the medium; the nodes stay idle and keep listening the medium. Whereas Sleep state is when the radio is turned off and the node is not capable

of detecting signals. The energy dissipated in transmitting or receiving one packet can be calculated as multiplying the transmit power or the receive power by duration respectively.

6.2. SIMULATION SCENARIO DESIGN

For the purpose of this simulation study, 50 nodes randomly distributed in simulation area of 500mx500m are used. Each simulation is executed for 100 seconds of simulation time. The pause time used is 10s and a maximum speed of 10m/s is used for all cases. The number of traffic connections (CBR) used is 20 with a flow rate of 4 packets/s and a packet size of 512B. To study the energy consumption, an initial energy of 100 J, transmission power of 1.3 W, receive power of 0.9W and idle power of 0.74W is used for all the simulation scenarios. The typical values of energy consumption for a wireless interface card are taken from [59]. The sleep power is ignored in this simulation as it is insignificant.

The attacker nodes behave like any other node in the network in all aspects except that they send RREQ packets frequently, which are used for route discovery, or useless data packets. The attackers are classified as internal or external attackers. The internal attackers are those who know the keys and embed the right information in the reserved bits while sending route requests whereas the external attackers do not. And in this study, the prevention system is studied for cases where all the attackers are external, and half of them are internal. The number of attackers, the rate of sending RREQ messages and the data packet rates are varied to study the effect of flooding attack and the efficiency of the prevention mechanism. The traffic files are generated by excluding the malicious nodes. Three simulation scenarios are designed to be studied for the simulation and are summarized as follows. The effect of RREQ flooding attack is studied in both scenarios 1 and 2 where as data flooding attack is studied in scenario 3.

i) Scenario 1

The parameters used for scenario 1 are shown in table 6-1. Two attacker nodes sending fake RREQ messages at rates of 0, 5, 10, and 15 RREQs/s are used. The attackers start flooding at 0 second and stop at 99 seconds. The number of attackers is fixed at 2 and the RREQ flooding rate is varied in this particular scenario.

Objective:

- To study how RREQ flooding attack affects AODV's performance with increasing attackers' flooding rate and evaluate the effectiveness of the filter method and the proposed solution in preventing the flooding attack

Table 6-1: Simulation parameters for scenario 1

No	Parameters	Value
1	Simulator	Network simulator 2 (NS-2)
2	Simulation area	500m x500m
3	No of nodes	50
4	Simulation time	100s
5	Mobility model	Random waypoint
6	Mobility speed	10 m/s
7	Pause time	10s
8	Packets Rate	4 pkts/s
9	Packet size	512 bytes
10	Traffic type	CBR(UDP)
11	No. of attacker nodes	2
12	RREQ Flooding rate	0, 5, 10, 15 RREQ/s
13	No of connections	20

ii) Scenario 2

In scenario2, the number of attackers is varied and the flooding interval is kept constant at 3 RREQ/s. The number of attackers is varied with "0" representing the original AODV protocol with no attack and 4, 10, 16 are number of attackers flooding the specified network. The attackers start flooding at 0

second and stop at 99 seconds. Table 6-2 shows the values of the parameters used in this specific scenario.

Objective:

- To study the effect of RREQ flooding attack where there are many attackers that flood with small rates or attackers that abide by the rate limit and then evaluate the effectiveness of the filter method and the proposed solution in preventing flooding attack

Table 6-2: Simulation parameters for scenario2

No	**Parameters**	**Value**
1	Simulator	Network simulator 2 (NS-2)
2	Simulation area	500m x500m
3	No of nodes	50
4	Simulation time	100s
5	Mobility model	Random waypoint
6	Mobility speed	10 m/s
7	Pause time	10s
8	Packets Rate	4 pkts/s
9	Packet size	512 bytes
10	Traffic type	CBR (UDP)
11	No. of attacker nodes	0,4,10,16
12	RREQ Flooding rate	3 RREQ/s
13	No of connections	20

iii) Scenario 3

In this case, four attackers send data packets to four destination nodes at varying packet intervals (0, 10, 30 and 50 packets/s). Two of the attackers start at 0 second and stop flooding data packets at 50 seconds and the other two attackers start flooding at 50 seconds and stop sending at 99 seconds. In addition to data flooding attack, the attackers in scenario 3 also send 1 RREQ per second. So, in this scenario, the number of attackers and the RREQ flooding rate are fixed and the data flooding rate is varied. Table 6-3 shows the values of the parameters used in this particular scenario.

Table 6-3: Simulation parameters for scenario 3

No	Parameters	Value
1	Simulator	Network simulator 2 (NS-2)
2	Simulation area	500m x500m
3	No of nodes	50
4	Simulation time	100s
5	Mobility model	Random waypoint
6	Mobility speed	10 m/s
7	Pause time	10s
8	Packets Rate	4 pkts/s
9	Packet size	512 bytes
10	Traffic type	CBR(UDP)
11	No. of attacker nodes	4
12	RREQ Flooding rate	1 RREQ/s
13	Data Flooding rate	0, 10, 30, 50 pkts/s
14	No of connections	20

Objective:

- To study the effect of DATA Flooding attack where attackers flood with varying data packets and then evaluate the effectiveness of the filter method and the proposed solution in preventing this flooding attack

6.3. PERFORMANCE METRICS

In this simulation study, four performance metrics, [60, 61], are used to study the effect of the flooding attack and the effectiveness of the prevention mechanism.

Packet Delivery Ratio: Packet Delivery Ratio is a ratio of the number of packets received by the destination to the number of packets sent by the source which illustrates the level of delivered data to the destination.

Normalized routing load: It is the number of routing packets transmitted per data packet delivered which is computed as the number of routing packets divided by the number of data packets.

Average end-to-end delay: It is defined as average time taken by data packets to propagate from source to destination. This includes all possible delays caused by buffering during routing discovery, queuing at the interface queue, and retransmission delays at the MAC, propagation and transfer times.

Average Energy remaining: The average of energy remaining of all mobile nodes at the end of the simulation time.

6.4. RESULTS AND DISCUSSIONS

In all scenarios, the Filter method [15] is included for comparison and four values are plotted together. “RREQ_Flooding” denotes the RREQ flooding attack; “Filter” denotes the prevention mechanism that uses two threshold values; and “New_Half_EX” denotes the new solution with half of the attackers are external whereas “New_All_EX” denotes the new solution with all the attackers being external. “DATA_Flooding” is for data flooding attack. The simulations are run 5 times with different mobility files and the average of the values is computed. This is due to the fact that mobility files are randomly generated and the result for a different mobility file becomes different. The performance of the new solution evaluated for all the malicious nodes being internal overlaps with the filter method

and is left from the graphs. That is because if all the attackers are internal, the new solution will behave like the filter method.

6.4.1. SCENARIO1

The simulations are run by varying parameters according to the scenario presented above. The number of attackers is kept constant for scenario1 and the flooding interval is varied (0, 5, 10, 15 RREQs per second). The results for scenario 1 are shown below in figures 6.1 to 6.4.

As it can be seen from figure 6.1, the PDR drops sharply with increasing RREQ flooding rate and decreases by 73.3%, from no attack scenario, for 2 attackers sending 15 RREQs/s each in the absence of any prevention mechanism. So, the flooding attack can flood the whole network at higher flooding rates and can be very severe if not prevented. The PDR for all the prevention mechanisms at higher flooding rates (greater or equal to 10 RREQs/s) is improved. The reason for this is that both the filter method and the new solution will blacklist the attackers and further flooding is avoided. So, at higher flooding rates, both the solutions are comparable. If the flooding rate is below 10 RREQs/s, then the filter method poorly prevents compared to the New_All_EX and New_Half_EX because the filter method delays RREQs of such nodes. For example, for rate of 5, the PDR is improved by 2.4%, 6.8% and 7.5% using the Filter method, New_Half_EX and New_All_EX respectively which was originally decreased by 7.64% by the attack. The PDR for New_All_EX is almost constant due to the fact that all the attackers will be blacklisted as they don't have the correct keys embedded in the reserved bits. New_Half_EX performs better than filter as half of the attackers, in this case one, are external and will be blacklisted the moment they start sending route requests.

The routing load, shown in figure 6.2, of the flooding attack is similarly is higher because of the attackers sending RREQs frequently. At higher flooding rates, the number of control packets needed to transmit one data packet increases dramatically. The improvement of the filter method and the new solution is again comparable for higher flooding rate. For example, for 15 RREQs/s, the improvement of the Filter, New_Half_EX and New_All_EX is 96.6%, 97.5% and 98.3% respectively. For flooding rates below 10, the Filter performs low as the malicious nodes are only punished by delaying the requests. For rate of 5 requests/s, the control packets have decreased by 7.5%, 51% and 86.7% using the Filter, New_Half_EX and New_All_EX respectively.

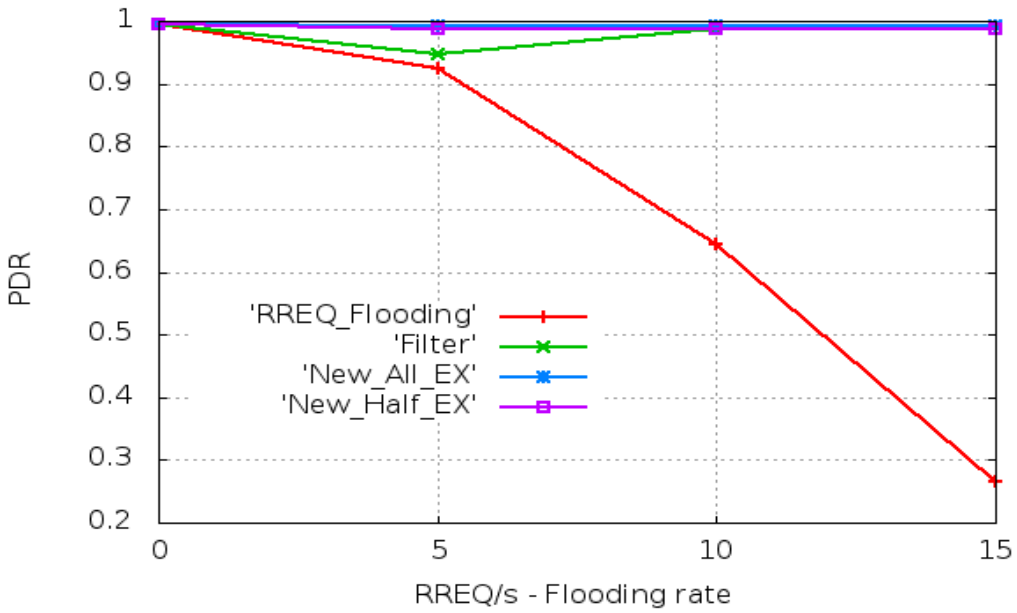

Figure 6.1: Packet delivery ratio for scenario 1

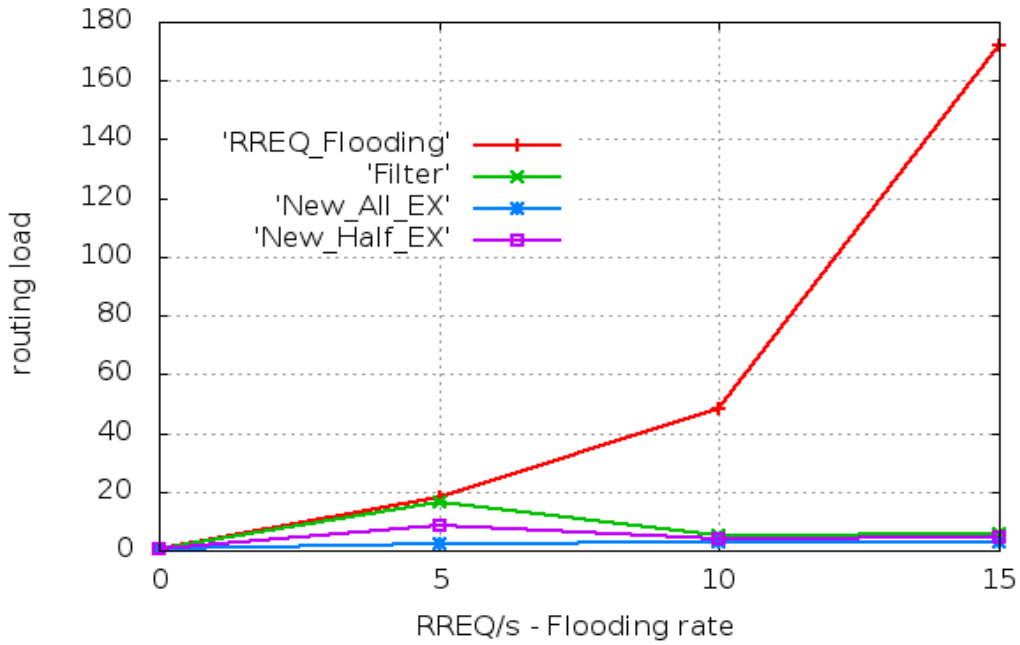

Figure 6.2: Routing load for scenario 1

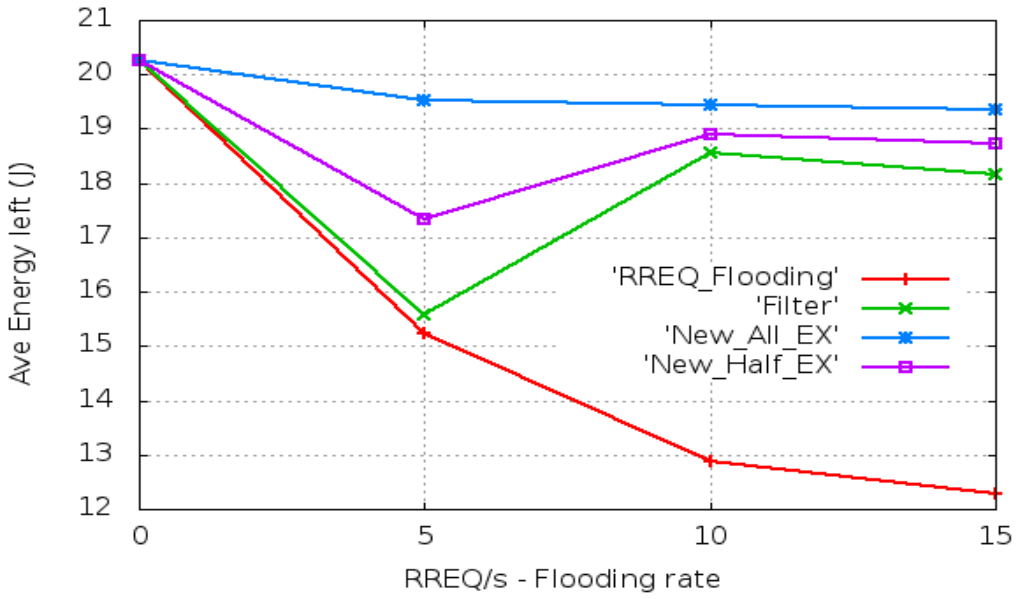

Figure 6.3: Average energy remaining at the end of the simulation for scenario 1

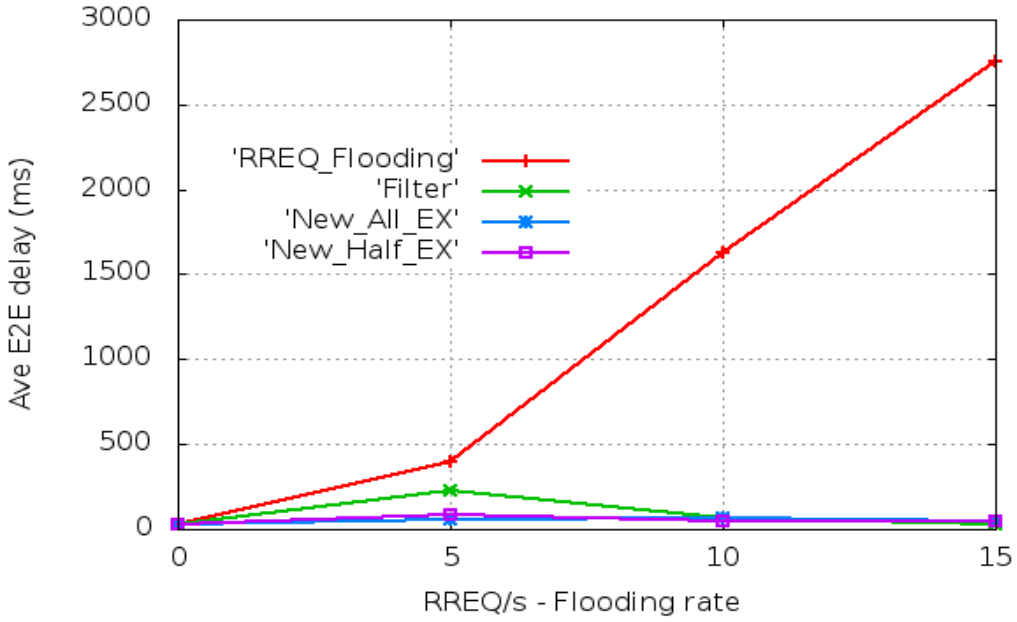

Figure 6.4: Average end-to-end delay for scenario 1

Figure 6.3 shows the average energy left at the end of the simulation. At normal network operation, the average remaining energy at the end of the simulation time is around 20J. But, with flooding attack the average energy left drops sharply. The average remaining energy for flooding attack with rate of 15 drops by 39.3% and the increment of the improvement over flooding is greater than by 48%

for both the filter method and the new mechanism. But the remaining energy for rates below 10 RREQs/s is smaller for filter method than the New_Half_EX and New_All_EX as shown in the diagram.

Similarly, the average end-to-end delay in the presence of flooding attack sharply rises with increasing flooding rate. The average end-to-end delay is shown in figure 6.4 graphically. The delay for all the prevention systems is comparable for rate beyond 10 RREQs/s as attackers will be blacklisted. But, delay below 10 RREQ/s is higher for filter method. Again, the end-to-end delay for New_All_EX is almost constant in all cases. For the case of 5 RREQs/s, the average end-to-end delay decreased by 42.7%, 77.8% and 86% using the filter, New_Half_EX and New_All_EX respectively as preventions for the attack.

In general, the results of scenario1 show that the effect of RREQ flooding attack is very severe for higher flooding rates that the performance of the network deteriorates to a large extent. As prevention mechanism, the filter method is effective for higher flooding rates (greater or equal to 10 RREQs/s) and the effect of the flooding attack is reduced if the rate is between the RATE_LIMIT and BALACKLIST_LIMIT using this mechanism. Using the new solution the effect of the flooding attack is avoided especially if all the attackers are external. The filter method and the new solution are comparable for higher flooding rates.

6.4.2. SCENARIO 2

In this scenario, many attackers cooperate to flood a given network at lower RREQ flooding rates. The results for this scenario are shown in figures 6.5 to 6.8.

The PDR, shown in figure 6.5, dropped to below 14% when 16 attackers send 3 RREQs/s each for both RREQ_Flooding and Filter method. The filter method cannot prevent such attacks because these attackers send RREQ without violating the threshold limit. The prevention mechanisms that are based on threshold limit face such difficulties. The combined effect of many attackers, sending few requests each, can severely affect the performance of a network. If all the attackers are external, the new solution (New_All_EX) improves the PDR and is almost constant at about 99% for all number of attackers. With half of the attackers being external, New_Half_EX, there is a lot improvement compared to the filter method. For example, for 16 attackers, the PDR of New_Half_EX is almost 73% compared to 13.3% in case of the filter method.

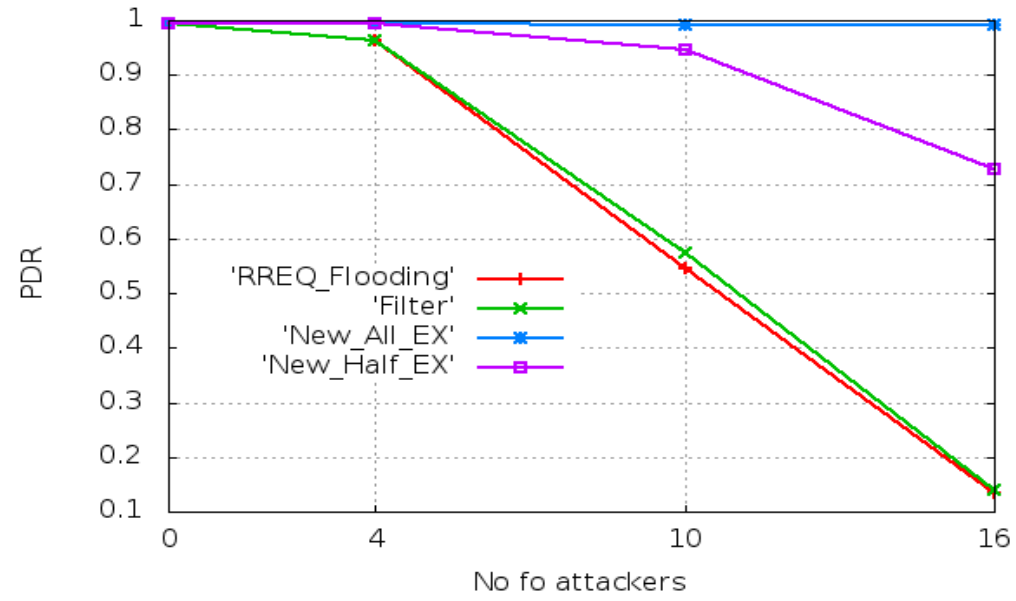

Figure 6.5: Packet delivery ratio for scenario 2

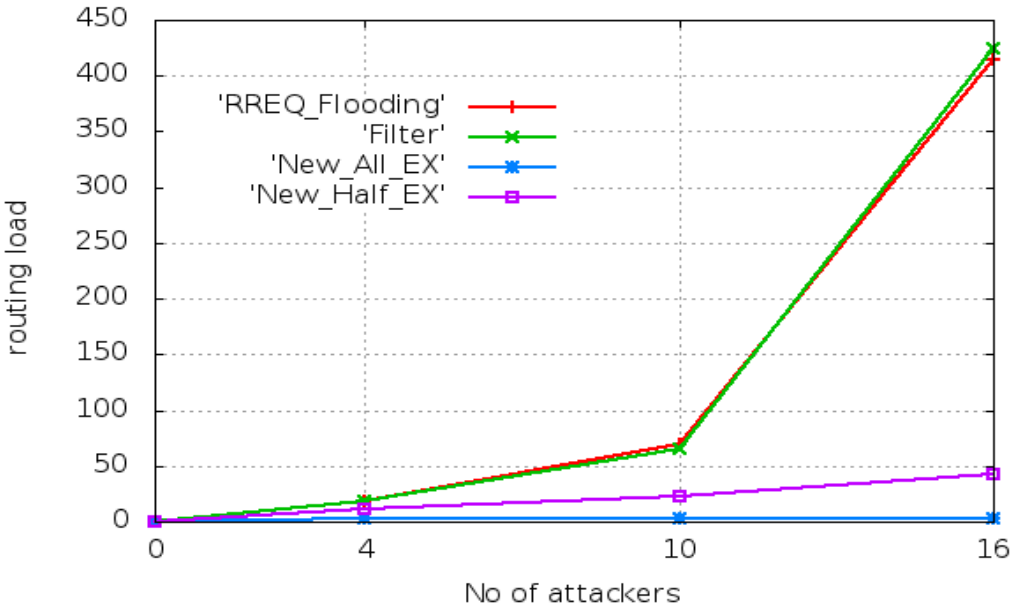

Figure 6.6: Routing load for scenario 2

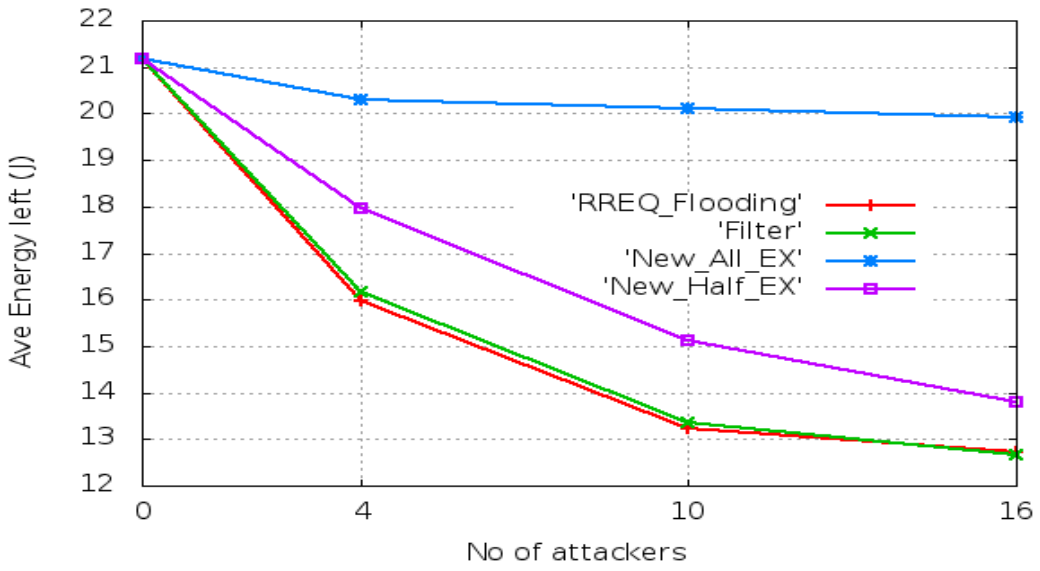

Figure 6.7: Average energy remaining at the end of the simulation for scenario 2

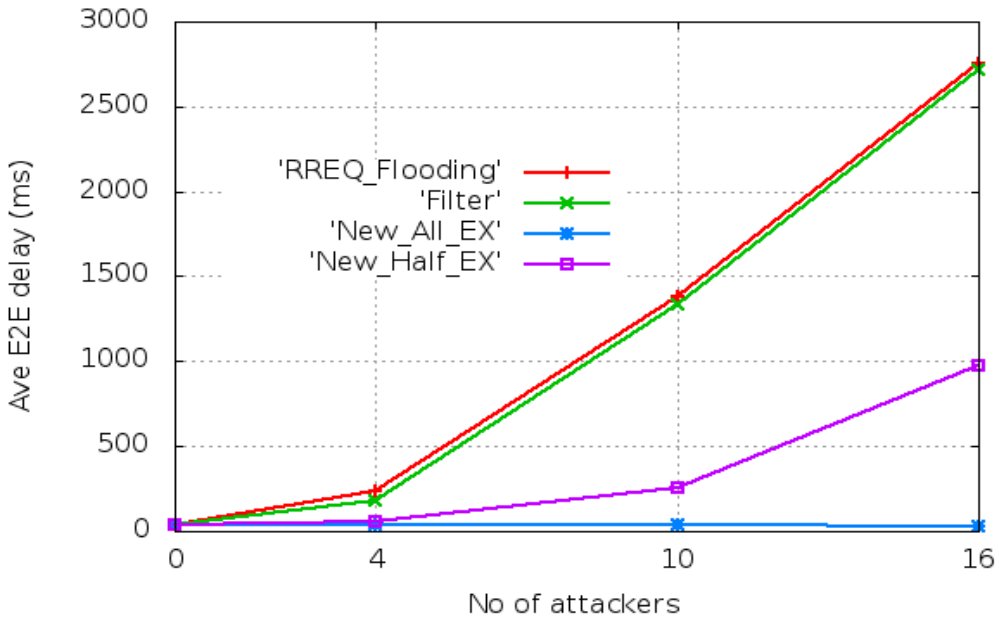

Figure 6.8: Average end-to-end delay for scenario 2

As can be seen from figure 6.6, the routing overhead of the flooding attack is higher. The control packets that are produced by 16 attackers are large enough to collapse a given network. The routing load of the filter method is equal to the flooding attack. The routing load of New_All_EX is again kept almost constant with increasing attackers because the attackers with no right key will be

blacklisted. If half of the attackers are internal, then the other half (external attackers) is blacklisted in New_Half_EX. For example, for 16 attackers, the number of control packets needed to transmit one data packet is extremely higher for both flooding and the filter method. Using the new solution, 99.1% and 89.6% of the control packets are avoided using New_All_EX and New_Half_EX respectively.

The average energy remaining is also lower for both flooding attack and the filter method which decreased by about 40% for 16 attackers. For the same number of attackers, the improvement with the new solution, New_All_EX, is 56% and with New_Half_EX is 8.3%. If all the attackers are external, the average energy left almost insignificantly varies with increasing number of attackers. The average energy left at the end of the simulation is shown in figure 6.7. The average end-to-end delay, shown in figure 6.8, also follows the same trend. That is, the delay dramatically increases for flooding attack and the filter method. But the average end-to-delay for the New_All_EX is very small and almost constant even with increasing number of attackers. The improvement of the New_All_EX over the flooding attack is 99% and for New_Half_EX , it is 64.5% when 16 attackers send RREQs below the rate limit.

In this scenario, the filter method in particular and other threshold based mechanisms in general poorly prevent the RREQ flooding attack that can be imposed by attackers sending RREQ messages without violating the allowed rate limit. The effect of such attack makes it severe as the number of flooding attackers increases. There is a tremendous enhancement in network performance over the flooding attack if the new solution is evaluated against all external attackers. There is still performance improvement if there exist any number of external attackers. The flooding rate used in this case is 3 RREQs/s and in some cases some attackers are observed to be punished by the filter method by delaying their requests. The reason is that, at some particular time, in addition to these 3 requests, other additional requests that can be generated by route re-discovery in the case of route failure or delayed requests may increase the number of requests to be received per unit of time. But, few numbers of requests are affected by the filter method and is insignificant to prevent the effect of the flooding attack.

6.4.3. SCENARIO 3

In this scenario, four attackers send fake data packets to four destinations at different packet rates. The attackers also send 1 RREQ/s in addition to the data flooding attack. The effect of these attackers is studied using the prevention mechanisms. The simulation results for scenario 3 are shown in figures 6.9 to 6.12.

The routing overhead due to the control packets is shown in figure 6.9. As it can be seen from the diagram, the routing load is higher for data flooding and the filter method of prevention. The routing load does not seem to increase proportionally as the packet rate of data flooding increases. This is due to, once a link is established, the fake data packets can be sent without generating any additional control packers as far as the links are not broken. The routing overhead for New_All_EX is very small compared to the filter method and is observed to be almost constant even with increasing data flooding rate as fake requests will be avoided using the reserved bits. The overhead for New_Half_EX lies in between as the effect of the external attackers is avoided. For example, for a rate of 50 packets/s and 1 RREQ/s, the improvement of routing load over the flooding attack is around 89% if all attackers are external and 46.6% if half of the attackers are internal.

Similarly, the average energy left is smaller for data flooding attack and the filter method. As can be seen in figure 6.10, irrespective of the data flooding rate, the average energy is almost constant for the new solution and, in the case of 50 packets/s, the improvement is around 83.2% for New_All_EX and around 26.6% for New_Half_EX over the flooding attack. Figure 6.11 shows the average end-to-end delay of this scenario. Using the reserved bits to prevent the flooding attacks has lowered the end-to-end delay very much. For example, for 50 packets/s, the improvement is around 94.2% and 67.4% for New_All_EX and New_Half_EX respectively. The PDR, shown in figure 6.12, also reflects similar output. The PDR for data flooding and the filter method is smaller compared with the new solution. The New_All_EX has enhanced the PDR by 8.6% and the New_Half_EX improved it by 5.8% over the flooding attack and the filter method at a rate of 50 attacking data packets.

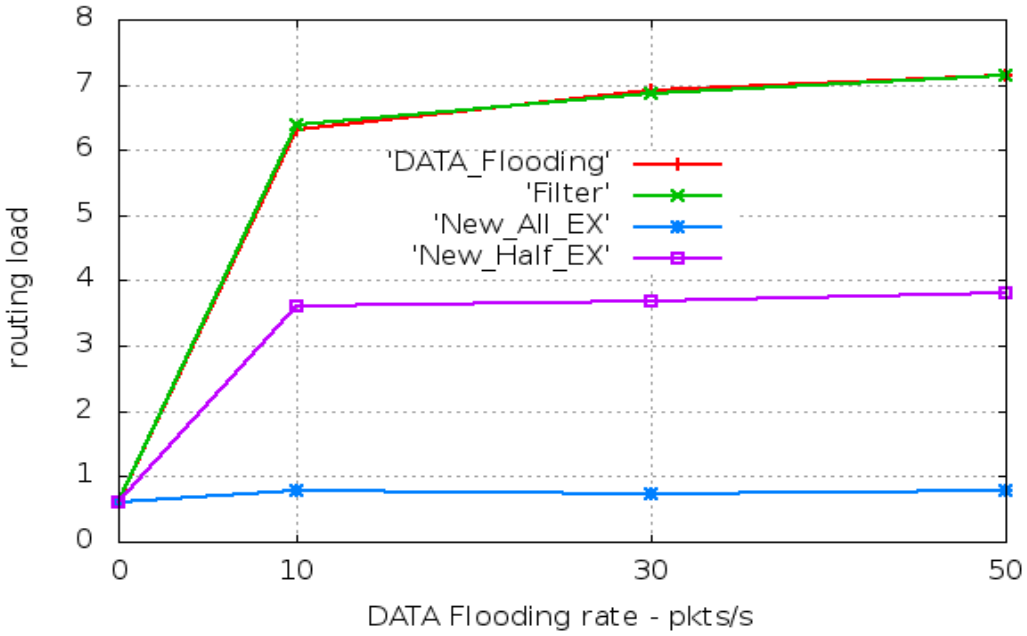

Figure 6.9: Routing load for scenario 3

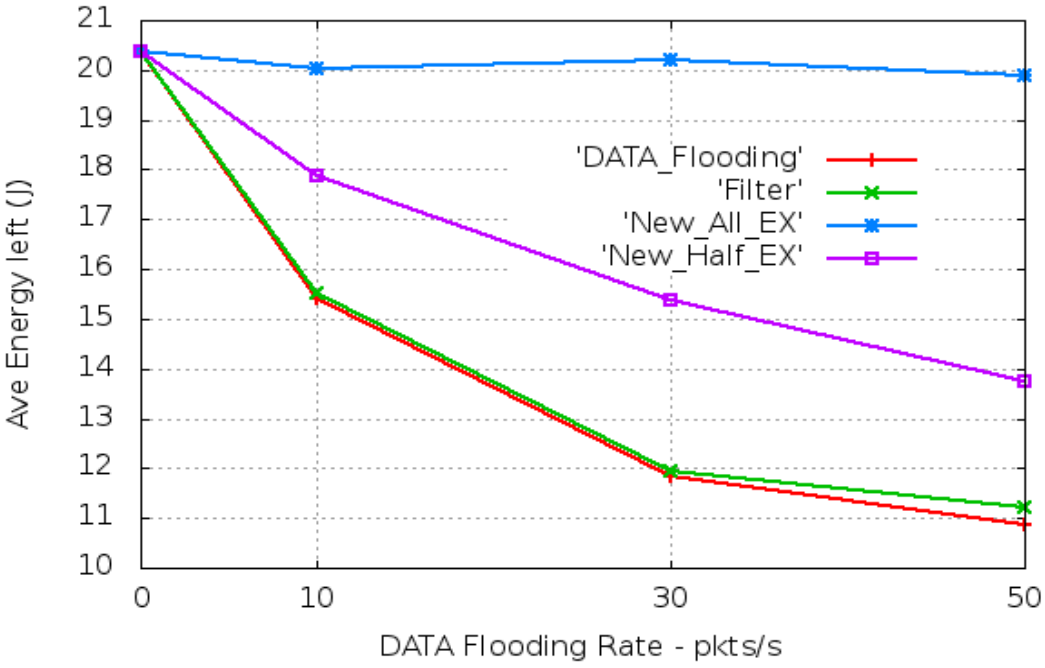

Figure 6.10: Average energy remaining at the end of the simulation for scenario 3

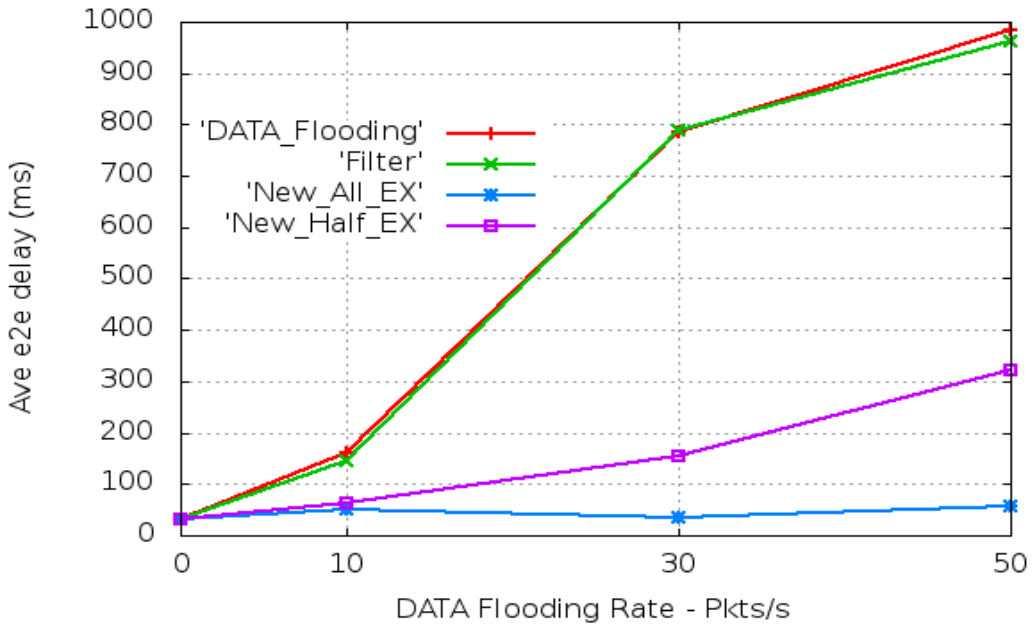

Figure 6.11: Average end-to-end delay for scenario 3

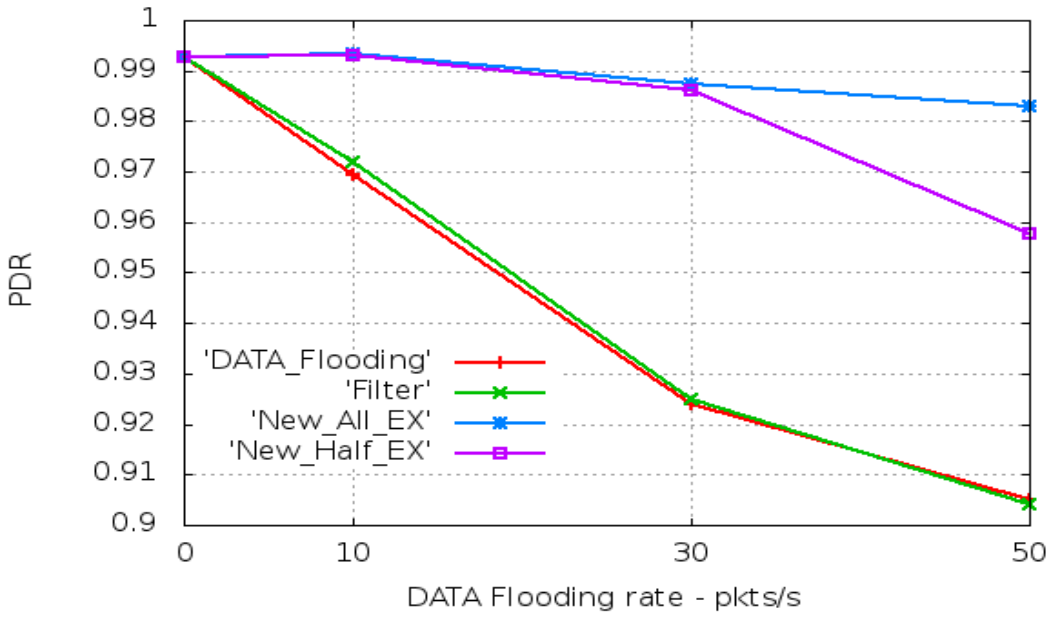

Figure 6.12: Packet delivery ratio for scenario 3

In conclusion, the data flooding attack has also significant impact on the network performance. The filter method has little preventive effect for this scenario. The reason for this is that the filter method treats normally for RREQs below the RATE_LIMIT that it cannot prevent 1RREQ/s and some few more requests to establish links. Furthermore, unless the attackers are blacklisted by violating the

threshold limit, the data flooding by such attackers cannot be prevented using the filter method. The prevention mechanisms based on threshold limitation have this problem in preventing flooding attack. Similarly, the prevention of data flooding attack from internal attackers could be difficult using this solution when the attackers tend to produce low rate of route request messages. So, to prevent internal attackers from data flooding attacks, other methods rather than rate limitation should be used, but external attackers can still be prevented using this solution. It can be shown from the results that the improvement of network performance is remarkable for the new solution if all the attackers are external. And there is still improvement if some proportion of the attackers is external.

Unlike the route request flooding attack, the data flooding attack seems to have less severe effect especially in increasing the amount control packets in the network. Moreover, in data flooding attack, certain links and nodes in the path of the source-destination pairs will most of the time be affected. Where as in RREQ flooding attack, the control packets that are generated by one specific node will be flooded to many nodes and these control packets will have higher priority over genuine data packets. So, as can be seen from the simulation results, if attackers combine both types of flooding attacks, then the effect could result in complete network collapse. In all the scenarios, the improvement of performance of the new prevention mechanism is significant especially when all the attackers are external.

CHAPTER SEVEN

CONCLUSIONS AND RECOMMENDATIONS

7.1. CONCLUSIONS

In this thesis work, the reserved bits of AODV RREQ messages are used to embed a key to identify external attackers sending fake RREQs and useless data packets. Using this mechanism, external attackers are prevented from imposing both route request and data flooding attacks by completely isolating them. Since, external attackers do not embed the appropriate keys; they can be identified the moment they start sending RREQ message. Route request flooding attack occurring below the rate limitation can also be prevented from intruders. Hence, avoiding the severe effect occurring by many external attackers sending RREQ messages abiding by the threshold limitation to flood a network cooperatively in addition to preventing data flooding is the main advantage of this solution.

The new solution also incorporated one of the existing prevention mechanisms, the Filter method in [15], to prevent flooding attack by internal attackers. The flooding attack and the prevention mechanism are implemented by modifying the C++ code of AODV and the system is simulated using NS-2 simulator. To study the effect of the flooding attack and the effectiveness of the prevention mechanism, three simulation scenarios are used. For all the scenarios, 50 nodes randomly moving in 500mx500m area are used.

In the first scenario, the results show that at higher RREQ flooding rates, Filter method and the new solution perform well in preventing the flooding attack, but if the rate of flooding is below the BLACKLIST_LIMIT, the new solution is better than the Filter method if some of the attackers are external. In the second scenario, it is shown that many attackers cooperatively flooding at a low flooding rate can significantly affect the network performance. In this case, the solutions based on threshold limitation poorly prevent the flooding attack as the rate limit is not violated. And the results show that the Filter method does nothing in preventing the flooding attack because the rate of flooding is below RATE_LIMIT and the new solution prevents such flooding attack and is tremendous especially if all the attackers are external.

In the case of the third scenario, the effect of data flooding attack is studied. In this case too, the Filter method poorly prevents the flooding attack but the performances are improved with the new solution. In second and third scenarios, other prevention mechanisms based on threshold limitation will also poorly perform, like the Filter method, as the number of route requests send by the attackers could be extremely low to be prevented by using these mechanisms. But, as can be seen from the results of these scenarios, the effect of such flooding could be significant especially in the presence of many attackers.

In all the scenarios, the new prevention mechanism is effective in preventing both flooding attacks particularly when it is evaluated against external attackers. The results also show that if some proportions of the attackers are external, then there is a significant improvement in the network performance using the new mechanism to prevent the RREQ flooding and the data flooding attacks. But, the usage of a key based prevention mechanism may create some limitations on the flexibility of the dynamism of mobile ad-hoc networks when mobile nodes are admitting to a given ad-hoc network.

7.2. RECOMMENDATIONS

This prevention mechanism is light weight and needs no modification in the existing message format of the routing protocol and it can easily be integrated into the routing protocol or other existing prevention mechanisms with little overhead. To use the reserved bits for prevention of attacks, other effective methods or operations can be used rather than manipulating the source address of the originator using the keys to prevent flooding attacks and other attacks in the routing protocols.

The prevention mechanism can be extended and analyzed to use it in the prevention of

- Other control packet flooding attacks: To prevent such control packets flooding attacks (e.g Hello flooding attack) from internal attackers, one of the existing solutions can be used and the reserved bits can be used to embed keys that can identify external attackers from flooding excessive control packers.
- Other active attacks in AODV (for e.g, packet dropping): In this case too, other existing solutions can be used to prevent the specific attack by internal attackers and reserved bits can

be used in preventing such active attacks from external attackers. The use of reserved bits will protect external attackers with little overhead on the performance of the routing protocol.

- Flooding attack in other reactive routing protocols: Control packet flooding attacks can be prevented from external attackers using the reserved bits in DSR routing protocol.
- The new solution can also be studied by using other existing prevention mechanisms other than the filter method to prevent internal attackers.

REFERENCES

[1] Imrich Chlamtac, et al, "*Mobile ad hoc networking: imperatives and challenges*", Ad Hoc Networks 1 (2003) 13–64, doi:10.1016/S1570-8705(03)00013-1, © 2003 Elsevier B.V

[2] Juan P. Maícas, "*Recent Developments in Mobile Communications – A Multidisciplinary Approach*", ISBN 978-953-307-910-3, © 2011 InTech

[3] Rajan Bansal, et al, "*Analytical Study the Performance Evaluation of Mobile Ad Hoc Networks using AODV Protocol"*, International Journal of Computer Applications (0975 – 8887), Volume 14– No.4, January 2011

[4] Fraser Cadger, et al, "*A Survey of Geographical Routing in Wireless Ad-Hoc Networks*", IEEE Communications Surveys & Tutorials, DOI 10.1109/SURV.2012.062612.00109, © 2012 IEEE

[5] Pravin Ghosekar, et al, "*Mobile Ad Hoc Networking: Imperatives and Challenges*", IJCA Special Issue on "Mobile Ad-hoc Networks" MANETs, 2010

[6] I. Chlamtac et al, "*Mobile ad hoc networking: imperatives and challenges*", doi: 10.1016/S1570-8705(03)00013-1, Ad Hoc Networks 1 (2003) 13–64, © 2003 Elsevier B.V.

[7] J. Hubaux, et al, "*The Quest for Security in Mobile Ad Hoc Networks*", Proceedings of the ACM Symposium on Mobile Adhoc Networking and Computing (MobiHOC 2001), © 2001 by ACM

[8] Hao Yang, et al, "*Security in Mobile Adhoc Networks: Challenges and Solutions*", IEEE Wireless Communications, February 2004

[9] Sudhir Agrawal , et al, "*A Survey of Routing Attacks and Security Measures in Mobile Ad-Hoc Networks "*, Journal of Computing, Volume 3, Issue 1, January 2011

[10] Anil G. N. ,et al, "*Semantic Probabilistic Modeling of novel routing Protocol with Implication of Cumulative Routing Attack in Mobile adhoc network "*, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 2, January 2012

[11] Ping Yi, et al, "**A** *New Routing Attack in Mobile Ad Hoc Networks*", International Journal of Information Technology, Vol. 11 No. 2, 2005

[12] Saman Desilva , et al, "*Mitigating Malicious Control Packet Floods in Ad Hoc Networks "*, IEEE Wireless Communications and Networking Conference, March 2005

[13] V. Balakrishnan, et al, "*Followship: Defense against Flooding and Packet Drop Attacks in MANET*", 1-4244-0143-7/06 © 2006 IEEE

[14] Zhi Ang EU, et al, "*Mitigating Route Request Flooding Attacks in Mobile Ad Hoc Networks*", Proceedings of International Conference on Information networking (ICOIN-2006), Sendai, Japan, 2006

[15] Jian-Hua Song, et. Al, "*Effective Filtering Scheme against RREQ Flooding Attack in Mobile Ad Hoc Networks*", Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06), IEEE Computer Society, 2006

[16] Venkat Balakrishnan, et al, "*Mitigating Flooding Attacks in Mobile Ad-hoc Networks Supporting Anonymous Communications*", The 2nd International Conference on Wireless Broadband and Ultra Wideband Communications (AusWireless 2007) , 0-7695-2842-2/07 © 2007 IEEE

[17] S. K.Shandilya, et. Al, "*A Trust Based Security Scheme for RREQ Flooding Attack in MANET*", International Journal of Computer Applications (0975 – 8887), Volume 5– No.12, August 2010

[18] H. Kim, et al, " *Novel Defense Mechanism against Data Flooding Attacks in Wireless Ad Hoc Networks* ", IEEE Transactions on Consumer Electronics, Vol. 56, No. 2, May 2010

[19] Ai-Fen Sui, et al, "*An effective method to mitigate route query floods in MANETs*", Communication Technology (ICCT), 2011 IEEE 13th International Conference on , vol., no., pp.709,713, 25-28 Sept. 2011

[20] Rajnish Choubey, et al, "*Flooding Attack Prevention Algorithm in AODV Protocol for Mobile Ad-hoc Network*", International Journal of Science and Advanced Technology (ISSN 2221-8386), Volume 1, No 6, August 2011

[21] Pratik Singh, et. al, " *Flood Tolerant AODV Protocol (FT-AODV "*, International Journal of Computer Applications (0975 – 8887) , Volume 53– No.6, September 2012

[22] Madhavi, S, et. Al, "*FLOODING ATTACK AWARE SECURE AODV*", Journal of Computer Science 2013, 9 (1), 105-113, ISSN 1549-3636, © 2013 Science Publications

[23] B. Kannhavong, et al, "*A Survey of Routing Attacks in Mobile Ad Hoc Networks*", IEEE Wireless Communications, October 2007

[24] Loay Abusalah, et al, "*A Survey of Secure Mobile Ad Hoc Routing Protocols*", IEEE Communications Surveys & Tutorials, Vol. 10, No. 4, Fourth Quarter 2008

[25] C. Perkins, et al, "*Ad hoc On-Demand Distance Vector (AODV) Routing*", Network Working Group , RFC 3561, July 2003

[26] Perkins, C.E., et al, "*Ad-hoc on-demand distance vector routing*," Mobile Computing Systems and Applications, 1999, Proceedings WMCSA '99 Second IEEE Workshop on, vol., no., pp.90,100, 25-26 Feb 1999

[27] Mahesh K. Marina, et al, "*Ad hoc on-demand multipath distance vector routing*", Wireless Communications and Mobile Computing, 2006;6:969–988, © 2006 John Wiley & Sons, Ltd

[28] Sung-Ju Lee, et al, "*Scalability study of the ad hoc on-demand distance vector routing protocol*", International Journal of Network Management, 2003; 13: 97–114, © 2003 John Wiley & Sons, Ltd

[29] W. Kassahun, "*Performance Evaluation and Nodes' Mobility Effect Analysis on AODV based Internet Gateway Discovery Algorithms in Social Networks*", Addis Ababa University, Faculty of Technology, Master's Thesis, March, 2010

[30], Zapata, et al, "*Securing Ad hoc Routing Protocols*", In Proceedings of the ACM workshop on Wireless security, Atlanta, USA (2002)

[31] A. K. Agrawal, et al, "*Time Factor Analysis on Prevention Algorithm in Mobile Ad-Hoc Network*", International Journal of Multidisciplinary Research, Vol.2 Issue 5, May 2012

[32] Eduardo Da Silva, et al, "*Identity-Based Key Management in Mobile Ad Hoc Networks: Techniques and Applications*", IEEE Wireless Communications, October 2008

[33] Suman Deswal, et al, "*Implementation of Routing Security Aspects in AODV*", International Journal of Computer Theory and Engineering, Vol. 2, No. 1 February, 2010

[34] Monika, et al, "*Security Aspects in Mobile Ad Hoc Network (MANETs): Technical Review*", International Journal of Computer Applications (0975 –8887), Volume 12–No.2, November 2010

[35] Djamel Djenouri, et al, "*A Survey of Security Issues in Mobile Ad Hoc and Sensor Networks*", IEEE Communications, Fourth Quarter 2005,Volume 7,No. 4 ,2005

[36] Bing Wu, et al, "*A Survey of Attacks and Countermeasures in Mobile Ad Hoc Networks*", Wireless/Mobile Network Security, © 2006 Springer

[37] M. N.Lima, et al, "*A Survey of Survivability in Mobile Ad Hoc Networks*", IEEE Communications Surveys & Tutorials, Vol. 11, No. 1, First Quarter 2009

[38] Manikandan, et al, "*Survey on Mobile Ad Hoc Network Attacks and Mitigation Using Routing Protocols"*, American Journal of Applied Sciences, 2012, 9 (11), 1796-1801, ISSN 1546-9239, © 2012 Science Publication

[39] A.Vani, et al, "*Providing of Secure Routing against Attacks in MANETs*", International Journal of Computer Applications (0975 – 8887), Volume 24– No.8, June 2011

[40] Priyanka Goyal, et. Al, "*A Literature Review of Security Attack in Mobile Ad-hoc Networks*", International Journal of Computer Applications (0975 – 8887), Volume 9– No.12, November 2010

[41] P. M. Jawandhiya, "*A Survey of Mobile Ad Hoc Network Attacks*", International Journal of Engineering Science and Technology, Vol. 2(9), 2010

[42] Ms.Supriya, et, al., "*MANET Security Breaches: Threat to a Secure Communication Platform*", International Journal on Ad Hoc Networking Systems (IJANS) Vol. 2, No. 2, April 2012

[43] S. Kannan, et. Al, "*A Study of Attacks, Attack Detection and Prevention Methods in Proactive and Reactive routing Protocols*", International Business Management 5 (3):178-183, Medwell Journals, 2011

[44] Sarvesh Tanwar, et, al. ,"*Threats & Security Issues in Ad hoc network: A Survey Report*", International Journal of Soft Computing and Engineering (IJSCE), ISSN: 2231-2307, Volume-2, Issue-6, January 2013

[45] Rusha Nandy, et. Al, "*Study of Various Attacks in MANET and Elaborative Discussion of Rushing Attack on DSR with clustering scheme*", Int. J. Advanced Networking and Applications Volume: 03, Issue: 01, May 13, 2011

[46] Ranu Patel, et al, "*Analysis of flooding attack using random waypoint mobility model in mobile adhoc network in NS-3*", Elixir Comp. Sci. & Engg, 56A (2013) 13534-13538, © 2013 Elixir

[47] Bhuvaneshwari. K, et, al, "*Examination of Impact of Flooding attack on MANET and to accentuate on Performance Degradation*", Int. J. Advanced Networking and Applications, Volume: 04 Issue: 04, ISSN : 0975-0290, 2013

[48] Saba Siraj, et al, "*Network Simulation Tools Survey*", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 1, Issue 4, June 2012

[49] T. Issariyakul, E. Hossain, "Introduction to Network Simulator NS2", DOI: 10.1007/978-0-387-71760-9 2, ©Springer Science+Business Media, LLC 2009

[50] The VINT Project, "*The ns Manual*", [Online]: http://www.isi.edu/nsnam/ns/ns-documentation [Access date: Feb 2013]

[51] Aliff Umair Salleh , et al, " *Trace Analyzer for NS-2 "*, 4th Student Conference on Research and Development (SCOReD 2006) , 1-4244-0527-0/06/ © 2006 IEEE

[52] [Online]: http://ns2ultimate.tumblr.com/post/32176398101/post-processing-ns2-result-using-ns2-trace-new [Access date: March 2013]

[53] [Online]: http://stanjuly.wordpress.com/2011/12/22/install-ns2-ns-allinone-2-35-on-ubuntu-11-04-for-beginners/, [Access date: Dec 2012]

[54] T. Issariyakul, et al, "*Introduction to Network Simulator NS2*", DOI 10.1007/978-1-4614-1406-3_12, © Springer Science+Business Media, LLC 2012

[55] Rehmani, "*A Tutorial on the Implementation of Ad-hoc On Demand Distance Vector (AODV) Protocol in Network Simulator (NS-2*)", [Online]: http://arxiv.org/pdf/1007.4065.pdf, [Access date: March 2013]

[56] S. Basha Shaik, “Effect of Placement Model on Average Jitter in Reactive Routing Protocols”, JEST-M, Vol.1, Issue 2, 2012

[57] [Online]: http://narentada.com/code-for-preventing-flood-attack-in-aodv/, [Access date: May 2013]

[58] Christian Bettstetter, et al, “ *Stochastic Properties of the Random Waypoint Mobility Model ”*, ACM/Kluwer Wireless Networks, Special Issue on Modeling & Analysis of Mobile Networks , Mar 10, 2003

[59] Marco Fotino, et al, “*Energy Issues and Energy aware Routing in Wireless Ad-hoc Networks-Mobile Ad-Hoc Networks: Protocol Design*”, ISBN 978-953-307-402-3, January, 2011

[60] N. Gandhewar, et al, “*Performance Evaluation of AODV protocol in MANET using NS2 Simulator*”, Proceedings published in International Journal of Computer Applications® (IJCA), 2nd National Conference on Information and Communication Technology (NCICT), 2011

[61] Charles E. Perkins, et, al, “*Performance Comparison of Two On-Demand Routing Protocols for Ad Hoc Networks*”, IEEE Personal Communications, February 2001, © 2001 IEEE

[62] Xiangpeng Jing, et. al, “*Energy-Aware Algorithms for AODV in Ad Hoc Networks*”, in Proceedings of Mobile Computing and Ubiquitous Networking (ICMU 2004), Yokosuka, Japan, Jan 2004

[63] Amandeep, et al, “*Performance Analysis of AODV Routing Protocol in MANETs*”, International Journal of Engineering Science and Technology (IJEST), Vol. 4, No.08, August 2012

APPENDIX

A: SAMPLE TCL FILE

```
#******** MOBILE AD-HOC NETWORKS *****************

     #This Tcl code is specifically for flooding attack without
prevention
     # No of nodes:50
     #No of attackers:10
     #Flooding rate: 3 req/s
     #Use the aodv codes with flooding attack enabled
#    #*********************************************
#the ff parameters are used in node configurations
set val(chan)  Channel/WirelessChannel;#channel type
set val(prop)  Propagation/TwoRayGround;#radio prop model
set val(netif)      Phy/WirelessPhy ;#ntk interface type
set val(mac)   Mac/802_11;#mac type
set val(ifq)   Queue/DropTail/PriQueue;#int queue type
set val(ll)    LL;#link layer type
set val(ant)   Antenna/OmniAntenna;#antenna model
set val(ifqlen)     50;#max pkt in ifq
#*********************************************
set val(nn)    50;# No of mobile nodes
#*********************************************
set val(rp)    AODV;
set opt(agent)          Agent/AODV
set val(x)     500;
set val(y)     500;
set val(SimTime) 100;
#*********************************************
set val(movfile) "../mobility_files/mob_50"
set val(cbrtraffic) "../traffic_files/cbr_50"
#*********************************************
set val(txp)  1.3;#in W
set val(rxp)  0.9;#in W
Set val (ip) 0.74;

set val(inip)  100;#in J
 set ns_ [new Simulator]
```

```
#*************************************************
set tracefd [open ../trace_flooding_50_3_10_1.tr w]
#*************************************************
set namtrace [open simwrls.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
#topology
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
#create-god $val(nn)
#create-god changed to this
set god_ [create-god $val(nn)]
##configuring a node
$ns_ node-config -adhocRouting $val(rp) \
          -llType $val(ll) \
          -macType $val(mac) \
          -ifqType $val(ifq) \
          -ifqLen $val(ifqlen) \
          -antType $val(ant) \
          -propType $val(prop) \
          -phyType $val(netif) \
          -channelType $val(chan) \
          -topoInstance $topo \
          -agentTrace ON \
          -routerTrace ON \
          -macTrace OFF \
          -energyModel    EnergyModel \
          -initialEnergy  $val(inip) \
          -rxPower        $val(rxp) \
          -txPower        $val(txp) \
          -idlePower      $val(ip)\

  #Use new trace format
$ns_ use-newtrace
#create the nodes
for {set i 0} { $i < $val(nn) } { incr i } {
set node_($i) [$ns_ node]
$node_($i) random-motion 0
}
#*****************************************************************
#Assigning attacker nodes
```

```
$ns_ at 0.0 "[$node_(40) set ragent_] intruder 1111 0.333"
$ns_ at 0.0 "[$node_(41) set ragent_] internal_attacker 1111 0.333"
$ns_ at 0.0 "[$node_(42) set ragent_] intruder 1111 0.333"
$ns_ at 0.0 "[$node_(43) set ragent_] internal_attacker 1111 0.333"
$ns_ at 0.0 "[$node_(44) set ragent_] intruder 1111 0.333"
$ns_ at 0.0 "[$node_(45) set ragent_] internal_attacker 1111 0.333"
$ns_ at 0.0 "[$node_(46) set ragent_] intruder 1111 0.333"
$ns_ at 0.0 "[$node_(47) set ragent_] internal_attacker 1111 0.333"
$ns_ at 0.0 "[$node_(48) set ragent_] intruder 1111 0.333"
$ns_ at 0.0 "[$node_(49) set ragent_] internal_attacker 1111 0.333"
#*************************************************************
#mobility file and traffic scenario file
puts "Loading scenario mobility file..."
source $val(movfile)
puts "Loading traffic file..."
source $val(cbrtraffic)
#initial location of nodes
#after mobility file
     set rng_ [new RNG]
     $rng_ seed 0
     for {set i 0} { $i < $val(nn) } { incr i } {
     $ns_ initial_node_pos $node_($i) 50
     }
#telling nodes when the simulation ends
for {set i 0} { $i<$val(nn) } {incr i } {
$ns_ at $val(SimTime) "$node_($i) reset";
}
#procedure finish
proc finish {} {
global ns_ tracefd namtrace
$ns_ flush-trace
close $tracefd
close $namtrace
}
#End the nam and the simulation
$ns_ at $val(SimTime) "$ns_ nam-end-wireless $val(SimTime)"
$ns_ at $val(SimTime) "finish"
$ns_ at 100.00 "puts \"Simulation Ended Successfully!\";$ns_ halt"
```

```
$ns_ run
```

B: SAMPLE MOBILITY FILE

```
#
# nodes: 50, pause: 10.00, max speed: 10.00, max x: 500.00, max y:
500.00
#
$node_(0) set X_ 284.911123747359
$node_(0) set Y_ 96.017174648953
$node_(0) set Z_ 0.000000000000
$node_(1) set X_ 16.149083818078
$node_(1) set Y_ 197.946792508693
$node_(1) set Z_ 0.000000000000
$node_(2) set X_ 93.861403996863
$node_(2) set Y_ 393.435163268474
$node_(2) set Z_ 0.000000000000
$node_(3) set X_ 192.847414317602
$node_(3) set Y_ 464.577752079855
$node_(3) set Z_ 0.000000000000
$node_(4) set X_ 266.233755712542
$node_(4) set Y_ 188.691236718506
$node_(4) set Z_ 0.000000000000
$node_(5) set X_ 187.511518830991
$node_(5) set Y_ 488.404601716473
$node_(5) set Z_ 0.000000000000
$node_(6) set X_ 82.795100920224
$node_(6) set Y_ 143.433809940292
$node_(6) set Z_ 0.000000000000
$node_(7) set X_ 403.744814214046
$node_(7) set Y_ 104.064260061217
$node_(7) set Z_ 0.000000000000
$node_(8) set X_ 55.572172989839
$node_(8) set Y_ 268.129919020232

….

….

..
```

```
$ns_ at 27.795340465608 "$node_(22) setdest 459.644860030649
132.961419214489 0.000000000000"

…..

$ns_ at 29.372535971717 "$node_(6) setdest 159.486756351135
3.693976618430 0.000000000000"

….

$ns_ at 34.592046377670 "$node_(44) setdest 496.418370109755
78.354377252448 0.000000000000"
$ns_ at 34.624255600915 "$god_ set-dist 4 42 2"
$ns_ at 34.708455705808 "$node_(19) setdest 235.952039849484
257.014688676934 0.000000000000"

…

$ns_ at 99.950738880934 "$node_(40) setdest 466.300501951600
390.037889505709 0.000000000000"


.....
```

C: SAMPLE TRAFFIC SCENARIO FILE

```
#
# nodes: 48, max conn: 20, send rate: 0.25, seed: 1.0
#
# 1 connecting to 2 at time 2.5568388786897245
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
```

```
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
#
# 4 connecting to 5 at time 56.333118917575632
#
set udp_(1) [new Agent/UDP]
$ns_ attach-agent $node_(4) $udp_(1)
set null_(1) [new Agent/Null]
$ns_ attach-agent $node_(5) $null_(1)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 512
$cbr_(1) set interval_ 0.25
$cbr_(1) set random_ 1
$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(1)
$ns_ at 56.333118917575632 "$cbr_(1) start"

….......

# 17 connecting to 19 at time 19.655724884781858
#
set udp_(18) [new Agent/UDP]
$ns_ attach-agent $node_(17) $udp_(18)
set null_(18) [new Agent/Null]
$ns_ attach-agent $node_(19) $null_(18)
set cbr_(18) [new Application/Traffic/CBR]
$cbr_(18) set packetSize_ 512
$cbr_(18) set interval_ 0.25
$cbr_(18) set random_ 1
$cbr_(18) set maxpkts_ 10000
$cbr_(18) attach-agent $udp_(18)
$ns_ connect $udp_(18) $null_(18)
$ns_ at 19.655724884781858 "$cbr_(18) start"
#
# 20 connecting to 21 at time 170.32769159894795
#
set udp_(19) [new Agent/UDP]
$ns_ attach-agent $node_(20) $udp_(19)
set null_(19) [new Agent/Null]
$ns_ attach-agent $node_(21) $null_(19)
```

```
set cbr_(19) [new Application/Traffic/CBR]
$cbr_(19) set packetSize_ 512
$cbr_(19) set interval_ 0.25
$cbr_(19) set random_ 1
$cbr_(19) set maxpkts_ 10000
$cbr_(19) attach-agent $udp_(19)
$ns_ connect $udp_(19) $null_(19)
$ns_ at 170.32769159894795 "$cbr_(19) start"
#
#Total sources/connections: 13/20
#
```

D : C++ CODES FOR FLOODING ATTACK

Modification to *aodv.h* file

```
///added class FloodTimer

class FloodTimer : public Handler {
friend class AODV;
public:
FloodTimer(AODV* a):    agent(a){}
void     handle(Event*);
private:
AODV     *agent;
Event     intr;
};

//Some modifications are also added to class AODV

class AODV: public Agent {
  /*
   * make some friends first
   */
     // added this to the friends list

        friend class FloodTimer;
// declared the following member functions

     void      RREQ_flood();
//to flood RREQ
```

```
     int *   DToBinary(int , int);//decimal to binary
     int ToDecimal(int [],int);//binary to decimal

//declared some member varibales

               bool   intruder;  //to identify if there is an
attacker
     bool internal_attacker;
     nsaddr_t   attack_dst;   // attacking node
     double flood_interval;  //interval at which RREQ is flooded
     int  src_bin[16];
     int key1;
     int key2;
//then declare a timer

             FloodTimer     ftimer;
```

Modifications to *aodv.cc* file

```
//added the follow following to the command function to assign
attacker nodes from TCL codes.

if(argc == 2) {

…...

if(strcmp(argv[1], "intruder") == 0) {
        intruder = true;
        return TCL_OK;
     }
   if(strcmp(argv[1], "internal_attacker") == 0) {
        internal_attacker = true;
        return TCL_OK;
     }

…...

else if(argc == 3) {

….

      if(strcmp(argv[1], "intruder") == 0) {
            intruder = true;
           attack_dst = atoi(argv[2]);
             return TCL_OK;
```

```
     }
     if(strcmp(argv[1], "internal_attacker") == 0) {
                    internal_attacker = true;
                    attack_dst = atoi(argv[2]);
                return TCL_OK;
     }

…..

else if(argc == 4) {
       if(strcmp(argv[1], "intruder") == 0) {
                intruder = true;
       attack_dst = atoi(argv[2]);
       flood_interval=atof(argv[3]);
          return TCL_OK;
        }
       if(strcmp(argv[1], "internal_attacker") == 0) {
                internal_attacker = true;
       attack_dst = atoi(argv[2]);
       flood_interval=atof(argv[3]);
                return TCL_OK;
       }
//the following initializations added to the constructor of AODV

    attack_dst=1999;
    intruder=false;
   internal_attacker=false;
   flood_interval=0.1;

…...

//the definition of member functions

void FloodTimer::handle(Event*) {
/*attacker and the flood_interval are set at the tcl code Or they
can simply take the default values
*/
if((agent->intruder==true) || (agent-> internal_attacker==true))
/* agent->flood() is run for any node specified as attacker*/
agent->RREQ_flood();
Scheduler::instance().schedule(this, &intr, agent->flood_interval);
}
```

```
//***********************************************

//Send rreq flood to unknown destination
void AODV::RREQ_flood() {
     Packet *p = Packet::alloc();
     struct hdr_cmn *ch = HDR_CMN(p);
     struct hdr_ip *ih = HDR_IP(p);
     struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
     aodv_rt_entry *rt = rtable.rt_lookup(attack_dst);
     // Fill out the RREQ packet
     // ch->uid() = 0;
              ch->ptype() = PT_AODV;
              ch->size() = IP_HDR_LEN + rq->size();
               ch->iface() = -2;
             ch->error() = 0;
            ch->addr_type() = NS_AF_NONE;
            ch->prev_hop_ = index;


          ih->saddr() = index;
          ih->daddr() = IP_BROADCAST;
          ih->sport() = RT_PORT;
          ih->dport() = RT_PORT;
          ih->ttl_ = NETWORK_DIAMETER;
       // Fill up some more fields.
        rq->rq_type = AODVTYPE_RREQ;
         rq->rq_hop_count = 1;
       rq->rq_bcast_id = bid++;
        rq->rq_dst = attack_dst;
         rq->rq_dst_seqno = (rt ? rt->rt_seqno : 0);
          rq->rq_src = index;
         seqno += 2;
        assert ((seqno%2) == 0);
       rq->rq_src_seqno = seqno;
         rq->rq_timestamp = CURRENT_TIME;
//internal attacker will have the keys
if(internal_attacker) {
/* change src address to binary, in src_bin[16]*/
nsaddr_t src=rq->rq_src;
int* ptr=DToBinary(src, 16);
/*flip a bit of the src addr using key1 */
```

```
src_bin[16-key1]=src_bin[16-key1]^1;
/*change back to deciaml*/
int dec=ToDecimal(src_bin,16);
/*modulo operation with key2*/
dec=dec%key2;
/*Embeded the modified key*/
//rq->reserved[0]=dec;
rq->reserved[0]=(char)dec;
}
Scheduler::instance().schedule(target_, p, 0.);
}

//*****************************************************
//retruns a decimal equivalent
int AODV::ToDecimal(int b[],int size)
{
int bin,dec=0,base=1,rem,i=0;
while(i<size)
{
rem=src_bin[size-1-i];
dec=dec+rem*base;
base=base*2;
i++;
}
return dec;
}
//************************************
//returns binary eqv of n
int * AODV::DToBinary(int n,int size)
{
     int i;
            int a[size];
     do
     {
     for(i=0;i<size;i++)
     {
     a[i]=n%2;
     n=n/2;
     }
     }
     while((n%2)!=0);
```

```
            //reverse the binary, LSB at the right side
       for(i=0;i<size;i++)
        src_bin[i]=a[size-1-i];
      return ((int*)src_bin);
}
```

E: C++ CODES FOR PREVENTION MECHANISM

//The modifications done to the flooding attack, in appendix D, also hold for prevention mechanism.

//the declarations and definitions in the flooding attacks are also part of this.

Modifications to *aodv_rtable.h* file

```
//definitions of classes

//*********************************************************
//table to cache rreq counts
class RREQ_count{
    friend class AODV;
        friend class aodv_rt_entry;
public:
    RREQ_count(u_int32_t c) {ct_addr=c; RREQentry=0;}
protected:
    LIST_ENTRY(RREQ_count) ct_link;
    u_int32_t   RREQentry;
    nsaddr_t    ct_addr;
    double          ct_expire;
};
LIST_HEAD(RREQ_count_cache, RREQ_count);
//---------------------------------------------------------
class Blacklist{
    friend class AODV;
      friend class aodv_rt_entry;
    public:
        //Blacklist(u_int32_t e){bd_addr=e;}
    protected:
        LIST_ENTRY(Blacklist) bd_link;
        nsaddr_t    bd_addr;
};
LIST_HEAD(Blacklist_cache, Blacklist);
```

Modifications to *aodv.h* file

```
//to purge the count table

//--------------------------------------------------------------------
--------
class CacheTimer : public Handler {
public:
    CacheTimer(AODV* a):    agent(a){}
    void    handle(Event*);
private:
    AODV    *agent;
    Event   intr;
};
//--------------------------------------------------------------------
---

//added the following to the friend list

friend class CacheTimer;

….

//declared the following member functions and variables

          void        ct_add(nsaddr_t id);
          void            ct_remove(nsaddr_t id);
         void           ct_flush(void);
        Blacklist * bd_lookup(nsaddr_t id);

        CacheTimer  ctimer;

…
```

Modifications to *aodv.cc* file

```
     ///added the schedule for a timer

void CacheTimer::handle(Event*) {
agent->ct_flush();
//check this definition
```

```
#define CAHCE_INTERVAL 1
  Scheduler::instance().schedule(this, &intr,  CAHCE_INTERVAL);
}

//the recvRequest function is modified as follows

void
 AODV::recvRequest(Packet *p) {
...

…

Blacklist *bd;
bd=bd_lookup(rq->rq_src);
if(!bd) //if not in black list table
{
//---------------------------------------------------------------
//check if it an external intruder
     int* ptr=DToBinary(rq->rq_src, 16);
     src_bin[16-key1]=src_bin[16-key1]^1;
     /*change back to deciaml*/
     int dec=ToDecimal(src_bin,16);
     //modulo
     dec=dec%key2;
     if(rq->reserved[0]!=(char)dec)
     {
          Blacklist *bd1 = new Blacklist();
                  bd1->bd_addr=rq->rq_src;
                  LIST_INSERT_HEAD(&bdhead, bd1, bd_link);
               Packet::free(p);
          return;
     }
//---------------------------------------------------------------

       else //for internal attacker
     {
          AODV_Neighbor *nb;
             //aodvplain_RREQcount *ct;
             nb = nb_lookup(rq->rq_src);
             if((nb) &&  (rq->rq_hop_count == 1) )
             {
                 double now = CURRENT_TIME;
```

```
                RREQ_count *ct = rt_ctlist.lh_first;
                //aodvplain_RREQcount *ct2;
                for(; ct; ct = ct->ct_link.le_next)
                {
                     if(ct->ct_addr == rq->rq_src)
                      {
                      if((ct->ct_expire <= now) && (ct->RREQentry
>= RATE_LIMIT)  && (ct->RREQentry < BLACKLIST_LIMIT))
                      {
                           rqueue.enque(p);
                           return;
                       }
                            else if((ct->ct_expire <= now) && (ct-
>RREQentry >= BLACKLIST_LIMIT) )
                             {


                                 Blacklist *bd1 = new Blacklist();
                                 bd1->bd_addr=ct->ct_addr;
                                 LIST_INSERT_HEAD(&bdhead, bd1,
bd_link);
                                 LIST_REMOVE(ct,ct_link);
                                 delete ct;
                           Packet::free(p);
                                    return;
                                               break;
                              }
                             else if (ct->ct_expire<=now)
                              {

                                 LIST_REMOVE(ct,ct_link);
                                 delete ct;
                                 break;
                              }
                             ct->RREQentry= ct->RREQentry+1;
                             break;

                       }//if (bellow for)
                }//for
          if(!ct)//if RREQ cache is empty
          {
```

```
                    RREQ_count *ct1 = new RREQ_count(rq->rq_src);
                    ct1->ct_expire = CURRENT_TIME + 1;
                    ct1->RREQentry= ct1->RREQentry+1;
                    LIST_INSERT_HEAD(&rt_ctlist, ct1, ct_link);
                }
        }//if (nb) &&
   }//else
}//if(!bd)
else
{
//The src node is in black list table, simply drops its request
   Packet::free(p);
   return;
}
…..

}

//// Modification added to SendRequest function

void
 AODV::sendRequest(nsaddr_t dst) {

…

if(!intruder) {
/* change src address to binary, in src_bin[16]*/
nsaddr_t src=rq->rq_src;
int* ptr=DToBinary(src, 16);
/*flip a bit of the src addr using key1 */
src_bin[16-key1]=src_bin[16-key1]^1;
/*change back to deciaml*/
int dec=ToDecimal(src_bin,16);
/*modulo operation with key2*/
dec=dec%key2;
/*Embeded the modified key*/
rq->reserved[0]=dec;
}

….

}
```

```
//some additional definitions of member functions

void AODV::ct_flush() {
RREQ_count *ct = rt_ctlist.lh_first;
RREQ_count *ct1;
double now = CURRENT_TIME;
 for(; ct; ct =ct1 ) {
ct1=ct->ct_link.le_next;


   if(ct->ct_expire <= now) {

 LIST_REMOVE(ct,ct_link);
     delete ct;
}
}
}
//--------------------------------------------------
Blacklist* AODV::bd_lookup(nsaddr_t id)
{
Blacklist *bd = bdhead.lh_first;
for(;bd;bd=bd->bd_link.le_next){
        if(bd->bd_addr ==id)
        break;
    }
return bd;
}

//**********************************************************
```

F: AWK AND GNUPLOT SCRIPTS

Gnuplot

```
#!/usr/bin/gnuplot
reset
#set terminal png
set term png size 600, 400
set output "e2edelay.png"
set xlabel "RREQ/s-No of attackers"
```

```
set ylabel "Ave E2E delay (ms) "
 set key at 7,2200;
set grid
set xtics (0,4,10,16)
#set style data linespoints
plot 'RREQ_Flooding' with linespoints pointtype 1 pointsize 1
linetype 5 linecolor 1, \
   'Filter' with linespoints pointtype 2 pointsize 1 linetype 2
linecolor 2, \
 'New_All_EX' with linespoints pointtype 3 pointsize 1 linetype 3
linecolor 3, \
'New_Half_EX' with linespoints pointtype 4 pointsize 1 linetype 4
linecolor 4;
```

Awk Scripts

Average end-to-end delay

```
# delay.awk
 BEGIN {
         for (i in send) {
                send[i] = 0
         }
        for (i in recv) {
                 recv[i] = 0
        }
         delay = avg_delay = 0
 }
 {
         # Trace line format: normal
         if ($2 != "-t") {
                 event = $1
                 time = $2
                 if (event == "+" || event == "-") node_id = $3
                 if (event == "r" || event == "d") node_id = $4
                 flow_id = $8
                 pkt_id = $12
         }
         # Trace line format: new
         if ($2 == "-t") {
```

```
                event = $1
                time = $3
                node_id = $5
    trace_level=$19;
     pkt_type=$35;
                flow_id = $39
                pkt_id = $41
        }
        # Store packets send time
        if (flow_id == 0 &&   send[pkt_id] == 0 && trace_level ==
"AGT" && pkt_type=="cbr" && (event == "+" || event == "s")) {
                send[pkt_id] = time
            }
        # Store packets arrival time
        if (flow_id == 0 && trace_level == "AGT" && pkt_type=="cbr"
&& event == "r") {
                recv[pkt_id] = time
             }
 }
 END {
        # Compute average delay
        for (i in recv) {
                if (send[i] == 0) {
                       # printf("\nError %g\n",i)
                }
                delay += recv[i] - send[i]
                num ++
        }
                if (num != 0) {
                avg_delay = delay / num
        } else {
                avg_delay = 0
        }
        print avg_delay*1000;
 }
```

Average energy left

```
BEGIN{
     #highest_packet_id = -1;
             n=0;
          totEnergy = 0.0;
              for (i in Renergy) {
              Renergy[i] = 0.0;
          }

     }
       {
         # Trace line format: new
     # if($1=="N")
           {
          time = $3;
          node = $5;
          energy = $7;
         }
         if ( $1!="N")
           {
                 event = $1;
                 time = $3;
                 node = $9;
      energy = $17;
            }
         #hold the remaining energy of the respective node
         if(time > 95)
           {
          Renergy[node]=energy;
   }
     }
END  {
     for (i in Renergy) {
          n++;}
     for (i in Renergy) {
          totEnergy += Renergy[i];}
      for (i in Renergy) {
     #    printf("node %d \t Energy %.6f \n", i, Renergy[i]);
```

```
        }
     print  totEnergy/n
    }
```

PDR

```
BEGIN {
        sent = 0;
        rcvd = 0;
        #fwd = 0;
}
{
     # Trace line format: normal
         if ($2 != "-t") {
                 event = $1;
                 time = $2;
                 if (event == "+" || event == "-") node_id = $3;
                 if (event == "r" || event == "d") node_id = $4;
                 flow_id = $8;
                 pkt_id = $12;
                 pkt_size = $6;
         }
         # Trace line format: new
         if ($2 == "-t") {
                 event = $1;
                 time = $3;
                 node_id = $5;
                 trace_level=$19;
                 source= $31;
                 destination= $33;
                 pkt_type=$35;
                 seq_num=$47;
                 flow_id = $39;
                 pkt_id = $41;
                 pkt_size = $37;
         }
     #received
     if (( event == "r") && ( pkt_type == "cbr" || pkt_type =="tcp"
) && ( trace_level=="AGT" ))
     rcvd++;
     #sent
```

```
     if ((event == "s" ) && ( pkt_type == "cbr" || pkt_type =="tcp"
) && ( trace_level=="AGT" ))
     sent++;
     }
     END {
     print  rcvd/sent;
     }
```

Routing load

```
BEGIN{
     data_pkt = 0;#data pkts recieved
     routing_pkt = 0;# routing packets send or forwarded
     }

     {
      # Trace line format: normal
         if ($2 != "-t") {
                 event = $1;
                 time = $2;
                 if (event == "+" || event == "-") node_id = $3;
                 if (event == "r" || event == "d") node_id = $4;
                 flow_id = $8;
                 pkt_id = $12;
                 pkt_size = $6;
         }
         # Trace line format: new
         if ($2 == "-t") {
                 event = $1;
                 time = $3;
                 node_id = $5;
                 trace_level=$19;
                 source= $31;
                 destination= $33;
                 pkt_type=$35;
                 seq_num=$47;
                 flow_id = $39;
                 pkt_id = $41;
                 pkt_size = $37;
         }
     ##### a data packet
```

```
     if (( event == "r") && (pkt_type == "cbr" || pkt_type =="tcp" )
&& (trace_level=="AGT" ))
     data_pkt++;
     ##### routing packet
     if ((event == "s" || event == "f") && trace_level == "RTR" &&
pkt_type =="AODV")
     routing_pkt++;

     }
     END {
     print  routing_pkt/data_pkt;
     }
```

……..//……