Enforcing Reliable Routing in Ad-Hoc Networks

BY

Ghada Vaseem

11MICT05

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MAY 2013

Enforcing Reliable Routing in Ad-Hoc Networks

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Information and Communication Technology

BY

Vaseem Ghada

11MICT05

GUIDED BY

Prof. Sharada Valiveti

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MAY-2013

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Information and Communication Technology at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgement has been made in the text to all other material used.

Ghada Vaseem

Certificate

This is to certify that the Major Project entitled " Enforcing Reliable Routing in Ad-Hoc Networks" submitted by Ghada Vaseem (11MICT05), towards the partial fulfillment of the requirements for the degree of Master of Technology in Information and Communication Technology of Nirma University, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part-II, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Sharada Valiveti
Guide, Associate Prof.
Department of C.S.E.,
Institute of Technology,
Nirma University, Ahmedabad.

Prof. Gaurang Raval
Associate Prof. and PGICT-Coordinator,
Department of C.S.E,
Institute of Technology,
Nirma University, Ahmedabad.

Prof. Sanjay Garg
Professor and Head,
Department of C.S.E.,
Institute of Technology,
Nirma University, Ahmedabad.

Dr Ketan Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad.

Abstract

An Ad Hoc network is the network of self-configuring nodes without having fixed infrastructure. Each node acts as a system and router. Many of the routing protocols of Ad Hoc network are designed based on the assumption that every node forwards every packet but practically many of them act as selfish nodes, they use network and its service but dont cooperate with other nodes so as to save resources for themselves. This report discusses the types of availability attack, malicious activity of selfish node, a Survey of techniques used to detect selfishness attack and some approach to detect selfishness attack. Here i have implemented the selfishness attack and analyze its effect on the Packet delivery ratio, latency, throughput. I have also implemented and designed the credit based algorithm to detect and overcome the activity of selfish nodes. I have checked the effect of mobility and number of active connection on the PDR, Throughput and Delay.

Acknowledgements

My deepest thanks to **Prof. Sharada Valiveti**, Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad the Guide of the project that I undertook for giving her valuable inputs and correcting various documents of mine with attention and care. She has taken the pain to go through the project and make necessary amendments as and when needed.

My deep sense of gratitude to **Prof. Gaurang Raval**, PGICT-Coordinator of Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad for an exceptional support and continual encouragement throughout part one of the Major project.

I would like to thanks **Prof. Sanjay Garg**, Hon'ble Head of Department, Institute of Technology, Nirma University, Ahmedabad for his unmentionable support, providing basic infrastructure and healthy research environment.

I would like to thanks **Dr Ketan Kotecha,** Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for his unmentionable support, providing basic infrastructure and healthy research environment.

I would also thank my Institution, all my faculty members in Department of Computer Science and my colleagues specially Pratiti Mankodi without whom this project would have been a distant reality. Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the first part of my dissertation work successfully.

- Ghada Vaseem

11MICT05

Abbreviations

AODV	Ad hoc On-Demand Distance Vector Routing Protocol
IDS	Intrusion Detection System
RREQ	Route Request
RREP	Route Reply
RRER	Route Error
MANETs	Mobile Ad-hoc Networks
CORE	Collaborative Reputation
AODV	Adhoc On-demand Distance Vector
CONFIDANT	Cooperation Of Nodes and Fairness In Dynamic Ad-hoc Network
OCEAN	Observation based Co-operation enforcement in Ad hoc Networks
SORI	Secure and Objective Reputation based Incentive
NS	Network Simulator
PDR	Packet Delivery Ratio
PLR	Packet Loss Rate
CBR	Constant Bit Rate

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Objective of the Work

- To implement and analyze effect of availability attack on Ad hoc networks and implement detection methods.
- Develop a reliable trust based routing protocol which works in the presence of Selfish node.

1.2 Scope of the Work

- Analysis of different availability attacks
- Focus on selfishness attack
- Finding Detection methods for this attack
- Defining a reliable routing protocol using this tables
- Analysis

1.3 Motivation of the Work

- Ad hoc networks are vulnerable for many attacks like Black hole, fabricated route, Resource consumption and Selfishness. It is highly necessary to secure the routing of the Ad hoc network, which inspired me to develop detection methods for Selfishness[3].

Chapter 2

Literature Survey

An Ad Hoc network is the network of self-configuring nodes without having fixed infrastructure. Each node acts as a system and router. Many of the routing protocols of Ad Hoc network are designed based on the assumption that every node forwards every packet but practically many of them act as selfish nodes, they use network and its service but dont cooperate with other nodes so as to save resources for themselves. This report discusses the types of availability attack, malicious activity of selfish node, a Survey of techniques used to detect selfishness attack and some approach to detect selfishness attack. Here i have implemented the selfishness attack and analyze its effect on the Packet delivery ratio, latency, throughput. I have also implemented and designed the credit based algorithm to detect and overcome the activity of selfish nodes. I have checked the effect of mobility and number of active connection on the PDR, Throughput and Delay.

2.1 Routing in Ad-Hoc Networks

Using limited recourses, routing helps to find and maintain routes between nodes in dynamic topology with preferably uni-directional links.

2.1.1 Pro-active (table-driven) Routing Protocols

This type of protocols maintains fresh lists of destinations and their routes by periodically distributing routing tables throughout the network[4]. The main disadvantages of such algorithms are:

- Respective amount of data for maintenance
- Slow reaction on restructuring and failures

Table-driven protocols are: Destination-Sequenced Distance Vector Routing Protocol(DSDV),Wireless Routing Protocol(WRP),Global State Routing(GSR),Hierarchical State Routing(HSR),Zone-based Hierarchical Link State Routing Protocol(ZHLS) and Clusterhead Gateway Switch Routing Protocol(CGSR)[4].

2.1.2 Reactive (on-demand) Routing Protocols

This type of protocol creates on demand routes because this type of protocols finds a route on demand by overflowing the network with route request packets[4]. The main disadvantages of such algorithms are:

- High lead time in route finding
- Excessive overflowing can lead to network jam

On-demand Protocols are: Ad-Hoc On-Demand Distance Vector Routing (AODV), Cluster based Routing Protocol (CBRP), Dynamic Source Routing (DSR), Temporally Ordered Routing Algorithm (TORA)[4].

2.2 Ad-Hoc On-Demand Distance Vector Routing Protocol (AODV)

The Ad-hoc On-demand Distance Vector (AODV) routing protocol is a routing protocol used for dynamic wireless networks where nodes can enter and leave the network[5].

The Ad hoc On Demand Distance Vector (AODV) routing algorithm is a routing protocol designed for dynamic wireless networks[5]. As the name suggest AODV builds routes between nodes as per the wish of source code. AODV is capable of both unicast and multicast routing. These routes are maintained by the time it is required by source node. Additionally, AODV is capable of forming trees to connect multicast group member with nodes.

The source node transmits a Route Request (RREQ) to its immediate neighbors to find route to a particular destination node. The neighbor replies back with Route Reply (RREP) if the neighbor has a route to the destination. Otherwise the neighbors in turn rebroadcast the request. This continues until the RREQ hits the final destination or a node with a route to the destination. At that point a chain of RREP messages is sent back and the original source node finally has a route to the destination.

Advantages:

- Here routes are established on demand and the latest route to the destination is found based on sequence number.. So the connection setup delay is lower.

Disadvantages:

- Here, if source code sequence number is very old, the intermediate nodes may follow inconsistent route and the intermediate nodes have higher but not the latest sequence number leads to stale entries.
- Multiple Route Reply packets in response to a single Route Request packet can lead to heavy control overhead.
- Periodic beaconing leads to unnecessary bandwidth consumption.

2.3 Vulnerabilities of Ad Hoc Networks

- Nodes of mobile ad hoc networks have limited ranges and because of that it requires multi hop communication. Ad hoc network runs on an assumption that once the node has promised to transmit the packet, it will not cheat but this does not holds true when nodes in the networks have contradictory goals. Due to this, neighbors of intermediate nodes can use the reputation of intermediate nodes to transmission.

- Node mobility leads to frequent change in network topology
- Use of wireless links into network increases the risk of link attacks
- Relatively poor protection
- Long life of network requires distributed architecture
- Risk of Denial of Service (DoS) attacks due to lack of infrastructure and chances of link breakage and channel errors due to mobility
- Need of scalability
- Nodes in Ad Hoc Networks have limited services and security provision due to limited memory and computational power
- Dynamic topology

2.4 Types of Attacks

Attacks on networks come in many varieties and they can be grouped based on different characteristics.

a. Availability Attacks

   Availability is the most basic requirement of any network. If the networks connection ports are unreachable, or the data routing and forwarding mechanisms are out of order, the network would cease to exist[5].

- Packet Dropping or Black-hole Attack[5]: In mobile ad hoc networks (MANETs), nodes usually cooperate and forward each other's packets in order to enable out of range communication. However, in hostile environments, some nodes may deny to do so, either for saving their own resources or for intentionally disrupting regular communications. This type of misbehavior is generally referred to as packet dropping attack or black hole attack.
- Fabricated route Attack[5]: Fabrication attacks generate false routing messages. Such attacks can be difficult to confirm as invalid constructs, especially in the case of fabricated false messages that claim a neighbor cannot be contacted.
- Resource Consumption Attack[5]: In this attack, a malicious node intentionally tries to consume the resources (e.g. battery power, bandwidth etc) of other nodes in the network. The attack can be of various types like unnecessary route requests, route discovery, control messages, or by sending stale information.
- Selfishness Attack[5]: Selfish and malicious nodes participate in route discovery stage properly to update their routing table, but as soon as data forwarding stage begins, they discard data packets.

b. Confidentiality Attacks

Confidentiality describes the need to protect the data roaming in the network from being understood by unauthorized parties. Essential information is encrypted to achieve confidentiality. By which, only the communicating nodes can analyze and understand it.

- Location Disclosure Attack[5]: A location disclosure attack can reveal something about the locations of nodes or the structure of the network. The information gained might reveal which other nodes are adjacent to the target, or the physical location of a node. Routing messages are sent with inadequate hop-limit values and the addresses of the devices sending the ICMP error-messages are recorded. In the end, the attacker knows which nodes are situated on the route to the target node. If the locations of some of the intermediary nodes are known, one can gain information about the location of the target.

- Content Disclosure Attack[5]: The content disclosure attack enables a malicious attacker to learn the contents of messages being transmitted.

c. Authenticity Attacks

Authenticity is crucial to keep eavesdroppers out of the network. With many services applicable in ad hoc networks, it is important to ensure that when communicating with a certain node, that node is really who/what we expect it to be (node authentication). Message authentication ensures that the contents of a message are valid.

d. Integrity Attacks

Integrity of communication data helps to ensure that the information passed on between nodes has not been altered in any way. Data can be altered by two ways- intentionally and accidentally (for example through hardware glitches, or interference in the case of wireless connections).

- False Route Propagation Attack[5]: In this attack, a malicious node advertises a route to a node with a destination sequence number greater than the authentic value. By doing this, it diverts the traffic towards the attacker because the nodes will select the RREP with the highest destination sequence number.
- Man-in-the-Middle Attack[5]: In this attack, a malicious node reads and possibly modifies the messages between two parties. The attacker can impersonate the receiver with respect to the sender, and the sender with

respect to the receiver, without having either of them realize that they have been attacked.

- Misrouting Attack[5]: In this class of attacks, a malicious node attempts to send a data packet to the wrong destination. For example, this can be achieved by forwarding a data packet to the wrong next hop in the route to the destination or by modifying the final destination address of the data packet.

e. Non-repudiation

Non-repudiation refers to the capability to guarantee that a party cannot deny the authenticity of their signature on a document or the sending of a message that they created.

2.5 IDS Schemes for Selfishness Attack

This IDS Schemes deal with problem of Selfishness on packet forwarding in MANET.

a. End-to-end Acknowledgements[3]:
This mechanism consists of monitoring the reliability of routes by acknowledging packets in an end-to-end manner, to render the routing protocol reliable. In this, the destination node gives acknowledgement of receipt of packets by sending a feedback to the source.
Advantages:

- Helps to avoid sending packets through unreliable routes and it can be combined with other technique.

Disadvantages:

- Lack of misbehaving node detection.

- This technique may detect routes containing misbehaving or malicious nodes and those which are broken, but without any further information regarding node causing packet loss.

b. Two-hop Acknowledgements[6]:

This scheme uses asymmetric cryptography.

Advantages:

- Mitigate Watchdog's problem related to power control technique usage.

c. Watchdog[3]:

It aims to detect misbehaving nodes that don't forward packets, by monitoring neighbors in the promiscuous mode. The solution also includes path-rater component, that selects route based on the link reliability knowledge.

Advantages:

- It is able to detect misbehaving nodes in many cases, and requires no overhead when no node misbehaves.

Disadvantages:

- It fails to detect misbehavior in cases of collisions, partial collusion and power control employment.
- It fails when two successive nodes collude to conceal the misbehavior of each other.
- It doesn't control detected misbehaving nodes.

d. PathRater[3]:

To check reliability of each path in the network, each node is preloaded with path rater. It gives the rate to path by averaging the reputation of each node of that path. If there are multiple path to reach destination in network, the path which has highest rate is selected for transmission of packet.

e. ABO (activity-based overhearing)[5]:

It is a generalization of Watchdog.

Advantages:

- Node constantly monitors in promiscuous mode the traffic activity of all its neighbors and oversees the forwarding of each packet whose next forwarder is also in its neighborhood. This can increase the number of observations and improve watchdog efficiency.
- It mitigates collusion problem.

f. Probing[7]:

It is a combination of route and node monitoring.This approach consists of simply incorporating into data packets commands to acknowledge their receipt. These commands are called probes and intended for selected nodes.Probes are launched when a route that contains a misbehaving node is detected.

Disadvantages:

- A selfish node could analyze each packet it receives before deciding either to forward this packet or not. When it gets a probe packet, it would notice that a probing is under way and would consequently choose to cooperate and forward packets for a limited time, until the probe is over.

g. Friends and foes[3]:

In this, nodes are permitted to publicly claim that they are unwilling to forward packets to some nodes.

Each node maintains basically three sets.

1.Set of friends-to which it is willing to provide services.

2.Set of foes-to which it is unwilling to provide services.

3.Set of nodes-known to act as if it is their foe(they don't provide service packets for it)named set of Selfish.

Advantages:

- It is used to secure control packet from dropping.

Disadvantages:

- Watchdog's problems remain same.
- More overhead
- Each node only keeps information about its current neighbors and information of nodes leaving its neighborhood are begun, a mobile selfish can easily avoid and would never be detected.

h. Nuglets[3]:
Each node spend nuglets (virtual currency) for utilizing the network services.
Nuglets are represented by counter at nodes and they are managed by temper restitant hardware module. Only this module can directly perform operation on this counter.
Nuglets are managed by two model PPM and PTM .
advantages:

- It enforce node to cooperate in forwarding mechanism

i. SPRITE[3]:
In this method the virtual currency is managed by CCS (credit clearance services)
No need of temper resistant hardware

j. Ex-Watchdog[5]:
It is implemented with encryption mechanism and maintaining a table that stores entry of source, destination, sum(Total number of packets+ the current node sends+ forwards or receives) and path. It's main feature is ability to discover malicious nodes which can partition the network by falsely reporting

other nodes as misbehaving.
Advantages:

- Solves problem of Watchdog.

Disadvantages:

- Fails when malicious node is on all paths from specific source and destination.

Chapter 3

Study of NS-2 Simulator

After detailed study of different existing simulators, NS-2 was selected as simulator for implementation.

3.1 Network Simulator-2

NS-2 stands for Network Simulator version 2. It is a discrete event simulator for networking research. It is used in the simulation of routing protocol. It works at packet level. It provides substantial support to simulate bunch of protocols like TCP, UDP, FTP, HTTP, AODV and DSR.It simulates wired and wireless network. It is primarily UNIX based. It uses TCL as its scripting language.NS-2 is a standard experiment environment in research community[2].

- otcl: Object-oriented support
- tclcl: C++ and otcl linkage
- Discrete event scheduler
- Data network components

From Figure, the C++ classes of ns-2 network components or protocols are implemented in the subdirectory "ns-2", and the TCL library in the subdirectory of

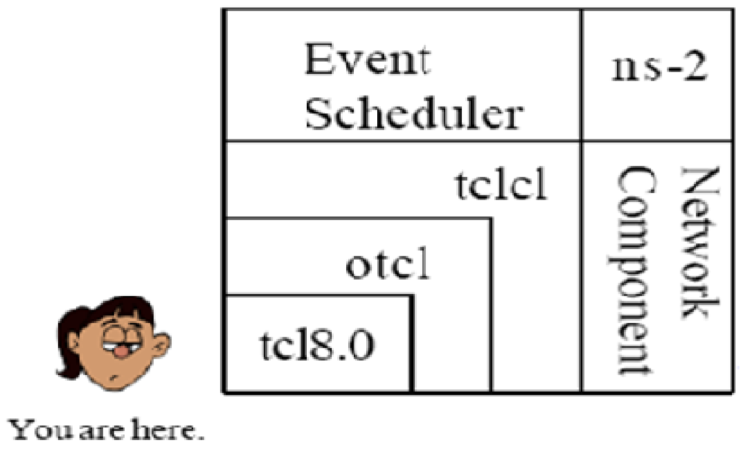


Figure 3.1: NS-2 Directory Structure[2]

"tcl".

- Network Components

Network components are Node, Link, Queue, etc. Some of them are simple components, i.e. they are created from the corresponding C++ classes; others are compound components, which are composed of multiple simple C++ classes, e.g. Links are composed of Delay (emulating propagation delay) and Queue. In general, in ns-2, all network components are created, plugged and configured from TCL.

- Event Scheduling

Events are something associated with time. class Event is defined by time, uid, next, handler, where time is the scheduling time of the event, uid is the unique id of the event, next is the next scheduling event in the event queue that is a link list, and handler points to the function to handle the event when the event is scheduled. Events are put into the event queue sorted by their time, and scheduled one by one by the event scheduler. Note that class Packet is subclass of class Event as packets are received and transmitted at some time. And all network components are subclass of class Handler as they need to handle events such as packets. The event at the head of the event queue is delivered to its hander of some network object. Then, this network object may call other network object, and finally some new events are inserted into the event queue.

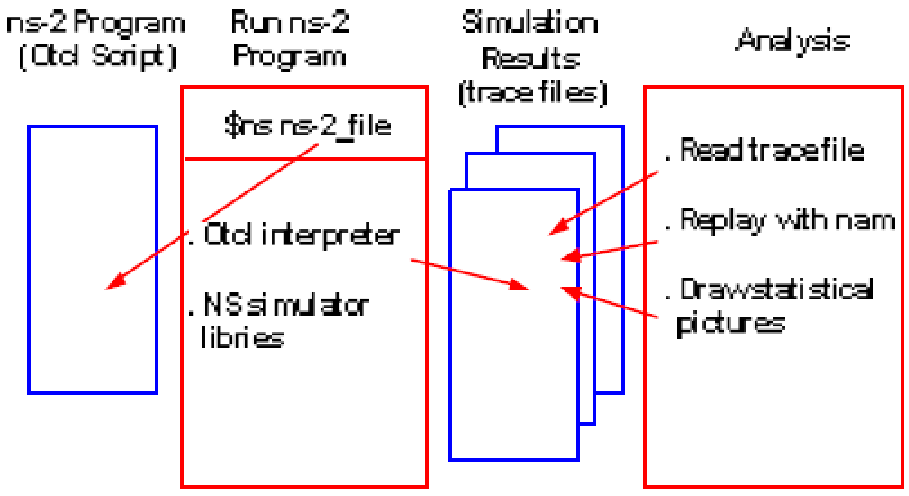


Figure 3.2: Running NS-2 Program[2]

3.1.1 Basic NS-2 program steps

- Create a new simulator object
- Create network (physical layer)
- Create link and queue (data-link layer)
- Define routing protocol
- Create transport connection (transport layer)
- Create traffic (application layer)
- Insert errors

3.2 Example of Simple wireless.tcl file

- **Components of a mobile node :** Link Layer, Interface Queue between LL and Network Layer, MAC Layer Routing Layer
- **Parameters of a mobile node :** Type of antenna, Radio propagation model

```
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(ant) Antenna/OmniAntenna ;# Antenna type
set val(ifq) Queue/DropTail/PriQueue ;# Interface queue type
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802 11 ;# MAC type
set val(rp) AODV ;# ad-hoc routing protocol
set val(nn) 2 ;# number of mobilenodes
```

Creates new instance of 'Simulator'.

```
set ns [new Simulator]
```

Creates new instance of trace file.

set tracefd [open simple.tr w]

$ns trace-all $tracefd

Creates new topology.

set topo [new Topography]

Sets resolution of topology to 500 x 500.

$topo load atgrid 500 500

#Configure nodes $ns node-config

-adhocRouting $val(rp)

-llType $val(ll)

-macType $val(mac)

-ifqType $val(ifq)

-antType $val(ant)

-channelType $val(chan)

-agentTrace ON

-routerTrace ON

-macTrace OFF

-movementTrace OFF

for {set i 0} {$i <$val(nn)} {incr i} {

set node_ ($i) [$ns_ node] $node_ ($i) random-motion 0 }

Random motion is disabled. So provide initial position and movement parameters.

$node_(0) set X_ 5.0

$node_(0) set Y_ 2.0

$node_(0) set Z_ 0.0

$node_(1) set X_ 390.0

$node_(1) set Y_ 385.0

$node_(1) set Z_ 0.0

```
    # Node (1) starts to move towards node (0)
$ns at 50.0 ”$node (1) setdest 25.0 20.0 15.0”
$ns at 10.0 ”$node (0) setdest 20.0 18.0 1.0”


    # Node (1) then starts to move away from node (0)
$ns at 100.0 ”$node (1) setdest 490.0 480.0 15.0”


    # TCP connections between node (0) and node (1)
set tcp [new Agent/TCP]
$tcp set class_2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 10.0 ”$ftp start”


    #Tell nodes when the simulation ends
for {set i 0} {$i <$val(nn)} {incr i} {
$ns_ at 150.0 ”$node_($i) reset”;}
$ns_ at 150.0001 ”stop”
$ns_ at 150.0002 ”puts NS EXITING...””;
$ns_ halt
proc stop { } {
global ns_tracefd
close $tracefd}
```

3.2.1 Generation of Traffic and Movement pattern file

cbr-test is connection-pattern file and scen-test is movement-pattern file.

For setting the value of cbr-test and scen-test, please write following lines.

set val(cp) and set val(sc).

For loading connection and movement pattern file,please write following lines.

source $val(cp) and source $val(sc)

Defines size of node in nam

for {set i 0} {$i <$val(nn)} {incr i} {

$ns_initial_node_pos $node_($i) 20}

- Traffic pattern file

  "cbrgen.tcl" is available at:

  /ns2.34/indep-utils/cmu-scen-gen

  ns cbrgen.tcl [-type cbr/tcp] [-nn nodes] [-seed seed] [-mc connections] [-rate rate] >[filenm]

  -type : type of connection cbr or tcp

  -nn : no. of nodes

  -seed : seed value used by random number generator

  -mc : no. of connections

  -rate : rate at with packets are sent

  Example:

  ns cbrgen.tcl -type cbr -nn 10 -seed 1.0 -mc 8 -rate 4.0>cbr-test

  Generates traffic-pattern and saves in file cbr-test.

- Movement-pattern file

  "setdest" utility available at:

  ns-2.34/indep-utils/cmu-scen-gen/setdest

  In this, there are two versions.

  Version 1:

  ./setdest - v 1 n <nodes>-p <pause time>-M <maxspeed>-t <sim time>-x <maxx>-y <maxy>

  Version 2:

  ./setdest - v 2 n <nodes>-s <speed type>-m <minspeed>-M <max speed>-t <sim time>-P <pause type>-p<pause time>-x <maxx>-y <maxy>

  Speed type: 1 - uniform speed and 2 - normal speed

  Pause type: 1 - constant and 2 uniform

  For e.g. ./setdest v 2 -n 50 -s 1 -m 1 -M 20 -t 200 -P 1 -p 0 -x 200 -y 200 >scen-test

3.3 AWK script

AWK is a programming language that is designed for processing text-based data, either in files or data streams, and was created at Bell Labs in the 1970s. The name AWK is derived from the family names of its authors - Alfred Aho, Peter Weinberger and Brian Kernighan[8].

”AWK is a language for processing files of text. A file is treated as a sequence of records, and by default each line is a record. Each line is broken up into a sequence of fields, so we can think of the first word in a line as the first field, the second word as the second field, and so on. An AWK program is of a sequence of pattern-action statements. AWK reads the input a line at a time. A line is scanned for each pattern in the program, and for each pattern that matches, the associated action is executed.”

Table I: Generalized explanation of trace format[1]

Column Number	What Happened?	Values for instance...
1	It shows the occured event	's' SEND, 'r' RECEIVED, 'D' DRPPED
2	Time at which the event occured?	10.000000000
3	Node at which the event occured?	Node id like 0
4	Layer at which the event occured?	'AGT' application layer, 'RTR' routing layer, 'LL' link layer, 'IFQ' Interface queue, 'MAC' mac layer, 'PHY' physical layer
5	show flags	-
6	shows the sequence number of packets	0
7	shows the packet type	'cbr' CBR packet, 'DSR' DSR packet, 'RTS' RTS packet generated by MAC layer, 'ARP' link layer ARP packet
8	shows size of the packet	Packet size increases when a packet moves from an upper layer to a lower layer and decreases when a packet moves from a lower layer to an upper layer
9	[...]	It shows information about packet duration, mac address of destination, the mac address of source, and the mac type of the packet body.
10	show flags	-
11	[...]	It shows information about source node ip : port number, destination node ip (-1 means broadcast) : port number, ip header ttl, and ip of next hop (0 means node 0 or broadcast).

3.3.1 Running awk script

For running awk script please write

awk -f file1.awk file2.tr >out.txt

where file1.awk is command file.

file2.tr is primary input file and

out.txt is an output file.

Chapter 4

AODV protocol in detail

4.1 Message formats of AODV

AODV have four different messages that it uses for route discovery and route maintenance. All messages are sent using UDP.

4.1.1 Route Request-RREQ

Type: Type of message.
Reserved: Reserved for future use. Currently sent as 0 and ignored on reception.
Hop count: Number of hops from the source IP address to the node handling the request.
Broadcast IP address: IP address of the destination for which a route is required.
Destination sequence number: The last sequence number received in the past by the source for any route towards the destination.
Source IP address: IP address of the node that originated the request.
Source sequence number: Current sequence number for route information generated by the source of the route request.

- Algorithm of Sending RREQ

  RREQ will only be sent by the source nodes (no intermediate node sends

RREQs), if there does not exist any route for the destination.

```
IF (no route exists)
    check-request buffer for requests already sent for destination
    IF (no request sent already)
        create a RREQ packet
        add (dest addr, broadcast ID) to request buffer
        locally broadcast RREQ
        set timer for RREP_WAIT_TIME before rebroadcasting RREQ
        increment broadcast ID
    ELSE
        buffer packet from stream or discard, according to need
    ENDIF
ENDIF
```

- Algorithm of Receiving RREQ

  When a node receives a RREQ, it must first of all decide if it already has processed the RREQ.

  The RREQ is discarded if it has been processed. Otherwise the source address and the broadcast ID from RREQ will be buffered to prevent it from being processed again.

  ```
  IF ((source addr, broadcast ID) in request buffer)
      discard request - already heard and processed
  ELSE
      add (source addr, broadcast ID) to request buffer
  ENDIF
  ```

  The next step is to create or update the route entry in the routing table. This route can be used by the RREP when a route is found.

  ```
  IF (no route to source)
      create a route entry for source addr
  ```

```
ELSE IF (source seqno in RREQ > source seqno in route entry)
update route entry for source addr
ELSE IF ((source seqno in RREQ = source seqno in route entry) AND (hop
count in RREQ< hop count in route entry))
update route entry for source sddr
ENDIF
```

Then , the node must check if it knows the route to the wanted destination. If the node knows the route, it will unicast a RREP to the source. Otherwise it will forward the RREQ.

```
IF (you are destination of RREQ)
create a RREP packet
unicast RREP to source of request
ELSE IF ((have route to destination) AND (destination seqno in route entry
>= destination seqno in RREQ))
create a RREP packet
unicast RREP to source of request
ELSE
forward RREEQ
ENDIF
```

- Algorithm of Forwarding RREQ

  When a node receiving a RREQ that it has not processed yet does not have a route, it will forward the RREQ.

  1. create a RREQ packet
  2. copy all fields from received RREQ into new packet
  3. incremnet hop count field
  4. locally broadcast new RREQ packet
  5. discard received RREQ

4.1.2 Route Reply-RREP

Type: Type of message.

L: If the L-bit is set the message is a hello message and contains a list of the nodes neighbors.

Reserved: Reserved for future use. Currently sent as 0 and ignored on reception.

Hop count: Number of hops from the source IP address to the destination IP address.

Destination IP address: IP address of the destination for which a route is supplied.

Destination sequence number: The destination sequence number associated to the route.

Lifetime: Time for which nodes receiving the Reply consider the route to be valid.

- Algorithm of Forwarding RREP

  When a node receives a RREP that is not addressed for the node, it will set up forward route by updating the table and forward the RREP back to the requesting source.

  ```
  IF (route to requested destination does not exist)
  create a route entry for requested destination
  ELSE IF (destination seqno in RREP > destination seqno in route entry)
  update-route entry for requested destination
  ELSE IF ((destination seqno in RREP = destination seqno in route entry) AND
  (hop count in RREP < hop count in entry))
  update route entry for requested destination
  IF (route to requesting source exists)
  forward RREP to requesting source
  ENDIF
  ENDIF
  ```

- Algorithm of Receiving RREP

  When the originating source receives the RREP it will update the routing table.

```
IF (route to destination does not exist)
create a route entry for destination
ELSE IF (destination seqno in RREP > destination seqno in route entry)
update route entry for destination
ELSE IF ((destination seqno in RREP = destination seqno in route entry) AND
(hop count in RREP < hop count in entry))
update route entry for destination
ELSE
discard RREP
ENDIF
```

4.1.3 Hello Messages

Hello messages are a special case of Route reply messages. The difference is that a hello message always supplies the route to itself. This means that the hop count field is set to 0, the destination address set to the nodes IP address and the destination sequence number set to the nodes latest sequence number.

- Algorithm of Hello handling

  Each node periodically broadcasts a hello message to all neighbors. When a node receives a hello message it knows that the sending node is a neighbor and will update the routing table.

  ```
  IF (route entry for HELLO source exists)
  update route entry
  IF (destination seqno in HELLO > destination seqno in route entry)
  update destination seqno in route entry
  ELSE
  create route entry for HELLO source
  ENDIF
  ENDIF
  ```

4.1.4 Link failure

Link failure messages are also special Route reply messages, but in this case the destination reflects the route that has broken. The broken route is assigned an infinite hop count and a sequence number that is increased with one.

4.1.5 Forwarding Packets

AODV uses an active neighbor list to keep track of which neighbors that are using a particular route. These lists are used when sending triggered route replies. The neighbor lists are updated every time a packet is forwarded.

```
IF (route entry to destination exists)
IF (neighbor who forwarded packet to you != active neighbor for route)
add neighbor to active neighbor list for route entry
ENDIF
ENDIF
```

4.1.6 Sending and Receiving Triggered RREP

- Algorithm of Sending Triggered RREP

  Link breakages are detected by either the link layer which notifies the routing agent or by using hello messages.

  If a node has not received hello messages from a node for a certain amount of time it will assume that the link is down. Every time a link detected as down,AODV will send a Triggered RREP to inform the affected sources.

  ```
  FOR (each address in the active neighbor list for a route entry)
  create a link failure notice packet
  unicast to active neighbor
  ENDFOR
  ```

- Algorithm of Receiving Triggered RREP

Every time a triggered RREP is received informing about a broken link, the affected route entry must be deleted and neighbors using this entry must be informed.

```
IF (have active neighbors for broken route)
send Triggered RREP
ENDIF
delete route entry for broken route.
```

4.2 Steps to design AODV

The TCL scripts that starts the AODV routing agent and creates all mobile nodes that are using AODV as routing protocol.

- AODV_Agent: Implements all AODV specific parts. Handles RREQ, RREP, Hello and Triggered RREP.It also has a send buffer that buffer packet while a route is searched for. The timer that handles timeouts on route entries and the send buffer are also implemented here.
- Hdr_AODV: Defines the message format for all messages that AODV uses.
- Request Buffer: Implements the request buffer that prevents a node to process the same RREQ multiple times.
- AODV_RTable: The routing table that AODV uses. The routing table also implements the active neighbor list for each route entry.
- AODV Constants: All AODV constants are defined here.

4.3 AODV flow

1. In the TCL script, the user configures AODV as a routing protocol by using the command,

$ns node-config -adhocRouting AODV
the pointer moves to the "start" and this "start" moves the pointer to the Command function of AODV protocol.

2. In the Command function, the user can find two timers in the "start"
* btimer.handle((Event*) 0);
* htimer.handle((Event*) 0);

3. Let's consider the case of htimer, the flow points to HelloTimer::handle(Event*) function and the user can see the following lines:
agent -> sendHello();
double interval = MinHelloInterval + ((MaxHelloInterval - Min-HelloInterval) * Random::uniform
assert(interval -> = 0);
Scheduler::instance().schedule(this, &intr, interval);
These lines are calling the sendHello() function by setting the appropriate interval of Hello Packets.

4. Now, the pointer is in AODV::sendHello() function and the user can see
Scheduler::instance().schedule(target , p, 0.0)
which will schedule the packets.

5. In the destination node AODV::recv(Packet*p, Handler*) is called, but actually this is done after the node is receiving a packet.

6. AODV::recv(Packet*p, Handler*) function then calls the recvAODV(p) function.

7. Hence, the flow goes to the AODV::recvAODV(Packet *p) function, which will check different packets types and call the respective function.

8. For example, flow can go to

case AODVTYPE HELLO:

recvHello(p);

break;

9. Finally, in the recvHello() function, the packet is received.

4.4 Description of main implementation files aodv.cc and aodv.h

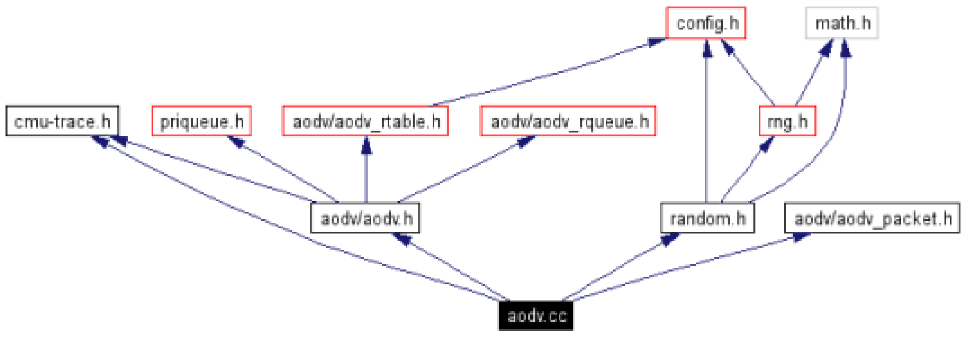


Figure 4.1: File Reference of AODV.CC[1]

4.4.1 Enabling Hello packets

By default, HELLO packets are disabled in the aodv protocol. To enable broadcasting of Hello packets, comment the following two lines present in aodv.cc.

#ifndef AODV LINK LAYER DETECTION

#endif LINK LAYER DETECTION and recompile ns2 by using the following commands on the terminal:

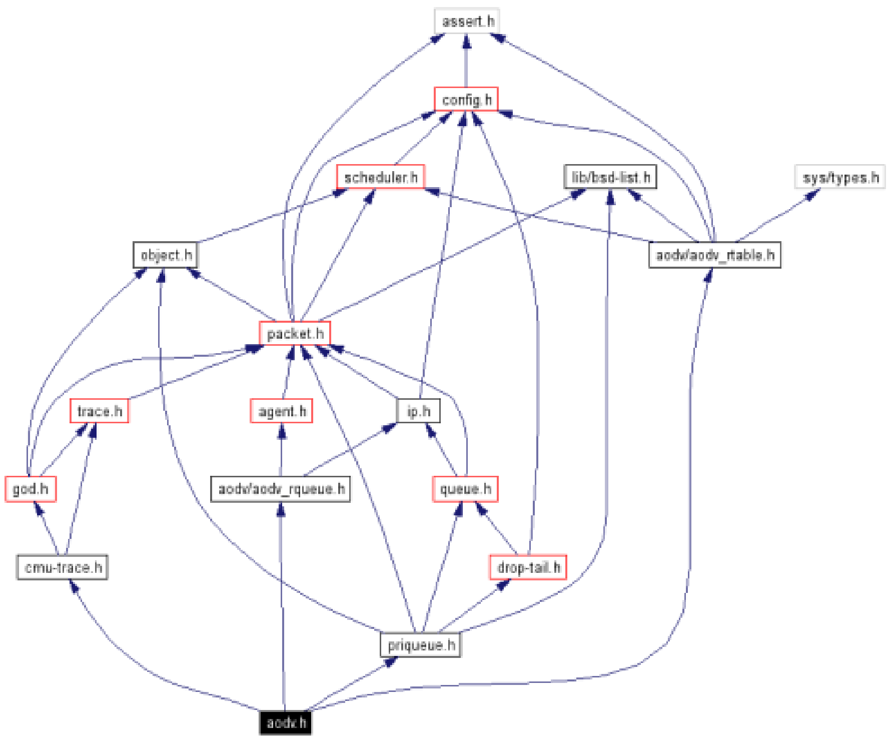


Figure 4.2: File Reference of AODV.H[1]

```
make clean
make
sudo make install
```

4.4.2 Timers

In ns2, timers are used to delay actions or can also be used for the repetition of a particular action like broadcasting of Hello packets after fixed time interval.
Following are the timers that are used in AODV protocol implementation:

- **Broadcast Timer:** This timer is responsible for purging the ID's of Nodes and schedule after every BCAST ID SAVE.

- **Hello Timer:** It is responsible for sending of Hello Packets with a delay value equal to interval, where double interval = MinHelloInterval + ((MaxHelloInterval - MinHelloInterval)* Random::uniform());

- **Neighbor Timer:** Purges all timed-out neighbor entries and schedule after every HELLO INTERVAL .

- **RouteCache Timer:** This timer is responsible for purging the route from the routing table and schedule after every frquency.

- **Local Repair Timer:** This timer is responsible for repairing the routes.

4.4.3 General Functions

- **void recv(Packet *p, Handler *):** At the network layer, the Packet is first received at the recv() function, sended by the MAC layer in up direction.The recv() function will check the packet type. If the packet type is AODV type, it will decrease the TTL and call the recvAODV() function. If the node itself generating the packet then add the IP header to handlebroadcasting, otherwise check the routing loop, if routing loop is present then drop the packet, otherwise forward the packet.

- **int command(int, const char *const *):** Every object created in NS-2 establishes an instance procedure, cmd{} as a hook to executing methods through the compiled shadow object. This procedure cmd invokes the method command() of the shadow object automatically, passes the arguments to cmd{} as an argument vector to the command() method.

4.4.4 Functions for Routing Table Management

- **void rt resolve(Packet *p):** This function first set the transmit failure callback and then forward the packet if the route is up else check if I am the source of the packet and then do a Route Request, else if the local repair is in progress then buffer the packet.If this function founds that it has to forward a packet for someone else to which it does not have a route then drop the packet and send error upstream. Now after this, the route errors are broadcasted to the upstream neighbors.

- **void rt update(aodv rt entry *rt, u int32 t seqnum,u int16 t metric, nsaddr t nexthop,double expire time):** This function is responsible for updating the route.

- **void rt down(aodv rt entry *rt):** This function first confirms that the route should not be down more than once and after that down the route.

- **void local rt repair(aodv rt entry *rt, Packet *p):** This function first buffer the packet and mark the route as under repair and send a RREQ packet by calling the sendRequest() function.

- **void rt ll failed(Packet *p):** Basically this function is invoked whenever the link layer reports a route failure. This function drops the packet if link layer is not detected. Otherwise, if link layer is detected, drop the non-data packets and broadcast packets. If this function founds that the broken link is closer to the destination than source then It will try to attempt a local repair, else brings down the route.

- **void handle link failure(nsaddr t id):** This function is responsible for handling the link failure. It first checks the DestCount, if It is equal to 0 then remove the lost neighbor. Otherwise, if DestCount $>$ 0 then send the error by calling sendError() function, else frees the packet up.

- **void rt purge(void):** This function is responsible for purging the routing table entries from the routing table. For each route, this function will check whether the route has expired or not. If It founds that the valid route is expired, It will purge all the packets from send buffer and invalidate the route, by dropping the packets and tracing DROP RTR NO ROUTE "NRTE" in the trace file. If It founds that the valid route is not expired and there are packets in the sendbuffer waiting, It will forward them. Finally, if It founds that the route is down and if there is a packet for this destination waiting in the sendbuffer, It will call sendRequest() function.

- **void enque(aodv rt entry *rt, Packet *p):** Use to enqueue the packet.
- **Packet* deque(aodv rt entry *rt):** Use to dequeue the packet.

4.4.5 Functions for Neighbors Management

- **void nb insert(nsaddr t id):** This function is used to insert the neighbor.
- **AODV Neighbor* nb lookup(nsaddr t id):** This function is used to lookup the neighbor.
- **void nb delete(nsaddr t id):** This function is used to delete the neighbor and It is called when a neighbor is no longer reachable.
- **void nb purge(void):** This function purges all timed-out neighbor entries and It runs every HELLO INTERVAL * 1.5 seconds.

4.4.6 Functions for Broadcast ID Management

- **void id insert(nsaddr t id, u int32 t bid):** This function is used to insert the broadcast ID of the node.
- **bool id lookup(nsaddr t id, u int32 t bid):** This function is used to lookup the broadcast ID.
- **void id purge(void):** This function is used to purge the broadcast ID.

4.4.7 Functions for Packet Transmission Management

- **void forward(aodv rt entry *rt, Packet *p, double delay):** This function is used to forward the packets.

- **void sendHello(void):** This function is responsible for sending the Hello messages in a broadcast fashion.

- **void sendRequest(nsaddr t dst):** This function is used to send Request messages.

- **void sendReply(nsaddr t ipdst, u int32 t hop count,nsaddr t rpdst, u int32 t rpseq,u int32 t lifetime, double timestamp):** This function is used to send Reply messages.

- **void sendError(Packet *p, bool jitter = true):** This function is used to send Error messages.

4.4.8 Functions for Packet Reception Management

- **AODV::recvAODV(Packet *p):** This function classify the incoming AODV packets. If the incoming packet is of type RREQ, RREP, RERR, HELLO,It will call recvRequest(p), recvReply(p), recvError(p), and recvHello(p) functions respectively.

- **AODV::recvRequest(Packet *p):** When a node receives a packet of type REQUEST, it calls this function.

- **AODV::recvReply(Packet *p):** When a node receives a packet of type REPLY, it calls this function.

- **AODV::recvError(Packet *p):** This function is called when a node receives an ERROR message.

- **AODV::recvHello(Packet *p):** This function receives the HELLO packets and look into the neighbor list, if the node is not present in the neighbor list, It inserts the neighbor, otherwise if the neighbor is present in the neighbor list, set its expiry time to: CURRENT TIME + (1.5 * ALLOWED HELLO LOSS * HELLO INTERVAL), where ALLOWED HELLO LOSS = 3 packets and HELLO INTERVAL= 1000 ms.

Chapter 5

Implementation of Selfishness Attack

5.1 Introduction of Selfish Nodes

There are two types of uncooperative nodes[5]:
1. Malicious nodes and
2. selfish nodes.
Selfish nodes use the network but do not cooperate, saving battery life for their own communications. They do not intend to directly damage other nodes.
Malicious nodes aim at damaging other nodes by causing network outage by partitioning while saving battery life is not a priority.

5.1.1 Analysis of Selfish Nodes

In the type 1 model, the packet forwarding function performed in the selfish node is disabled for all packets that have a source address or a destination address different from the current selfish node. However, selfish node participates in the Route Discovery and Route Maintenance phases of the on-demand protocol[5].
In the type 2 model, selfish nodes do not participate in the Route Discovery phase

of the reactive protocol. The impact of this model on the network maintenance and operation is more significant than the first one. A selfish node of this type uses the node energy only for its own communications[5].

5.1.2 Behaviours of Selfish Nodes

Selfish node can do the following possible actions in Ad hoc network:

- Turn off its power when it does not have active communications with other nodes.
- Does not re-broadcast Route Request (RREQ) when it receives a RREQ.
- Re-broadcasts RREQ but does not forward Route Reply (RREP) on reverse route, therefore the source does not know a route to the destination and it has to rebroadcast a RREQ.
- Re-broadcasts RREQ, forward RREP on reverse route but does not forward data packets.
- Does not unicast/broadcast Route Error (RERR) packets when data packets are received but there is no route.
- Selectively drop data packets.

5.2 Performance Metrics

Following performance metrics are used for analyzing the effect of selfishness attack on Ad hoc network.

- Packet Delivery Ratio (PDR): It is defined as the ratio of total number of packets that have reached the destination node to the total number of packets created at the source node.The larger this metric, the more efficient MANET will be.

$$PDR = \frac{\sum Packets\, received\, by\, destination}{\sum Packet\, sent\, by\, source} * 100 \tag{5.1}$$

- End-to-end Delay: It is defined as time taken for a packet to be transmitted across network from source to destination. The metric should have lower value for the efficient network.

$$End - to - end\, Delay = \frac{Sum\, of\, delays\, of\, each\, CBR\, packet\, received}{number\, of\, CBR\, packet\, received} \tag{5.2}$$

- Packet Loss Rate (PLR): It is defined as the ratio of data packets lost over number of data packet sent during simulation. The metric should have lower value for the efficient network.

$$PLR = 100 - \left(\frac{\sum Packets\, received\, by\, destination}{\sum Packet\, sent\, by\, source} * 100 \right) \tag{5.3}$$

5.3 Implementing Selfishness behaviour

First you need to modify aodv.cc and aodv.h files. In aodv.h after

/* The Routing Agent */

class AODV: public Agent

......

/* * History management */

```
double PerHopTime(aodv_rt_entry *rt);
......
```

add following line

```
bool malicious;
```

With this variable we are trying to define if the node is malicious or not. In aodv.cc after

```
/*
Constructor
/
AODV::AODV(nsaddr_t id) : Agent(PT_AODV),btimer(this), htimer(this), ntimer(this),
rtimer(this), lrtimer(this),
rqueue()
index = id;
seqno = 2;
bid = 1;
...
```

add following line

```
malicious = false;
```

The above code is needed to initialize, and all nodes are initially not malicious. Then we will write a code to catch which node is set as malicious. In aodv.cc after

```
if(argc == 2)
Tcl& tcl = Tcl::instance();
if(strncasecmp(argv[1], ”id”, 2) == 0)
tcl.resultf(”% d”, index);
return TCL_OK;
```

add following line

```
if(strcmp(argv[1], ”hacker”) == 0)
malicious = true;
```

return TCL_OK;

Now we will do some work in TCL to set a malicious node. Using script in my post , we add following line to set node 5 as malicious node.
$ns at 0.0 "[$mnode_(5) set ragent_] hacker"
You may add this line after
for set i 0 $i &lt; $val(nn) incr i
$ns initial_node_pos $mnode_($i) 10

...
Alright, we have set malicious node but we did not tell malicious node what to do. As it is known, rt_resolve(Packet *p) function is used to select next hop node when routing data packets. So, we tell malicious node just drop any packet when it receives. To do that after
/*
Route Handling Functions
/
void
AODV::rt_resolve(Packet *p)
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
aodv_rt_entry *rt;
...
We add a few lines
// if I am malicious node
if (malicious == true)
drop(p, DROP_RTR_ROUTE_LOOP);
// DROP_RTR_ROUTE_LOOP is added for no reason. }

Chapter 6

Detection of Selfishness Attack

6.1 Selfishness Detection Using Watchdog

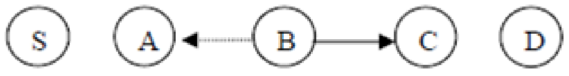


Figure 6.1: watchdog monitoring[3]

As shown in Figure suppose a path exists from node S to node D through intermediate nodes A, B, and C. Here we cant send packet directly from node A to C. To send packet from S to D, A sends packet to B and then node B sends it to C. This way packet will reach to destination. Here each node maintains a buffer for recently sent packets to check the match. If the match is found, the packet is removed from buffer and forgotten by the watchdog, since it has been forwarded on. If a packet has remained in the buffer for longer time, the watchdog considers it as failure and increments a failure counter for the node responsible. If the tally exceeds a certain threshold bandwidth, it means that the node is selfish and about it the notification is sent to source node. The problem with watchdog are power limitation, collision at receiver, ambiguous collision and partial dropping.

6.2 Promiscuous mode in AODV protocol

1. Modify aodv /aodv.h

```
    #include < mac.h >
    class AODV: public Tap, public Agent
{
public:
void tap(const Packet *p);
potected:
Mac *mac_; }
```

2. Modify aodv /aodv.cc

```
    int AODV::command(int argc, const char*const* argv)
    {
    printf(”..........Other code of AODV command class.......”);
    else if(argc == 3)
    {
    ......
    else if (strcmp(argv[1], ”install-tap”) == 0)
    { mac_ = (Mac*)TclObject::lookup(argv[2]);
if (mac_ == 0) return TCL_ERROR;
mac_ → installTap(this);
return TCL_OK;
}
    }
return Agent::command(argc, argv);
}

    void AODV::tap(const Packet *p)
    {
```

```
    // Your code goes here......
    printf("Our AODV is in promiscuous mode, ");
    }
    3. Modify tcl/lib/ns-mobilenode.tcl
    Node/MobileNode instproc add-target  agent port
    $self instvar dmux_ imep_ toraDebug_ mac_

    # Special processing for AODV
set aodvonly [string first "AODV" [$agent info class]]
if {$aodvonly!= -1} {
$agent if-queue [$self set ifq_(0)] ; # ifq between LL and MAC

    $agent install-tap $mac_(0)
     }
```

Detection of this attack is triggered whenever a node forwards routing traffic to its neighbouring nodes. A structure called SELFISH_Node was developed to hold information necessary to monitor the neighbouring nodes that are suspected for malicious behaviour.

The SELFISH_Node data structure holds the following information:

- node_id: the IP address of the node to which the routing traffic was forwarded.
- send_reply: a boolean value that becomes true whenever the offending node replies to a RREQ packet that was forwarded to it.
- pre_alarm: a boolean value that becomes true if the node does not respond as expected to the forwarded traffic.
- alarm: a boolean value that becomes true whenever we decide that the offending node performs the dropping routing packets attack.

- time: a double variable that keeps the time where the offending node was added in the data structure.

Hence, whenever a node forwards routing traffic for which a neighbouring node is not the destination it adds each neighbouring node to the data structure and waits to observe their behaviour. Then in the tap method if it overhears that a neighbouring node has replied to the forwarded RREQ, it means that it has acted appropriately and it can be removed from the monitoring list. If this is not the case and the packet was a RREP then the offending node has to forward the packet. If it fails to do so within the pre_alarm_time_threshold time period, which was determined by experiments to be 0.01 seconds, the pre_alarm state becomes true. This remains in the pre_alarm state for 0.45 seconds which is the alarm_threshold time period. If the offending node fails to forward the routing packet within this time limit, it moves to the Alarm state. In case of an alarm the legitimate node marks this node as malicious and stops forwarding traffic to it for 2 seconds and it also sends a RERR message to all its upstream neighbours to inform them that all the routes that include this node are not valid.

Chapter 7

Implementation of Credit Based System

7.1 Algorithm for Credit Based System

1. Initialize all the node with some amount of Virtual money (Credit) and set the packet drop index as zero.
2. During Packet Forwarding Mechanism if node is forwarding the packet successfully than increase the Virtual money by two unit else Drop() function will be called and reduce the virtual money by two unit and increase the drop index.
3. Now set the Minimum Thresold value of credit and than check following condition.

```
if (Creditis <min_Thresold)
{
Discard the node from Route for this give them runtime mobility using Setdest() function
} OR
if(Drop[index] > Thresold)
{ Discard the node from Route for this give them runtime mobility using Setdest() function }
```

7.2 Implementation step for Credit using NS2.34

step-1. In NS2.34/common/agent.h file add the following line

```
Class Agent:public Connector{
public:
inline void set_credit(int credit)
{ credit_=credit;
}
protected:
int credit_;
```

step-2. In NS2.34/common/agent.cc file add the following line

```
void Agent::initpkt(packet *p) const
{
ch→set_Credit()=Credit_;
```

step-3. In NS2.34/common/packet.h file add the following line

```
Struct hdr_cmn{
int Credit_;

inline int& set_Credit()
{
return(Credit_);
}
```

step-4. In NS2.34/common/packet.cc file add the following line

```
void export_offset()
{
field_offset(”Credit_”,OFFSET(hdr_cmn,credit_));
}
```

step-5. In NS2.34/common/cbr_traffic.cc file add the following line

```
protected:
```

```
int credit_;
CBR_Traffic::CBR_Traffic():seqno_(0)
{ bind(”Credit_”,&Credit_);
} void CBR_Traffic::init()
{
agent_→set_credit(credit_);
}
```

step-6 NS2.34/tcl/lib/ns-default.tcl add the following line

```
Application/Traffic/CBR set Credit_ 100;
```

step-7 In NS-2.34/aodv/adov.cc/forward() add the following line

```
int t=1;
int credit[5000];
int counter=0;
static int d[5000]; void forward()
{
if (t==1 && counter==0) {
credit[index]=ch→set_credit();
t=0;
counter=1;
}
credit[index]=credit[index]+2;
printf(”credit after increment=%d of node=%d,credit[index],index);
if(ih→ttl_=0) {
d[index]=d[index]+1;
drop(p,DROP_RTR_TTL);
if(d[index]> 25)
{
printf(”index=%d” is malicious,index); }
```

Chapter 8

Results and Analysis

For simulation, we have used ns-2.34 simulator. Here Table-I shows parameter for simulating original AODV with various number of nodes, Figure 8.1 shows Comparision of PDR, Throughput and Delay with respect to various number of nodes.Table-III and Figure-8.2 shows the effect of Active connection on PDR,Throughput and Delay. Table-IV and Figure-8.3 shows the mobility speed on PDR,Throughput and Delay.Table-V shows the parameter for simulating AODV with various number of Selfish nodes. Figure-8.4 shows running AODV.tcl without any selfish nodes and Figure 8.5 shows running AODV.tc with Selfish nodes. Figure 8.6, 8.7, 8.8 shows the effect of Selfish nodes on PDR, Throughput and Delay respectively.Figure 8.11 shows watchdog detection output.

Table I: Simulation parameters

Property	Value
Nodes	10,20,30,40,50,60,70
Simulation time	500Sec
Mobility model	Random way point
Coverage area	750m*750m
Maximum speed	20 m/s
Pause time	1.0 Sec
Traffic type	Constant Bit Rate(CBR/UDP)
Send Rate	10 packets/sec
Packet size	512 bytes

Table II: Effect of number of Nodes on PDR, Throughput, and Delay.

Nodes	PDR(%)	Throughput (kbps)	End-to-end Delay(ms)
10	80.24	258.56	252.13
20	76.69	180.46	308.15
30	71.36	162.33	405.26
40	65.68	196	588.13
50	58.44	160.23	700.65
60	53.4	143.55	865.16
70	45.46	110.5	987.65

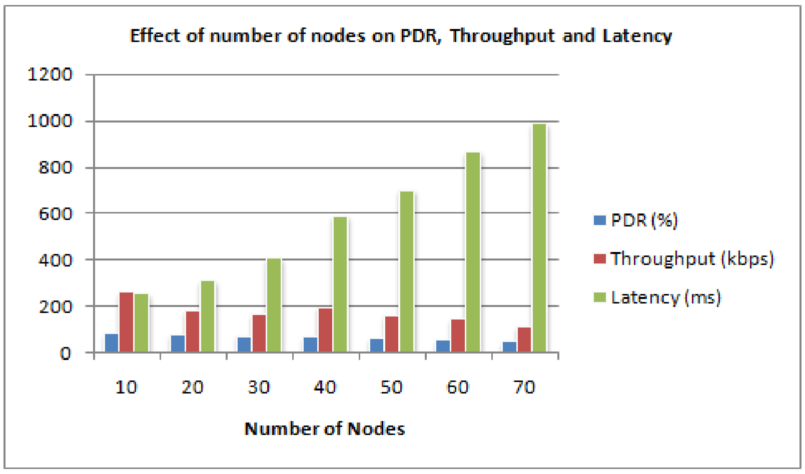


Figure 8.1: Effect of number of nodes on PDR, Throughput, Delay.

Table III: Effect of number of Active connection on PDR, Throughput, and Delay.

Active Connections	PDR(%)	Throughput (kbps)	End-to-end Delay(ms)
5	96.99	74.82	134.947
10	88.68	139.08	170.311
15	73.40	170.29	421.946
20	66.60	201.4	493.753
25	63.80	210.97	495.493

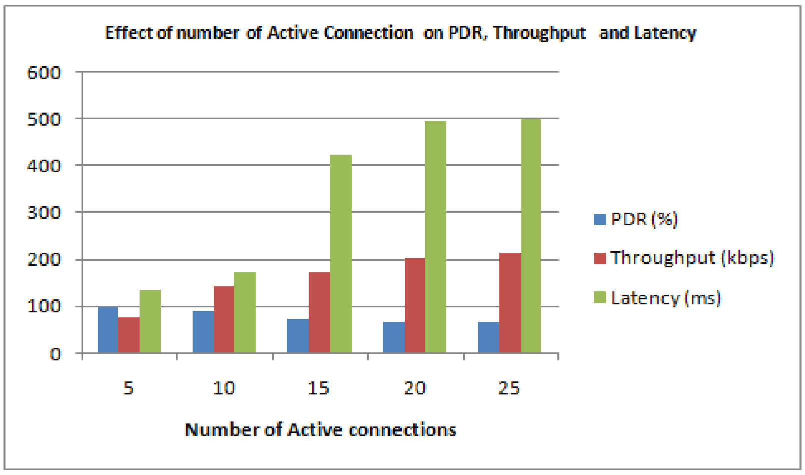


Figure 8.2: Effect of number of Active Connections on PDR, Throughput, Delay.

Table IV: Effect of Mobility Speeds on PDR, Throughput, and Delay.

Mobility speed(m/s)	PDR(%)	Throughput (kbps)	End-to-end Delay(ms)
8	82.86	192.26	178.534
12	81.92	190.09	335.166
16	72.35	167.91	471.146
20	70.00	162.53	475.016
25	69.87	161.53	480.120

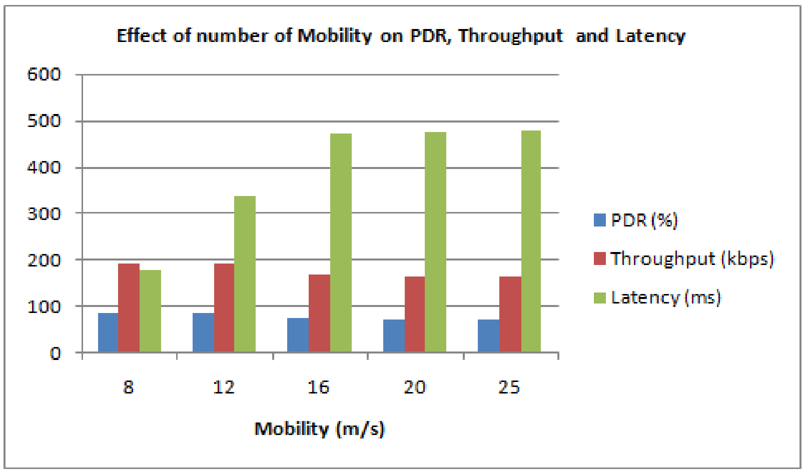


Figure 8.3: Effect of number of Mobility Speeds on PDR, Throughput, Delay.

Table V: Simulation parameters

Property	Value
Nodes	25
Simulation time	600Sec
Mobility model	Random way point
Coverage area	1000m*1000m
Maximum speed	10 m/s
Pause time	2.0 Sec
Traffic type	Constant Bit Rate(CBR/UDP)
Send Rate	10 packets/sec
Packet size	512 bytes

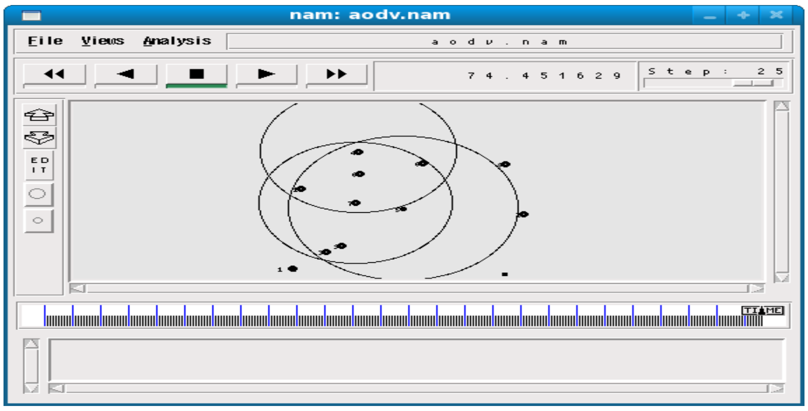


Figure 8.4: Running AODV.tcl without malicious node.

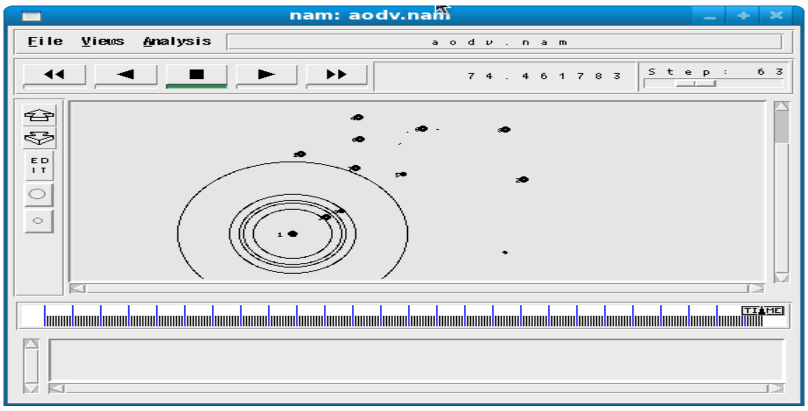


Figure 8.5: Running AODV.tcl with malicious node.

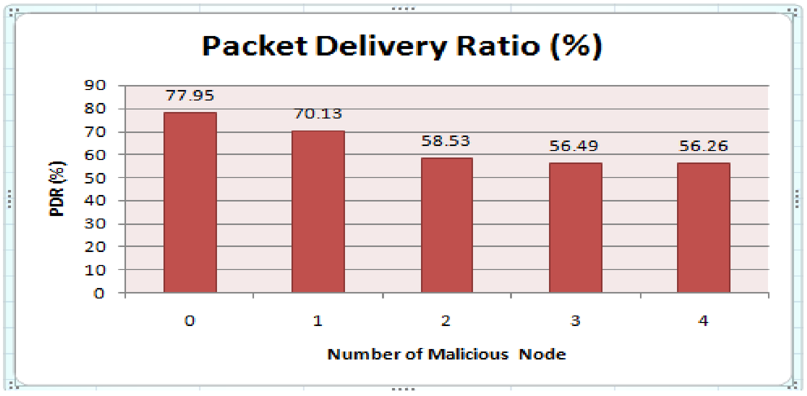


Figure 8.6: Effect of Malicious node on PDR

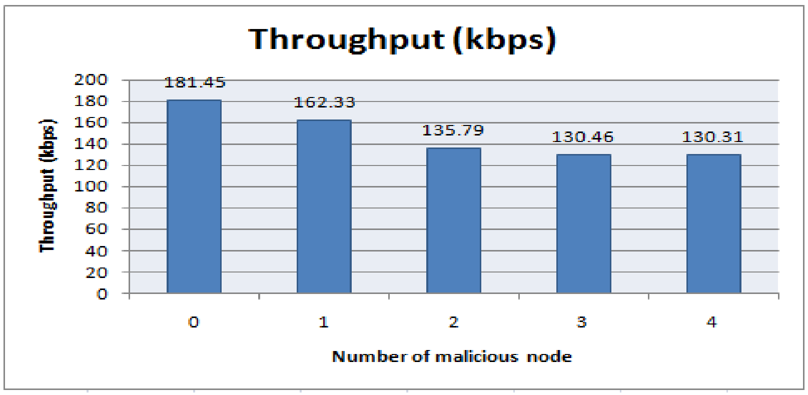


Figure 8.7: Effect of Malicious node on Throughput

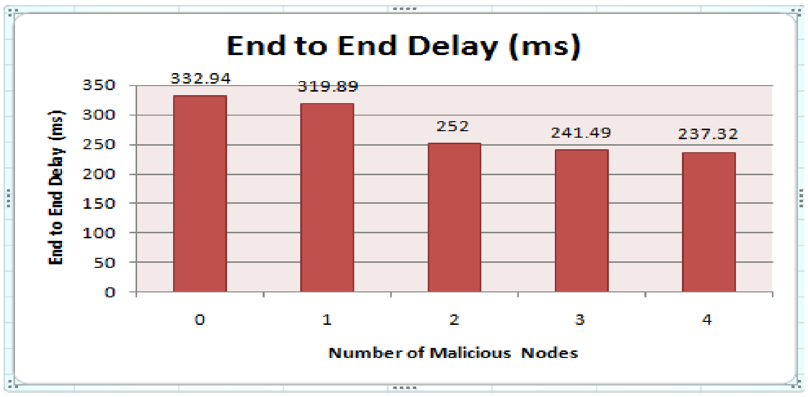


Figure 8.8: Effect of Malicious node on Latency

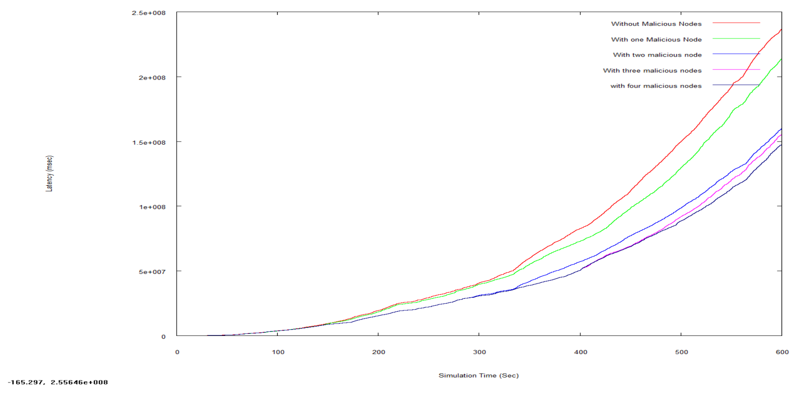


Figure 8.9: Effect of Malicious node on Latency

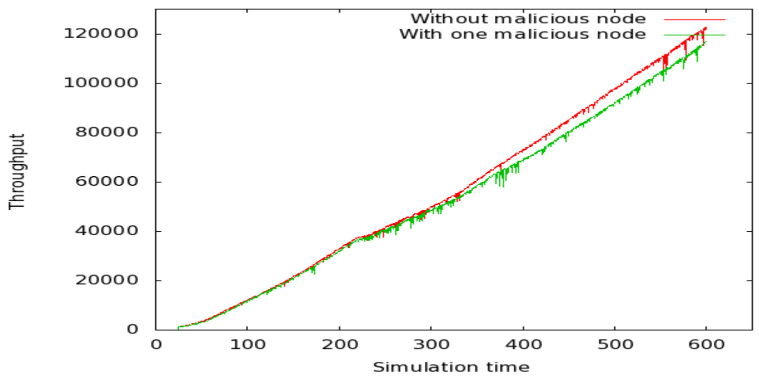
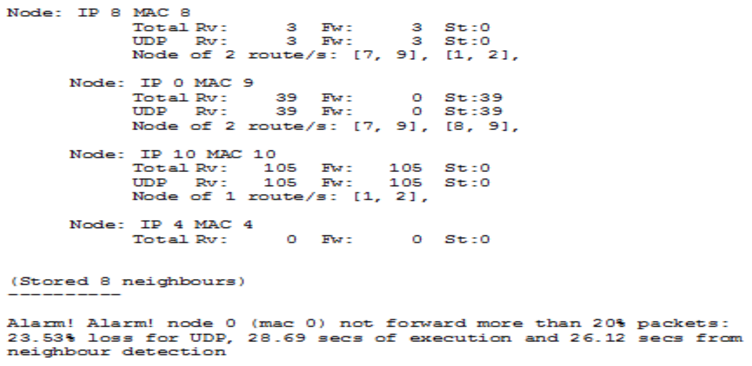


Figure 8.10: Effect of Malicious node on Throughput



```
Node:  IP 8 MAC 8
            Total Rv:      3   Fw:      3   St:0
            UDP   Rv:      3   Fw:      3   St:0
            Node of 2 route/s: [7, 9], [1, 2],

     Node:  IP 0 MAC 9
            Total Rv:     39   Fw:      0   St:39
            UDP   Rv:     39   Fw:      0   St:39
            Node of 2 route/s: [7, 9], [8, 9],

     Node:  IP 10 MAC 10
            Total Rv:    105   Fw:    105   St:0
            UDP   Rv:    105   Fw:    105   St:0
            Node of 1 route/s: [1, 2],

     Node:  IP 4 MAC 4
            Total Rv:      0   Fw:      0   St:0

(Stored 8 neighbours)
----------

Alarm! Alarm! node 0 (mac 0) not forward more than 20% packets:
23.53% loss for UDP, 28.69 secs of execution and 26.12 secs from
neighbour detection
```

Figure 8.11: Watchdog Detection output

Chapter 9

Conclusion and Future work

9.1 Conclusion

Ad-hoc networks are vulnerable due to absence of infrastructure, limited physical security, restricted power supply, dynamically changing network topology, lack of centralized monitoring and mobility. In this thesis, effect of the selfishness attack in an AODV network is analyzed. For this purpose, AODV protocol is implemented and then selfish node is appended in that to analyze its effectiveness. AODV protocol with various number of selfish node is simulated, where PDR, Throughput and Delay is reduced. For detection of the selfishness attack we have used Watchdog module. After implementing the IDS, to make more reliable routing credit based approach is implemented. I also tested the effect of number of nodes, mobility speeds and number of active connection on PDR, Throughput and Delay. PDR and Throughput are increased and Delay is reduced with increasing the number of nodes. PDR and Delay are increased and Throughput is reduced with increasing the number of active connections. PDR and Throughput are reduced and Delay is increased with increasing the number of nodes.

9.2 Future work

- Selfish node detection can be done using machine learning concepts.
- In this, we have proposed solution for the selfishness problem. Here, we have implemented selfishness attack, watchdog module and credit based system only for 25 number of nodes and four number of nodes, we may take more number of nodes and test its effect on perfomance metrics.
- Apply the selfishness attack, watchdog module and credit based system into other applications and other routing protocols of the MANET (e.g., DSR and DSDV).
- We also plan to investigate more attack scenarios in MANETs, not only at the routing layer, but also at other layers.
- We also plan to developed centralized IDS, genuine and ungenuine identification of alert for attacker, reduce and combine the alert log using clustering techniques
- We have used UDP connection as traffic type. Instead of that TCP connection can be used as traffic type.

Appendix A

List of publications

a. Vaseem Ghada, Sharada Valiveti, "Secure Routing using Detection Method in Wireless Ad Hoc network", *International Journal for Scientific Research and Development (IJSRD),Volume-I, Issue-II, May-2013 (ISSN no: 2321-0613).*

References

[1] S.Dokurer,"Simulation od blackhole attack in wireless ad-hoc networks",2006

[2] J.Wang,"ns-2 tutorial",2004

[3] Dipali Koshti, Supriya Kamoji, "Comparative Study of Techniques used for Detection of Selfish Nodes in MANET", IJCSE Vol-1 Issues-4,Sept-2011

[4] D.S.Nishant Chaurasia, Sanjay Sharma,"Review study of routing protocols and versatile challanges of manet",vol.1

[5] Niyati Shah, Sharada Valiveti, Intrusion Detection System for the Availability Attacks in Ad-Hoc Networks, IJECSE,2011.

[6] Sangheethaa Sukumaran, Venkatesh. J, Arunkorath, "A survey of method to mitigate selfishness in MANET", IJICT Vol-1 No.2, June 20111

[7] Charlie Obimbo, Liliana Maria Arboleda-cobo, "An Intrusion Detection System for MANET" , CISME, VOl-2 No. 3,2012

[8] Mohit p. tahiliani-blogspot", tech-report

[9] A.P. Yih-chun Hu, David B. Johnson, Sead: Secure efficient distance vector routing for mobile wireless Ad Hoc, networks2003.

[10] L.Abusalah,A.Khokhar,G.BenBrahim,W.ElHajj, "TARP:Trust Aware Routing Protocol", ACM July-2006.

[11] A novel approach for selfish node detection in MANETs: proposel and petri nets based modeling,8th IEEE internation conference on telecommunication,2005

[12] Charlie Obimbo, Liliana Maria Arboleda-cobo, "An Intrusion Detection System for MANET" , CISME, VOl-2 No. 3,2012

[13] Michiardi,Molva,CORE:collaborative reputation mechanism to enforce node cooperation in MANET,6th IFIP communication and multimedia security cinference,2002.

[14] Aleksandar Lazarevic, Vipin Kumar, Jaideep Srivastava, "INTRUSION DETECTION: A SURVEY", Computer Science Department, University of Minnesota

[15] Niyati Shah,"Trust based Routing using IDS in Ad-hoc networks, Thesis report, May-2012

[16] Preeti Nagrath,Ashish Kumar,Shikha Bhardwaj, "Authenticated Routing Protocol based on Reputation System For Adhoc Networks",*International Journal on Computer Science and Engineering(IJCSE),Vol.2: 3095-3099,2010*

[17] S.Tamilarasan,Dr.Aramudan, "A performance and Analysis of Misbehaving node in MANET using IDS",*International Journal on Computer Science and Network Security(IJCSNS),Vol.11, May 2011*

[18] SADIA HAMID KAZI, "Congestion control inMobile Ad-Hoc Networks(MANETs)", BRAC UNIVERSITY, April 2011

[19] Ioanna Stamouli, "Real-time Intrusion Detection for Ad hoc Networks", University of Dublin, September 2003