Routing Misbehavi... and Performance Analysis of AODV in MANET

Dissertation submitted to ***Jawaharlal Nehru University*** *in partial fulfillment of the requirements for the award of the degree of*

Master of Technology

In

Computer Science

by

PRAMIL KUMAR KUCHHAL

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

JANUARY 2002

जवाहरलाल नेहरू विश्वविद्यालय
JAWAHARLAL NEHRU UNIVERSITY
School of Computer & Systems Sciences
NEW DELHI - 110 067, INDIA

CERTIFICATE

This is to certify that the dissertation entitled " ***Routing Misbehavior and Performance Analysis of AODV in MANET*** " which is being submitted by **Mr. Pramil Kumar Kuchhal** to the school of Computer and system sciences, Jawaharlal Nehru University, for the award of **Master of Technology in Computer Science and Technology**, is a record of bonafide work carried out by him.

This work is original and has not been submitted in part or full to any university or institution for the award of any degree.

Prof. K.K. Bhardwaj	Dr. D.K. Lobiyal	Prof. G.V. Singh
Dean, SC&SS,	Asst. Professor, SC&SS	Professor, SC&SS,
JNU, New Delhi.	JNU, New Delhi.	JNU, New Delhi.

Gram : JAYENU o Tel : 6107676, 6167557 (Extn. 2767, 2768, 2769) o Fax : 616-5886

DECLARATION

This is to certify that the dissertation entitled "***Routing Misbehavior and Performance Analysis of AODV in MANET*** " which is being submitted to the school of Computer and system sciences, Jawaharlal Nehru University, for the award of **Master of Technology in Computer Science and Technology**, is a record of bonafide work carried out by me.

This work is original and has not been submitted in part or full to any university or institution for the award of any degree.

Pramil Kumar Kuchhal

ACKNOWLEDGEMENTS

I would like to sincerely thank my supervisors Prof. G.V. Singh, Dr D.K. Lobiyal, School of Computer And System Sciences, Jawaharlal Nehru University for the help, encouragement and support extended by them in successful completion of this project. Their ideas were innovative and the valuable discussions we had were very much helpful in keeping the project on the right track.

I would like to record my sincere thanks to the dean, Prof. K.K. Bharadwaj for providing the necessary lab facilities.

I extend my sincere gratitude to my lab mates for their continuous academic as well as morale support through out my dissertation work.

I will acknowledge the efforts and knowledge put by the development team of VINT project team at UC Berkeley, for developing the network simulator and making its source code available on the Internet as a freeware.

Last, but not the least, I take this opportunity to thank all the faculty members and friends for their, help and encouragement during the course of the project.

Pramil kumar Kuchhal

...dedicated to my parents

Abstract

In the present work, we have analyzed the effect of increase in percentage of routing misbehavior on the performance of the network for Ad hoc On demand Distance Vector Routing Algorithm (AODV). We have varied the network size, traffic pattern and mobility rate to see it effect on End-to-End delay, delivery rate, overhead ratio and bandwidth utilization.

Our simulation results show that as the percentage of routing misbehavior increases there is a significant decoration of the network performance. Main factors responsible for detoriate of network performance are queuing delay and uncontrolled spread of control REQUEST and REPLY packets of AODV routing algorithm.

In the implementation, Tcl script has been used to configure the network and to simulate it. For simulation analysis trace file was analyzed using UNIX script and C code.

Contents

Introduction 1

Mobile hosts such as notebook computers, featuring powerful CPUs, large main memories, hundreds of megabytes of disk space, multimedia sound capabilities, and color displays, are now easily affordable and are becoming quite common in everyday business and personal life. At the same time, network connectivity options for use with mobile hosts have increased dramatically, including support for a growing number of wireless networking products based on radio and infrared. With this type of mobile computing equipment, there is a natural desire and ability to share information between mobile users. Often, mobile users will meet under circumstances that are not explicitly planned for and in which no connection to a standard wide area network such as the Internet is available. For example, employees may find themselves together in a meeting room; friends or business associates may run into each other in an airport terminal; a hotel ballroom for a workshop or conference. Requiring each user to connect to a wide area network in such situations, only to communicate with each other, may not be possible due to lack of facilities, or may be inconvenient or impractical due to the time or expense required for such connection.

The solution to these types of networking problems has come up in the form of Mobile ad hoc networks. An ad hoc network is a collection of wireless mobile hosts forming a temporary network without the aid of any centralized administration or standard support services regularly available on the wide area network to which the hosts may normally be connected.

Some form of routing protocol is in general necessary in such an environment, since two hosts that may wish to exchange packets might not be able to communicate directly. For example, Figure 1.1 illustrates a simple ad hoc network of three mobile hosts using wireless network interfaces. Host C is not within the range of host A's wire less transmitter (indicated by the circle around A) and host A is not within the range of host C's wireless transmitter. If A and C wish to exchange packets, they may in this case enlist the services of host B to forward packets for them, since B is within

the overlap between A's range and C's range. The maximum number of network hops needed to reach another mobile host in any practical ad hoc network is likely to be small, but may often be greater than one as shown here. The routing problem in a real ad hoc network may be even more complicated than this example suggests, due to the inherent nonuniform propagation characteristics of wireless transmissions, and since any or all of the hosts involved may move at any time.

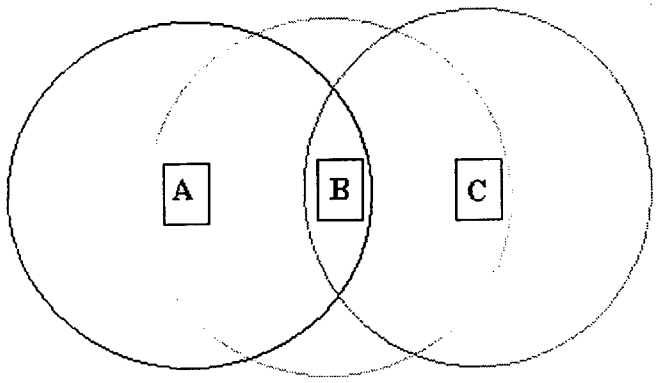

Figure 1.1 An ad hoc network of three wireless mobile hosts

1.1 Mobile Ad-hoc networks (M.A.NETs)

A mobile, ad hoc network (MANET) is a collection of wireless mobile hosts forming a network without the aid of any established infrastructure or centralized administration[12]. A MANET may be considered an autonomous system of mobile nodes. Such networks have dynamic, sometimes rapidly changing, random, multihop topologies, which are likely composed of relatively bandwidth-constrained wireless links.

MANETs have several salient characteristics that have to be taken into account when considering their design and deployment:

- **Dynamic topologies:** Nodes are free to move arbitrarily; thus, the network topology (which is typically multihop) may change randomly and rapidly at unpredictable times, and may consist of both bi-directional and unidirectional links.
- **Bandwidth-constrained, variable capacity links:** Wireless links typically have significantly lower capacity than their hardwired counterparts. In addition, the

realized throughput of wireless communications (after accounting for the effects of multiple access, fading, noise, and interference conditions) is often much less than a radio's maximum transmission rate.

- **Energy-constrained operation**: Some or all of the nodes in a MANET may rely on batteries or other exhaustible means for their energy. For these nodes, a finite energy capacity may be the most significant performance constraint, and thus its utilization should be viewed as a primary network control parameter.

While the above characteristics are common to any kind of wireless network, MANETs are further distinguished by their "*Infrastructure less*" property : There is no centralized administration or preexisting infrastructure that takes care of the network management and existence. Mobile nodes are themselves responsible for establishing and maintaining connection between them. In such an environment, it is necessary for one mobile host, to enlist the aid of other hosts in forwarding a packet to its destination, due to the limited range of each mobile host's wireless transmissions. Thus, robust and efficient operation in mobile wireless networks is supported by incorporating routing functionality into all the mobile nodes, in contrast to fixed networks such as the Internet where only some nodes in the network perform the routing function.

It is obvious that the routing function is of utmost importance for the viability of an ad-hoc network. It is also a big challenge since all the characteristics of the mobility and wireless channel previously mentioned, must be taken into account when designing such protocols.

1.2 Classification of Routing Protocols

There has been extensive research work in the field of routing algorithm's of Mobile Ad hoc Networks (MANET). Many routing algorithms have been proposed and are in the different stages of development like AODV[2], DSR[5], DSDV[3], TORA[14], etc. The routing protocols can be separated in two main classes, namely proactive or reactive, according to the way routes are created and maintained.

1.2.1 Proactive Routing Protocols

These algorithms maintain a route to the destination much like the traditional fixed networks. When the packet is to be transmitted it just picks up a root from the cache and uses it. This leads to the instant transmission of the first packet, but the nodes have to work hard in the background to establish roots, wasting precious radio resources.

One prominent protocol in the proactive class is the Destination Sequenced Distance Vector Protocol (DSDV) [3] .

1.2.1.1 Destination Sequenced Distance Vector Protocol (DSDV)

DSDV [3] is a hop-by-hop distance vector routing protocol that in each node has a routing table that for all reachable destinations stores the next-hop and number of hops for that destination. Like distance-vector, DSDV requires that each node periodically broadcast routing updates. The advantage with DSDV over traditional distance vector protocols is that DSDV guarantees loop-freedom.

To guarantee loop-freedom DSDV uses a sequence numbers to tag each route. The sequence number shows the freshness of a route and routes with higher sequence numbers are favorable. A route R is considered more favorable than R' if R has a greater sequence number or, if the routes have the same sequence number but R has lower hop-count. The sequence number is increased when a node A detects that a route to a destination D has broken. So the next time node A advertises its routes, it will advertise the route to D with an infinite hop-count and a sequence number that is larger than before.

DSDV basically is distance vector with small adjustments to make it better suited for ad hoc networks. These adjustments consist of triggered updates that will take care of topology changes in the time between broadcasts. To reduce the amount of information in these packets there are two types of update messages defined: full and incremental dump. The full dump carries all available routing information and the incremental dump that only carries the information that has changed since the last dump.

Because DSDV is dependent on periodic broadcasts it needs some time to converge before a route can be used. This converge time can probably be considered

negligible in a static wired network, where the topology is not changing so frequently. In an ad hoc network on the other hand, where the topology is expected to be very dynamic, this converge time will probably mean a lot of dropped packets before a valid route is detected. The periodic broadcasts also add a large amount of overhead into the network.

1.2.2 Reactive Routing Protocols

Reactive routing protocols or "on demand" routing protocols create and maintain routes only in an "as needed" basis, unlike proactive protocol's where routes are maintained to all potential destinations. When a route is needed, a global route discovery procedure is initiated to find the path for the specific information that has not been cached. The route discovery is usually done by employing classical flooding mechanisms. The three prominent protocols in the reactive class are the Dynamic Source Routing Protocol (DSR) [5], Ad hoc on Demand Distance Vector Routing Algorithm (AODV)[2] and Temporary Ordered routing Algorithm (TORA)[14].

1.2.2.1 Dynamic Source Routing (DSR)

The Dynamic Source Routing protocol (DSR) [5], uses source routing a technique where the source of a data packet determines the complete sequence of nodes through which to forward the packet. The source explicitly lists this route in the packet's header and then transmits the packet over its wireless network interface to the first hop identified in the source route. When a host receives the packet, if this host is not the final destination of the packet, it strips from the packet header its own address and simply transmits the packet to the next hop identified in the packet header. This process goes on until the packet reaches its destination. The two basic operations supported by DSR are **Route Discovery** and **Route Maintenance.**

DSR builds routes on demand by flooding the network in a controlled manner (for example by using a Time To Live (TTL) field in the packet header). In Route Discovery, the source node sends a query in the form of a *route request* packet that carries the sequence of hops it passed through. Once a query reaches the destination, the destination replies with a *route reply* packet that simply copies the route from the query packet and traverses it backwards. To reduce the cost and frequency of the

Route Discovery, each node has a route cache, where complete routes have been stored as learned from the reply packets. These routes are used by the data packets until they fail as determined by the failure of attempted message transmissions.

Route Maintenance procedure monitors the operation of the routes and informs the sender of any routing errors. Error detection is supported in the data link layer. If the data link layers reports a transmission problem for which it cannot recover, this host sends a *route error* packet to the original sender of the packet. The route error packet contains the addresses of the hosts at both ends of the hop in error :

1. the host that detected the error, and
2. the host to which it was attempting to transmit this packet on this hop.

When a route error packet is received by the source host, the hop in error is removed from this host's route cache and all routes which contain that hop are truncated at that point. Then, a new route discovery for the destination is initiated by the sender.

DSR has a unique advantage by virtue of source routing. The key advantage of source routing is that intermediate nodes need not maintain up-to date routing information in order to route the packets they forward, since the packets themselves already contain all the routing decisions. This fact coupled with the on demand nature of the protocol eliminates the need for the periodic route advertisement and neighbor detection packets present in other protocols. Another nice feature of DSR is that since the route is part of the packet itself, routing loops, either short or long lived, cannot be formed, as they are immediately detected and eliminated. A node can learn a route to a destination while passing on route reply packets. Also routes can be improved by having nodes promiscuously listen to conversations between other nodes in proximity.

1.2.2.2 Ad-Hoc On Demand Distance Vector Protocol (AODV)

AODV[2] is an on-demand variation of distance vector protocols. It is essentially a combination of DSR and DSDV. It borrows the basic on demand mechanism of Route Discovery and Route Maintenance from DSR plus the use of hop-by-hop routing, sequence number and periodic beacons from DSDV. AODV uses sequence numbers maintained at destinations to determine freshness of routing information. In AODV, a query flood is used to create a route, with the destination responding to the first such query, much as in DSR. However, AODV maintains routes in a distributed fashion, as routing table entries on all intermediate nodes of the route.

Routing table entries are tuples in the form of < destination, next hop, distance >. Nodes propagating query packets "remember" the earlier hop taken by such a query packet. This hop is used to forward the reply packet back to the source. The reply packet, in turn, sets the routing table entries on the nodes in its path. Distributing routing tables in such a fashion makes them smaller than the case of DSR where each node must keep the whole path in its cache. However, in order to maintain routes, AODV normally requires that each node periodically transmit a HELLO message, with a default rate of one per second. Failure to receive three consecutive HELLO messages from a neighbor is taken as an indication that the link to the destination in question is down.

When a link goes down, any upstream node that has recently forwarded packets to a destination using that link is notified via an UNSOLICITED ROUTE REPLY containing an infinite metric for that destination. Upon receipt of such a ROUTE REPLY, a node must acquire a new route to the destination using Route Discovery as described above.

AODV maintains the addresses of the neighbors through which packets destined for a given destination were received. A neighbor is considered *active* (for a destination) if it originates or relays at least one packet for that destination, within the past *active timeout period*. A routing table entry is active if it is used by an active neighbor. The path from a source to destination via the active routing table entries is called an *active path*. On a link failure, all routing table entries for which the failed link is on the active path, are erased. This is accomplished by an error packet going backwards to the active neighbors, which forward them to their active neighbors and so on. This technique effectively erases the route backwards from the failed link.

Much like DSR, AODV advocates use of *"early quenching"* of request packets, i.e., any node having a route to the destination can reply to a request. AODV also uses a technique called *route expiry*, where a routing tale entry expires after a predetermined period, after which fresh route discovery must be initiated. Neither DSR nor AODV guarantee shortest path.

1.2.2.3 Temporary Ordered routing Algorithm (TORA)

The unique feature of TORA [5] is that it is a protocol designed to minimize reaction to topological changes. This is accomplished by maintaining multiple routes to a

specific destination, so that many topological changes need no reaction at all, unless all routes to a specific destination are lost. In that case routes are re-established via a temporary ordered sequence of diffusing computations, which are essentially link reversals that eventually establish a path to the destination.

TORA does not maintain routes between a given source/destination pair at all times but creates new routes on demand. Route optimality (shortest paths) is considered secondary importance, and longer routes are often used to avoid the overhead of discovering newer routes. This ability to initiate and react infrequently serves to minimize the communication overhead at the expense of multiple non-optimal (shortest path) routes to the destination. In order to select one of the multiple routes two alternatives are suggested :

1. choosing a neighbor randomly so that the loads are more or less evenly distributed, or
2. choosing the "lowest" neighbor.

TORA is based in part on algorithms that try to maintain the destination oriented property of a directed acyclic graph (DAG). A DAG is defined to be *destination oriented* if there is always at least one path to a specific destination. The DAG becomes *destination disoriented* when one or more link fails. In this case by employing link reversals these algorithms ensure that the DAG will become again destination oriented in a finite time.

TORA uses the notion of node "height" to maintain the destination oriented DAG. Each node maintains a height and exchanges this value with each neighbor. The significance of the height is that a link is always directed from a "higher" node to a "lower" node. The notion of "height" and link reversals are destination specific. This means that each node of the network runs a logically separate copy of TORA for each destination.

The **route discovery** process for a specific destination creates a destination oriented graph for this destination in a source initiated fashion: The source node broadcasts a QUERY packet containing the address of the destination for which it requires a route. This packet propagates through the network until it reaches the destination or an intermediate node having a route to the destination. The recipient of he QUERY then broadcasts an UPDATE packet listing its height with respect to the destination. As this packet propagates through the network, each node that receives the UPDATE sets its height to a value greater than the height of the neighbor from

which the update was received. This has the effect of creating a series of directed links from the original sender of the QUERY to the node that initially generated the update.

Route maintenance is achieved as follows : when a node *i* discovers that a route to a destination is no longer valid, it adjusts its height so that it is a local maximum with respect to its neighbors and transmits an *UPDATE* packet. This is effectively a link reversal and it means that now all links emanating from node *i* are directed from *I* towards its neighbors (since the neighbors have now lower heights). This has as an effect all traffic entering *i* to flow back out of *i* towards the neighbor nodes that had previously been routing packets to the destination via i. If the node has no neighbors of finite height with respect to this destination, then the node attempts to discover a new route via the route discovery process.

1.3 Applications of Mobile Ad hoc Networks

A mobile ad hoc network includes several advantages over traditional wireless networks, including: ease of deployment, speed of deployment, and decreased dependence on a fixed infrastructure. MANET is attractive because it provides an instant network formation without the presence of fixed base stations and system administrators. MANET is being viewed as a suitable systems for some specific applications including:

- Personal communication like cell phones, laptops, PDA,
- Group communication such as communication set-up in exhibitions, conferences, presentations, meetings, lectures,
- Military, emergency, discovery and civil communication.

1.4. Problem Description

In MANET all nodes are mobile and can be connected dynamically and in arbitrary manner. All nodes of this network behave as routers and take part in discovering and maintenance of routes to other nodes in the network. But some times a node may agree to forward packet and then failing to do so. This phenomenon is known as routing misbehavior. A node may misbehave because it may be overloaded node, selfish node, malicious node and broken node.

In the present work, a study of routing misbehavior of Ad hoc on Demand Distance Vector (AODV) routing algorithm in the following network context has been done:

- Network size
- Bandwidth
- Traffic pattern
- Mobility rate
- Battery power

In the situation of routing misbehavior the performance of the AODV routing algorithm has been evaluated. The parameters for evaluation are :

- End-to-End Delay
- Delivery Rate
- Overhead Ratio
- Percentage Bandwidth Utilization

Routing Misbehavior 2

Ad hoc networks maximize total network throughput by using all available nodes for routing and forwarding. Therefore, the more nodes that participates in packet routing, the greater the aggregate bandwidth, the shorter the possible routing paths, and the smaller the possibility of a network partition. However, misbehaving nodes could be a significant problem. A node may misbehave by agreeing to forward packets and then failing to do so, because it is overloaded, selfish, malicious, or broken.

- An overloaded node lacks the CPU cycle, buffer space or available network bandwidth to forward packets.
- A selfish node is unwilling to spend battery life, CPU cycle or available network bandwidth to forward packets not of direct interest to it, even though it expects others to forward packets on its behalf.
- A malicious node launches a denial-of-service attack by dropping packets.
- A broken node might have a software fault that prevents it from forward packets.

Misbehaving nodes can be a significant problem. Our simulation shows that even for 10%- 30% routing misbehavior in the network there is a severe impact on the overall performance of the network.

2.1 Detection of routing misbehavior in MANET

Current versions of mature ad hoc routing algorithms, like DSR[5], AODV[2], TORA[14], DSDV[3], and others only detects if the receiver's network interface is accepting packets, but they otherwise assume that the routing nodes do not misbehave. Sergio Marthi, T.J. Giuli, Kevin Lai, and Mary Baker[13] have proposed certain solutions to handle misbehaving nodes.

One of the solution is to forward packets only through nodes that share an a priori trust relationship. A priori trust relationship are based on pre-existing

relationships build outside of the context of the network (e.g. friendship, companies, and armies) . However this solution has some drawbacks. It require key distribution. Moreover trusted nodes can be overloaded or it can be compromised.

Another solution to misbehaving node is to attempt to forestall or isolate these nodes from within the actual routing protocol for the network. However, this would add significant complexity to protocols whose behavior must be very well defined.

Due to the drawbacks and complexities of the above stated solutions, they presented a different approach. In this approach, they introduced two extensions to the Dynamic Source Routing protocol to mitigate the effect of routing misbehavior – *Watchdog* and *Pathrater*. Though they implemented these tools on top of DSR[5], some of the concepts can be generalized to other source routing protocol.

2.1.1 Watchdog

The watchdog method detects misbehaving nodes. Figure 2.1 illustrates how the watchdog works. Suppose there exists a path from node S to D through intermediate nodes A, B, and C. Node A cannot transmit all the way to node C, but it can listen in on node B's outgoing traffic. Thus, when A transmits a packet for B to forward to C, A can often tell if B transmitted the packet. A buffer of recently sent packets are maintained and compared with each overheard packet to see if there is a match. If , so the packet in the buffer is removed and forgotten by the watchdog, since it has been forwarded on. If a packet has remained in the buffer for longer than a certain timeout period, the watchdog increments a failure tally for the node responsible for forwarding on the packet. If the tally exceeds a certain threshold bandwidth, it determines that the node is misbehaving and sends a message to the source notifying it of the misbehaving node.

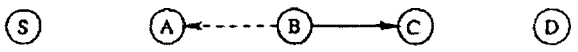

Figure 2.1

The watchdog technique has advantages and weaknesses. This technique has t the advantage that it can detect misbehavior at the forwarding level and not just the

link level. Watchdog's weaknesses are that it might not detect a misbehaving node in the presence of the following:

- Ambiguous collisions
- Receiver collisions
- False misbehavior
- Collusion
- Partial dropping

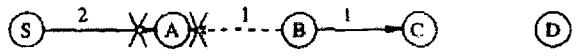

Figure 2.2

The *Ambiguous collision problem* prevents A from overhearing transmission from B. As figure 2.2 illustrates, a packet collision can occur at A while it is listening for B to forward on a packet. A does not know if the collision was caused by B forwarding on a packet as it should or if B never forwarded the packet and the collision was caused by other nodes in A's neighborhood. Because of this uncertainty, A should not immediately accuse B of misbehaving, but should instead continue to watch B over a period of time. If A repeatedly fails to detect b forwarding on packets, then A can assume that B is misbehaving.

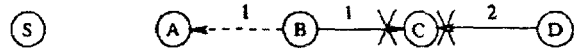

Figure 2.3

In the *Receiver collision problem,* node A can only tell whether B sends the packet to C, but it cannot tell if C receives it (figure 2.3). If a collision occurs at C when B first forwards the packet, A only sees B forwarding the packet and assumes that C successfully receives it. Thus, B should skip retransmitting the packet. B could also purposefully cause the transmitted packet to collide at C by waiting until C is transmitting and then forwarding on the packet. In the first case, a node could be selfish and not want to waste power with retransmissions. In the latter case, the only reason B would have for taking the actions that it does is because it is malicious. B waste battery power and CPU time, so it is not selfish. An overloaded node would not

engage in this behavior either, since it waste badly needed CPU time and bandwidth. Thus, this case should be a rare occurrence.

False misbehavior problem, occurs when node falsely report other nodes as misbehaving. A malicious node could attempt to partition the network by claiming that some nodes following it in the path are misbehaving. For instance, node A could report that node B is not forwarding packets when in fact it is. This cause S to mark B as misbehaving even though A is the culprit. This behavior, however will be detected. Since A is passing message on to B (as verified by S), then any acknowledgements from D to S will go through A to S, and S will wonder why it receives replica from D when supposedly B dropped packets in the forward direction. In addition, if A drop acknowledgements to hide from S, then node B will detect this misbehavior and will report it to D. But this problem can't be solved if the packets move over UDP as no acknowledgement will be generated.

In *Partial dropping problem,* a node can circumvent the watchdog by dropping packets at a lower rate than the watchdog's configured minimum misbehavior threshold. Although the watchdog will not detect this node as misbehaving, this node is forced to forward at the threshold bandwidth in other to prevent itself from been identified itself as a misbehaving node.

2.1.2 Pathrater

The pathrater, runs on each node in the network, combining knowledge of misbehaving nodes with link reliability data to pick the route most likely to be reliable. Each node maintains a rating for every other node it knows about in the network. It calculates a path metric by averaging the node ratings in the path. Metric was chosen because it gives a comparison of the overall reliability of different paths and allows pathrater to emulate the shortest length path algorithm when no reliability information has been collected, as explained below. If there are multiple paths to the same destination, the path with the highest metric is chosen. Since the pathrater depends on knowing the exact path a packet has traversed, it must be implemented on top of a source routing protocol like DSR[5].

The pathrater assigns ratings to nodes according to the following algorithm. When a node in the network becomes known to the pathrater (through route discovery), the pathrater assigns it a "neutral" rating of 0.5. A node always rates itself with a 1.0. This ensures that when calculating path rates, if all other nodes are neutral nodes (rather than suspected misbehaving nodes), the pathrater picks the shortest length path. The pathrater increments the ratings of nodes on all actively used paths by 0.01 at periodic intervals of 200 ms. An actively used path is one on which the node has sent a packet within the previous rate increment interval. The maximum value a neutral node can attain is 0.8. We decrement a node's rating by 0.05 when we detect a link break during packet forwarding and the node becomes unreachable. The lower bound rating of a "neutral" node is 0.0. The pathrater does not modify the ratings of nodes that are not currently in active use.

A special highly negative value, is assigned to nodes suspected of misbehaving by the watchdog mechanism. When the pathrater calculates the path metric, negative path values indicate the existence of one or more suspected misbehaving nodes in the path. If a node is marked as misbehaving due to a temporary malfunction or incorrect accusation it would be preferable if it were not permanently excluded from routing. Therefore nodes that have negative ratings should have their ratings slowly increased or set back to a non-negative value after a long timeout.

When the pathrater learns that a node on a path that is in use misbehaves, and it cannot find a path free of misbehaving nodes, it sends out a ROUTE REQUEST.

The Ad-hoc On-Demand Distance Vector Algorithm 3

3.1 Introduction

The Ad Hoc On-Demand Distance Vector (AODV)[2] 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. AODV allows mobile nodes to respond quickly to link breakages and changes in network topology. The operation of AODV is loop-free, and by avoiding the Bellman-Ford ``counting to infinity" problem offers quick convergence when the ad hoc network topology changes (typically, when a node moves in the network). When links break, AODV causes the affected set of nodes to be notified so that they are able to invalidate the routes using the broken link.

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 for any route information it sends to requesting nodes. Using destination sequence numbers ensures loop freedom and is simple to program. Given the choice between two routes to a destination, a requesting node always selects the one with the greatest sequence number.

3.2 Overview

As long as the endpoints of a communication connection have valid routes to each other, AODV does not play any role. Nodes that do not lie on active paths neither maintain any routing information nor participate in any periodic routing table exchanges. Further, a node does not have to discover and maintain a route to another node until the two need to communicate, unless the former node is offering its

services as an intermediate forwarding station to maintain connectivity between two other nodes. When the local connectivity of the mobile node is of interest, each mobile node can become aware of the other nodes in its neighborhood by the use of several techniques, including local (not system-wide) broadcasts known as hello messages. The routing tables of the nodes within the neighborhood are organized to optimize response time to local movements and provide quick response time for requests for establishment of new routes. The algorithm's primary objectives are:

- To broadcast discovery packets only when necessary
- To distinguish between local connectivity management (neighborhood detection) and general topology maintenance
- To disseminate information about changes in local connectivity to those neighboring mobile nodes that are likely to need the information.

AODV uses a broadcast route discovery mechanism , as is also used (with modifications) in the Dynamic Source Routing (DSR) algorithm [5]. Instead of source routing, however, AODV relies on dynamically establishing route table entries at intermediate nodes. This difference pays off in networks with many nodes, where a larger overhead is incurred by carrying source routes in each data packet. To maintain the most recent routing information between nodes, we borrow the concept of destination sequence numbers from DSDV [3]. Unlike in DSDV, however, each ad hoc node maintains a monotonically increasing sequence number counter which is used to supersede stale cached routes. The combination of these techniques yields an algorithm that uses bandwidth efficiently (by minimizing the network load for control and data traffic), is responsive to changes in topology, and ensures loop-free routing.

3.3 Generating Route Requests

A node broadcasts a *Route Request Packet* (RREQ) when it determines that it needs a route to a destination and does not have one available. This can happen if the destination is previously unknown to the node, or if a previously valid route to the destination expires or is broken. The RREQ contains the following fields:

{ Source IP Address; Source Sequence No; Broadcast id;
Destination IP Address; Destination Sequence No; Hop Count }

The Destination Sequence Number field in the RREQ message is the last known destination sequence number for this destination and is copied from the Destination Sequence Number field in the routing table (see section 3.10). If no sequence number is known, a sequence number of zero is used. The Source Sequence Number in the RREQ message is the node's own sequence number. The Broadcast ID field is incremented by one from the last broadcast ID used by the current node. Each node maintains only one broadcast ID. The Hop Count field is set to zero.

After broadcasting a RREQ, a node waits for a *Route Reply Packet* (RREP) (see section 3.5). If the RREP is not received within NET_TRAVERSAL_TIME milliseconds, the node MAY rebroadcast the RREQ, up to a maximum of RREQ_RETRIES times. Each rebroadcast MUST increment the Broadcast ID field.

3.3.1 Controlling Route Request Broadcasts

To prevent unnecessary network-wide broadcasts of RREQs, the source node SHOULD use an expanding ring search technique as an optimization. In an expanding ring search, the source node initially uses a TTL = TTL_START in the RREQ packet IP header and sets the timeout for receiving a RREP to 2 * TTL * NODE_TRAVERSAL_TIME milliseconds. Upon timeout, the source rebroadcasts the RREQ with the TTL incremented by TTL_INCREMENT. This continues until the TTL set in the RREQ reaches TTL_THRESHOLD, beyond which a TTL = NET_DIAMETER is used for each rebroadcast. Each time, the timeout for receiving a RREP is calculated as before. Each rebroadcast increments the Broadcast ID field in the RREQ packet. The RREQ can be rebroadcast with TTL = NET_DIAMETER up to a maximum of RREQ_RETRIES times.

3.4 Forwarding Route Requests

When a node receives a broadcast RREQ, it first checks to determine whether it has received a RREQ with the same Source IP Address and Broadcast ID within at least the last BROADCAST_RECORD_TIME milliseconds. If such a RREQ has been

received, the node silently discards the newly received RREQ. The rest of this subsection describes actions taken for RREQs that are not discarded.

3.4.1 Processing Route Request

When a node receives a RREQ, the node checks to determine whether it has an active route to the destination. If the node does not have an active route, it rebroadcasts the RREQ from its interface(s) but using its own IP address in the IP header of the outgoing RREQ. The Destination Sequence Number in the RREQ is updated to the maximum of the existing Destination Sequence Number in the RREQ and the destination sequence number in the routing table (if an entry exists) of the current node. The TTL or hop limit field in the outgoing IP header is decreased by one. The Hop Count field in the broadcast RREQ message is incremented by one, to account for the new hop through the intermediate node.

If the node, on the other hand, does have an active route for the destination, it compares the destination sequence number for that route with the Destination Sequence Number field of the incoming RREQ. If the existing destination sequence number is smaller than the Destination Sequence Number field of the RREQ, the node again rebroadcasts the RREQ just as if it did not have an active route to the destination.

The node generates a RREP (as discussed further in section 3.5) if either:

- it has an active route to the destination, and the node's existing destination sequence number is greater than or equal to the Destination Sequence Number of the RREQ, or
- it is itself the destination.

The node always creates or updates a reverse route to the Source IP Address in its routing table. If a route to the Source IP Address already exists, it is updated only if either

- the Source Sequence Number in the RREQ is higher than the destination sequence number of the Source IP Address in the route table, or
- the sequence numbers are equal, but the hop count as specified by the RREQ is now smaller than the existing hop count in the routing table.

When a reverse route is created or updated, the following actions are carried out:

- the Source Sequence Number from the RREQ is copied to the corresponding destination sequence number;
- the next hop in the routing table becomes the node broadcasting the RREQ (it is obtained from the source IP address in the IP header and is often not equal to the Source IP Address field in the RREQ message);
- the hop count is copied from the Hop Count in the RREQ message;
- the lifetime of the route is the higher of its current lifetime (for an active route) and current time plus REV_ROUTE_LIFE.

This reverse route would be needed in case the node receives an eventual RREP back to the node which originated the RREQ (identified by the Source IP Address).

3.5 Generating Route Replies

If a node receives a route request for a destination, and either has a fresh enough route to satisfy the request or is itself the destination, the node generates a RREP message and unicasts it back to the node indicated by the Source IP Address field of the received RREQ. A RREP contains the following information :

{ Source IP Address; Destination IP Address; Destination Sequence No; Hop Count; lifetime }

The node generating the RREP message copies the Source and Destination IP Addresses in RREQ message into the corresponding fields in the RREP message which is to be sent back toward the source of the RREQ. Additional operations are slightly different, depending on whether the node is itself the requested destination, or instead if it is an intermediate node with an admissible route to the destination.

As the RREP is forwarded to the source, the Hop Count field is incremented by one at each hop. Thus, when the RREP reaches the source, the Hop Count represents the distance, in hops, of the destination from the source.

3.5.1 Route Reply Generation by the Destination

If the generating node is the destination itself, it uses a destination sequence number at least equal to a sequence number generated after the last detected change in its neighbor set and at least equal to the destination sequence number in the RREQ. If the

destination node has not detected any change in its set of neighbors since it last incremented its destination sequence number, it MAY use the same destination sequence number. The destination node places the value zero in the Hop Count field of the RREP.

3.5.2 Route Reply Generation by an Intermediate Node

If node generating the RREP is not the destination node, but instead is an intermediate hop along the path from the source to the destination, it copies the last known destination sequence number in the Destination Sequence Number field in the RREP message. The intermediate node places its distance in hops from the destination (indicated by the hop count in the routing table) plus one in the Hop Count field in the RREP.

When the intermediate node updates its route table for the source of the RREQ, it puts the last hop node (from which it received the RREQ, as indicated by the source IP address field in the IP header) into the precursor list for the forward path route entry i.e., the entry for the Destination IP Address. Furthermore, the intermediate node puts the next hop towards the destination in the precursor list for the reverse route entry i.e., the entry for the Source IP Address field of the RREQ message data.

3.6 Forwarding Route Replies

When a node receives a RREP message, it first compares the Destination Sequence Number in the message with its own copy of destination sequence number for the Destination IP Address in the RREP message. The forward route for this destination is created or updated only if

- the Destination Sequence Number in the RREP is greater than the node's copy of the destination sequence number, or
- the sequence numbers are the same, but the route is no longer active or the Hop Count in RREP is smaller than the hop count in route table entry.

If a new route is created or the old route is updated, the next hop is the node from which the RREP is received, which is indicated by the source IP address field in the

IP header; the hop count is the Hop Count in the RREP message plus one; the expiry time is the current time plus the Lifetime in the RREP message; the destination sequence number is the Destination Sequence Number in the RREP message. The current node can now begin using this route to send data packets to the destination.

When any node generates or forwards a RREP, the precursor list for the corresponding destination node is updated by adding to it the next hop node to which the RREP is forwarded. Also, at each node the (reverse) route used to forward a RREP has its lifetime changed to current time plus ACTIVE_ROUTE_TIMEOUT.

3.7 Local Connectivity Management

Nodes learn of their neighbors in one of two ways. Whenever a node receives a broadcast from a neighbor, it updates its local connectivity information to ensure that it includes this neighbor. In the event that a node has not sent any packets to all of its active downstream neighbors within hello interval, it broadcasts to its neighbors a hello message (a special unsolicited RREP), containing its identity and sequence number. The node's sequence number is not changed for hello message transmissions. This hello message is prevented from being rebroadcast outside the neighborhood of the node because it contains a time to live (TTL) value of 1. Neighbors that receive this packet update their local connectivity information to the node. Receiving a broadcast or a hello from a new neighbor, or failing to receive allowed hello loss consecutive hello messages from a node previously in the neighborhood, is an indication that the local connectivity has changed. Failing to receive hello messages from inactive neighbors does not trigger any protocol action. If hello messages are not received from the next hop along an active path, the active neighbors using that next hop are sent notification of link failure as described in Section 3.3. The local connectivity management with hello messages can also be used to ensure that only nodes with bi-directional connectivity are considered to be neighbors. For this purpose, each hello sent by a node lists the nodes from which it has heard. Each node checks to make sure that it uses only routes to neighbors that have heard the node's hello message. To save local bandwidth, such checking should be performed only if explicitly configured into the nodes.

3.8 Hello Messages

Nodes learn of their neighbors in one of two ways. Whenever a node receives a broadcast from a neighbor, it updates its local connectivity information to ensure that it includes this neighbor. In the event that a node has not sent any packets to all of its active downstream neighbors within hello interval, it broadcasts to its neighbors a hello message. 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 generate a broadcast RREP with TTL = 1, called a Hello message, with the message fields set as follows:

Destination IP Address : The node's IP address.

Destination Sequence Number : The node's latest sequence number.

Hop Count : 0

Lifetime : ALLOWED_HELLO_LOSS * HELLO_INTERVAL

A node MAY determine connectivity by listening for packets from its set of neighbors. If it receives no packets for more than
ALLOWED_HELLO_LOSS * HELLO_INTERVAL milliseconds,
the node SHOULD assume that the link to this neighbor is currently broken. When this happens, the node SHOULD proceed as in Section 3.9.

3.9 Maintaining Local Connectivity

Each forwarding node SHOULD keep track of its active next hops (i.e., which next hops have been used to forward packets towards some destination within the last ACTIVE_ROUTE_TIMEOUT milliseconds). This is done by updating the Lifetime field of a routing table entry used to forward data packets to current time plus ACTIVE_ROUTE_TIMEOUT milliseconds. For purposes of efficiency, each node may try to learn which of these active next hops are really in the neighborhood at the current time using one or more of the available link or network layer mechanisms.

If a link to the next hop cannot be detected, the forwarding node SHOULD assume that the link is broken, and take corrective action by following the methods specified in Section 3.10.

3.10 Route Error Messages

A *Route Error Packet* (RERR) contains the following information:

{ Unreachable Destination IP Address; Unreachable Destination Sequence no; Destcount }

A node initiates a *Route Error* (RERR) message in the following situations:

i. if it detects a link break for the next hop of an active route in its routing table, or

ii. if it gets a data packet destined to a node for which it does not have an active route

For cases (i) and (ii), the destination sequence numbers in the routing table for the unreachable destination(s) are incremented by one. Then RERR is broadcast with the unreachable destination(s) and their incremented destination sequence number(s) included in the packet. For case (i), the unreachable destinations are the broken next hop, and any additional destinations which are now unreachable due to the loss of this next hop link. These additional destinations are those that also use the lost link as next hop in the routing table. For case (ii), there is only one unreachable destination, which is the destination of the data packet that cannot be delivered. The DestCount field of the RERR packet indicates the number of unreachable destinations included in the packet.

For cases (i) and (ii), for each unreachable destination the node copies the value in the Hop Count route table field into the Last Hop Count field, and marks the Hop Count for this destination as infinity, and thus invalidates the route.

When a node broadcasts a RERR message, it always deletes the precursor list of each unreachable destination included in the message. When a node invalidates a route to a neighboring node, it must also delete that neighbor from any precursor lists for routes to other nodes. This prevents precursor lists from containing stale entries of neighbors with which the node is no longer able to communicate. The node should inspect the precursor list of each destination entry in its routing table, and delete the lost neighbor from any list in which it appears.

3.11 Maintaining Routing Table

For each valid route maintained by a node (containing a finite Hop Count metric) 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.

Each time a route is used to forward a data packet, its Lifetime field is updated to be current time plus ACTIVE_ROUTE_TIMEOUT.

3.12 Configuration Parameters

This section gives default values for some important values associated with AODV protocol operations. A particular mobile node may wish to change certain of the parameters, in particular the NET_DIAMETER, NODE_TRAVERSAL_TIME, MY_ROUTE_TIMEOUT, ALLOWED_HELLO_LOSS, RREQ_RETRIES, and possibly the HELLO_INTERVAL. Choice of these parameters may affect the performance of the protocol.

Parameter	Value
ACTIVE_ROUTE_TIMEOUT	3,000 Milliseconds
ALLOWED_HELLO_LOSS	3
BROADCAST_RECORD_TIME	2 * NET_TRAVERSAL_TIME
HELLO_INTERVAL	1,000 Milliseconds
LOCAL_ADD_TTL	2
MY_ROUTE_TIMEOUT	2 * ACTIVE_ROUTE_TIMEOUT
NET_DIAMETER	35
NEXT_HOP_WAIT	NODE_TRAVERSAL_TIME + 10
NODE_TRAVERSAL_TIME	40
REV_ROUTE_LIFE	NET_TRAVERSAL_TIME

NET_TRAVERSAL_TIME :

3 * NODE_TRAVERSAL_TIME * NET_DIAMETER) / 2

RREQ_RETRIES	2
TTL_START	1
TTL_INCREMENT	2
TTL_THRESHOLD	7

Network Simulator And Simulation Methodology 4

The simulation software used in this thesis is network simulator (ns) version 2.1b8a on a Linux platform. Reason for choosing network simulator was its ability to supports mobile ad hoc routing algorithms.

4.1 Overview

Network simulator(ns) is a event driven simulator developed at UC Berkeley[9,6,10] that simulates variety of IP networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra. and more. Currently, NS (version 2) written in C++ and OTcl (Tcl script language with Object-oriented extensions developed at MIT) is available.

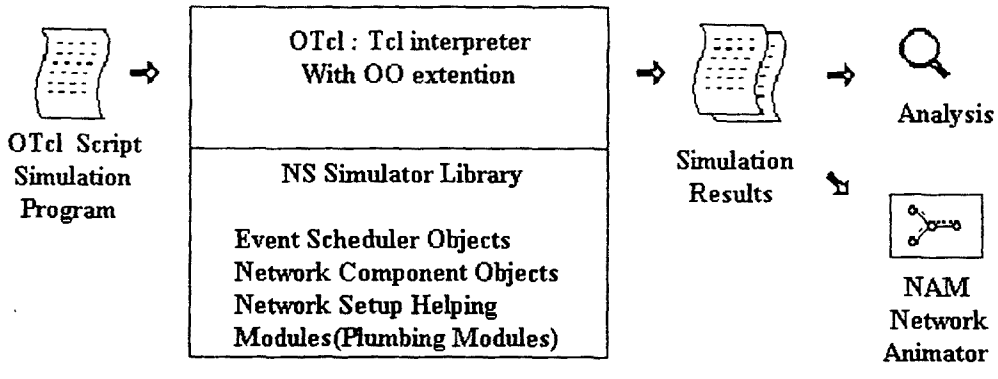

Figure 4.1 Simplified User's View of NS

As shown in Figure 4.1, in a simplified user's view, NS is Object-oriented Tcl (OTcl) script interpreter that has the following three components :

- simulation event scheduler,
- network component object libraries, and
- network setup (plumbing) module libraries

To use NS, simulation script is written in OTcl script language. To setup and run a simulation network, a user write an OTcl script that initiates an event scheduler, sets

up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler.

- **Plumbing Module** - The term "plumbing" is used for a network setup, because setting up a network is plumbing possible data paths among network objects by setting the "neighbor" pointer of an object to the address of an appropriate object. When a user wants to make a new network object, he or she can easily make an object either by writing a new object or by making a compound object from the object library, and plumb the data path through the object. This may sound like complicated job, but the plumbing Otcl modules actually make the job very easy. The power of NS comes from this plumbing.
- **Event Scheduler** - Any event in NS has a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with packet pointed by the event. Network components communicate with one another passing packets, however this does not consume actual simulation time.
- **Network Component** - All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet. For example, a network switch component that simulates a switch with 20 microseconds of switching delay issues an event for a packet to be switched to the scheduler as an event 20 microsecond later. The scheduler after 20 microsecond dequeues the event and fires it to the switch component, which then passes the packet to an appropriate output link component.

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the

basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the Otcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object. In this way, the controls of the C++ objects are given to OTcl. It is also possible to add member functions and variables to a C++ linked OTcl object. The objects in C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to OTcl. Likewise, an object (not in the data path) can be entirely implemented in OTcl.

Figure 4.2 shows an object hierarchy example in C++ and OTcl. One thing to note in the figure is that for C++ objects that have an OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.

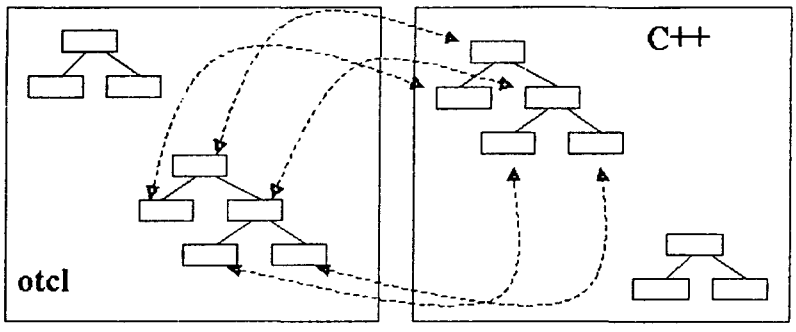

Figure 4.2 C++ and Otcl : The Duality

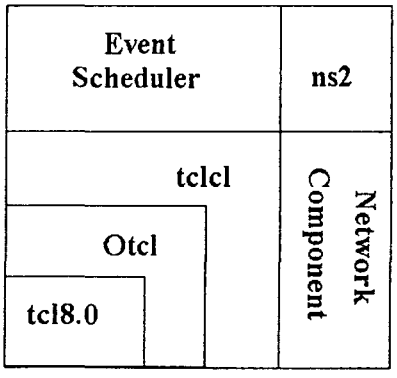

Figure 4.3 Architectural View of NS

Figure 4.3 shows the general architecture of NS. In this figure a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and

running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is an OO extended Tcl interpreter with network simulator libraries.

This section briefly examined the general structure and architecture of NS. At this point, one might be wondering about how to obtain NS simulation results. As shown in Figure 4.1, when a simulation is finished, NS produces one or more text-based output files called trace file that contain detailed simulation data , if specified to do so in the input Tcl (or more specifically, OTcl) script. The data can be used for simulation analysis or as an input to a graphical simulation display tool called Network Animator (NAM). NAM has a nice graphical user interface and also has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis.

4.2 Extensions to Network simulator

The present version of network simulator does not support routing misbehavior. The present version support Error model, through which we can make each node to abruptly drop certain percentage of packets. But this error model cannot be used to induce routing misbehavior. According to the definition of routing misbehavior, a misbehaving node is one that agrees to participate in forwarding packets but then indiscriminately drops all data packets that are routed through it.

So, to induce routing misbehavior in our simulation we have created a new variable in the node structure that can have either 0 or 1 value. If the value of the variable is 1 then it means that the node is a misbehaving node. The number of nodes whose value is set 1 depends on the percentage of nodes misbehaving. This variable value is used to decide whether to drop the packet or not, after the route discovery has been made. When a new packet is received by the node, it checks the value of the new variable before forwarding the packet. If the value of the new variable is 0 then it simply forwards the packet to next node. On the other hand, if the value of that new variable is 1 then it starts misbehaving i.e. it drops the packet.

A new traffic generation OTcl script has also been written in order to generate a constant bit rate(CBR) [9] traffic over TCP. This script generate the specified number of connections between randomly selected nodes.

4.3 Simulation Methodology

For our simulation purpose, we used a version of Berkeley's Network simulator (ns)[9] that include wireless extension made by the CMU Monarch project. We also used a visualization tool called Network Animator to view the result of our simulations and to detect overall trends in the network.

In our simulation, we varied the percentage of routing misbehaviour from 0% to 30% in 10% increments. The effect of routing misbehavior has been studied by varying the following parameters.

- **Network size :** In our simulation, we have varied the number of wireless nodes, scattered randomly in a 670 by 670 meter flat space. Number of wireless mobile nodes chosen are 25, 50, 75 and 100.
- **Bandwidth :** Bandwidth between each connection is set up at the link layer. The bandwidth chosen in our simulation is 1 Mbps as supported by 802.11 standards[4].
- **Traffic Pattern :** In our simulation, nodes communicate using constant bit rate (CBR)[9] node-to-node connection established over TCP. A constant packet size of 1020 bytes is used. In order to create low, heavy and busty traffic pattern, the CBR sends the packet at different packet rates. Packet rates in our simulation are 5, 50, and 120 packets /sec.
- **Mobility Rate :** In our node movement scenario, the node choose a random destination and more in a straight line towards the destination at a constant speed. Once the node reaches its destination, it waits for the pause time of 2.0 sec before again choosing another random destination and repeat the process. The maximum speed with which the nodes move in our simulation are 0, 5, 10 and 25 m/s.

We evaluated the effect of routing misbehavior using the following parameters

- **End-to-End Delay :** The term end-to-end delay is used as an average measure of performance between nodes in a network. It is the sources and the receivers that are involved. The end-to-end delay is therefore the total delay that a data packet experiences as it is traveling through a network. This delay is build up by several smaller delays in the network that adds together. These delays might be time spent in route discovery, packet queues, forwarding delays, propagation delay (the time it takes for the packet to travel through the medium) and time needed to make retransmissions if a packet got lost etc.
- **Packet delivery ratio :** The packet delivery ratio presents the ratio between the number of packets send CBR and the number of packets actually received at the destination nodes.
- **Bandwidth utilization :** It is desirable that a routing protocol keeps this rate at a high level since efficient bandwidth utilization is important in wireless networks where available bandwidth is a limiting factor. This is an important metric because it revels the loss rate seen by the transport protocols and also characterizes the completeness and correctness of the routing protocol.
- **Overhead Ratio :** It is the ratio of the amount of traffic generated by the control packets to the actual data send. As mobility in the network increase reactive protocols will have to send more routing message and thus have a higher overhead ratio. This is where the real strength or weakness of the routing protocol can be revealed.

For simulation analysis the trace file was analysed using the UNIX script and C code. This code was written in order to extract meaningful data from the trace file to calculate the End-to-End delay, Delivery rate, Overhead ratio and percentage bandwidth utilization.

Simulation Results 5

Based on the simulation methodology described in chapter 4, simulation was done in network simulator(ns)[9]. All the simulations were performed on a Pentium II (400 Mhz) machine with 64 MB RAM.

The output of the simulation is in the form of a network animator(NAM) trace file that is going to be used as an input to NAM and a simulation trace file that will be used for our simulation analysis. Section 5.1 shows the trace format.

We have used traffic-pattern and node-movement generation files available with the ns, to generate different traffic pattern and node movement scenario files. We have made certain changes to the traffic-pattern generation file, in order to generate a CBR traffic over TCP[9]. These scenario files are used as the input to our simulation script to simulate different traffic pattern and mobility rate.

Different parameters that are used for all our wireless simulation are described below:

- **Queuing Delay** : Priority queue is used which gives priority to routing protocol packets, inserting them at the head of the queue.
- **Radio Propagation Model** : Two Ray Ground propagation model has been used. It uses Friss-space attenuation $1/r^2$ at near distances and an approximation to Two Ray Ground $1/r^4$ at far distances. The approximation assumes specular reflection off a flat ground plane.
- **Antenna** : An omni-directional antenna having unity gain is used by mobile nodes.
- **Network Interfaces** : The Network Interphase layer serves as a hardware interface which is used by mobile nodes to access the channel. The wireless shared media interface used is WirelessPhy. This interface subject to collisions and the radio propagation model receives packets transmitted by other node interfaces to the channel.

- **Mac Layer :** The IEEE 802.11 Mac protocol has been used. It uses a RTS/CTS/DATA/ACK pattern for all unicast packets and simply sends out DATA for all broadcast packets.

5.1 Format of Simulation Trace file

The trace format generated by the network simulator is as shown below:

r -t 10.000000000 -Hs 0 -Hd -2 -Ni 0 -Nx 5.00 -Ny 2.00 -Nz 0.00 –Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 1.0 -It tcp -Il 1000 –If 2 -Ii 2 -Iv 32 -Pn tcp -Ps 0 -Pa 0 -Pf 0 -Po 0

The trace format as seen above can be can be divided into the following fields :

- **Event type** In the traces above, the first field describes the type of event taking place at the node . It can be **s** for send, **r** for receive, **d** for drop, **f** for forward.
- **General tag** The second field starting with "-t" stands for time.
- **Next hop info** This field provides next hop info and the tag starts with a leading "-H".

  -Hs: id for this node

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

  -Ni: node id

  -Nx: node's x-coordinate

  -Ny: node's y-coordinate

  -Nz: node's z-coordinate

  -Ne: node energy level

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

  -Nw: reason for the event. Example:

  "TOUT" DROP_RTR_QTIMEOUT i.e. packet has expired.
- **Packet information at IP level** The tags for this field start with a leading "-I" and are listed along with their explanations as following:

-**Is:** source address.source port number

-**Id:** dest address.dest port number

-**It:** packet type

-**Il:** packet size

-**If:** flow id

-**Ii:** unique id

-**Iv:** ttl value

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

  -**Ma:** duration

  -**Md:** dst's ethernet address

  -**Ms:** src's ethernet address

  -**Mt:** ethernet type

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

  -**P tcp** Information about TCP flow is given by the following sub tags:

  -**Ps:** seq number

  -**Pa:** ack number

  -**Pf:** how many times this pkt was forwarded

  -**Po:** optimal number of forwards

5.2 Simulation Variable

The different values of simulation variables that are used in our simulation work are

- Network size 25,50,75 and 100 Nodes
- Mobility Rate 0.5,10 and 25 m/sec
- Traffic Pattern 5. 50 and 120 packets/sec

Area that we are you using for simulation is 670 X 670 meter flat space and bandwidth of 1Mbps. At a time we change one variable and others are constant. Constant values of simulation variable are Network size - 50 Nodes, Mobility Rate - 5m/s, Traffic Pattern - 5 packets/sec.

5.3 Simulation Results

In this section we present the results of all our simulations. The parameters that are used for evaluation are described below. Due to time constraints routing misbehavior has not been induced in ns by making modification to its source code. Therefore we used a different strategy. In this strategy we have induced misbehavior by configuring the network for 10 % - 30 % packet drops uniformly distributed over all the nodes.

5.3.1 End-to-End Delay

End-to-End delay is analyzed with varying simulation variables one by one. For example, when we vary network size other variable are kept constant. This is repeated for other variable as well. When we change Network Size from 25 to 50, 75 and 100, the other variables remain constant with Mobility Rate - 5m/s, Traffic Pattern - 5 pkts/sec, Bandwidth - 1Mbps. The figures 5.1, figure 5.2, and figure 5.3 shows the End-to-End delay of the network for variable network size, traffic pattern and mobility rate respectively.

- **Effect of Network size on End-to-End Delay**

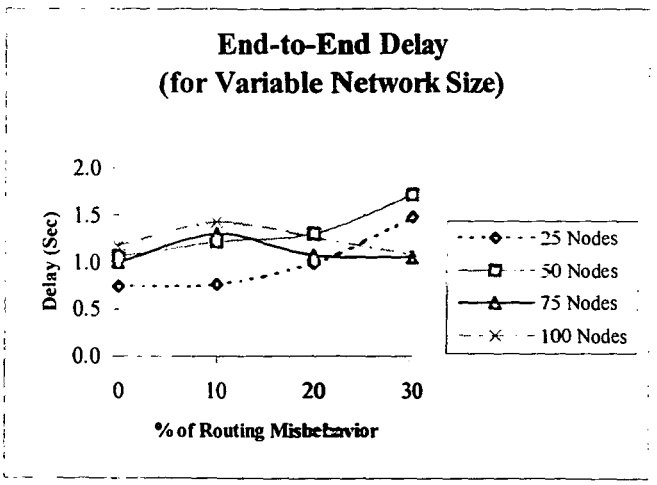

Fig 5.1 Overall network End-to-End Delay as a function of percentage of routing misbehavior.

As expected the End-to-End delay increased as the percentage of routing misbehavior increases for the network with 25 and 50 nodes. But as the number of nodes in the network increases to 75 and 100 , the End-to-End delay does not show

any major changes. This could be because of the excess number of nodes in the network and the source has multiple routes to the destination. Thus End-to-End delay does not show any major change as the node starts misbehaving for network with very high number of mobile nodes.

Since we have used priority queue, the control packets are added at the head of the queue. As the number of nodes in a network increases the number of uncontrolled routing control packets increases exponentially. So this leads to increased queuing delay for the data packets and hence increase in End-to-End delay with respect to the increase in network size.

- **Effect of Traffic Pattern on End-to-End Delay**

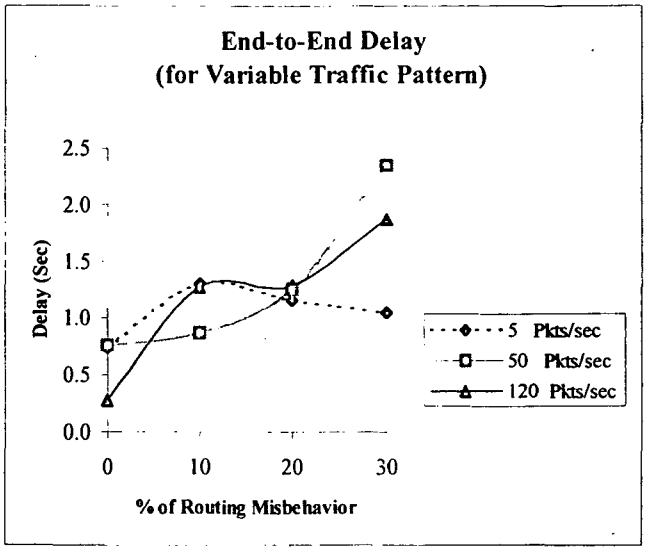

Fig 5.2 Overall network End-to-End Delay as a function of percentage of Routing Misbehavior.

The End-to-End delay is significantly low for the bursty traffic, since after the route discovery has been done a greater number of packets can be send. Also as the percentage of routing misbehavior increases the End-to-end delay rapidly increases. But for the low traffic of 5 packets/sec, there is not a significant change in End-to-End delay. This may be because the time it takes for the source to discover the new route is not much more than the rate of packet generation.

- **Effect of Mobility rate on End-to-End Delay**

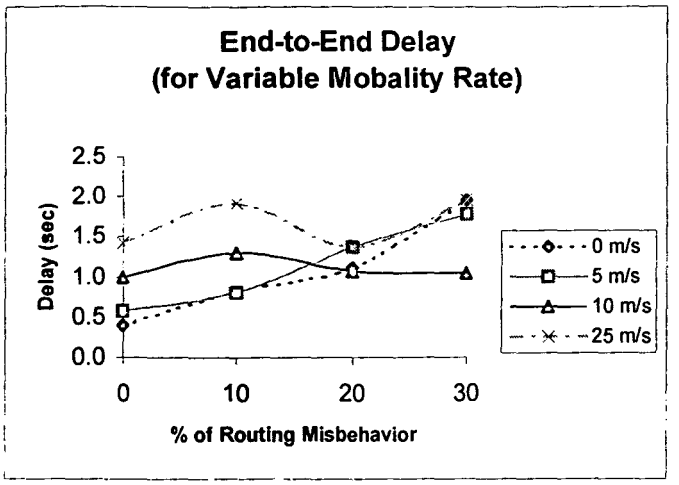

Fig 5.3 Overall network End-to-End Delay as a function of percentage of Routing Misbehavior.

End-to-End delay is increasing significantly as mobility rate increases from 0 to 25 m/s when there is no misbehaving node. As the percentage of routing misbehavior increases the End-to-End delay increases sharply for low mobility of 0 and 5 m/s as expected. But for high mobility rate there has been no relevant change in End-to-End delay as the percentage of routing misbehavior increases, because at high mobility rate the route change is much more frequent.

5.3.2 Delivery Rate

Delivery rate is analyzed with varying simulation variables one by one. For example, when we vary network size other variable are kept constant. This is repeated for other variable as well. When we changes Network Size from 25 to 50, 75 and 100, the other variables remain constant with Mobility Rate - 5m/s, Traffic Pattern - 5 pkts/sec, Bandwidth – 1Mbps.

The figures 5.4, figure 5.5, and figure 5.6 shows the delivery rate of the network for variable network size, traffic pattern and mobility rate respectively. As the percentage of routing misbehavior increases from 0% to 30%, all the three factors i.e. Network size, Traffic Pattern and Mobility rate shows a similar pattern. All three factors show a decline in delivery rate(figure 5.4, figure 5.5, figure 5.6). However the delivery rate remains at a very high value even when 30% of the node misbehave

because the traffic source in our simulation i.e. CBR is implemented over TCP. Thus if TCP ACK does not come back to the source, it again sends the packet.

- **Effect of Network size on Delivery Rate**

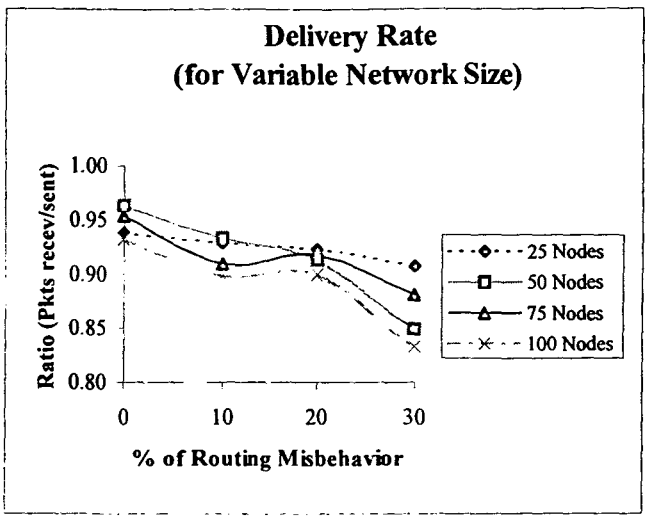

Fig 5.4 Overall network Delivery Rate as a function of percentage of Routing Misbehavior.

The delivery rate varies from 0.83 to 0.90 even when 30% of the nodes misbehave.

- **Effect of Traffic Pattern on Delivery Rate**

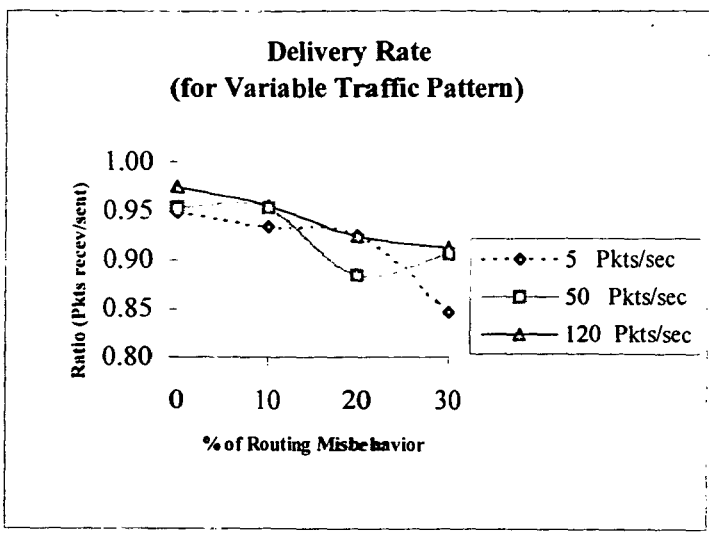

Fig 5.5 Overall network Delivery Rate as a function of percentage of Routing Misbehavior

The delivery rate varies from 0.84 to 0.91 even when 30% of the nodes misbehave.

- **Effect of Mobility rate on Delivery Rate**

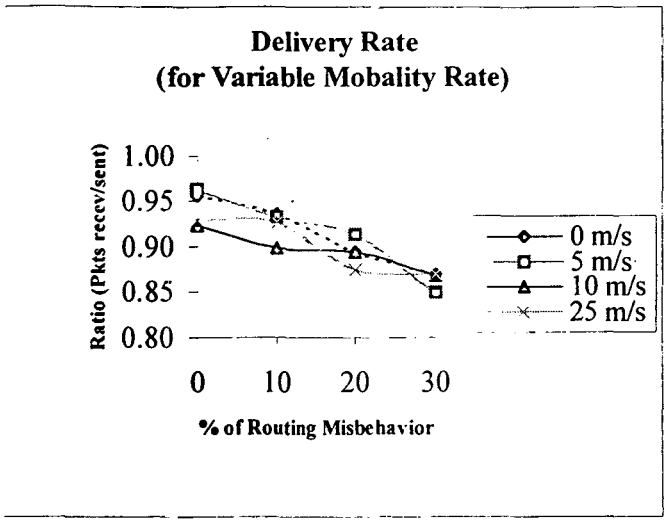

Fig 5.6 Overall network Delivery Rate as a function of percentage of Routing Misbehavior.

The delivery rate varies from 0.85 to 0.87 even when 30% of the nodes misbehave.

5.3.3 Overhead Ratio

The figures 5.7, figure 5.8, and figure 5.9 shows the Overhead ratio of the network for variable network size, traffic pattern and mobility rate respectively. Overhead ratio is analyzed with varying simulation variables one by one. For example, when we vary network size other variable are kept constant. This is repeated for other variable as well. When we changes Network Size from 25 to 50, 75 and 100, the other variables remain constant with Mobility Rate - 5m/s, Traffic Pattern - 5 pkts/sec, Bandwidth – 1Mbps

- **Effect of Network size on Overhead Ratio**

The Overhead ratio significantly increases from 0.129 to 0.757, as the number of nodes in a network increases from 25 to 100. This is because of the spread of control REQUEST packets in the network with the increase in the number of mobile nodes in the network. Also as the percentage of misbehaving node increases from 0% to 30%, there is a clear increase in Overhead Ratio. This is because of the new route discovery that takes place due to misbehaving nodes.

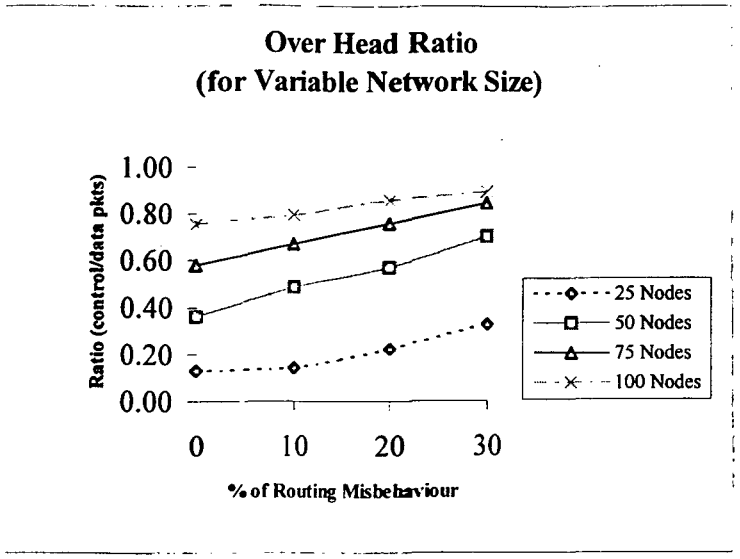

Fig 5.7 Overall network Overhead Ratio as a function of percentage of Routing Misbehavior.

- **Effect of Traffic Pattern on Overhead Ratio**

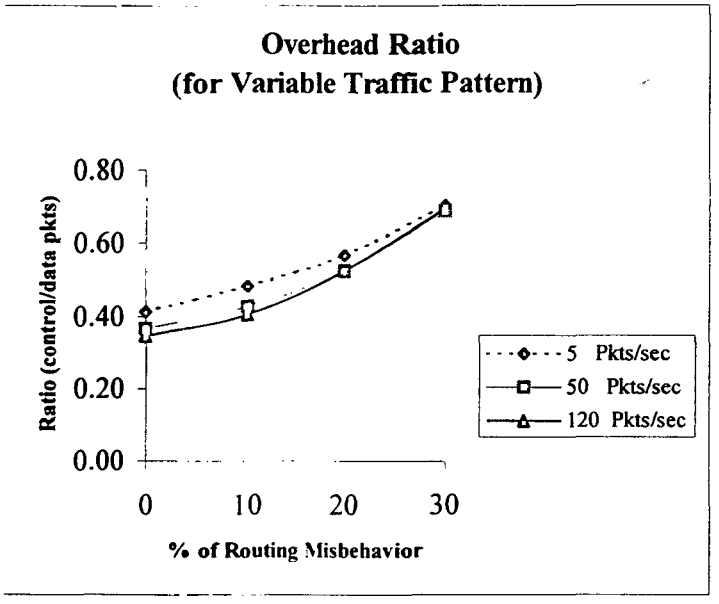

Fig 5.8 Overall network Overhead Ratio Rate as a function of percentage of Routing Misbehavior

Overhead ratio, increases significantly as the percentage of routing misbehavior increases for all three traffic patterns. This is due to increase in control REQUEST and REPLY packets for all the traffic patterns. Also, overhead ratio shows a minor decline as the traffic pattern changes from low to busty. This is because once the route has been discovered more number of packets can be transmitted before another route discovery has to be initiated.

- **Effect of Mobility rate on Overhead Ratio**

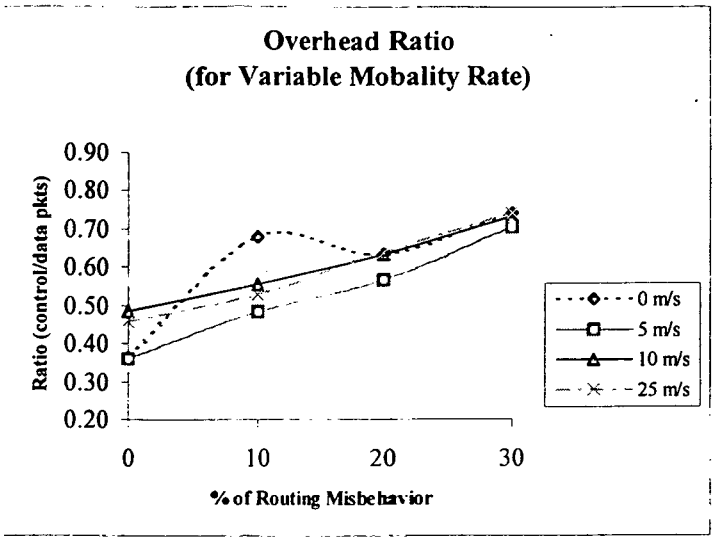

Fig 5.9 Overall network Overhead Ratio Rate as a function of percentage of Routing Misbehavior

As the percentage of routing misbehavior increases, the overall Overhead ratio of the network also increases. This is again due to increase in control REQUEST and REPLY packets for mobility rates. For a low mobility rate of 0 m/s and 5 m/s, the overhead ratio is almost same and much lower than high mobility nodes. This is due to lesser requirement for the nodes to initiate a new route discovery for low mobility rates.

5.3.4 Percentage Bandwidth Utilization

Percentage bandwidth utilization is analyzed with varying simulation variables one by one. For example, when we vary network size other variable are kept constant. This is repeated for other variable as well. When we changes Network Size from 25 to 50, 75 and 100, the other variables remain constant with Mobility Rate - 5m/s, Traffic Pattern - 5 pkts/sec, Bandwidth – 1Mbps

The figures 5.10, figure 5.11, and figure 5.12 shows the percentage Bandwidth utilization of the network for variable network size, traffic pattern and mobility rate respectively. It is observed that the overall bandwidth utilization in all the cases remains at a very low level from a maximum of 32% to a minimum of 12%, even when the routing misbehavior is 0%. This low % bandwidth utilization is due to delay in route discovery and queuing delay.

- **Effect of Network size on Bandwidth Utilization**

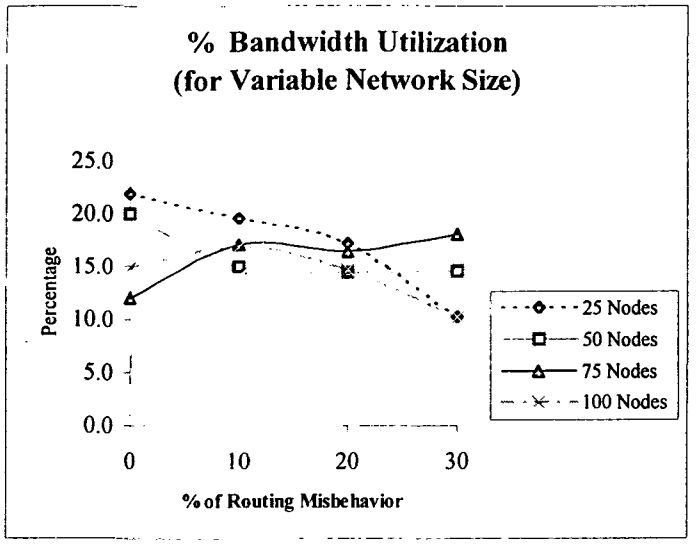

Fig 5.10 Overall network % Bandwidth utilization as a function of percentage of Routing Misbehavior

The percentage bandwidth utilization decreases as the size of the network increases. This is due to the increase in the number of uncontrolled route discovery packets, with the increase in the network size. These additional route discovery packets lead to more queuing delay, and increased number of packet drops due to time out (TOUT) of the packets. With the increase in % of misbehaving nodes, the graph does not show any specific trend for increasing network size.

- **Effect of Traffic Pattern on Bandwidth Utilization**

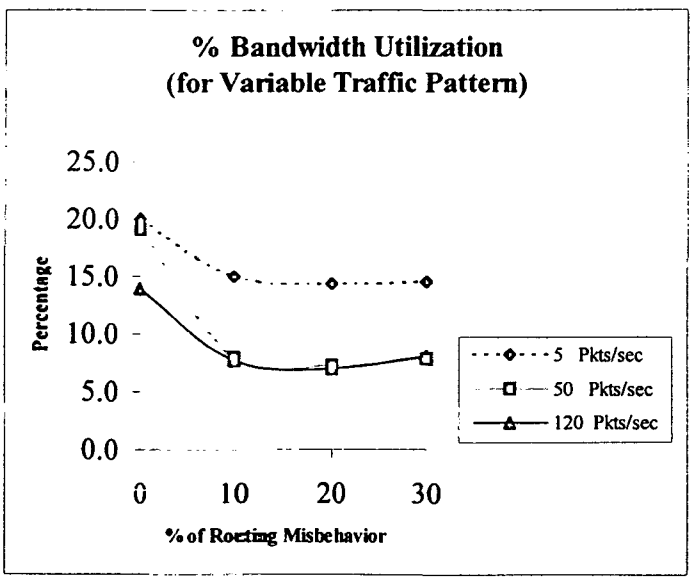

Fig 5.11 Overall network % Bandwidth utilization as a function of percentage of Routing Misbehavior.

Percentage Bandwidth utilization decreases as the traffic pattern changes from 5 packets/sec to 120 packets/sec. This is due to the increase in queuing delay caused by the increase in the traffic. Increased queuing delay leads to more number of packet drops and hence decreases Percentage bandwidth utilization with increasing traffic loads. With the increase in % of routing misbehavior to 10%, the heavy and busty traffic show a sharp decline. This is due to the additional requirement for the network to find new routes to the destination. Thus adding delay due to route discovery and hence decreases % bandwidth utilization.

- **Effect of Mobility rate on Bandwidth Utilization**

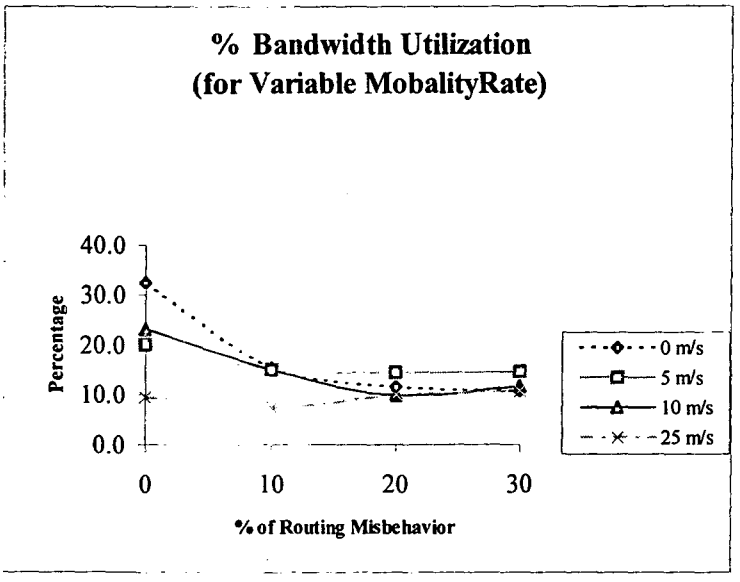

Fig 5.12 Overall network % Bandwidth utilization as a function of percentage of Routing Misbehavior

With the increase in the mobility rate from 0 m/s to 25 m/s, there is a significant decline in the bandwidth utilization. This decline is due to the increase in the number of route discovery, as the mobility rate increases. With the increase in the percentage of routing misbehavior in the network, there is a slight decrease in the % bandwidth utilization. This is due to the increase in alternate route discovery made by the source as the nodes starts misbehaving.

Conclusion and Future Work 6

This present work, showed the effect of variation in network size, traffic pattern and mobility rate on End-to-End delay, delivery rate, overhead ratio and percentage bandwidth utilization as the percentage of routing misbehavior increases in mobile ad hoc network for AODV routing algorithm. For simulation analysis trace file was analyzed using UNIX script and C code.

Simulation results showed that the End-to-End delay degrades with the increase in the control packets like route REQUEST and REPLY packets as it increases queuing delay for the data packets. Increase in the control packets can be either due to increase in network size or due to higher mobility rates. Control packets also increase as the percentage of routing misbehavior increases.

As the percentage of routing misbehavior increases from 0% to 30%, all the three factors i.e. Network size, Traffic Pattern and Mobility rate showed a decline in delivery rate. However the delivery rate remains at a very high value even when 30% of the nodes misbehave for the CBR traffic source in our simulation.

Overhead ratio increases with the increase in the control packets like route REQUEST and REPLY packets. Increase in the control packets can be either due to increase in network size or due to higher mobility rates. Control packets also increase as the percentage of nodes misbehaving increases.

Percentage bandwidth utilization for the entire network was found to be very low. This under utilization of bandwidth was a direct result of queuing delay and due to the number of uncontrolled packets generated by the AODV algorithm. Bandwidth utilization decreased with the increase in the number of routing control packets generated by the algorithm.

6.1 Future Work

Based on the study it was found that there are certain other aspects that have to be looked into to evaluate the performance of AODV routing algorithm. Following are the recommendations for the future research and enhancement to the existing work.

- Simulation results shows, very low (32% to 12%) percentage bandwidth utilization for AODV[2] routing algorithm. Some more simulation studies can be done to find out the contribution of each factor that leads to low % bandwidth utilization.
- We have used constant bit rate(CBR) traffic over TCP in all our simulations. Similar study can also be done with CBR traffic over UDP. Since UDP does not generate acknowledgments.
- Similarly simulations can be done using variable bit rate (VBR) traffic.
- We have used the priority queue in all our simulation, this has lead to significant performance degradation for a network with large number of mobile nodes. Studies can be made using different types of queues.
- Due to time constraints, misbehavior could not be implemented on the individual nodes, instead it was implemented by inducing the error model in this study. A similar study can be done after implementing the changes.
- Due to time constraints, the effect of misbehavior could not be analyzed with variable battery power and bandwidth. This present work can be extended to include these factors.

References

[1] Andrew S. Tanenbaum, " *Computer Networks* ", PHI publications, 3rd edition 2000.

[2] Charles E. Perkins, Elizabeth M. Royer, Samir R. Das, "*Ad Hoc on Demand Distance Vector (AODV) Routing*", Internet Draft, March 2001, www.ietf.org/internet-drafts/draft-ietf-manet-aodv-08.txt.

[3] Charles E. Perkins, P. Bhagwat,"*Highly dynamic DSDV for Mobile computer*", SIGCOM' 94 conference on Communication Architecture, protocols and Applications, www.cs.umd.edu/projects/meml/papers/Sigcomm94.ps

[4] Charles E. Perkins, "*Mobility support, Mobile IP and Wireless Channel Support for ns-2* ", www.svrloc.org/~charliep/mobins2

[5] David B Johnson, David A. Maltz, Yih-Chun Hu , Jorjeta G. Jetcheva "*The Dynamic Source Routing algorithm for Mobile Ad hoc networks*" , Internet Draft from IETF MANET Working Group, www.ietf.org/internet-drafts/draft-ietf-manet-dsr-05.txt

[6] Jae Chung, Mark Claypool, "*NS by Example*", WPI Computer Science, www.nile.wpi.edu/NS.

[7] J. Broch, D. Maltz, D. Johnson, Y. HU, J. Jetcheva, "*A Performance Comparison of Multi-hop Ad-hoc Network Routing Protocols*", ACM/IEEE International Conference on Mobile Computing and Networking, oct 1998.

[8] Jochen H Shiller ,"*Mobile Communications*", PHI publications.

[9] Kevin Fall, Kannan Varadhan, "*Ns Manual*", VINT project, May 2001, www.mach.cs.berkeley.edu/ns/ns-man.html.

[10] Mark Gries, " *Tutorial for Network Simulator NS-2"* , www.isi.edu/nsnam/ns/tutorial.

[11] Nitin H. Vaidya. Texas A & M University *"Mobile Ad-Hoc Networks: Routing, Mac and Transport issues"*, www.cs.tamu.edu/faculty/vaidya.

[12] Santhosh R. Thampuran,"*Routing Protocols for AD Hoc Networks of Mobile Nodes*", www.unix.ecs.umass.edu/~sthampur/Papers/AdhocRouting.pdf (Mobicom 98).

[13] Sergio Marti, T. J. Giuli, Kevin Lai, Mary Baker, "*Mitigating Routing Misbehavior in Mobile Ad Hoc Networks*", MOBICOM 2000.

[14] V Park, S Corson, "*Internet Draft for TORA*", IETF MANET working group, July, 2001, www.ietf.org/internet-drafts/draft-ietf-manet-tora-spec-04.txt