Global Positioning System (GPS) based Protocols for Routing in Mobile Adhoc Networks

THESIS

Submitted in partial fulfilment

of the requirements for the degree of

DOCTOR OF PHILOSOPHY

By

K.R.ANUPAMA

Under the supervision of

Prof. S.BALASUBRAMANIAN

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI (RAJASTHAN) INDIA

2004

Acknowledgements

I wish to convey my sincere gratitude and heartfelt thankfulness to my guide **Prof.S.Balasubramanian**, Group Leader, EEE BITS, Pilani for introducing me to the area of mobile networking, providing me all the guidance, helping me understand the concepts, and for his continuous encouragement and moral support.

I thank **Prof.S.Venkateswaran**, Vice-Chancellor, BITS, for providing me the necessary infrastructure and facilities.

I wish to express my deep gratitude to **Prof. L.K. Maheswari**, Director, BITS, Pilani and **Prof. K.E. Raman**, Deputy Director, BITS, Pilani for their constant encouragement and moral support.

I am grateful to **Dr. Sudhir Dixit**, Nokia Research Centre, Boston for offering me the Nokia Research Fellowship, and for his invaluable, thought provoking suggestions from time to time.

I am sincerely indebted to **Prof.G.Raghurama**, Dean (FD-II) and **Prof.S.Gurunarayanan**, Assistant Dean (ESD) for their keen interest and constructive suggestions.

I thank **Prof.S.Ravi Praksash**, Dean (R&C) and **Prof. A.K.Sarkar,** Dean (ID) for providing necessary administrative help.

I thank all the **staff members of EEE, Instrumentation and Computer Science** for their constant support in completion of my work.

I am thankful to **Mr. S.D.Pohekar**, Lecturer and PhD in-charge (R&C), for making necessary arrangements for seminars and providing guidelines for proper organization of the work.

I express my sincere thanks to all those who have directly or indirectly contributed to the completion of my work.

K.R.Anupama

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI RAJASTHAN

CERTIFICATE

This is to certify that the thesis entitled **Global Positioning System (GPS) based Protocols for Routing in Mobile Adhoc Networks** and submitted by **Ms. K.R.Anupama, ID.No. 2000PHXF016,** for award of Ph.D. Degree of the institute, embodies original work done by her under my supervision.

S Balasubramanian

Signature of the Supervisor

Prof. S.Balasubramanian

Name

Associate Professor

Designation

Date: October 27, 2004

ABSTRACT

Mobile Wireless communication is becoming increasingly popular due to the recent advances in wireless devices and applications. A desirable capability of portable devices is the ability for communication between them. Collectively, these devices can form an adhoc network. An adhoc network is a group of mobile nodes with no fixed infrastructure; nodes communicate directly with one another over wireless channels. Because the transmission range of these nodes is limited, a routing protocol is needed to enable communication between them. However, because of the portable nature of these devices and the wireless transmission medium, adhoc networks have many characteristics that render routing protocols designed for wired networks inapplicable.

This thesis presents the GPS-based Predictive Energy Aware Routing (GPEAR) Protocol. GPEAR is a reactive protocol in that it discovers route only when a source node needs them. It provides unicast communication capability and is able to maintain routes even when the topology of the network is dynamic. GPEAR in addition uses location information to predict route breakages in advance and to control the energy at which packet transmission is done. GPEAR is well suited for mobile wireless networks in that it has low processing and memory overhead, and low network and node energy utilization.

GPEAR has been analyzed in detail, using simulation studies for various performance metrics under varied mobility and traffic. Many factors contribute to the overall performance of the protocol. These factors include caching structures, routing criterion, and node density. Studies of these factors are presented to determine their effect in mobile adhoc networks.

To improve the scaling potential of GPEAR and adhoc networks in general, an inter-zone routing protocol (GPS-based Inter-Zone Routing: GIZR) has been proposed as part of the thesis. The protocol for inter-zone routing requires geographical routing, so Geocast and Multicast Routing protocols also have been proposed. GIZR makes uses of underlying unicast, multicast and geocast routing mechanisms for improving the scalability of adhoc networks.

Table of Contents

List of Symbols and Abbreviations

General

ACK	Acknowledgement
AP	Access Points
ARP	Address Resolution Protocol
BS	Base Station
CTS	Clear To Send
DBF	Distributed Bellman Ford Algorithm
DCF	Distribution Co-ordination Function
FIFO	First In First Out
IETF	Internet Engineering Task Force
IFQ	Interface Queue
LL	Link Layer
LS	Link State
MAC	Medium Access Control
MACA	Multiple Access with Collision Avoidance
MACW	Medium Access Control Wireless
MANET	Mobile Adhoc Network
NETIF	Network interface
NS	Network Simulator
PDA	Personal Digital Assistant
PERL	Practical Extraction and Report Language
RTS	Request To Send
SNK	Sink

SRC	Source
TCL	Tool Command Language
OTCL	Object-oriented Tool Command Language
TCLCL	Tool Command Language Class

Adhoc Networking Protocols

ABR	Associativity Based Routing
AODV	Adhoc On Demand Distance Vector
CBR	Cluster Based Routing
CGSR	Clusterhead Gateway Switch Routing
DLAR	Dynamic Load Aware Routing
DREAM	Distance Routing Effect Algorithm for Mobility
DSDV	Dynamic Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
FSR	Fisheye State Routing
GGPEAR	Geocast GPS-based Predictive Energy Aware Routing
GIZR	GPS-based Inter-Zone Routing
GPEAR	GPS-based Predictive Energy Aware Routing
GSR	Global State Routing
HSR	Hierarchical State Routing
LAR	Location Aware Routing
MAODV	Multicast Adhoc On-Demand Distance Vector
MGPEAR	Multicast GPS-based Predictive Energy Aware Routing
PARO	Power Aware Routing

SSA	Signal Stability Analysis
TORA	Temporally Ordered Routing Algorithm
WRP	Wireless Routing Protocol

MANET messages and Terminologies

BG	Border Group
DRET	Dynamic RET
GACK	Group Acknowledge
GERR	Group Error
GLEAVE	Group Leave
GJOIN	Group Join
GREP	Group Reply
GUPDATE	Group Update
IRERR	Inter-Zone Route Error
IRREQ	Inter-Zone Route Request
IRREP	Inter-Zone Route Reply
IUPDATE	Inter-Zone Update
LET	Link Expiry Time
LRO	Least Recently Overheard
LRU	Least Recently Used
MC	Multicast Count
PDR	Packet Delivery Ratio
RET	Route Expiry Time
RREQ	Route Requests
RREP	Route Reply

RUPDATE	Route Update
SGJOIN	Secondary Group Join
SR	Source Routing
SRET	Static RET
SRREQ	Secondary Route Request
TTL	Time to Live

Constants

G_ROUNDTRIP	Time for which a node waits before replying to a SGJOIN
Lopt	Minimum number of nodes by which two routes can be linked to be termed as disjoint.
L_t	Value of LET calculated by a node
MAX_RET	Maximum time for which is network is expected to be active
MAX_TTL	Maximum Path length between source and destination nodes that is supported by the protocol
P_size	Size of Primary Cache
REQ_Wait	Time for which a multicast node waits before acknowledging a GREP
Route_Length	Length of source route between source and destination node
S_size	Size of Secondary Cache
T_opt	Minimum rime for which a link should be active to prevent reconstruction of route by using LET prediction
TTL_threshold	Used in ring search; TTL value up to which the RREQ broadcast range will be incremented by 2. When TTL_threshold is reached the RREQ broadcast range is sent to MAX_TTL.

Flags used by the various Protocols

GPEAR

G	Gratuitous RREP
R	RREP generated through a process of local repair

Multicast

M	Indicates that the node issuing the control message supports Multicasting

GIZR

R	Used in case of IRREQ by a leaf node to relinquish the responsibility of propagating the IRREQ further, to some other node present in the same geocast group, but in the next zone.
S	Indication that the RREP is unsolicited and obtained through a process of route repair.
Z	Indicates that the Route passes through multiple Zones.

List of Figures

Effect of Routing Criteria on Network Performance

GPEAR Analysis

Optimum Node Density Analysis

Scalability Analysis

Chapter 1- Introduction

1.1 Motivation

In recent years, mobile computing has enjoyed a tremendous rise in popularity. The continued miniaturisation of mobile computing devices and the extraordinary rise of processing power available in mobile laptop computers combine to put more and better computer based applications into the hands of a growing segment of population. Advances in battery technologies have also allowed these devices to be used for increasingly longer periods of time away from electrical sources. At the same time, the markets for wireless telephones and communication devices are experiencing a rapid growth. Wireless devices can communicate with each other using either infrared ports or radio modems. The applications of infrared are limited due to line-of sight requirement and low data rate characteristics of these waves. Radio Modems, on the other hand, have the capability of higher transmission ranges and data rates, although they may still suffer from multipath interference and fading. Current Wireless Modems offer a wide range of transmission power and connectivity levels. Radio Modems can transmit at rates as high as 11Mbps for ranges up to 600m, depending on the data rate and surrounding environmental conditions.

Mobile telephony has gained huge popularity. A similar transformation awaits mobile computer users. Much of the context of the transformation has to do with keeping in touch with the Internet. One expects to have "the network " at one's disposal for innumerable little conveniences.

Mobile networks have many unique characteristics that make traditional routing protocols inapplicable. The topology of a mobile network is often highly dynamic due to the mobile nature of the nodes. Whereas a broken link in a wired network is considered an exception, links within wireless networks tend to break frequently as nodes move in and out of transmission range of one another. Furthermore, atmospheric effects and physical objects

also play a role in limiting the communication between wireless nodes. Additional characteristics of mobile wireless networks include limited power and bandwidth, and high error rates due to the wireless transmission.

Routing protocols designed for wired networks generally do not perform well over wireless channels. Traditional routing protocols designed for wired networks are either distance- vector [1] or link-state protocols [2]. These protocols maintain routes for each and every node in the network through the periodic exchange of routing table messages. Early routing protocols for mobile networks attempted to adapt these basic protocols for mobile scenarios. In a mobile scenario constant updates are required based on the node movement. This may result in significant control overhead and bandwidth consumption in a mobile network.

For similar reasons, multicast protocols designed for wired networks are not well suited for operation in mobile networks. For instance core-based trees [3] have a drawback that all branches emanate from a single node, the core. In a mobile network, it may frequently happen that nodes are shutdown temporarily or transmission is temporarily impaired. Algorithms with a single point of failure are likely to suffer from frequent temporary disconnections witnessed by mobile networks. Protocols designed for dense mode multicast are likely to have too much overhead for use in mobile networks, due to frequent node movement. For instance in PIM-Dense Mode [4], prune packets are sent whenever a multicast packet arrives via a "wrong" tunnel. Multicast packets are only accepted and forwarded when they arrive over the "right" tunnel. In a mobile environment, the optimal path to some destination may change from moment to moment, and there is no way of knowing the "right" tunnel at any given time. Additionally, due to the inherent difficulty of routing in mobile networks, multicast nodes should accept data packets destined for them, from whichever direction they may arrive.

Mobile wireless networks have numerous advantages over their traditional wired counterparts. Wireless networks can be established in areas of the world without pre-existing wired infrastructure. Installing cellular infrastructure is much cheaper than burying cables, making wireless networks an attractive option in developing nations. A mobile network allows a user flexibility of movement. A user can walk up and down a hallway in an office building and maintain connectivity without having to worry about finding an Ethernet connection once the destination is reached. Finally, wireless networks can result in the elimination of wire clutter in office spaces by reducing the need for Ethernet cables.

1.2 Infrastructured Wireless Networks

There are two distinct types of wireless networks: infrastructured and infrastructureless. Infrastructured wireless networks have a wired backbone of stationary nodes that are connected to the rest of the network or the Internet. These stationary nodes are generally called either Base Stations (BS) or Access Points (AP). Mobile nodes communicate to these access points and generally do not establish point-to-point connections with other mobile nodes. Each AP has a coverage area, or cell, in which it is able to send signals to, and receive signals from, other nodes. This coverage area is dependent on the AP's transmission radius. Nodes within the cell of an AP are able to communicate directly with that AP. Because the mobile nodes are likely to be moving, it is possible that they will not always stay within the coverage area of a single AP. As the mobile nodes moves from the coverage area of one AP to that of another, a "handoff" occurs, where the node ceases to have communication with the old AP and begins communicating with the new AP. Figure 1.1 illustrates the hand-off of a mobile node from one AP to another. The hand-off should be completely seamless so that the user is not aware of the transition. As long as mobile node stays within its home network, it should be able to access the Internet, receive email, etc., as if it were a wired node on the network.

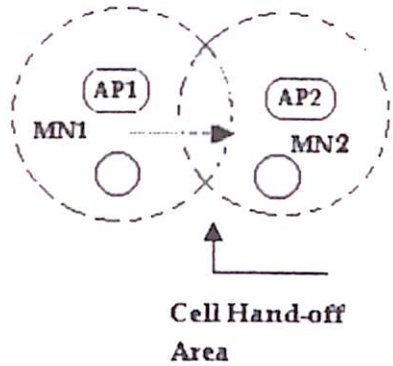


Fig1.1 Cell Hand-off Area

However, once the node leaves its local network, routing difficulties commence because the subnet IP address of the node and the network to which it has moved are likely to differ. Hence, the node can no longer receive packets addressed to it. To solve this problem Mobile IP [5,6,7,8] has been developed. Mobile IP is an extension of IP that enables the mobile node to utilise two IP addresses. The first is for its identification (Home Address), and second for its routing (Care-of Address). These addresses allow nodes to send and receive data in networks other than its home network. Such non-local networks are referred to as foreign networks.

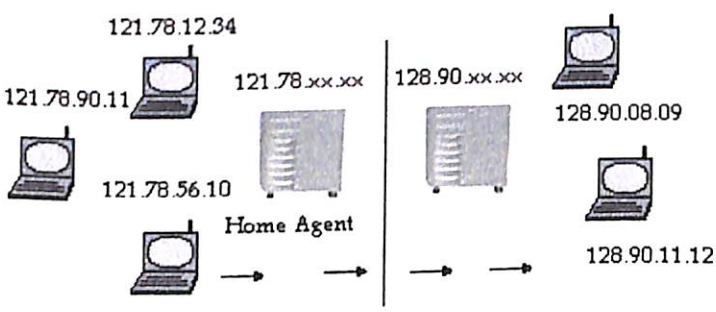


Fig 1.2a Mobile Node Movement

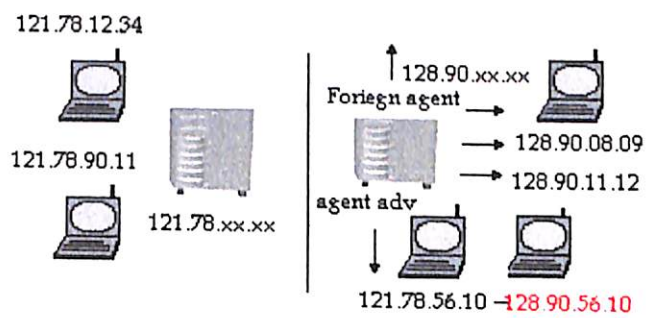


Fig1.2b Agent Advertisements

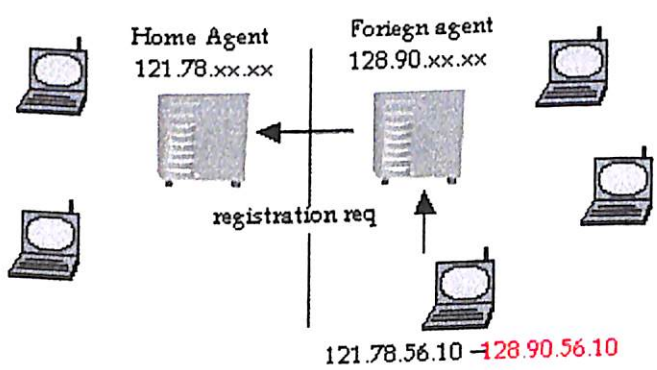


Fig 1.2c Registration

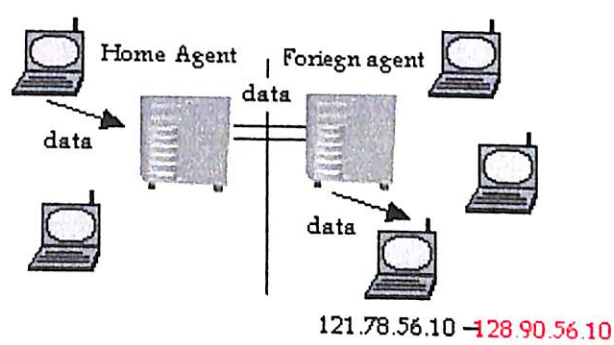


Fig1.2d Tunnelling

each node must serve as a router for the other nodes in the network so that data packets can be forwarded to their destinations. Figure 1.3 illustrates an example of an adhoc network [9]. Because there is no wired backbone along which routing can occur, an adhoc network needs a routing protocol that can establish and maintain routes, even in networks with dynamic topologies.

As previously described, mobile nodes have many unique characteristics that make traditional routing protocols inapplicable. Because of the limitations of wireless nodes, an adhoc routing protocol should be able to provide routes with a minimum control overhead, and should require as little processing time as possible. Furthermore, due to the characteristics of wireless transmissions, the range of the nodes is often limited.

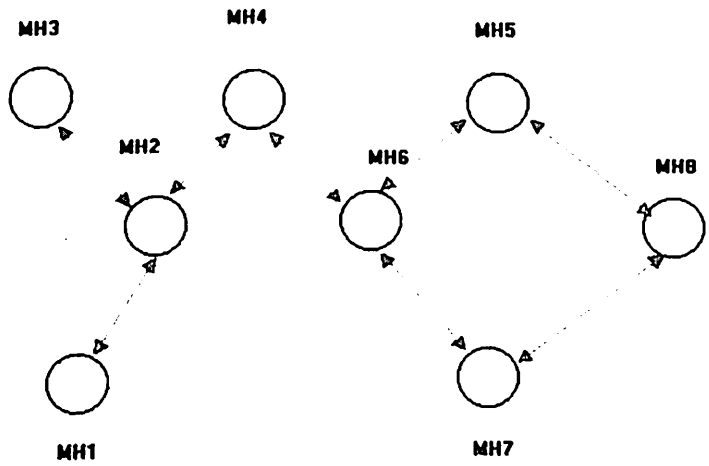


Fig 1.3 Adhoc Network

It is usually the case that paths between sources and destinations require multiple hops. Hence a routing protocol must be able to find multi-hop paths between nodes. The protocol should not only be self-starting, but it should also be loop free at all times, because even a temporary routing loop wastes, already scarce bandwidth allocated.

Mobile Adhoc Networks (MANETs) generally have reduced administrative cost as compared to wired networks. These networks are self-configuring, and so they are able to maintain network connections and routing information without the need for explicit route set-up by a system administrator. Typical examples of adhoc networks occur in emergency, search and rescue operations, and conference scenarios where attendees want to easily share

information. Adhoc networks are also suitable for networking in locations without existing wired infrastructure, such as data collection in open fields and sensor networks.

1.4 Summary of Thesis

This thesis addresses the problem of providing communication between nodes in an adhoc mobile wireless network. This problem is addressed at the network layer. Methods for discovery and maintenance of routes between nodes wishing to communicate are presented.

A number of protocols have been suggested for routing in Mobile Adhoc networks. Each of these protocols has had a varying degree of success. Protocols suggested so far, have proved to be scenario and mobility dependent. When a route breaks, routes have to be re-established and the route set-up time varies with the scenarios; meanwhile a number of packets are lost. The protocols presented in this thesis use location information for advance prediction of route breakages and repair.

The first protocol, GPS-based Predictive Energy Aware Routing Protocol (GPEAR) is for unicast routing. Two protocols have been developed for multicast routing MANETs. The first multicast protocol presented in this thesis can be used with any unicast protocol. The second protocol: Multicast GPS-based Energy Aware Routing Protocol (MGPEAR) suggested is specifically for networks, which use GPEAR as their unicast protocol. MGPEAR has also been modified to support geocast functions; the resultant protocol is GGPEAR (Geocast-GPS based Energy Aware Routing).

The scalability of the protocols developed has also been analysed. GPS-based Inter-Zone Routing Protocol has been developed to resolve scalability issues. The network area is divided into smaller areas called Zones. While GPEAR is the protocol used for routing within these zones; Inter-Zone routing is done using a combination of Multicast, Unicast and Geocast Messages, instead of introducing new types of routing messages.

The protocols are designed to function on Layer 3 of the OSI/TCP-IP stack. The protocols are independent of the underlying and overlying areas, though optimal performance has been obtained with a bi-directional MAC protocol such as 802.11. The route changes that occur due to mobility are invisible to the stack areas above Layer 3.

.

Chapter 2 - Adhoc On Demand Distance Vector Vs Dynamic Source Routing

2.1 Proactive and Reactive Routing Protocols

Several adaptive routing protocols for adhoc networks have been proposed to solve the multi-hop routing problem, each based on different assumptions and concepts. In general, these protocols can be classified either as *proactive* or *reactive*.

Proactive protocols attempt to continuously evaluate the routes within the network, so that when a packet needs to be forwarded, the route is already known and can be immediately used. Proactive protocols are also termed as Table-driven protocols. Examples of proactive protocol are:

- Dynamic Destination Sequenced Distance Vector Routing Protocol (DSDV) [10]
- Wireless Routing Protocol (WRP) [11]
- Global State Routing (GSR) [12]
- Fisheye State Routing (FSR) [13]
- Hierarchical State Routing (HSR) [14]
- Clusterhead Gateway Switch Routing Protocol (CGSR) [15].

Reactive protocols, on the other hand, invoke the route determination procedures on demand only. Thus, when a route is needed, some sort of global search procedure is employed. Hence reactive protocols are also termed as on-demand protocols.

Examples of reactive protocols are:

- Dynamic Source Routing (DSR)
- Adhoc On Demand Distance Vector (AODV)
- Temporally Ordered Routing Algorithm (TORA) [16]
- Associativity Based Routing (ABR) [17]

- Signal Stability Routing (SSA) [18]
- Location Aware Routing (LAR) [19]
- Distance Routing Effect Algorithm for Mobility (DREAM) [20]
- Power Aware Routing (PARO) [21].

The advantage of the proactive schemes is that, once a route is requested, there is hardly any delay until a route is determined. In reactive protocols, because route information may not be available at the time when a routing request is received, the delay to determine a route can be quite significant. Because of this long delay, pure reactive routing protocols may not be applicable to real-time communication. However, pure proactive schemes are likewise not appropriate for the adhoc environment, as they continuously use large portion of the network capacity to keep the routing information current. Since in an adhoc network nodes move quite fast, and as the changes may be more frequent than the routing requests, most of this routing information may never be used! This results in an excessive waste of the network capacity. What is needed is a protocol that, on the one hand, initiates the route-determination procedure on-demand, but on the other hand contains the cost of the global search.

2.2 Dynamic Source Routing (DSR)

The Dynamic Source Routing protocol (DSR) [22] is a simple and efficient routing protocol designed specifically for use in multi-hop wireless adhoc networks of mobile nodes. Using DSR, the network is completely self-organizing and self-configuring, requiring no existing network infrastructure or administration. Network nodes cooperate to forward packets for each other to allow communication over multiple "hops" between nodes not directly within wireless transmission range of one another. As nodes in the network move about or join or leave the network, and as wireless propagation conditions such as sources of interference change, all routing is automatically determined and maintained by the DSR routing protocol. Since the number or sequence of intermediate hops needed to reach any

destination may change at any time, the resulting network topology may be quite rich and rapidly changing.

The DSR protocol allows nodes to dynamically discover a source route across multiple network hops to any destination in the adhoc network. Each data packet sent, then carries in its header the complete, ordered list of nodes through which the packet will pass, allowing packet routing to be trivially loop-free and avoiding the need for up-to-date routing information in the intermediate nodes through which the packet is forwarded. By including this source route in the header of each data packet, other nodes forwarding or overhearing any of these packets may also easily cache this routing information for future use. The DSR protocol provides highly reactive service to help ensure successful delivery of data packets in spite of node movement or other changes in network conditions.

The DSR protocol is composed of two mechanisms that work together to allow the discovery and maintenance of source routes in the adhoc network:

- **Route Discovery** is the mechanism by which a node (S) wishing to send a packet to a destination node (D) obtains a source route to D. Route Discovery is used only when S attempts to send a packet to D and does not already know a route to D.
- **Route Maintenance** is the mechanism by which node S is able to detect, while using a source route to D, if the network topology has changed such that it can no longer use its route to D because a link along the route no longer works. When the Route Maintenance mechanism indicates a source route is broken, S can attempt to use any other route it happens to know to D, or can invoke Route Discovery again to find a new route for subsequent packets to D. Route Maintenance for this route is used only when S is actually sending packets to D.

In DSR, Route Discovery and Route Maintenance each operate entirely "on demand". In particular, unlike other protocols, DSR requires no periodic packets of any kind at any level within the network. For example, DSR does not use any periodic routing

advertisement, link status sensing, or neighbor detection packets, and does not rely on these functions from any underlying protocols in the network. This entirely on-demand behaviour and lack of periodic activity allows the number of overhead packets used by DSR to scale all the way down to zero, when all nodes are approximately stationary with respect to each other and all routes needed for current communication have already been discovered. As nodes begin to move more, or as communication patterns change, the routing packet overhead of DSR automatically scales to only that needed to track the routes currently in use. Network topology changes not affecting routes currently in use are ignored and do not cause reaction from the protocol. In response to a single Route Discovery (as well as through routing information from other packets overheard), a node may learn and cache multiple routes to any destination. This allows the reaction to routing changes to be much more rapid, since a node with multiple routes to a destination can try another cached route if the one it has been using should fail. This caching of multiple routes also avoids the overhead of needing to perform a new Route Discovery each time a route in use, breaks.

The operation of both Route Discovery and Route Maintenance in DSR are designed to allow unidirectional links and asymmetric routes to be easily supported. In particular, in wireless networks, it is possible that a link between two nodes may not work equally well in both directions, due to differing antenna patterns or propagation condition changes or sources of interference. DSR allows such unidirectional links to be used when necessary, improving overall performance and network connectivity in the system.

The key feature of DSR is the use of **source routing**. That is, the sender knows the complete hop-by-hop route to the destination. These routes are stored in a **route cache**. The data packets carry the source route in the packet header. When a node in the adhoc network attempts to send a data packet to a destination for which it does not already know the route, it uses a *route discovery* process to dynamically determine such a route. Route discovery works by flooding the network with route request (**RREQ**) packets. Each

node receiving a RREQ rebroadcasts it, unless it is the destination or it has a route to the destination in its route cache. Such a node replies to the RREQ with a route reply (**RREP**) packet that is routed back to the original source. RREQ and RREP packets are also source routed. The RREQ builds up the path traversed so far. The RREP routes itself back to the source by traversing this path backwards.

The route carried back by the RREP packet is cached at the source for future use. If any link on a source route is broken, the source node is notified using a route error (**RERR**) packet. The source removes any route using this link from its cache. A new route discovery process must be initiated by the source, if this route is still needed. DSR makes very aggressive use of source routing and route caching. No special mechanism to detect routing loops is needed. Also, any intermediate node caches the source route in a packet it forwards for possible future use. Several additional optimisations have been proposed and have been evaluated to be very effective by the authors of the protocol [23] as described in the following.

(i) **Salvaging:** An intermediate node can use an alternate route from its own cache, when a data packet meets a failed link on its source route.

(ii) **Gratuitous route repair:** A source node receiving a RERR packet piggybacks the RERR in the following RREQ. This helps clean up the caches of other nodes in the network that may have the failed link in one of the cached source routes.

(iii) **Promiscuous listening:** When a node overhears a packet not addressed to itself, it checks if the packet could be routed via itself to gain a shorter route. If so, the node sends a **Gratuitous RREP** to the source of the route with this new, better route. Apart from this, promiscuous listening helps a node to learn different routes without directly participating in the routing process.

2.3 Adhoc On-Demand Distance Vector Routing (AODV)

The Adhoc On-Demand Distance Vector (AODV) [24] algorithm enables dynamic, self-starting, multi-hop routing between participating mobile nodes wishing to establish and maintain an adhoc 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 adhoc 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.

Route Requests (RREQs), Route Replies (RREPs), and Route Errors (RERRs) are the message types defined by AODV. These message types are handled by UDP, and normal IP header processing applies. So, for instance, the requesting node is expected to use its IP address as the source IP address for the messages. The range of dissemination of broadcast RREQs can be indicated by the Time To Live (TTL) in the IP header. Fragmentation is typically not employed.

As long as the endpoints of a communication connection have valid routes to each other, AODV does not play any role. When a route to a new destination is needed, the node uses a broadcast RREQ to find a route to the destination. A route can be determined when the RREQ reaches either the destination itself, or an intermediate node with a 'fresh enough'

route to the destination. A 'fresh enough' route is an unexpired route entry for the destination whose associated sequence number is at least as great as that contained in the RREQ message.

The route is made available by unicasting a RREP back to the source of the RREQ. Since each node receiving the request caches a route back to the source of the request, the RREP can be unicast back from the destination to the source, or from any intermediate node that is able to satisfy the request back to the source. A RREQ can be conditioned by requirements on the path to the destination, namely bandwidth or delay bounds. Nodes monitor the link status of next hops in active routes. When a link break in an active route, is detected, a RERR message is used to notify the other nodes regarding link loss that has occurred. The RERR message indicates the destinations that are now not reachable due to loss of the link.

AODV is a routing protocol, and hence it also deals with route table management. Route table information must be kept even for ephemeral routes, such as those that are created to temporarily store reverse paths towards nodes originating RREQs.

AODV shares DSR's on-demand characteristics, in that, it also discovers routes on an "as needed" basis via a similar route discovery process. However, AODV adopts a very different mechanism to maintain routing information. It uses traditional routing tables, normally one entry per destination. This is a departure from DSR, which can maintain multiple route cache entries per destination. Without source routing, AODV relies on routing table entries to propagate a RREP back to the source and, subsequently, to route data packets to the destination. AODV uses sequence numbers maintained at each destination to determine freshness of routing information and to prevent routing loops. All routing packets carry these sequence numbers. An important feature of AODV is maintenance of timer-based states in each node, regarding utilization of individual routing table entries. A routing table entry is "expired" if not used recently. A set of predecessor nodes is maintained per routing

table entry, which denotes the set of neighboring nodes that use this entry to route data packets. These nodes are notified with RERR packets when the next hop link breaks. Each predecessor node, in turn, forwards the RERR to its own set of predecessors, thus effectively erasing all routes using the broken link. The recent specification of AODV [25] includes an optimisation technique to control the RREQ flood in the route discovery process. It uses an **expanding ring** search initially to discover routes to an unknown destination. In the expanding ring search, increasingly larger neighborhoods are searched to find the destination. The TTL field in the IP header of the RREQ packets controls the search. If the route to a previously known destination is needed, the prior hop-wise distance is used to optimise the time for route search.

2.4 Comparison of DSR and AODV

The two on-demand protocols, DSR and AODV share certain salient characteristics. In particular, they both discover routes only in the presence of data packets in the need for a route to a destination. Route discovery in either protocol is based on query and reply cycles and route information is stored in all intermediate nodes on the route in the form of route table entries (AODV) or in route caches (DSR). However, there are several important differences in the dynamics of these two protocols, which may give rise to significant performance differentials.

First, by virtue of source routing, DSR has access to a significantly greater amount of routing information than AODV. For example, in DSR, using a single request-reply cycle, the source can learn routes to each intermediate node on the route in addition to the intended destination. Each intermediate node can also learn routes to every other node on the route. Promiscuous listening on data packet transmissions can also give DSR access to a significant amount of routing information. In particular, it can learn routes to every node on the source route of that data packet. In the absence of source routing and promiscuous listening, AODV

can gather only a very limited amount of routing information. In particular, route learning is limited only to the source of any routing packets being forwarded. This usually causes AODV to rely on a route discovery flood more often, which may carry a significant network overhead.

Second, to make use of route caching aggressively, DSR replies to *all* requests reaching a destination from a single request cycle. Thus the source learns many alternate routes to the destination, which will be useful in the case the primary (shortest) route fails. Having access to many alternate routes saves route discovery floods, which is often a performance bottleneck. However, there may be a possibility of a route reply flood. In AODV, on the other hand, the destination replies only once to the request arriving first and ignores the rest. The routing table maintains at most one entry per destination.

Third, the current specification of DSR does not contain any explicit mechanism to expire stale routes in the cache, or prefer "fresher" routes when faced with multiple choices. As noted in, stale routes, if used, may start polluting other caches. Some stale entries are indeed deleted by route error packets. But because of promiscuous listening and node mobility, it is possible that more caches are polluted by stale entries than are removed by error packets. In contrast, AODV has a much more conservative approach than DSR. When faced with two choices for routes, the fresher route (based on destination sequence numbers) is always chosen. Also, if a routing table entry is not used recently, this entry is expired. However the latter technique is not problem-free. It is possible to expire valid routes this way, if they remain unused beyond a specified expiry time. As sending rates for sources, as well as, node mobility may differ widely and can change dynamically, the estimation of a suitable expiry time is quite difficult.

Fourth, the route deletion activity using RERR is also conservative in AODV. By the use of a predecessor list, the error packets reach *all* nodes using a failed link on its route to any destination. In DSR, however, a route error simply backtracks the data packet that meets

a failed link. Nodes that are not on the upstream route of this data packet but using the failed link are not notified promptly.

Even though DSR and AODV share the on-demand behaviour, much of their routing mechanisms are different. In particular, DSR uses source routing and route caches and does not depend on any periodic or timer-based activities. DSR exploits caching aggressively and maintains multiple routes per destination. AODV, on the other hand, uses routing tables, one route per destination, and destination sequence numbers (a mechanism to prevent loops and to determine freshness of routes).

It can be said that DSR with its use of source routing and caching follows an aggressive approach, and AODV with its routing table and sequence number driven approach follows a more conservative approach.

DSR outperforms AODV in less "stressful" situations, i.e., smaller number of nodes and lower load and/or mobility. AODV, however, outperforms DSR in more stressful situations, with widening performance gaps with increasing stress (e.g., more load, higher mobility). DSR, however, consistently generates less routing load than AODV.

DSR may give a very poor route set-up time delay and throughput performance due to aggressive use of caching and lack of any mechanism to expire stale routes or to determine the freshness of routes when multiple choices are available. Aggressive caching, however, may help DSR at low loads and also keeps its routing load down. Mechanisms to expire routes and/or determine freshness of routes will benefit DSR's performance significantly. On the other hand, AODV's routing loads can be reduced considerably by source routing the request and reply packets in the route discovery process. Since AODV keeps track of actively used routes, multiple actively used destinations also can be searched using a single route discovery flood to control routing load. In general, both protocols could benefit from the following:

(i) Use of congestion-related metrics (such as queue lengths) to evaluate routes instead of emphasizing the hop-wise shortest routes

(ii) Removal of "aged" packets from the network. The aged packets are not critical for the upper layer. They will probably be retransmitted. But they contribute to the load in the routing layer.

(iii) Introduction of location information

(iv) Dynamic metrics other than shortest hop.

Chapter 3 - Caching Policy

3.1 Introduction

Caching is an important part of any on-demand routing protocol for wireless adhoc networks. The route caches are distributed across different nodes over the entire network. Leveraging caches in the mobile adhoc networks brings up the challenge of keeping the distributed caches up-to-date even with frequent route changes. Utilizing cached information without robust mechanisms to keep it up-to-date can actually degrade performance and thus making caches counter-productive.

The cache in GPEAR is modelled on the cache structure of DSR [23]. DSR has an aggressive caching policy. However, the current specifications of DSR lack a mechanism to determine the freshness among routes in the cache, or even to purge all stale routes from the cache effectively. DSR also has a policy of caching any route whether obtained through the formal process of *route discovery* or *overheard*. This policy gives rise to huge and unmanageable route caches, draining the limited system memory. AODV on the other hand, is conservative in its approach and has a policy of caching only the best route. Any time a route crashes, route discovery has to be carried all over again.

3.2 Cache Structure

Each node in an adhoc network that employs GPEAR as the routing protocol, maintains two caches, a *Primary cache* and a *Secondary cache*. All routes obtained using route discovery mechanism are stored in the primary cache. Routes overheard from data and control packets are stored in the secondary cache (Promiscuous Listening) are stored in the secondary cache. A node always searches its primary cache for a route before it turns to the secondary cache. Before using a route in the secondary cache, it is transferred to the primary cache, thereby authenticating the route.

Each path obtained by the process of Route discovery is stored in the cache. By caching each of these paths to the destination separately a path cache is formed. A path cache is very simple to implement and easily guarantees that all routes are loop-free. To find a route in a path cache, the sending node can simply search its cache for any path (or prefix of a path) that leads to the intended destination node. An example of path cache is shown in figure 3.1

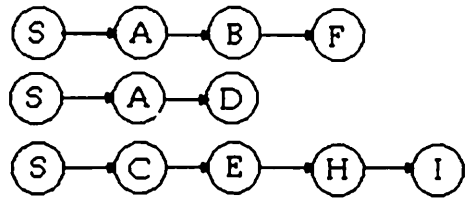


Fig 3.1 Path Cache

3.3 Caching Policy

As a result of a single route discovery process a source can obtain multiple routes. Routes between source and destination pairs may have a number of common intermediate nodes in-between. In a highly mobile scenario, if one of these nodes fails all routes that use this node for forwarding will also fail. Hence instead of caching all possible routes to a destination, only routes that are maximally disjoint are cached by GPEAR.

For example, from figure 3.2 various routes exist from node 4 to node16. Of all routes *4-5-9-11-16, 4-3-7-10-16, 4-2-6-8-12-14-16* are the routes that are maximally disjoint. If routes 4-5-9-11-16, 4-6-8-9-11-16 and 4-6-9-11-16 are cached; and nodes 9 or 11 fail then all three routes are invalidated, so the purpose of storing multiple routes is defeated.

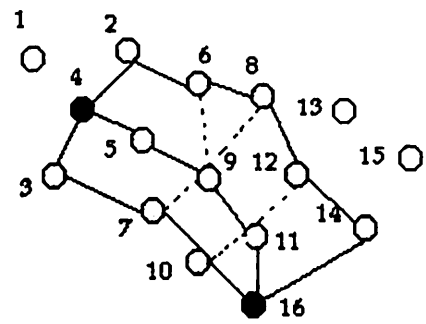


Fig 3.2 Multiple Routes

When a node receives or overhears a route, it compares the route obtained with the contents of the primary cache and secondary cache, if the route is maximally disjoint then it is transferred into the primary cache or secondary cache, depending upon how the route had been obtained. If the route that is being examined is better in terms of route length or RET [sec 4.2.1], then the new route replaces the one that is already cached. Also if the route that has been cached earlier is a stale route, then the newly obtained route replaces it.

Routes are maximally disjoint if for a route length the number of node matches are less than (Route_Length –Lopt). The size of the Primary Cache and Secondary Cache are fixed to P_size and S_size respectively. In the case when the cache becomes full, then old routes are deleted to make way for new ones.

If the cache had a FIFO policy then a route, which has been in use for a long time and still in use, may be deleted, because it was discovered first. To prevent a reliable and an active route from being deleted a Least Recently Used (LRU) policy is employed.

Each cache entry has a Time stamp against it. Every time a route is picked up from the cache for use, it automatically updates its Time stamp. So when a route has to be deleted to make way for a newly discovered route, the Time stamp is used to pick the Least Recently Used route. RET is also used as a metric for route replacement, stale routes with low RETs are replaced with newly cached routes.

This scheme will not work with the secondary cache, since routes stored there are not directly used, but maintained as an alternative. A route may be overheard multiple times. Each time the route is overheard its Time stamp in the secondary cache is updated. And the replacement policy here is Least Recently Overheard (LRO).

Other than the time stamp, against each entry in the cache, the Route Expiry Time (RET) is stored; every time a route recalculation occurs the RET entry is updated.

An intelligent caching scheme plays an important role in improving the network performance. The results presented in sec 5.3.2 shows the effect of caching scheme on the performance of a network.

3.4 Address Table

Since Source Routing is employed by GPEAR, the entire route is stored in the cache. MAX_TTL (i.e. maximum size of route allowed) can be as large as 25 in a network of 100 nodes. This makes the size of each entry at least 120 bytes if the IPv4 address format is used, and 420 bytes if IPv6 address format is used. If memory constraints in terms of bytes are applied, then the number of routes that can be stored in the primary or secondary cache will be very less. Hence it is possible that the same node is a part of multiple routes.

Instead of using the complete IP address each node is allotted a one byte local ID. This reduces the size of each entry in the cache to 25 bytes.

The address table (figure 3.3) in a node stores the IP address along with the local ID. Every time a new node is discovered, it is added to the address table and a local ID is generated.

Address	ID

Fig 3.3 Address Table

When a route is used, the local IDs of the node are replaced with the IP address using the address table. The ID allotted for each node is local to the node in which the address table is present.

Chapter 4 - UNICAST ROUTING
GPS - BASED PREDICTIVE ENERGY AWARE ROUTING (GPEAR)

4 .1 Introduction

The GPS based Predictive Energy Aware Routing Protocol (GPEAR) [26,27] can be called a pure on-demand route acquisition system; nodes that do not lie on active paths neither maintain any routing information nor participate in any periodic routing table exchanges. Furthermore, 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.

GPEAR's primary objectives are:

- **To provide unicast communication to all nodes in the adhoc network**
- **To minimise transmission of control packets**
- **To reduce route breaks by predicting link breakages in advance by using location information.**
- **To reduce energy consumed at each node by controlling the energy at which packets are transmitted or received by using location information.**

GPEAR uses a broadcast route discovery mechanism, and relies on dynamically establishing route table entries at a node, as is the case in any reactive protocol. GPEAR uses *Source Routing;* by virtue of source routing, GPEAR has access to a significantly greater amount of routing information than any other form of routing.

Protocols such as LAR [19] or DREAM [20] use location information to improve routing performance. However, neither of these protocols makes use of location information for route reconstruction or control of power at which packets are transmitted or received. Mobility of nodes to some extent is regular. By making use of the non-random movement pattern, we can predict expiry of routes as well as use this information for advance route

4 .2 Overview of the Protocol

4.2.1 Description of the Protocol

The GPS-based Predictive Energy Aware Routing protocol (GPEAR) is a simple and efficient routing protocol designed specifically for use in multi-hop wireless adhoc networks of mobile nodes. Using GPEAR, the network is completely self-organizing and self-configuring, requiring no existing network infrastructure or administration or prior initialisation. Network nodes cooperate to forward packets for each other to allow communication over multiple "hops" between nodes not directly within wireless transmission range of one another. As nodes in the network move about or join or leave the network, and as wireless propagation conditions such as sources of interference change, all routing is automatically determined and maintained by the GPEAR routing protocol. Since the number or sequence of intermediate hops needed to reach any destination may change at any time, the resulting network topology may be quite rich and rapidly changing.

The GPEAR protocol allows nodes to dynamically discover a source route across multiple network hops to any destination in the adhoc network. Each data packet sent, then carries in its header the complete, ordered list of nodes through which the packet will pass, allowing packet routing to be trivially loop-free and avoiding the need for up-to-date routing information in the intermediate nodes through which the packet is forwarded. By including this source route in the header of each data packet, other nodes forwarding or overhearing any of these packets may also easily cache this routing information for future use. The GPEAR protocol provides highly reactive service to help ensure successful delivery of data packets in spite of node movement or other changes in network conditions.

GPEAR is composed of two mechanisms that work together to allow the discovery and maintenance of source routes in the adhoc network:

- **Route Discovery** is the mechanism by which a node (S) wishing to send a packet to a destination node (D) obtains a source route to D. Route Discovery is used only when S

attempts to send a packet to D and does not already know a route to D. The source node (S) needs a route to any destination it broadcasts Route Request (**RREQ**) messages. Any node with a current route to that destination (particularly the destination itself) can unicast a Route Reply (**RREP**) back to the source node.

- **Route Maintenance** is the mechanism by which node S is able to detect, while using a source route to D, if the network topology has changed such that it can no longer use its route to D because a link along the route no longer works. Position information of all nodes on a route is constantly updated through the use of Route Update (**RUPDATE**) messages. RUDATE messages are generated at the destination and carry the current geographical co-ordinates of the nodes along the route. The expiry time of the route (termed as RET) is calculated using the position information available in the RUPDATE messages. Route breakages can be predicted in advance by the use of periodic Route Updates. When the Route Maintenance mechanism indicates a source route may be broken, S can attempt to use any other route it happens to know to D, or can invoke Route Discovery again to find a new route for subsequent packets to D. Route Maintenance for this route is used only when S is actually sending packets to D.

In GPEAR, Route Discovery is entirely "on demand". In particular, unlike other protocols, GPEAR requires no periodic packets of any kind at any level within the network. The Route Maintenance Mechanism in GPEAR on the other hand in proactive, since RUPDATE messages are sent out regular intervals along an active route. The proactive route mechanism however does not use any periodic broadcast of routing advertisement, link status sensing, or neighbor detection packets. This lack of periodic broadcast activity allows the number of overhead packets used by GPEAR to scale all the way down to constant value, when all nodes are approximately stationary with respect to each other and all routes needed for current communication have already been discovered. Network topology changes not affecting routes currently in use are ignored and do not cause reaction from the protocol. In

response to a single Route Discovery (as well as through routing information from other packets overheard), a node may learn and cache multiple routes to any destination. This allows the reaction to routing changes to be much more rapid, since a node with multiple routes to a destination can try another cached route if the one it has been using should fail. This caching of multiple routes also avoids the overhead of needing to perform a new Route Discovery each time a route in use, breaks.

The key feature of GPEAR is the use of **source routing**. That is, the sender knows the complete hop-by-hop route to the destination. These routes are stored in a **route cache**. The data packets carry the source route in the packet header. When a node in the adhoc network attempts to send a data packet to a destination for which it does not already know the route, it uses a *route discovery* process to dynamically determine such a route. Route discovery works by flooding the network with route request (**RREQ**) packets. Each node receiving a RREQ rebroadcasts it, unless it is the destination or it has a route to the destination in its route cache. Such a node replies to the RREQ with a route reply (**RREP**) packet that is routed back to the original source. RREQ and RREP packets are also source routed. The RREQ builds up the path traversed so far. The RREP routes itself back to the source by traversing this path backwards.

The route carried back by the RREP packet is cached at the source for future use. If any link on a source route is broken, the source node is notified using a route error (**RERR**) packet. The source removes any route using this link from its cache. A new route discovery process must be initiated by the source, if this route is still needed. GPEAR makes very aggressive use of source routing and route caching. No special mechanism to detect routing loops is needed. Also, any intermediate node caches the source route in a packet it forwards for possible future use.

A detailed description of the protocol is presented in the following sections.

4.2.2 Route Expiry Time

A route is as strong as its weakest link; hence the Route Expiry Time (RET) is the minimum Link Expiry Time (LET) of the route. For eg from figure 4.1 for the route A B C D E F, between nodes A and F the link expiry times are L1, L2, L3, L4, L5 for the links A-B, B-C, C-D, D-E, E-F respectively. The RET is then min (L1, L2, L3, L4, L5).

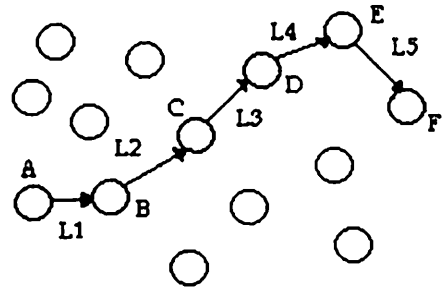


Fig 4.1 Route Expiry Time

Predicting Route Expiry Time

If the motion parameters of two neighboring nodes (i.e. speed, direction, radio propagation range, etc.) are known, the time for which the two nodes will remain connected can be estimated. For example from figure 4.1, nodes A and B are within transmission range **r** of each other. The current co-ordinates (time **t**) of mobile node A are (x_a, y_a) and the co-ordinates of node B are (x_b, y_b). If the previously cached co-ordinates (at time t_p) of mobile node A are (x_{ap}, y_{ap}) and the co-ordinates of mobile node B are (x_{bp}, y_{bp}). The velocities of A and B are v_a and v_b respectively and θ_a and θ_b $(0 \leq \theta_a, \theta_b < 2\pi)$ are the directions in which A and B are moving. The velocity and direction information can be obtained from the current and previous position of Node A and Node B.

The velocity and the direction information of nodes A and B are obtained by:

$$v_a = \sqrt{(v_{xa})^2 + (v_{ya})^2}\ ; \qquad v_b = \sqrt{(v_{xb})^2 + (v_{yb})^2}$$

$$\theta_a = \tan^{-1}(v_{ya} / v_{xa})\ ; \qquad \theta_b = \tan^{-1}(v_{yb} / v_{xa})$$

Where:

$$v_{xa} = \frac{x_a - x_{ap}}{t_p - t}\ ; \qquad v_{ya} = \frac{y_a - y_{ap}}{t_p - t}\ ; \qquad v_{xb} = \frac{x_b - x_{bp}}{t_p - t}\ ; \qquad v_{yb} = \frac{y_b - y_{bp}}{t_p - t}$$

Then, the amount of time the mobile hosts will stay connected, L_t is predicted by:

$$L_t = \frac{-(mn+op)+\sqrt{(m^2+n^2)r^2-(mp-on)^2}}{m^2+n^2}$$

Where:

$m = v_a \cos\theta_a - v_b \cos\theta_b$, $n = x_a - x_b$, $o = v_a \sin\theta_a - v_b \sin\theta_b$, and $p = y_a - y_b$

In case there has been no movement then L_t (Link Expiry Time) becomes ∞. The minimum of the estimated LETs gives the RET.

4.2.3 Routing Structures

Other than the Route Cache described in Chapter3, GPEAR maintains various tables to assist the protocol in the process of route establishment and maintenance.

4.2.3.1 Neighbor Table (Table 4.1)

Each node running GPEAR, maintains a Neighbor table that is used for recording the position of next hop nodes. Neighbor Table aids GPEAR in the process of RET prediction. The fields of the neighbor table are as follows:

- Neighbor IP Address *(Node ID)*
- Node's Previous X Position *(My Position – Xprev)*
- Node's Previous Y Position *(My Position – Yprev)*
- Node's Current X Position *(My Position – Xcur)*
- Node's Current Y Position *(My Position – Ycur)*
- Neighbor's Previous X Position *(Neighbors Position – Xprev)*
- Neighbor's Previous Y Position *(Neighbors Position – Yprev)*
- Neighbor's Current X Position *(Neighbors Position – Xcur)*
- Neighbor's Current Y Position *(Neighbors Position – Ycur)*
- Time, at which the Entry was last updated. *(Time)*
- LET *(LET)*

Node Id	My Position				Neighbors Position				Time	Let
	Xprev	Yprev	Xcur	Ycur	Xprev	Yprev	Xcur	Ycur		

Table 4.1 Neighbor Table

On receiving the position of its neighbors, as a part of control packets, a node updates its table by transferring Xcur, Ycur to Xprev and Yprev and transferring the positions received (from the control packet) to Xcur and Ycur. Each time a neighbor's position is updated, a node also updates its own position information and the time stamp. LET is calculated using the formula for L_t given in section 4.2.2. When a node is added to the neighbor table for the first time (i.e. its Xcur and Ycur fields are empty) a Static LET is calculated using, the distance between the nodes and an estimated average speed of movement (V_{avg}). Then the LET is given by:

$$L_t = \frac{r - \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}}{V_{avg}}$$

Where x_2, y_2 – are the neighboring nodes co-ordinates

x_1, y_1 – are the nodes own co-ordinates.

4.2.3.2 Request Table (Table 4.2)

Every Time a node attempts an RREQ (sec 4.2.4) to the destination, either for itself or on behalf of another node it updates its Request Table. The fields in the request table are:

- ID of the requesting node
- Route Request sequence number.
- Number of pending Route Requests
- The time at which the last Route Request was sent
- The time at which the last ARP was sent

When a node sends or receives a RREQ it updates its request table. It checks its request table for any previously received RREQ from the same node, if there exists such an

entry, it checks the sequence number field. If the stored sequence number is less than the sequence number on the RREQ message, then the Request Table is updated and the RREQ message is propagated further. Otherwise it is a stale RREQ propagating around the network, and needs to be ignored. Thus GPEAR is able to filter out stale route requests.

Node Id source	Node Id dest	Req No.	No.of pending RtReq	LTime	
				ARP	RtReq

Table 4.2 Request Table

The request table is constantly monitored by a timer-based function, which removes any stale RREQ entry, and if the number of outstanding RREQ exceeds a constant MAX_RREQ_RETRIES, then the entry for that node is deleted. The RREQ entry for a node is also removed when a RREP is received for the requested destination.

4.2.3.3 SRRoute Table (Table 4.3)

SRRoute Table is used for Ring Search (sec 4.2.4) by the source node. The fields in the SRRoute are:

- ID of destination node, for which RREQ was sent out.
- TTL.

Net ID	TTL

Table 4.3 SrRoute Table

When a source node sends out a RREQ it records the TTL at which the RREQ was sent in its SRRoute table.

4.2.3.4 Energy Table (Table 4.4)

While other Routing Tables are used for Route Acquisition, Energy Table is used for the forwarding data packets. The fields in the Energy Table are:

- Neighbors ID (*Node Id*)
- Power at which data packets have to be forwarded to the neighbor. (*TxPow*)
- Power consumed because of this process. (*TxCons*)

Node Id	TxPow	TxCons

Table 4.4 Energy Table

4.2.3.5 Dest Table (Table 4.5)

The Dest Table is used for prediction of LET, as a part of route maintenance (sec 4.2.5). The fields of the Dest Table are:

- ID of the Source Node
- The Source Route along which the last data packet was received.
- The time at which the last Route Update (RUPDATE) was sent.

SrcID	SrcRoute	Tupdate

Table 4.5 Dest Table

4.2.4 Route Discovery

Route Discovery in GPEAR is purely on-demand and occurs when a node requires a route to destination for which it does not already have a recorded route. Such a node initiates route discovery by broadcasting a Route Request (RREQ) packet. The message format of the RREQ is as shown in fig 4.2

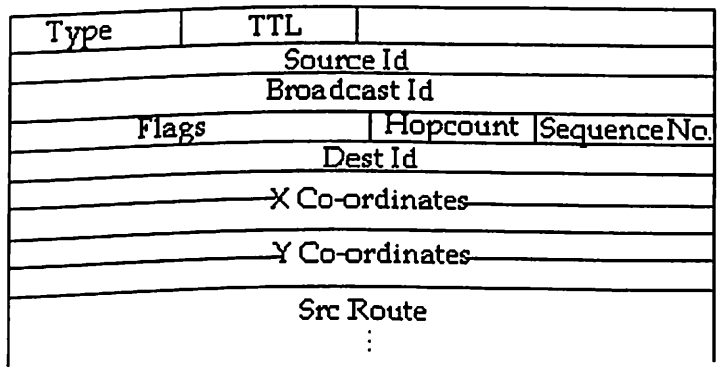


Fig 4.2 Format of RREQ

Route Requests (RREQ)

No flags are used in case of RREQ messages. The Hop Count field is initialised to zero by the source of the RREQ and is incremented each time the RREQ is forwarded. The sequence number of the RREQ ensures the freshness of the RREQ to the node. Each time a RREQ is re-propagated for the same destination the RREQ sequence number is incremented

by the source. The node requesting the route places its own IP address, broadcast ID, the destination's IP address, sequence number of the request and its X and Y co-ordinates in their respective fields and broadcasts the RREQ.

Forwarding of Route Requests

A node receiving a RREQ first updates its neighbor table (the last node in the source route is the neighbor), and its request table. The source route in the header is reversed, and stored its primary cache. This reverse route may be used later. The node checks its cache to see whether it has a route to the requested destination. In order to respond to a RREQ, a node must be the destination itself, or it must have an unexpired route to the destination with a TTL lesser than MAX_TTL. If this is the case, the node generates a RREP. Otherwise it rebroadcasts the RREQ. Before it does so, it increments the hop count, replaces the X-Y co-ordinates in the header with its own, adds its IP address to the source route. Figures 4.3 a, 4.3b, 4.3c show the propagation of route request through the network (the shaded nodes are the ones that have received the RREQ broadcasts).

A node may receive the same RREQ multiple times. The RREQ is propagated only if the sequence number of the RREQ is greater than the sequence number stored in request table, otherwise it is dropped. ***Ring Search*** is used to limit the broadcast range of RREQ. This technique allows the source node to search increasingly larger areas of the network if a route to the destination is not found. Each time a node initiates a route discovery process for some new destination, it must broadcast a RREQ across the network. For a small network, the impact of this flooding is minimal. However, for a large network, the impact may become increasingly detrimental. To control network-wide broadcasts of RREQs, the source node can use expanding ring search. This technique allows the source node to search increasingly larger areas of the network if a route to the destination is not found. To use the expanding ring search the source nodes sets the Time To Live (TTL) value of the RREQ to an initial value. If a RREP is not received within the discovery period, then the TTL is increased by an

incremental value, this process is repeated until a threshold (TTL_threshold) is reached. After which the TTL is set to MAX_TTL and then propagated through the entire network. The TTL with which each RREQ is propagated is recorded in the Request Table. If a route expires and a RREQ (secondary RREQ) is propagated through the network to re-establish a route then, the RREQ is propagated with the TTL recorded in the request table. This allows the source node to first search the area where the destination was last present.

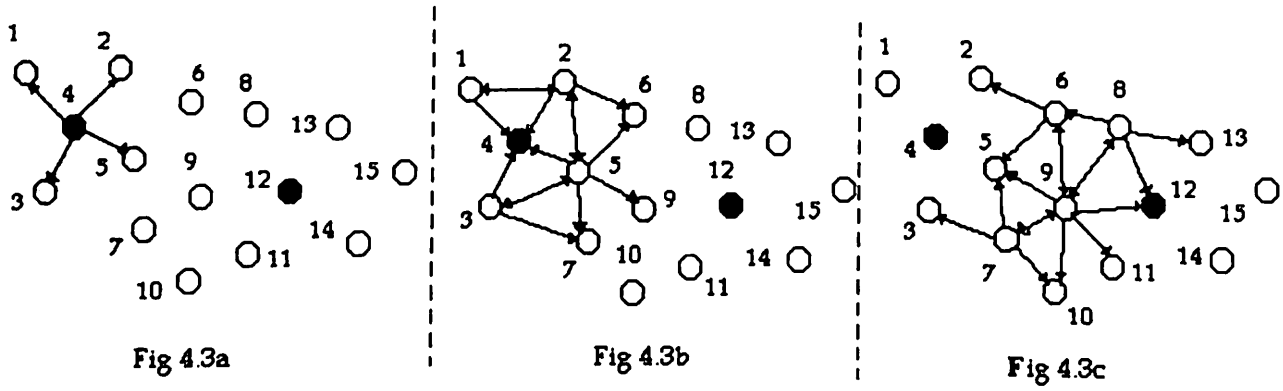


Fig 4.3 Propagation of Route Request

Route Reply (RREP)

As stated in the previous section, a node can respond to the RREQ if it is the destination itself, or if it has current route to the destination. When a node fulfils either of these requirements, it unicasts a RREP back to the source node. The format of the RREP message is as shown in fig 4.4.

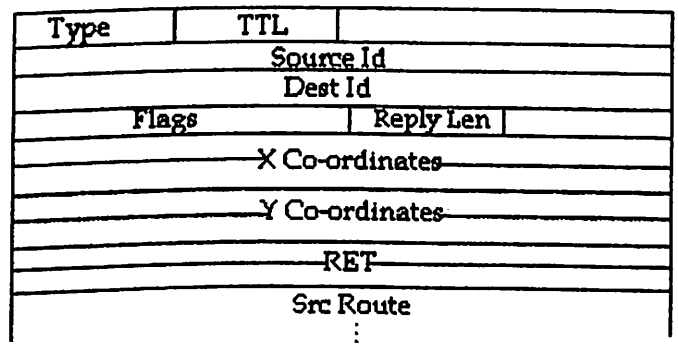


Fig 4.4 Route Reply Format

The only flag that is of any significance in RREP is the Gratuitous Flag (G), to indicate whether the RREP is gratuitous. The Reply length field gives the hop count of the path between source and destination. The destination places its X-Y co-ordinates and RET in

their respective fields in the RREP message; RET placed by destination will be MAX_RET. If the node responding is not the destination; then the route length is the sum of the number of hops between the source node and responding node, and the number of hops between the responding node and the destination node. The RET will be the RET of the route between the responding and the destination nodes.

A node receiving the RREP, updates its cache and neighbor table, it then unicasts the RREP to the next hop in the source route. Before doing so, it replaces the x-y co-ordinates in the RREP with its own, calculates the LET between itself and the node from which it has received the RREP. The RET in the RREP is replaced by the minimum of LET calculated and the RET received with the RREP. This continues until the RREP reaches the source node. Figure 4.5 is an example of the destination node responding by sending RREP back to the source.

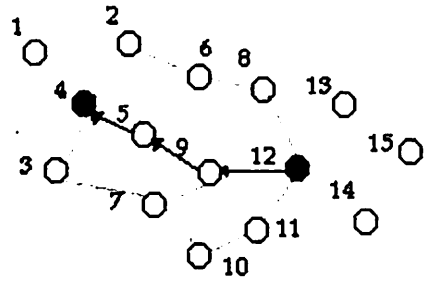


Fig 4.5
Propagation of Route Reply

Once the source node receives the RREP, it can use the route to send data packets to the destination. The source route and its RET are transferred to the primary route cache. In, the event, that more than one RREP is received the selection of the route is obtained using the relationship:

W1 * (RET) + W2 * (number of hops) + W3 * (time at which the route was cached)

Where: RET – is the Route expiry time that is calculated based on position information.

Number of hops – is the number of intermediate hops between the source and destination node pair.

The weights (W_1, W_2, W_3) allotted to the three selection criteria need not be evenly distributed. Allocation of weights is based on the mobility pattern, as the mobility increases the weightage allotted to RET is increased [27,28]. During initial route formation, the weights are allotted statically; as the mobility pattern emerges, the protocol (GPEAR) automatically updates weights. GPEAR can allocate any value ranging from 0 –1, based on the estimated movement of nodes within the network. The movement pattern is obtained from the position information that is circulated in the RUPDATE messages (sec 4.2.5). All routes obtained using the formal route discovery process are stored in the primary cache.

Gratuitous Route Replies

Since overhearing is allowed, and it is possible because of the broadcast nature of a wireless network, a node overhearing a RREQ, can respond with a RREP if it has a recently cached route to the destination. In this case the G-flag is set indicating a Gratuitous RREP. The format of RREP remains the same. The source ID will be that of the node sending the RREP, and X-Y co-ordinates will also be that of the responding node.

In case a RREP is not received within RREQ_TIMEOUT seconds with TTL value equal to MAX_TTL, the source node assumes that a route is not available to the destination.

4.2.5 Route Maintenance

Conventional Routing Protocols integrate route discovery with route maintenance by continuously sending periodic routing updates. If the status of a link or router changes, the periodic updates will eventually reflect the changes to all other routers, presumably resulting in the computation of new routes. However, using route discovery, there are no periodic messages of any kind from any mobile hosts. Instead, while a route is in use, the route maintenance procedure monitors the operation of the route and informs the sender of any possible routing errors.

Since the protocol is predictive in nature, route breakages are detected in advance, and routes to a destination in use are repaired in advance. To predict the state of a route regular Route Update (RUPDATE) messages are unicasted from the destination to the source, along the route over which it had last received a data packet. The format of a RUPDATE message is shown in figure 4.6

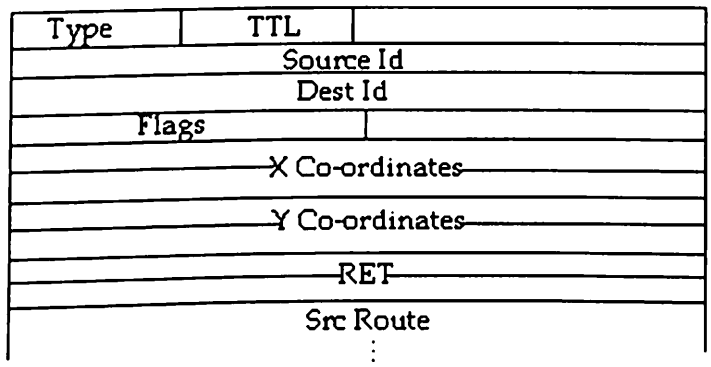


Fig 4.6 Format of RUPDATE message

The destination places its X-Y co-ordinates and an RET equal to MAX_RET. The packet is forwarded in the direction of the source. An intermediate node receiving the RUPDATE packet updates its neighbor table and calculates the LET using the X-Y information sent by the node forwarding the packet. The node checks whether the LET calculated is greater than T_OPT. If the value is greater, the node replaces the RET in the RUPDATE header with the minimum of the calculated LET and the received RET. It also replaces the X-Y information in the header with its own X-Y co-ordinates and forwards it to the next node on the source route. This process is continued until the RUPDATE reaches the source. A LET lesser than T_OPT is an indication of an imminent link break, so the node attempts *local repair.*

Local Repair

From figure 4.7 the path between source node 4 to destination node 12 is 4-5-9-12. The route fails if any of the links 4-5 or 5-9 or 9-12 break. A RUPDATE message originated by node 12 will travel via node 9 and node 5 to node 4. If node 9 is moving in a direction

opposite to that of node 12, it will be detected when node 9 calculates the LET, hence it will attempt route repair.

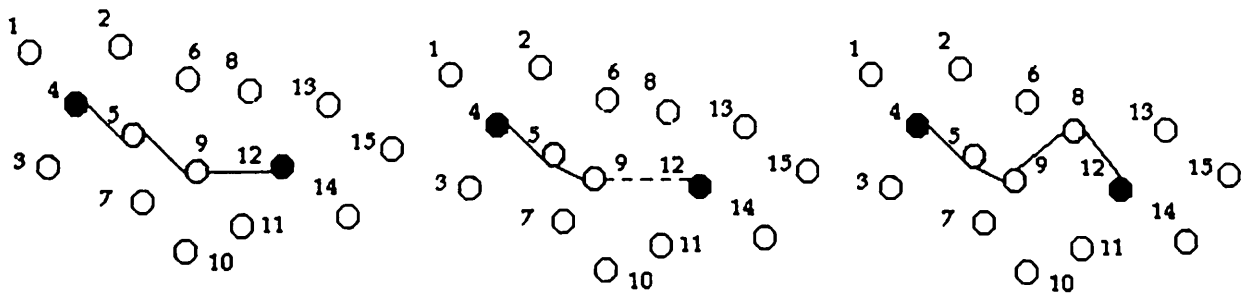


Fig 4.7 Route Maintenance thro' Prediction and Local Repair

Node 9 sends a Secondary Route Request (SRREQ) for node12. The format of SRREQ is shown in figure 4.8. The main difference between the normal RREQ and SRREQ is that the SRREQ will have the last known position of the destination on it. When a node receives a SRREQ it checks the destinations co-ordinates in the header. If it is along the direction of destination, then SRREQ is forwarded otherwise it is dropped. This limits the RREQ broadcasts directionally. The TTL in header is set to the hop count between nodes 12 and 9 plus 1. Ring search is also employed in case of local repair.

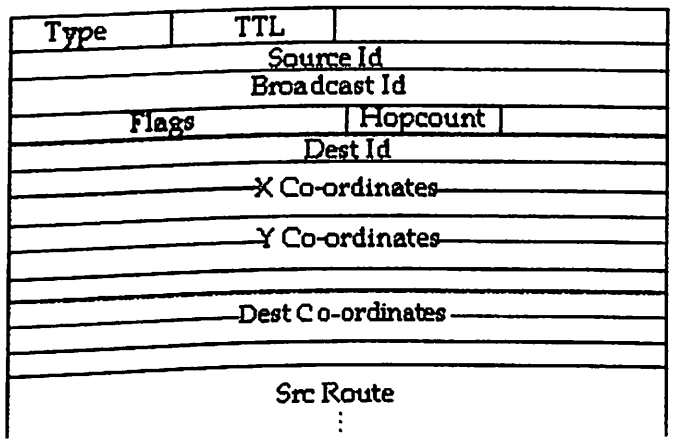


Fig 4.8 Format of SRREQ message

As shown in fig 4.7 node 9 may form a route to node 12 through node 8. When a new route is formed as a result of local repair a gratuitous RREP is sent to the source, with the Repair flag (R) set to indicate that the route has been obtained as a result of an update process. Henceforth the data packets will be routed through this new path.

In cases, where local repair completely fails, or a node shuts down, then Route Error process takes over.

Route Error

When a link failure with a neighbor occurs, the node removes the entry of that neighbor from the neighbor table, removes any route that will involve the neighbor from its cache, it then unicasts a RERR message to the source and the destination. The format of the RERR message is shown in figure 4.9.

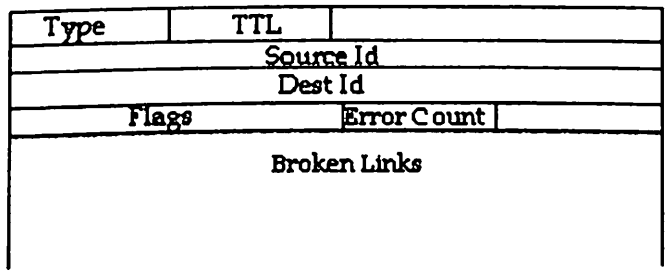


Fig 4.9 Format of RERR Message

The error count field gives the number of links that are down. When a node receives a RERR it updates its routing structures and forwards the RERR message. When the RERR reaches the source, it examines its cache for an alternate route, if not available, it sends out a SRREQ with the last known co-ordinates of the destination.

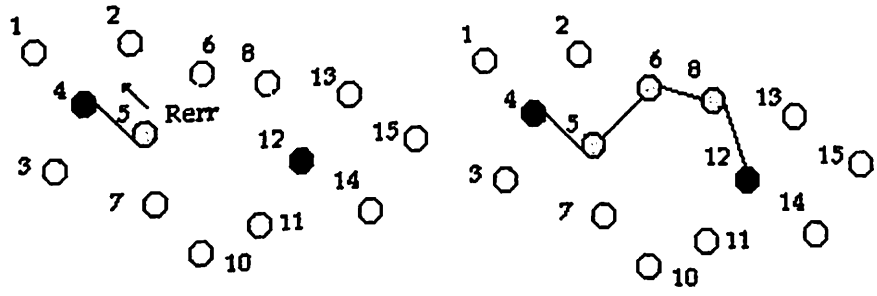


Fig 4.10 Route re-formation

Fig 4.10 illustrates the process by which RERR messages and routes are repaired, due to the shutdown of node 9.In case of figure 4.10 the RERR is sent only in the direction of the source, since node 9 is the penultimate node in the source route and it is node 5 that has detected the link error. The source does route discovery process again to obtain the route 4-5-6-8-12.

In the case when a node at the centre of the source route fails, the RERR message will be sent either in the direction of the source or destination depending upon the node detecting the link error. If the node upstream, to the erroneous node, detects the link error then the RERR is propagated in the direction of the source, else it is sent in the direction of the destination

4.2.6 Energy Consumption

A node employing GPEAR varies the energy level at which each data packet is transmitted according to the transmission distance. Control packets, except for the RUPDATE messages, are sent with maximum possible energy. The Energy Table (sec 4.2.3.4) carries information regarding the neighbors' last known position. Based on this information, the energy with which the data packets have to be transmitted is calculated.

The data stored in the energy table determines the energy with which each data packet is forwarded to its neighbors. RUPDATE messages are sent at a slightly higher energy level as compared to the data packets; this is to ensure that the RUPDATE messages are not lost due to node movement. The direction of transmission, as in case of any on-demand routing protocol, remains omni-directional. Due to the control of energy levels at which data packets are transmitted, the overall energy consumption in the network is reduced considerably.

A detailed analysis of GPEAR and its results are provided in the next chapter (Chapter 5).

Chapter 5 - IMPLEMENTATION AND RESULTS

GPS based Predictive Energy Aware Routing For MANETs (GPEAR)

5.1 Introduction

The validation and evaluation of adhoc routing protocols can be exceedingly difficult. Creating repeatable scenarios with tens, hundreds, or even thousands of mobile nodes, or creating multiple scenarios with only small variations, is quite challenging. One of the methods by which protocol design can be tested is through formal verification of the protocol.

Simulation is another tool that can validate the operation of the protocol. Simulation enables a protocol to be tested in numerous scenarios, where one parameter or metric can be isolated to test the effects of that variable. To study the characteristics of GPEAR, simulations of the protocol have been created to test it in variety of repeatable scenarios. Simulation provides a method by which, the basic functionality of the protocol can be validated, and it can be an invaluable tool in refining the protocol design. Numerous simulations, with minor variations that address specific aspects of the protocol, can be run.

To study the performance of GPEAR, its unicast operation has been simulated. For unicast, it is important to establish that GPEAR is able to find routes whenever such routes exist and are needed. This can be evaluated through examination of the number of data packets that are delivered to their destinations. The more the data packets that GPEAR is able to deliver, the better its route finding and maintaining ability.

The simulations described here were performed using the *ns-2* Network Simulator developed by University of California, Berkley [28]. The simulator *ns-2* is written in C++

[29] with an Object-orientated Tool Command Language (Otcl) [30,31] interpreter as front end. The simulator supports a class hierarchy in C++, and a similar class hierarchy within OTcl interpreter. The root of this hierarchy is the class Tcl Object. New simulator objects are created through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class TclClass; user instantiated objects are mirrored through methods defined in the class TclObject.

The simulator uses two languages because *ns-2* has two different kinds of functions to perform. On the one hand, detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks, the run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important.

On the other hand, a large part of network research involves sensitivity of the network behavior to slightly varying parameters or configuration variations, or quickly exploring a number of scenarios. In such cases, iteration time (change the model and re-run) is more important. Since the node configuration script runs only once (at the beginning of the simulation), run-time of this part of the task is less important.

C++ and Otcl meet both of these needs of *ns-2*. C++ is fast to run, but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. Tclcl provides a glue to make objects and variables appear on both languages.

The GPEAR *ns-2* simulations were carried out on the *Windows* operating system.

5.2 Simulation Environment

The network stack that is implemented in *ns-2* is shown in fig 5.1. This network stack is a permanent format that is used by *ns-2* for routing in MANETs. The routing protocol GPEAR has been implemented and added to the network stack in *ns-2*.

5.2.1 Mobility Model

Node mobility in these simulations has been modeled using the Random Waypoint mobility model [32]. At the beginning of the simulation, the Random Waypoint Mobility model randomly places the nodes within the predefined simulation area. Each node then selects a destination within that area and a speed from a user-defined maximum speed. The node travels to its selected destination at the selected speed. Once it reaches the destination, it is a stationary for some pre-defined *pause time*. At the end of the pause time, it selects a new destination-speed combination, and then resumes movement. This model causes continuous changes in the topology of the network.

Programs for scenario-generation are available in *ns-2*. These programs use *UNIX-OS* specific commands and hence run only on *UNIX* environment. Using the basic algorithm for random waypoint model available in *ns-2*, scenario-generation has been implemented using VC++. The routines developed for scenario-generation are independent of *ns-2*, and do not use any *ns-2* specific classes or methods.

While running the simulations, an interesting behavior of the mobility model was noticed. The average number of neighbors observed at a given node periodically increases and decreases as the simulation progresses, where the frequency of change is relative to the speed of the nodes. Figures 5.2 to 5.5 illustrate the average number of neighbors per node in a 100-node network during 1000 seconds of simulated time. The room size for these

simulations was varied from 500mx1000m to 1500x1500m. The nodes have a transmission radius of 250m.

The fluctuation seen in these figures is due to the inherent characteristics of the mobility model. Because a node must choose a destination in the simulation area, the node is most likely to travel in the direction in which there are the most destinations from which to choose. This predisposes nodes to choose destinations in middle of the area, or destinations that they reach by traveling through the middle. This characteristic creates a situation in which density waves occur. Nodes seemingly converge in the center of the area, then diverge, and then re-converge. This effect can be seen in figures 5.6 to 5.8. The figures show the movement pattern of some of the nodes at varying speeds of 1m/s to 30m/s.

In order to enable direct, fair comparisons between the protocols, it was essential to ascertain the ruggedness of the protocols with identical loads and environmental conditions. Each run of the simulator accepts as input a *scenario file* **(APPENDIX-A has an e.g. of a typical scenario file)** that describes the exact motion of each node and the exact sequence of packets originated by each node, together with the exact time at which each change in motion or packet origination is to occur. **1750 different scenario files** (details are described in sec 5.3.4) with varying movement patterns, speeds and network area were pre-generated. All routing protocols were run against each of these scenario files. Since each protocol was tested under similar conditions in an identical fashion, it is reasonable to directly compare the behavior of the protocols.

Around 30,000 simulations where run to obtain the results that have been analyzed in this chapter.

5.2.2 Physical and Data Link Layer Model

To accurately model the attenuation of radio waves between antennas close to the ground, radio engineers typically use a model that attenuates the power of a signal as $1 / r^2$ at short distances (where r is the distance between the antennas), and as $1/r^4$ at longer distances. The crossover point is called the *reference distance*, and is typically around 100 meters for outdoor low-gain antennas 1.5m above the ground plane operating in the 1–2GHz band [33]. Following this practice, the signal propagation model combines both a free space propagation model and a two-ray ground reflection model. When a transmitter is within the reference distance of the receiver, the free space model is used where the signal attenuates as $1 / r^2$. Outside of this distance, we use the ground reflection model where the signal falls off as $1/r^4$.

Each mobile node has one or more wireless network interfaces, with all interfaces of the same type (on all mobile nodes) linked together by a single physical channel. When a network interface transmits a packet, it passes the packet to the appropriate physical channel object. This object then computes the propagation delay from the sender to every other interface on the channel and schedules a "*packet reception*" event for each. This event notifies the receiving interface that the first bit of a new packet has arrived. At this time, the power level at which the packet was received is compared with two different values: **the carrier sense** threshold and the **receive** threshold.

If the power level falls below the carrier sense threshold, the packet is discarded as noise. If the received power level is above the carrier sense threshold but below the receive threshold, the packet is marked as a packet in error before being passed to the MAC layer. Otherwise, the packet is simply handed up to the MAC layer.

Once the MAC layer receives a packet, it checks the packet to ensure that its receive state is presently "idle." If the receiver is not idle, it indicates that a packet (p_1) is already

being received; hence, one of the two things can happen. If the power level of the packet (p_1) already being received is at least 10 dB greater than the received power level of the new packet at the wireless interface, capture is assumed. As a result of the capture, the new packet is discarded, and the receiving interface is allowed to continue with its current receive operation of packet p_1. Otherwise, it is presumed that a collision has occurred and both packets are dropped.

If the MAC layer is idle when an incoming packet is passed up from the network interface, it simply computes the transmission time of the packet and schedules a "*packet reception complete*" event for itself. When this event occurs, the MAC layer verifies that the packet is error-free, performs destination address filtering, and passes the packet up the protocol stack.

5.2.3 Medium Access Control

The link layer of the simulator implements the complete IEEE 802.11 standard [34,35] Medium Access Control (MAC) protocol with Distributed Coordination Function (DCF) in order to accurately model the contention of nodes for the wireless medium. DCF is similar to MACA [36] and MACAW [37] and is designed to use both *physical carrier sense* and *virtual carrier sense* mechanisms to reduce the probability of collisions due to hidden terminals. The transmission of each unicast packet is preceded by a Request-to-Send/Clear-to-Send (RTS/CTS) exchange that reserves the wireless channel for transmission of a data packet. An Acknowledgment (ACK) follows each unicast packet correctly received to the sender, which retransmits the packet a limited number of times until this ACK is received. Broadcast packets are sent only when virtual and physical carrier sense indicate that the medium is clear, but they are not preceded by an RTS/CTS and are not acknowledged by their recipients.

5.2.4 Address Resolution

Since the routing protocols all operate at the network layer using IP addresses, an implementation of ARP [38], modeled after the *BSD UNIX* implementation [39], was included in the simulation and used to resolve IP addresses to link layer addresses. The broadcast nature of an ARP REQUEST packet and the interaction of ARP with on-demand protocols make ARP an important component of the simulation.

5.3 Simulation Results and Analysis

5.3.1 AODV Vs DSR

Both AODV and DSR have received a lot of attention in recent times. Numerous comparisons of AODV and DSR have been carried out on various simulators [40,41,42,43]. These comparisons provide varied results and conclusions. To confirm these results and supplement them, simulation based analysis of AODV and DSR was done.

All previous comparisons of the two protocols have been carried out using a 50-node scenario, low traffic load and low speeds. These comparisons were done for a short periods of time ranging from 100 seconds to 900 seconds. It was assumed that the behavior of a network could be predicted from its behavior in its first 900 seconds. To verify the validity of this assumption simulations were carried out for varying time periods. The simulations were carried out using DSR as the routing protocol. The results of these simulations are shown in fig 5.9. Packet delivery ratio (**PDR**) is the metric that has been used as a basis for this comparison and the protocol tested is DSR.

The network scenarios on which the test was carried out were stationary (i.e.) node mobility was zero. In case of mobile scenarios the density pattern of the network, changes with node mobility, causing variations in Packet delivery ratio. The number of nodes in the network was set to 100, within a network area of 1000x1000m. The results in fig 5.9 are an average of 10 simulations. As node positions are chosen at random, different scenarios produce varying results. Hence an average of 10 such scenarios is taken so as to gauge the actual performance of the protocol. A total of 25 connections were set-up with a traffic rate of 4 packets/sec. The type of traffic simulated was UDP-CBR [44,45].

The packet delivery ratio at the end of 100 secs was found to be 99.99%, and the packet delivery ratio (PDR) drops with time. At 500 secs, the PDR has dropped to 99.87%, dropping further to 99.58% at 1000 secs. The decrease in PDR (fig 5.9) is almost exponential in nature. The PDR relatively stabilizes around 1500 secs. The stability in PDR can be explained by examining the cache statistics. The cache statistics of one particular scenario is shown through figs 5.10 to 5.15. The scenario for which cache statistics was plotted is shown in fig 5.16. The nodes, which are either the source or the destination of the traffic, are highlighted in fig 5.16.

The number of routes recorded in the primary and the secondary cache has been traced at various time intervals. The number of entries in the primary and secondary cache has been restricted to 100 entries, with the maximum route length set to 25 (which is a fourth of the number of nodes in the network).

The activity in the cache, which is the number of routes added or deleted from the cache, is high for the first 500 seconds of the simulation. The cache activity continues albeit at a lesser rate even after 900 seconds. Not only the secondary cache, but the primary cache also gets updated, i.e. the formal route discovery process was still going on. DSR has no

mechanism to halt a route discovery process, once a route to the destination has been discovered. The cache activity reduced at around 1500 seconds and reached an insignificant level at 3000 seconds, especially in case of the source and destination nodes.

DSR searches the cache for the shortest route for every packet it sends out. As newer routes are discovered or overheard, the routes used by the data packets keep changing, providing varied PDR. The change in PDR due to the latently cached routes may not be significant in case of DSR, as it uses the shortest route as its routing criteria. But GPEAR uses a combination of shortest route, the time at which the route was cached and RET as the routing criteria. Since the time at which the route was cached is also a routing criterion, the variation in the route used by each packet is higher in case of GPEAR [26,27].

Another reason for the varying PDR is the interface queue, which buffers the data and the control packets. DSR uses a drop-tail priority queue, which gives a higher priority to control packets as against the data packets, as a result, as long as the control traffic persists, data packets will be dropped, causing a variation in PDR.

Since network behavior stabilizes at about 3000 seconds with the current cache size and because GPEAR also uses source routing mechanisms, most of the simulations that were run to analyze the behavior of GPEAR were restricted to a simulation time of 3000 seconds.

A similar analysis was carried using AODV as the routing protocol. The PDR as a function of simulation time is depicted in fig 5.17. In order to obtain a fair comparison, the network scenarios and traffic loads, used for analyzing the performance of AODV as a function of the simulation time, were same as the ones used for the analysis of DSR.

The PDR of AODV unlike that of DSR was found to be relatively constant for the first 1000 second of the simulation dropping only from 99.85% to 99.54%. The PDR dropped

exponentially from 98.799% at 1500 seconds to 90.73% at 10,000 seconds, after which the PDR stabilized to around 91%. This peculiar behavior of AODV as compared to DSR is due to the heavy broadcast storms that occur due to the nature of AODV's route maintenance. The initial stability in AODV's behavior is due to the fact that AODV does not follow an aggressive caching policy, and any route discovery process is halted once a route to the destination has been obtained

The number of control packets as a function of simulation time is shown in fig 5.18. The number of control packets rose rapidly from 11,489 packets at 100 seconds to 14,77,197 packets at 15,000 seconds; whereas in case of DSR (fig 5.19), the increase in control packets was from only 521 at 100 seconds to 22,870 at 3000 seconds, with very little increase in control overhead to 23,570 at 10,000 seconds.

In case of DSR very few routes are cached or deleted after 3000 seconds. Consequently, the control overhead is nearly constant after 3000 seconds. In AODV every node, as a part of the route maintenance mechanism generates *"HELLO"* packets to its neighbors, every few seconds, to indicate it is alive. As the simulation proceeds, the number of these "HELLO" packets increases (fig 5.20), and the control packets clash with the data packets for transmission. Due to the nature of the drop-tail queue, which has control packet priority, the number of data packets dropped as against the control packets increases.

The pattern of the control packet increase (fig 5.18) is entirely different from the HELLO packet increase pattern (fig 5.20). Fig 5.21 depicts the composition of HELLO packets in the control overhead as a function of the simulation time. The ratio of HELLO packets to the total number of control packets decreased with time from 87% at 100 seconds to 14% at 10,000 seconds, before it increased to68% at 15000 seconds. This behavior when

interpolated with the plot of the ratio of the RERR packets to HELLO packets as a function of time (fig 5.22) can be used to explain the behavior of AODV as regards the PDR.

As broadcast storming increases due to the HELLO packets, a number of data packets are dropped, causing RERR messages to be sent out. These RERR trigger either local repair or rediscovery of routes, as a result RREQ messages and RREP are generated, adding to control overhead. The ratio of RERR packets to HELLO packets increased from 0 at 100 seconds to 21% at 10,000 seconds before it dropped to 2.8% to 15,000.

Due to the effect of broadcast storming the PDR drops with the simulation time. When the storm settles down at around 10,000 seconds, the network stabilizes, producing a relatively stable, though low PDR of 91%. This can be furthered explained using fig 5.23, which is a plot of the percentage of packets received non-optimally (i.e.) packets that are received using longer routes when shorter ones are available. The rate at which the number of packets received non-optimally rose from 18.78% at 100 seconds rapidly to 59.73 % at 10,000 seconds. This increase in route length, despite only a single route being cached by the route discovery mechanism of AODV (AODV drops all routes except the shortest one), could only mean that the routes where lengthened due to the local repair mechanism triggered as a result of route errors. In a stationary network, with a huge amount of initial battery power (1000 Joules) the main reason for route breaks, or drop of packets, can be solely attributed to broadcast storming.

In case of DSR (fig 5.23b) non-optimality increased slightly from around 20 % at 100 seconds to 28% at 3000 seconds before it relatively stabilized at that value. This is again consistent with the caching behavior, and the relatively stable control overhead of DSR.

The effect of broadcast storms is particularly pronounced in AODV. ***Broadcast storms were one of the reasons why GPEAR's route maintenance mechanism was***

restricted to the active route and is unicasted along the reverse route instead of being broadcasted.

This comparison of AODV and DSR was carried out to supplement the already done work and to analyze the nuances of the behavior of an adhoc network, so as to develop a protocol (GPEAR) that will overcome the inherent disadvantages that may exist in either of these benchmark protocols and improve upon their performance under mobile scenarios.

5.3.2 Effect of an Intelligent Caching Scheme on Network Performance

On-demand routing protocols for MANETS utilize route caching in different forms in order to reduce the routing overheads as well as to improve the route discovery latency. For route caches to be effective, they need to adapt to frequent topology changes. Cache staleness in DSR can significantly degrade its performance. GPEAR employs an intelligent caching scheme to supplement the network's performance. In order to study the effects of caching policy on the network behavior, the caching scheme implemented by DSR was modified to support intelligent addition and replacement of routes (Chap3).

The metrics that were used for evaluating the effect of the caching policy were: ***Packet delivery ratio, Path optimality, Cache size and Simulation Time Variations.*** Simulation analysis was carried out for varied network scenarios. The network area covered was 1000 x 1000m with 100 nodes. The simulation was carried out for varied pause times. The pause time variation catered to varied network scenarios- completely stationary scenarios to highly mobile scenarios. The speed of the nodes was varied from 1m/s to 30m/s. The simulation was carried out for a total time of 3000 seconds. The traffic scenario used was same as the one employed for the AODV vs DSR simulations.

Packet Deliver Ratio (PDR)

Fig 5.24a shows the PDR as a function of simulation time. Three caching schemes were analyzed - (i) the cache scheme originally employed by DSR (ii) the caching scheme modified to include intelligent addition and (iii) the caching scheme modified to include both intelligent addition and replacement (LRU/LRO) policy [Chap 3.3].

Under low mobility conditions the caching policy used had little effect. The PDR gain due to intelligent addition (scheme-1) was just 0.002% and in case of intelligent add and replace (scheme-2) it was just 0.04%. As the node mobility increased, the effect on PDR was more pronounced. An expanded view of the PDR vs pause time in shown in figs 5.24b, 5.24c and 5.24d. The gain in PDR in case of scheme-1 increased from 0.02% in case of a stationary scenario to 4.5% in case of a completely mobile scenario (pause time – 0 seconds). In the case of scheme-2 the gain in PDR increased from 0.04% to 6.1%.

Hence as mobility increases the effect of the caching scheme on PDR increases. This pattern was repeated with different node speeds (20m/s: fig 5.25 a-d; 15m/s: fig 5.26 a-d,;10m/s: fig 5.27a-c; 5m/s: fig 5.28; 1m/s: fig 5.29) with the best results being obtained for a node speed of 15m/s. The gain in PDR as a function of pause time for different node speeds when cache scheme-1(i.e. intelligent addition of routes) is used, is plotted in fig 5.30. In case of low mobility scenarios there is a slight drop in PDR when scheme-1 is used. As mobility increases more routes are broken, added and replaced. The effect of the caching scheme hence is more pronounced. When nodes are stationary, cache activity is limited. From the earlier analysis carried out on DSR, it takes 3000 seconds for the cache to be completely filled in the case of stationary scenarios. Consequently the advantages of the caching scheme are minimal. In case of scheme-2 the gain in PDR vs pause time for varying speeds is plotted in fig 5.31. In case of scheme-2 as well, the best results were obtained for nodes with high mobility. The gain in PDR was the highest at a node speed of 15m/s for both cache schemes

1 and 2. As the node speed drops, the effect of caching scheme on the PDR also drops. This is due to two reasons: (1) as mobility drops the number of changes in the cache contents also drops. As a consequence, the effect of caching scheme is minimal at low node speeds. (2) The PDR of DSR is already high with very little scope for improvement in terms of PDR.

The simulation results show that a good caching scheme can improve the PDR by about 6% in case of high mobility scenarios. This underlines the importance of a good caching scheme to improve the network performance.

Path Optimality

Most protocols use minimum number of forwarding hops between source and destination i.e., the shortest route as the route selection criterion, and the shortest route is considered as the optimal route. In the absence of congestion or other "noise," path optimality measures the ability of the routing protocol to efficiently use network resources by selecting the shortest path from a source to a destination. It is calculated as the difference between the shortest path found internally by the simulator, when the packet was originated, and the number of hops the packet actually traversed to reach its destination.

The percentage of packets sent non-optimally is plotted as a function of pause time for different node speeds ((figs 5.33 through fig 5.38). All the figures indicate that the caching scheme affects the route that a packet takes enormously. There is a 10-15 % increase in the number of packets that are sent through non-optimal (longer) routes irrespective of node mobility or speed in case of the intelligent cache scheme.

Any route obtained by a node is recorded in the cache (primary or secondary) only if it is maximally disjoint to the other routes that are available in the cache. Due to this restriction in caching of routes, there is a possibility that shorter routes are lost. Despite longer routes being used by the packets the delivery ratio does not drop. On the contrary

there is an increase in PDR. This is possible only if the routes that are dropped are the ones that hardly contribute to the network performance. Another possibility that emerges from these results is that: ***the shortest route need not always be the best possible route for the highest PDR.***

Cache Size

The cache statistics are shown in fig5.39 to fig 5.46. The cache statistics have been plotted for a particular scenario for varying pause times. The node speed for which the cache statistics has been plotted is 30m/s. The network behavioral pattern is similar for other scenarios as well as node speeds. The primary and secondary cache size is defined in the protocol in terms of routes that can be recorded. The maximum number of primary and secondary route entries can be 100 in case of each cache.

In case of complete mobility (0-pause time) the primary cache and secondary cache is nearly full. In case of DSR an aggressive caching policy is followed. As there is no restriction on the type or the number of routes per destination cached, the cache is nearly full. In case of intelligent caching, though the policy still continues to be aggressive, only disjoint routes are cached. As a result fewer routes are cached. This, however, does not affect the PDR, which in fact improves. ***Therefore the routes that are not recorded are not the ones that contribute significantly to the network performance.*** In the case of some of nodes (especially source nodes) the number of routes recorded by the intelligent caching scheme is more than DSR (such cases are shaded in the figure).

This pattern can be observed in case of all pause times, except in case of stationary scenario. In case of a stationary scenario (pause time-3000) there is no mobility, hence very few route changes.

The cache size in terms of bytes has been shown for 0-pause time in fig 9.40. Due to the judicious use of the address table (sec 3.4) **GPEAR's memory requirements are approximately ¼ th of DSR's.**

Simulation Time Variations

The gain in PDR also varies as a function of simulation time. The PDR as the function of simulation time is plotted in figs5.47 through 5.52, for a complete mobility (pause time-0). The gain in PDR, which is around 1.5 % at 500 seconds, increases to 7 % at 2000 seconds, for a node speed of 30m/s, after which the gain in PDR stabilizes. This is consistent with the simulation time analysis discussed in the previous section.

This pattern is consistent for various node speeds and the gain in PDR stabilizes at its maximum value at around 2000 seconds. This is an indication that once the effect of the cache and its contents stabilizes the gain stabilizes. ***Hence as the cache stabilizes, the network behavior irrespective of the caching scheme, also stabilizes.***

5.3.3 Effect of Routing Criteria on Network Performance [46]

The two most popular On-demand protocols are AODV and DSR. Both these protocols use **Minimum Number of Hops** as routing criteria. Traditionally the shortest path to the destination has been used as a routing criterion, whether it is stationary networks or MANETS. The question is whether the shortest route is the best possible routing criteria for MANETS?

In case of MANETS the following factors have to be have to contended with:

(a) Randomly changing network Topology (b) Non-Uniform Traffic (c) Random Movements of nodes. So minimum number of Hops as a routing criterion may not be suitable for all possible traffic and mobility patterns. All the nodes in a MANET are wireless. So density of

a network also is an important factor in deciding the routing criteria. Hence a detailed analysis of the network performance for different traffic and mobility scenarios with varying routing criteria was also carried out.

While most of the protocols proposed for MANETS use the traditional minimum hop routing criterion, a few of them use different routing criteria. For e.g. (i) Dynamic Load Aware Routing Protocol (DLAR) [47] uses dynamic traffic load on a node. (ii) Signal Stability Analysis Protocol (SSA) uses relative route stability [18]. (iii) Power Aware Routing Optimization (PARO) uses distance between neighbors, so as to reduce power consumption at a node [21]. It has been observed that all these protocols work well for some network scenarios, but for not all scenarios. In order to discover the ideal routing criterion DSR has been used as the benchmark protocol, and the routing criterion was varied to study the network performance.

The four variations introduced in DSR were:

1.***DSR*** – *DSR* with minimum number of hops as routing criteria (the original version).

2.***SRET1***- *DSR* with **Route Expiry Time** as the routing criteria. The Route Expiry Time (RET) used here is static route expiry time, based on distance between nodes. If there exists a route A-B-C-D-E; the LET of A-B is obtained as the fraction of distance between nodes A and B and the estimated average speed of movement of a node A and B. A and B are here assumed, to be moving in opposite directions. As usual **RET is the minimum Link Expiry Time**. This form of RET calculation is termed as Static RET (SRET) as the strength of a link is decided based on only its initial position and estimated speed and direction information. The actual speed of movement of nodes or their direction of movement is not known. When the estimated Static RET time becomes lesser than time $\mathbf{T_{opt}}$, the route discovery process is triggered all over again. T_{opt} is calculated using average route set-up time.

3.*SRET2-* DSR with a combination of static RET and minimum number of hops as the routing criteria. When the estimated SRET time of the route becomes lesser than T_{opt}, then route discovery process to the destination is initiated.

4.*DRET*- DSR with Dynamic RET as the routing criteria. The Dynamic RET of a path is obtained by constant update of position information along a path. To achieve this Route Update (RUPDATE) messages were introduced into DSR. ***In case of Dynamic RET as in case of Static RET, a route is repaired in anticipation of a link break.*** Dynamic RET gives a better estimate of network dynamics, since it is obtained using real-time information.

The metrics that were used for evaluating the effect of the caching policy were: ***Packet delivery ratio, Path optimality and Control Overhead.*** Simulation analysis was carried out for varied network scenarios. The network area covered was 1000 x 1000m with 100 nodes and the simulation was carried for varied pause times. The pause time was varied to simulate completely stationary to highly mobile scenarios. The speed of the nodes was varied from 1m/s to 30m/s. The simulation was carried out for a total of 3000 seconds. The traffic scenario used was same as the one used for the AODV vs DSR and the caching policy simulations.

Packet Delivery Ratio

Figs 5.53 – 5.58 show the plots of the PDR analysis. The results indicate that the normal DSR protocol (fig 5.53) was unable to converge in case of high node mobility (0, or 50 pause times). In case of SRET1 where static RET was used as a routing criteria, the gain dropped from +2% at complete mobility to – 2% in case of a stationary scenario. In case of SRET2 there is a slight improvement in performance when compared to SRET. Since equal weightage has been attributed to minimum number of hops and SRET as a routing criterion, the loss in case of stationary scenarios is lesser as compared to the SRET1 version of DSR.

The best performance in case of high mobility scenarios is achieved by DRET, but in case of low mobility or stationary scenarios the performance of DSR appeared to be the best.

In case of node speeds of 30m/s the gain in PDR shown by DRET was 9%. As the node speed was decreased, the gain in PDR as compared to SRET1 increased to around 13% at node speeds of 15m/s (fig 5.55) and 10m/s (fig 5.56). At lower speeds, the gain dropped to around 9 % for a node speed of 5m/s(fig 5.57) and to 2% at 1m/s(fig 5.58). The drop in performance does not mean that DRET is not an effective Routing criterion at low node speeds. The PDR in fact is as high as 98% and the low gain is due to point of saturation being reached, for packet delivery ratio.

At low mobility scenarios ***'minimum number of hops'*** emerges as the best possible routing criterion and at high mobility conditions ***'dynamic RET'*** emerges as the best possible routing criterion. Static RET when used as a routing criterion does produce some gain at high mobility conditions, but as the nodes move, since RET information is not updated the information becomes stale. As a result, after the first major node movement, static RET no longer affects the PDR positively.

Path Optimality

Figs 5.59 - 5.64 show the percentage of packets sent non-optimally is plotted as a function of pause time for different node speeds. DSR uses minimum number of hops as the routing criterion. Hence it was able to pick up the optimal route in most cases. In case of SRET the RET is taken as a routing criteria and the RET in turn depends on distance between two nodes. Therefore the longer route is invariably picked (for e.g. if the routes available from node A to node D are A-C-D and A-B-C-D, A-B-C-D, is picked, since distance between adjacent nodes will be lesser when compared to route A-B-C). Therefore SRET1 invariably picks up a longer route.

SRET2 uses a combination of SRET and minimum number of hops as a routing criterion hence as far as optimality is considered the performance of SRET2 is better than SRET1 though lesser than DSR.

DRET constantly updates RET information and uses it as the routing criterion. Though DRET path optimality performance is better than that of SRET1, where minimum number of hops is not used as a criterion, it does not select out the shortest route.

Control Overhead

Figs 5.65 – 5.70 show the plots for the control overhead (in terms of number of packets) as a function of pause time for various node speeds. The control overhead in case of high mobility scenarios follows a similar pattern. The control overhead increases with pause time, as well as, node speed. The qualitative trend, in terms of difference in control overhead between various versions of the protocol remains the same.

The control overhead is lesser at lower mobility scenarios in case of DSR but higher as compared to SRET1 and SRET2 in case of high mobility scenarios. DSR has a lower PDR as compared to SRET1 and SRET2 in case of high mobility, since more packets are dropped, the number of RERR packets broadcasted increases.

Position information is used for routing by GPEAR. Addition of position information increases the size of the control packets. The question is, does this affect the PDR? The addition of position information does not affect the control overhead plot as the control overhead is shown in terms of number of packets. But the size of the control packet does affect the PDR albeit to a small extent as shown in fig 5.71. The loss in PDR is more pronounced at lower mobility but lesser than 1%. The effect has been shown only for a node speed of 30m/s. In case of other node speeds the effect was found to be insignificant. The position information was added only to RREQ and RREP packets. The loss is due to the

larger effect these packets have on broadcast storms and packet clashes, due to their increased size.

The control overhead for low mobility scenarios, in case of SRET1 and SRET2 is high due to drop in PDR, which causes more RERR packets to be broadcasted.

In case of DRET the control overhead is large since regular RUPDATE packets are transmitted, but the increase in control overhead is to some extent compensated by decrease in RERR packets.

In case of highly mobile scenarios it was found that a higher weightage attached to Dynamic RET gives a better packet delivery ratio, while Minimum number of hops seems to be the best routing criteria when the nodes are stationary. **In case of highly mobile scenarios the shortest route is not always the best route, but in case of least mobile scenarios the shortest route emerges as the best routing criterion.**

From the results, it can be inferred that if the mobility in a network can be estimated in advance and weightage attached to each criterion varied accordingly, the performance of the routing protocol can be improved. Varying ***weightage on routing criterion with varying mobility is implemented in GPEAR and consequent improvements in PDR have been observed..***

5.3.4 GPEAR Performance

The primary objective of these simulations is to show that GPEAR is able to discover routes between a source and destination whenever needed, and to maintain these routes as long as they are required. GPEAR should accomplish this with minimum control overhead and minimum consumption of a node's resources. Additionally it is important that GPEAR is able to pick the best available route.

To demonstrate these characteristics of GPEAR, different network configurations of 100 nodes with varying mobility levels and traffic were simulated. **Six different mobility levels were chosen to study the performance of GPEAR: 1.0m/s, 5.0m/s, 10.0 m/s, 15.0 m/s 20.0m/s and 30.0 m/s. The movement pattern was also varied, by varying the pause time. Seven different pause times were simulated: 3000 seconds (to determine GPEAR's performance in a static network), 1000 seconds, 500 seconds, 250 seconds, 100 seconds, 50 seconds and 0 seconds. Each pause time/mobility speed combination was run for ten different randomly generated initial node configurations. To investigate the performance of GPEAR's further, each of the simulations was run for four different traffic levels: 15 connections, 25 connections (under varying traffic densities), 35 connections and 50 connections.** Sessions commence at the beginning of the simulation and continue until 3000 seconds later, when the simulation ends.

The GPEAR performance analysis was carried out using DSR as the benchmark protocol. Both protocols maintain a *send buffer* of 64 packets. It buffers all data packets waiting for a route, e.g., packets for which route discovery has started, but no reply has been received as yet. To prevent buffering of packets indefinitely, packets are dropped if they wait in the send buffer for more than 30 seconds. All packets (both data and routing) sent by the routing layer are queued at the *interface queue* until the MAC layer can transmit them. The interface queue is FIFO, with a maximum size of 25. Routing packets were given higher priority than data packets in the interface queue.

Four key performance metrics were evaluated **(i) PDR (ii) Control Overhead (iii) Route length (route optimality) (iv) Energy Consumption.** These metrics are not completely independent of each other. For example high control overhead means a sharp drop in packet delivery ratio.

Packet Delivery Ratio

Figures 5.72 to 5.77 show the PDR as a function of pause time for varying mobility, with 25 active source-destination pairs and a traffic density of 4 packets per second. The plots show that GPEAR outperforms DSR irrespective of the pause time or node mobility. The gain in PDR varies with pause time and mobility.

At low mobility of 1m/s, 5m/s and 10m/s the PDR obtained using GPEAR was greater than 98% even in the case of continuous mobility (0-pause time). It can be observed that as mobility increases the PDR drops. The PDR dropped to an all time low of around 72% at a node speed of 30m/s, but still GPEAR outperforms DSR which is unable to converge at high speeds, with PDR dropping to nearly 50% at a node speed of 30m/s.

This behavior of both the protocols repeated itself for varying traffic densities of 6packets/ second (fig 5.78-fig 5.83) and 8 packets/ second (fig 5.84-fig 5.89). The drop in PDR with increase in traffic density is more pronounced in GPEAR than DSR This is due to the higher control overhead in GPEAR as compared to DSR. When the data traffic increases, the number of packets that queue up increase, and data packets are dropped in favor of control packets. In spite of this GPEAR is able to achieve a PDR of nearly 69 % with a traffic density of 8packets/s at a node speed of 30m/s as against DSR whose PDR is below 50%. At lower speeds GPEAR continues to perform irrespective of the traffic density. Only at speeds greater than 15m/s (54 km/hr) does the PDR drop.

For further analysis of GPEAR the number of connections was varied keeping the traffic density constant at 4packets/ second (figs 5.90 –5.101). From the plots it can be concluded that as the number of connections increase, the network performance deteriorates both in case of GPEAR and DSR, though GPEAR performs better than DSR.

The load on the network is technically the same whether the traffic is 25 connections with a density of 8-packets/ s or 50 connections with a density of 4-packets/s [(i.e.) 800 kbps]. The performance of GPEAR is better at 25 connections with traffic of 8-packets per second. The reason is that with increasing number of connections, the control overhead required to establish as well as maintain these connections increases, and the PDR drops in the wake of the increasing control overhead.

A further set of experiments (Figs. 5.102 and 5.108) was done to demonstrate the effect of loading the network. The highest mobility pattern (i.e., zero pause time) was chosen so as to make the situation fairly challenging for the routing protocols. The 100-node model was used and the number of sources was kept fixed (either 25 or 50 sources). Packet rate was slowly increased until the throughput saturated. The throughput here represents the combined "received" throughput at the destinations of the data sources. The "offered load" in the performance plots indicates the combined sending rate of all data sources. Without any retransmission, the ratio of throughput and offered load is simply the packet delivery ratio. Here, the units were chosen to be Kbits/sec (instead of packets/sec) indicating the network capacity that is being used. The loads at which saturation occurs have been circled in the plot.

At low speeds as the load increases the throughput increases. In case of DSR around 900 kbps the rate of increase in throughput drops and the throughput tends to saturate. At a speed of 15m/s DSR saturates at 1100kbps and at higher speeds of 20m/s and 30m/s DSR saturates at 900kbps itself. This is due to a bad packet delivery ratio (fig 5.110). In case of GPEAR, the throughput increases as the load increases and there is no tendency to saturate, though saturation may occur at higher bit rates (simulation was discontinued at 1200 kbps due to an overflow in system resources). GPEAR does not saturate like DSR because GPEAR's control overhead remains comparatively constant, even with increase in traffic.

The qualitative scenario is similar (fig.5.108 and fig 5.109) at 50 connections, but the quantitative picture is very different. Both GPEAR and DSR now saturate much earlier, though DSR saturates earlier than GPEAR. DSR saturates at 400kbps while GPEAR saturates at 1000kbps. It is observed that in case of both DSR and GPEAR the packet delivery ratio is low for 50 connections (fig 5.112 and fig 5.113). The result for 50 connections has been shown only for two speeds 15m/s and 30m/s, the results for other speeds are very similar.

In addition to the characteristic differences, the load tests in fig. 5.102 through fig. 5.109 show that network capacity is not utilized well by on-demand routing. Even the better performing protocol (GPEAR) saturates too early with increasing offered load. This is due to an upper bound on the capacity, assuming that each node is transmitting and is able to get a $1/(n+1)$ fraction of the nominal channel bandwidth, where n is the number of neighbors of the node in the adhoc network. This means that the delivered throughput to the application was at most about 2–5% of the network capacity. This figure may seem low, but is justified given that bandwidth consumed by the delivered data packets is in fact equal to delivered throughput times the average number of hops traversed. Besides the data packets that are dropped consume additional bandwidth, depending on the number of hops they traverse before being dropped. Also routing load consumes a significant portion of the bandwidth in addition to MAC control packets (e.g., RTS, CTS etc.). Finally RTS/CTS/Data/ACK exchanges for reliable delivery of unicast packets often slow down packet transmissions.

Control Overhead

The Control overhead, in terms of number of packets, versus pause time, as a function of node mobility, is shown in figs 5.114 through 5.119. The qualitative pattern of the control overhead varies with mobility. At low node mobility (1m/s or 5m/s) the control overhead is

higher in case of GPEAR as compared to DSR. This difference is more pronounced at higher pause times (stationary and nearly-stationary scenarios). Irrespective of the mobility, GPEAR generates RUPDATE packets along each route at a time interval of 3.0 – 4.0 seconds. These RUPDATE packets add to the control overhead. These RUPDATE packets are the reason that GPEAR is predictive in nature. Due to its predictive nature, GPEAR is able to reduce the control overhead by reducing the RERR packets that are generated as a result of link failure. Link failures are repaired in advance due to the predictive nature of GPEAR. Consequently RERR packets are reduced due to the introduction of RUPDATE messages. This effect can be observed clearly in figs 5.120 through 5.125. At lower mobility few route breaks occur, so fewer RERR are broadcasted. Even though the number of RERR packets generated by GPEAR is reduced, the number of RUPDATE messages generated increases the control overhead. ***Since GPEAR in spite of the high control overhead is able to produce a good PDR, it indicates that the control overhead is within acceptable limits.***

At higher node mobility [(i.e.) high node speed and low pause time], the number of control packets generated by GPEAR is lesser than DSR. The number of RUPDATE messages generated are almost constant (figs5.126 – fig 5.131). Hence as the mobility increases, more links break, resulting in more RERR packets being broadcasted along the network by DSR. GPEAR, by the virtue of its predictive nature, repairs the broken links in advance, thereby reducing the RERR messages; consequently reducing the control overhead.

Another interesting effect that was observed was that the control overhead in GPEAR is nearly constant when compared to DSR. It is the RUPDATE messages that contribute mainly to the control overhead. RERR messages are kept to a minimum by the RUPDATE messages. Hence the control overhead in case of GPEAR follows the pattern of the RUPDATE messages rather than the RERR messages. ***Due to low control overhead GPEAR***

outperforms DSR in terms of PDR at high mobility. It can be concluded that at high mobility, where the control overhead is large, broadcast storms are produced and these have a pronounced effect on the PDR. **GPEAR is able to contain the control overhead at high mobility, by reducing the RERR broadcast, and hence is able to avert a broadcast storm, leading to a good performance at high node mobility.**

This pattern can again be observed in figs 5.132-fig 5.135, where the number of control packets is plotted as a function of the offered load. As the load increases, the control overhead increases in DSR, whereas in a case of GPEAR the load has very little effect on the control overhead. The control overhead in GPEAR is nearly constant, as the number of RUPDATE messages remains constant. In a contest for airspace between data packets and control packets, control packets are given a higher priority, as the load increases, the probability of such a contest occurring increases. In case of GPEAR, since the control overhead is constant, the PDR remains nearly constant. As the load increases the throughput increases, and there is a lesser tendency to saturate. But in case of DSR as more packets are dropped, more RERR messages are generated, leading to a further drop in PDR. This effect mushrooms until DSR saturates and is unable to deliver packets any further.

As the number of connections is increased to 50, the control overhead increases and DSR saturates earlier (around 400kbps). GPEAR also saturates but at relatively higher loads (> 900kbps). At 50 connections, every node in the network becomes a part of the data traffic or control traffic. The buffers overflow, and more data packets are dropped. This increases the control traffic further until the network saturates.

Furthermore as GPEAR uses ring search, once a route is discovered at a particular TTL, further route discovery is terminated completely. But in case of DSR the route discovery process continues until MAX_TTL is reached, leading to more control overhead in

terms of RREQ and RREP. The still circulating RREQ and RREP messages not only add to the control overhead but also fill the cache of the nodes with routes that may never be used, as DSR uses minimum number of hops as the routing criteria.

Such caching provides a significant benefit up to a certain extent. With higher loads the extent of caching is deemed too large to benefit performance. When faced with multiple choices, often stale routes are chosen, as the route length is the only metric used. Picking stale routes causes two problems: (i) consumption of additional network bandwidth and interface queue slots even though the packet is eventually dropped or delayed (ii) possible pollution of caches in other nodes. When compared to GPEAR, a much smaller number of packets were dropped in DSR for lack of route availability (e.g., indicating high cache hit ratio). However, significantly more packets were dropped, as the interface queue was full.

The control packets are plotted in detail only for the traffic scenario having 25 connections, since for all other traffic scenarios, the results were found to be qualitatively similar.

Path Optimality

The percentage of packets sent non-optimally is plotted as a function of pause time for different node speeds from fig 5.134 through fig 5.141. Path optimality has been plotted in detail only for a traffic scenario of 25 connections, since the pattern remains the same for all other traffic scenarios. GPEAR uses a varied combination of factors such as, minimum number of hops, RET and time at which the route was cached, as a route selection criteria. The weightage attached to each of these factors varies with mobility as described in the previous section. ***Even though minimum number of hops is not taken as the only routing criterion, the percentage of packets sent non-optimally is not high since ring search is employed by GPEAR, which ensures that shorter routes are selected.***

Energy Consumed

GPEAR is energy aware, since each data packet is transmitted at different levels of energy based on the distance between two nodes. Control packets except for RUPDATE messages are sent at maximum possible energy. RUPDATE messages are sent at a higher level of energy as compared to data packets but continue to be energy aware. Each packet carries an energy stamp for the physical layer to determine the energy at which each packet should be transmitted. This helps in reducing the energy consumption in the network. Each node in the network is initially assigned an energy of 1000 Joules.

Depending on whether a node is transmitting or receiving packets or whether it is in an active or sleep state, different energy levels have been allotted, for each of the states.

Unlike the earlier plots, energy consumed in the network is plotted for varying traffic (fig 5.142-5.146). It was found that irrespective of the mobility in the network, the energy consumption pattern remains the same. Figs 5.142 to fig 5.144 shows the energy consumed as a function of pause time .The traffic density (number of packets/sec) has been varied to get different plots. All the plots show that the energy consumed for a particular traffic density remains nearly constant. As the mobility varies, the amount of routing information that travels along the network varies, but the size of the control packets as compared to the data packets is small. Hence the energy consumed for sending, forwarding or receiving control packets is considerably less than the energy required for sending data packets.

The PDR of GPEAR as well as the percentage of packets sent using longer routes is still higher when compared to DSR. This indicates that GPEAR delivers more number of data packets using longer routes. It appears that the energy consumed by GPEAR must be higher when compared to DSR. But due to the energy awareness scheme incorporated in the protocol energy consumed by GPEAR is lesser than that of DSR.

The same trend can be observed in case of figs 5.145 and 5.146, when the number of connections is varied. It can be seen that the energy varies with the number of connections. In case of 50 connections, the plots show that the nodes are nearly drained of energy when DSR is used as the routing protocol. This could be another reason for the low PDR in case of 50 connections.

Energy consumed is an important factor that affects the network performance. As the network is mobile, the nodes are entirely dependent on the battery power available. The battery power is also a factor that determines how long a node lives in the network. ***The nodes employing GPEAR still have considerable energy left at the end of the simulation, and hence are able to outlive their counterparts that employ DSR.***

5.3.5 Optimum Node Density and Scalability Analysis of GPEAR

In 1978 Klienrock and Silvester published their well-known paper "Optimum Transmission Radii for Packet Radio Networks [48]". The paper provides an analysis that explores the tradeoffs between increased transmission radius, resulting in fewer hops to reach a destination, and the effective bandwidth lost at each node as a result of the increase in transmission power. The paper shows that the optimum number of neighbors for a given node is 6 (actually 5.89), and concludes that a node's transmission radius should be adjusted so that it has not more than 6 neighbors.

While this result may be valid for stationary networks, it does not consider the ramifications of node movement on the optimum transmission power. As mobile networking becomes popular, it is important to understand the characteristics of this type of communication so that users can communicate in an optimal manner without wasting battery life or bandwidth.

though the effective bandwidth seen at individual nodes suffers due to the increased transmission power and collisions, the number of packets delivered still increases, relative to shorter transmission ranges. This is because link breaks are less frequent and routes are maintained for a longer period of time. This can be observed from the fact that the 1000x1000m network performance is better as compared to 1000x1500m network at higher mobility. The throughput suffers as a result of the increased number of hops to reach the destination. However, the simulations show that at low node densities, as in the case of 1500x1500m scenario, the network does not, in fact, remain completely connected. Numerous nodes or group of nodes become disconnected from the node majority. The result of the disconnected operation is that many of the sessions abort because routes to the destination are unavailable. As the node density increases, the number of packets delivered drops as can be observed in case of 500mx1000m network. This is due to the increased number of collisions, as well as reduced channel access, which leads to buffer overflow.

Control Overhead

As the node density drops, the number of control packets increase. This is due to increase in average path length with decrease in node density. The main control overhead as shown in the previous section is the RUPDATE messages. As the mean path length increases the number of RUPDATE messages propagating through the network increases raising the control overhead in the network. In case of low-density networks as route breaks, it is possible some nodes get isolated as a result RERR messages have to be propagated. The control overhead follows the expected pattern as shown in figs 5.150-fig 5.152. The quantitative pattern is similar at all speeds.

Path Optimality

The percentage of packets sent non-optimally is plotted as a function of pause time for different node speeds from fig 5.153 through fig 5.155. More number of packets are sent non-optimally in case of 1000x1000m and 1000x 1500m networks as compared to the 500x 1000m and 1500x1500m networks. At high node density as the neighbors per node is high, it is possible that multiple short routes to a destination exist, keeping the optimality low. In case of low-density scenarios the number of routes to a particular destination is very less. This is because of the availability of only a single route per destination, which is invariably the route that is picked.

Energy Consumed

Energy consumed is plotted in fig 5.156 for a node speed of 30m/s. Energy consumption remains constant irrespective of the speed. Energy consumption increases as the network area increases, since the mean path length increases with the network area. Consequently the overall energy consumed also increases. As the node density decreases, the distance between adjacent nodes increases as a result packets are transmitted at increasingly higher energy levels. Consequently energy consumed increases.

The plots and analysis show that though Klienrock and Silvester's calculations are valid to a certain extent in case of stationary networks. A higher node density produces better results as mobility increases. If node density is increased further, the network saturates leading to a drop in PDR. The ideal number of neighbors in case of a mobile network would vary between 10-15 depending upon the degree of mobility. As the mobility increases the ideal number of neighbors tend towards 15.

Figs 5.157 –5.160 show sample layouts of various network scenarios.

The node density results show that as the network area increases the PDR varies if the number of nodes is kept constant. In case of very low node density, as well as very high node density networks, the PDR drops. The same network scenarios can be used for studying the scalability of an adhoc network, by varying the number of nodes in the network. Other than the normal 1000x1000m network which has been analyzed in detail, two other network performances have been studied in terms of PDR: 1500x1500m, 500x1000m (the 1000x1500m network results are quite similar to that of 1000x1000m). In order to keep the number of neighbors constant around 15, the number of nodes in the 500x1000m network was decreased to 50 and the number of nodes in the 1500x1500m network was increased to 200.

To demonstrate scalability, PDR alone has been plotted. Both path optimality and control overhead continue to follow the same quantitative trend. As the mean path length and number of connection increases, the control overhead increases. Since the number of neighbors are maintained at approximately 15, the path optimality retains the qualitative and quantitative trend of the 1000x 1000m results. Since the average number of neighbors is same as the 1000x1000m network, it follows, that the number of alternate routes available will be almost the same. Since the routing criterion remains the same, path-optimality will also remain the same.

Packet Deliver Ratio

Figs 5.160-5.166 show the PDR for different speeds, as well as, traffic for the case of 50 nodes with a network area of 1000x1000m and for 200 nodes with a network area of 1000x1500m. The 100 node networks were analyzed earlier for 25 connections [(i.e.) $1/4^{th}$ of the total number of nodes in the network]. In order to maintain this ratio between traffic and number of nodes, 13 connections were setup for the 50-node network and 50 connections

were setup for the 200-node network. From Fig 5.160 it can be seen that the PDR is quite high for 13 connections in a 50-node network. In the earlier section simulation analysis was done for 15 connections in a 100 node network in an area of 1000x 1000m (fig 5.95 for 10m/s, fig 5.99 for 20m/s and fig 5.101 for 30m/s). It can be analyzed by comparing the plots that the performance 50-node 500x1000m is better than the 100-node 1000x1000m network. A similar trend has been observed for the case of 25 connections as well. When 25 connections is setup in a 50-node network, every node is part of the data traffic, either as a source or destination. Hence the load is high, but still the 50-node network performs better than the 100-node network.

As the network area is increased to 1500m x1500m, the performance of the network deteriorates, despite maintaining the node density constant. Not only does the 50 connections setup fail badly, even the 25-connection setup is not able to provide a PDR on par with the 1000m x 1000m network.

Hence it can be seen as the network area increases, the performance drops, despite the constant node density. A number of factors contribute to this downward trend in network performance. As the number of nodes in the network increase, a probability of longer routes increases, thereby increasing the possibility of route breakages. It is possible that a route may break even during the process of route formation. Longer routes cause the cache and various buffer structures to overflow. As GPEAR uses source routing, longer routes mean longer headers, which take up a large chunk of the available channel bandwidth.

As the network size increases the network performance drops. Hence in case of large networks, the network can be divided into smaller area zones of 50 or 100 nodes, and an inter-zonal routing protocol such as GIZR can be used for communication between various zones.

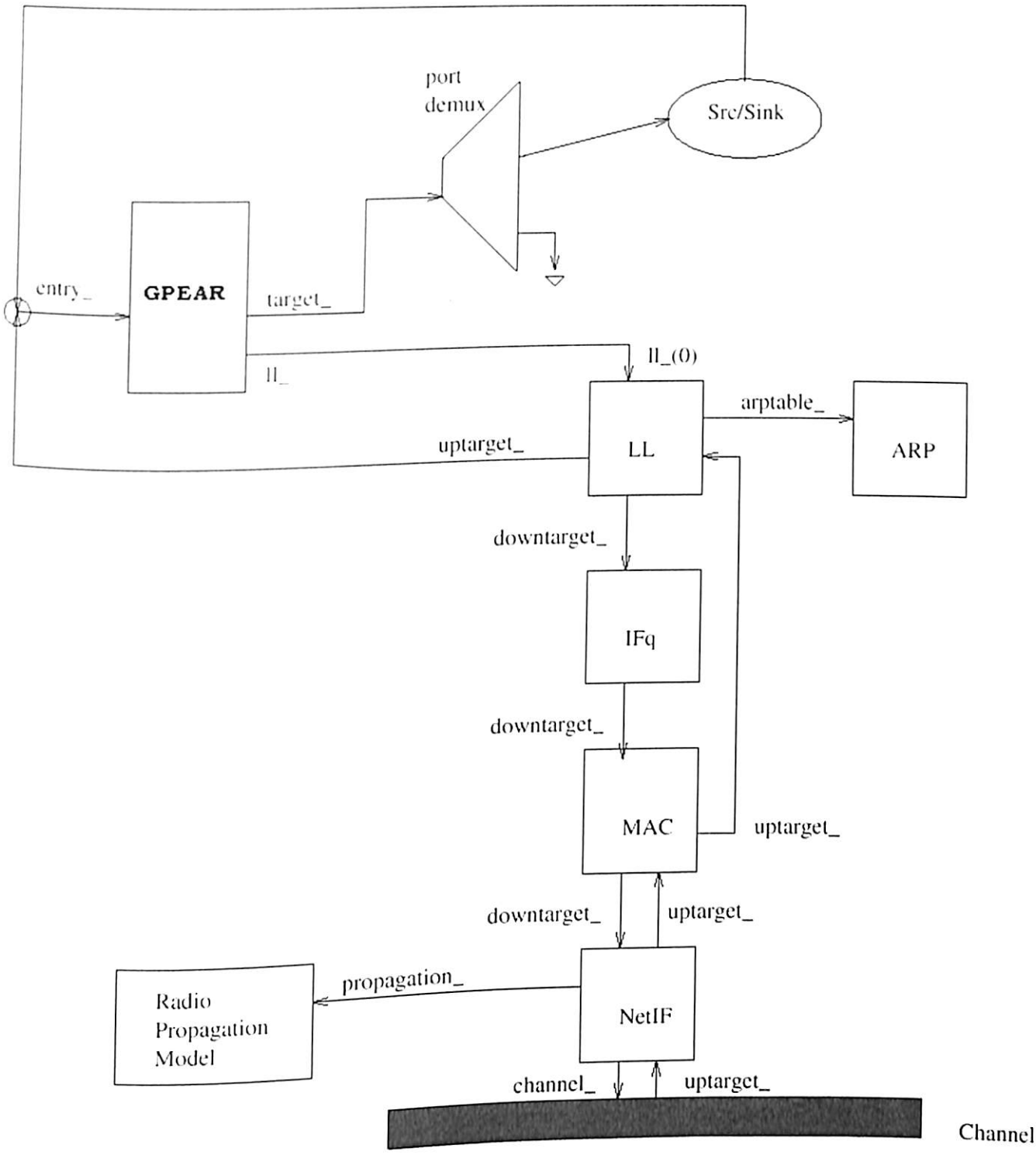


Fig 5.1 Schematic of a protocol stack in *ns2*

Mobility Pattern in ns-2 (fig. 5.2 - fig. 5.8)

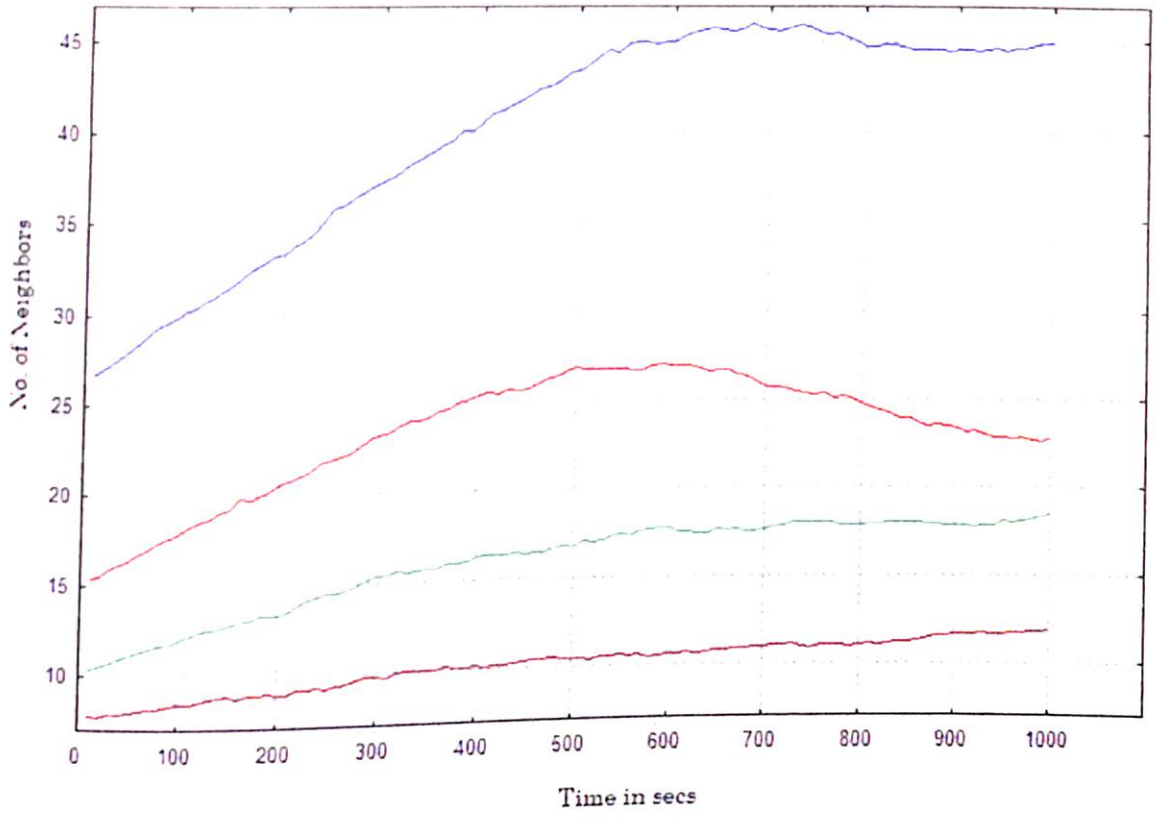


Fig 5.2 Average neighbors per node at 1m/s mobility

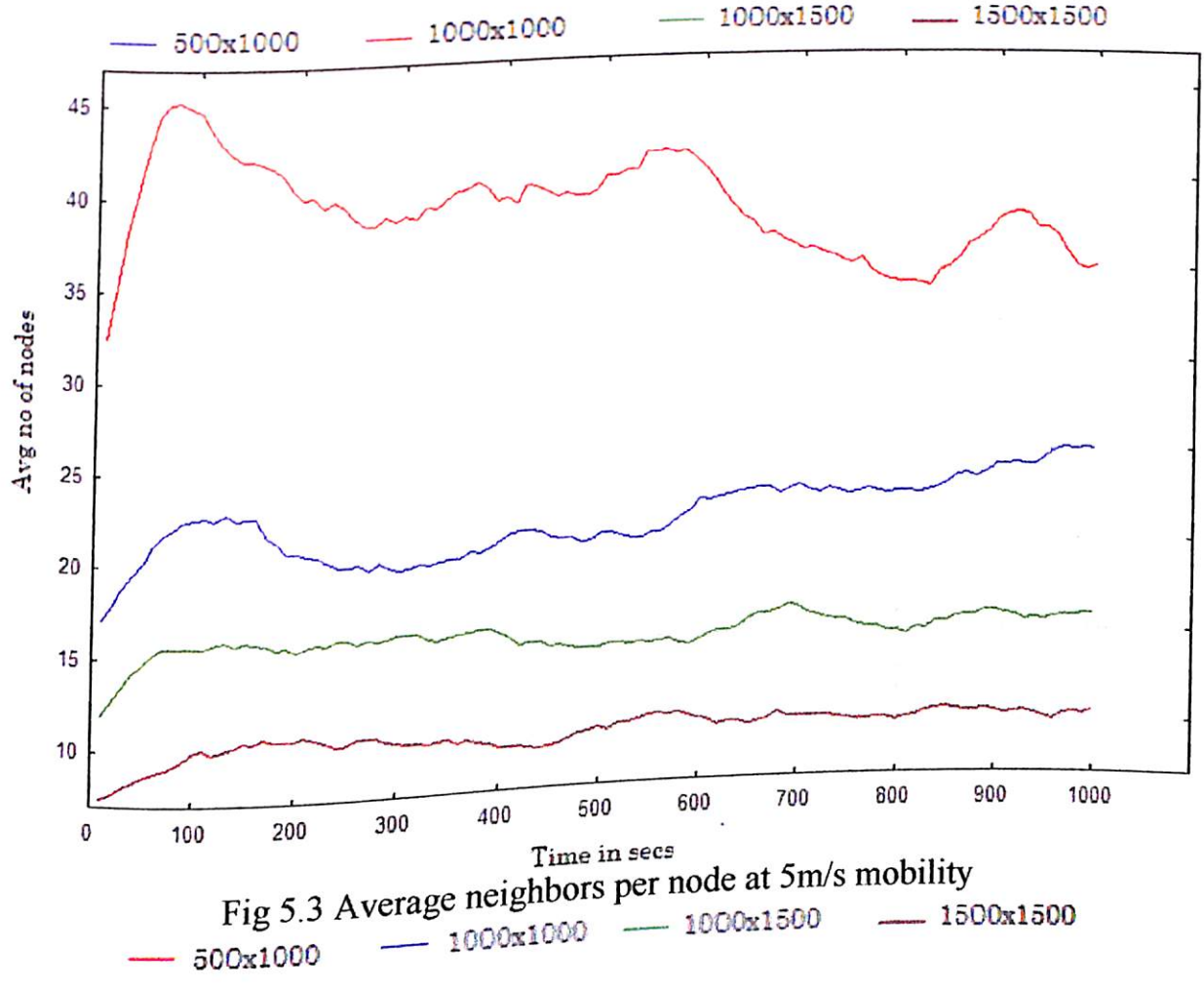


Fig 5.3 Average neighbors per node at 5m/s mobility

500x1000
1000x1000
1000x1500
1500x1500

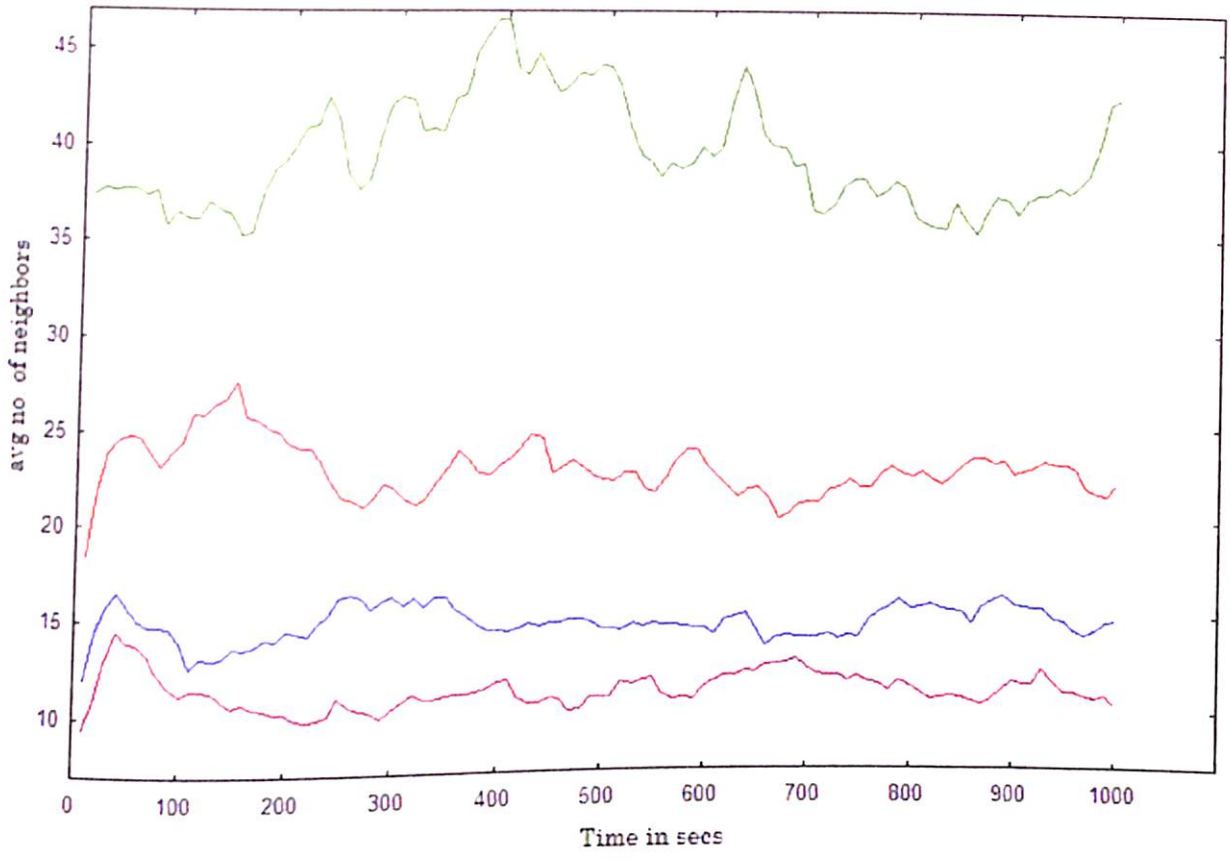


Fig 5.4 Average neighbors per node at 20m/s mobility

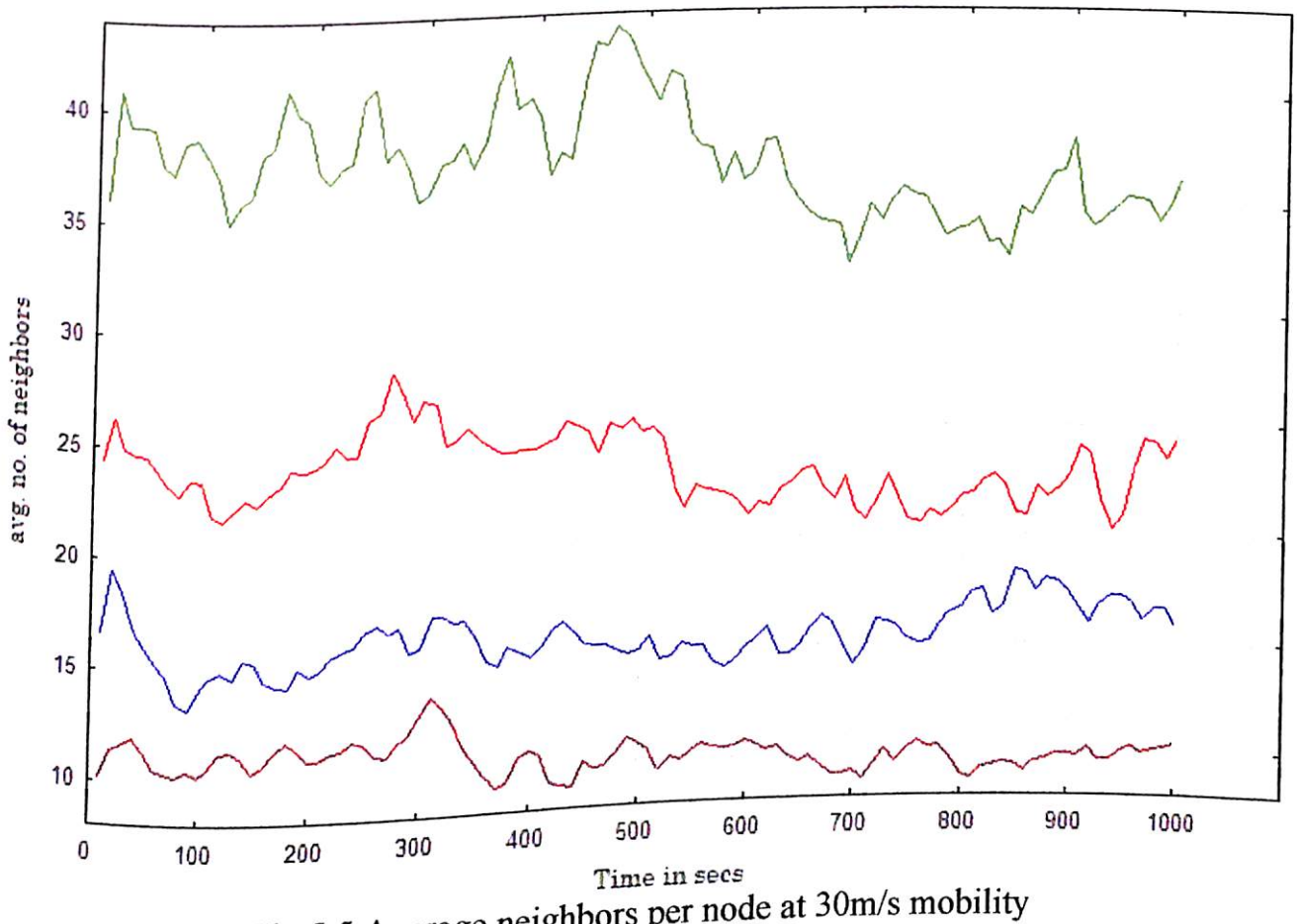


Fig 5.5 Average neighbors per node at 30m/s mobility

500x1000 1000x1000 1000x1500 1500x1500

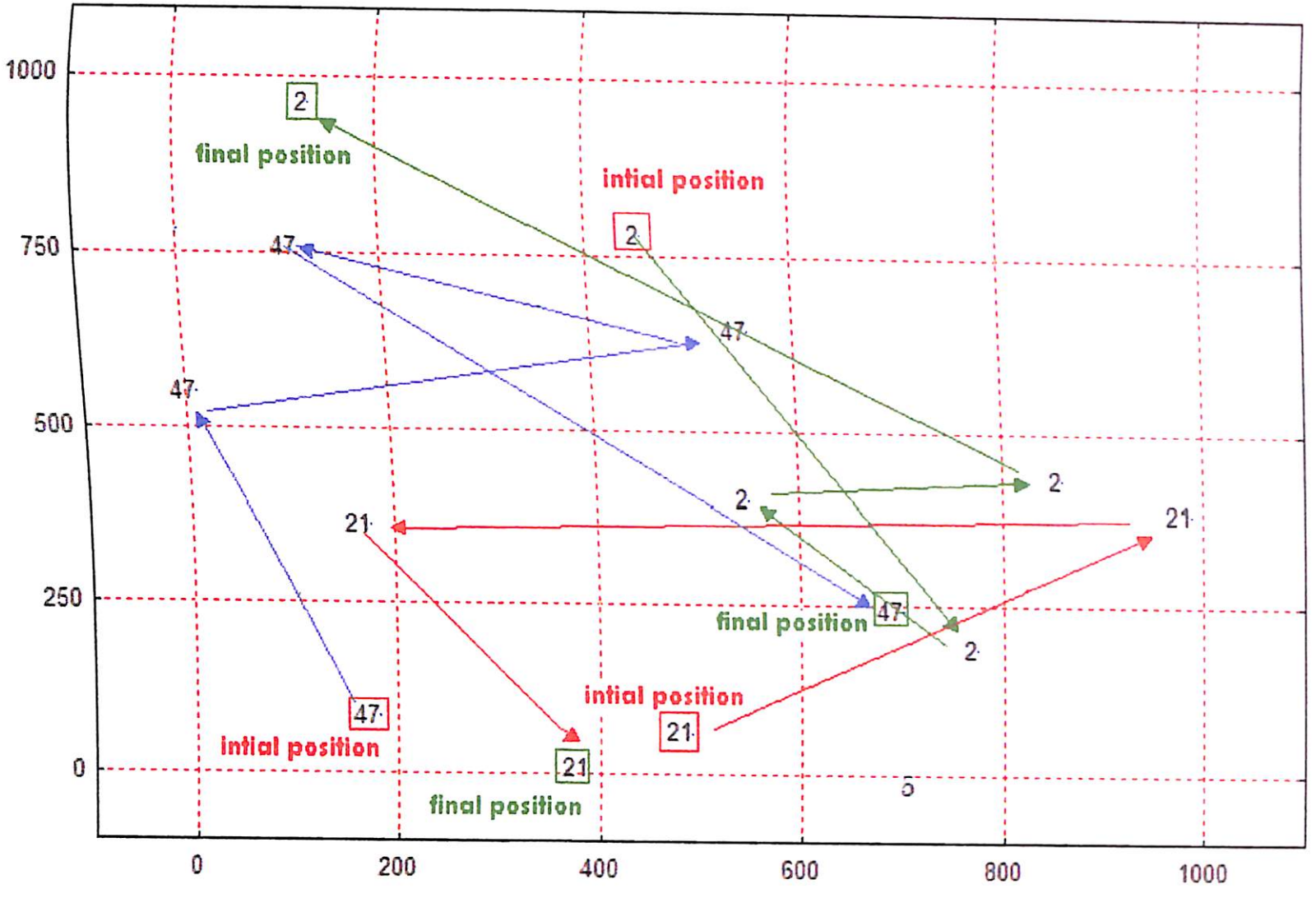


Fig 5.6 Movement Pattern of nodes 2 21 and 47 at 5m/s

Network Area 1000 m x 1000m

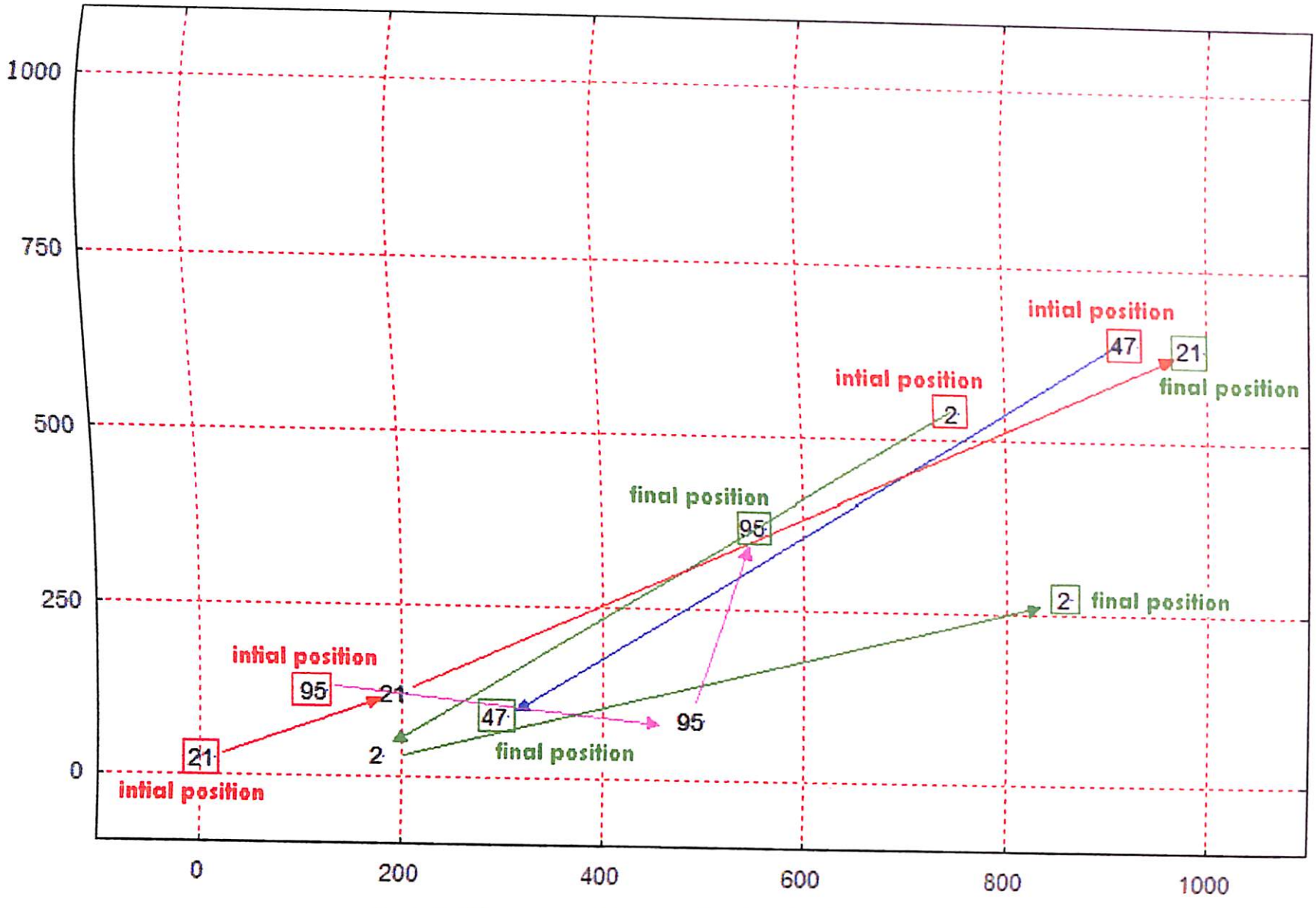


Fig 5.7 Movement Pattern of nodes 2 21 47 and 95 at 10m/s

Network Area 1000mx1000m

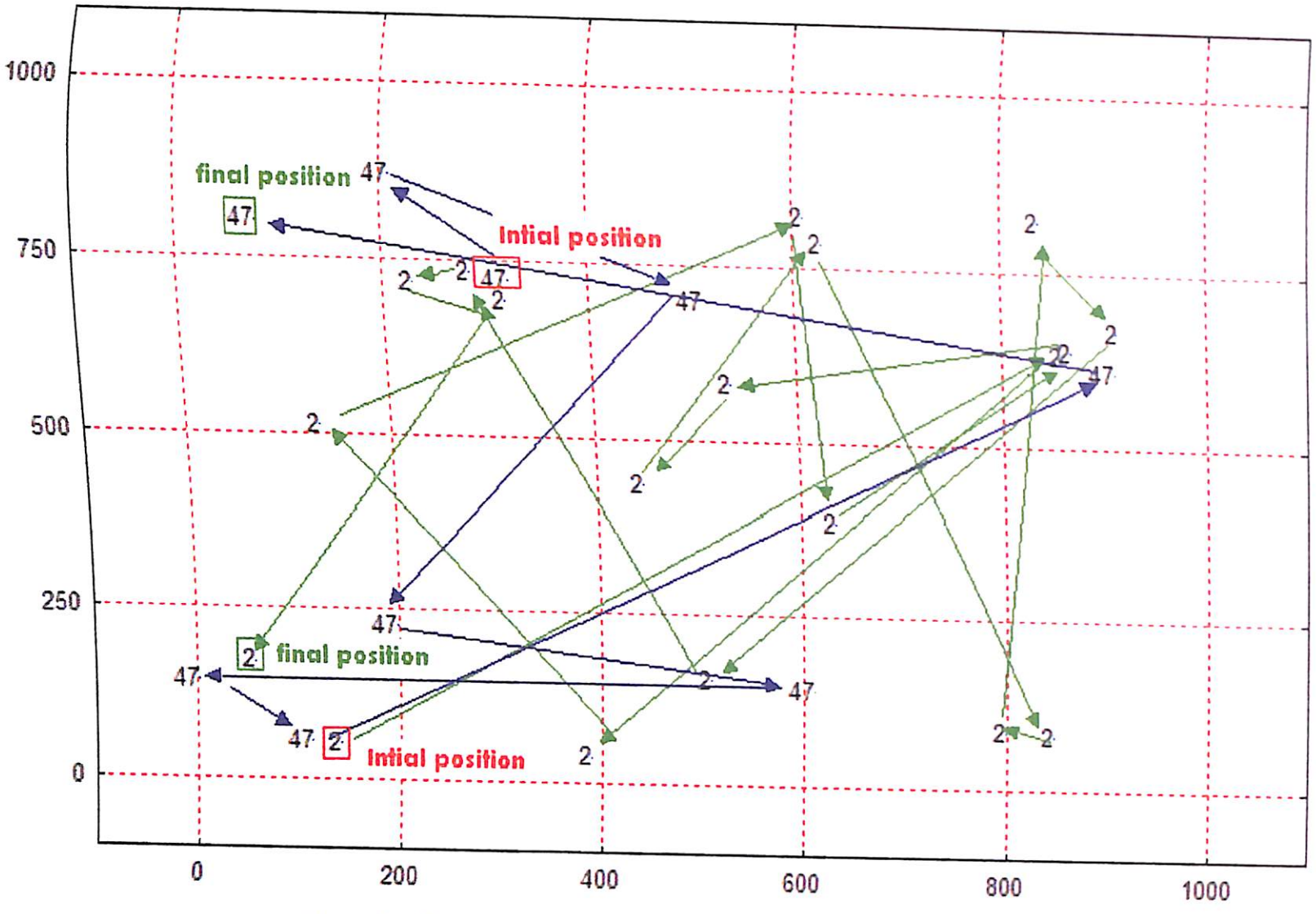


Fig 5.8 Movement pattern of nodes 2 and 47 at 20m/s

Network Area 1000m x 1000m

AODV Vs DSR (fig. 5.9 - fig. 5.23b)

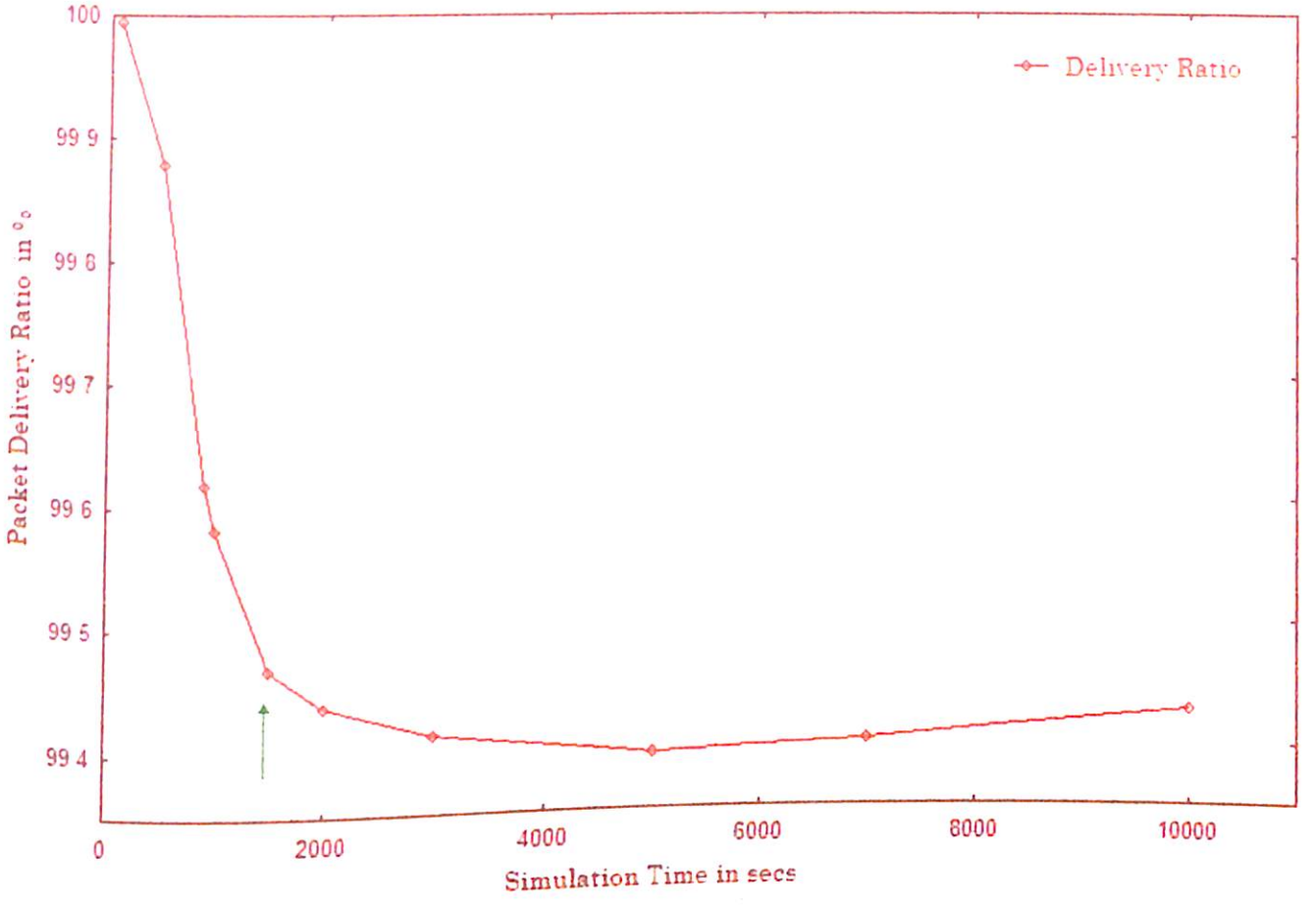


Fig 5.9 Packet Delivery Ratio Vs Simulation Time for DSR

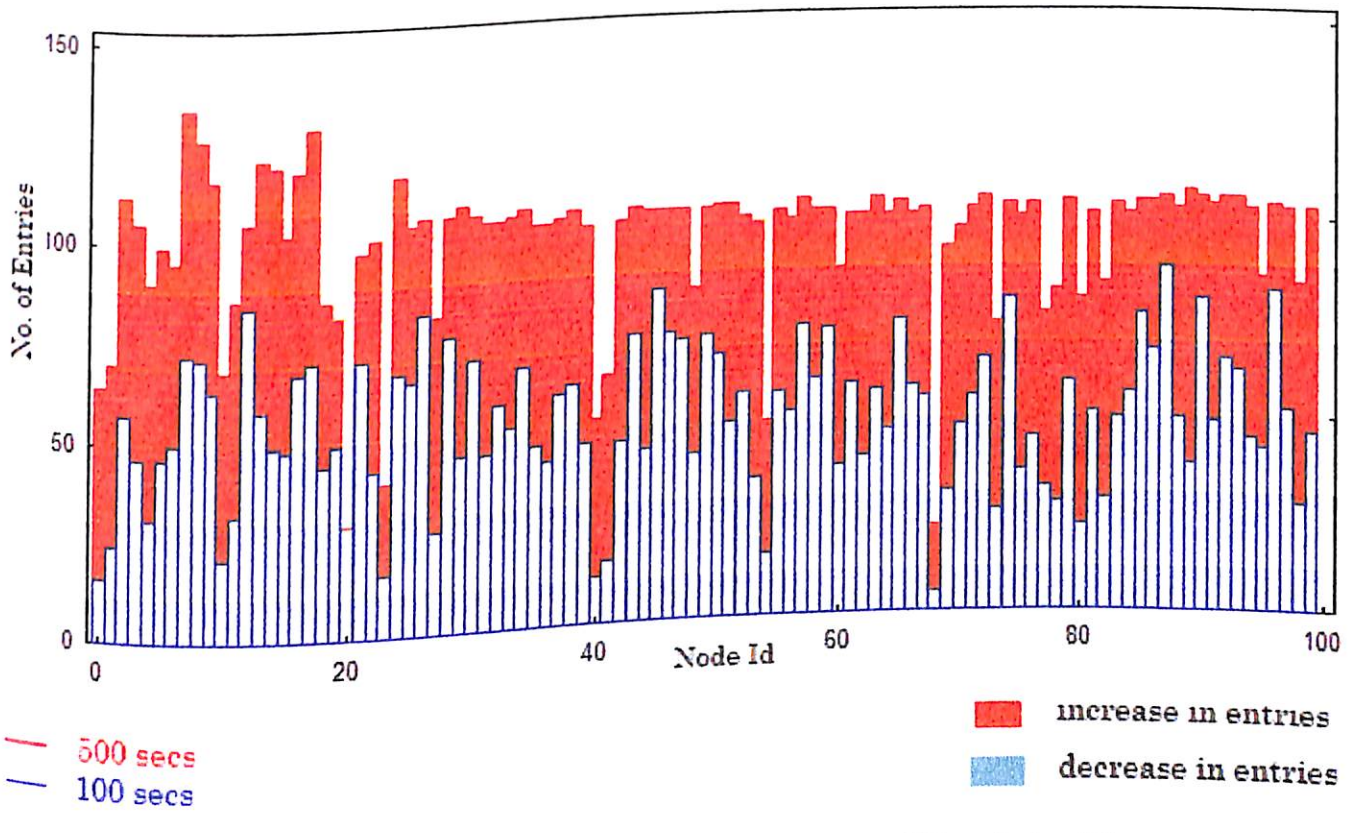


Fig 5.10 Cache Statistics at the end of 500 secs of Simulation Time

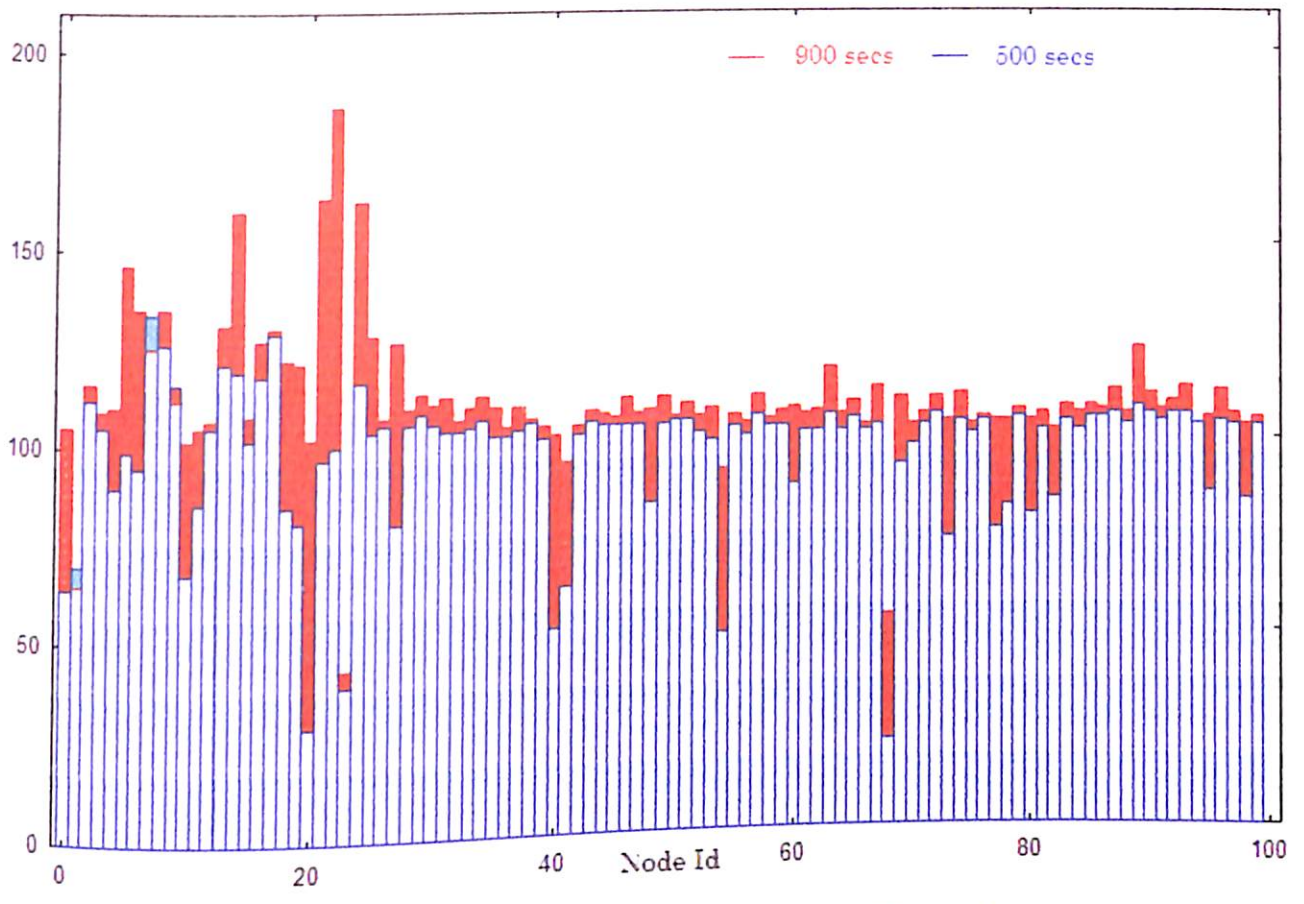


Fig 5.11 Cache Statistics at the end of 900 secs of Simulation Time

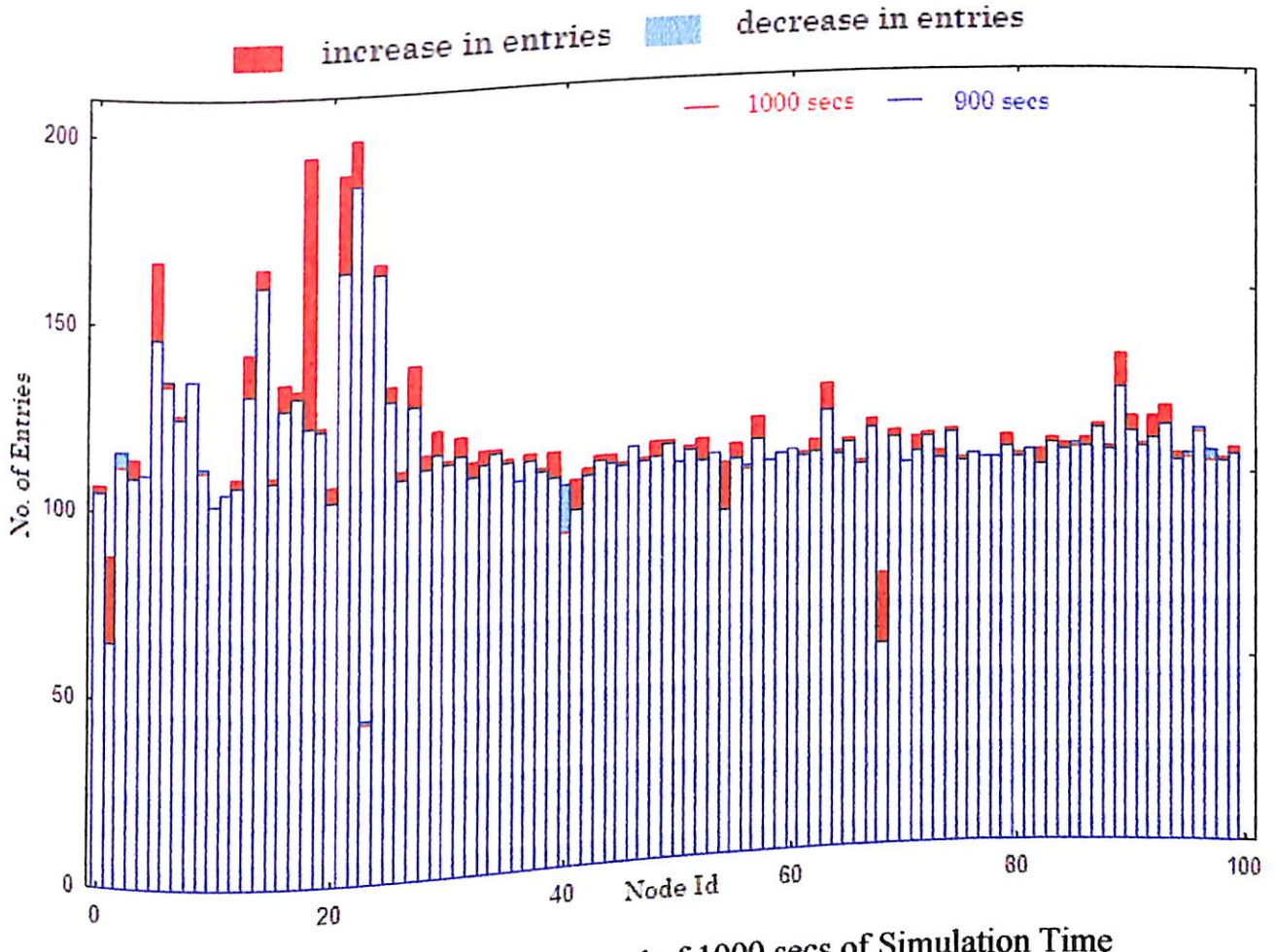


Fig 5.12 Cache Statistics at the end of 1000 secs of Simulation Time

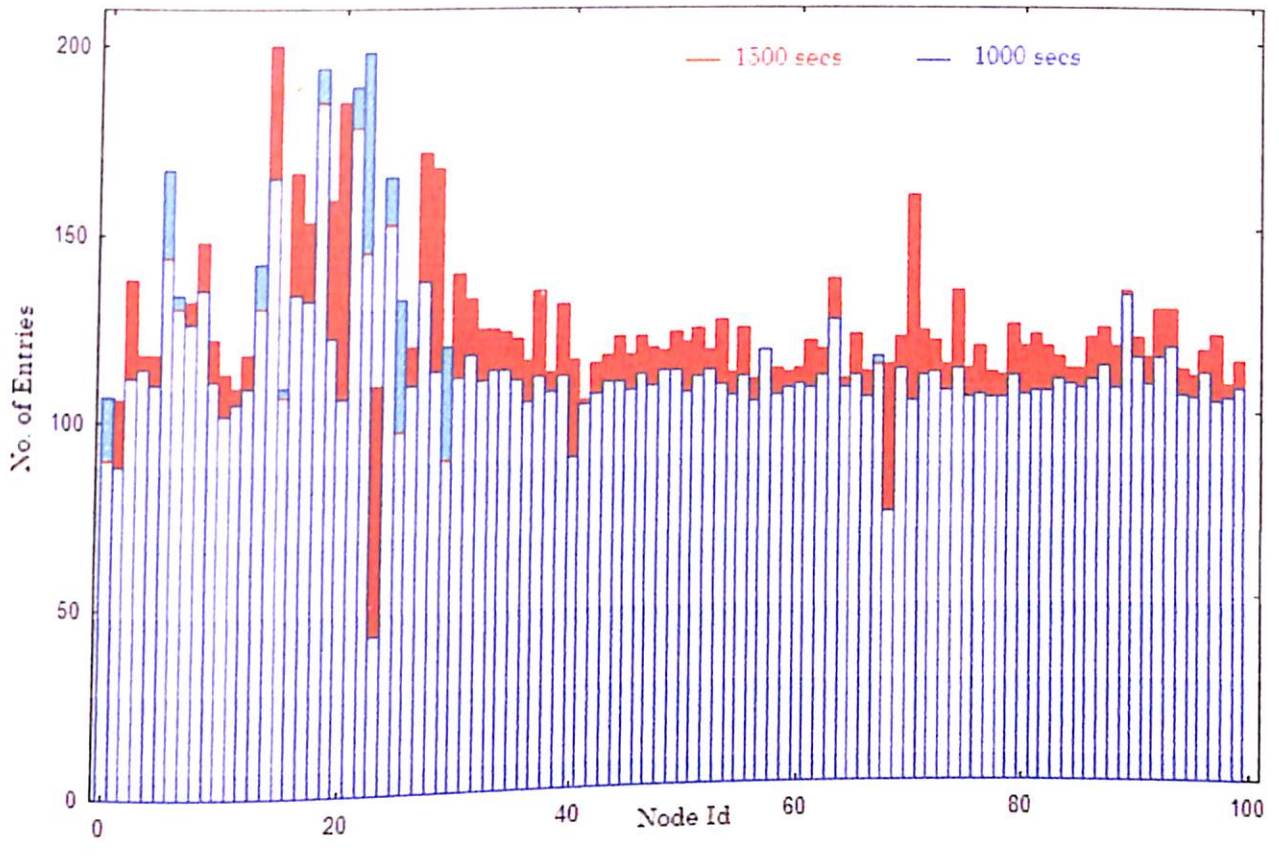


Fig 5.13 Cache Statistics at the end of 1500 secs of Simulation Time

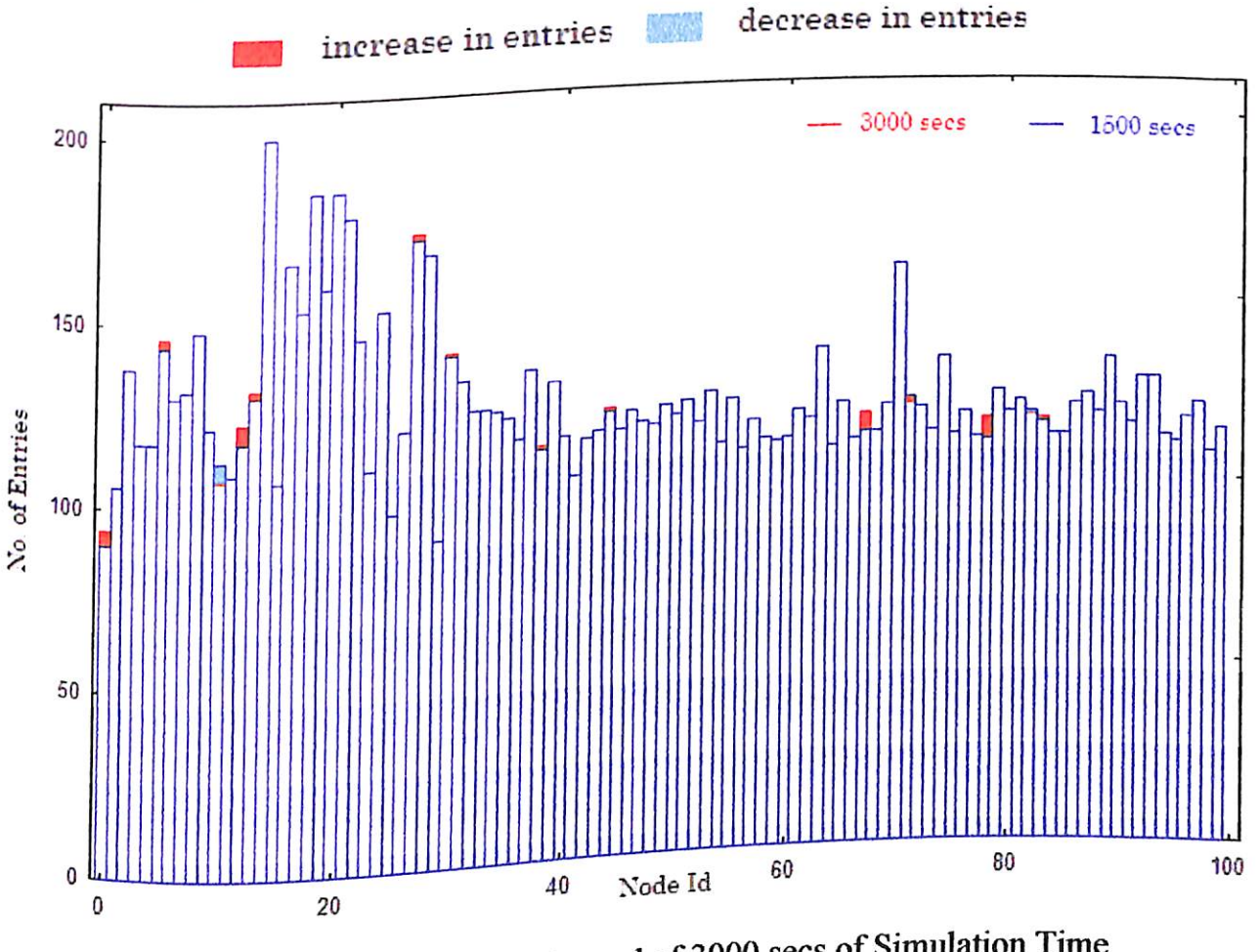


Fig 5.14 Cache Statistics at the end of 3000 secs of Simulation Time

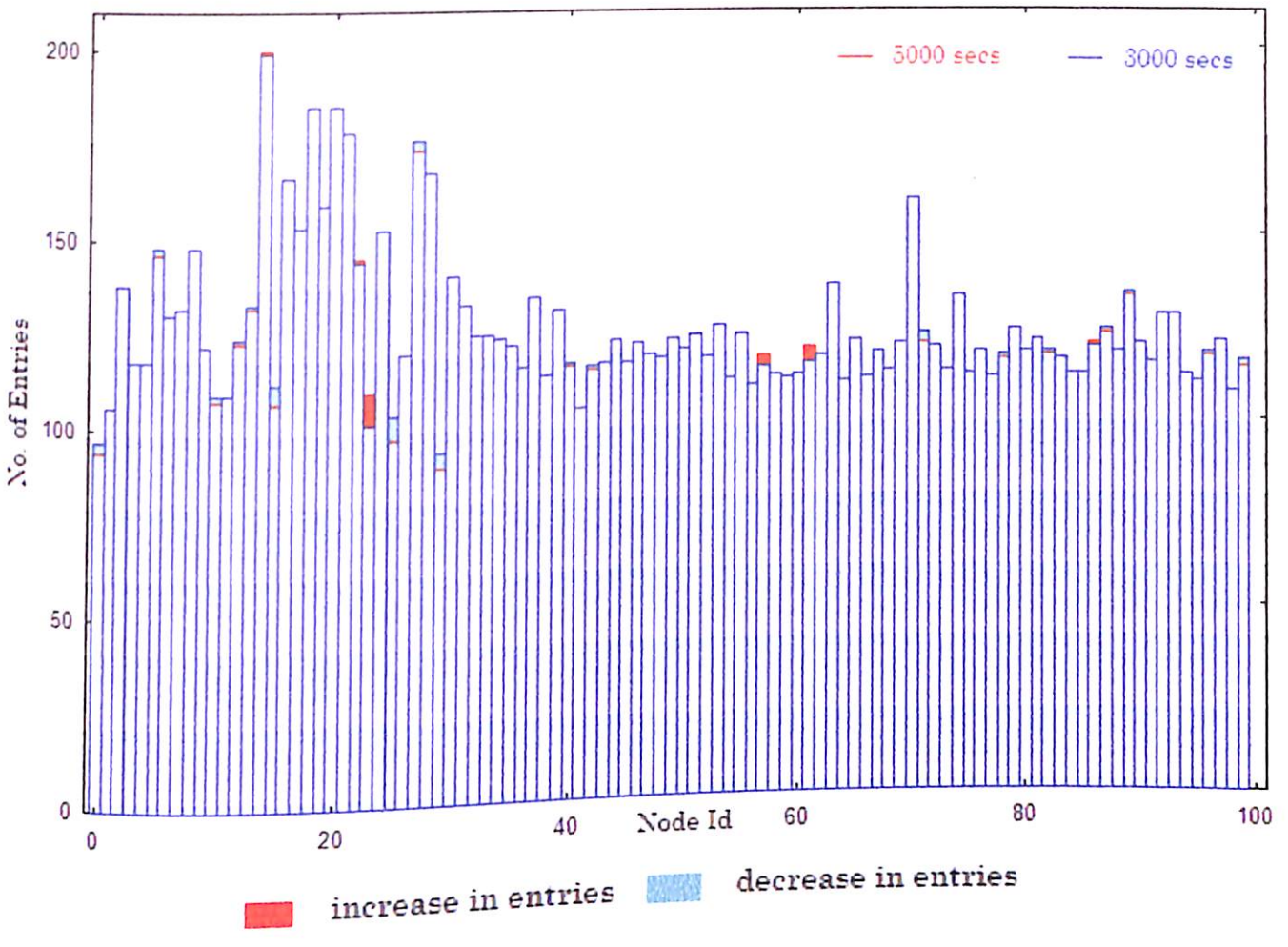


Fig 5.15 Cache Statistics at the end of 5000 secs of Simulation Time

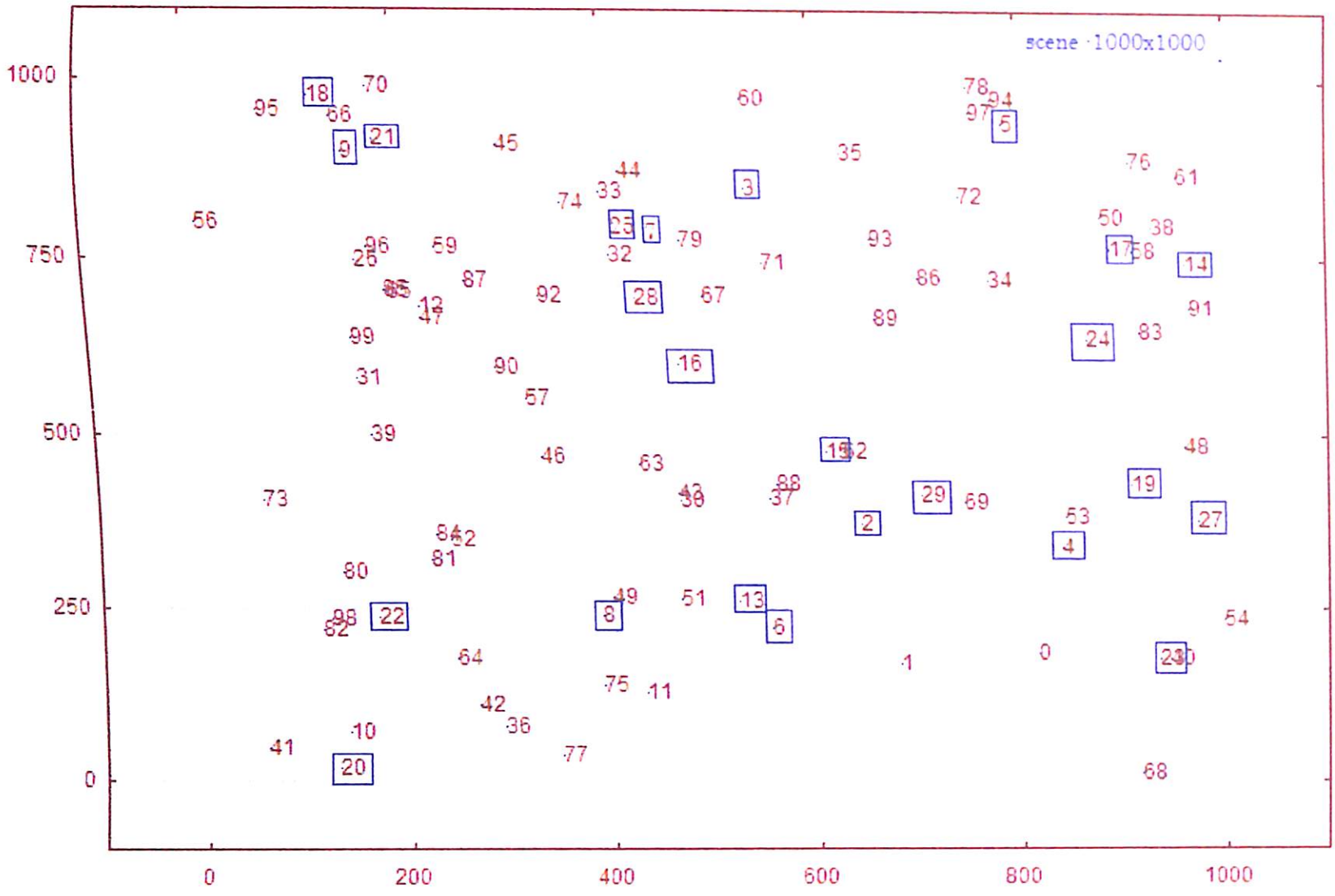


Fig 5.16 Network Scenario

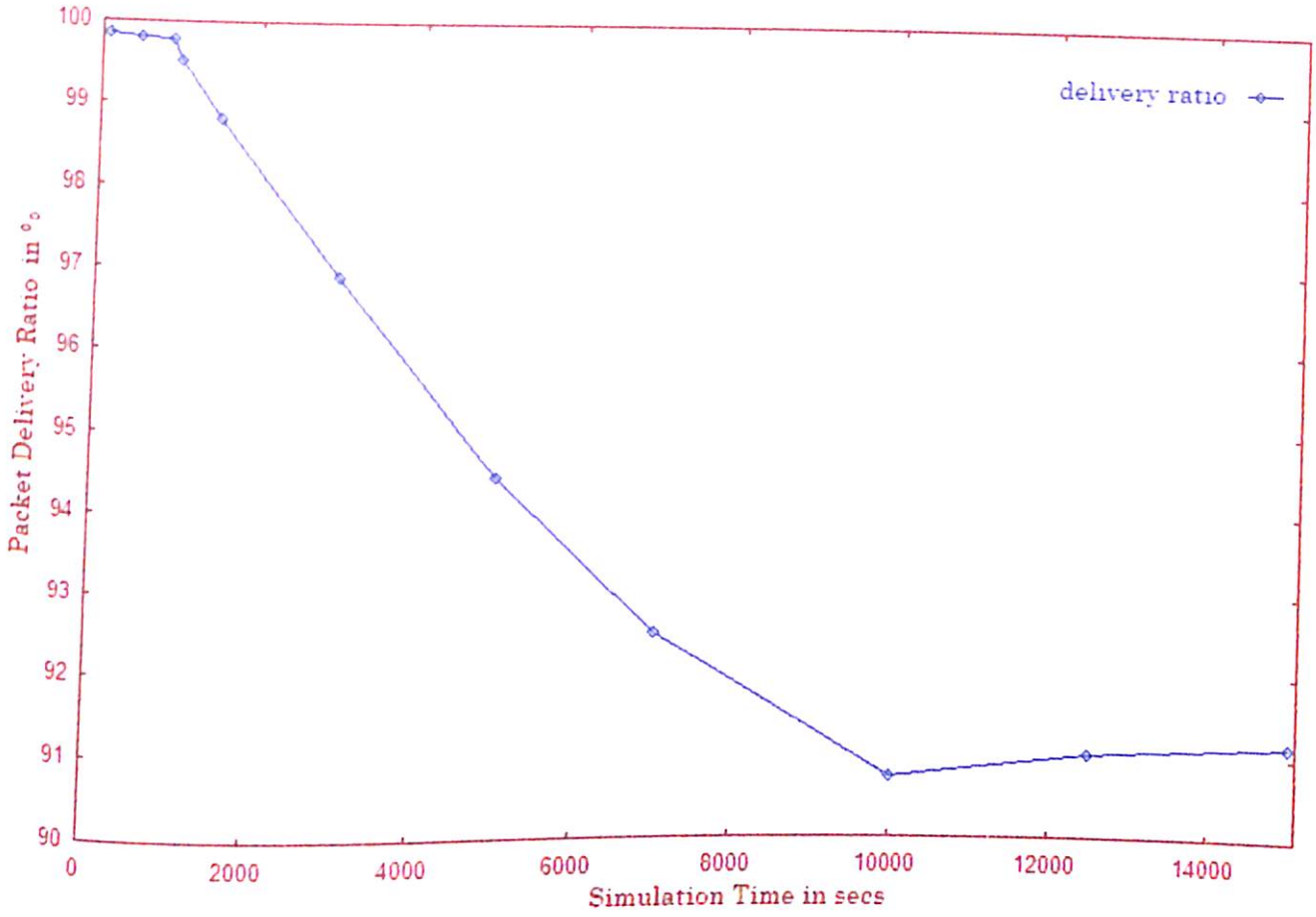


Fig 5.17 PDR Vs Simulation Time (AODV)

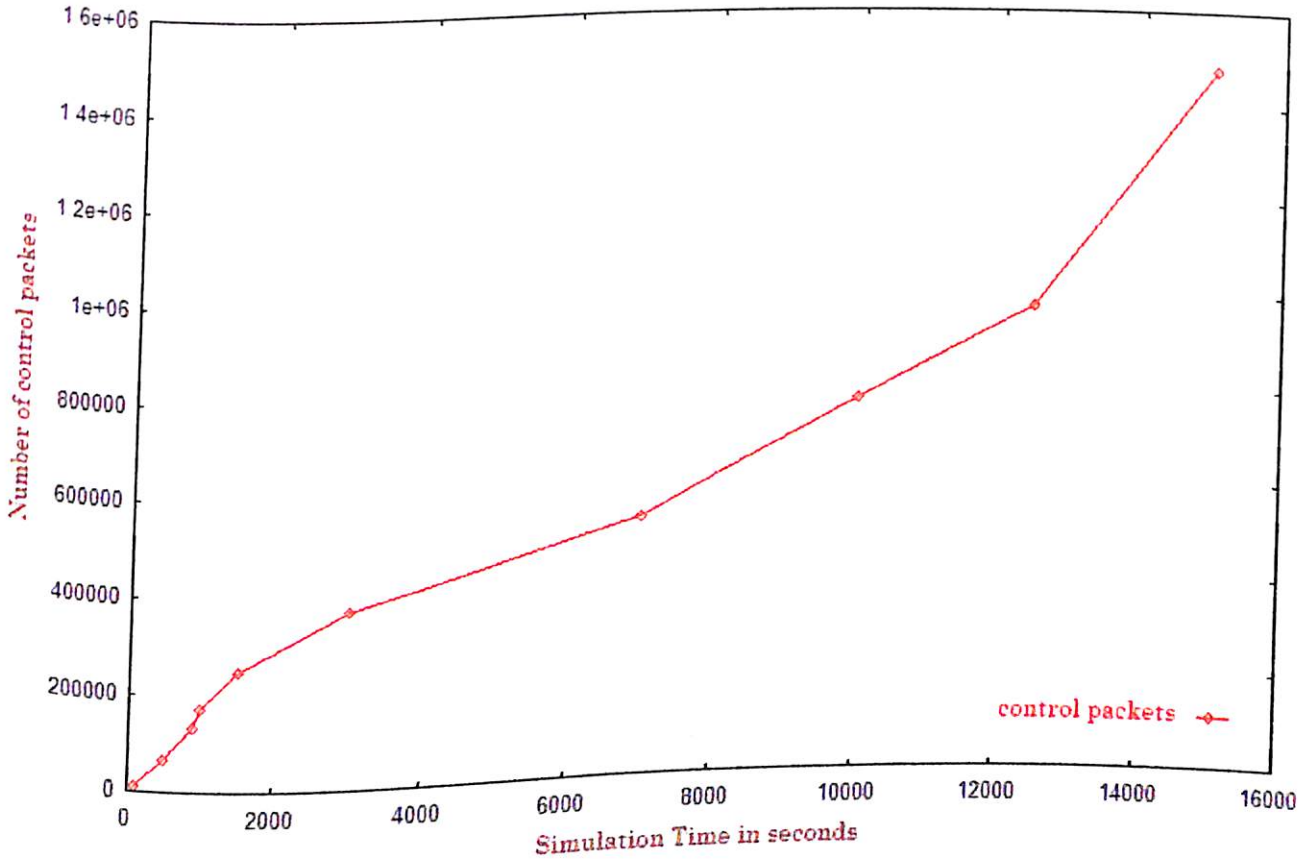


Fig 5.18 Number of Control Packets Vs Simulation Time (AODV)

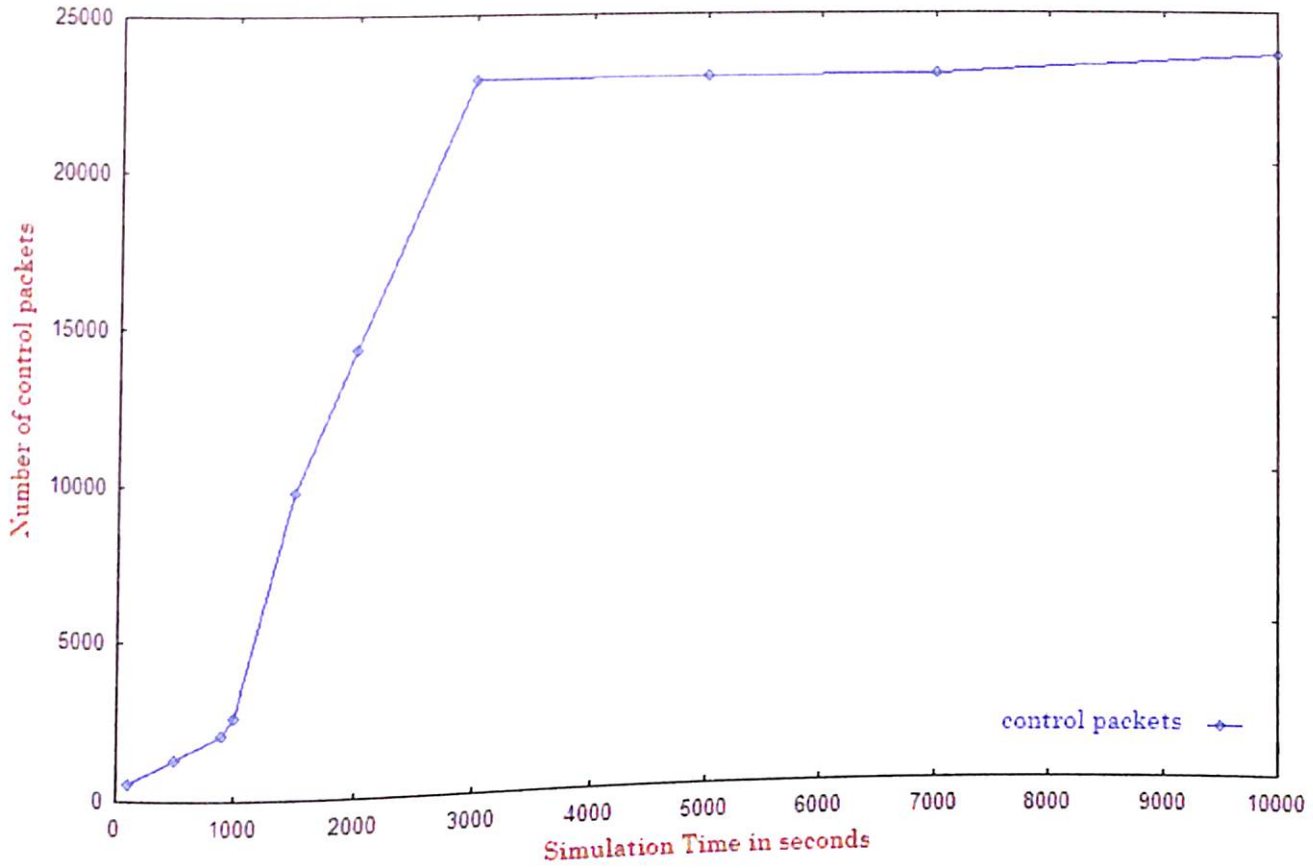


Fig 5.19 Number of Control Packets Vs Simulation Time (DSR)

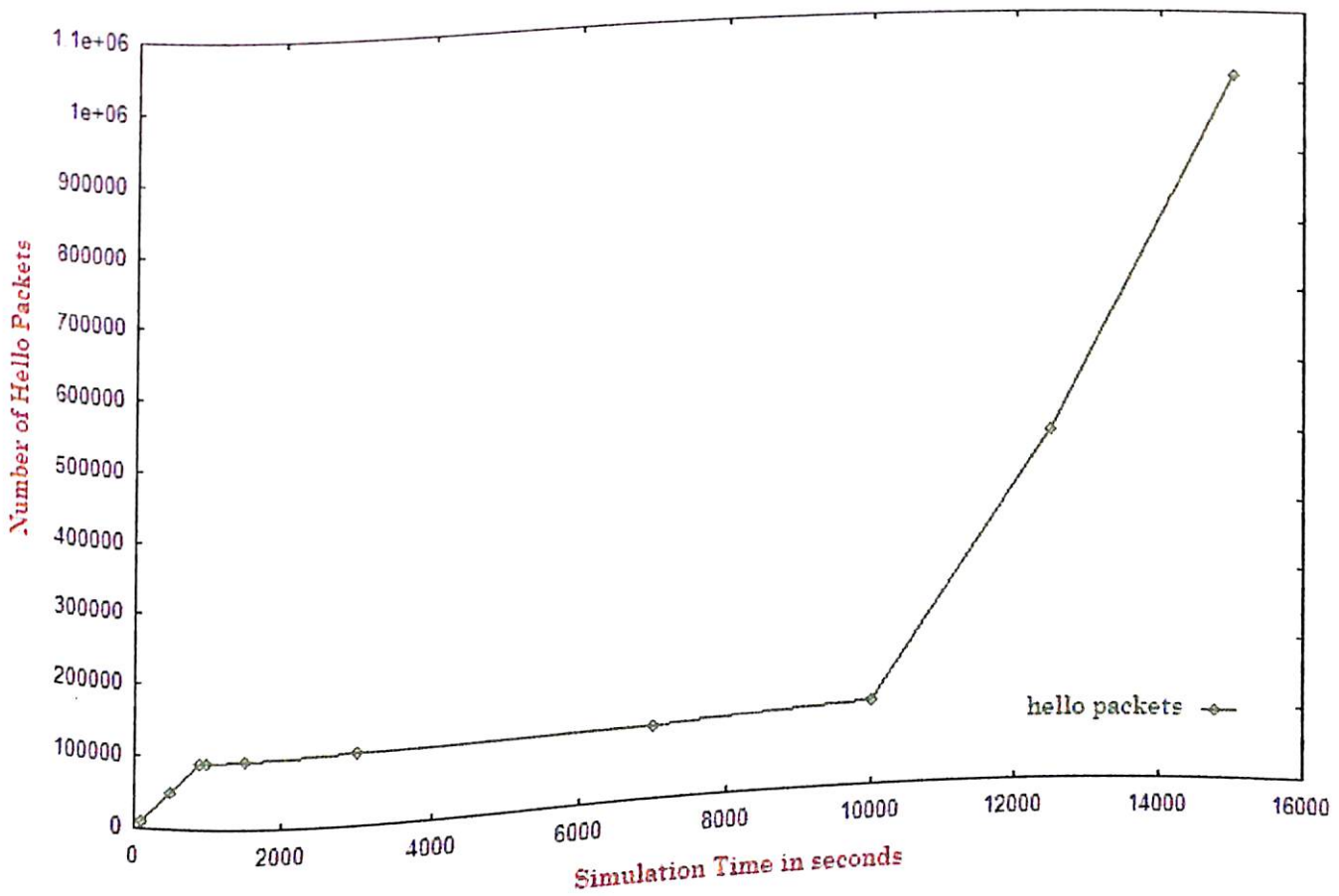


Fig 5.20 Number of Hello Packets Vs Simulation Time in seconds

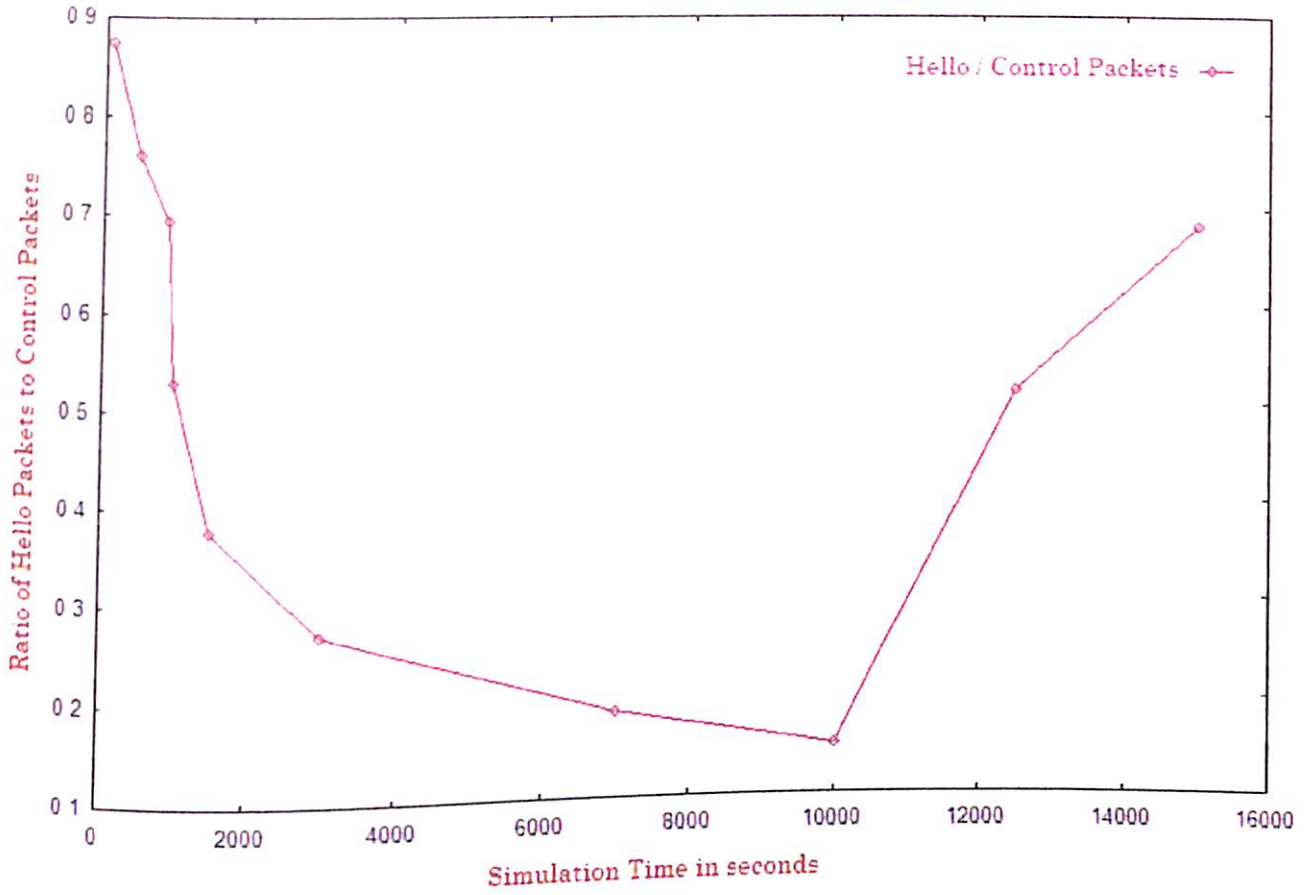


Fig 5.21 Ratio of Hello Packets to Control Packets Vs Simulation Time in seconds

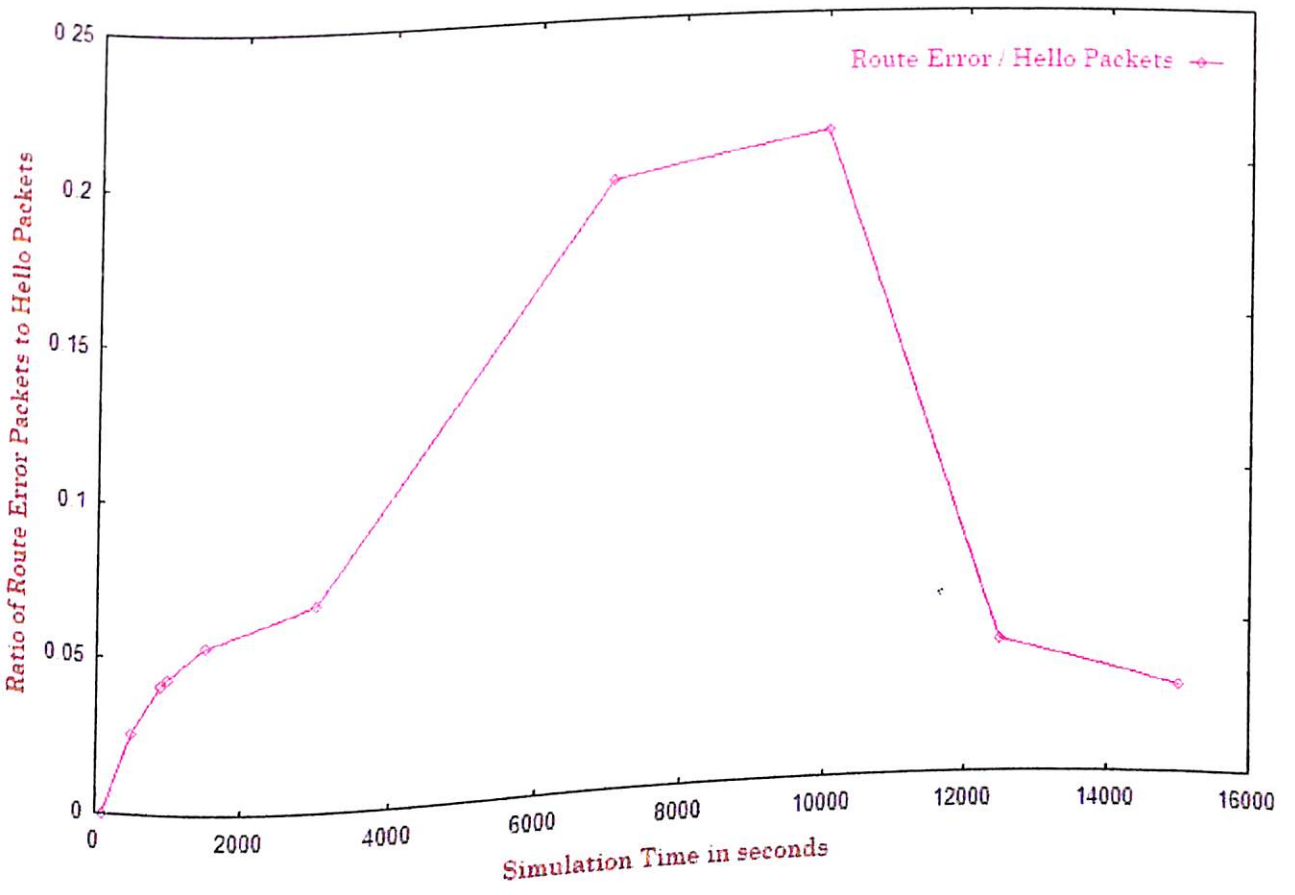


Fig 5.22 Ratio of RERR Packets to Hello Packets Vs Simulation Time in seconds

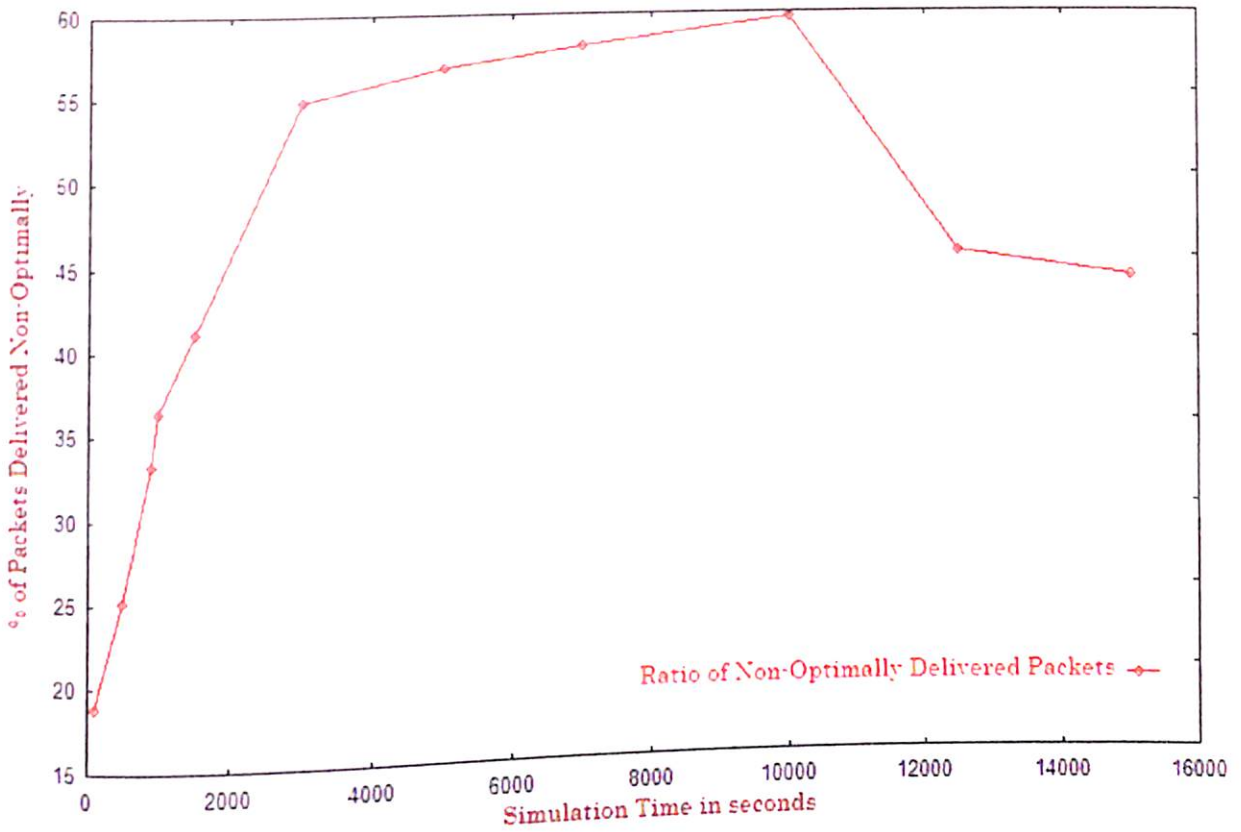


Fig 5.23a % of Packets Delivered Non-Optimally Vs Simulation Time in seconds for AODV

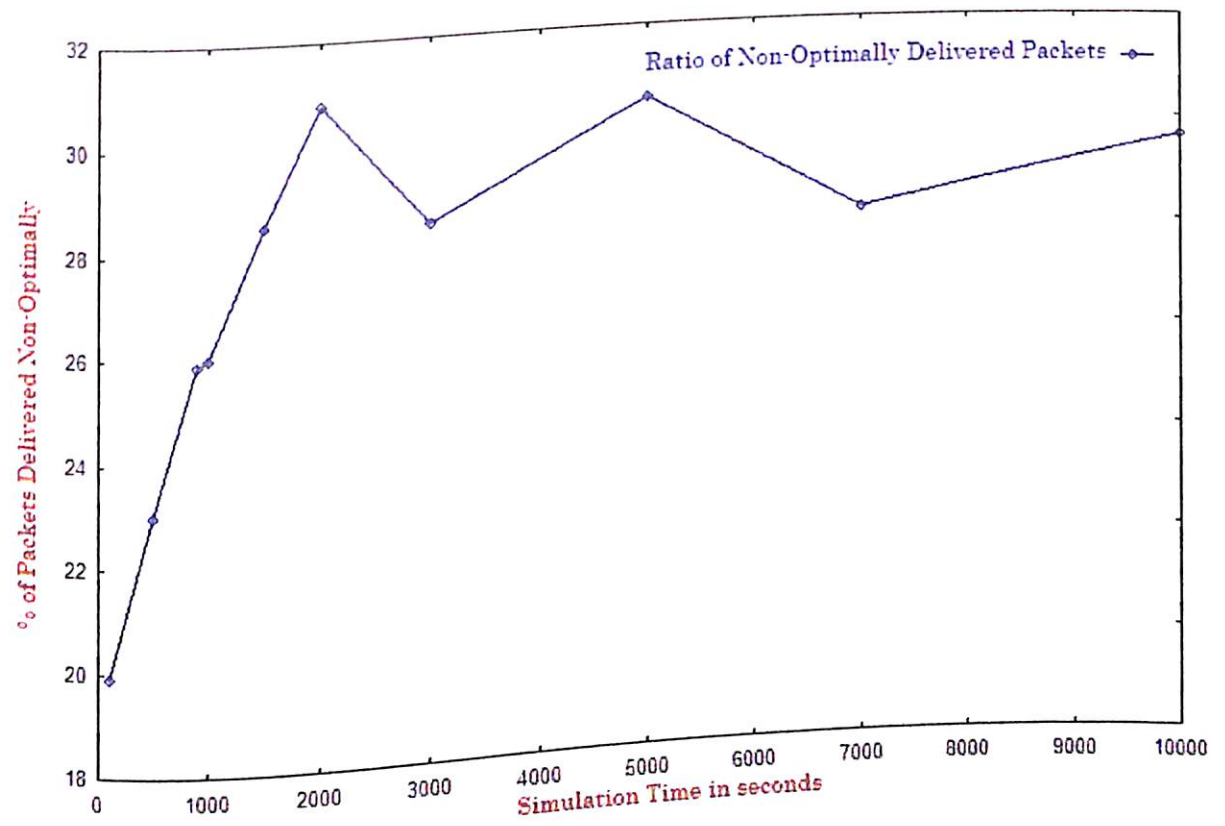


Fig 5.23b % of Packets Delivered Non-Optimally Vs Simulation Time in seconds for DSR

Effect of Intelligent Caching Scheme on Network Performance
Packet Deliver Ratio Analysis
(fig. 5.24a - fig. 5.32c)

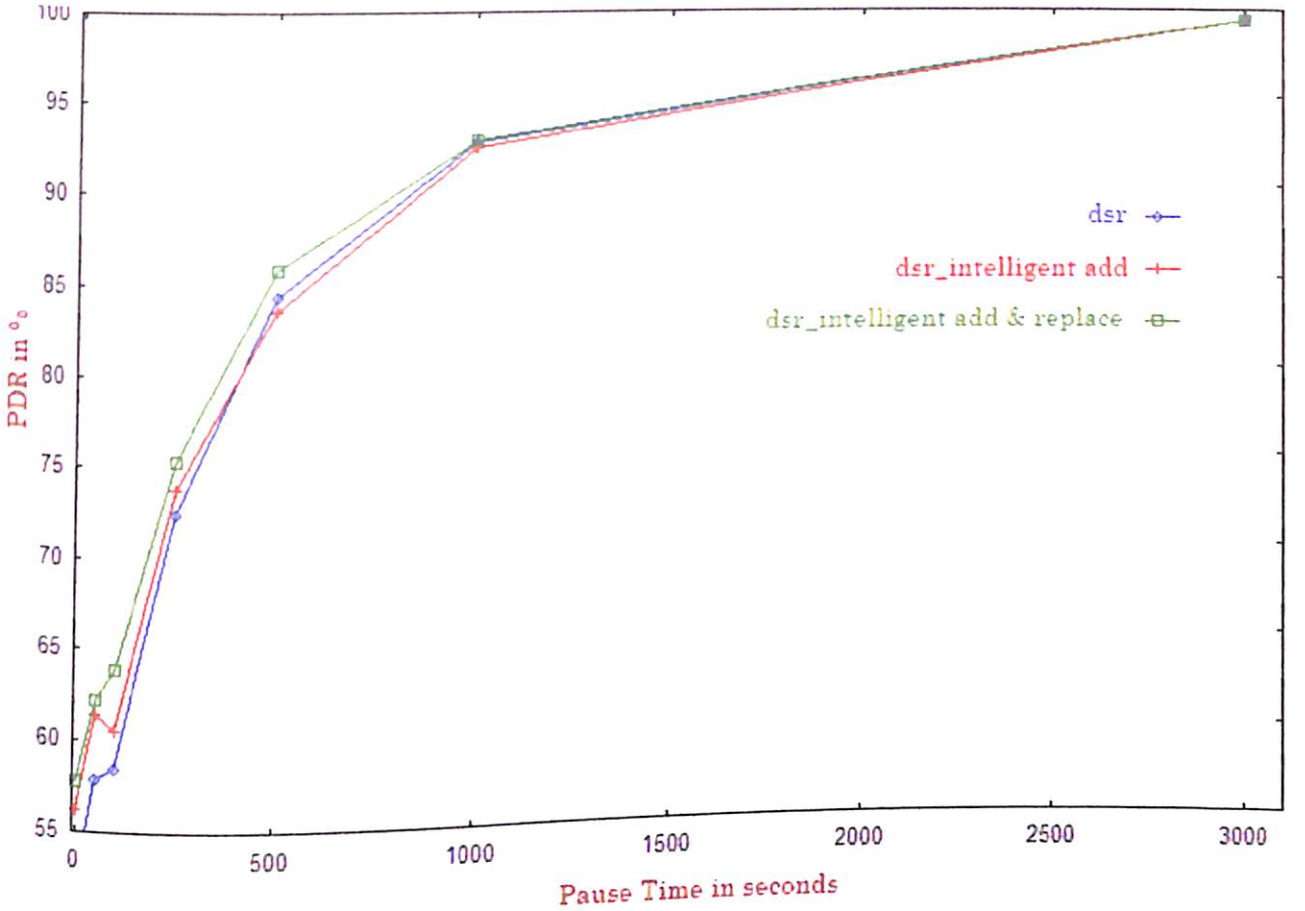


Fig 5.24a PDR Vs Pause Time at a maximum node speed of 30m/s

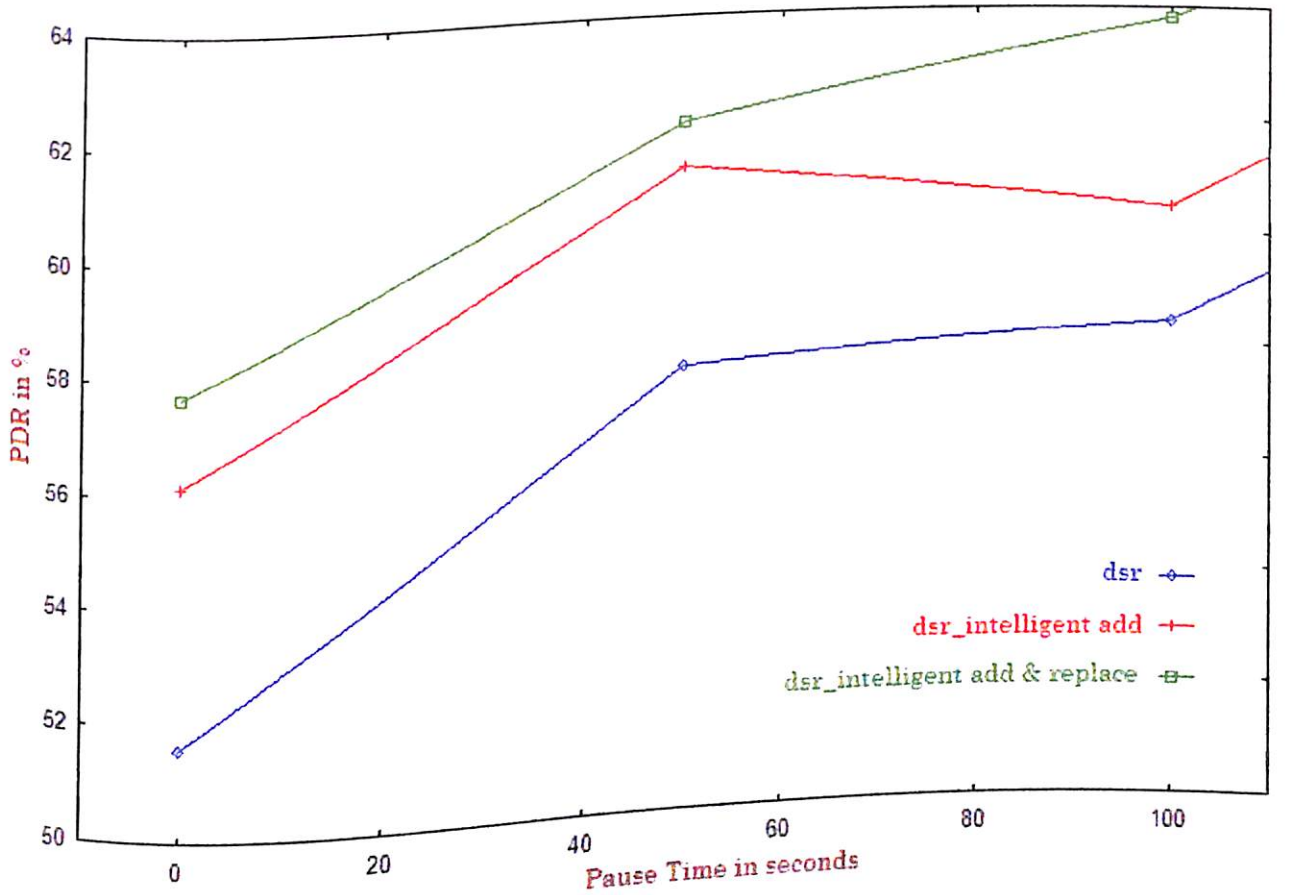


Fig 5.24b PDR Vs Pause Time (0-100s) at a maximum node speed of 30m/s

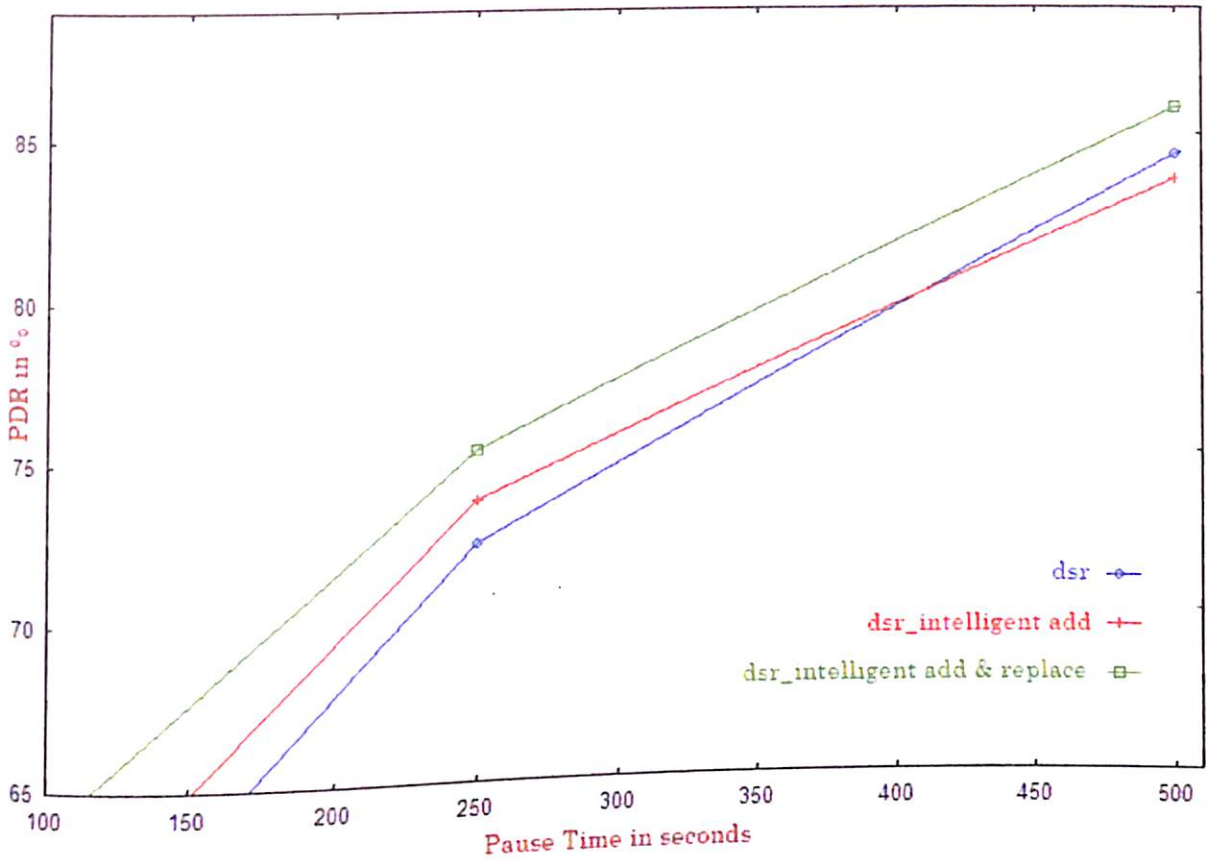


Fig 5.24c PDR Vs Pause Time (100-500s) at a maximum node speed of 30m/s

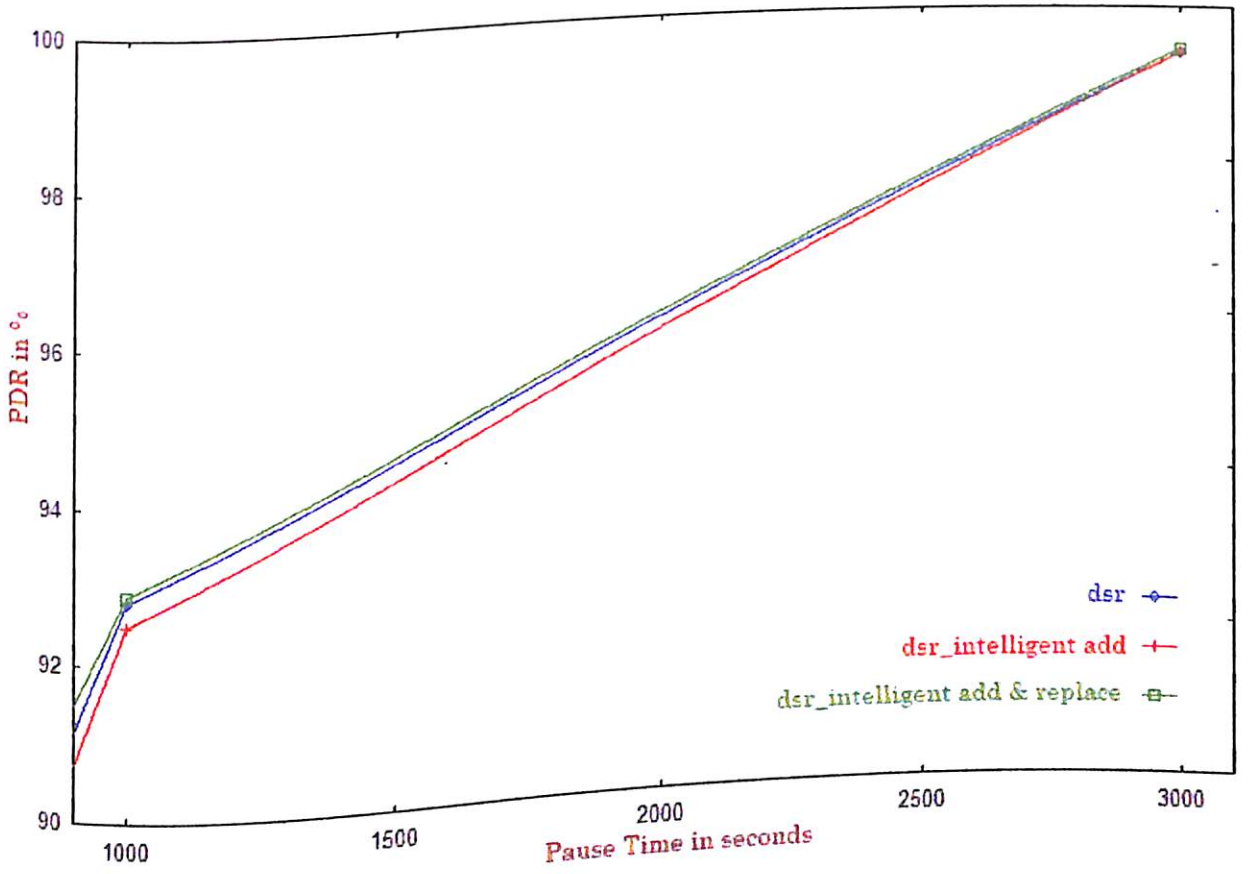


Fig 5.24d PDR Vs Pause Time (1000-3000s) at a maximum node speed of 30m/s

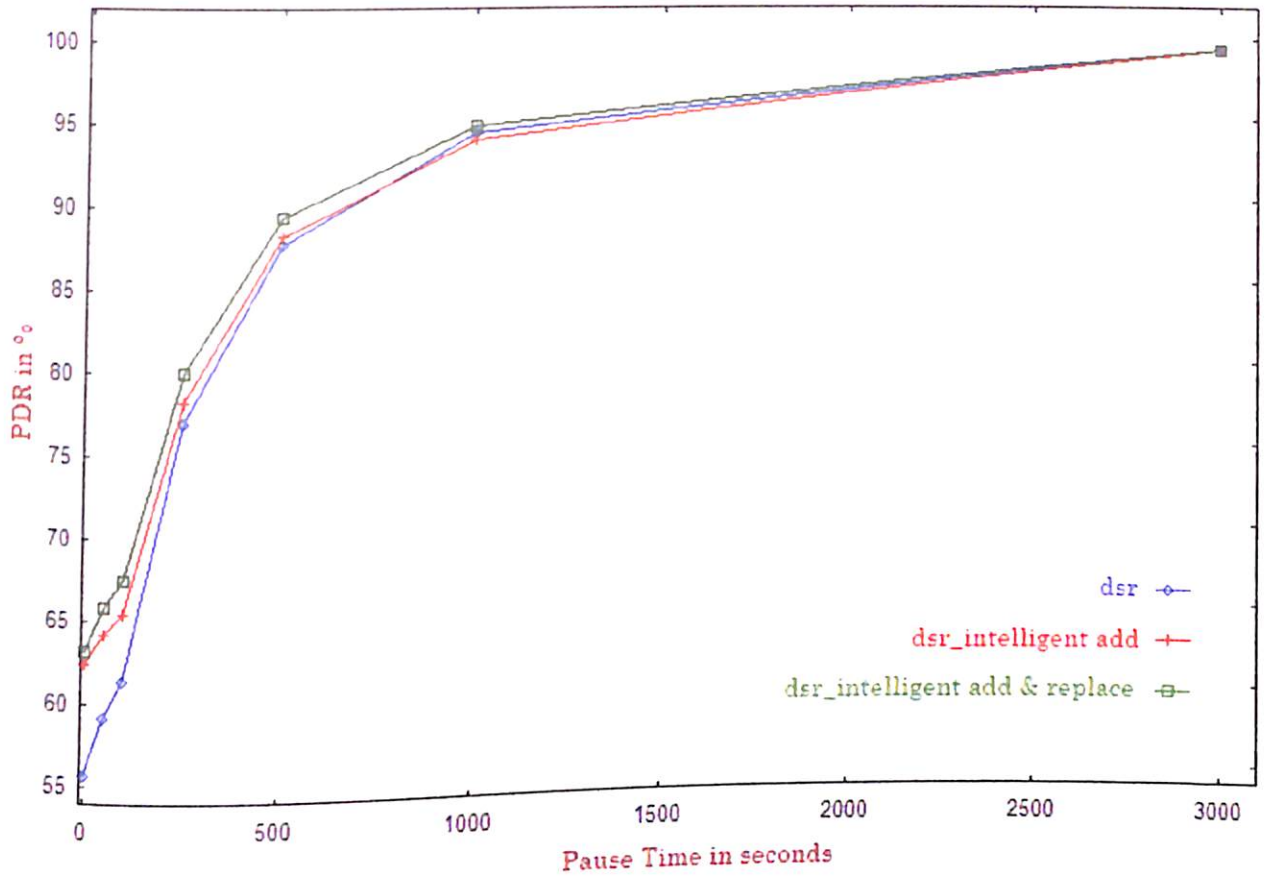


Fig 5.25a PDR Vs Pause Time at a maximum node speed of 20m/s

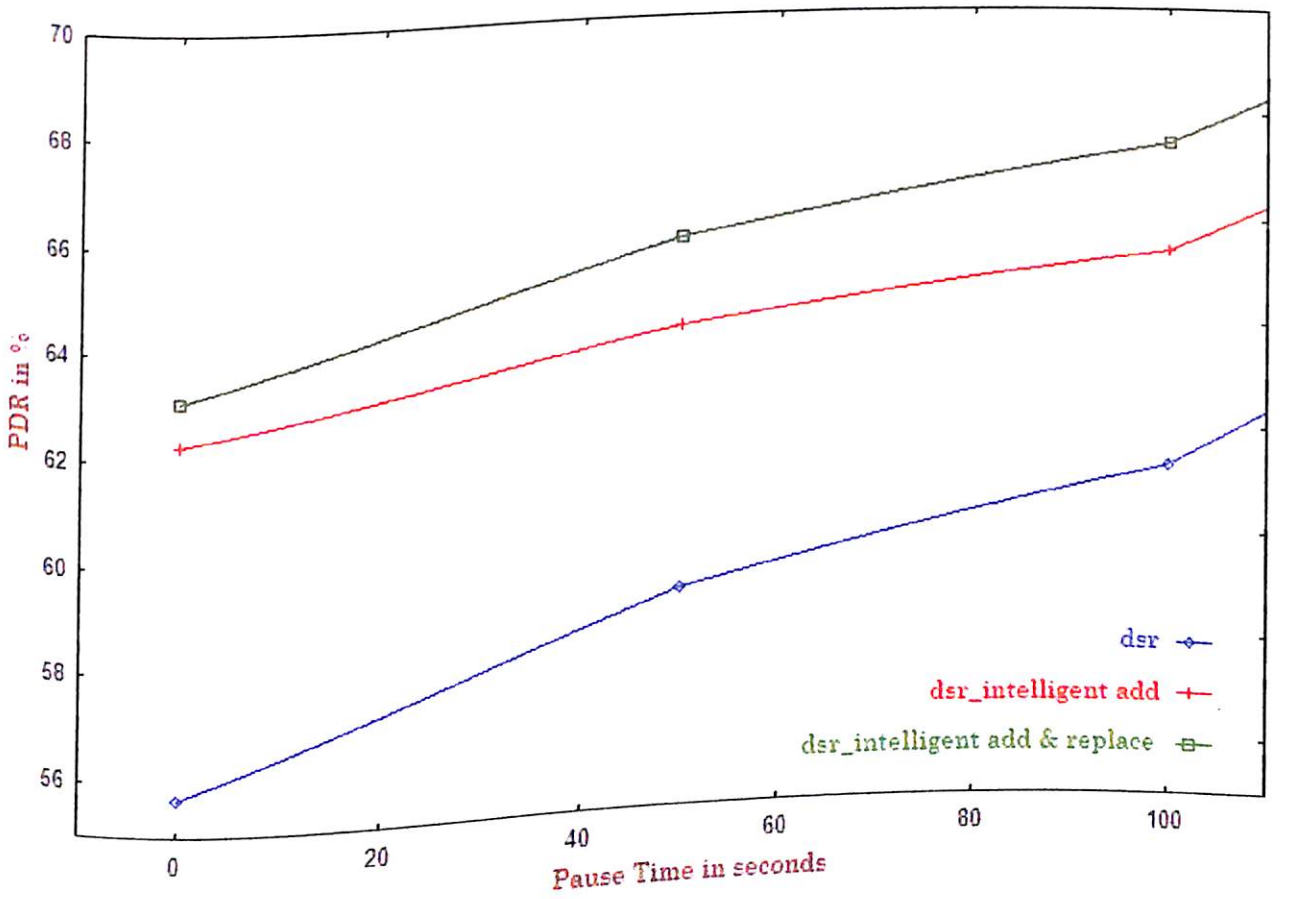


Fig 5.25b PDR Vs Pause Time (0-100s) at a maximum node speed of 20m/s

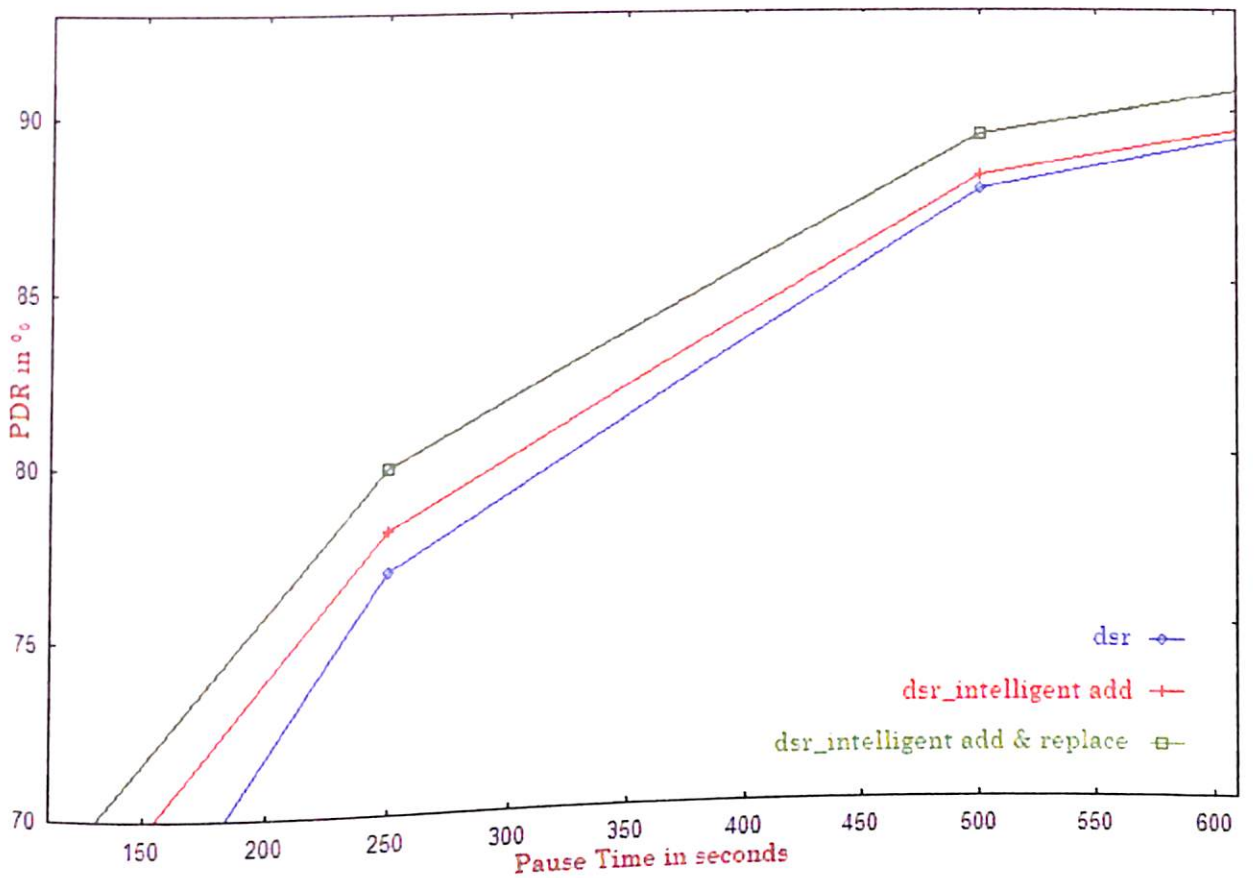


Fig 5.25c PDR Vs Pause Time (100-500s) at a maximum node speed of 20m/s

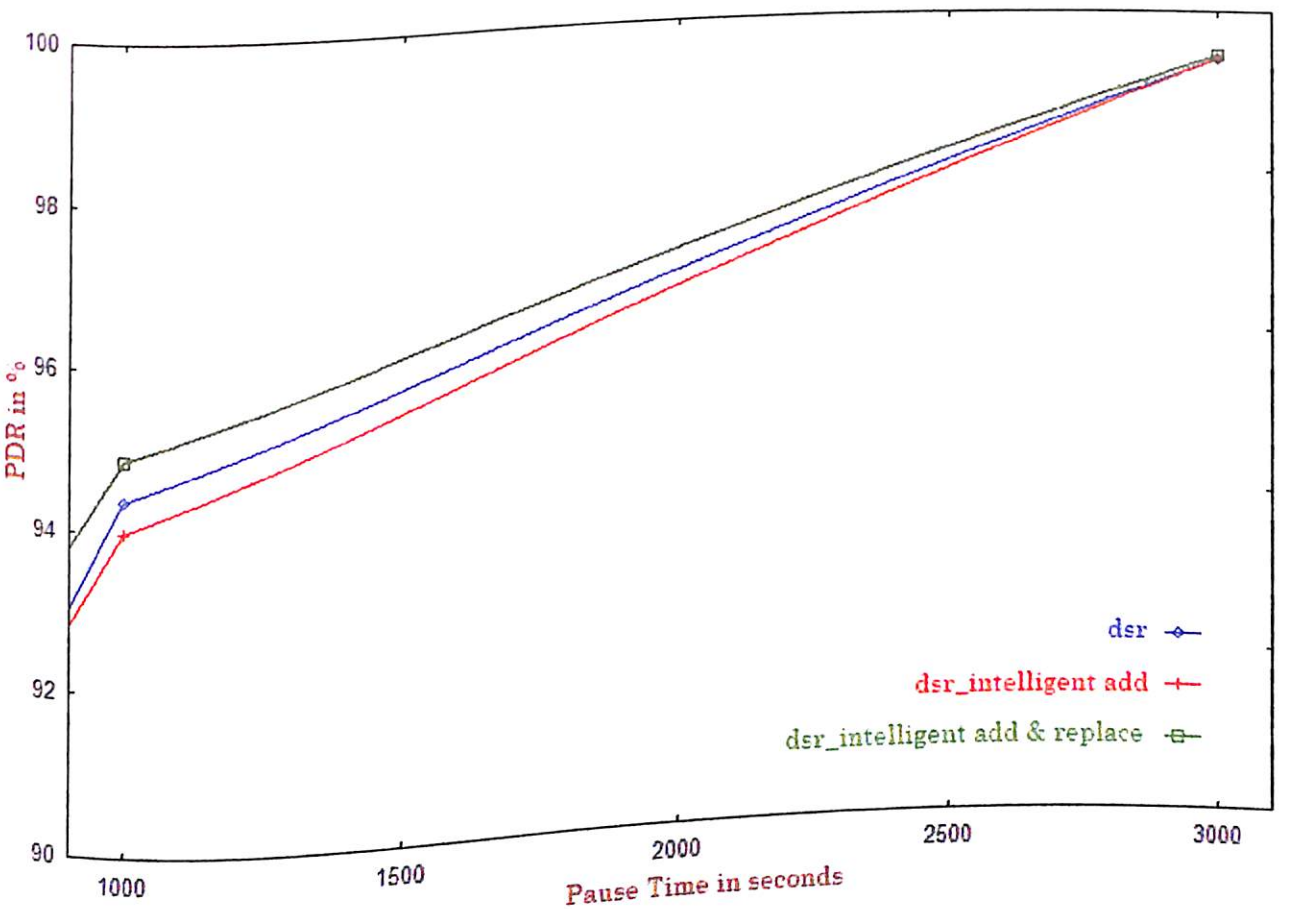


Fig 5.25d PDR Vs Pause Time (1000-3000s) at a maximum node speed of 20m/s

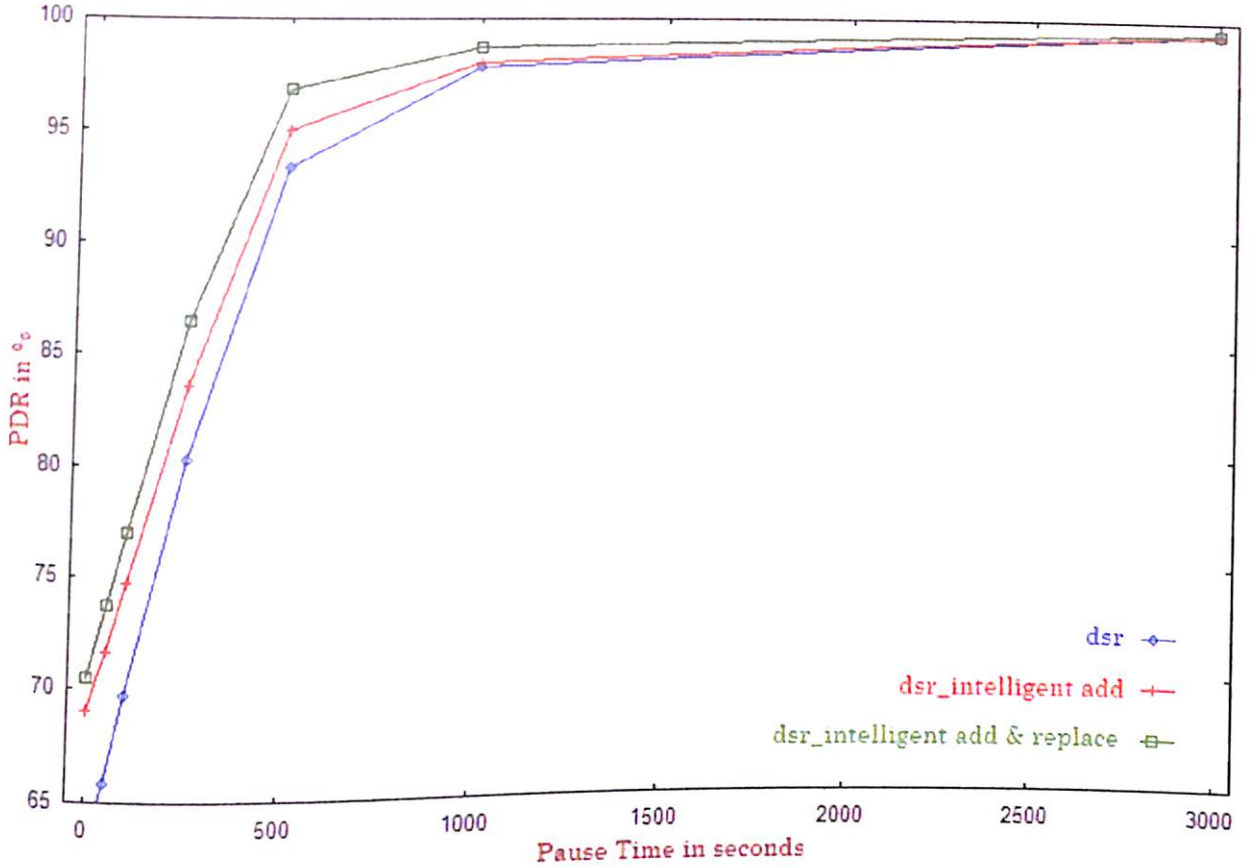


Fig 5.26a PDR Vs Pause Time at a maximum node speed of 15m/s

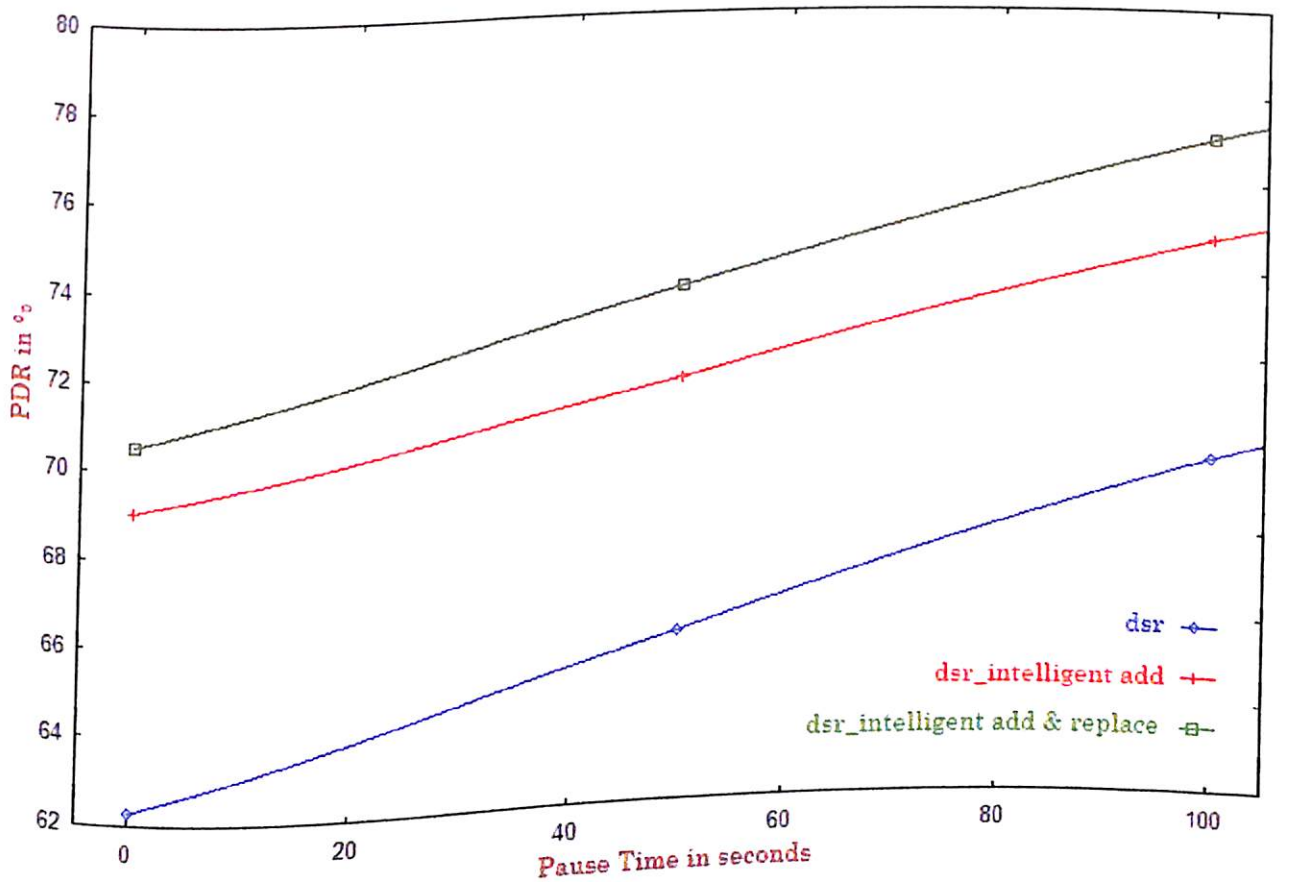


Fig 5.26b PDR Vs Pause Time (0-100s) at a maximum node speed of 15m/s

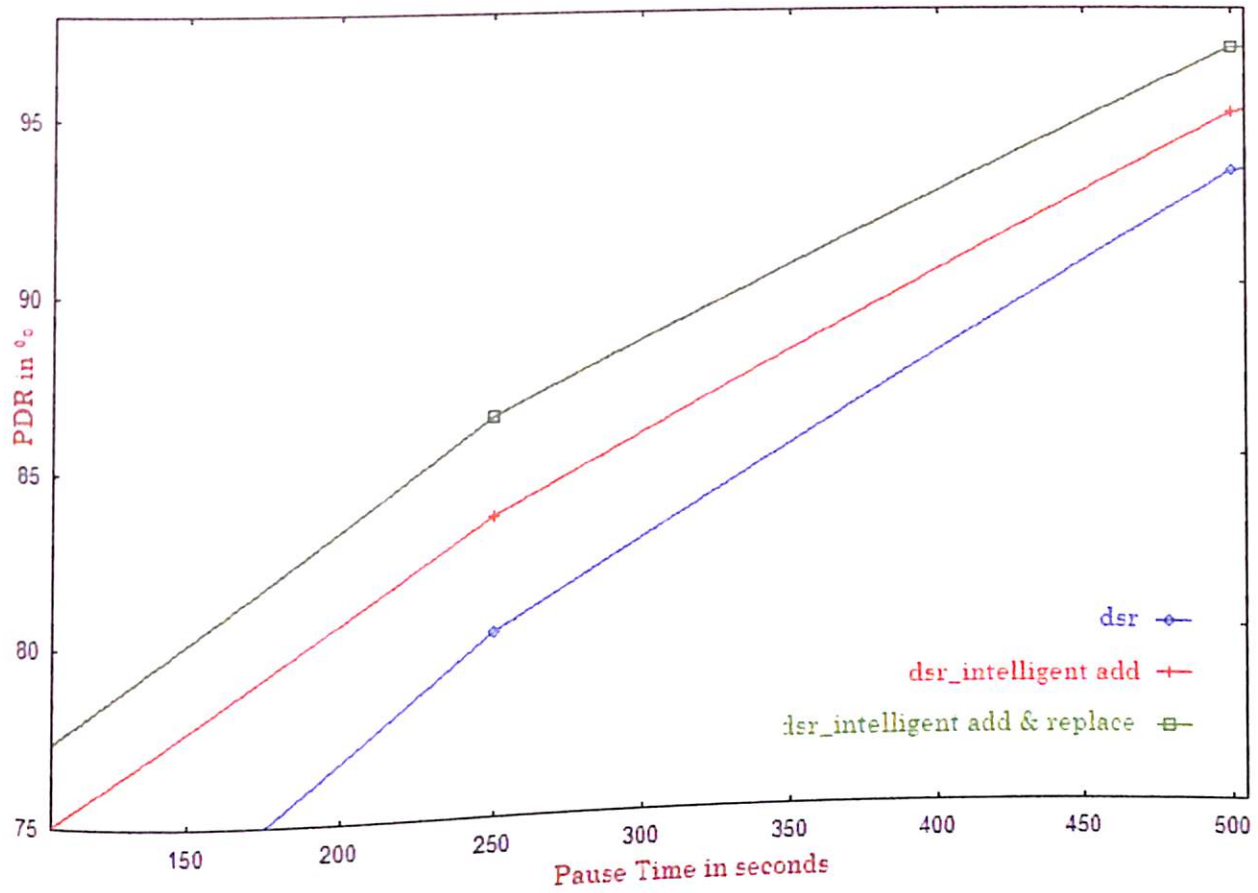


Fig 5.26c PDR Vs Pause Time (100-500s) at a maximum node speed of 15m/s

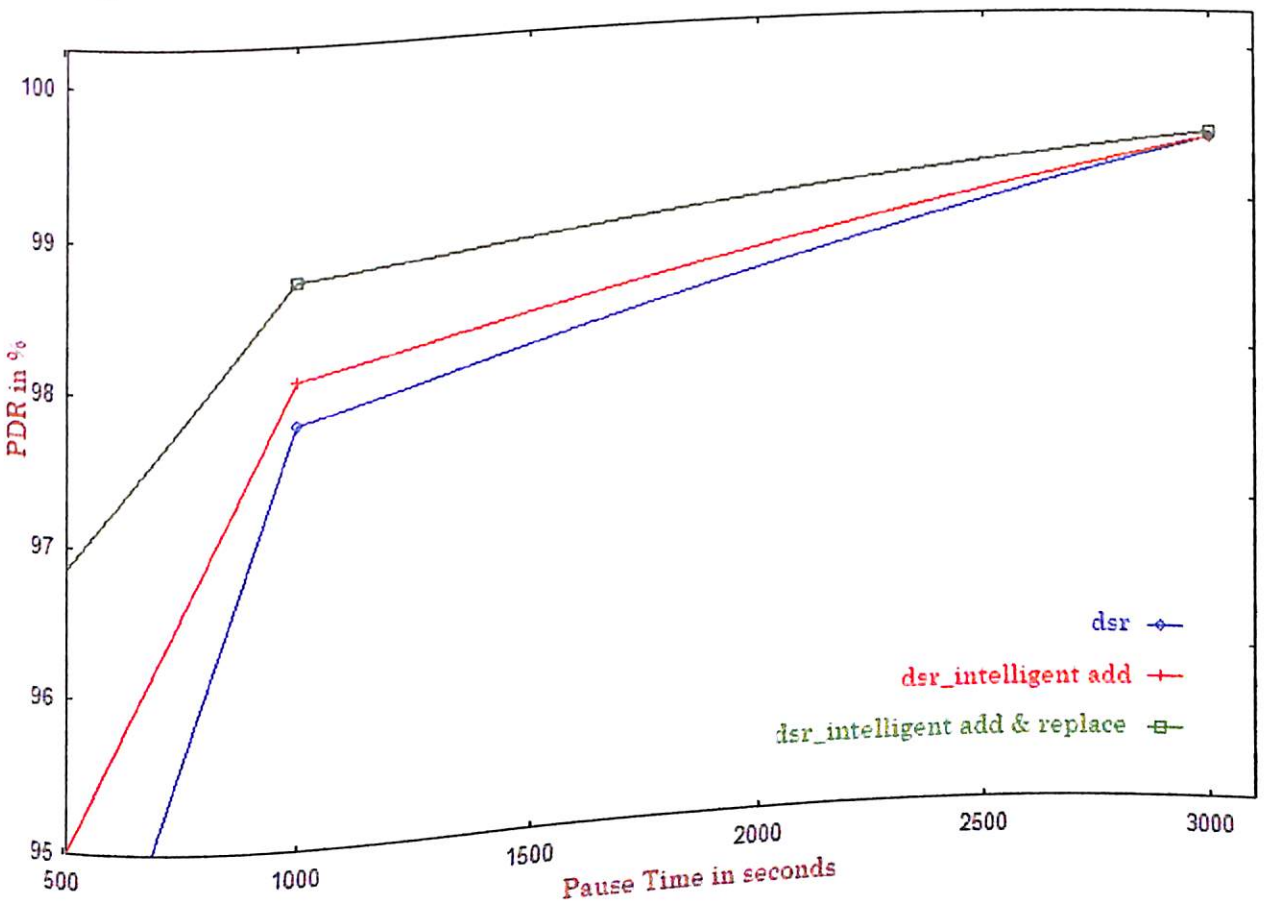


Fig 5.26d PDR Vs Pause Time (1000-3000s) at a maximum node speed of 15m/s

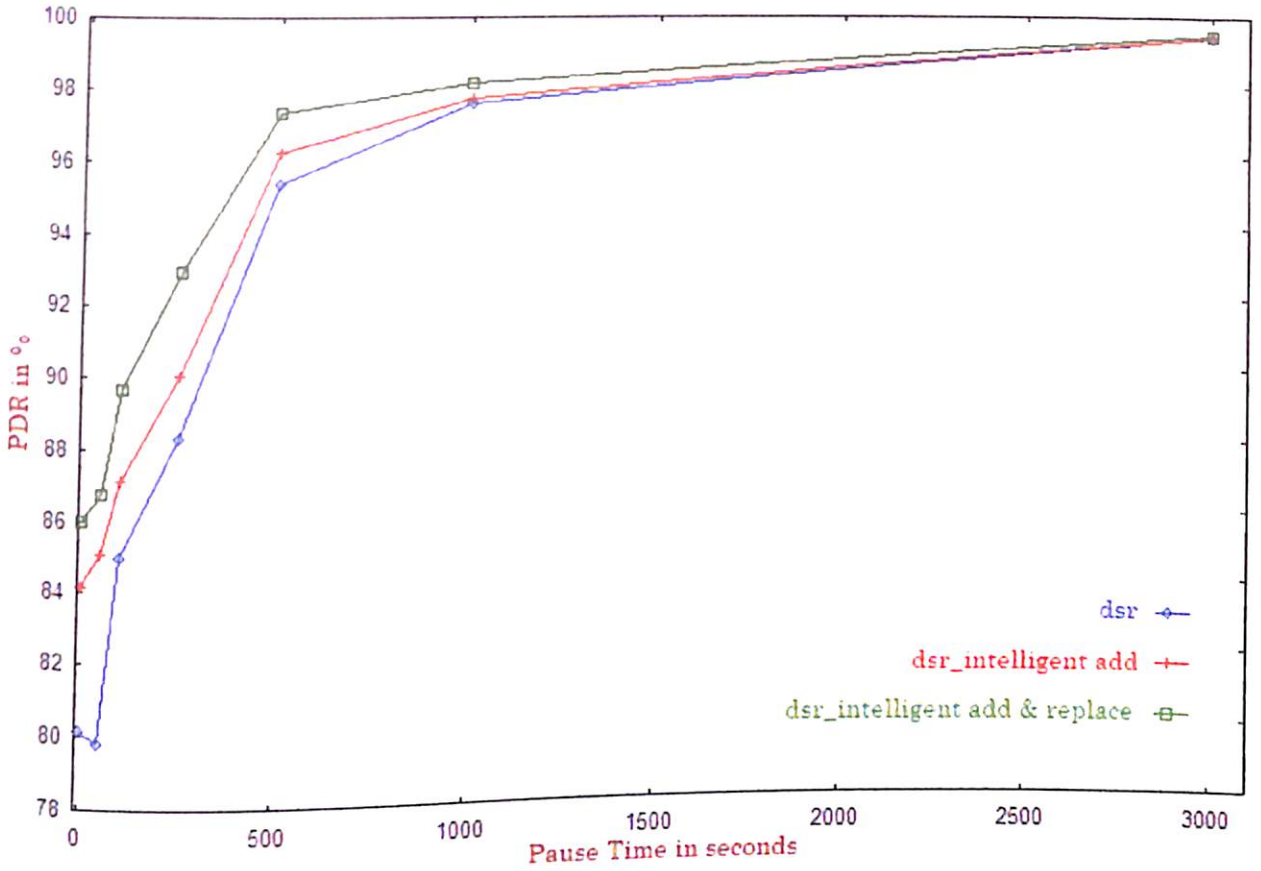


Fig 5.27a PDR Vs Pause Time with a maximum node speed of 10m/s

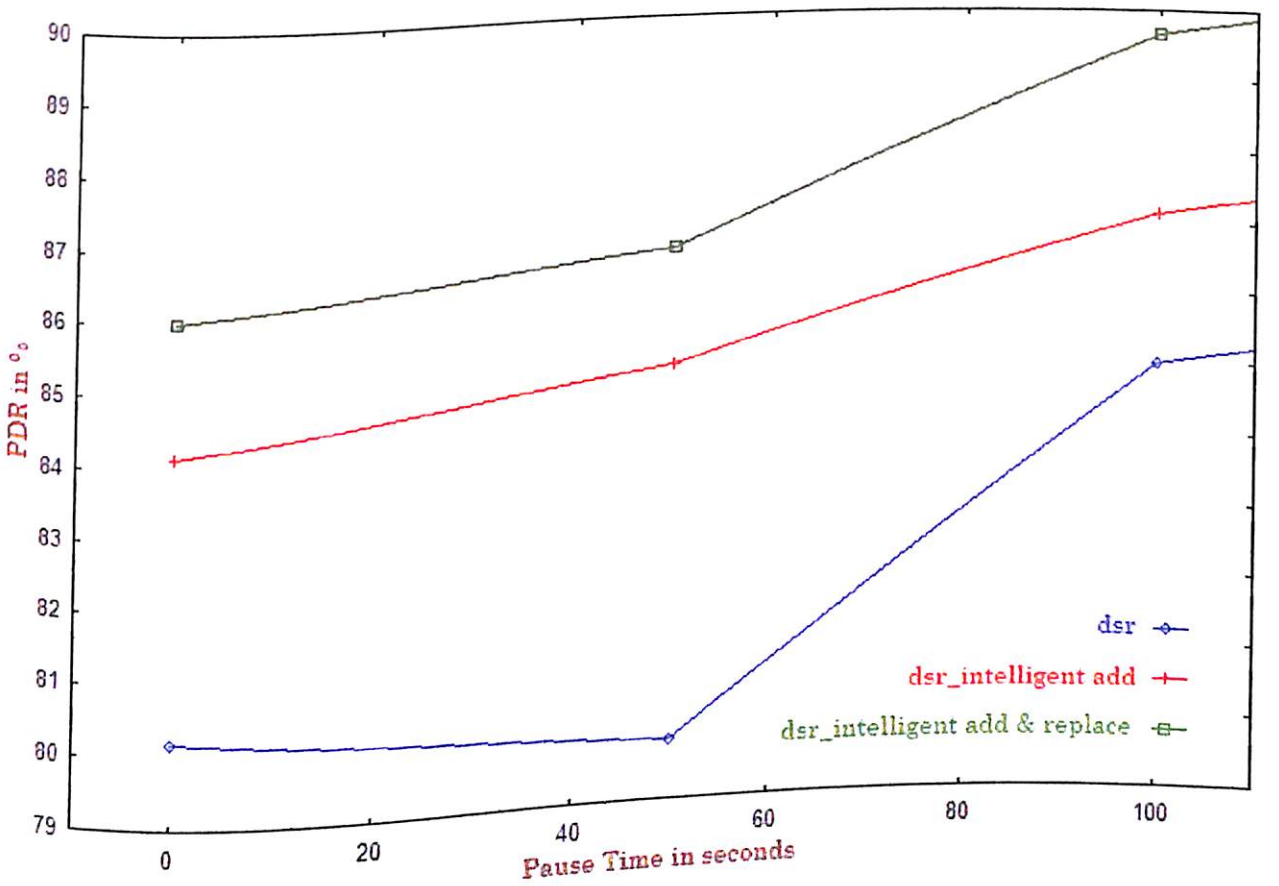


Fig 5.27b PDR Vs Pause Time (0-100s) with a maximum node speed of 10m/s

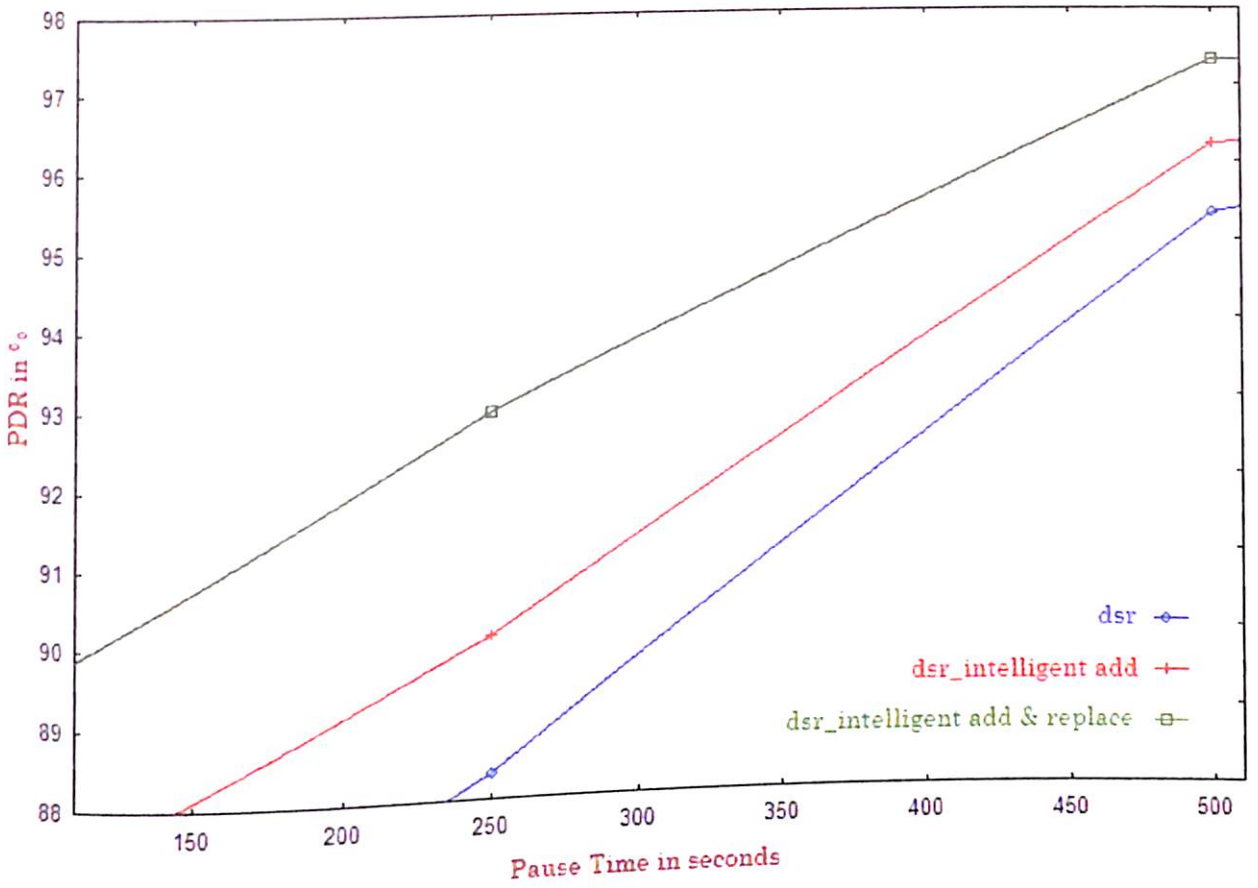


Fig 5.27c PDR Vs Pause Time (100-500s) with a maximum node speed of 10m/s

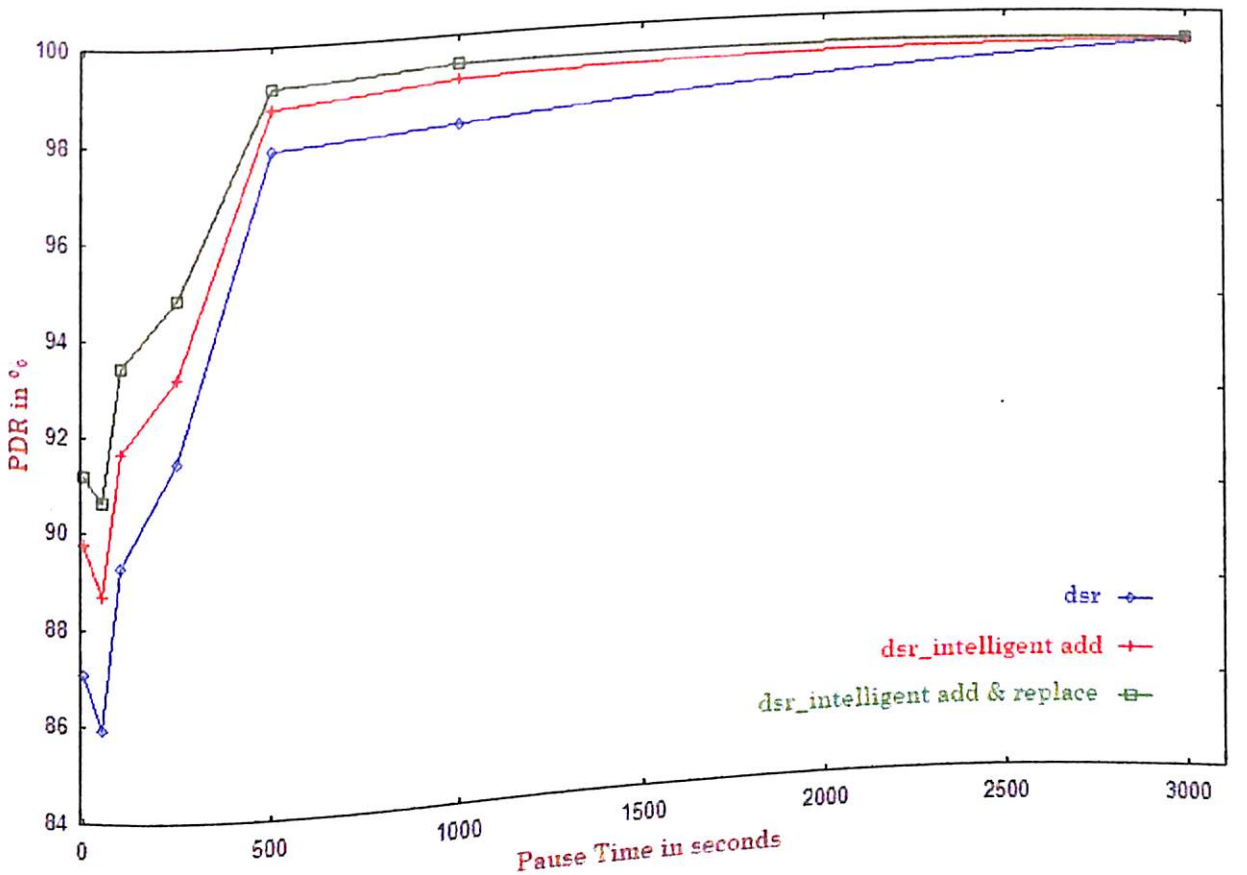


Fig 5.28 PDR Vs Pause Time with a maximum node speed of 5m/s

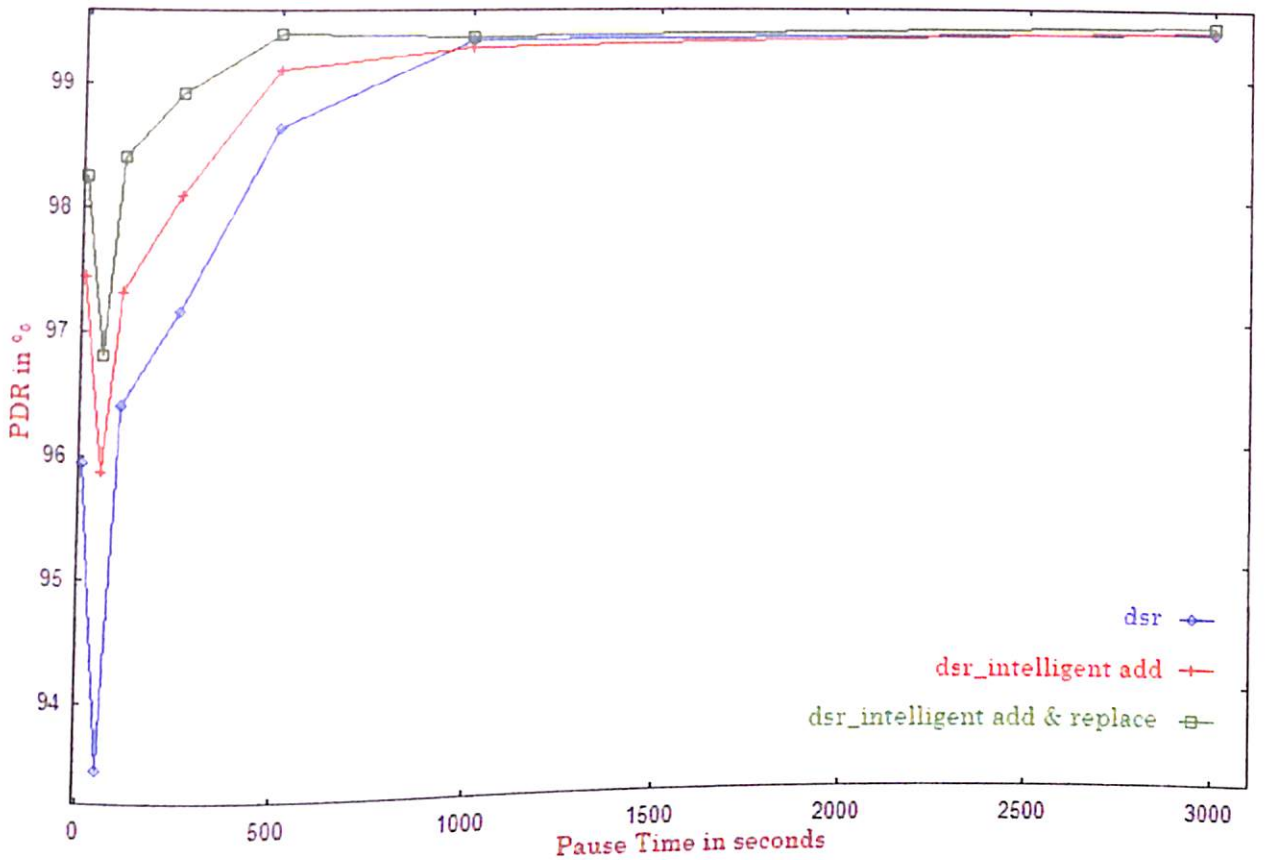


Fig 5.29 PDR Vs Pause Time with a maximum node speed of 1m/s

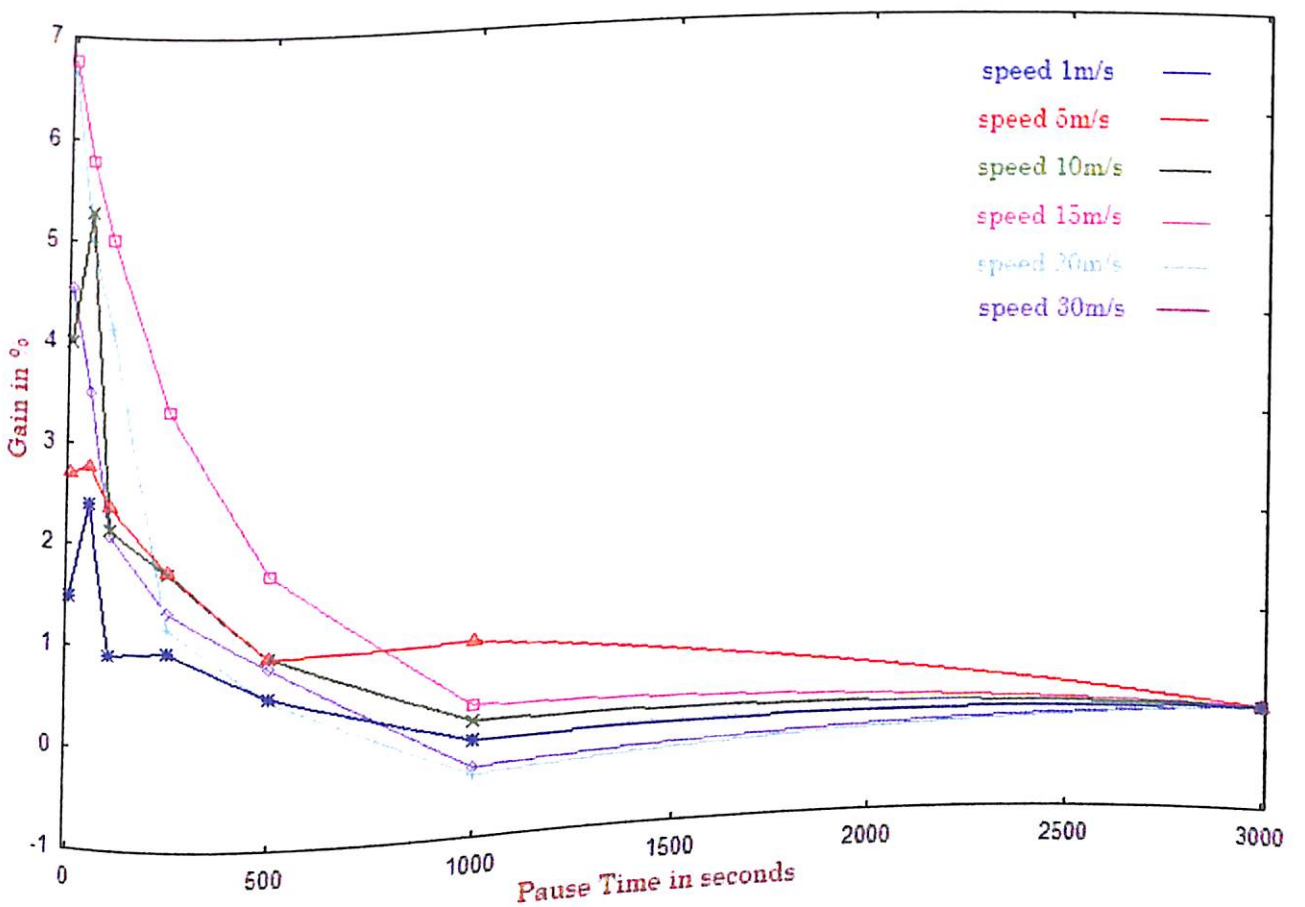


Fig 5.30 Gain in PDR Vs Pause Time when Cache Scheme 1 is used

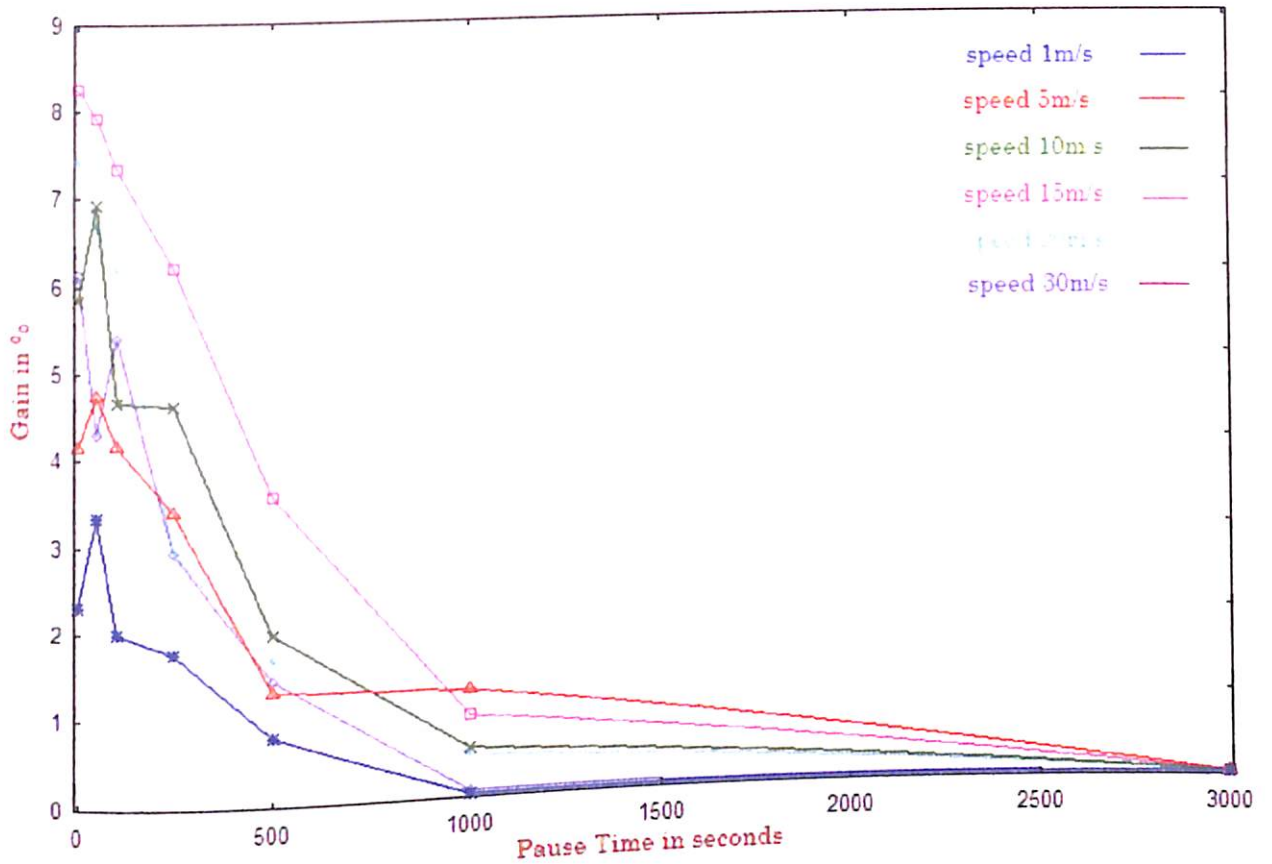


Fig 5.31 Gain in PDR Vs Pause Time when Cache Scheme 2 is used

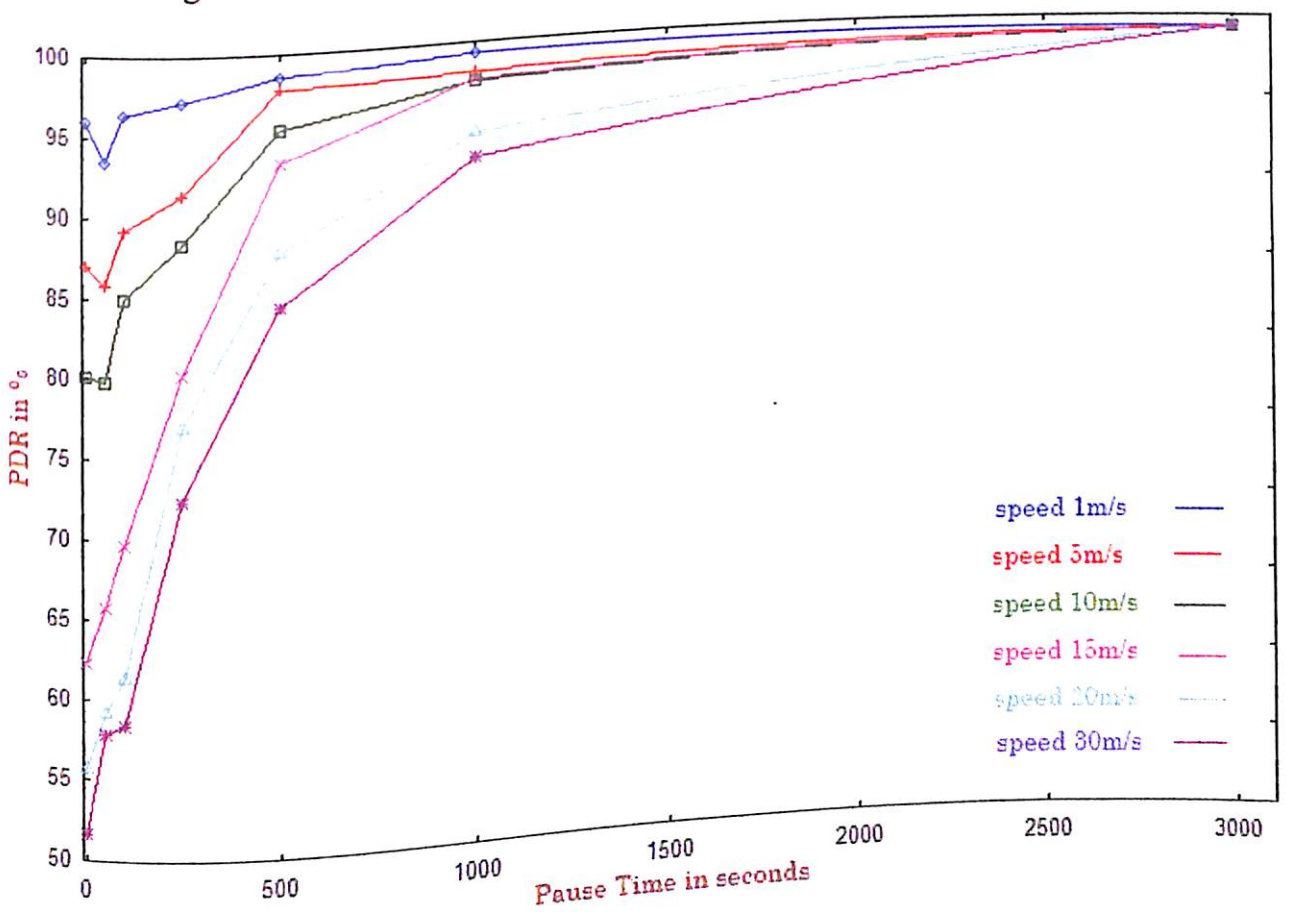


Fig5.32a PDR Vs Pause Time at various speeds for DSR

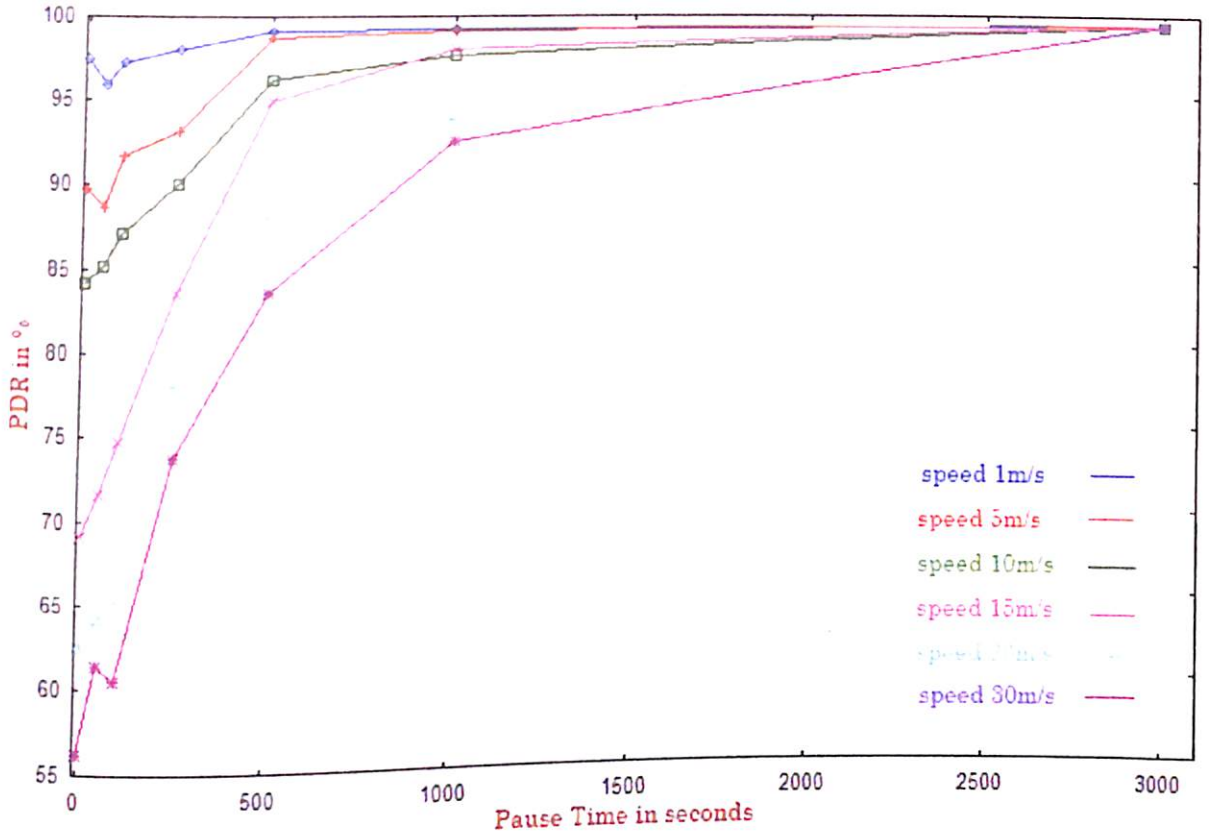


Fig 5.32b PDR Vs Pause Time at varying speeds for DSR with intelligent add scheme

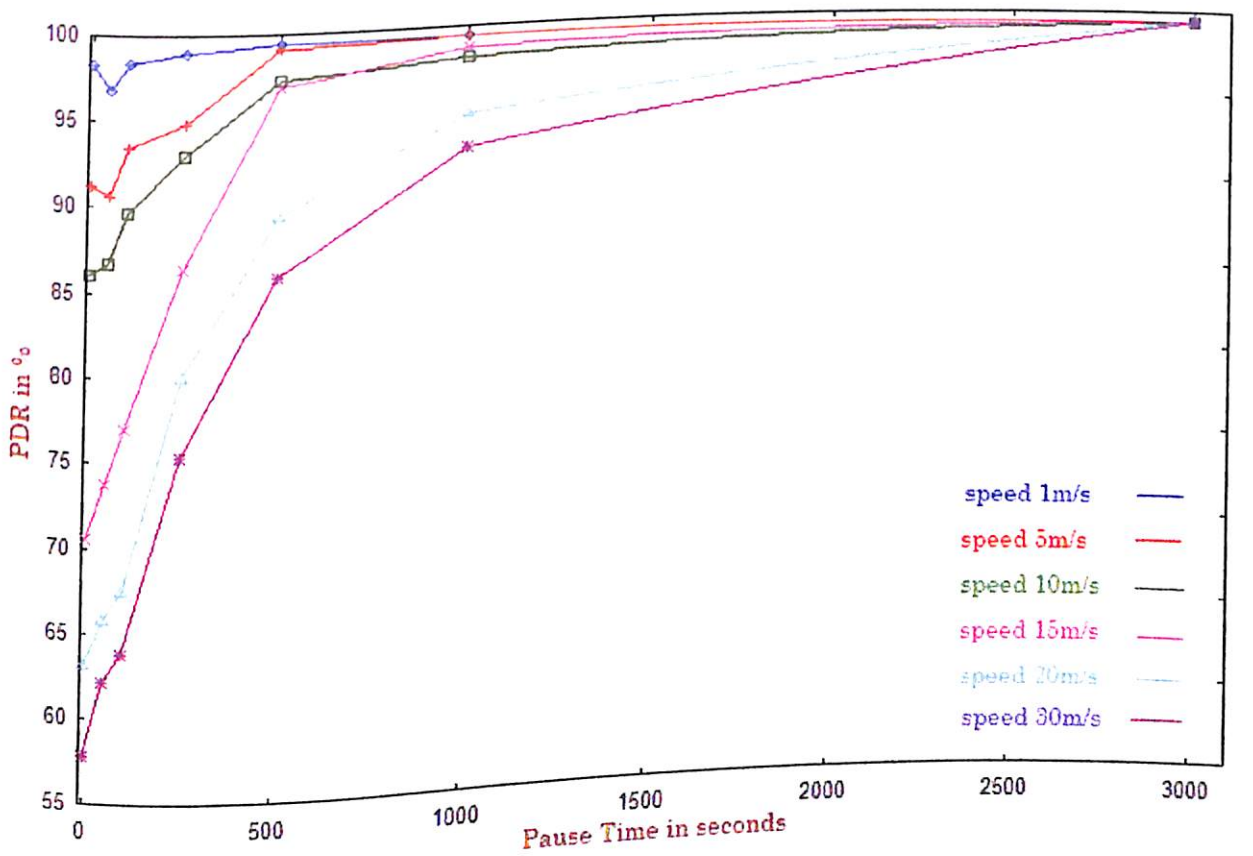


Fig 5.32c PDR Vs Pause Time at varying speeds for DSR with intelligent add and replace scheme

Effect of Intelligent Caching Scheme on Network Performance
Path Optimality Analysis
(fig. 5.33 - fig. 5.38)

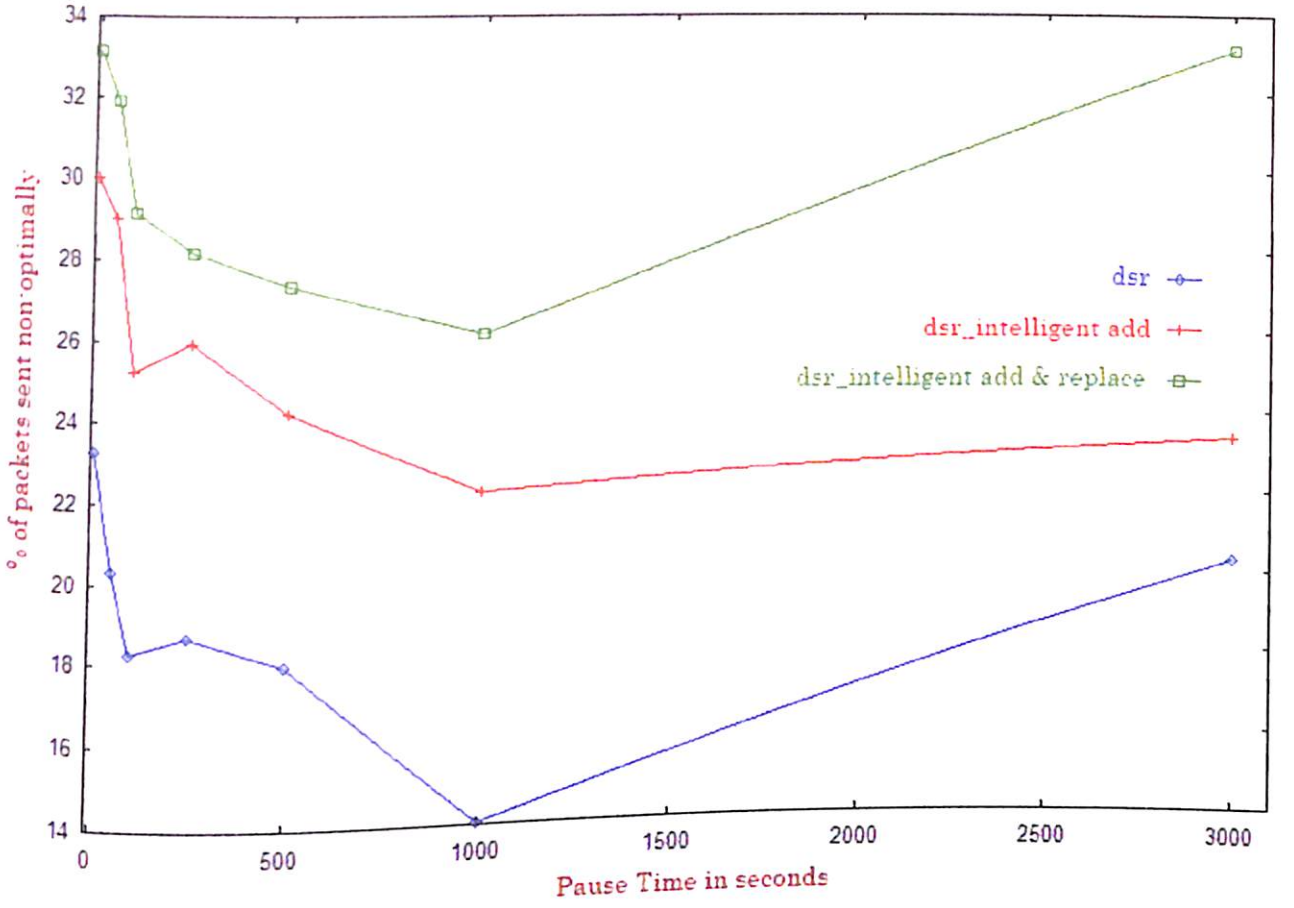


Fig 5.33 % of packets received non-optimally at a maximum node speed of 30m/s

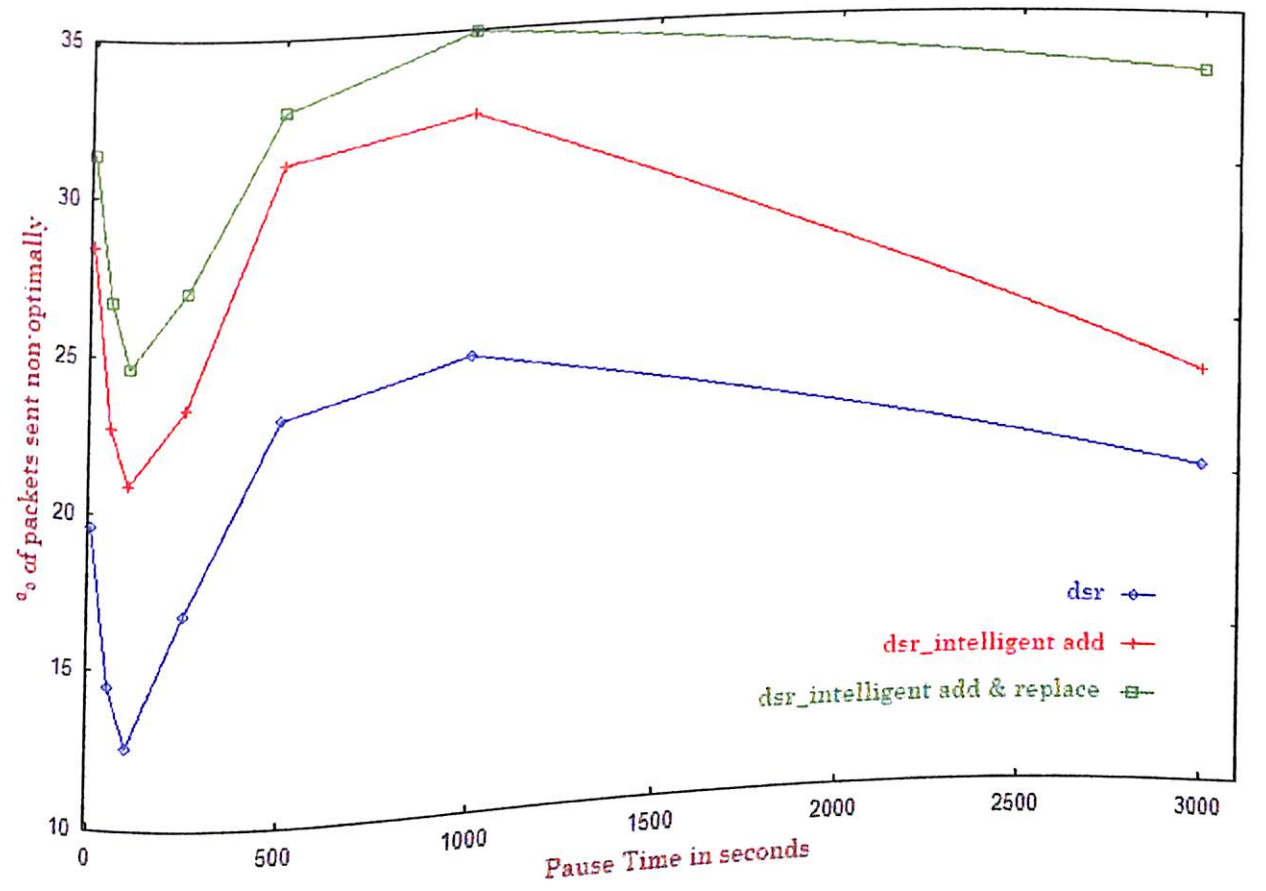


Fig 5.34 % of packets received non-optimally at a maximum node speed of 20m/s

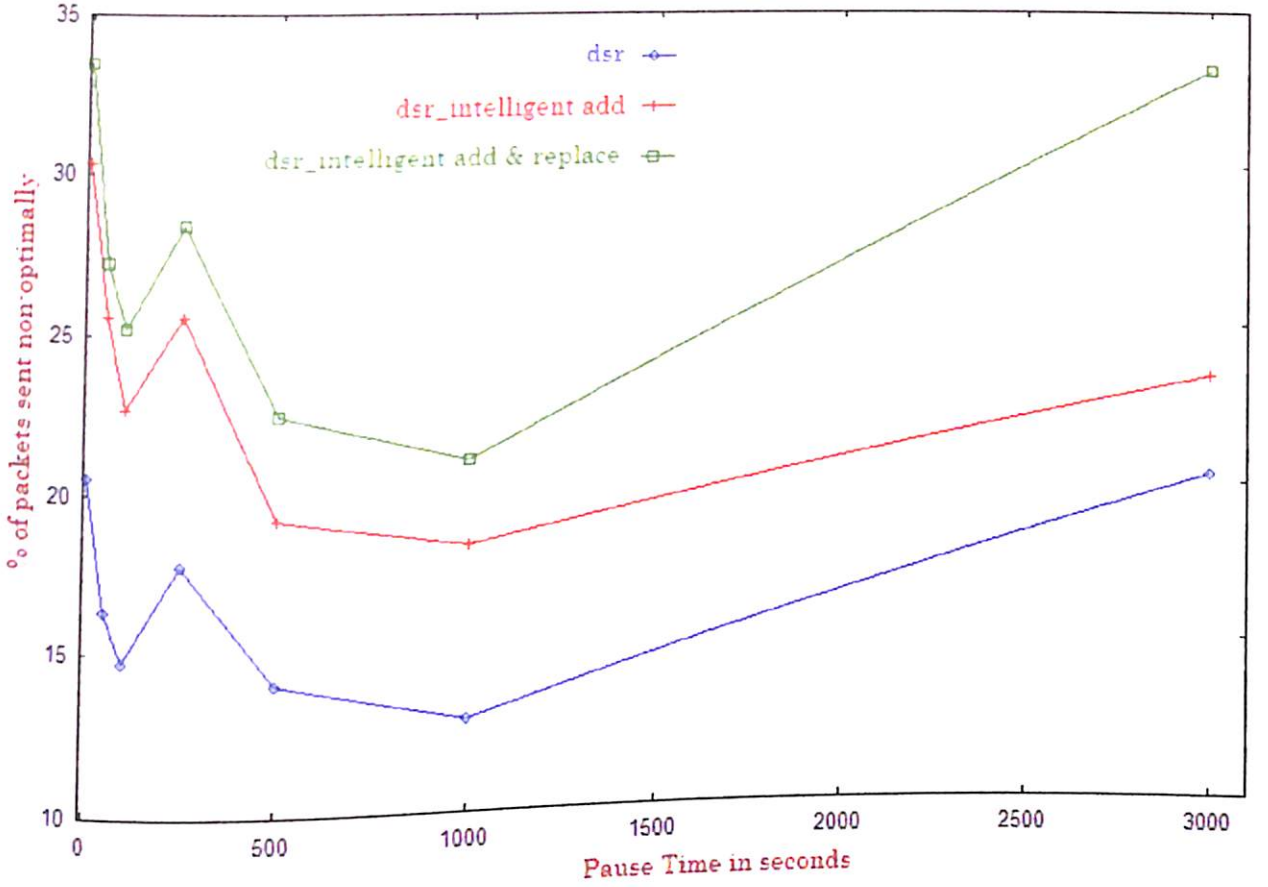


Fig 5.35 % of packets received non-optimally at a maximum node speed of 15m/s

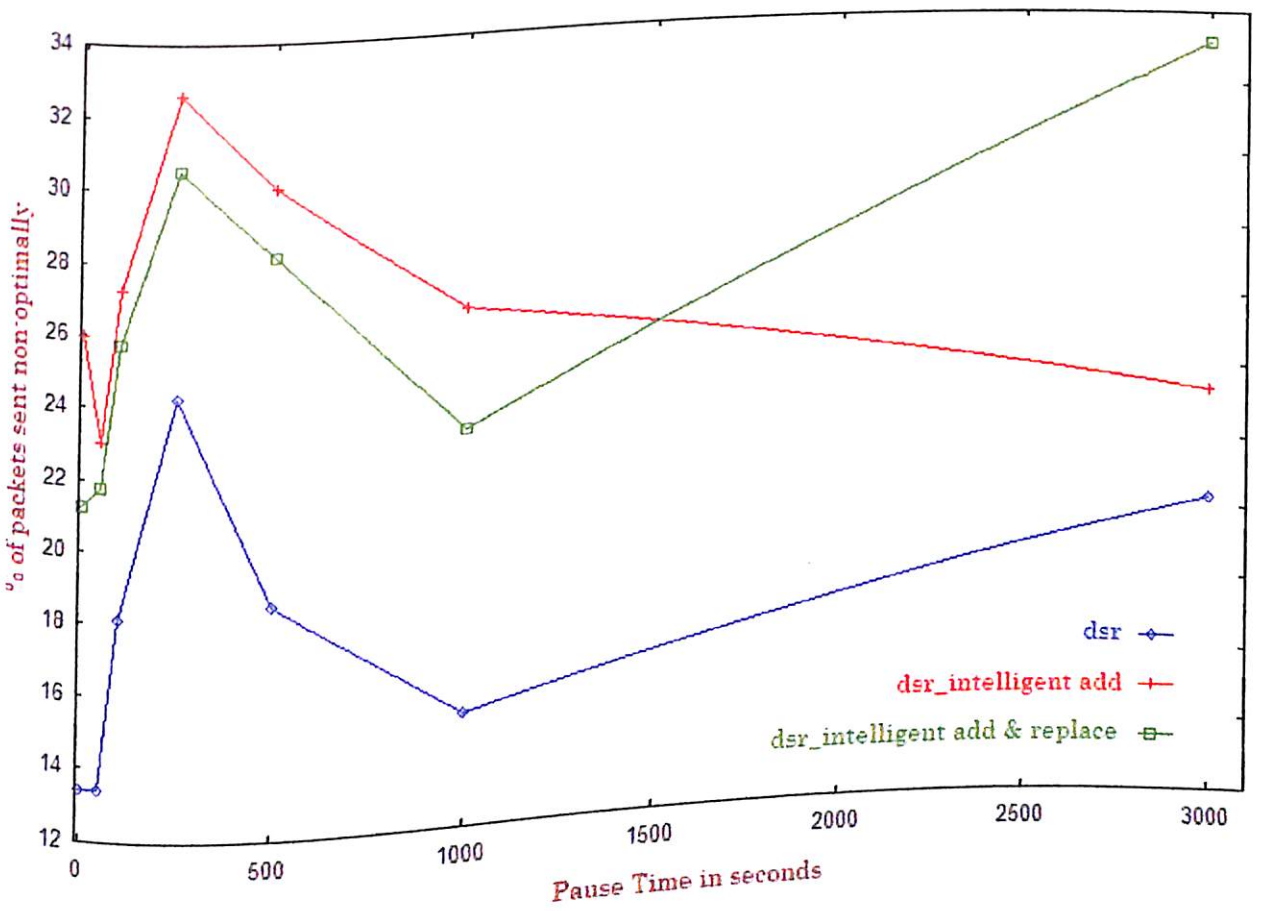


Fig 5.36 % of packets received non-optimally at a maximum node speed of 10m/s

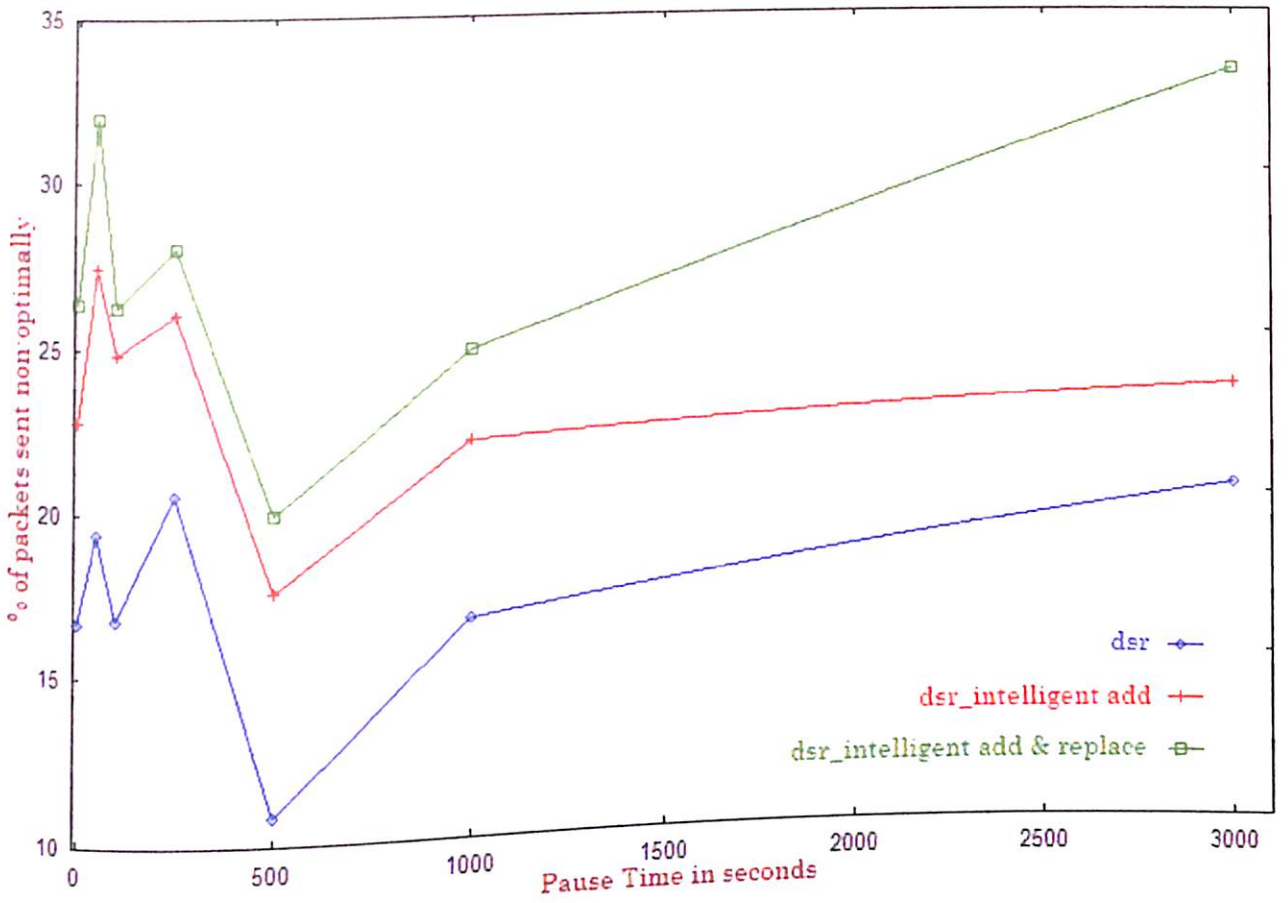


Fig 5.37 % of packets received non-optimally at a maximum node speed of 5m/s

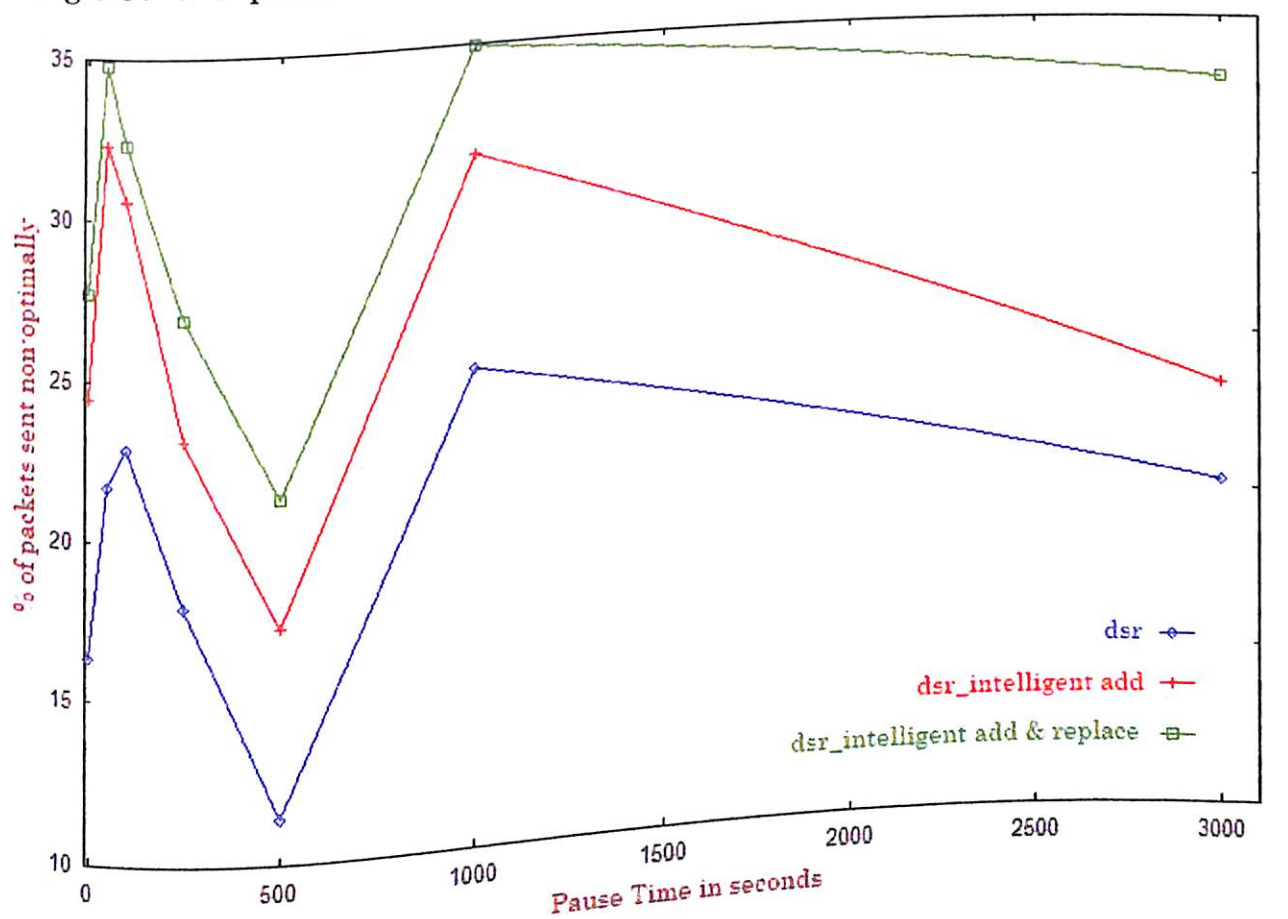


Fig 5.38 % of packets received non-optimally at a maximum node speed of 1m/s

Effect of Intelligent Caching Scheme on Network Performance
Cache Statistics
(fig. 5.39 - fig. 5.46)

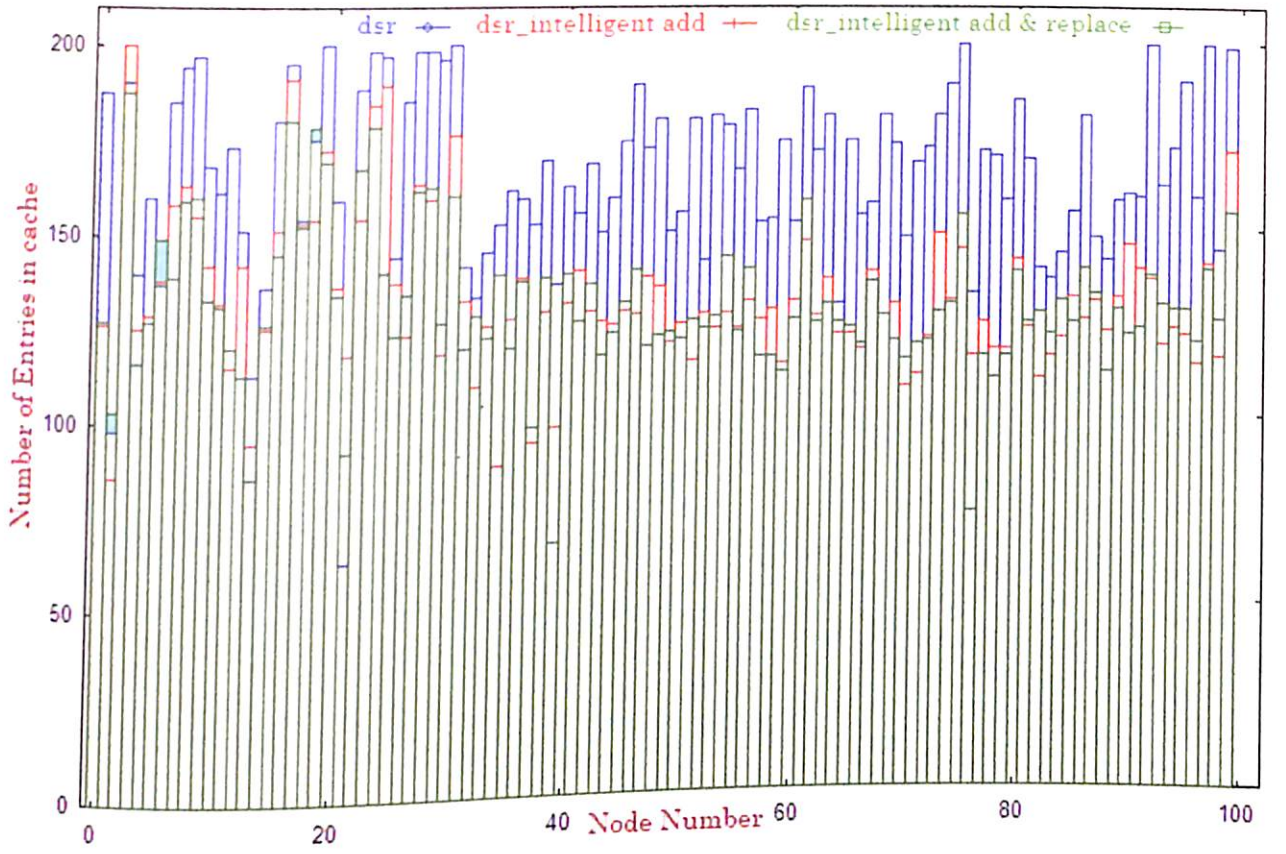


Fig 5.39 Number of Routes recorded in the primary and secondary cache of a node (at a pause time of 0 secs)

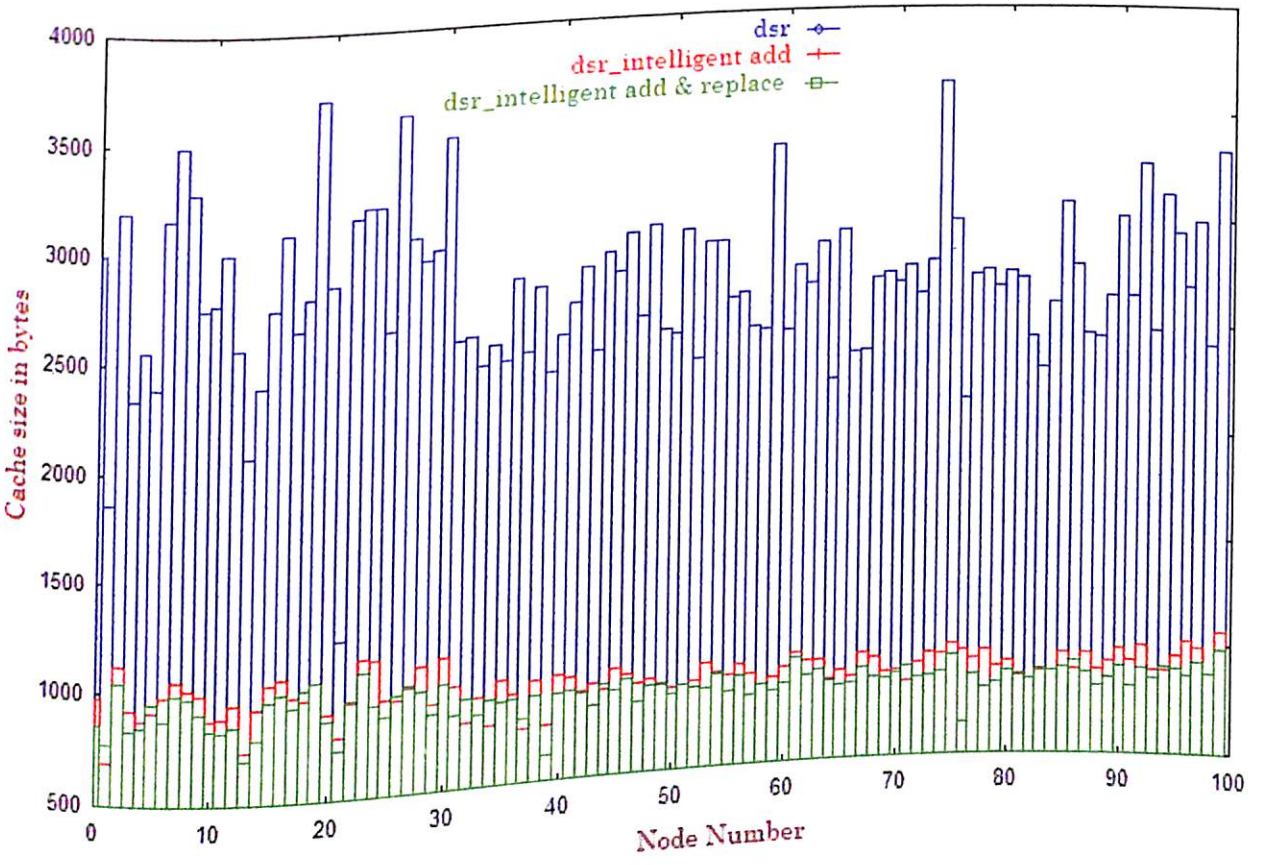


Fig 5.40 Size of the cache structures at each node in bytes (at a pause time of 0 secs)

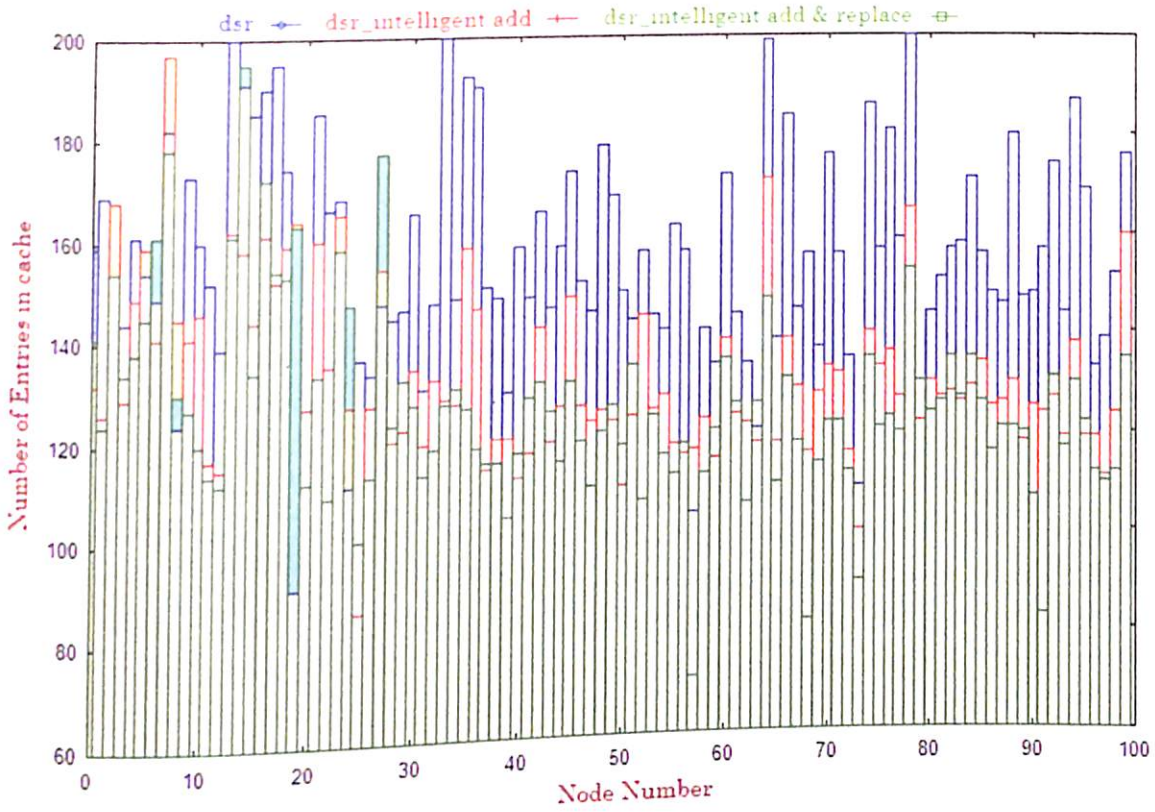


Fig 5.41 Number of Routes recorded in the primary and secondary cache of a node (at a pause time of 50 secs)

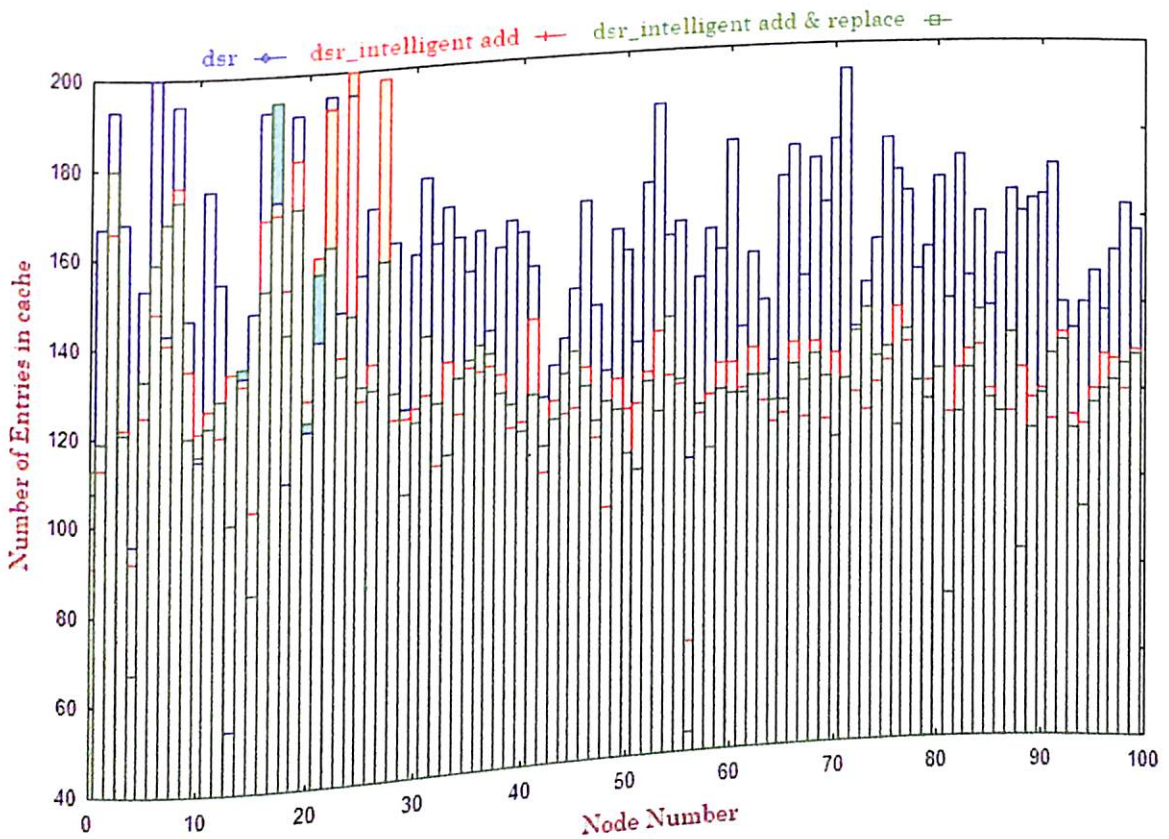


Fig 5.42 Number of Routes recorded in the primary and secondary cache of a node (at a pause time of 100 secs)

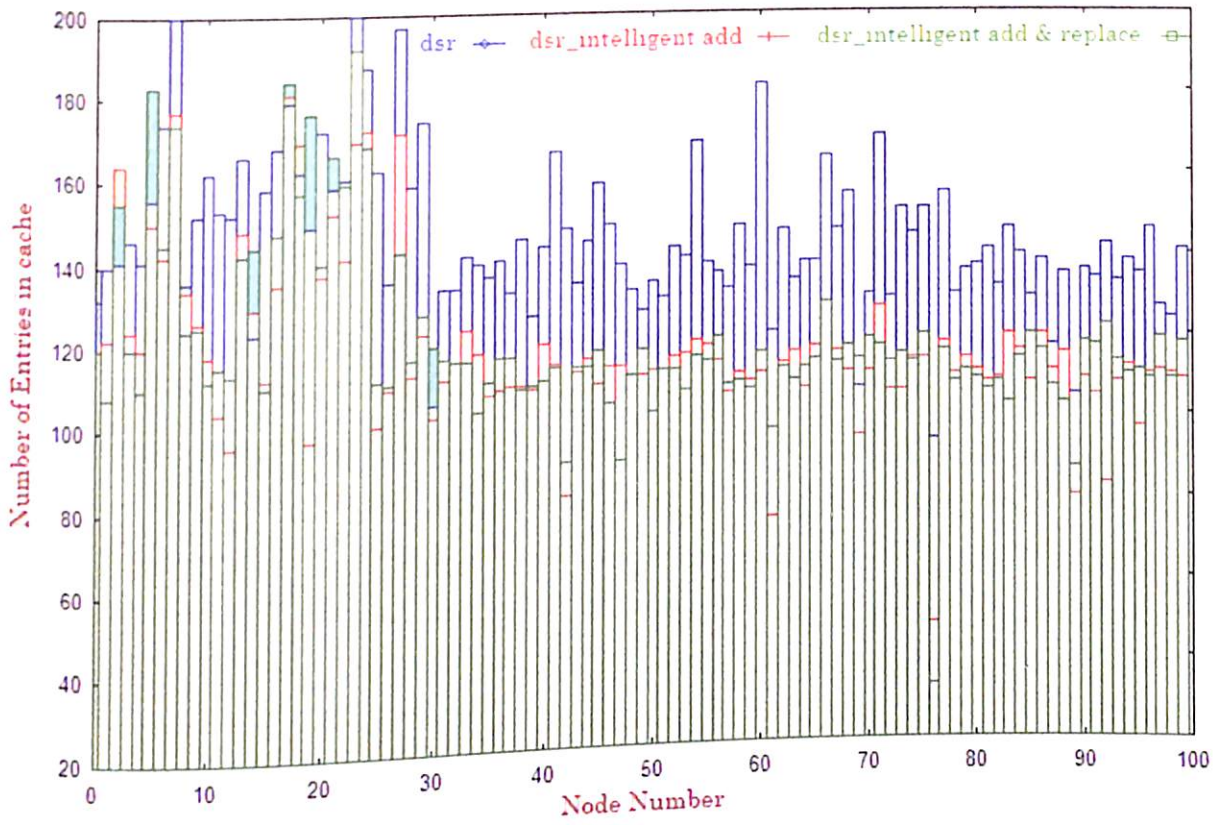


Fig 5.43 Number of Routes recorded in the primary and secondary cache of a node (at a pause time of 250 secs)

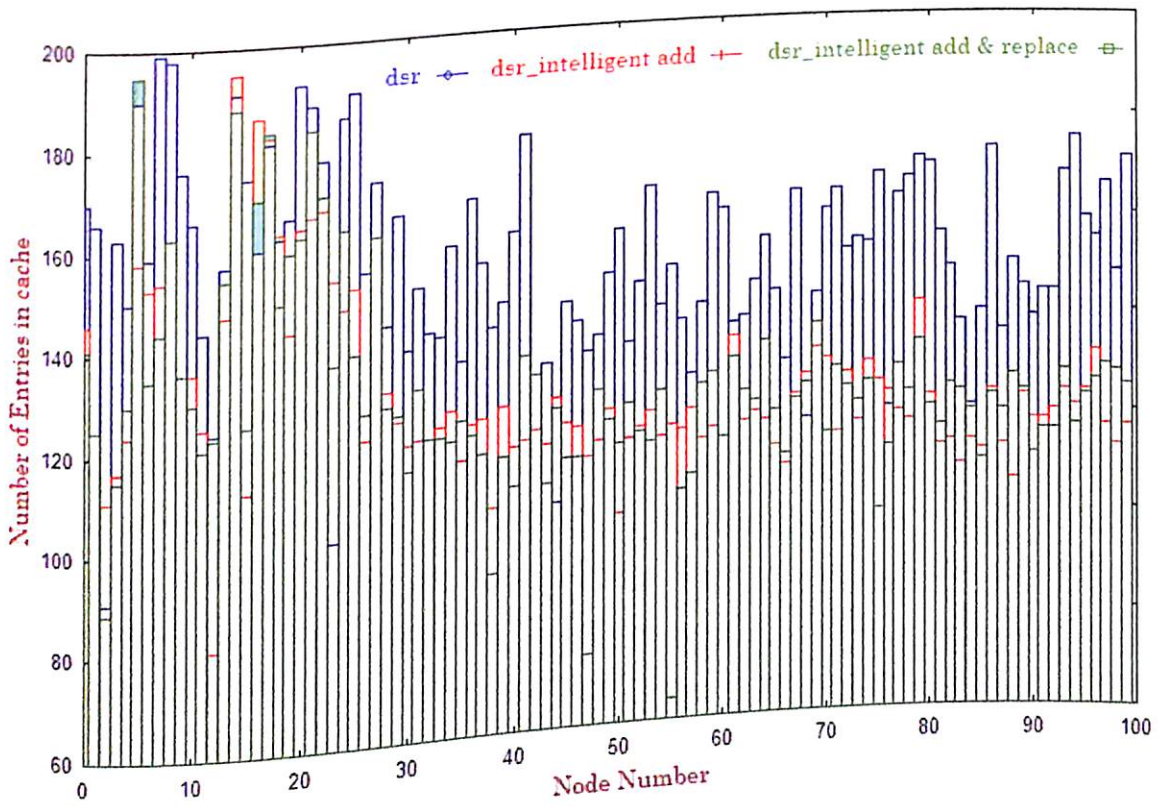


Fig 5.44 Number of Routes recorded in the primary and secondary cache of a node (at a pause time of 500 secs)

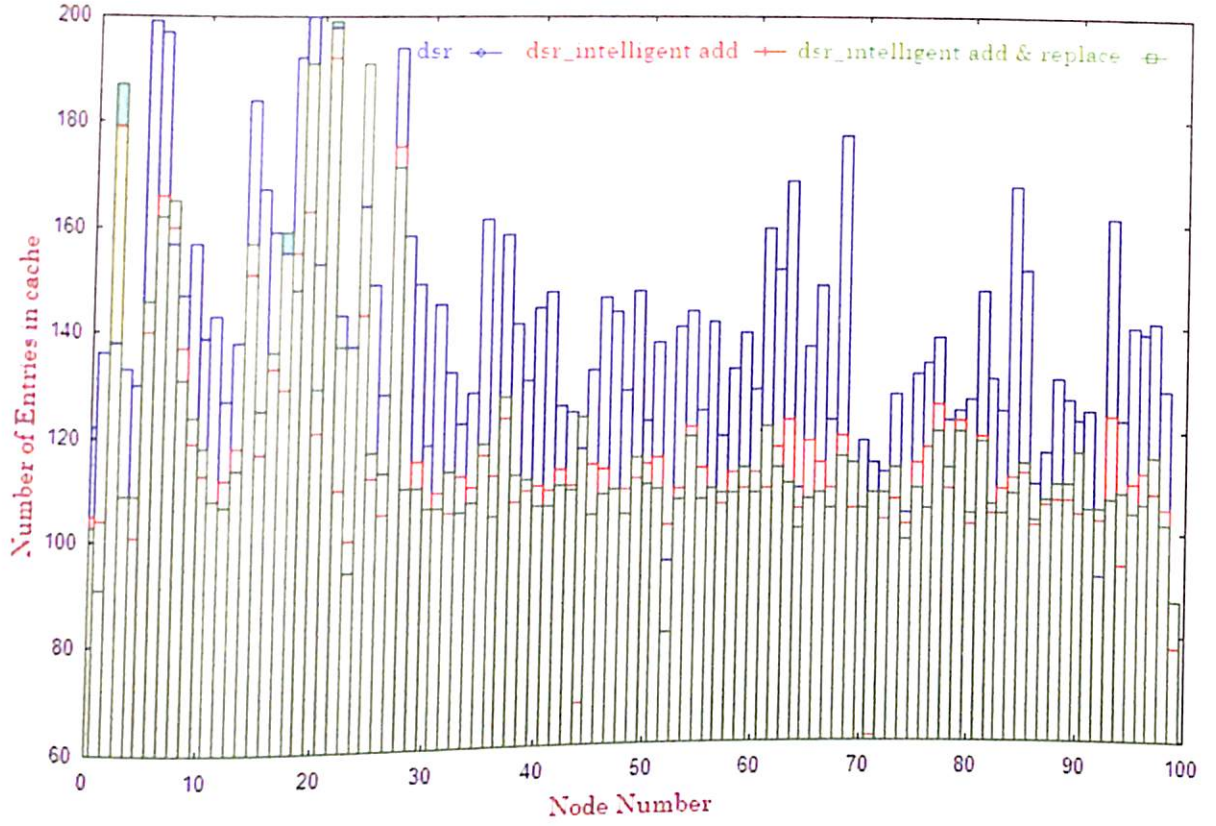


Fig 5.45 Number of Routes recorded in the primary and secondary cache of a node (at a pause time of 1000 secs)

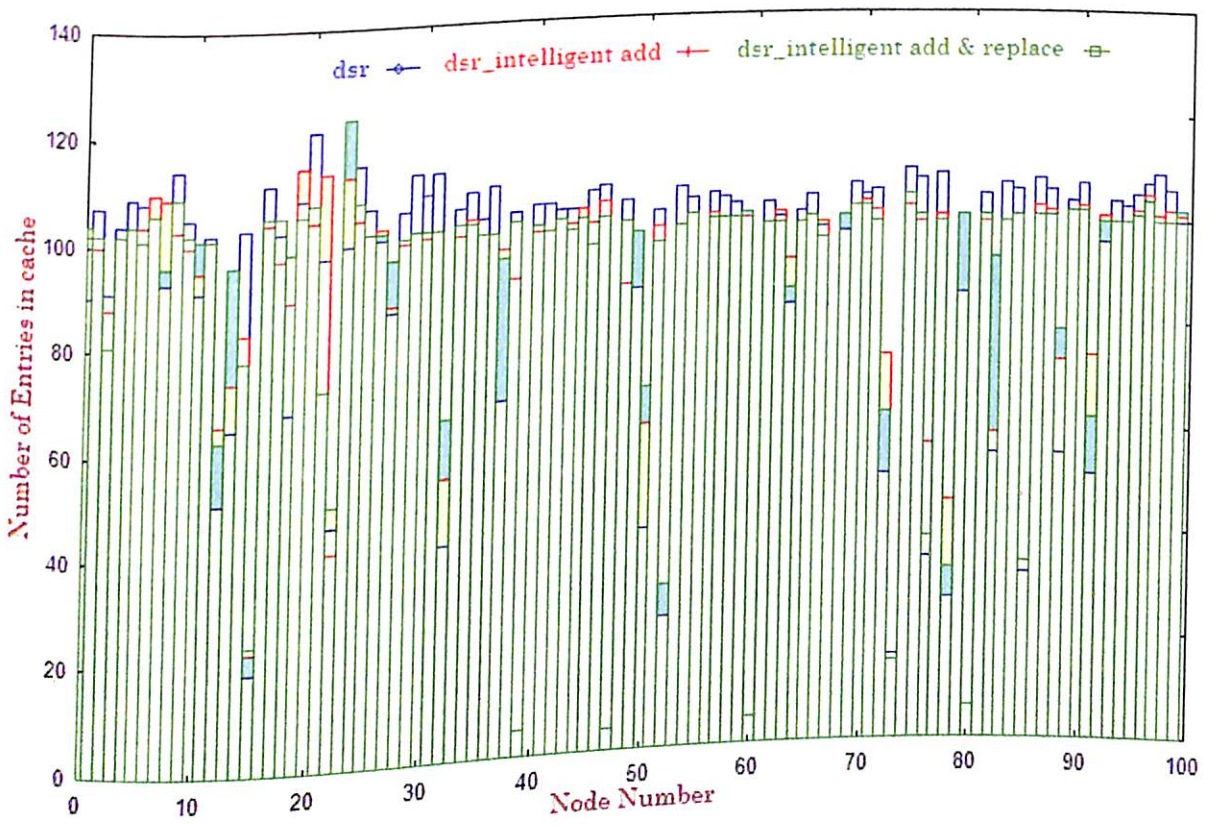


Fig 5.46 Number of Routes recorded in the primary and secondary cache of a node (at a pause time of 3000 secs)

Effect of Intelligent Caching Scheme on Network Performance

Analysis of Network Performance over Varying Time Periods

(fig. 5.47 - fig. 5.52)

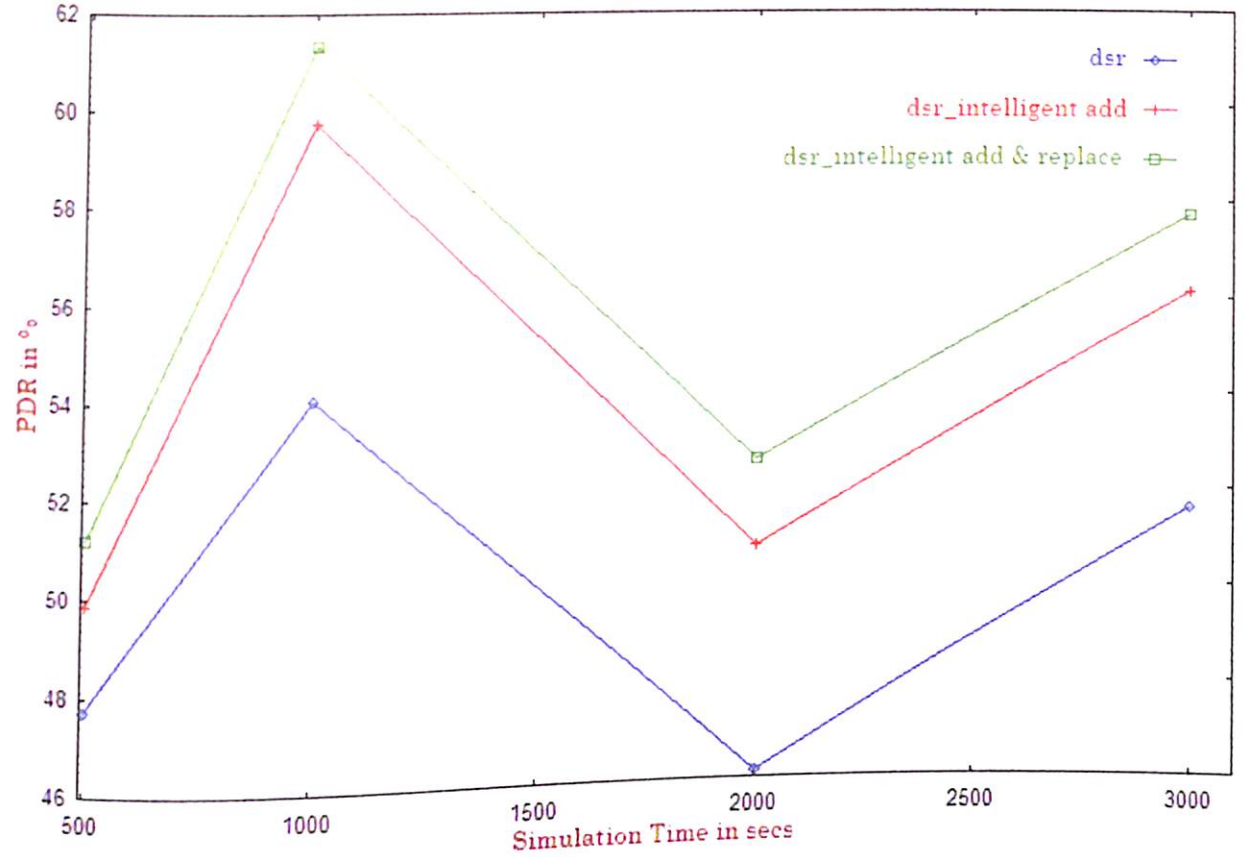


Fig 5.47 PDR Vs Simulation Time at a maximum node speed of 30m/s

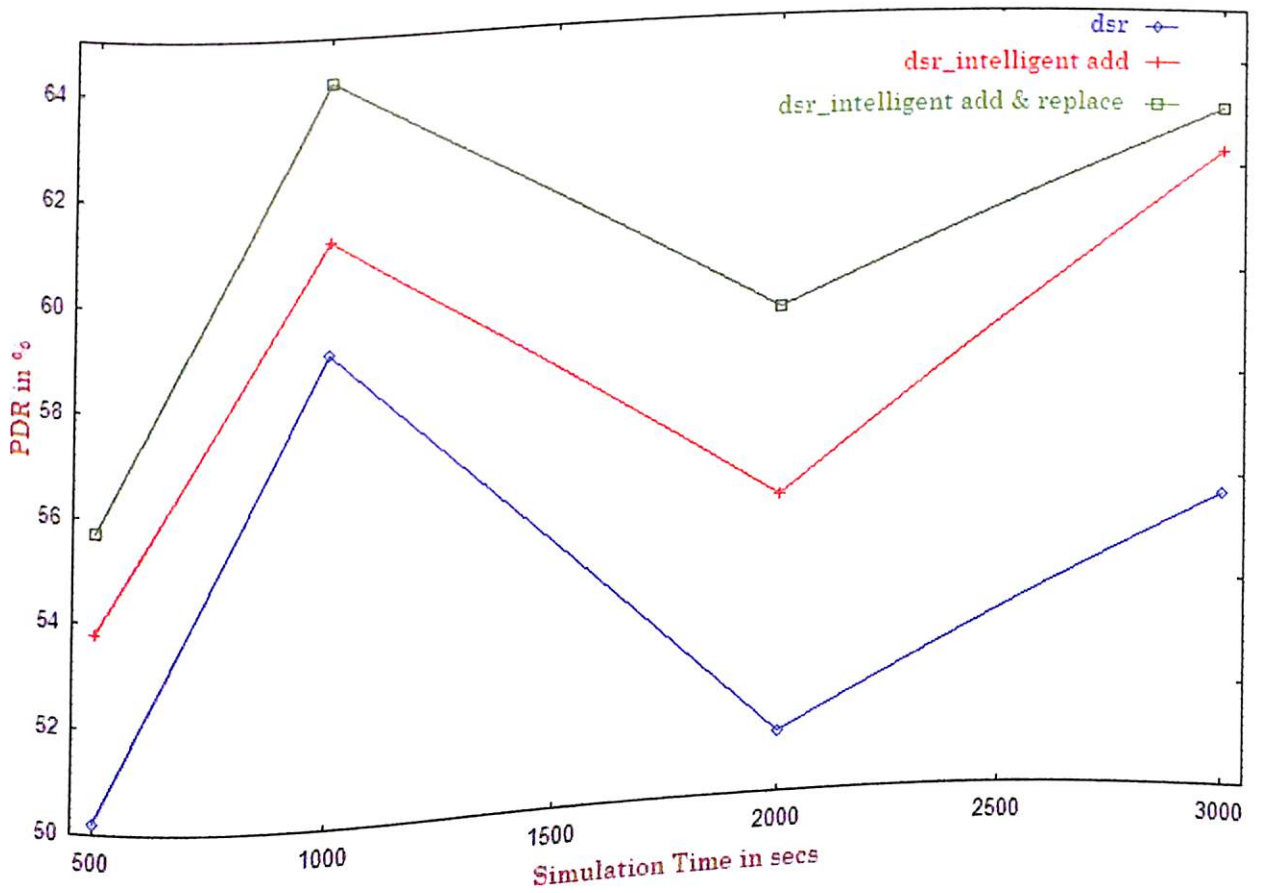


Fig 5.48 PDR Vs Simulation Time at a maximum node speed of 20m/s

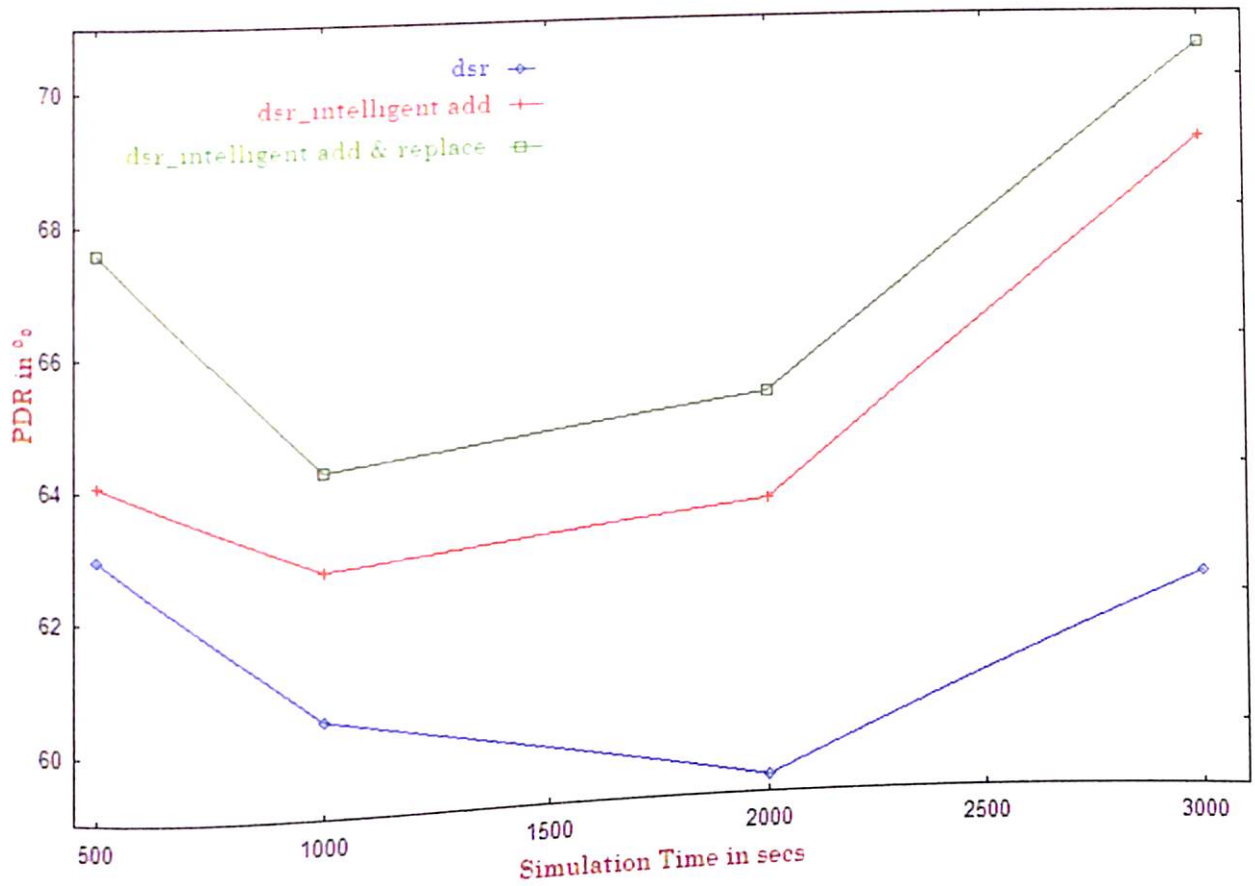


Fig 5.49 PDR Vs Simulation Time at a maximum node speed of 15m/s

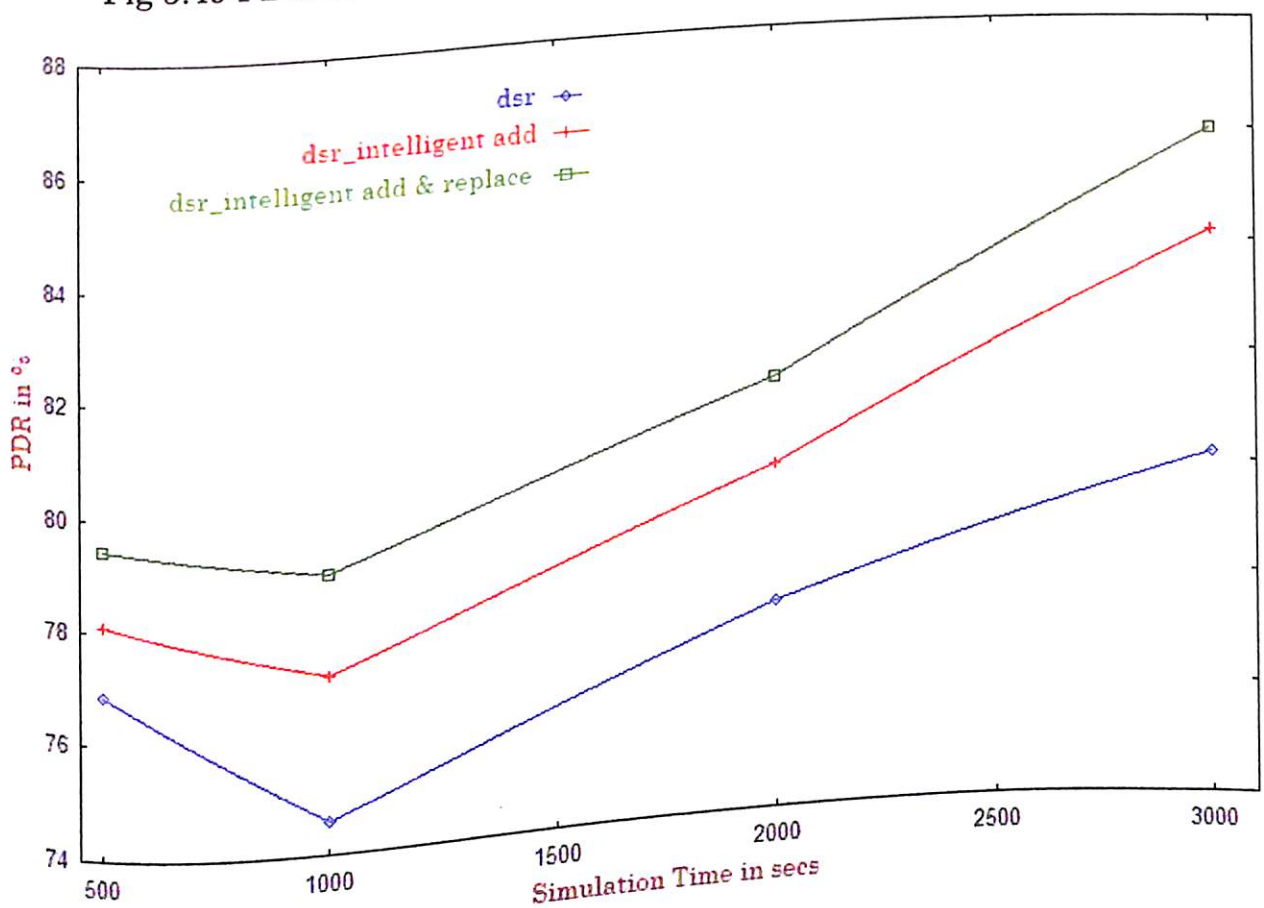


Fig 5.50 PDR Vs Simulation Time at a maximum node speed of 10m/s

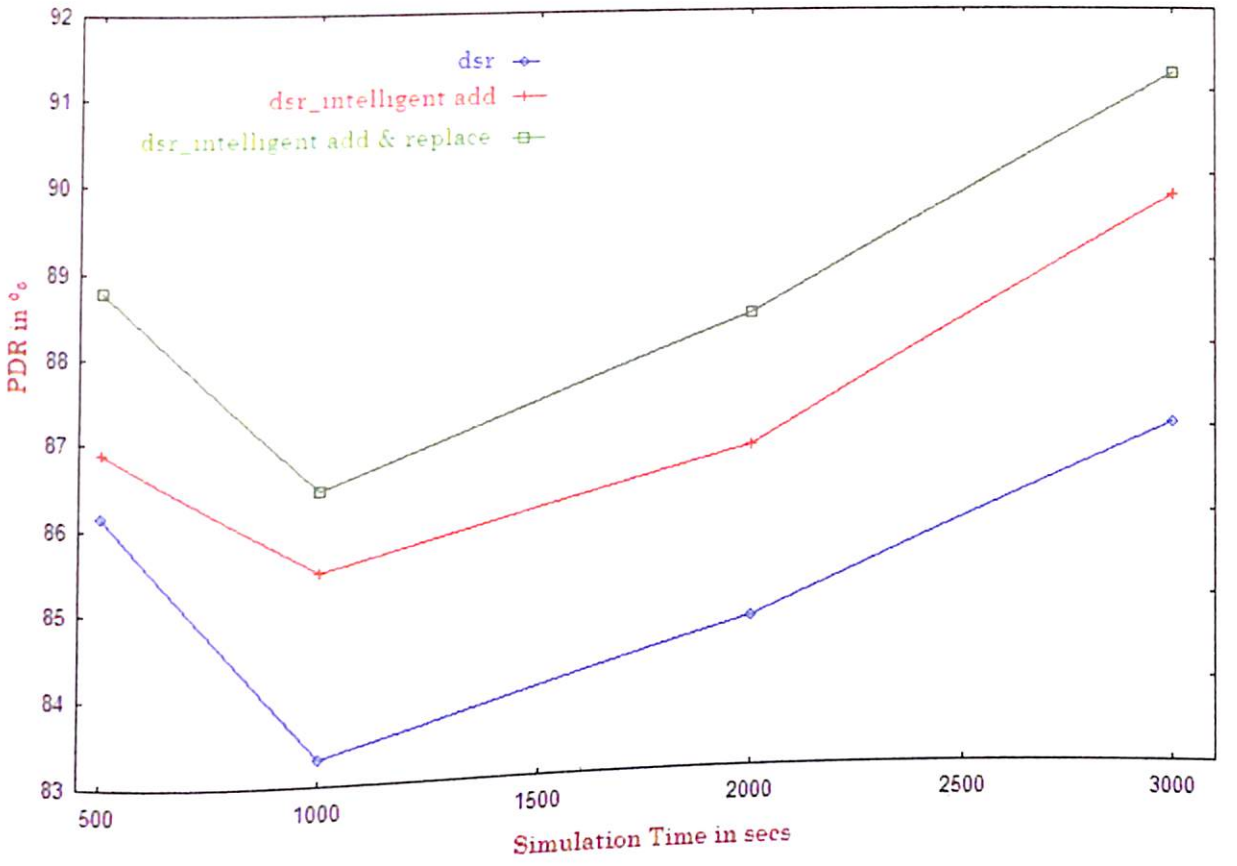


Fig 5.51 PDR Vs Simulation Time at a maximum node speed of 5m/s

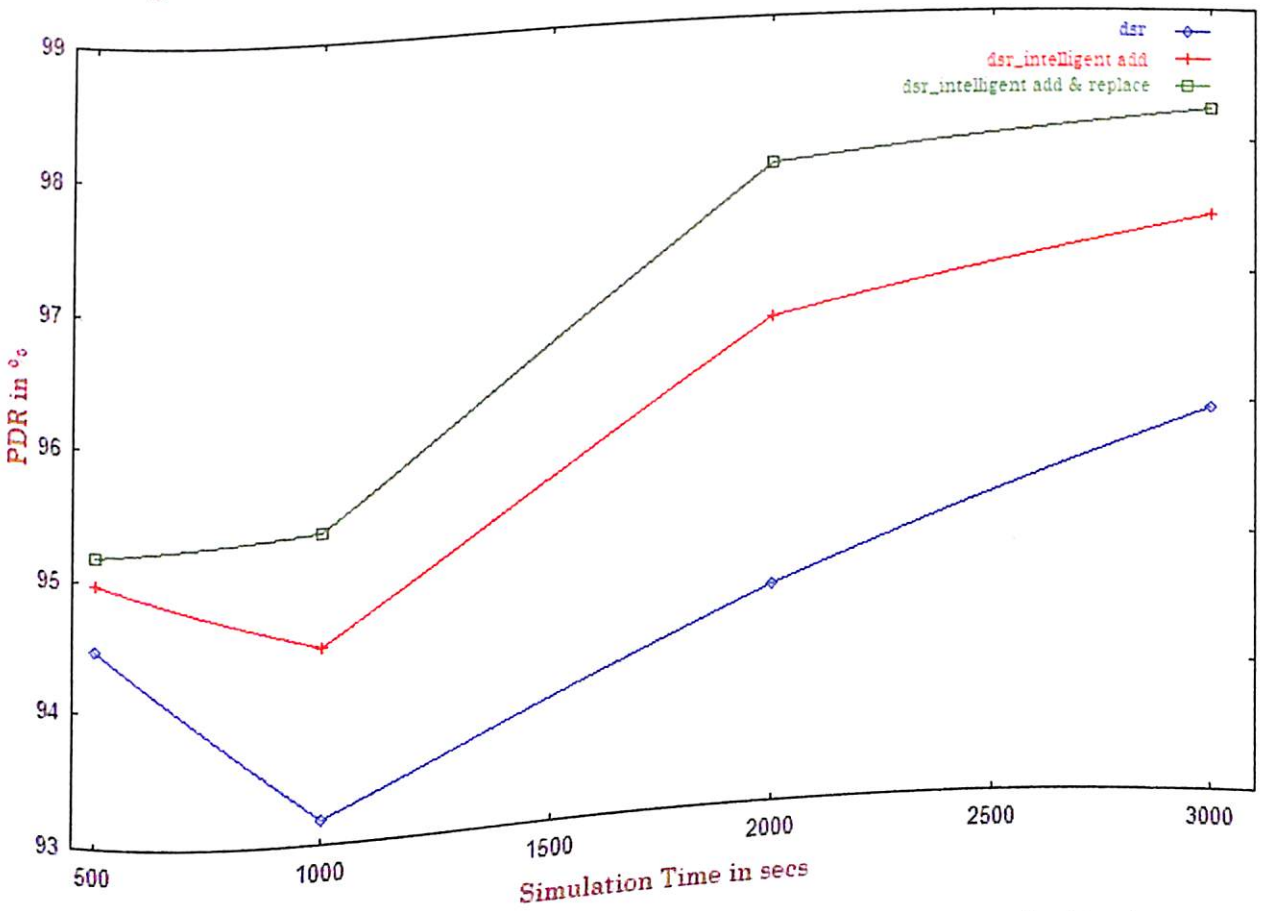


Fig 5.52 PDR Vs Simulation Time at a maximum node speed of 1m/s

Effect of Routing Criteria on Network Performance
Packet Deliver Ratio Analysis
(fig. 5.53 - fig. 5.58)

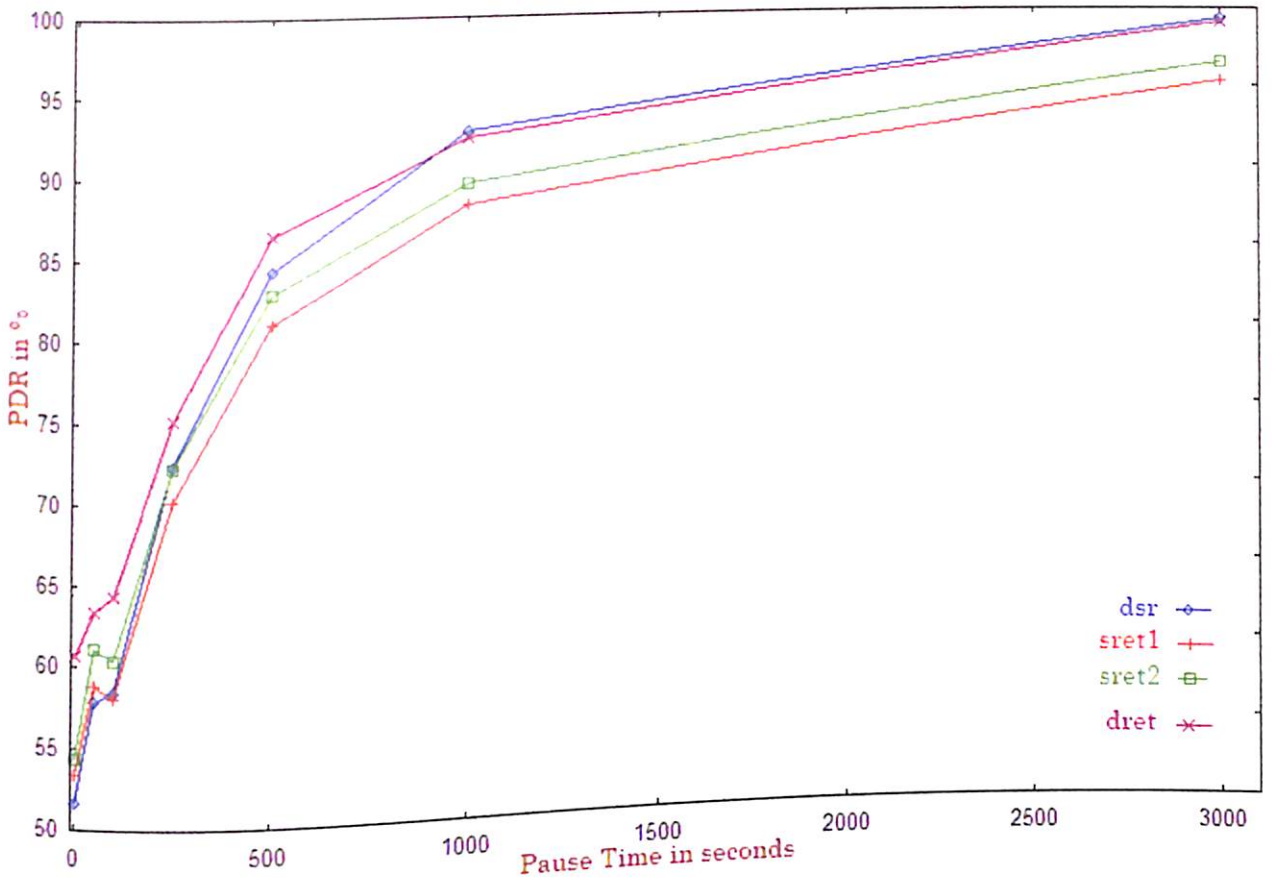


Fig 5.53 PDR Vs Pause Time at a maximum node speed of 30m/s

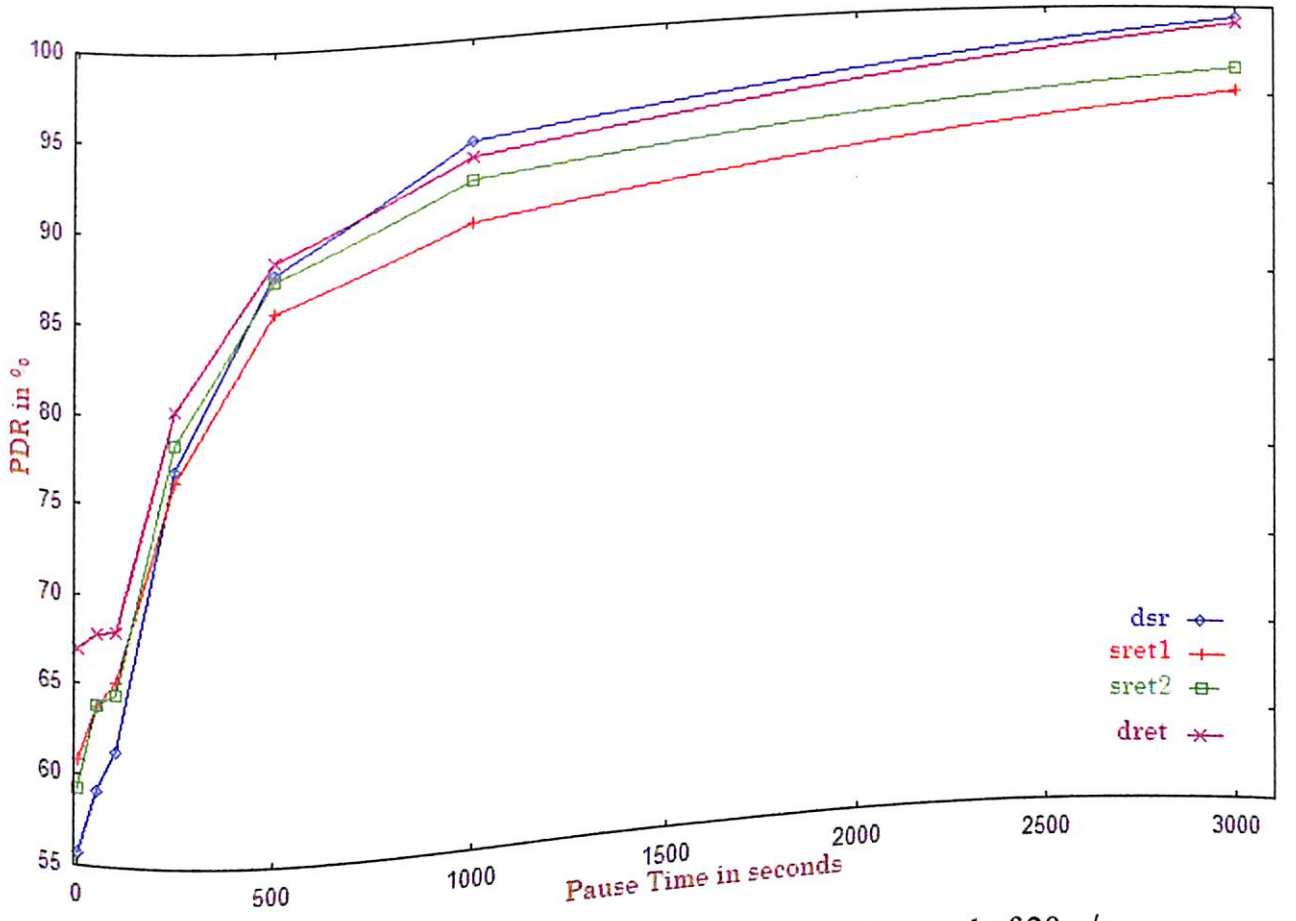


Fig 5.54 PDR Vs Pause Time at a maximum node speed of 20m/s

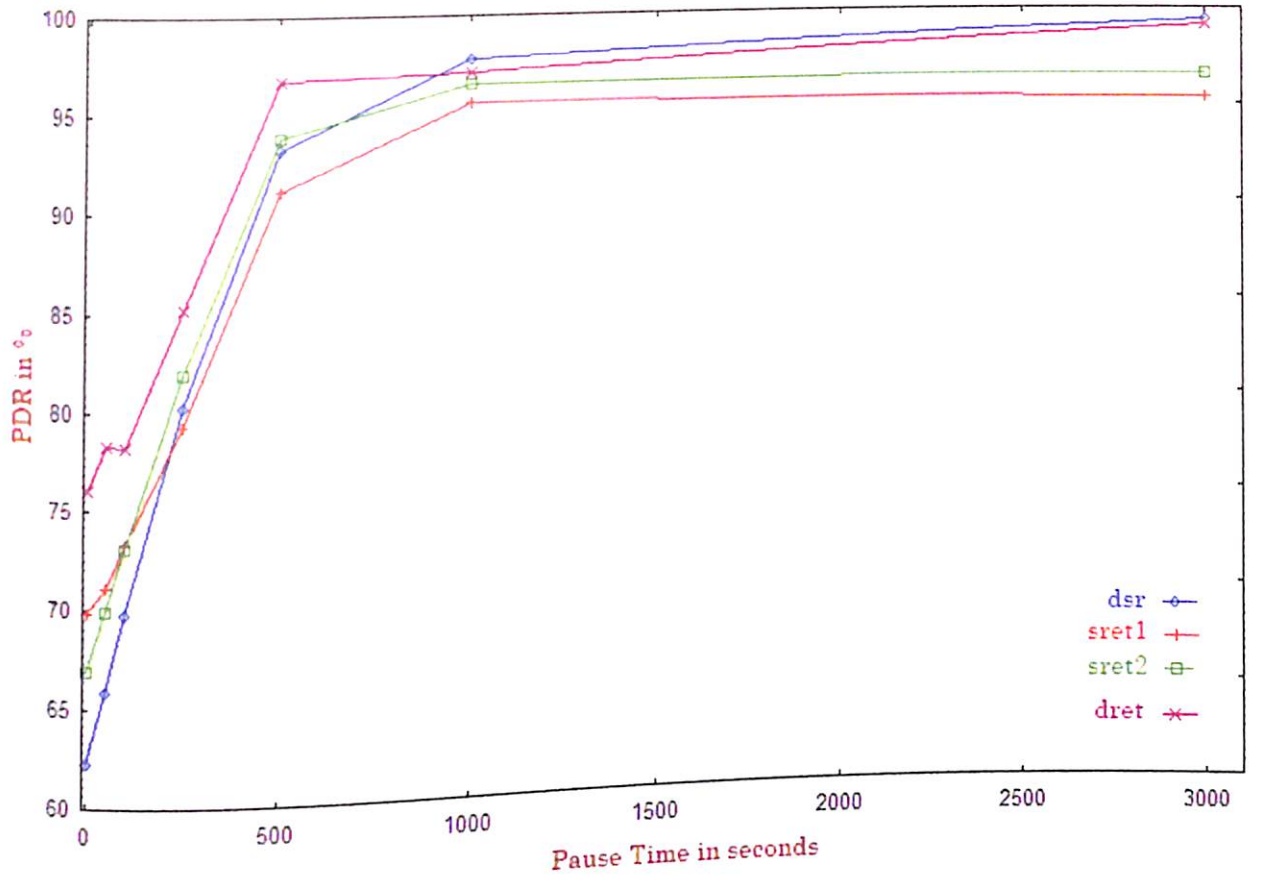


Fig 5.55 PDR Vs Pause Time at a maximum node speed of 15m/s

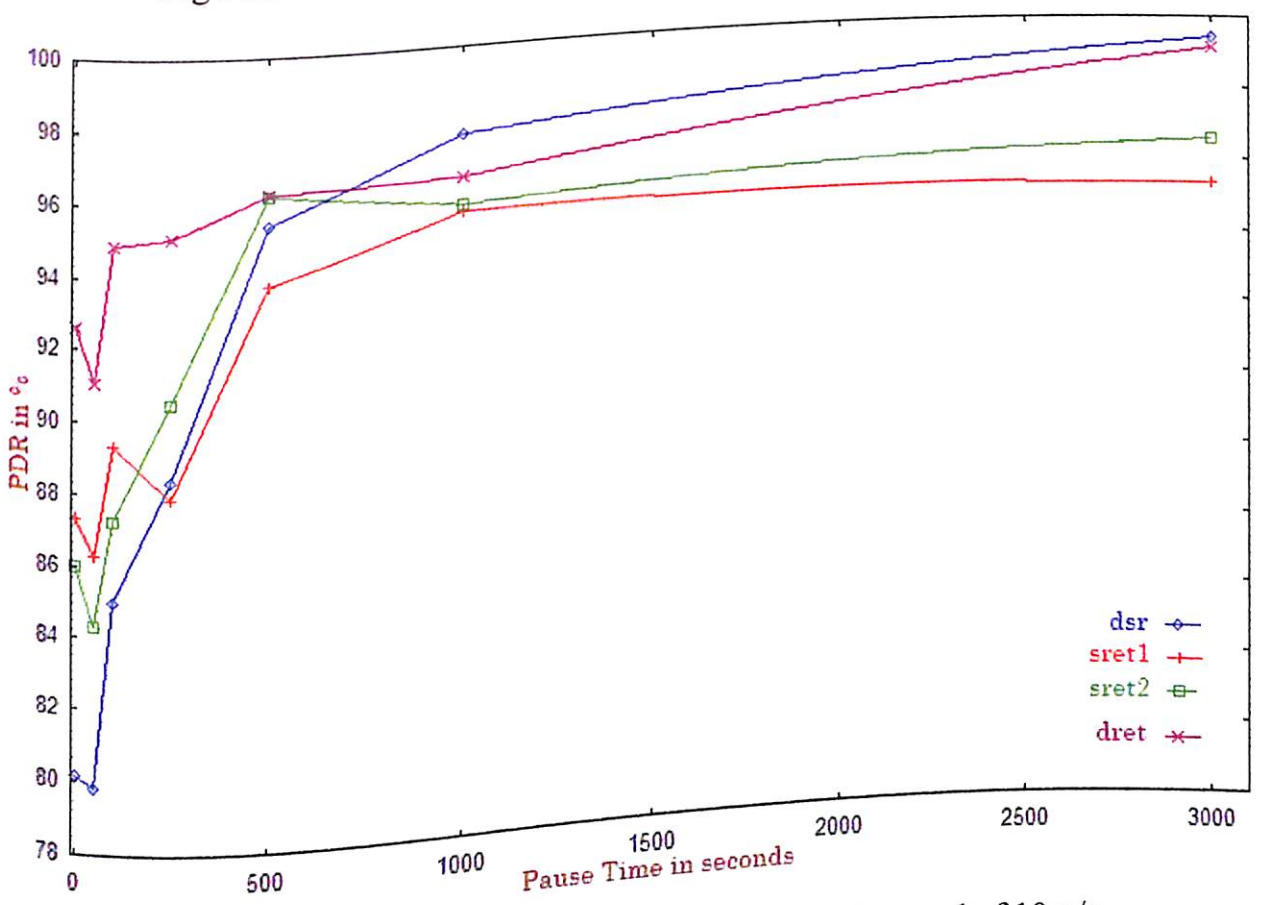


Fig 5.56 PDR Vs Pause Time at a maximum node speed of 10m/s

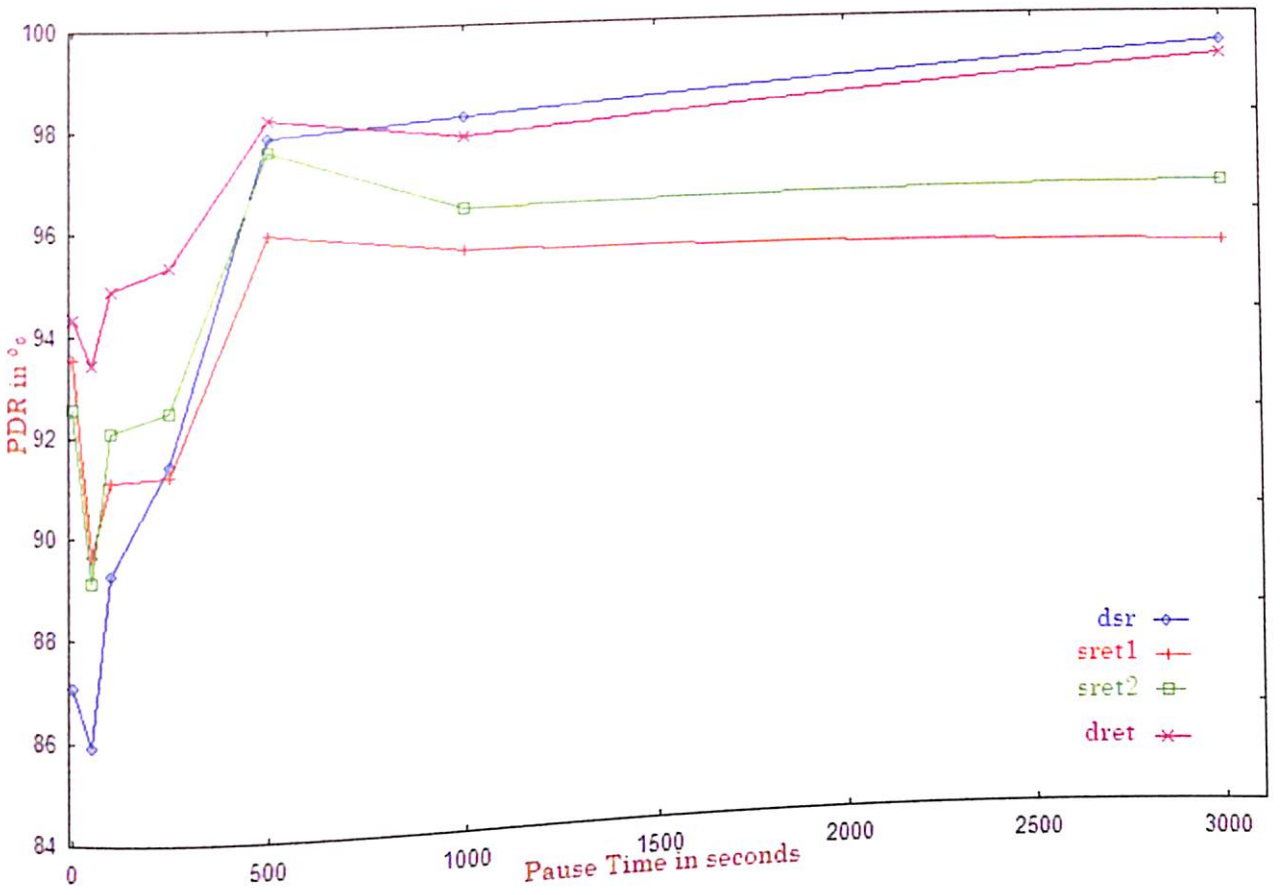


Fig 5.57 PDR Vs Pause Time at a maximum node speed of 5m/s

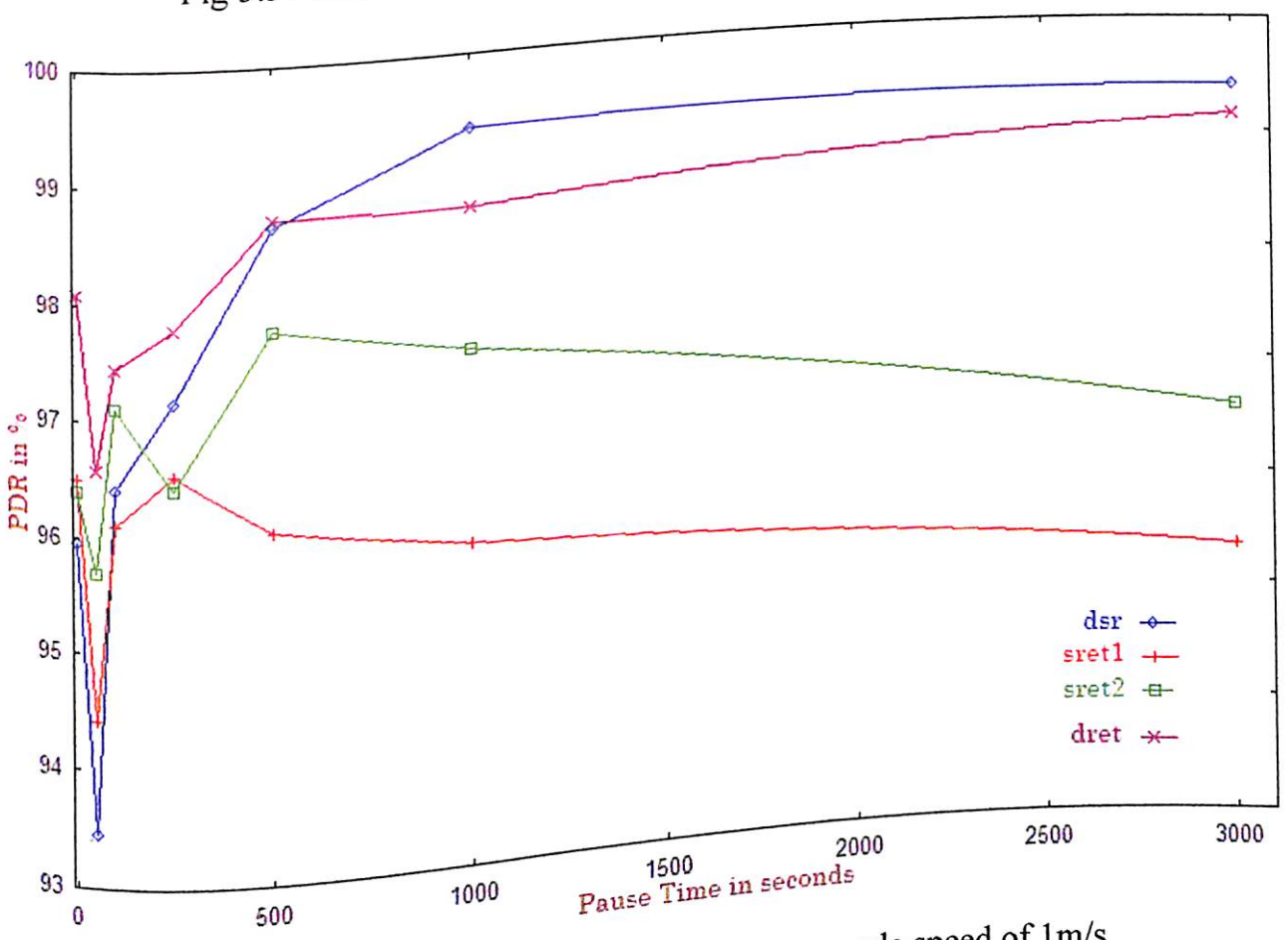


Fig 5.58 PDR Vs Pause Time at a maximum node speed of 1m/s

Effect of Routing Criteria on Network Performance Path Optimality Analysis (fig. 5.59 - fig. 5.64)

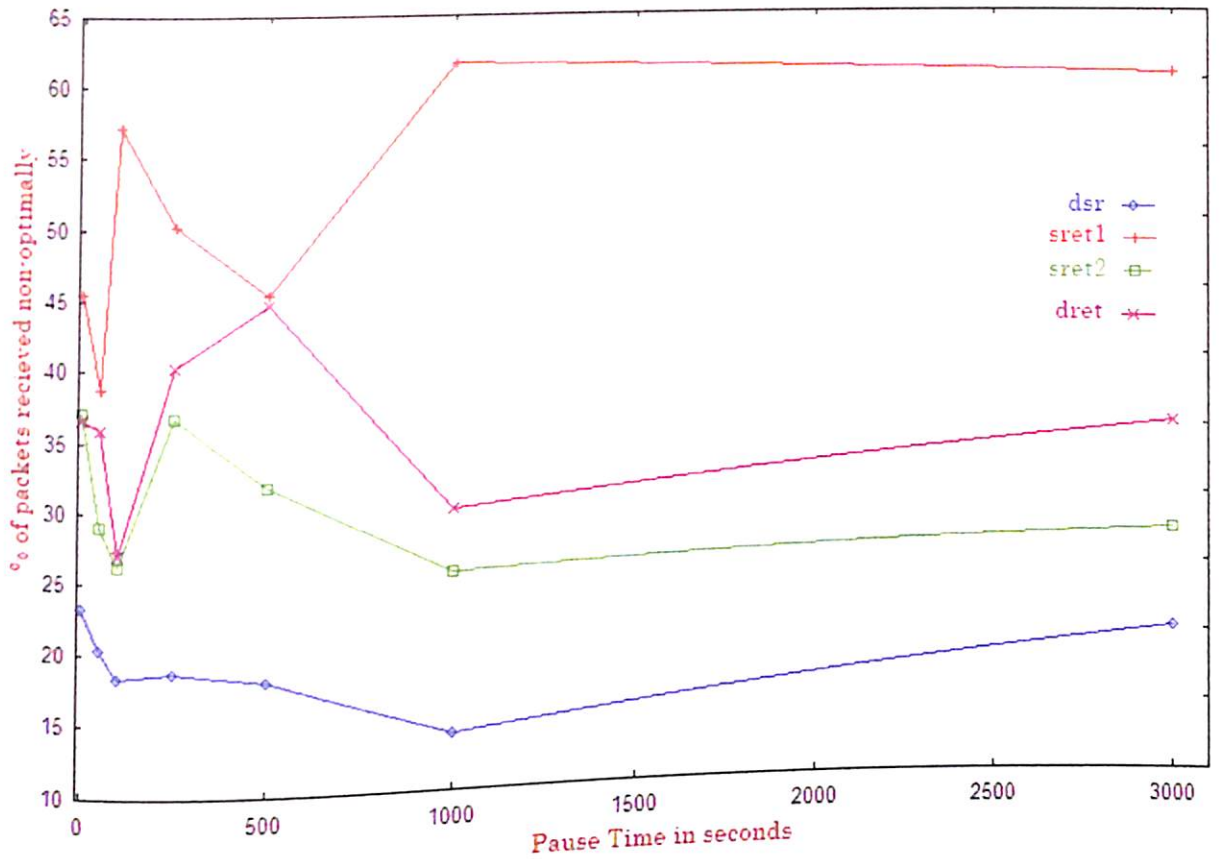


Fig 5.59 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 30m/s

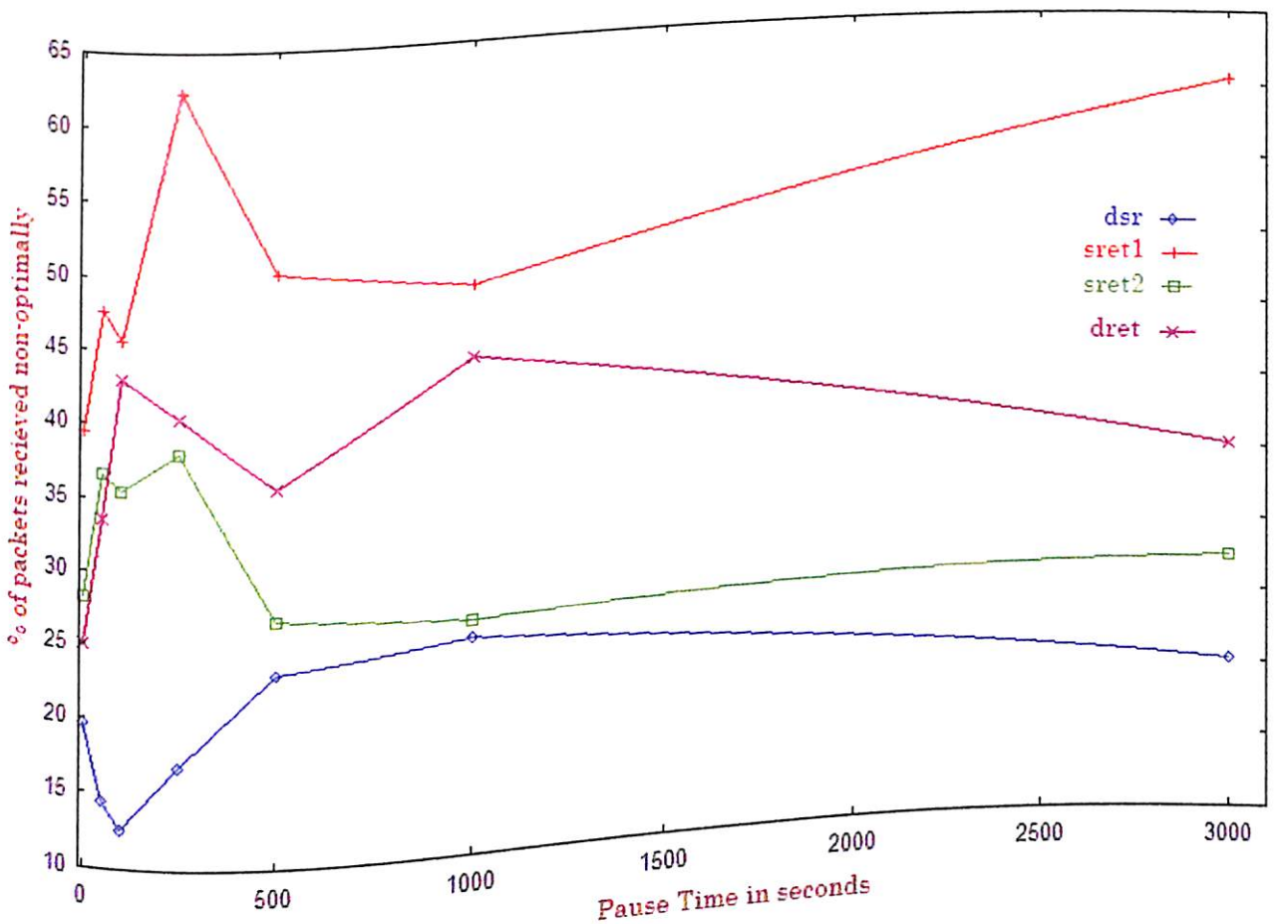


Fig 5.60 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 20m/s

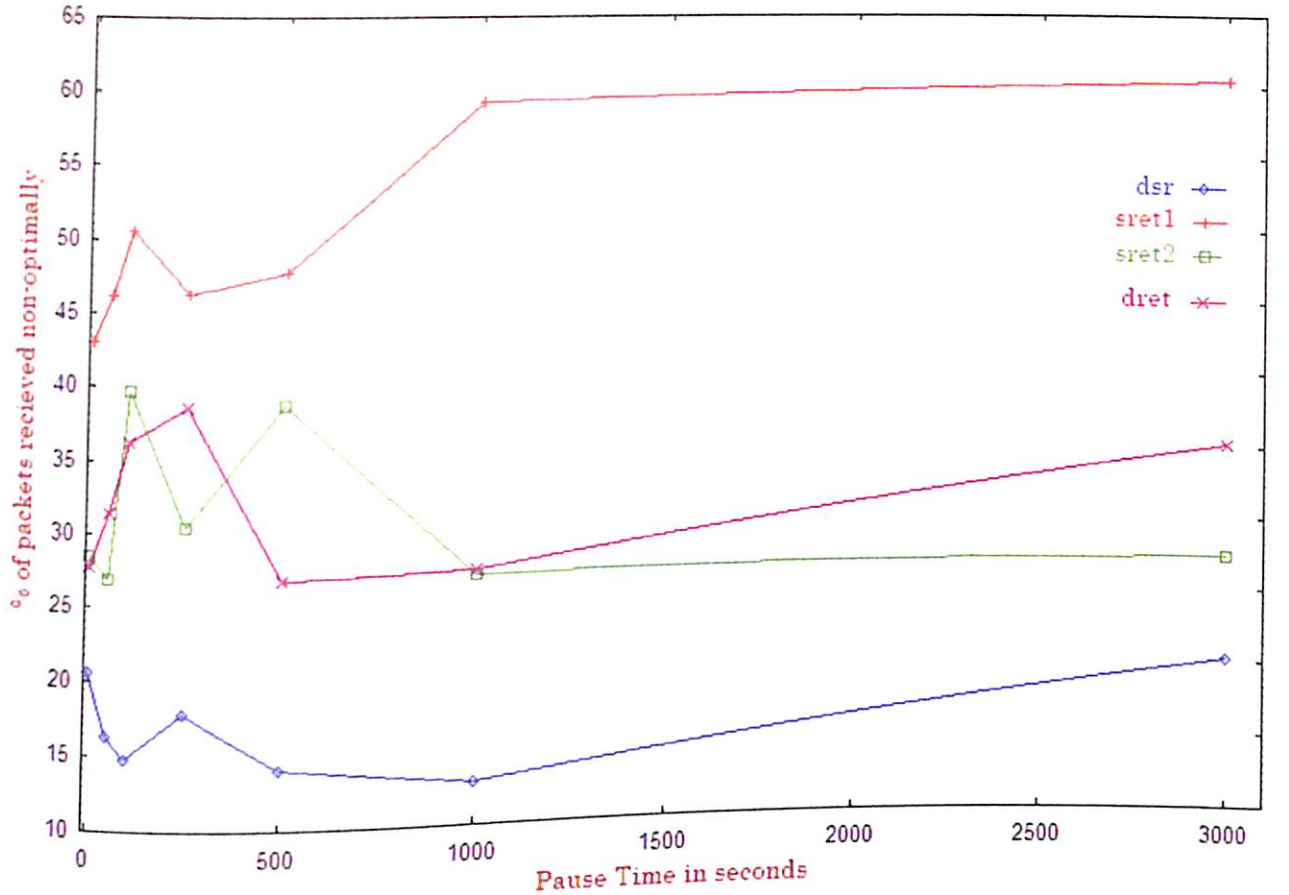


Fig 5.61 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 15m/s

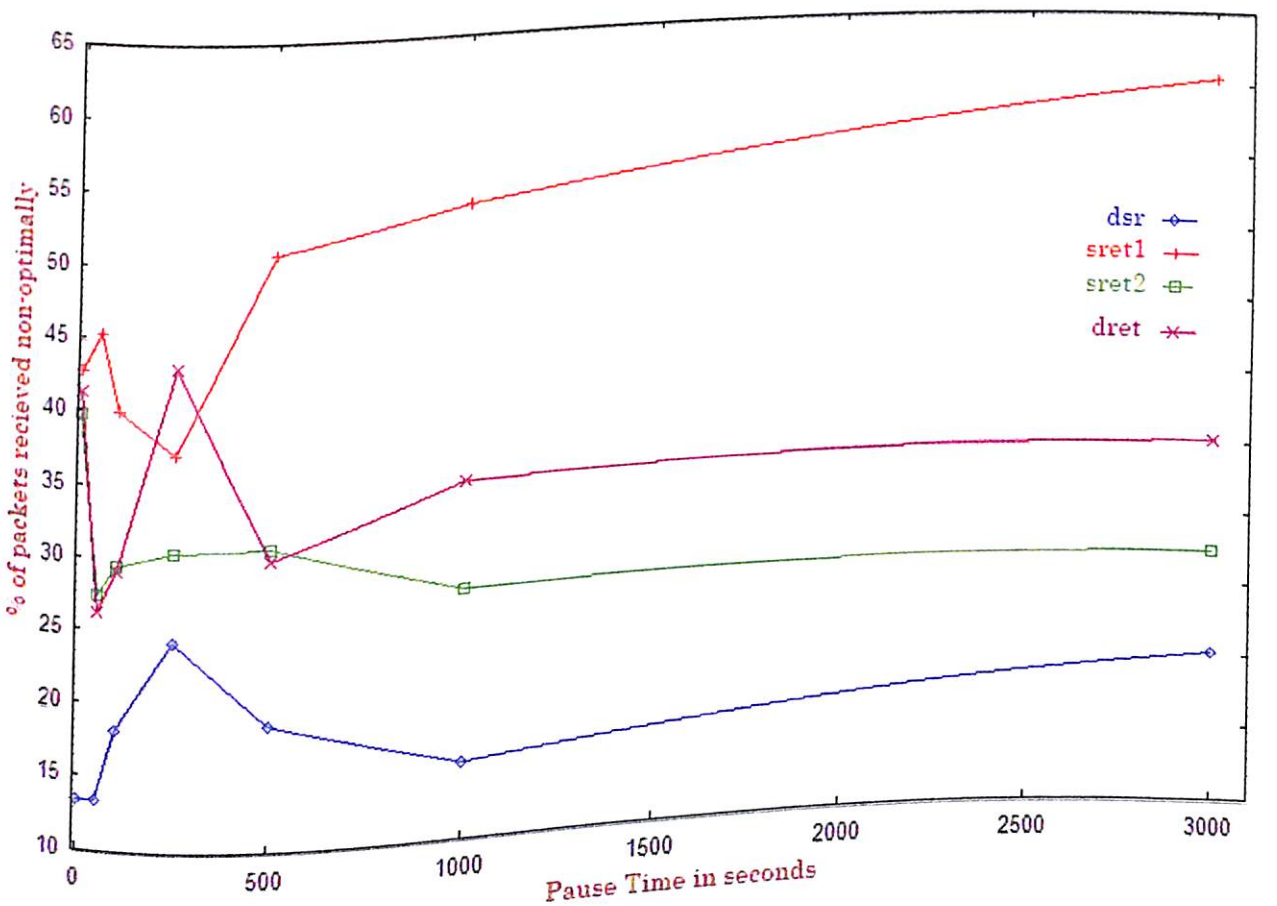


Fig 5.62 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 10m/s

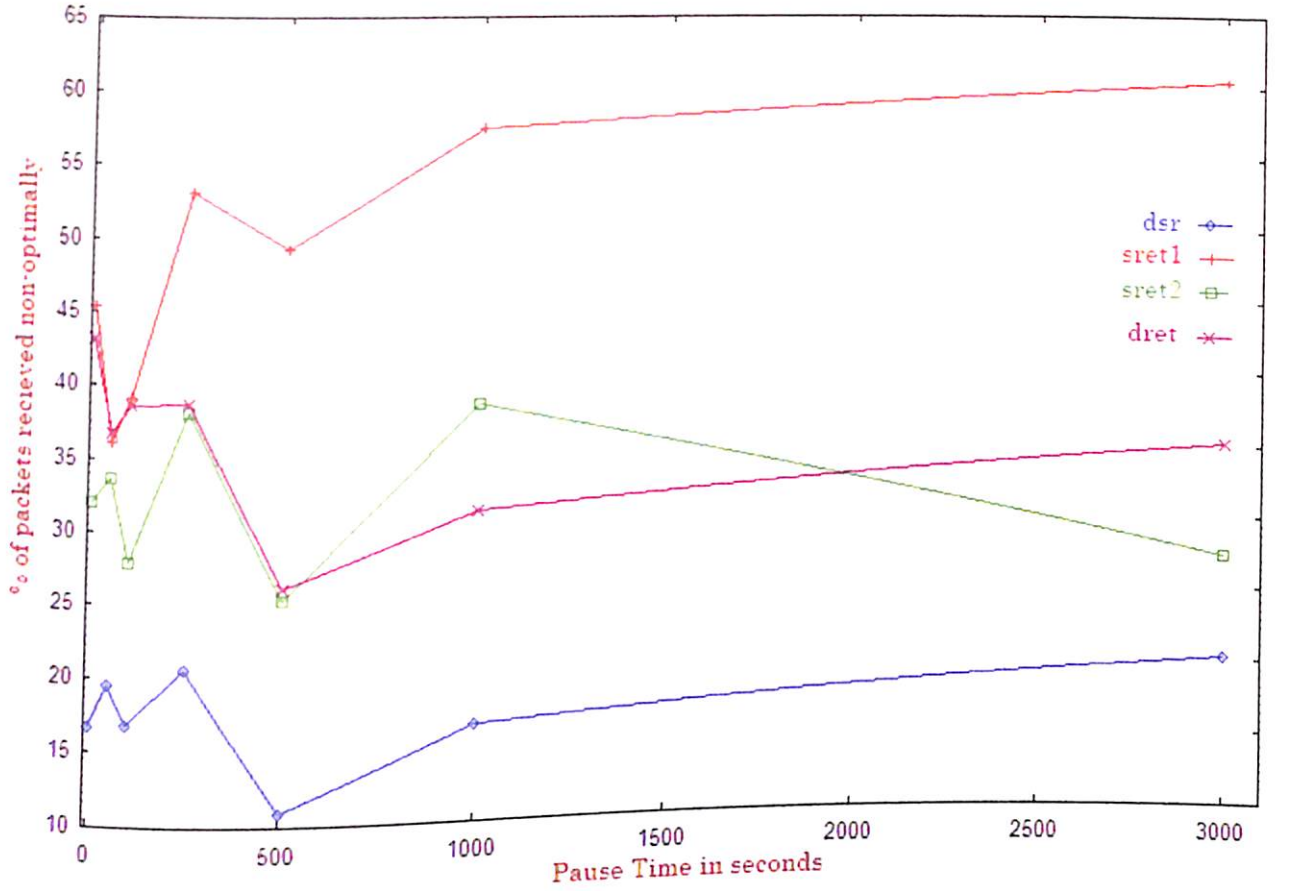


Fig 5.63 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 5m/s

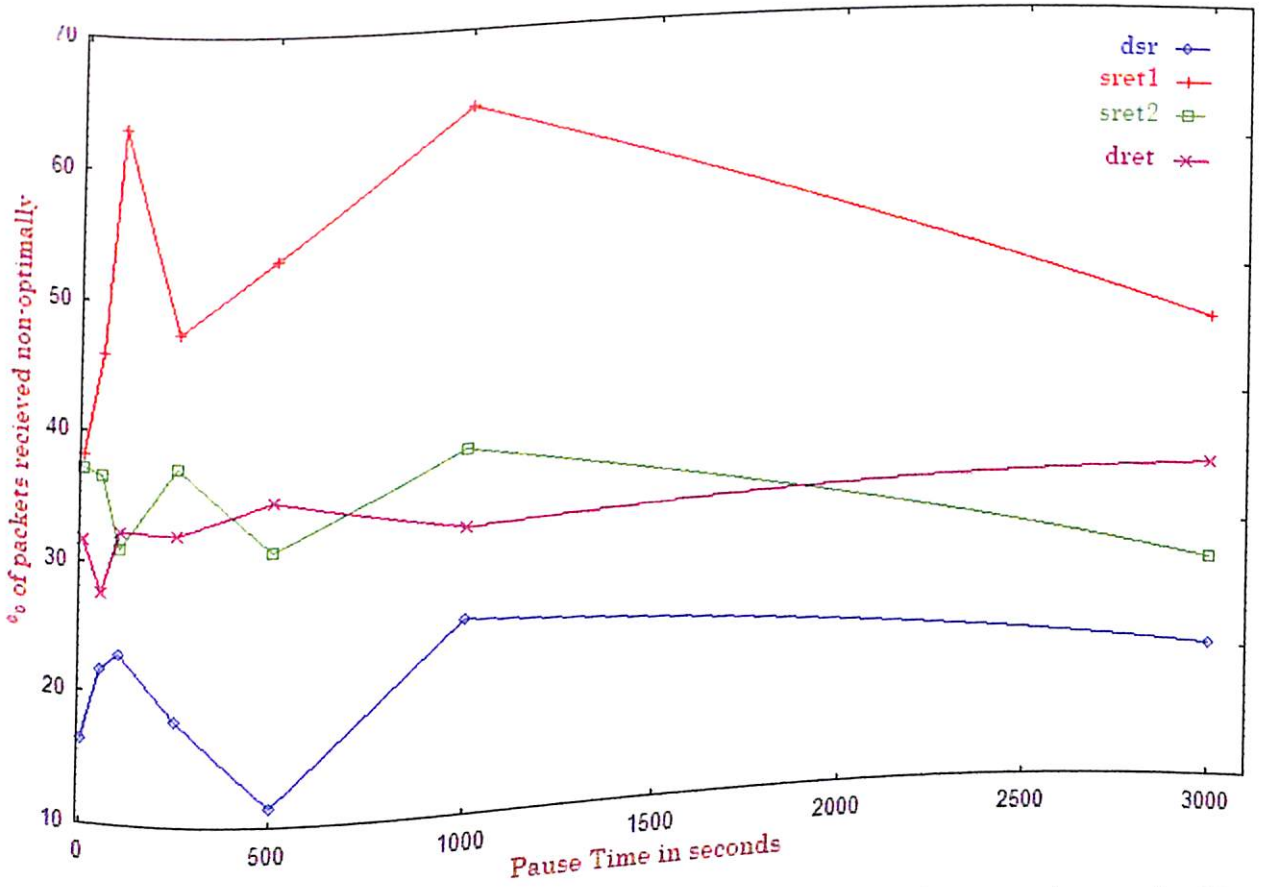


Fig 5.64 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 1m/s

Effect of Routing Criteria on Network Performance Control Overhead Analysis (fig. 5.65 - fig. 5.71)

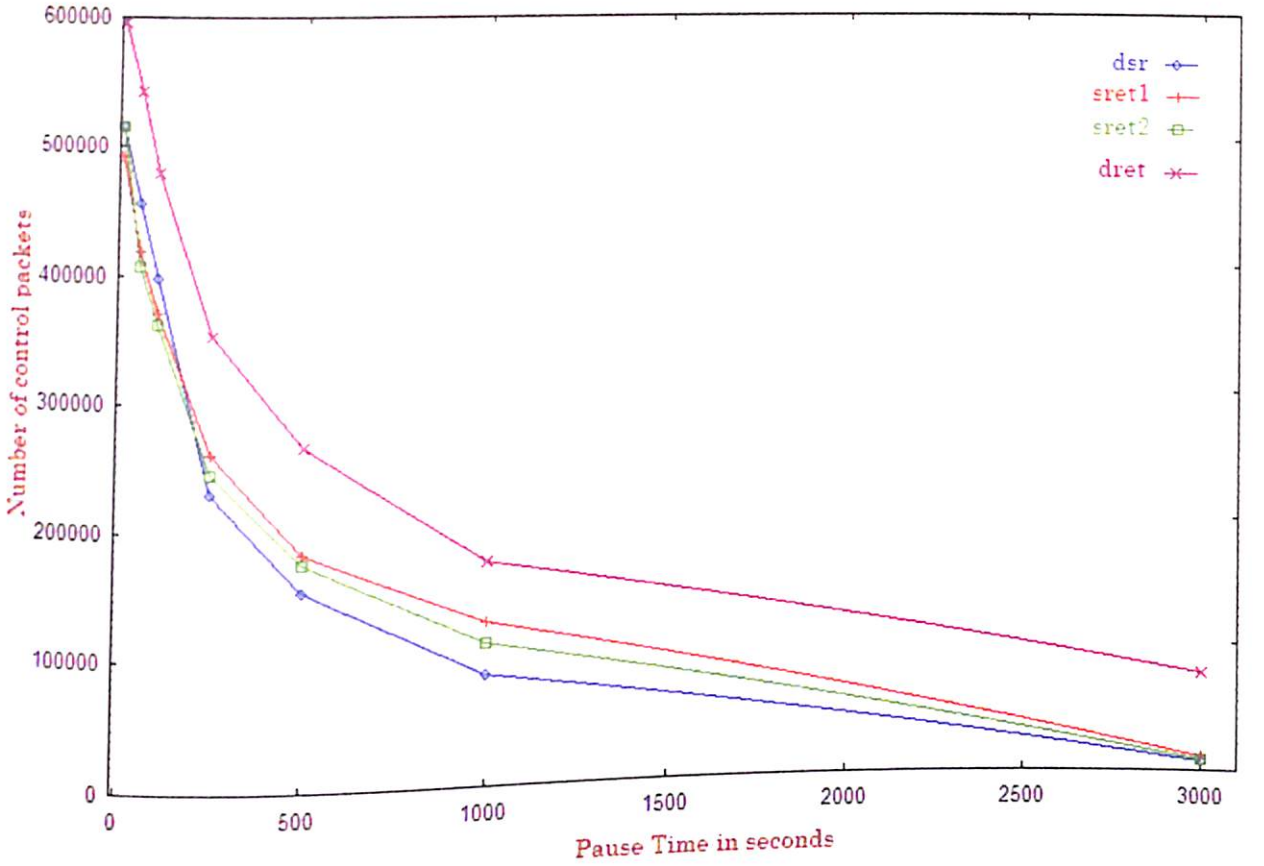


Fig 5.65 Number of Control Packets Vs Pause Time at a maximum node speed of 30m/s

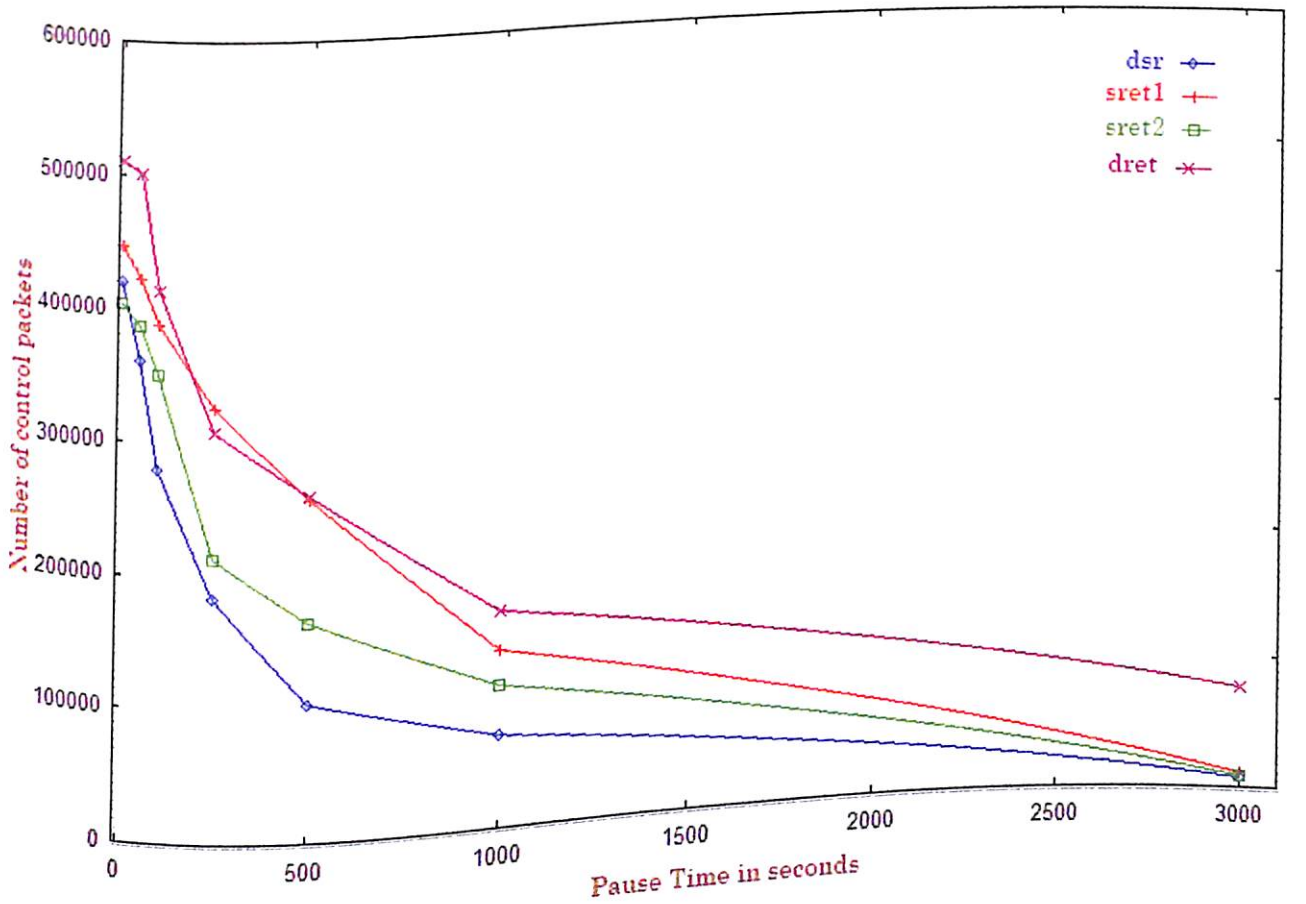


Fig 5.66 Number of Control Packets Vs Pause Time at a maximum node speed of 20m/s

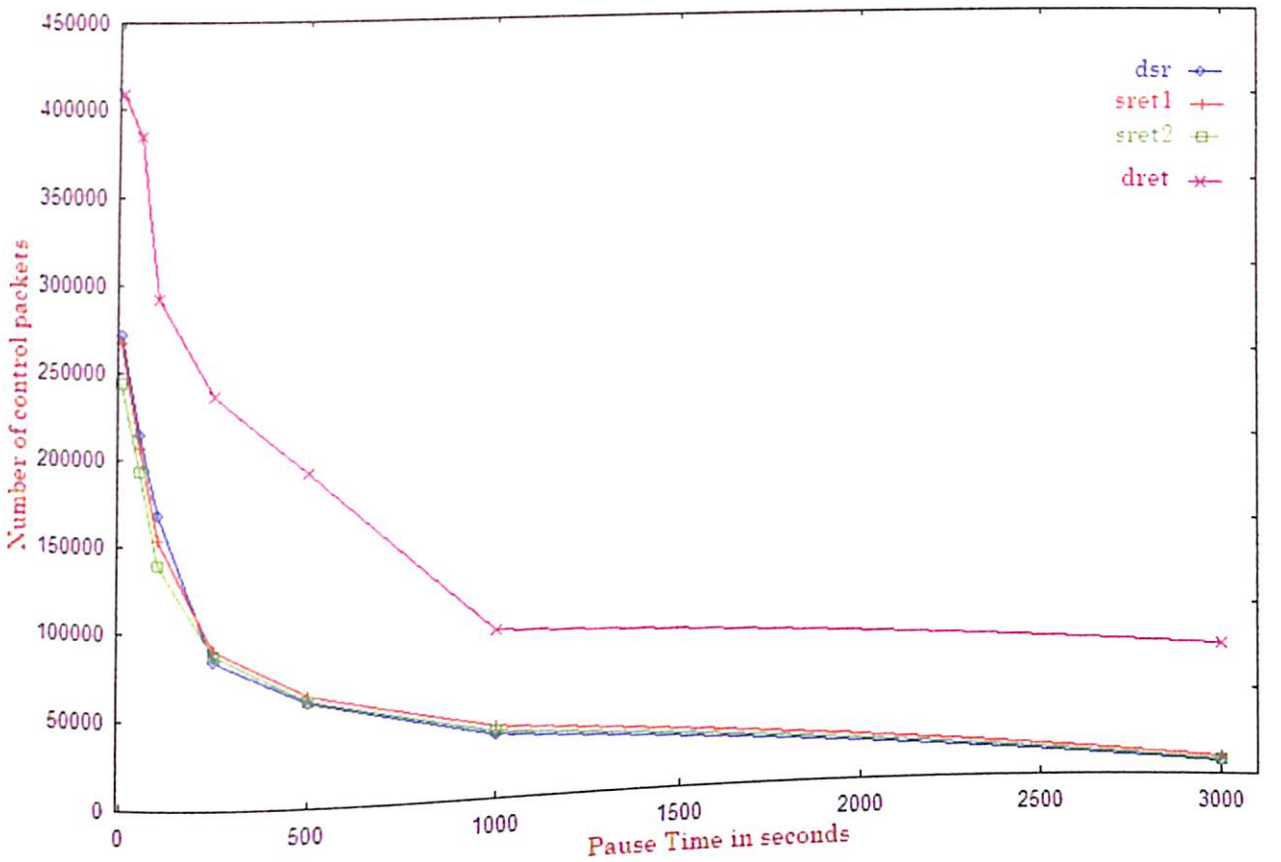


Fig 5.67 Number of Control Packets Vs Pause Time at a maximum node speed of 15m/s

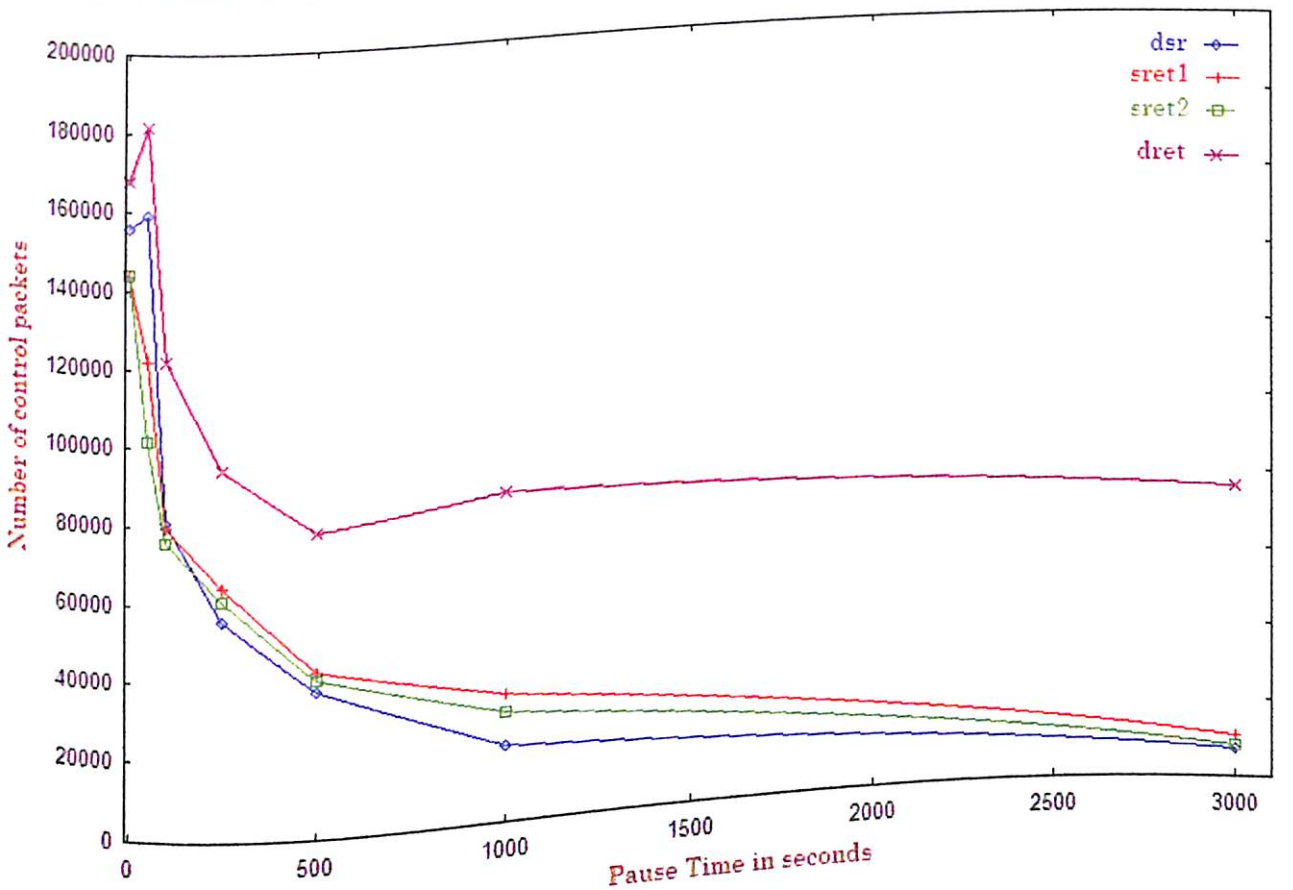


Fig 5.68 Number of Control Packets Vs Pause Time at a maximum node speed of 10m/s

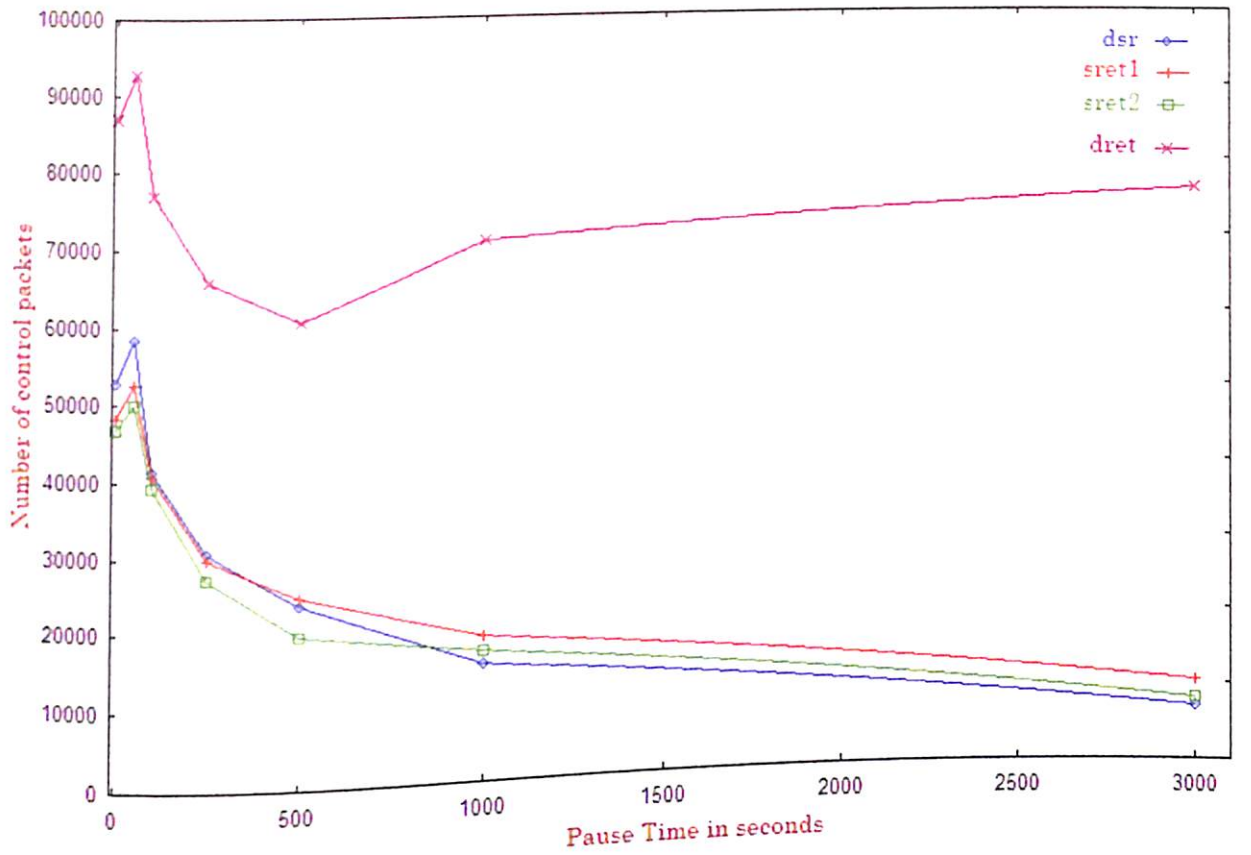


Fig 5.69 Number of Control Packets Vs Pause Time at a maximum node speed of 5m/s

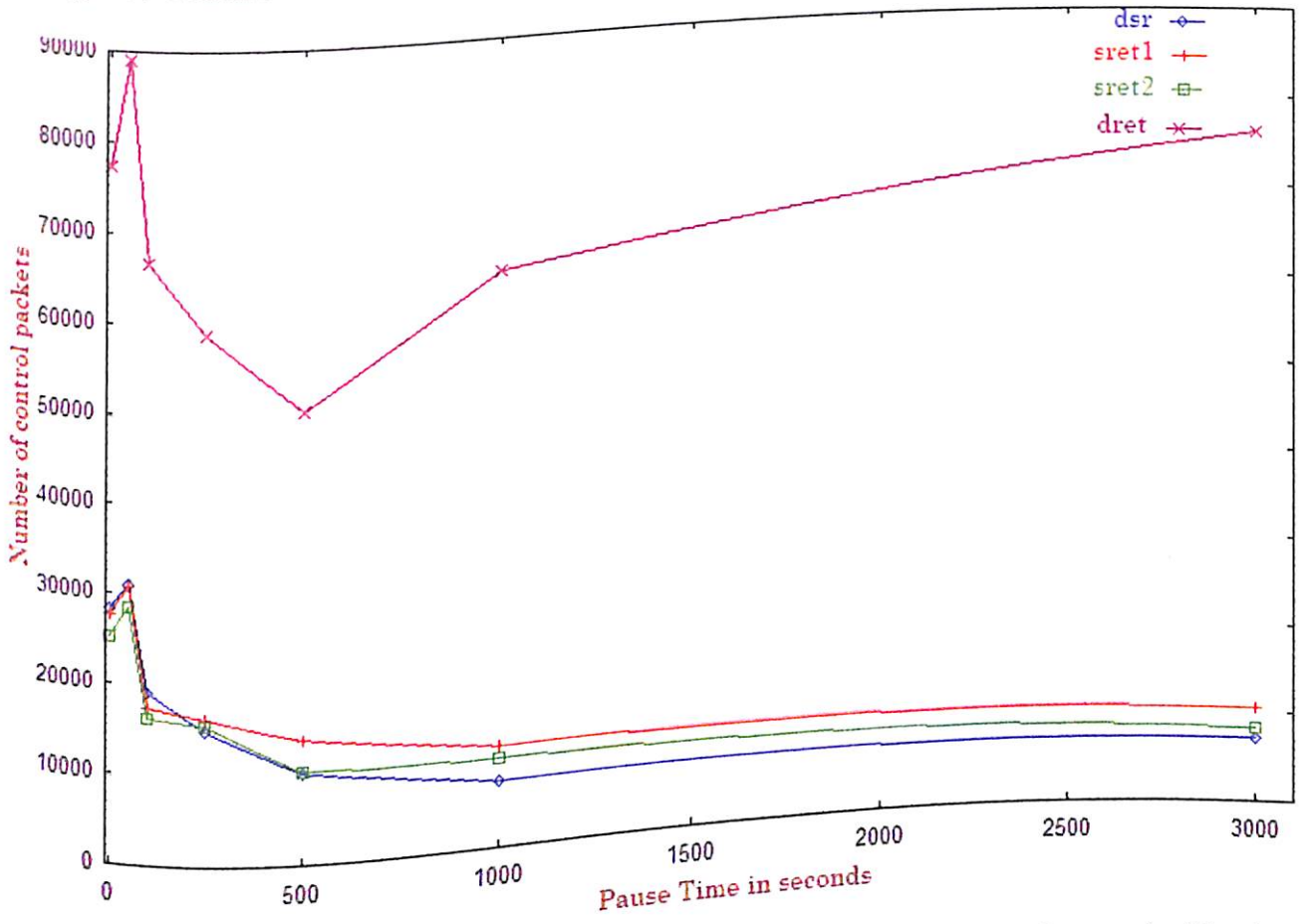


Fig 5.70 Number of Control Packets Vs Pause Time at a maximum node speed of 1m/s

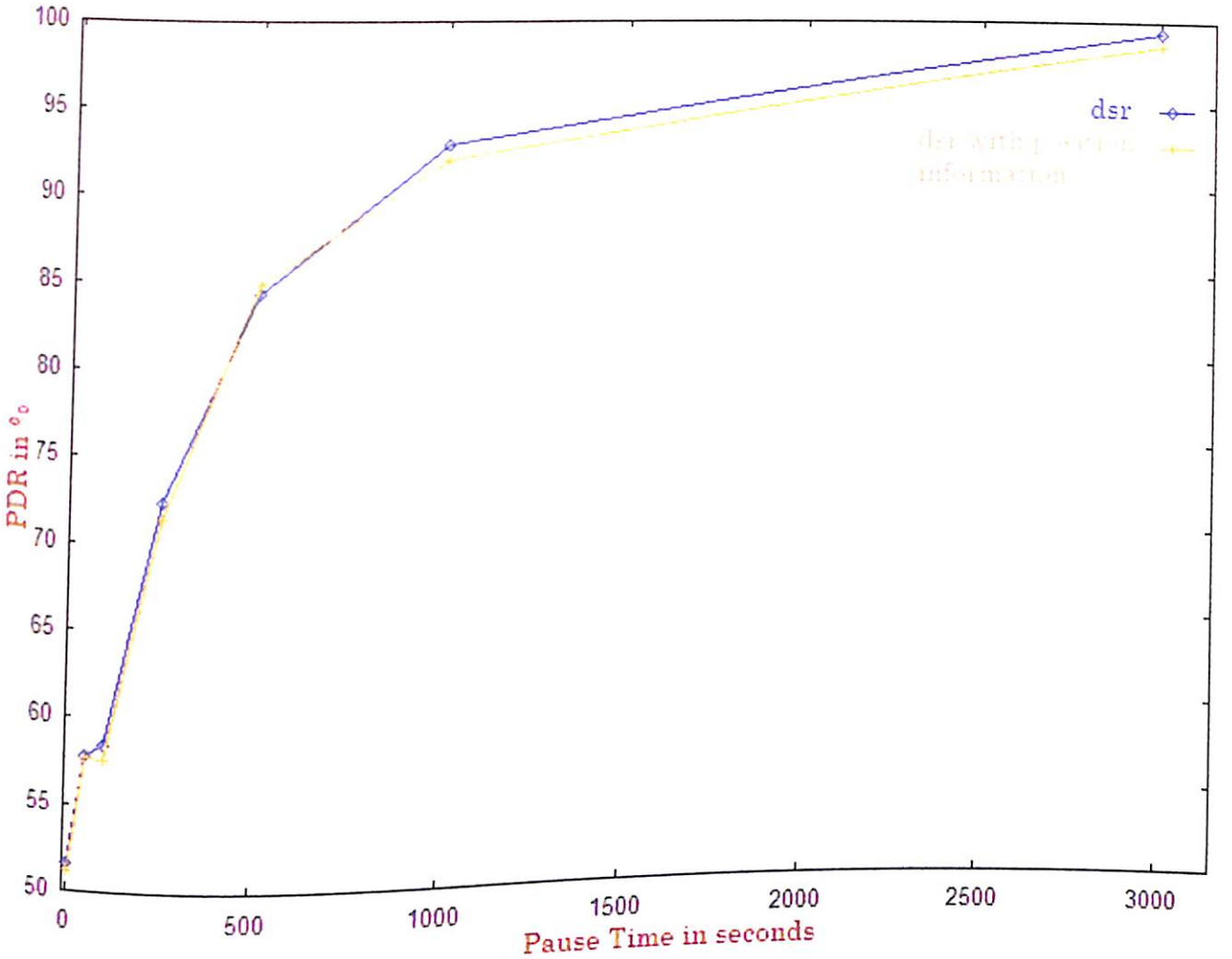


Fig 5.71 PDR Vs Pause Time at a maximum node speed of 30m/s indicating effect of position information

Analysis of GPEAR Packet Delivery Ratio (fig. 5.72 - fig. 5.101)

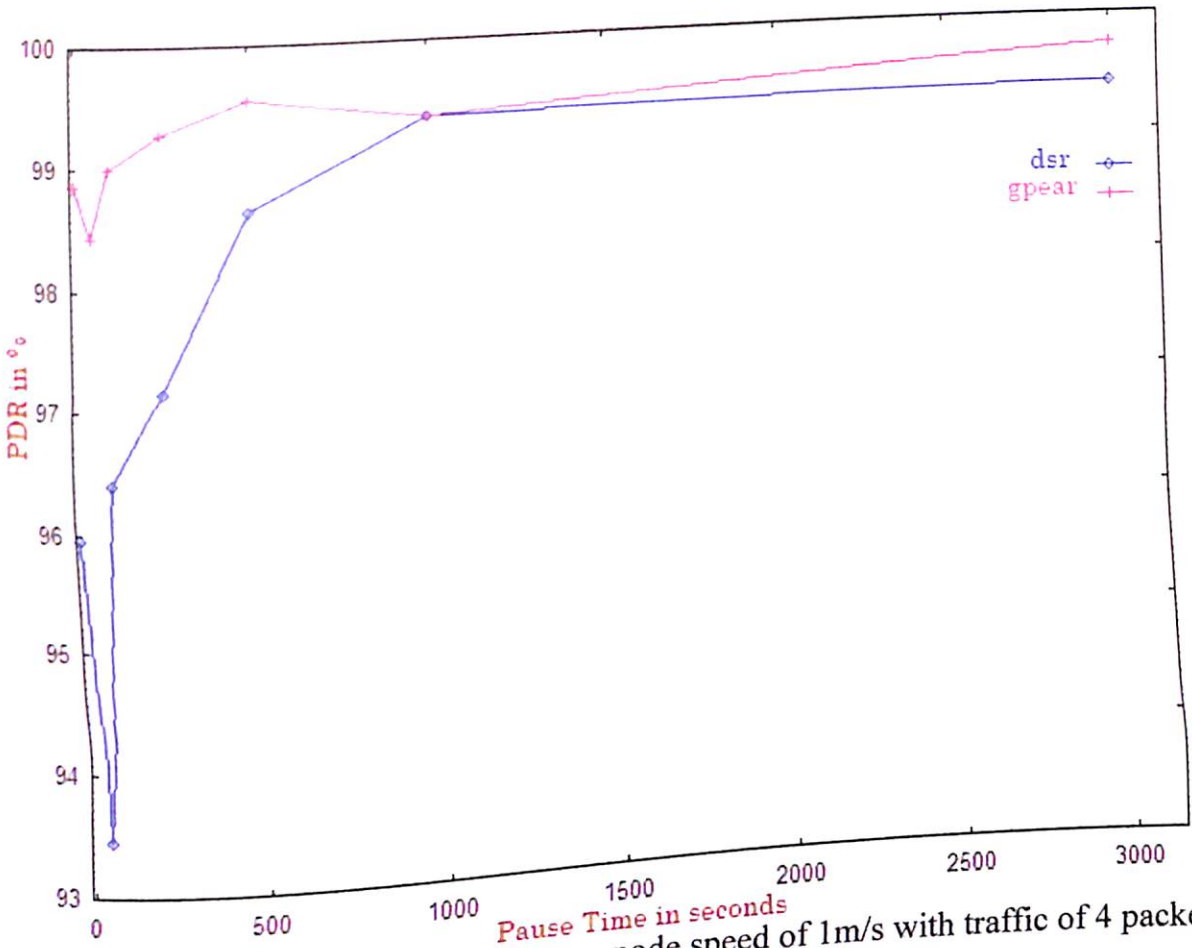


Fig 5.72 PDR Vs Pause Time at a maximum node speed of 1m/s with traffic of 4 packets/s

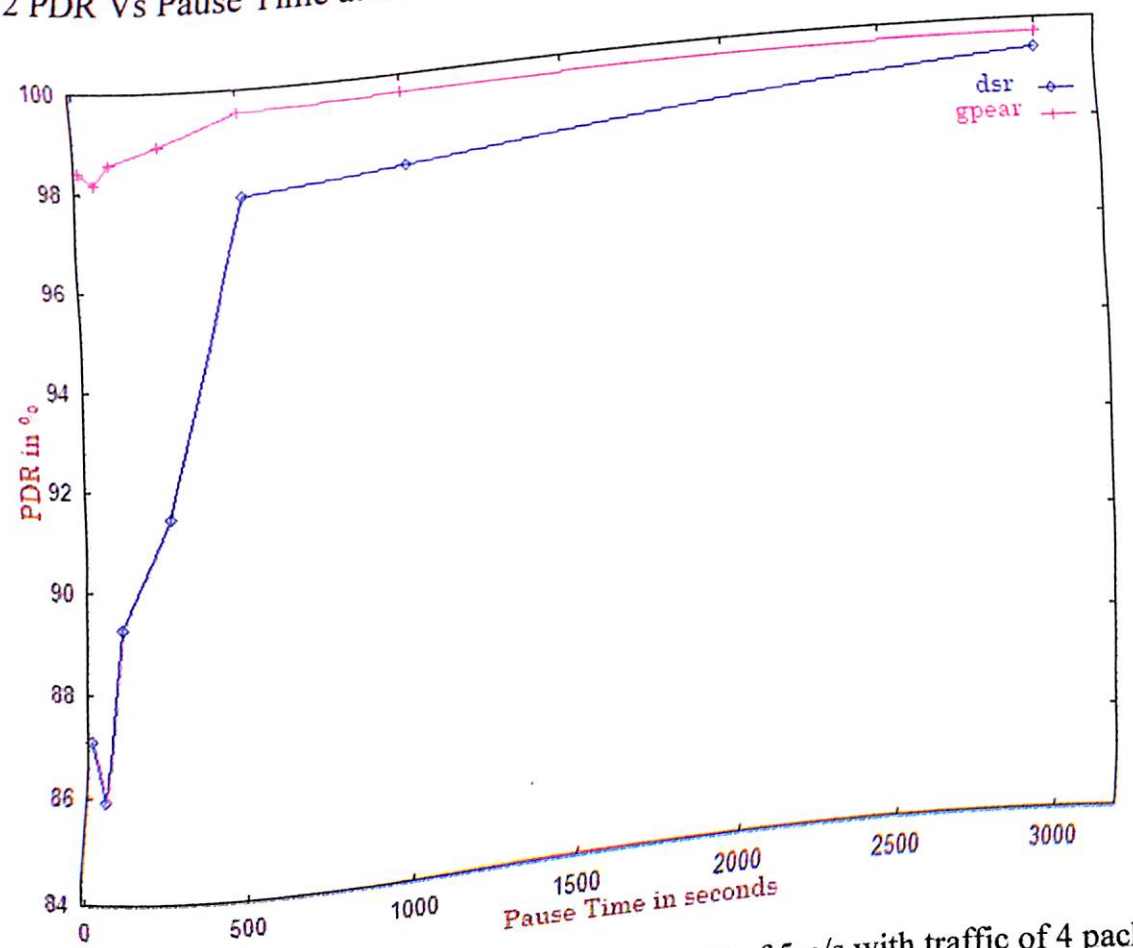


Fig 5.73 PDR Vs Pause Time at a maximum node speed of 5m/s with traffic of 4 packets/s

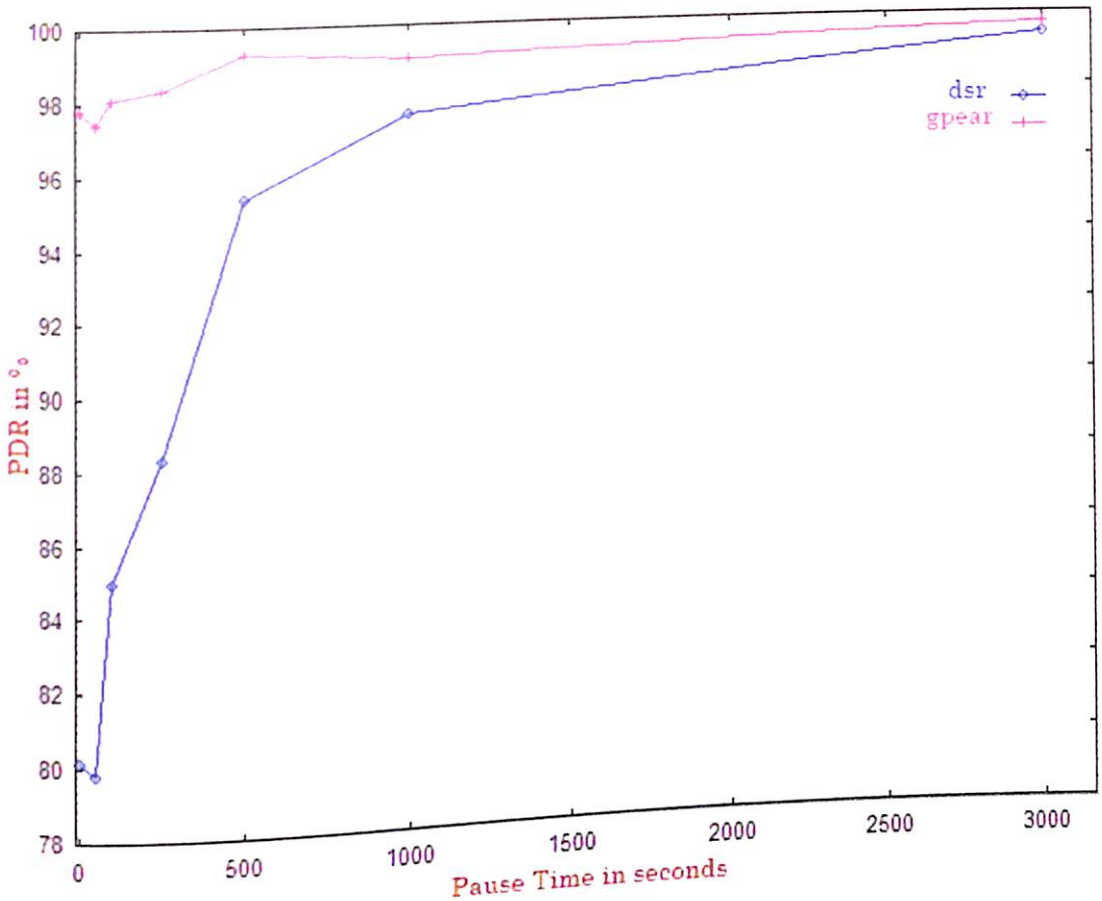


Fig 5.74 PDR Vs Pause Time at a maximum node speed of 10m/s with traffic of 4 packets/s

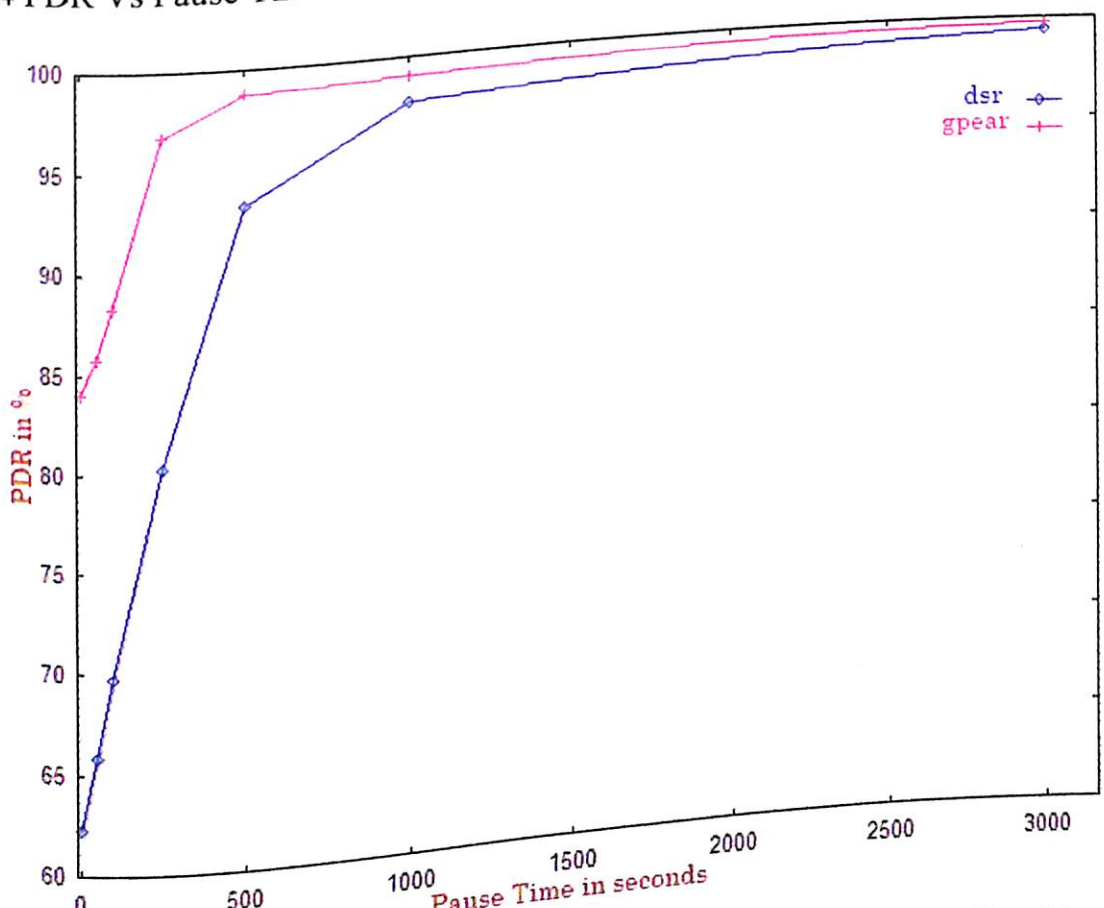


Fig 5.75 PDR Vs Pause Time at a maximum node speed of 15m/s with traffic of 4 packets/s

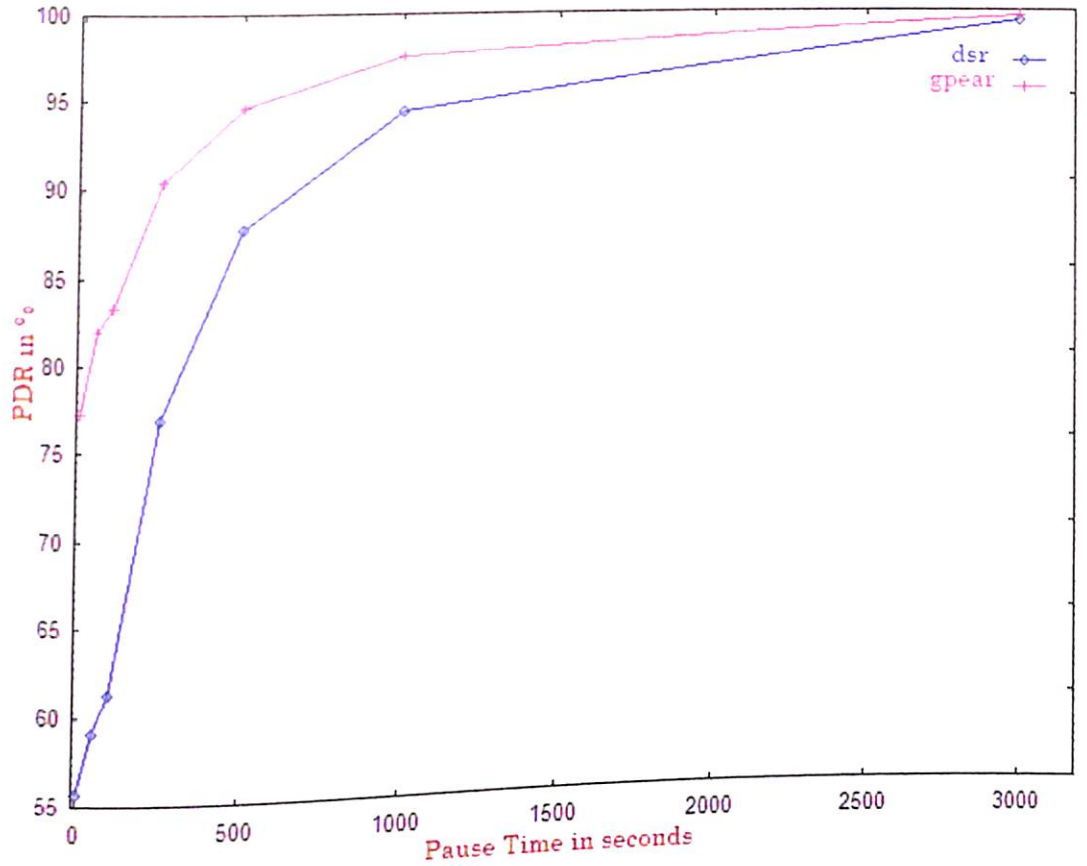


Fig 5.76 PDR Vs Pause Time at a maximum node speed of 20m/s with traffic of 4 packets/s

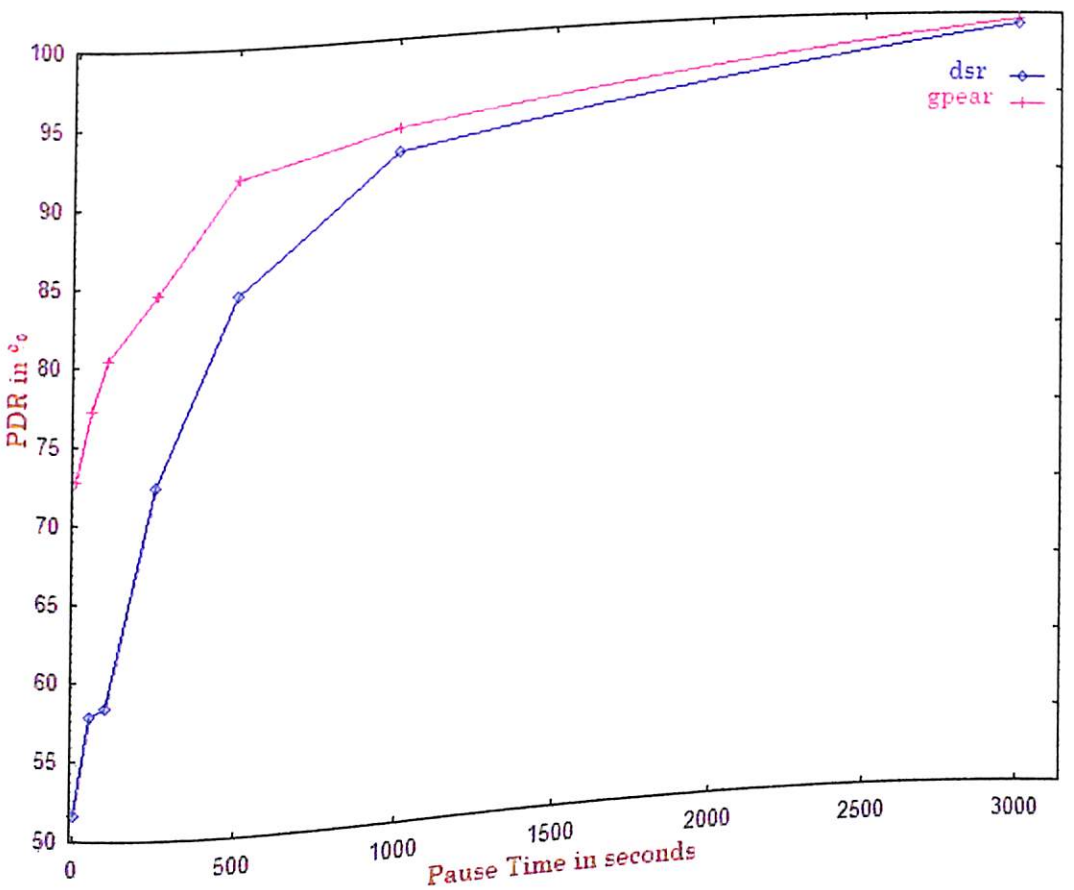


Fig 5.77 PDR Vs Pause Time at a maximum node speed of 30m/s with traffic of 4 packets/s

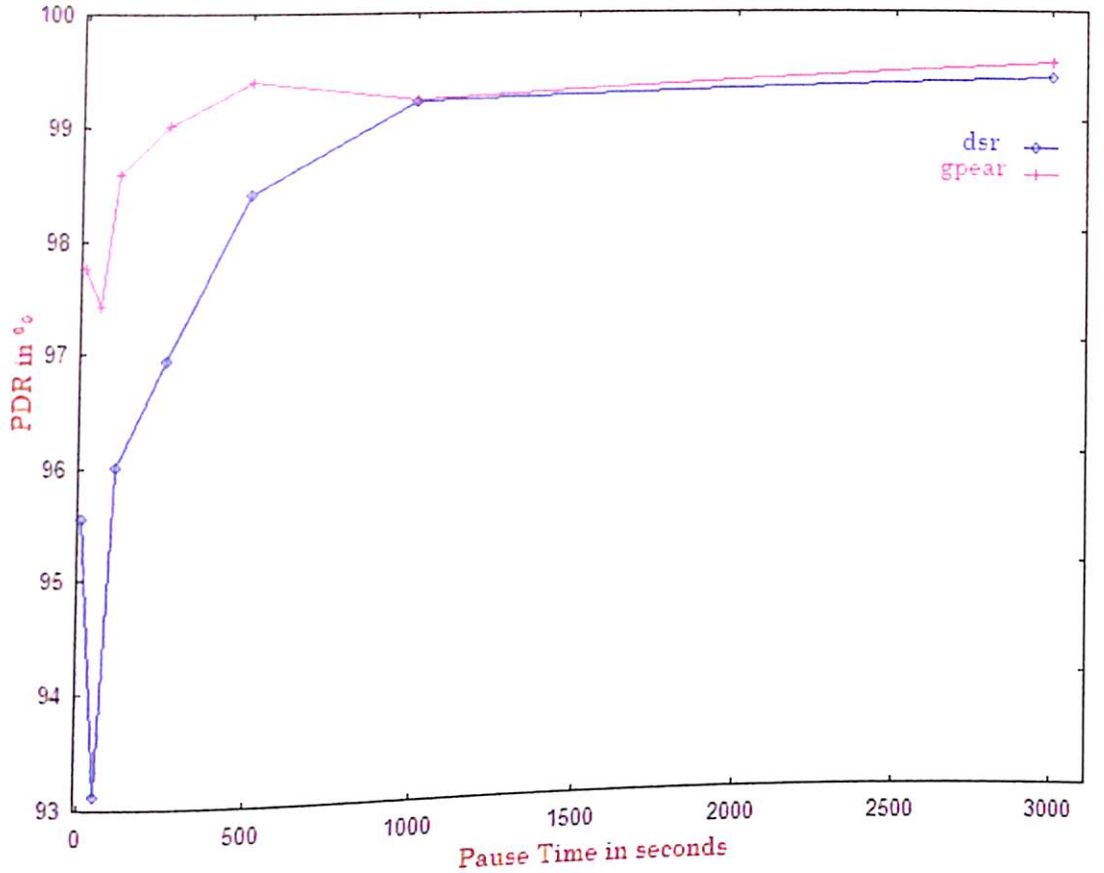


Fig 5.78 PDR Vs Pause Time at a maximum node speed of 1m/s with traffic of 6 packets/s

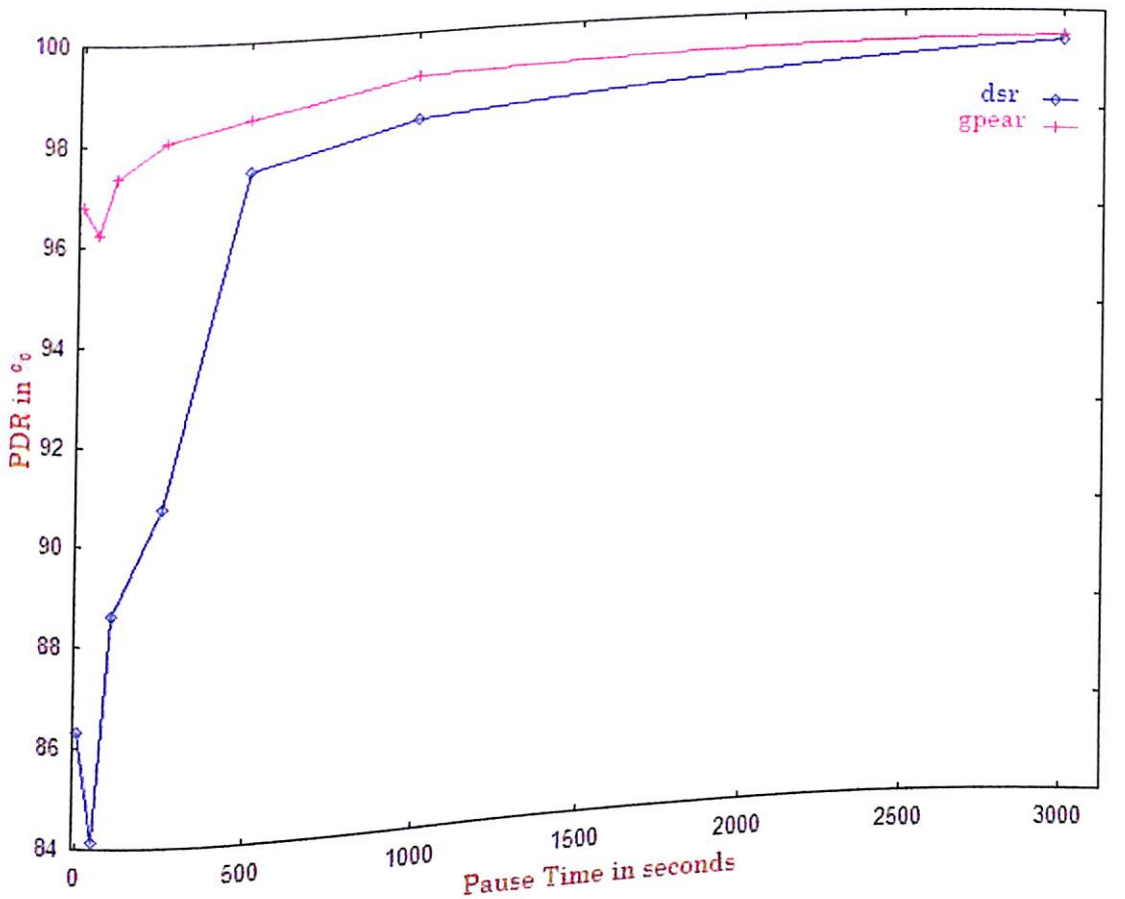


Fig 5.79 PDR Vs Pause Time at a maximum node speed of 5m/s with traffic of 6 packets/s

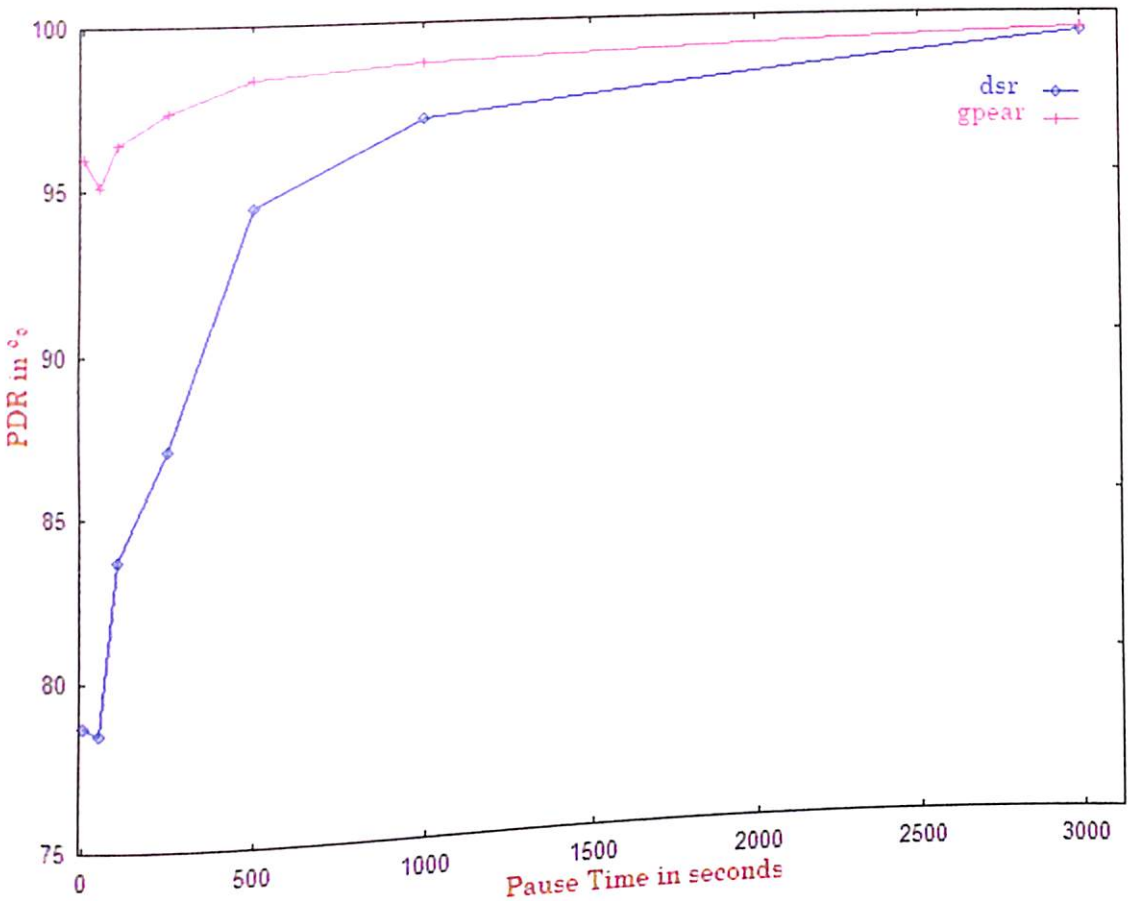


Fig 5.80 PDR Vs Pause Time at a maximum node speed of 10m/s with traffic of 6 packets/s

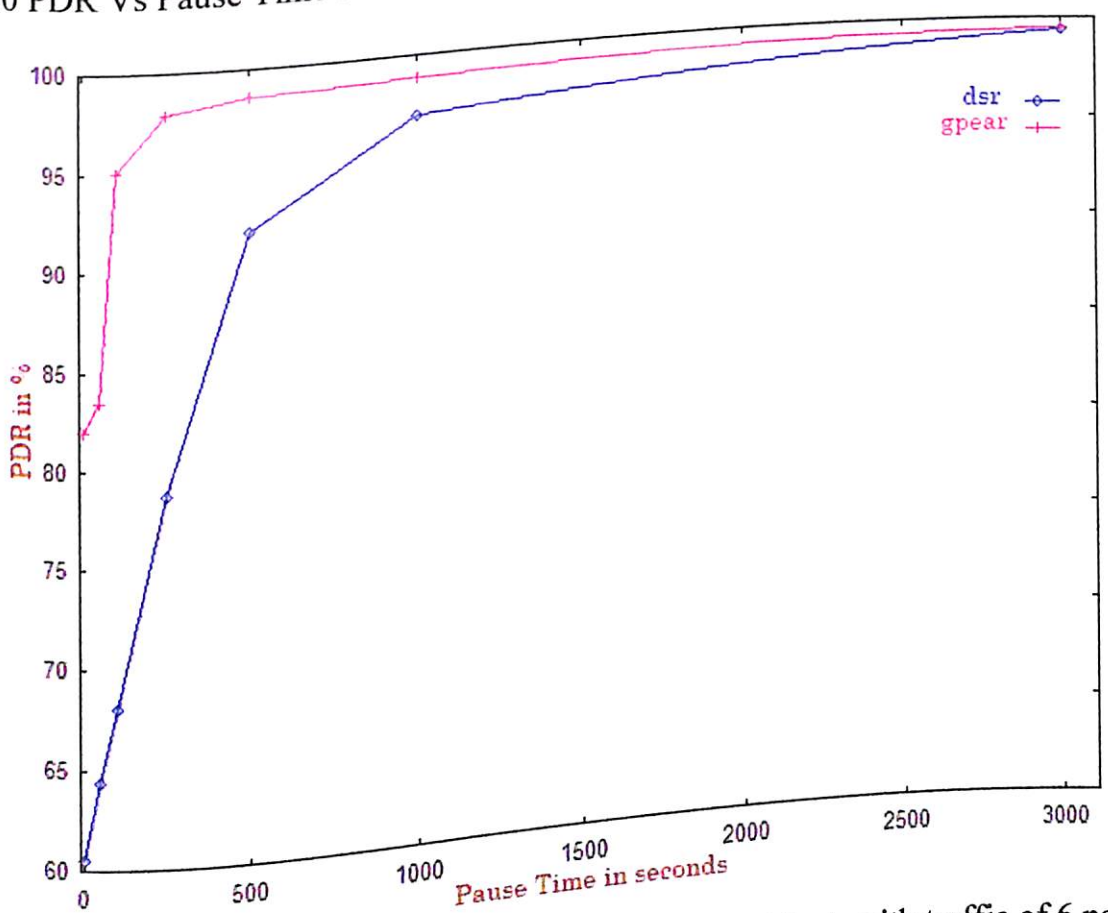


Fig 5.81 PDR Vs Pause Time at a maximum node speed of 15m/s with traffic of 6 packets/s

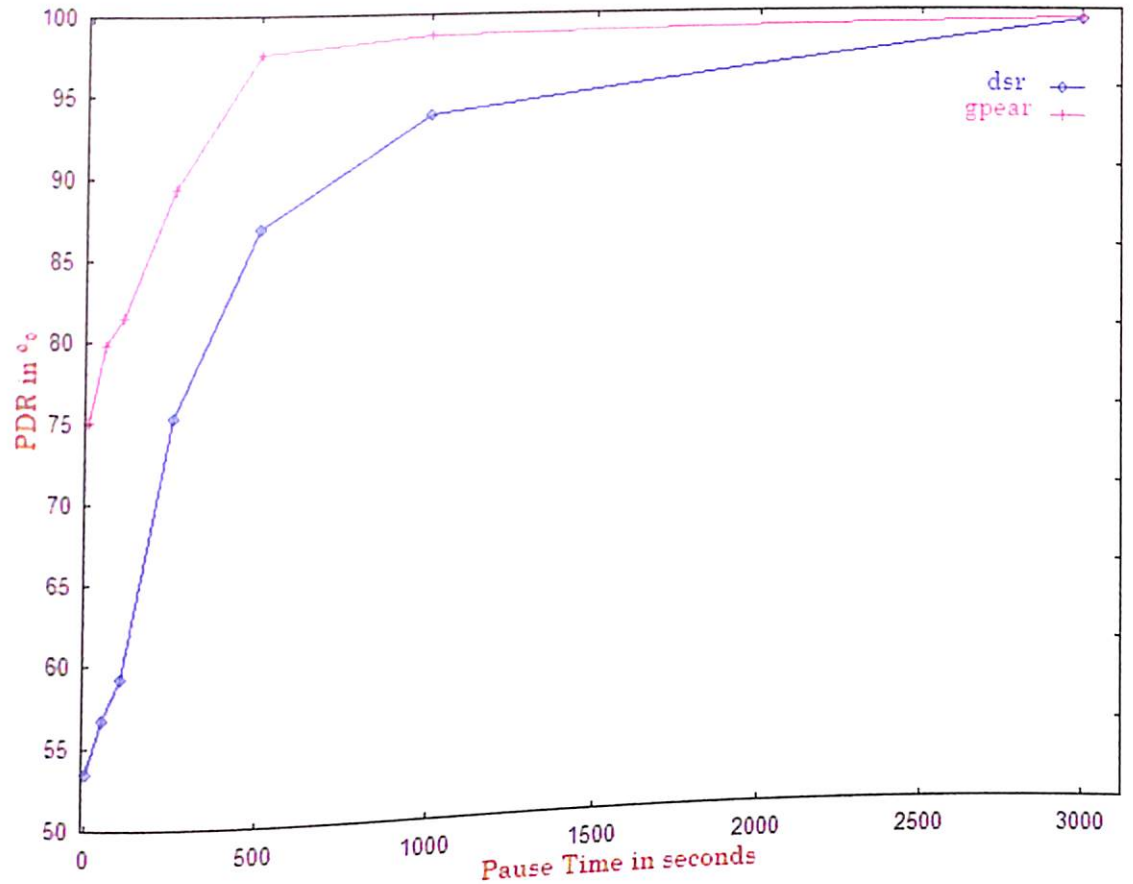


Fig 5.82 PDR Vs Pause Time at a maximum node speed of 20m/s with traffic of 6 packets/s

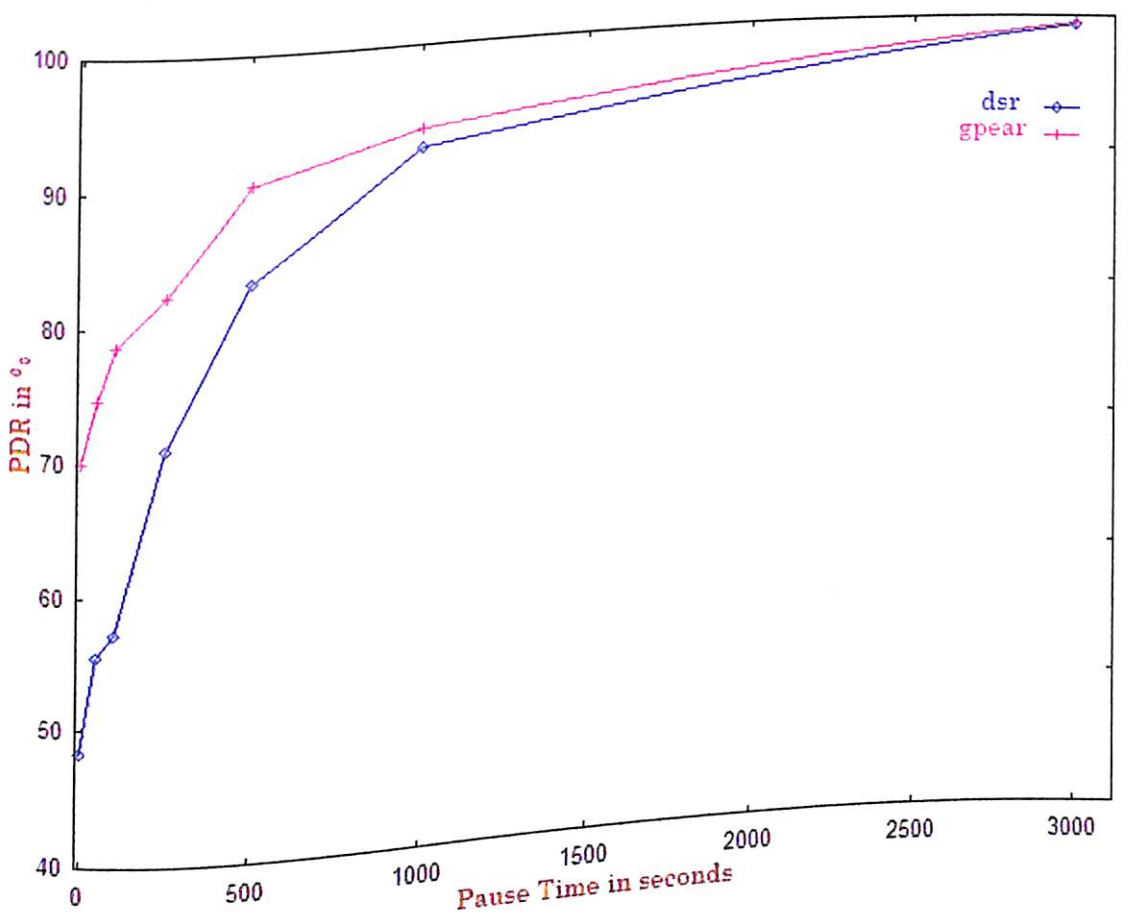


Fig 5.83 PDR Vs Pause Time at a maximum node speed of 30m/s with traffic of 6 packets/s

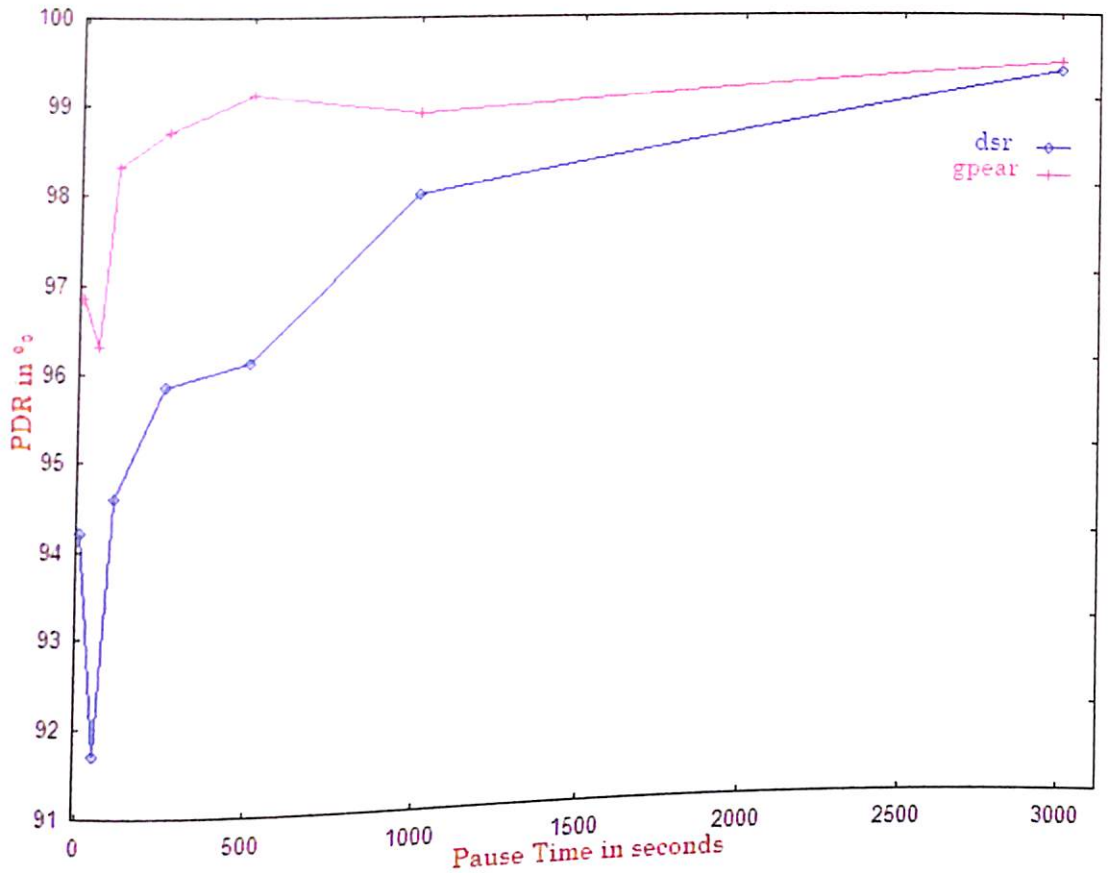


Fig 5.84 PDR Vs Pause Time at a maximum node speed of 1m/s with traffic of 8 packets/s

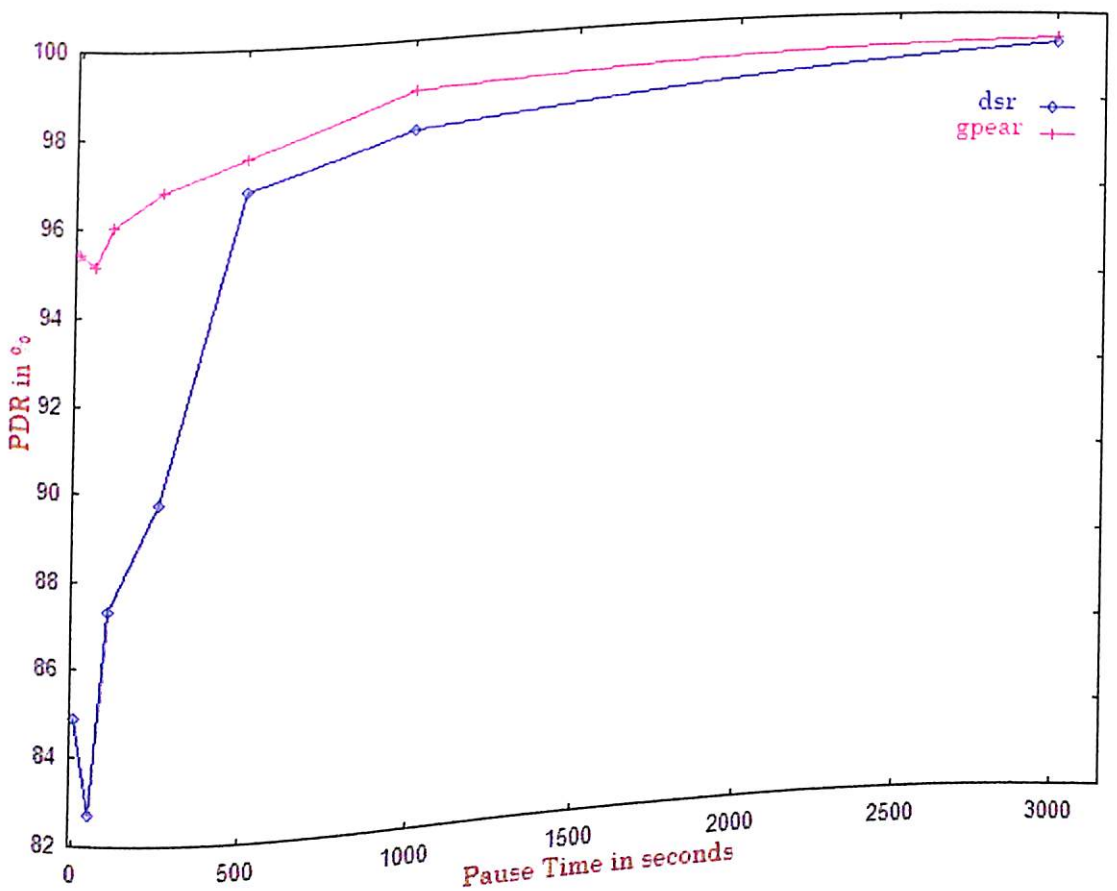


Fig 5.85 PDR Vs Pause Time at a maximum node speed of 5m/s with traffic of 8 packets/s

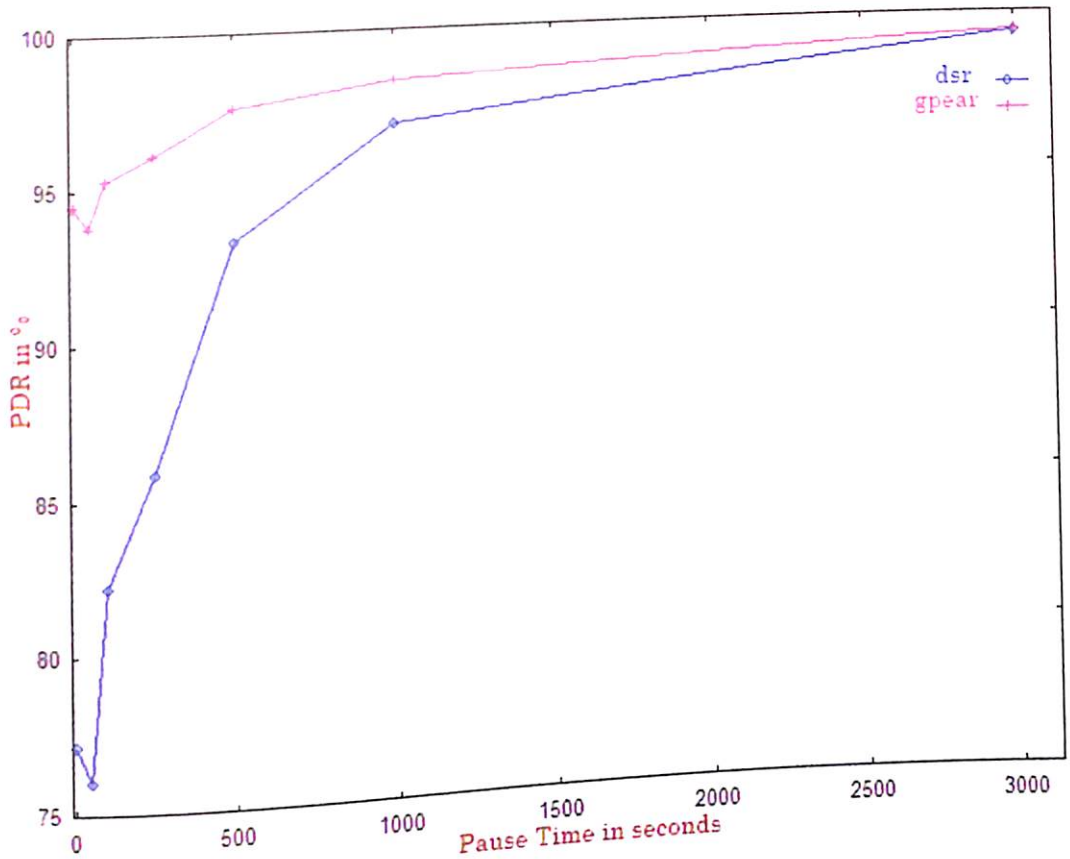


Fig 5.86 PDR Vs Pause Time at a maximum node speed of 10m/s with traffic of 8 packets/s

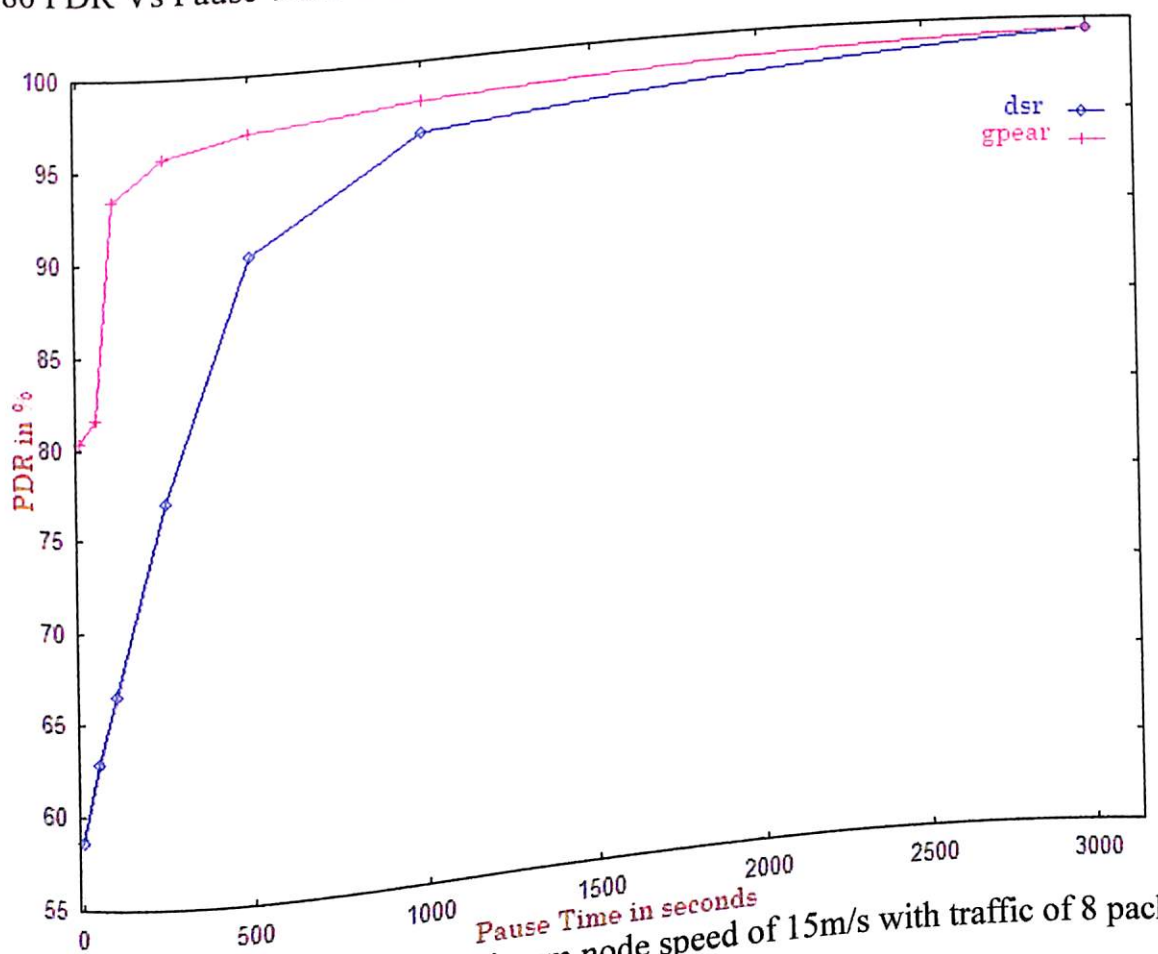


Fig 5.87 PDR Vs Pause Time at a maximum node speed of 15m/s with traffic of 8 packets/s

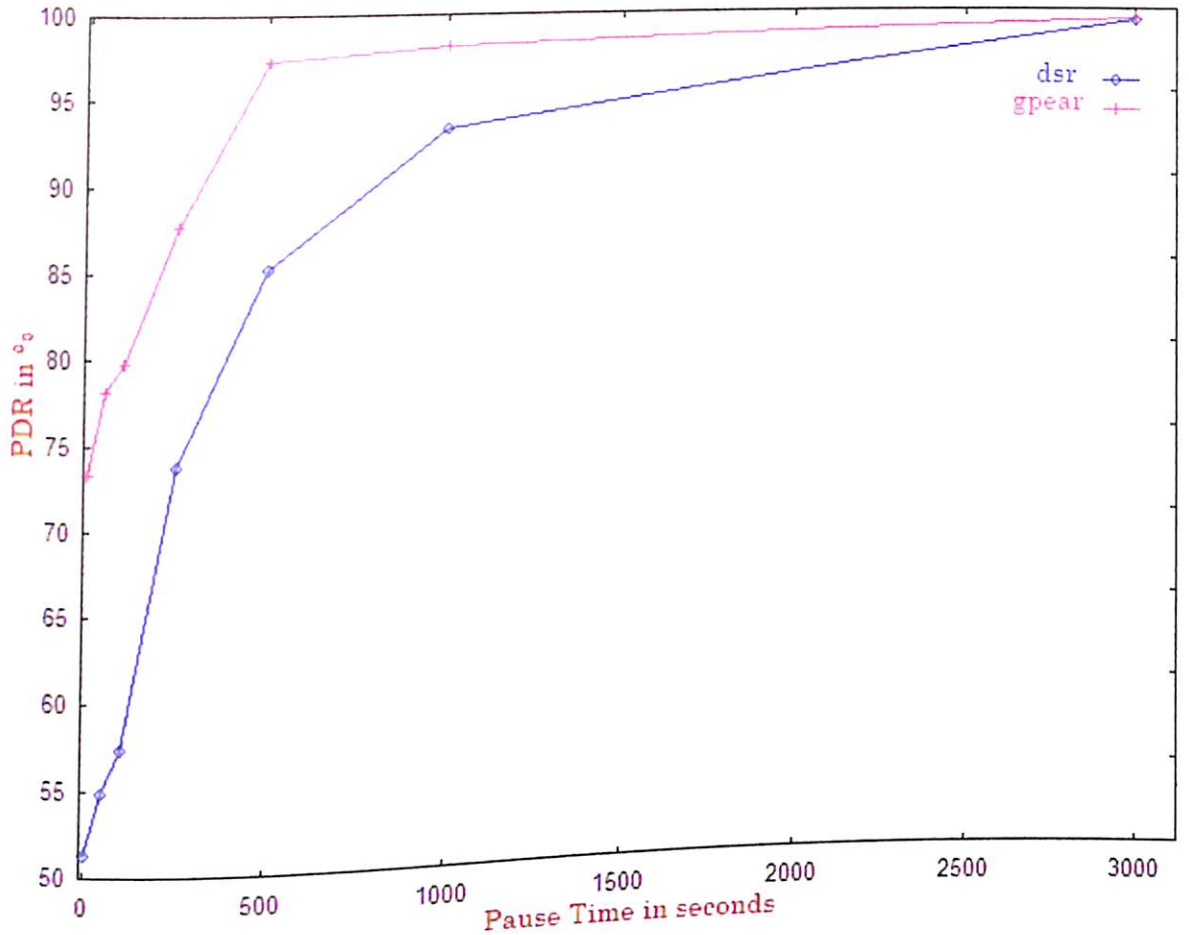


Fig 5.88 PDR Vs Pause Time at a maximum node speed of 20m/s with traffic of 8 packets/s

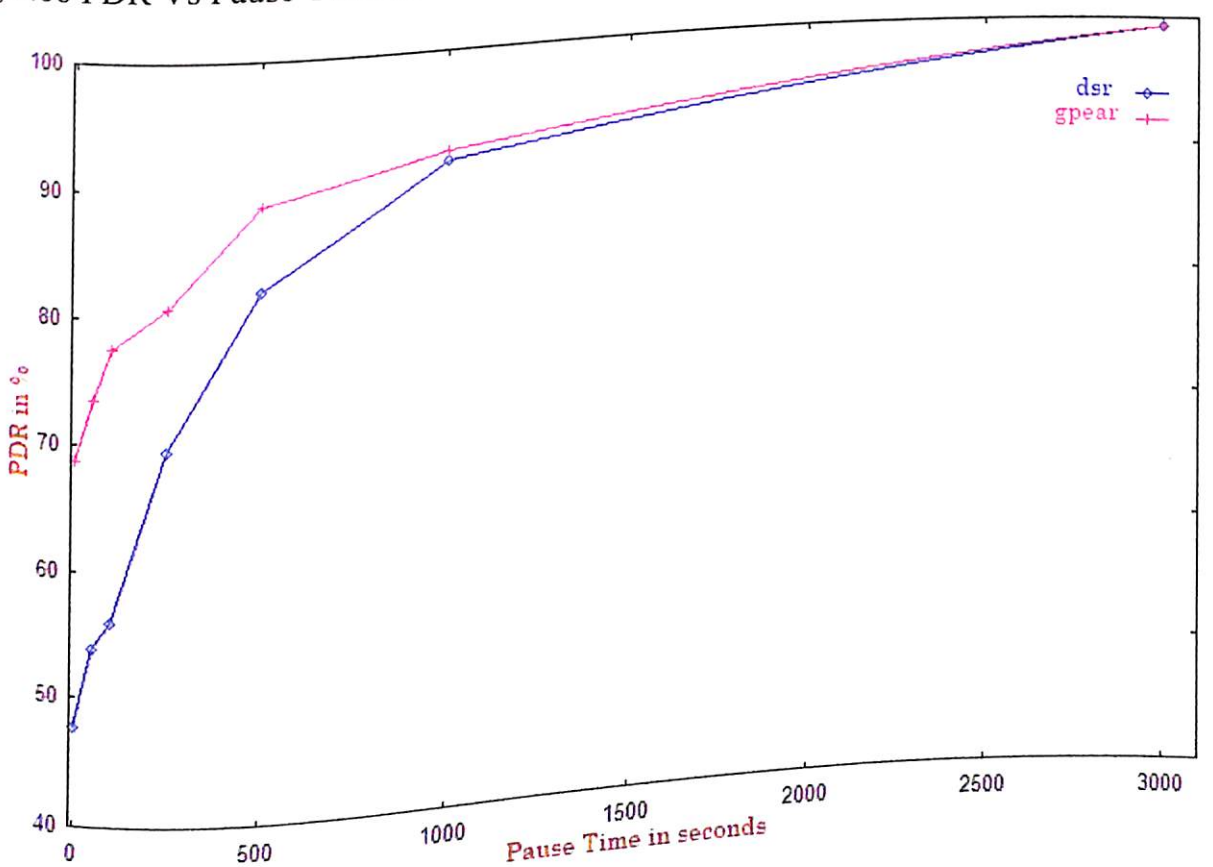


Fig 5.89 PDR Vs Pause Time at a maximum node speed of 30m/s with traffic of 8 packets/s

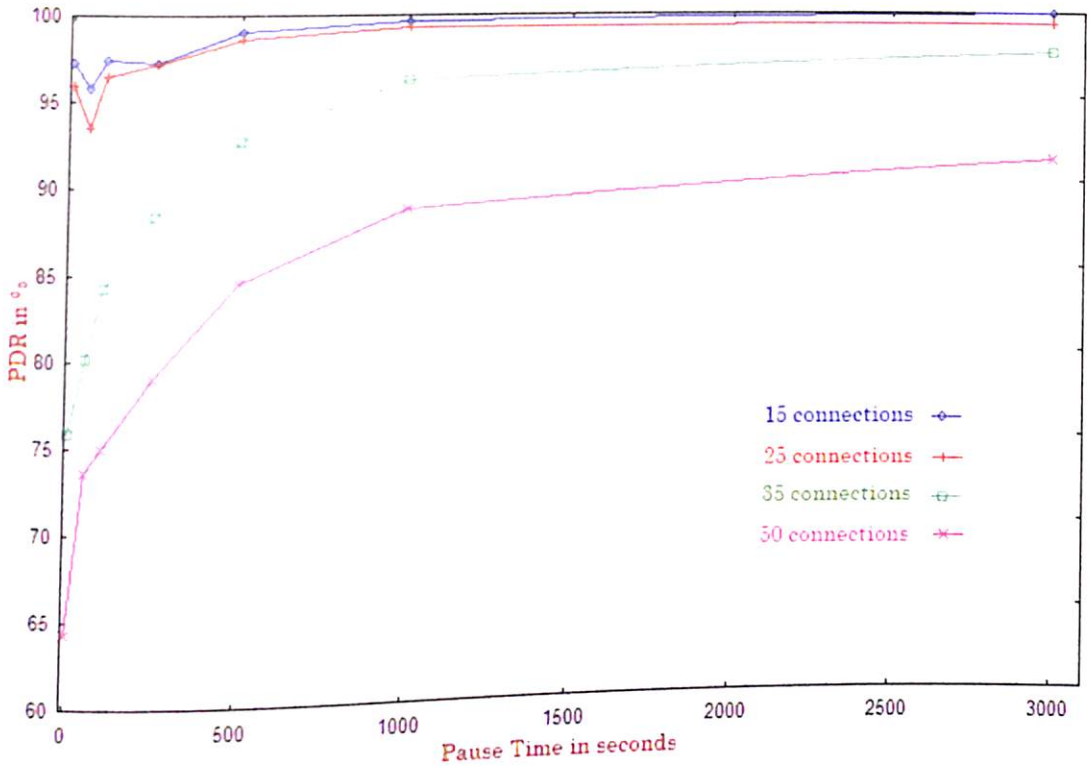


Fig 5.90 PDR Vs Pause Time at a maximum node speed of 1m/s using DSR as the routing protocol for varying number of connections

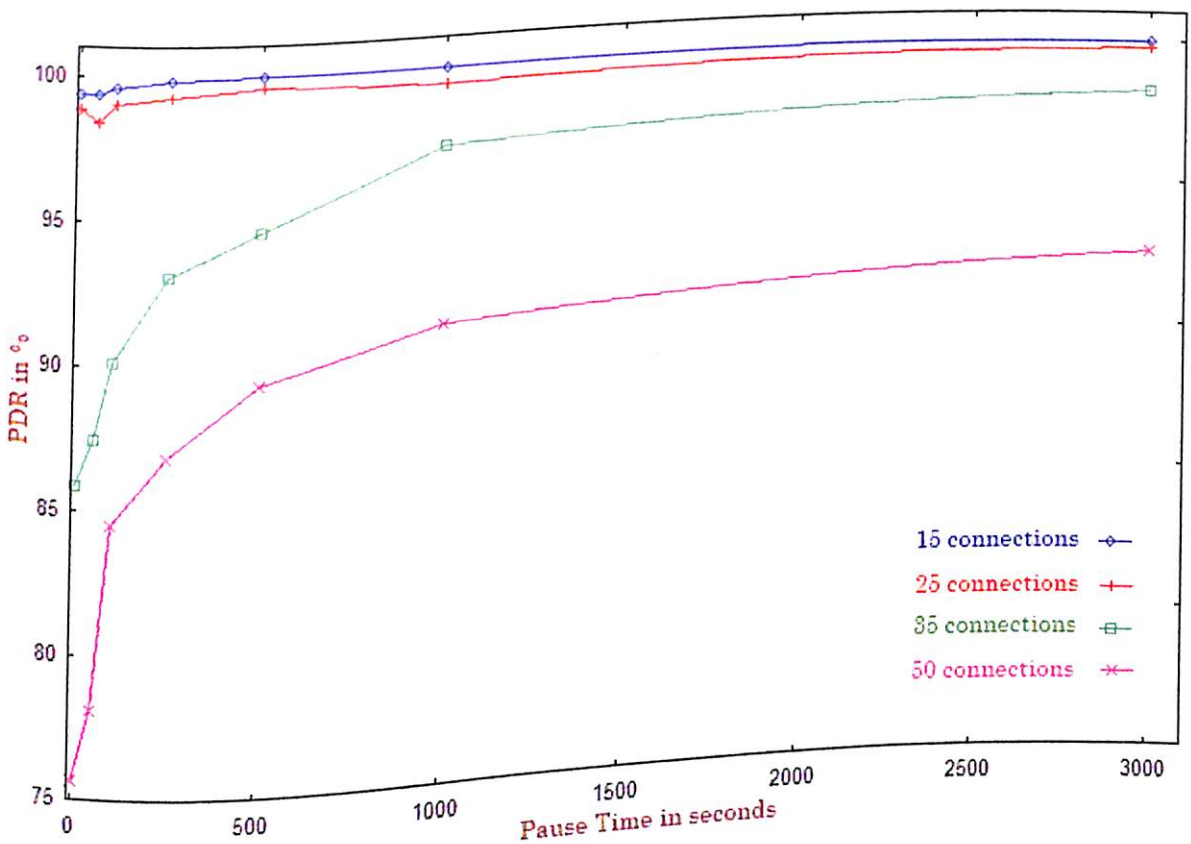


Fig 5.91 PDR Vs Pause Time at a maximum node speed of 1m/s using GPEAR as the routing protocol for varying number of connections

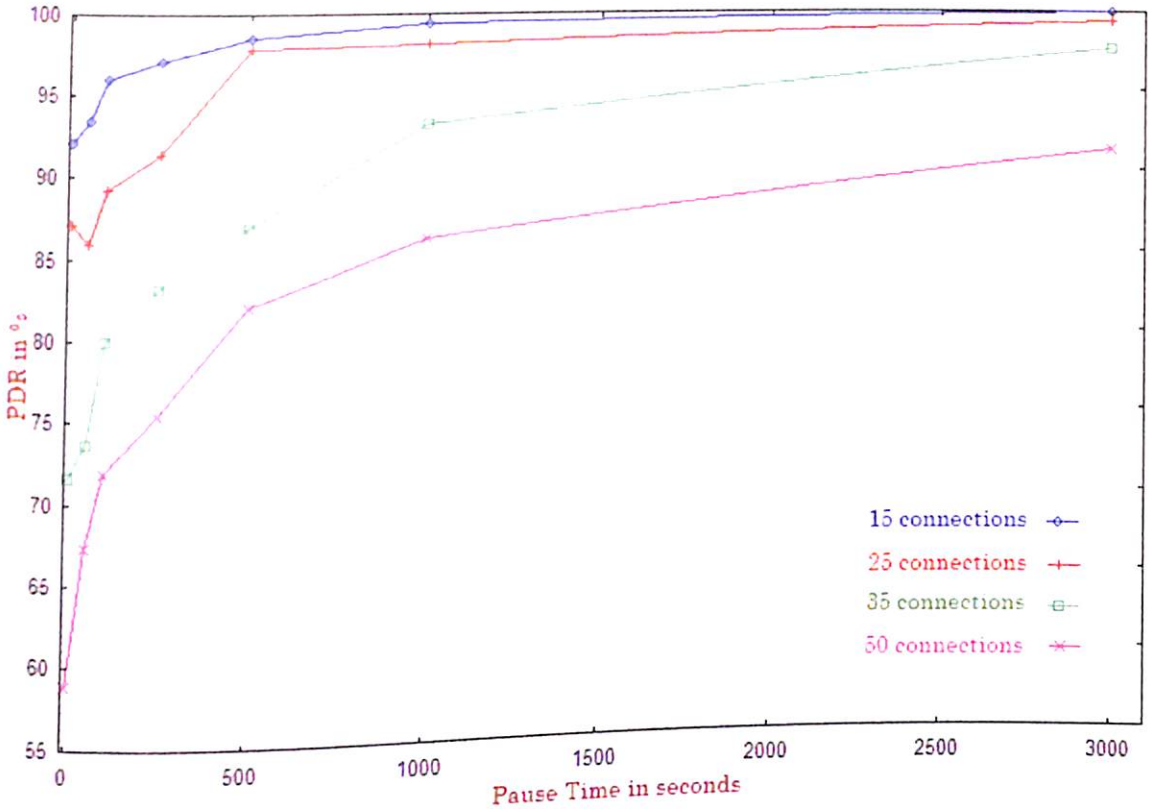


Fig 5.92 PDR Vs Pause Time at a maximum node speed of 5m/s using DSR as the routing protocol for varying number of connections

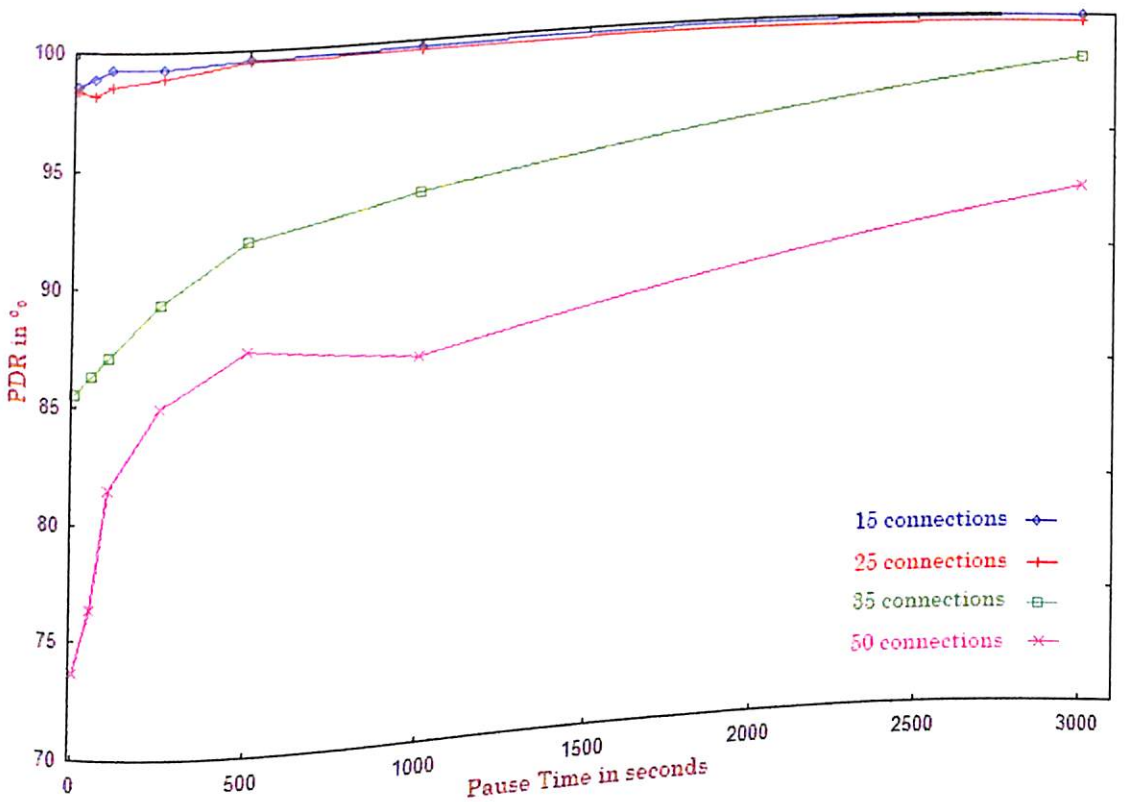


Fig 5.93 PDR Vs Pause Time at a maximum node speed of 5m/s using GPEAR as the routing protocol for varying number of connections

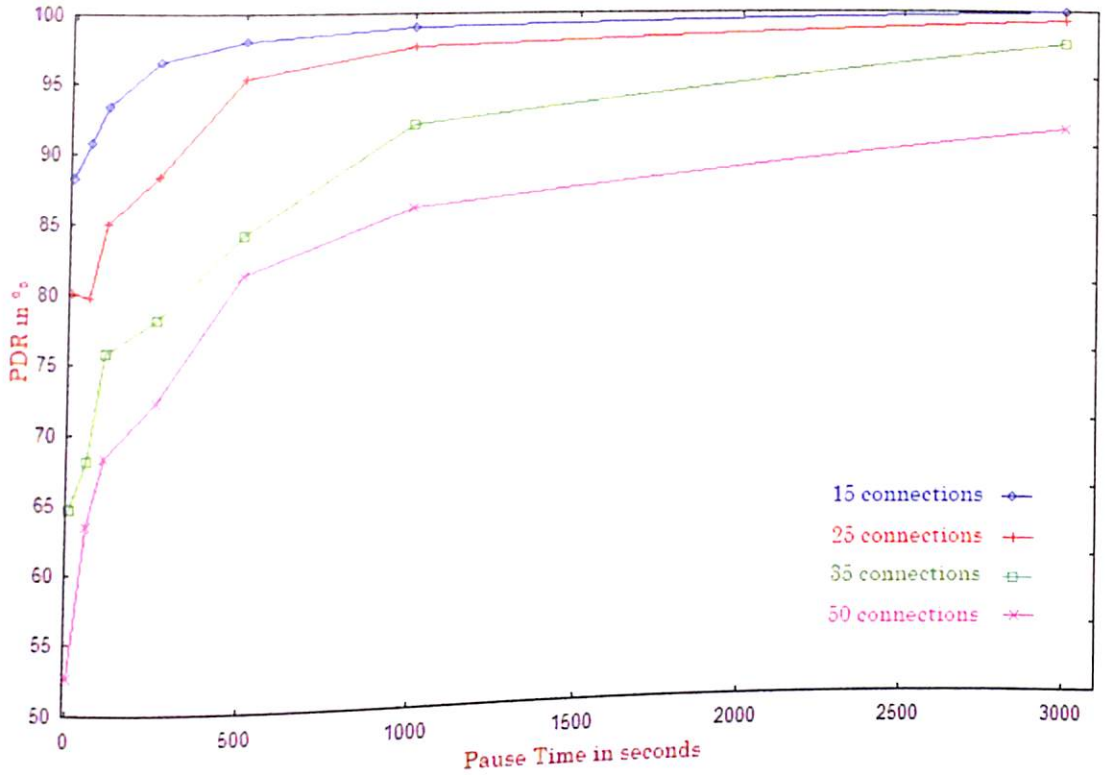


Fig 5.94 PDR Vs Pause Time at a maximum node speed of 10m/s using DSR as the routing protocol for varying number of connections

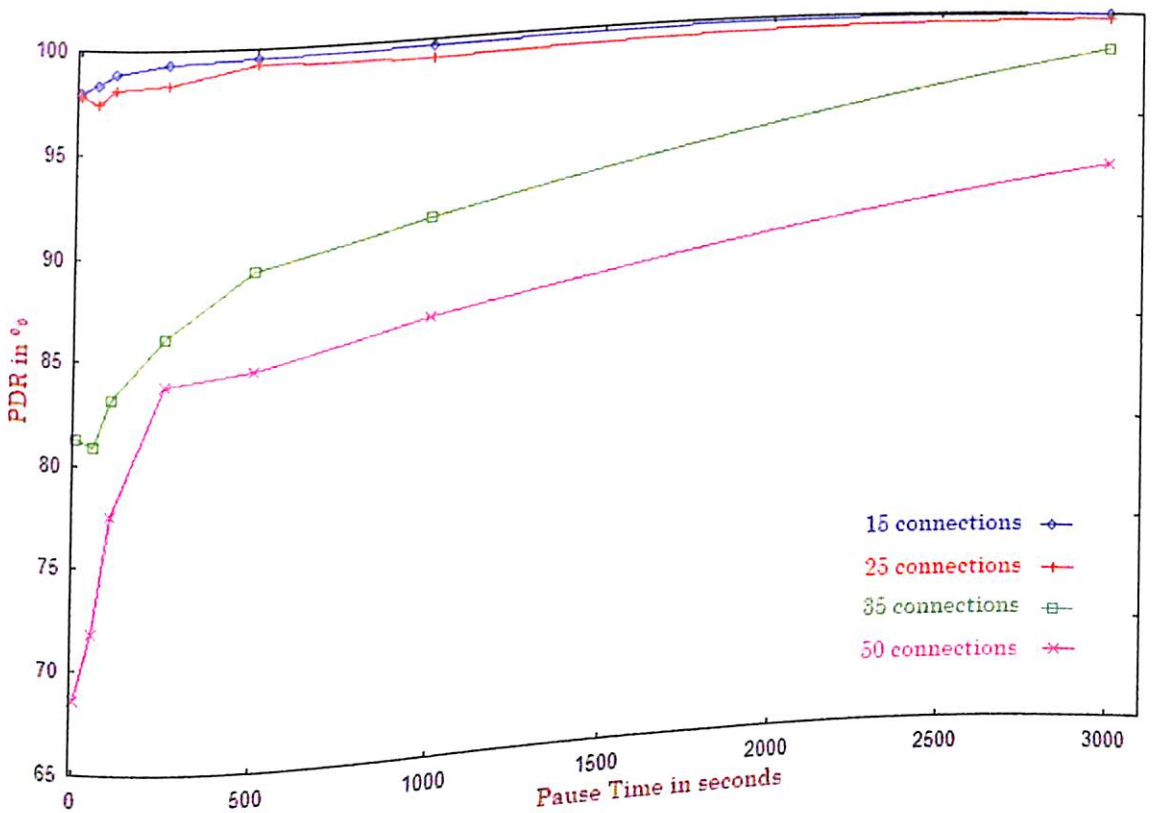


Fig 5.95 PDR Vs Pause Time at a maximum node speed of 10m/s using GPEAR as the routing protocol for varying number of connections

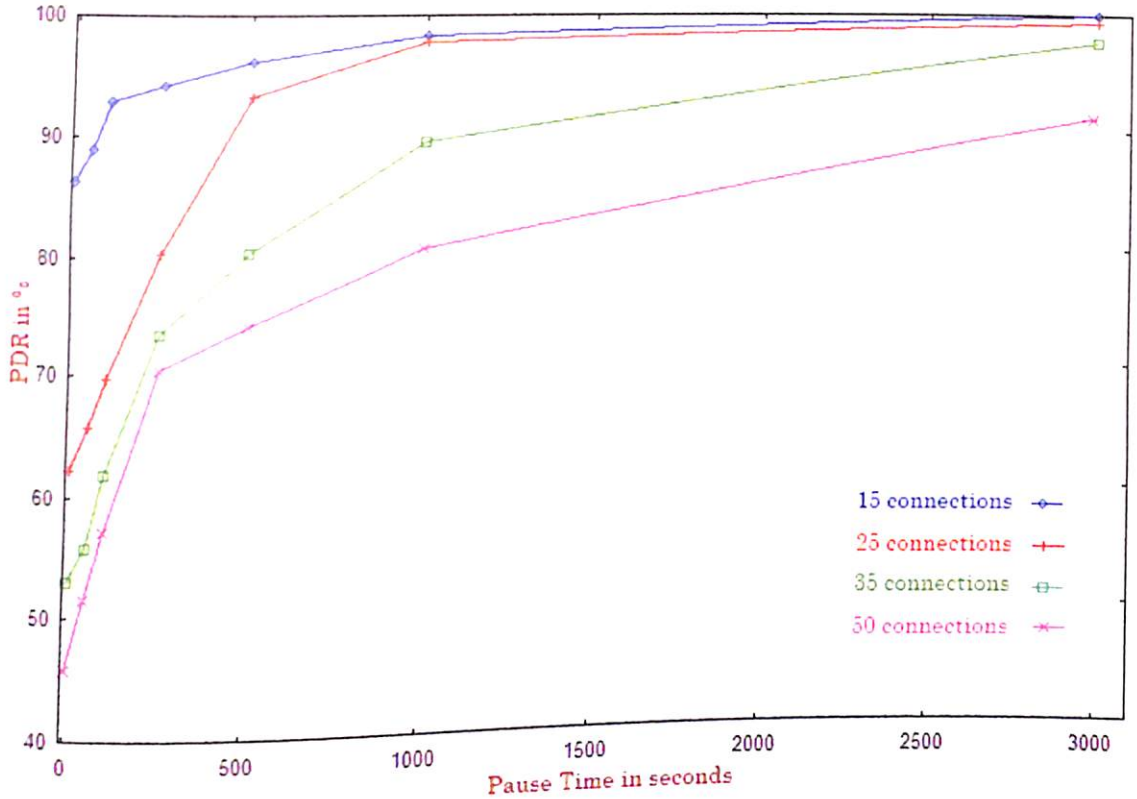


Fig 5.96 PDR Vs Pause Time at a maximum node speed of 15m/s using DSR as the routing protocol for varying number of connections

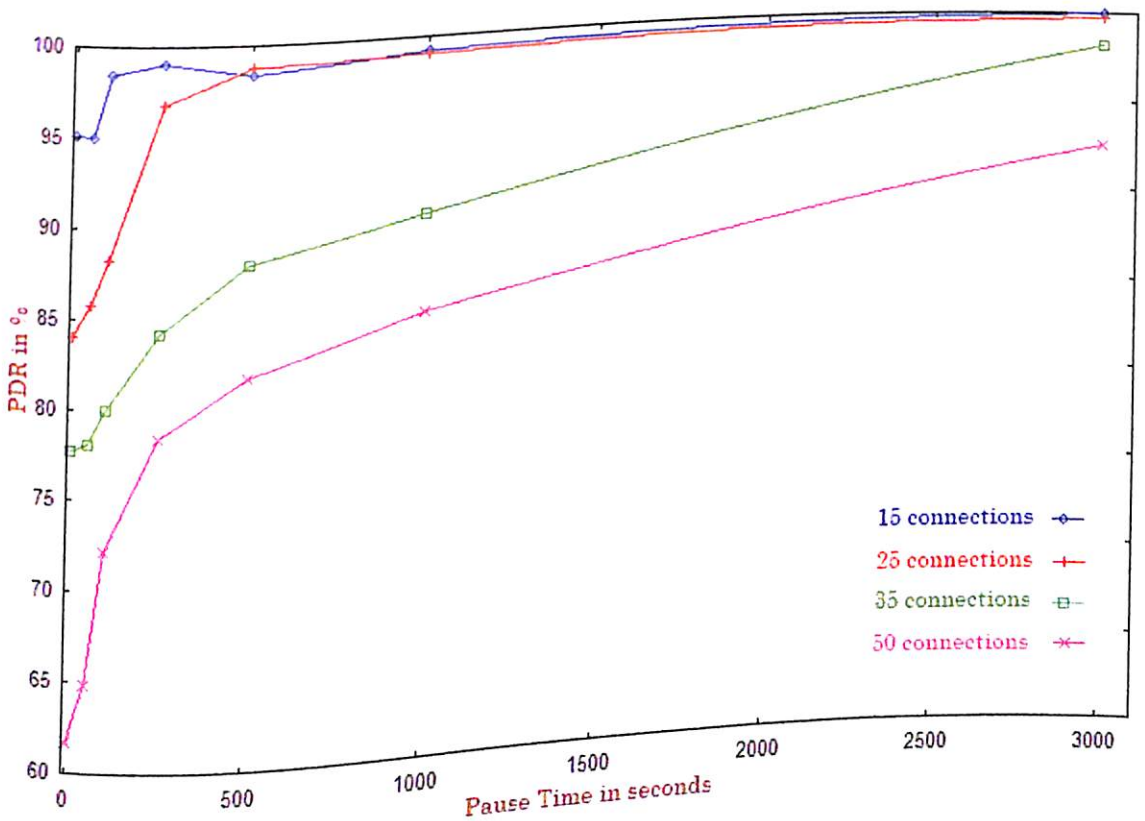


Fig 5.97 PDR Vs Pause Time at a maximum node speed of 15m/s using GPEAR as the routing protocol for varying number of connection

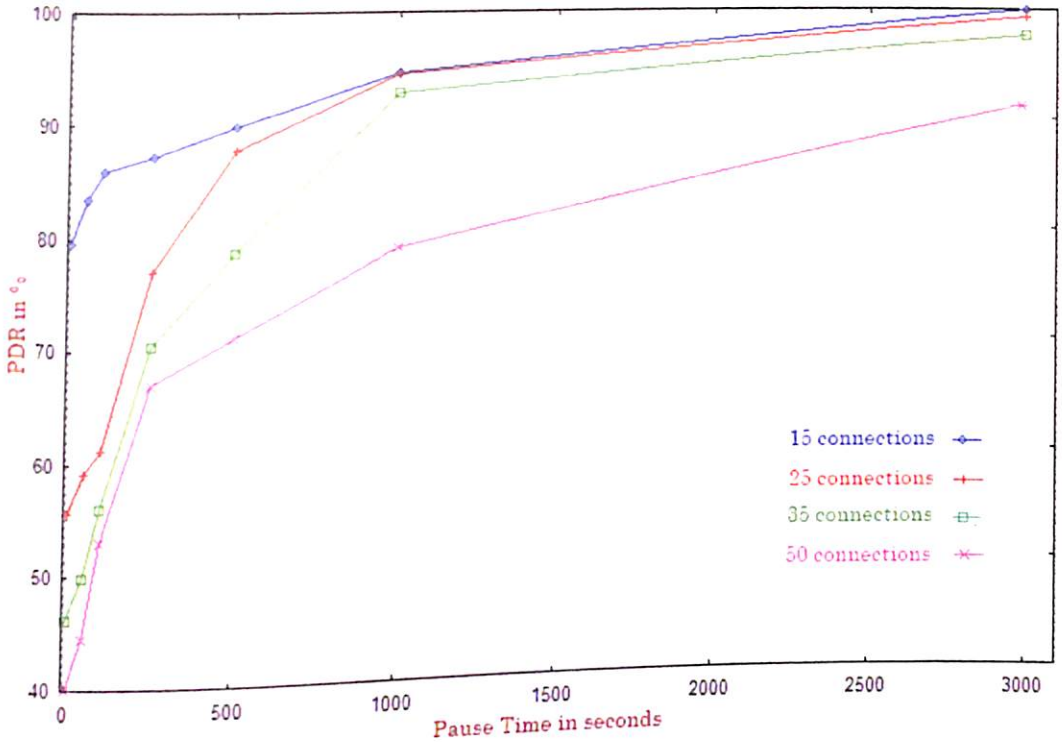


Fig 5.98 PDR Vs Pause Time at a maximum node speed of 20m/s using DSR as the routing protocol for varying number of connection

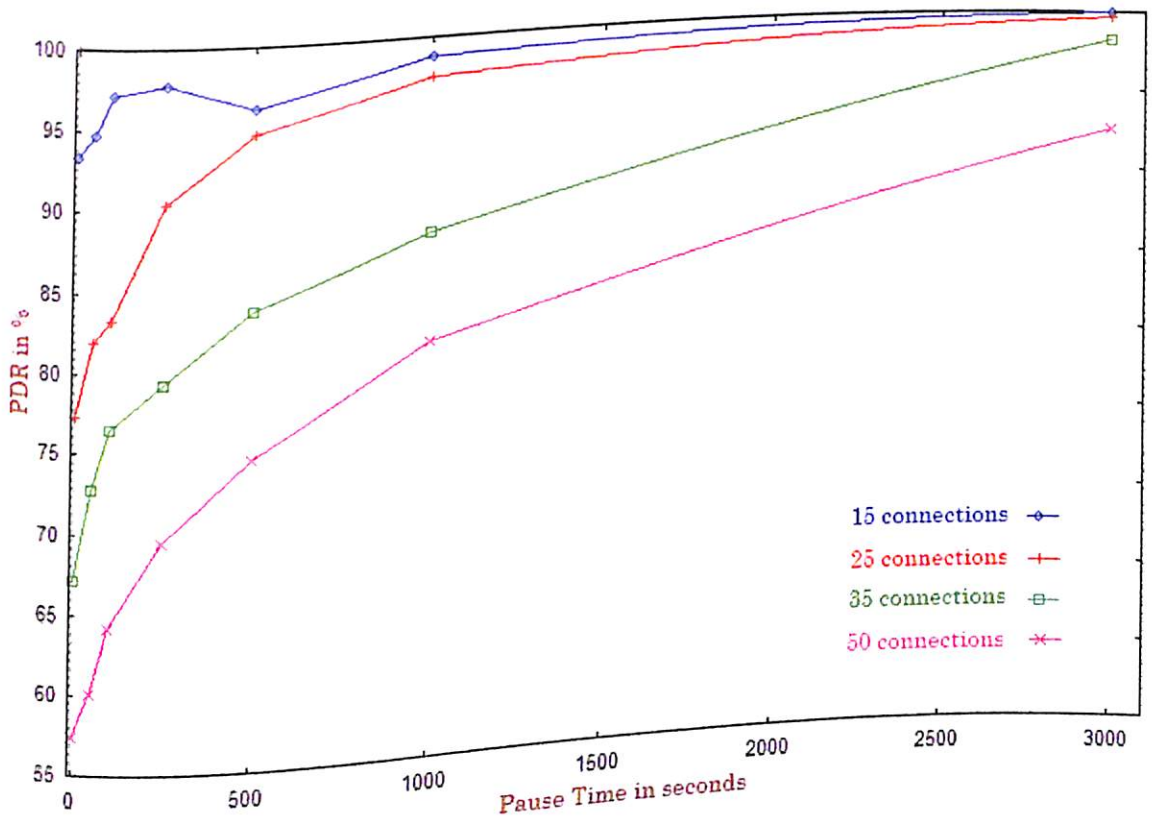


Fig 5.99 PDR Vs Pause Time at a maximum node speed of 20m/s using GPEAR as the routing protocol for varying number of connection

Analysis of GPEAR Network Throughput (fig. 5.102 - fig. 5.113)

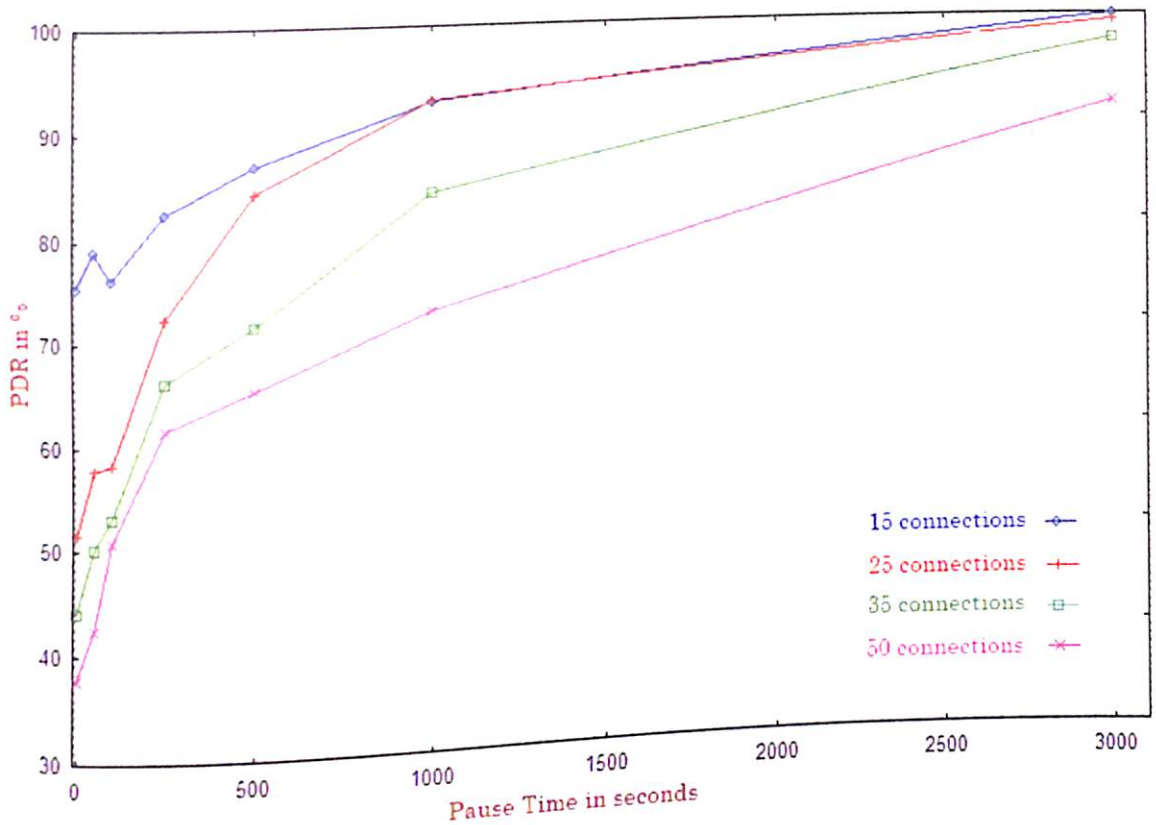


Fig 5.100 PDR Vs Pause Time at a maximum node speed of 30m/s using DSR as the routing protocol for varying number of connection

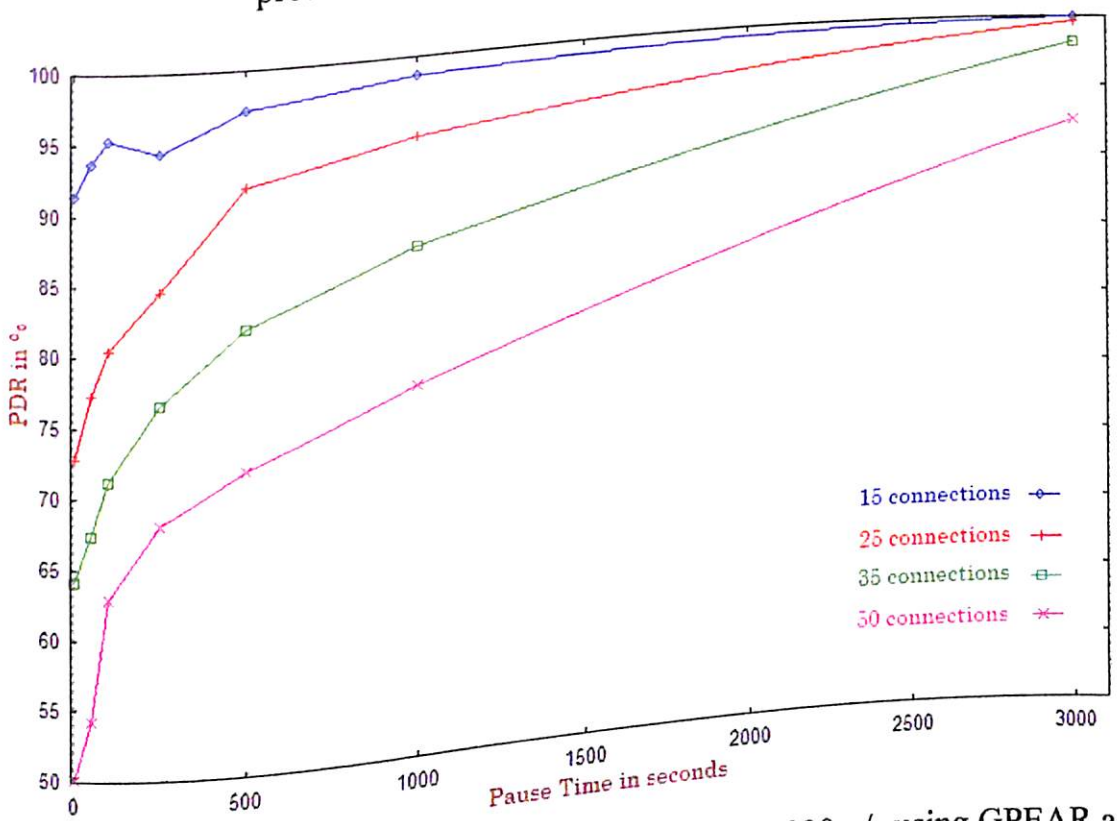


Fig 5.101 PDR Vs Pause Time at a maximum node speed of 30m/s using GPEAR as the routing protocol for varying number of connection

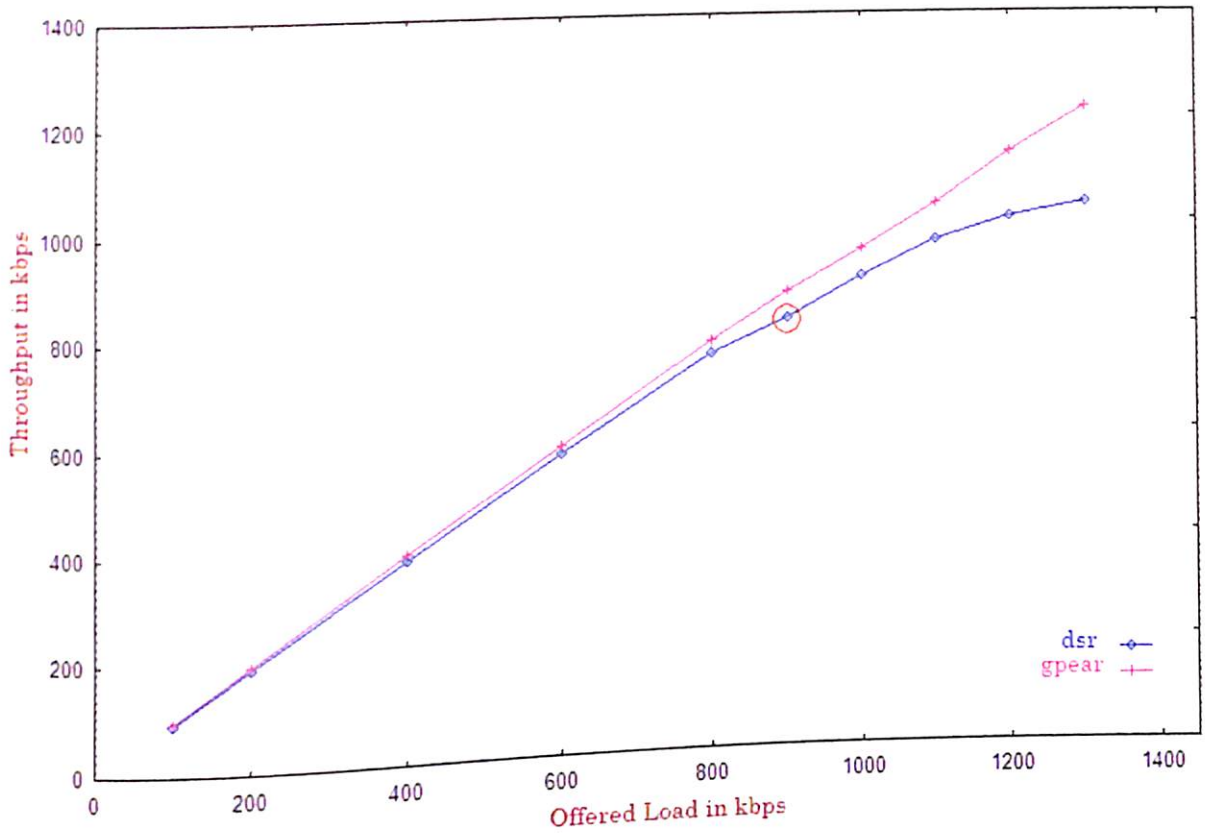


Fig 5.102 Network Throughput at a maximum node speed of 1m/s

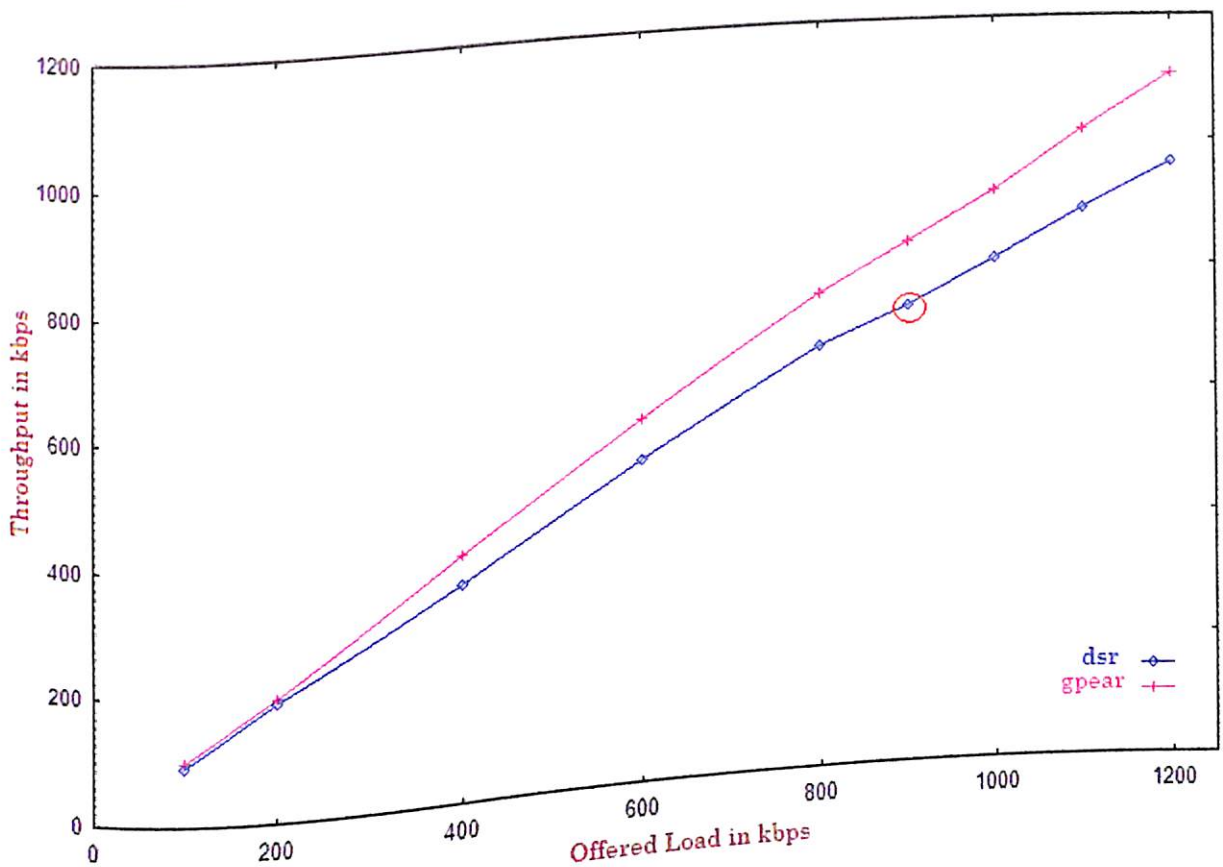


Fig 5.103 Network Throughput at a maximum node speed of 5m/s

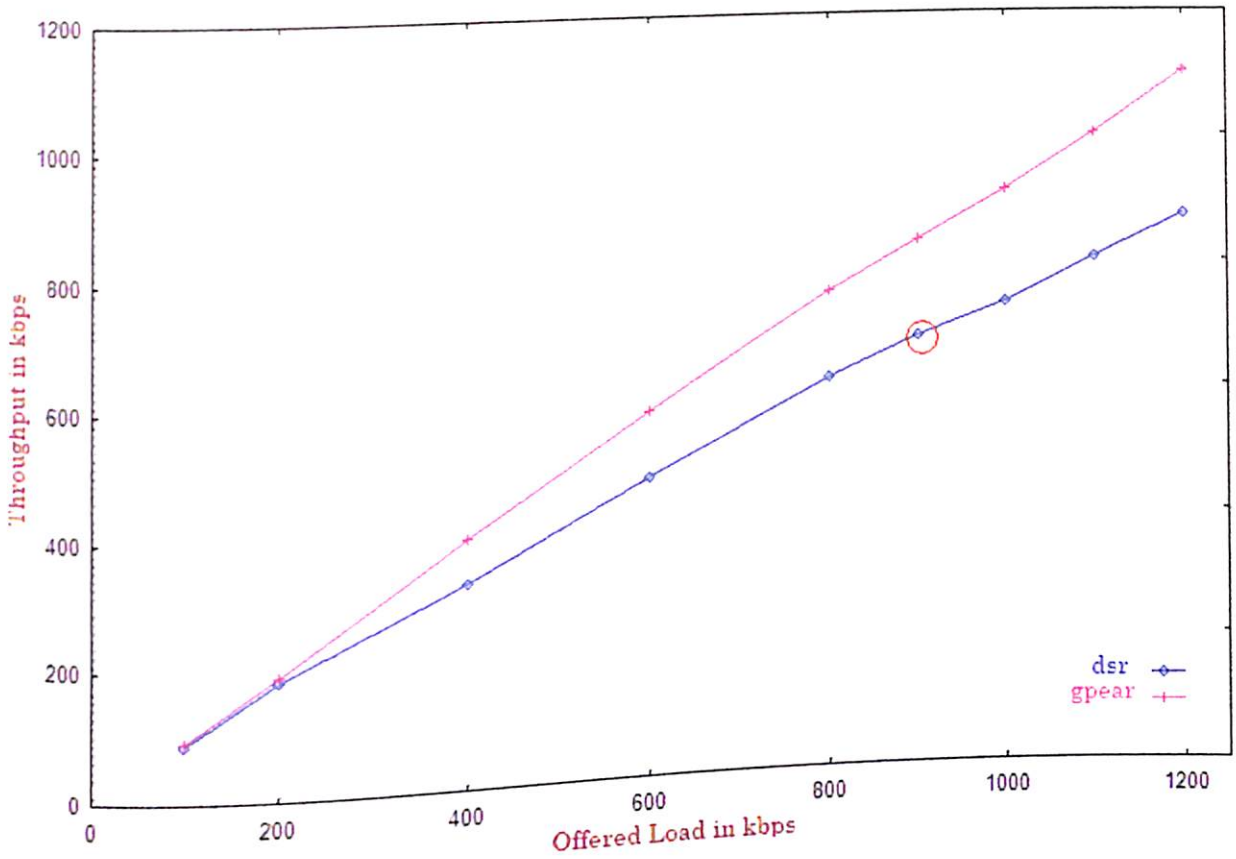


Fig 5.104 Network Throughput at a maximum node speed of 10m/s

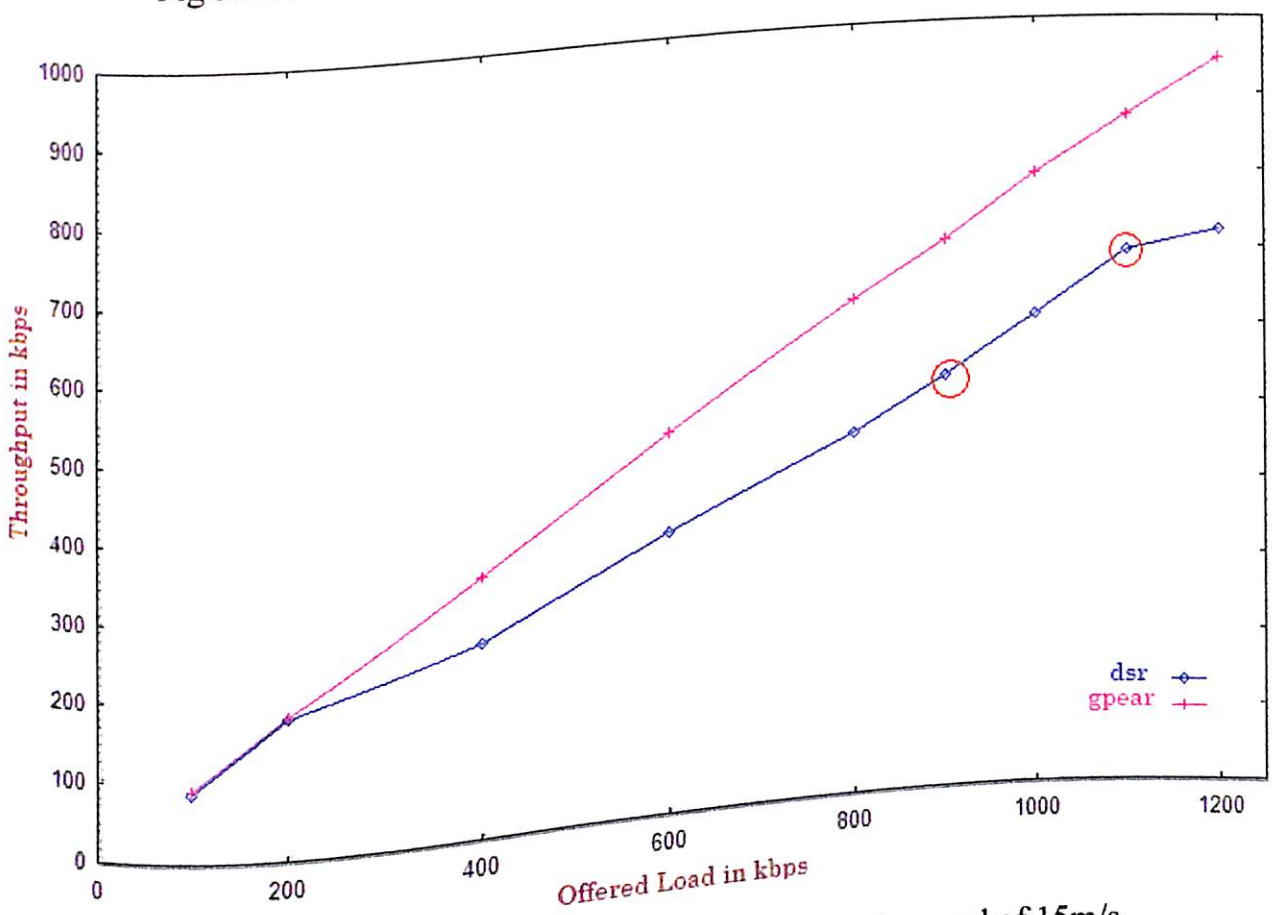


Fig 5.105 Network Throughput at a maximum node speed of 15m/s

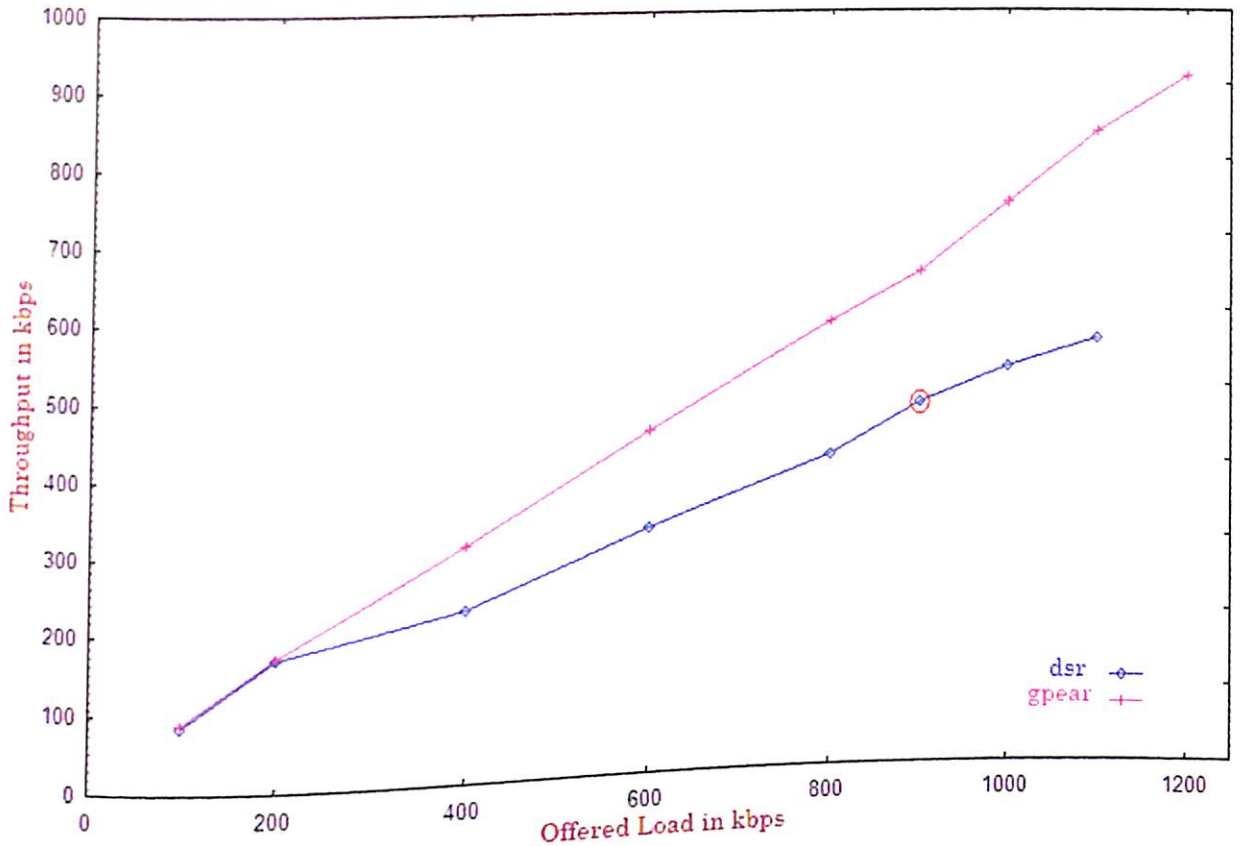


Fig 5.106 Network Throughput at a maximum node speed of 20m/s

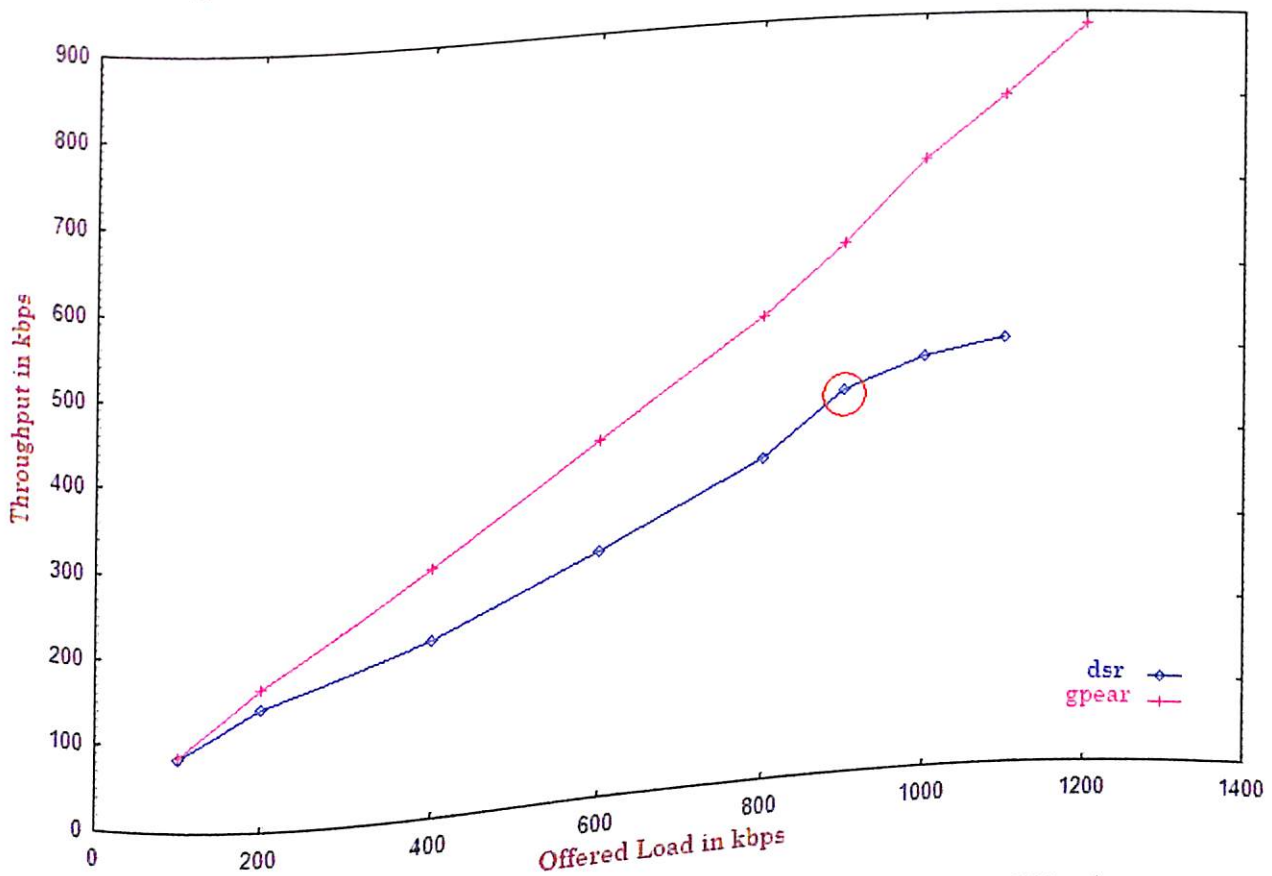


Fig 5.107 Network Throughput at a maximum node speed of 30m/s

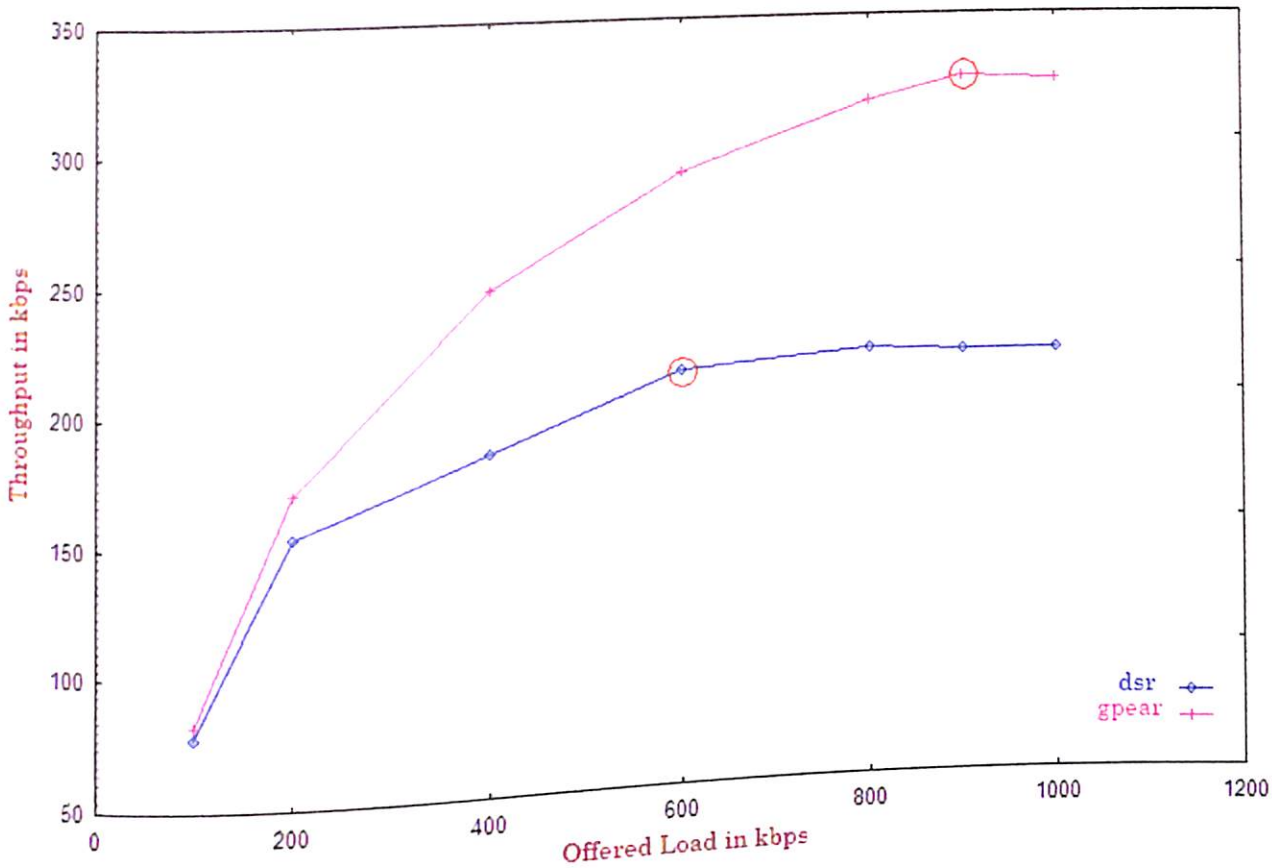


Fig 5.108 Network Throughput at a maximum node speed of 15m/s with 50 connections

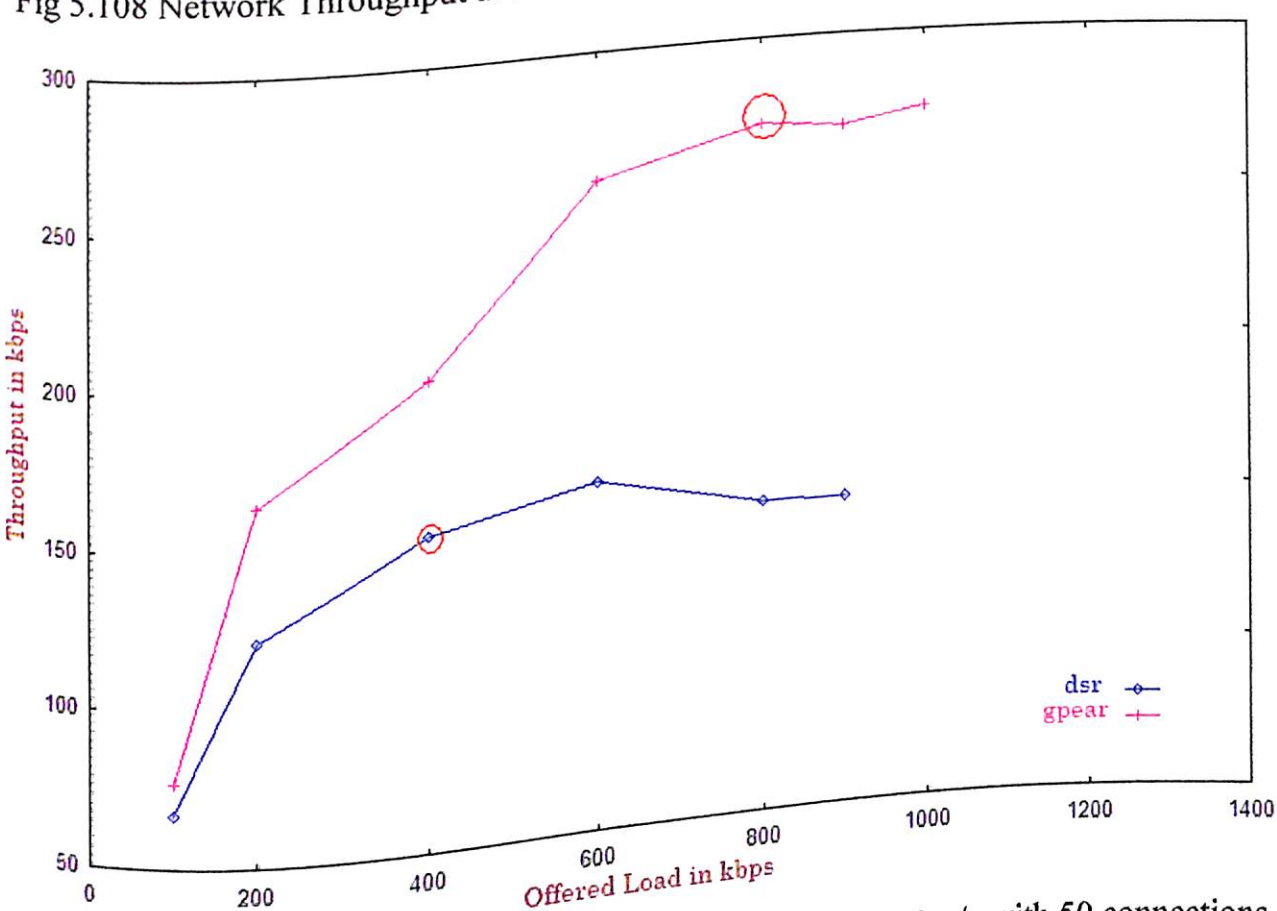


Fig 5.109 Network Throughput at a maximum node speed of 30m/s with 50 connections

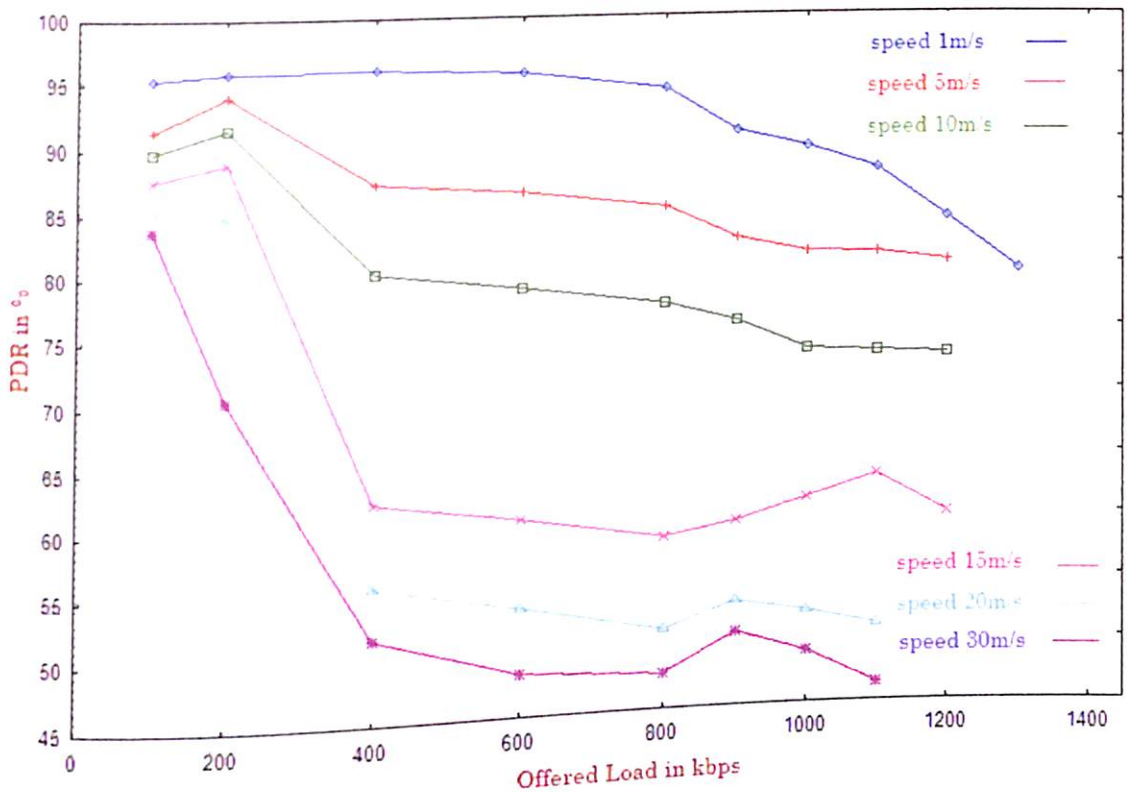


Fig 5.110 PDR as a function of load on the network with 25 connections for varying speeds with DSR as the routing protocol

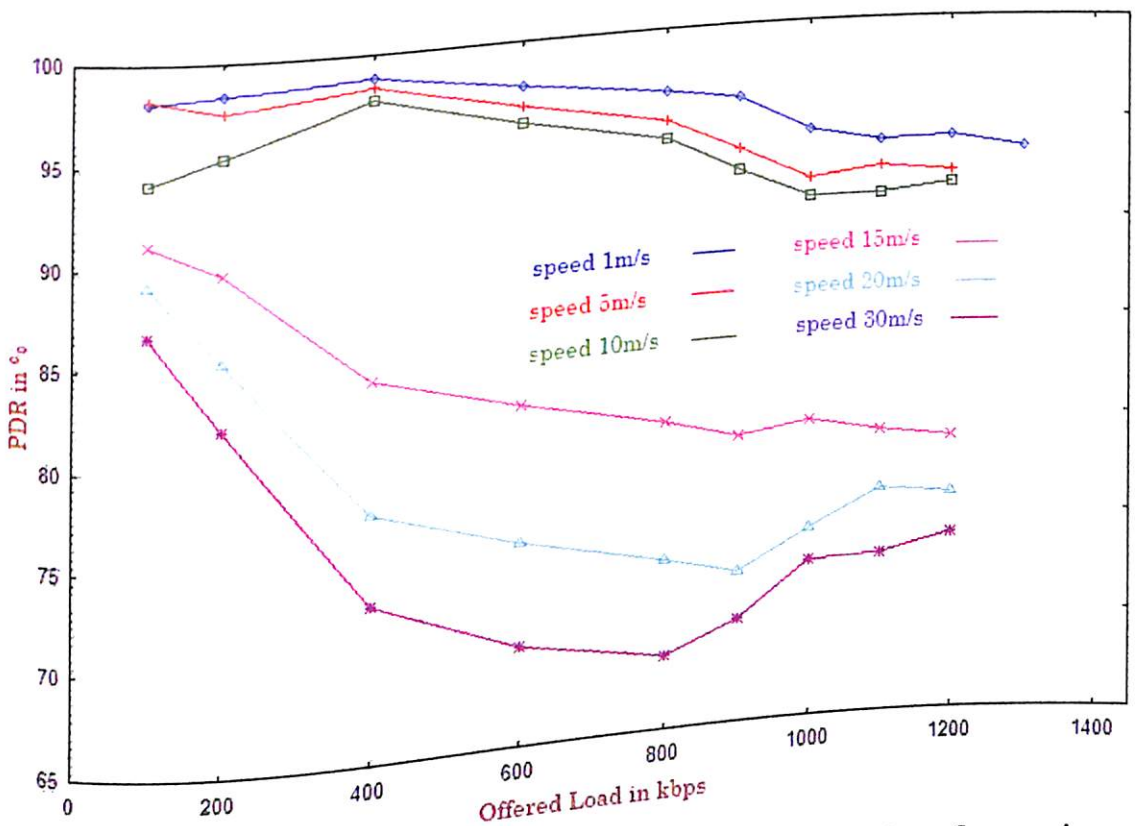


Fig 5.111 PDR as a function of load on the network with 25 connections for varying speeds with GPEAR as the routing protocol

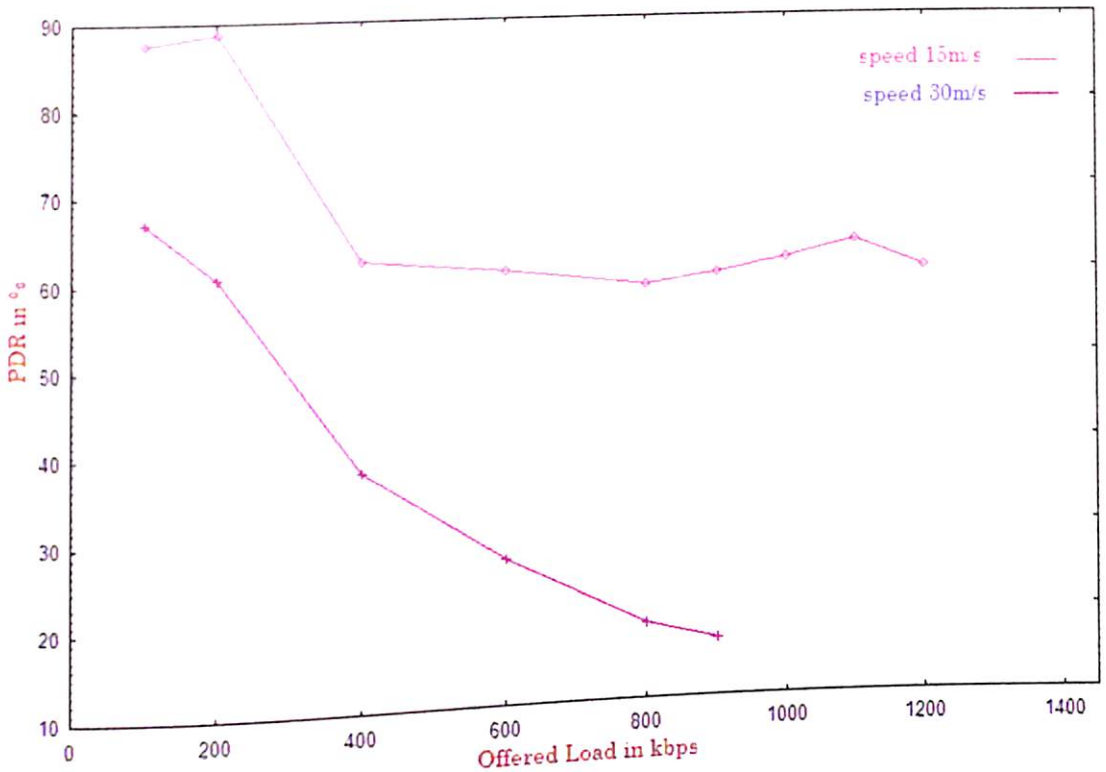


Fig 5.112 PDR as a function of load on the network with 50 connections for varying speeds with DSR as the routing protocol

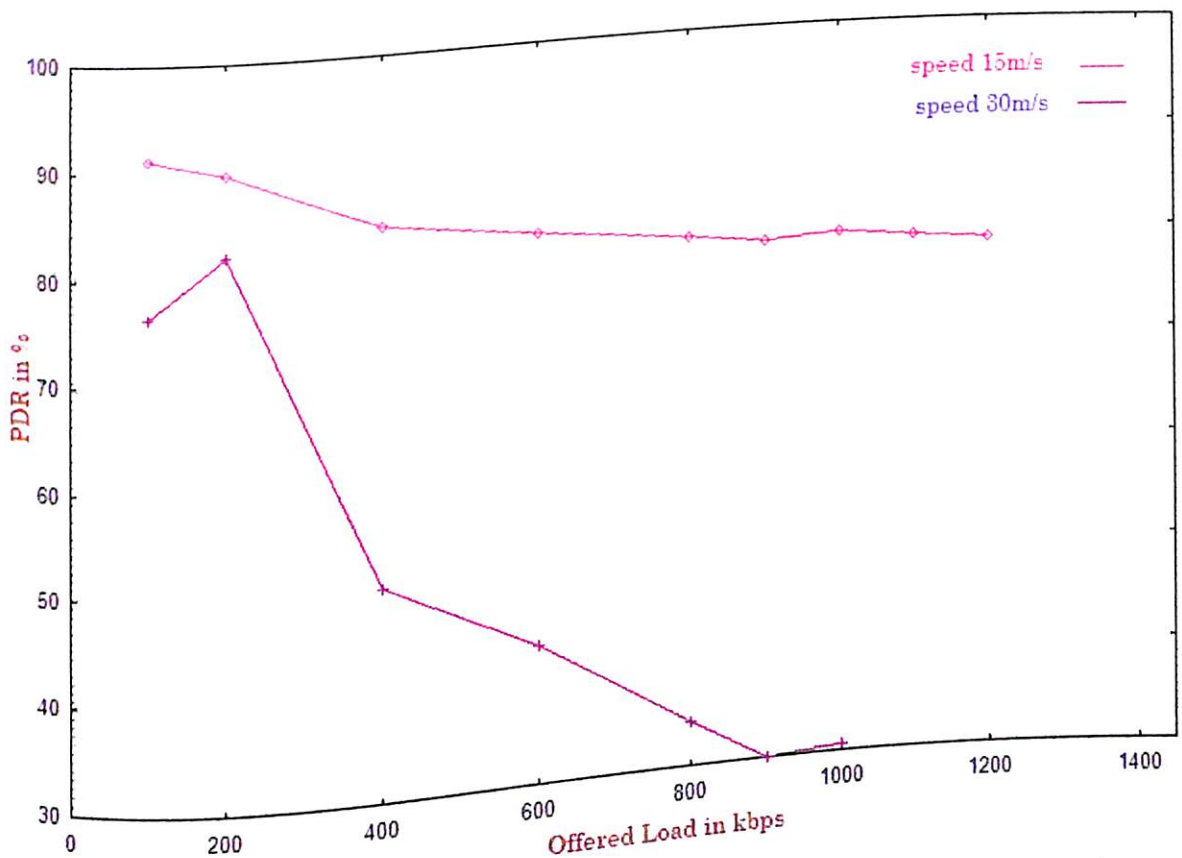


Fig 5.113 PDR as a function of load on the network with 50 connections for varying speeds with GPEAR as the routing protocol

Analysis of GPEAR Control Overhead (fig. 5.114 - fig. 5.135)

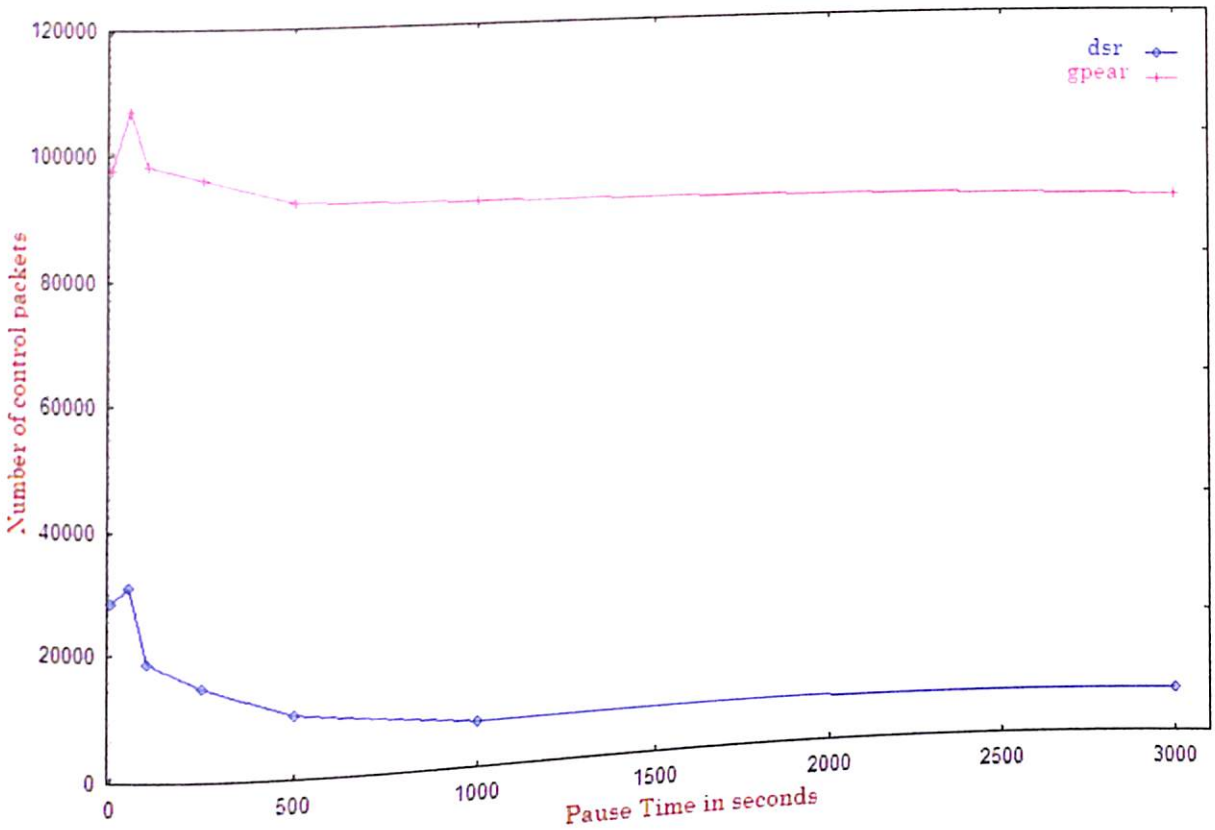


Fig5.114 Number of Control Packets Vs Pause Time at a maximum node speed of 1m/s

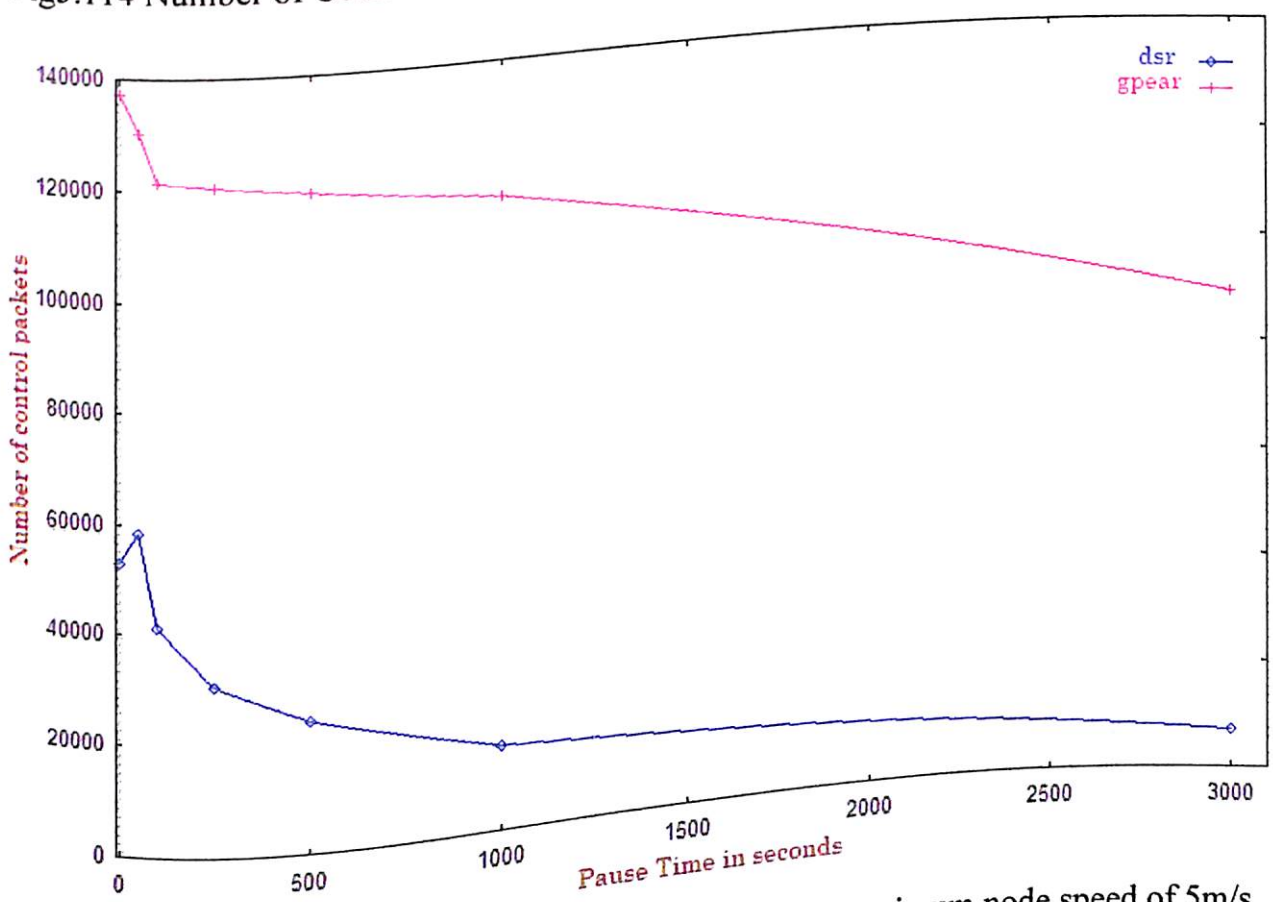


Fig5.115 Number of Control Packets Vs Pause Time at a maximum node speed of 5m/s

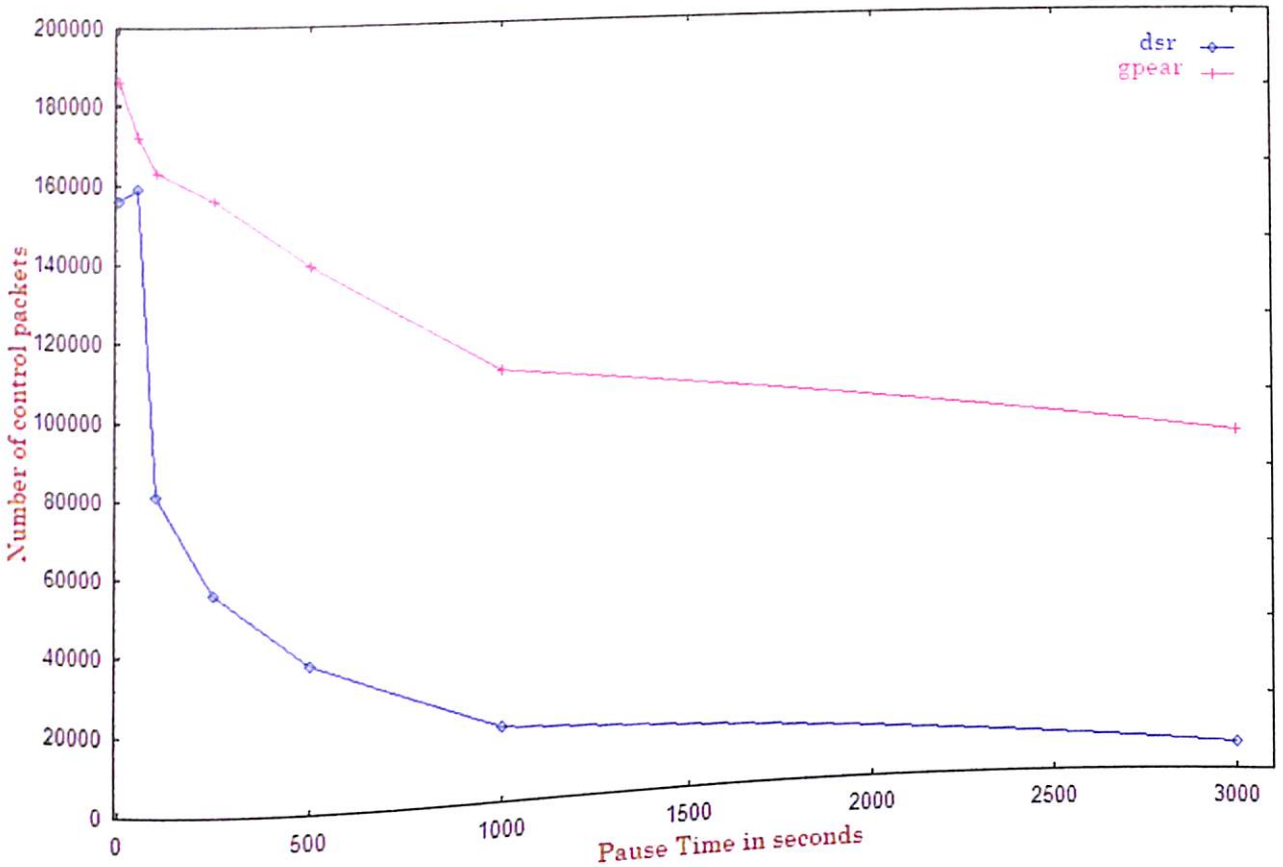


Fig5.116 Number of Control Packets Vs Pause Time at a maximum node speed of 10m/s

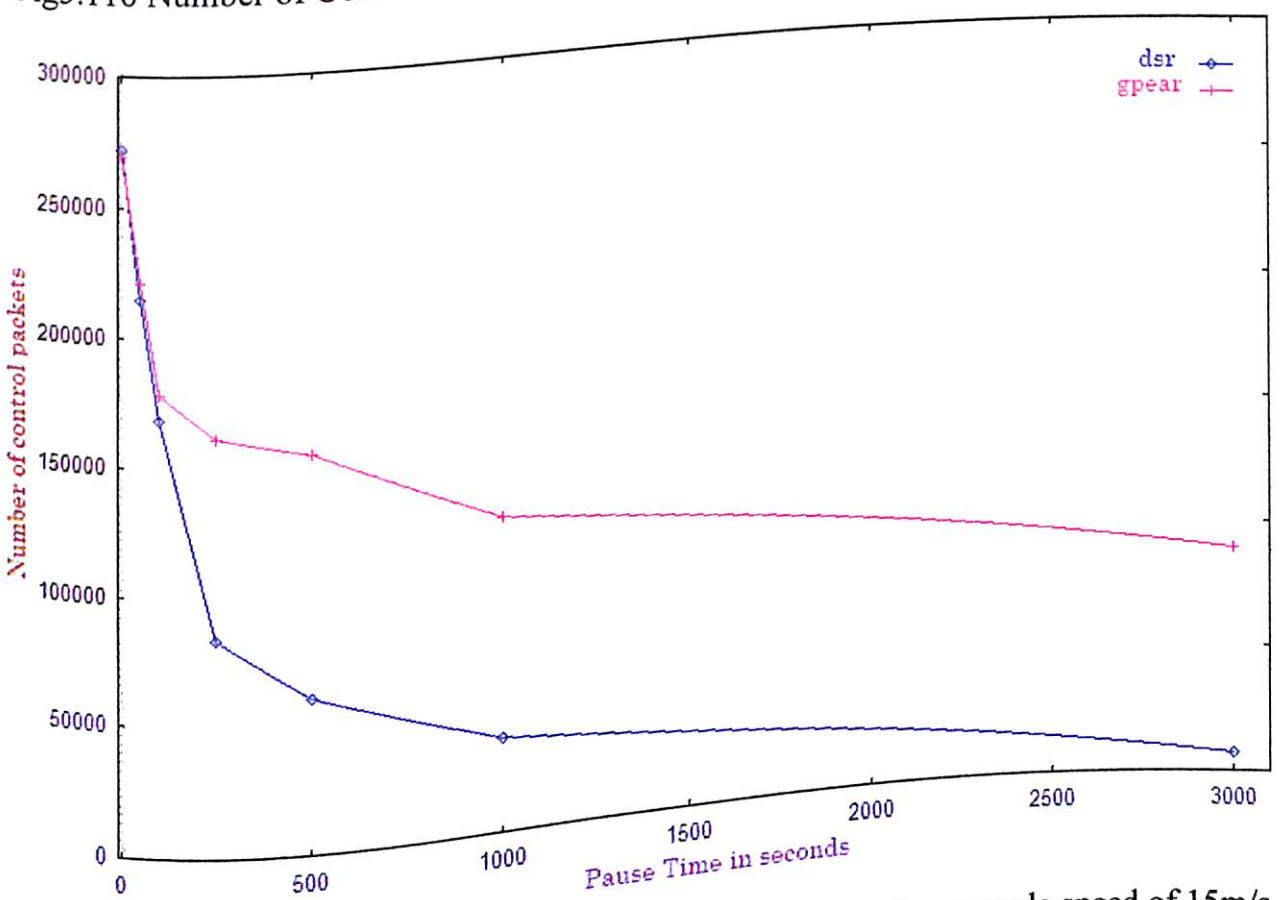


Fig5.117 Number of Control Packets Vs Pause Time at a maximum node speed of 15m/s

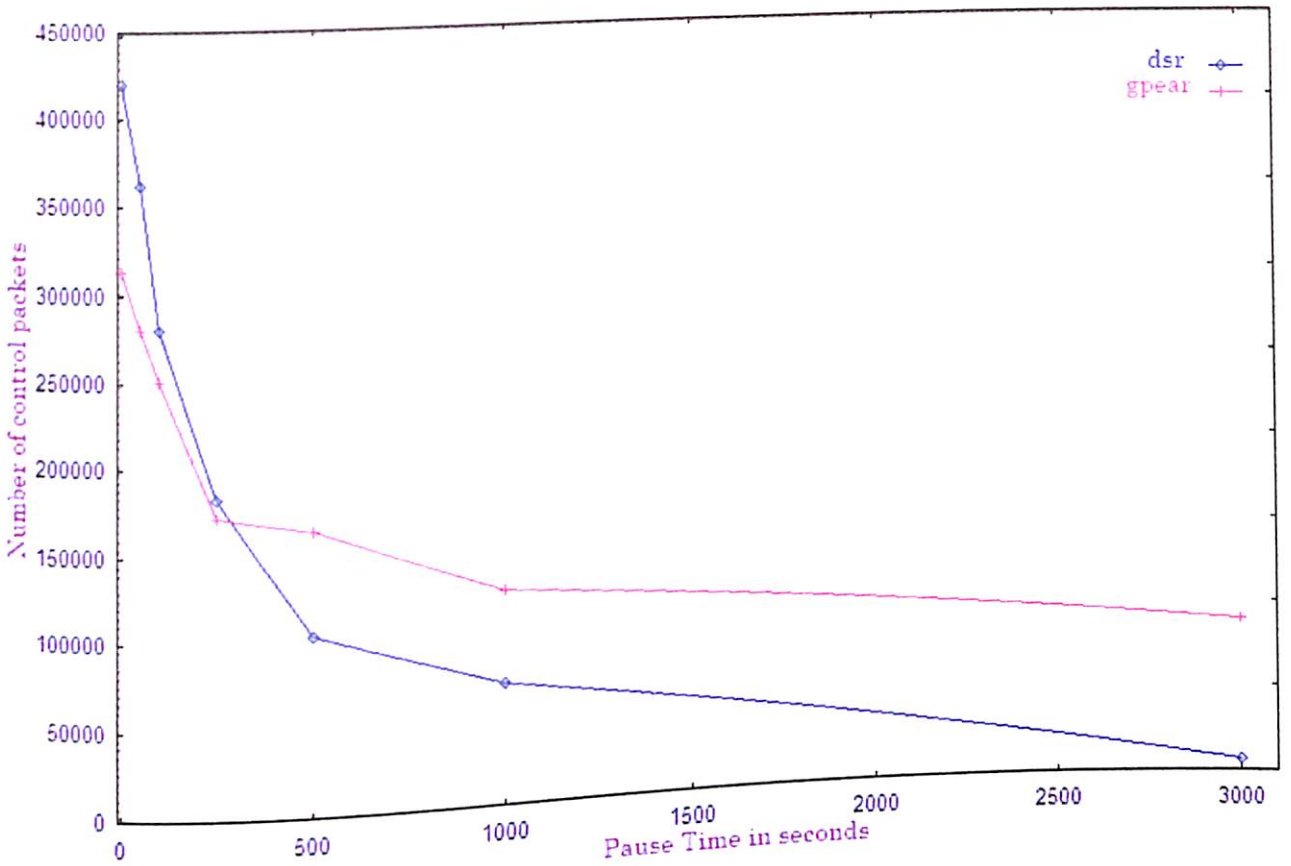


Fig5.118 Number of Control Packets Vs Pause Time at a maximum node speed of 20m/s

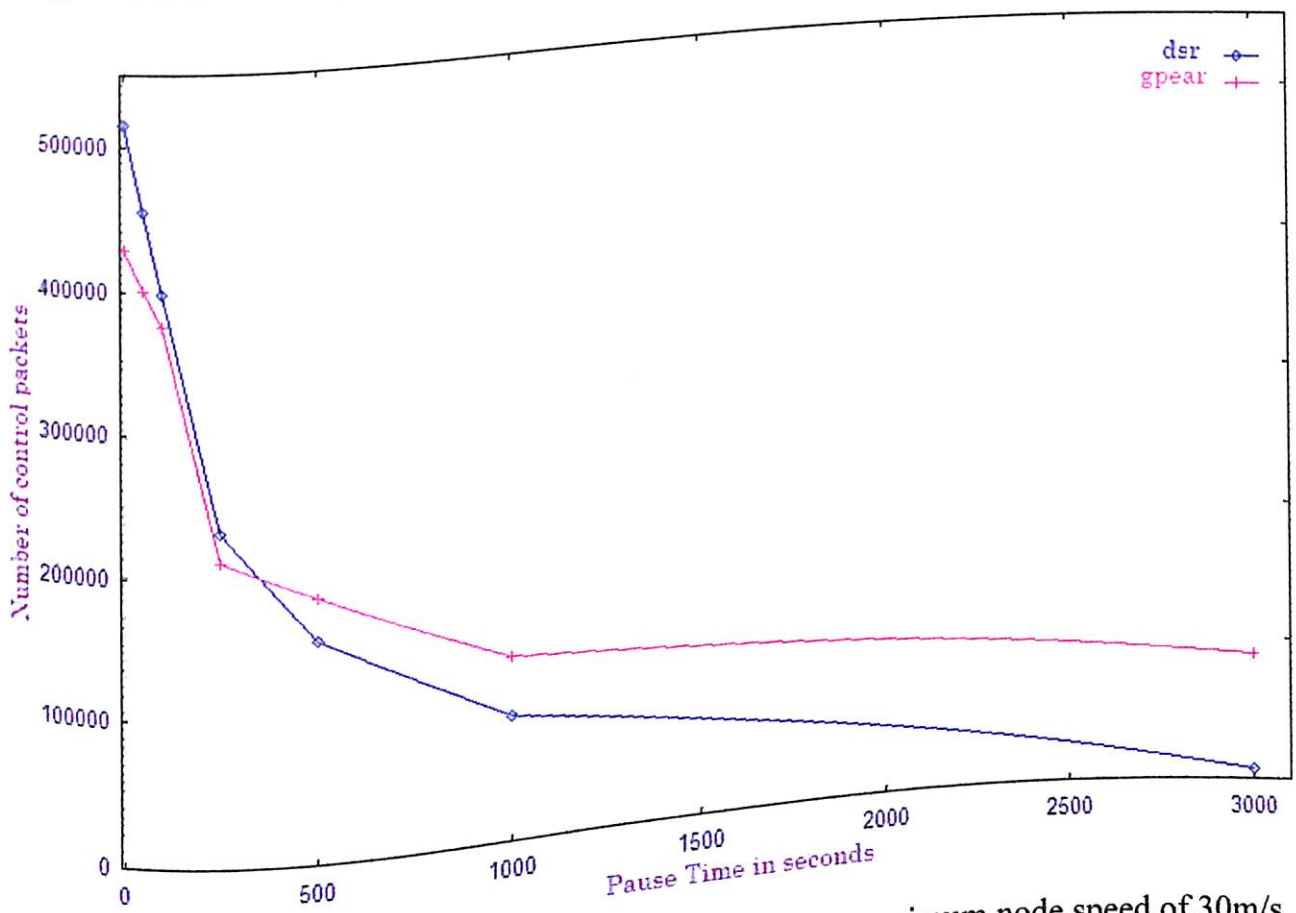


Fig5.119 Number of Control Packets Vs Pause Time at a maximum node speed of 30m/s

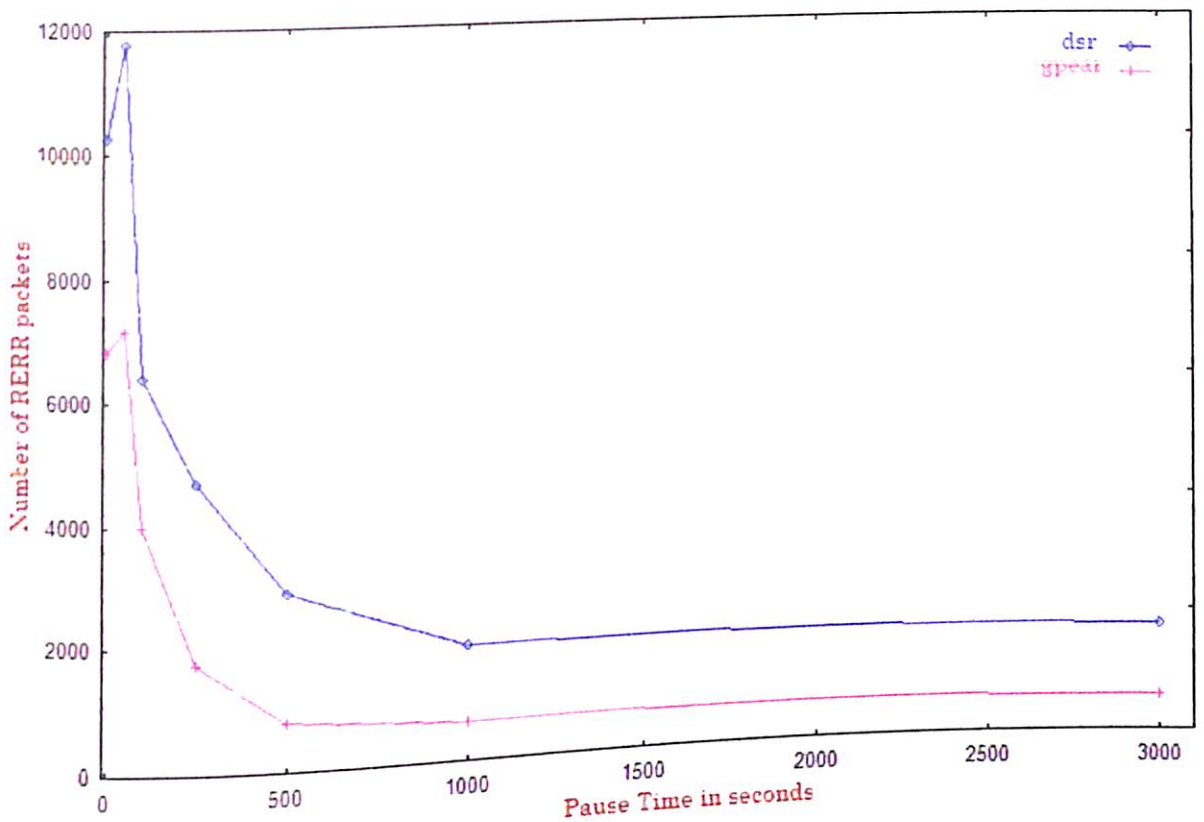


Fig5.120 Number of RERR Packets Vs Pause Time at a maximum node speed of 1m/s

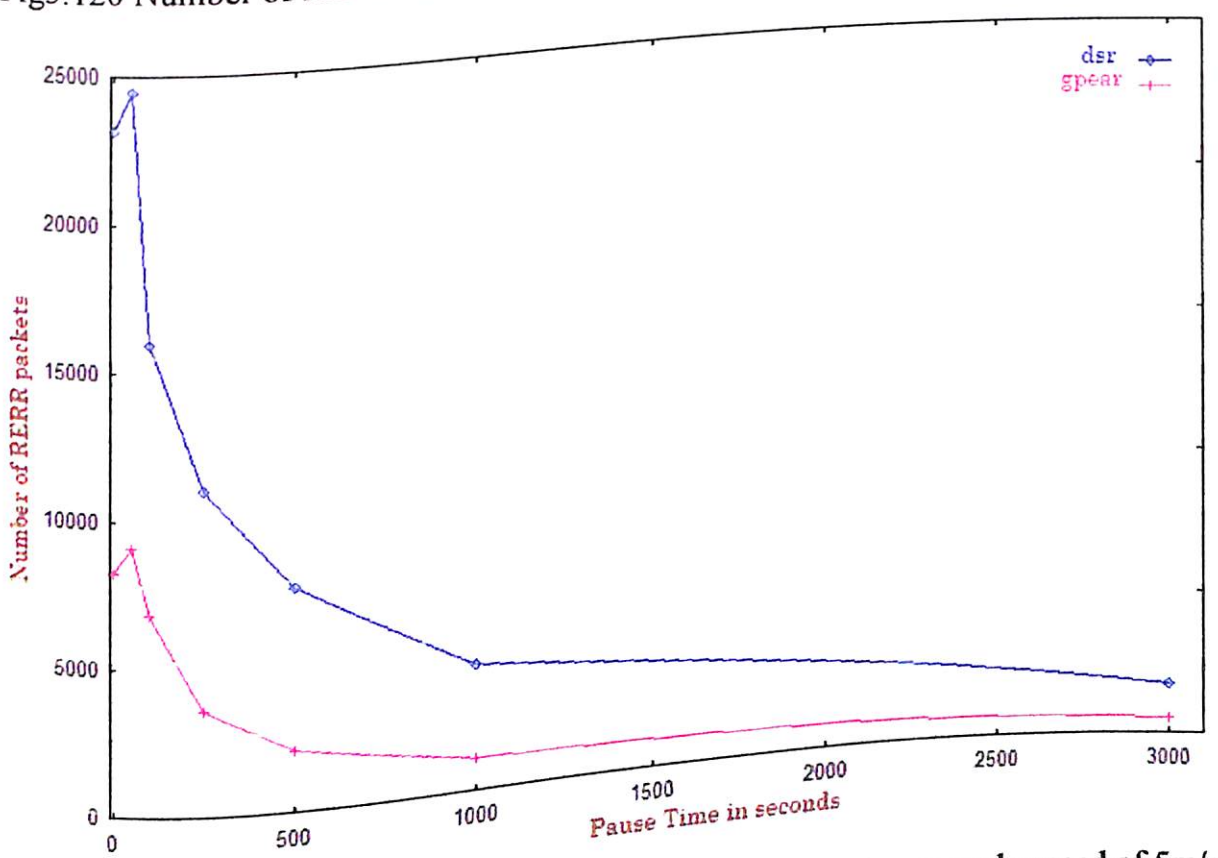


Fig5.121 Number of RERR Packets Vs Pause Time at a maximum node speed of 5m/s

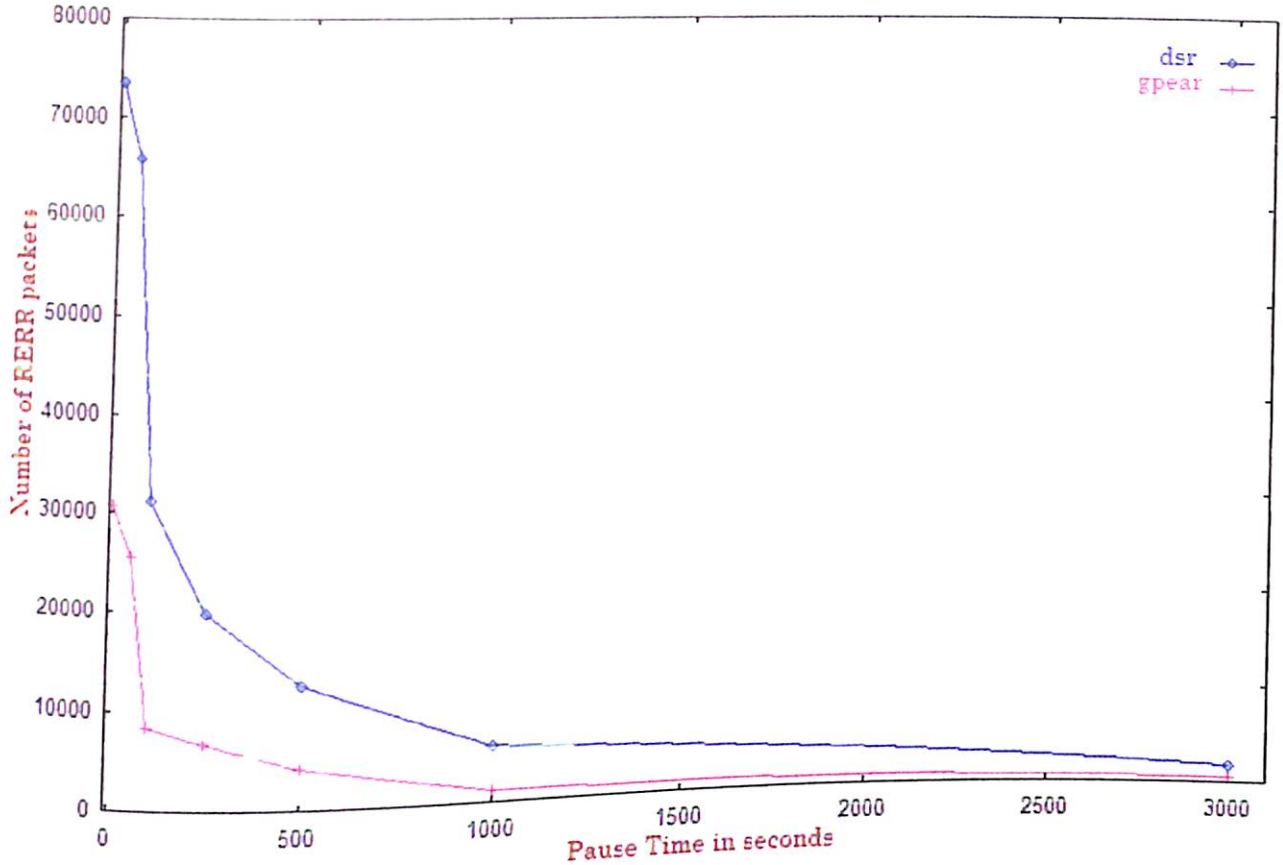


Fig5.122 Number of RERR Packets Vs Pause Time at a maximum node speed of 10m/s

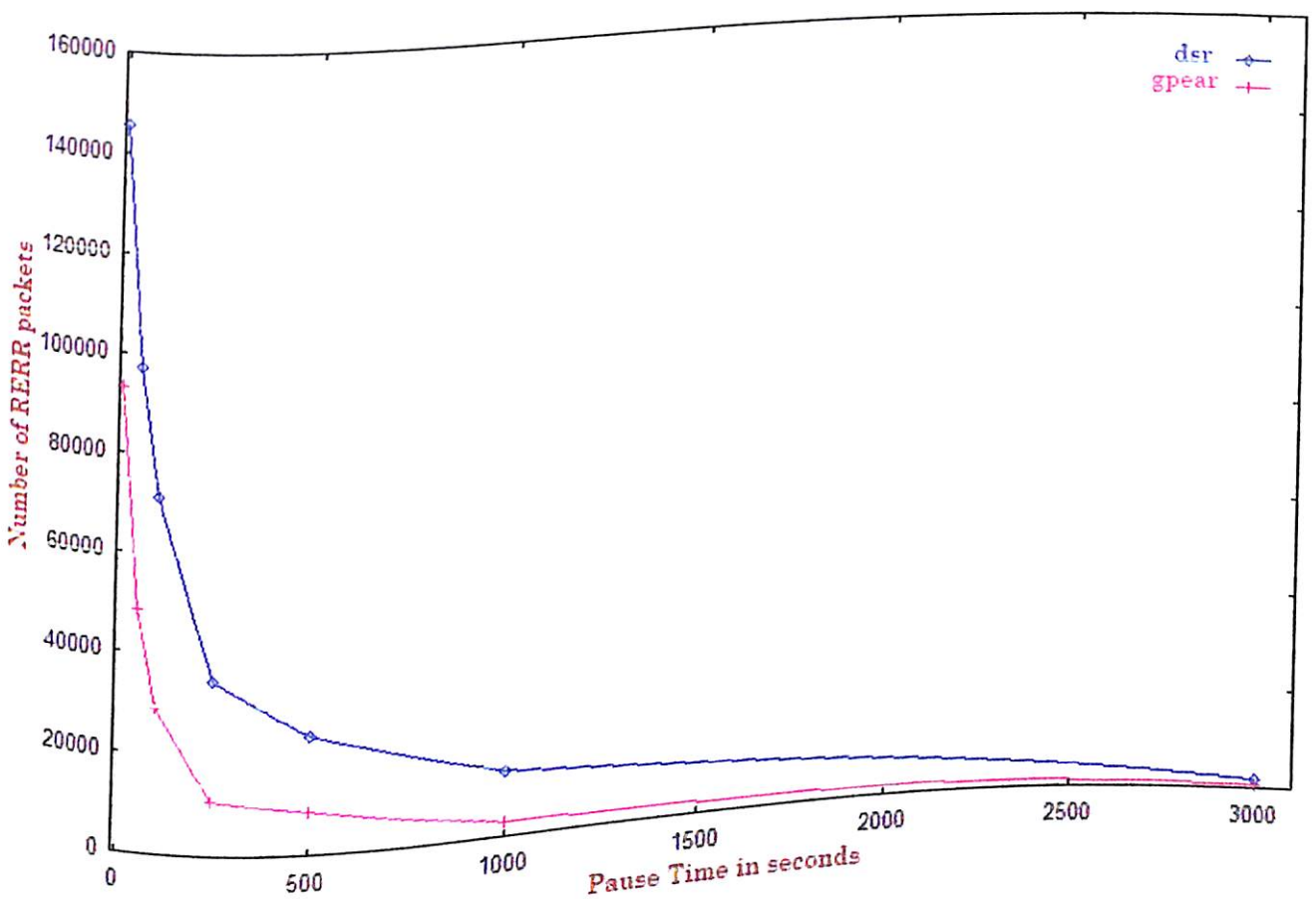


Fig5.123 Number of RERR Packets Vs Pause Time at a maximum node speed of 15m/s

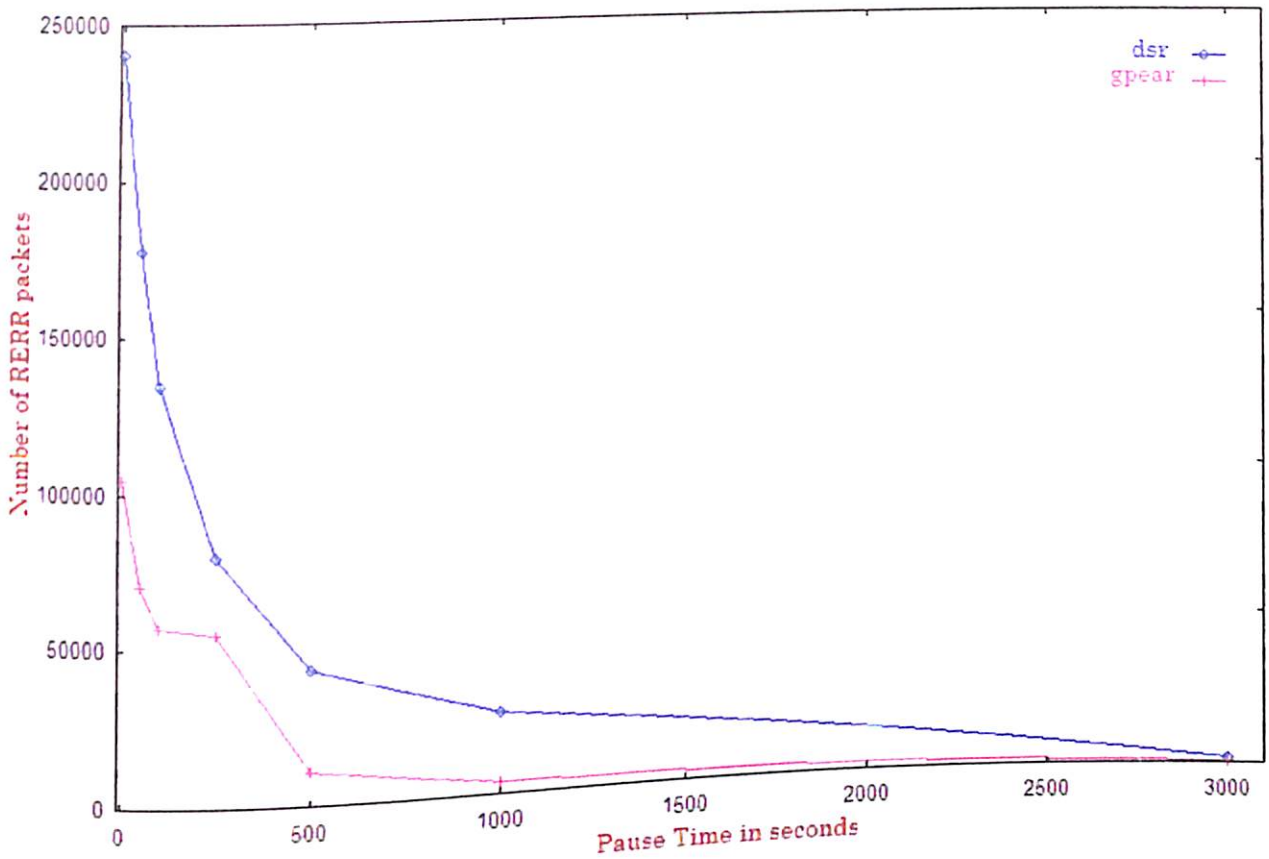


Fig5.124 Number of RERR Packets Vs Pause Time at a maximum node speed of 20m/s

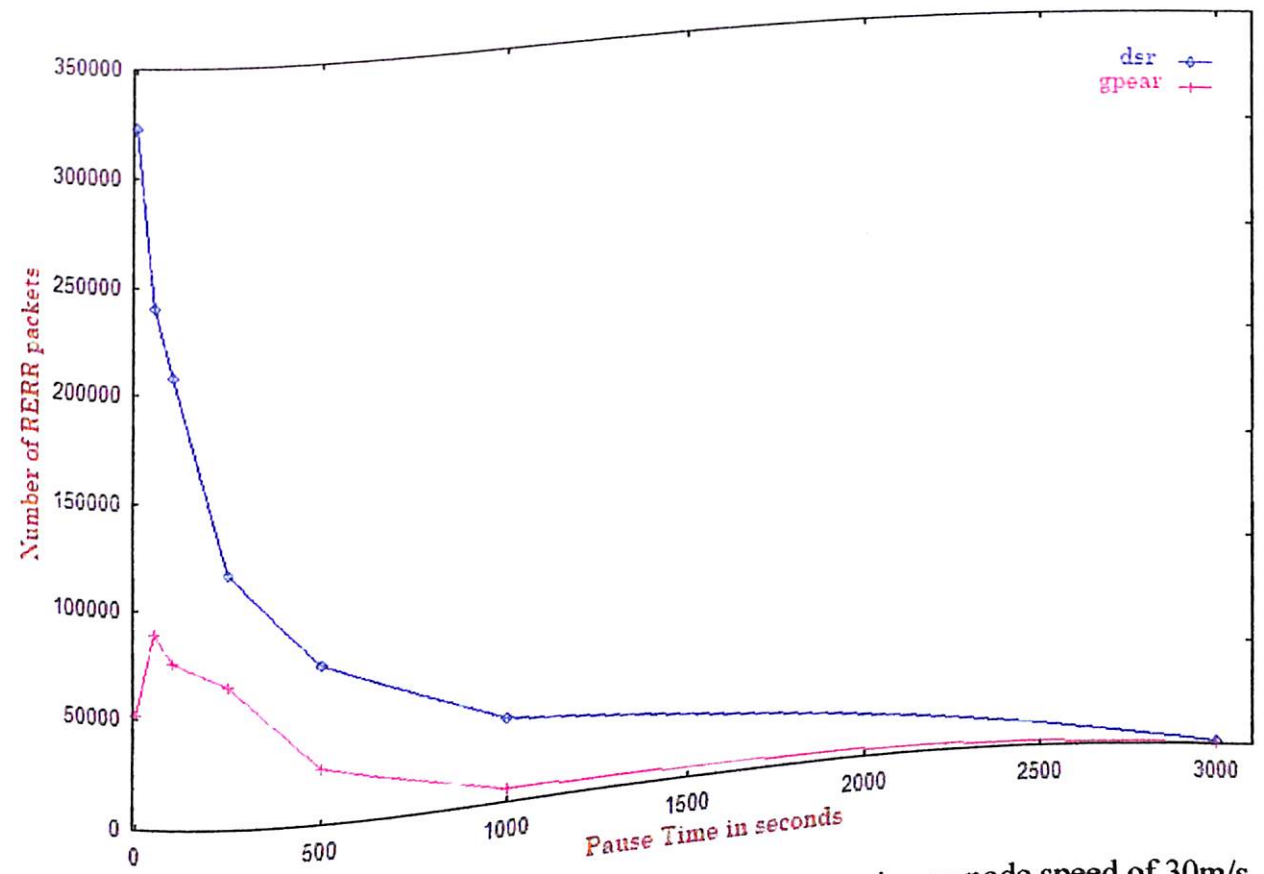


Fig5.125 Number of RERR Packets Vs Pause Time at a maximum node speed of 30m/s

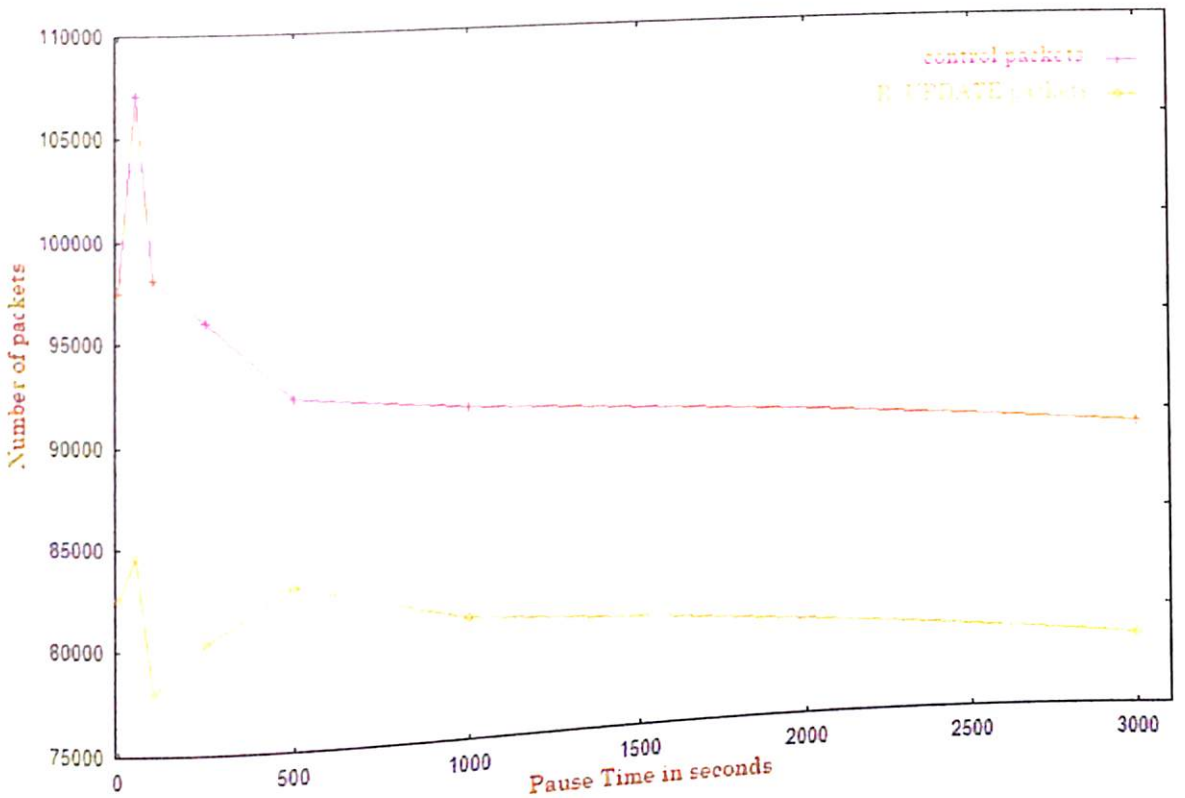


Fig5.126 Number of R_UPDATE and Control Packets Vs Pause Time at a maximum node speed of 1m/s

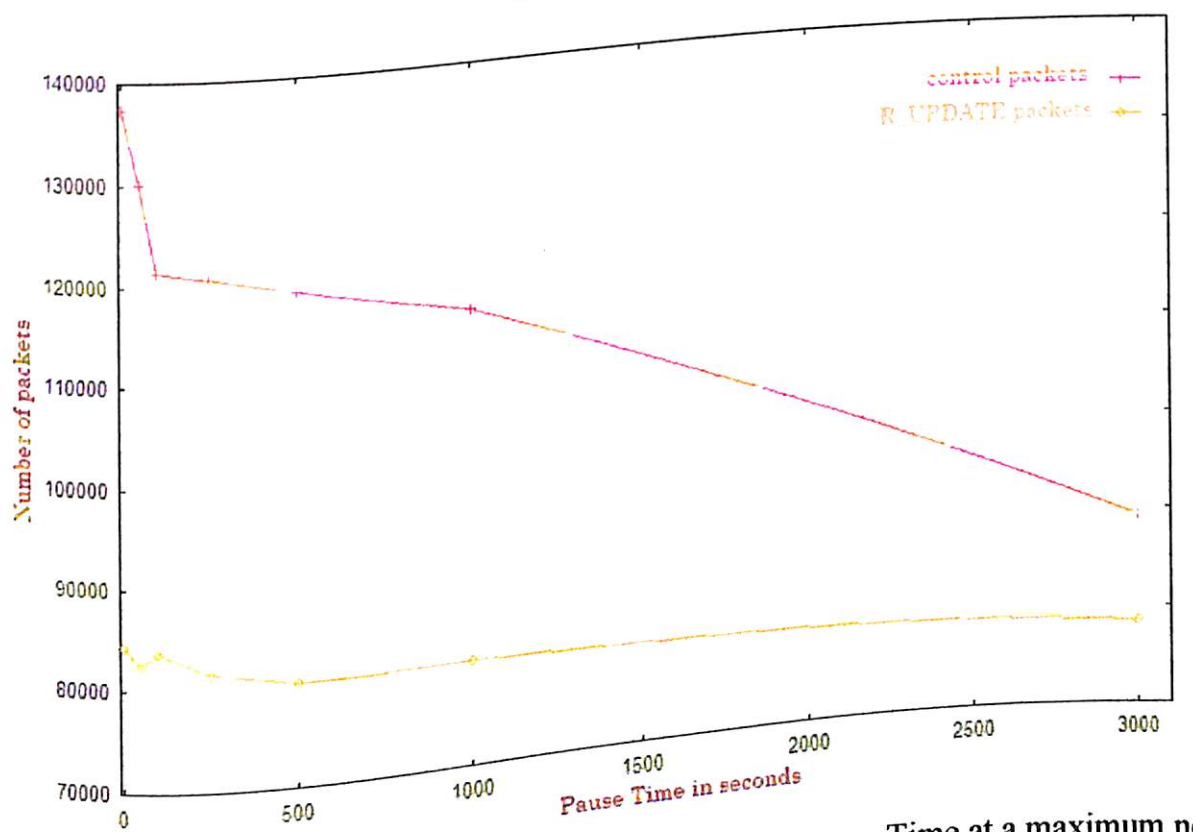


Fig 5.127 Number of R_UPDATE and Control Packets Vs Pause Time at a maximum node speed of 5m/s

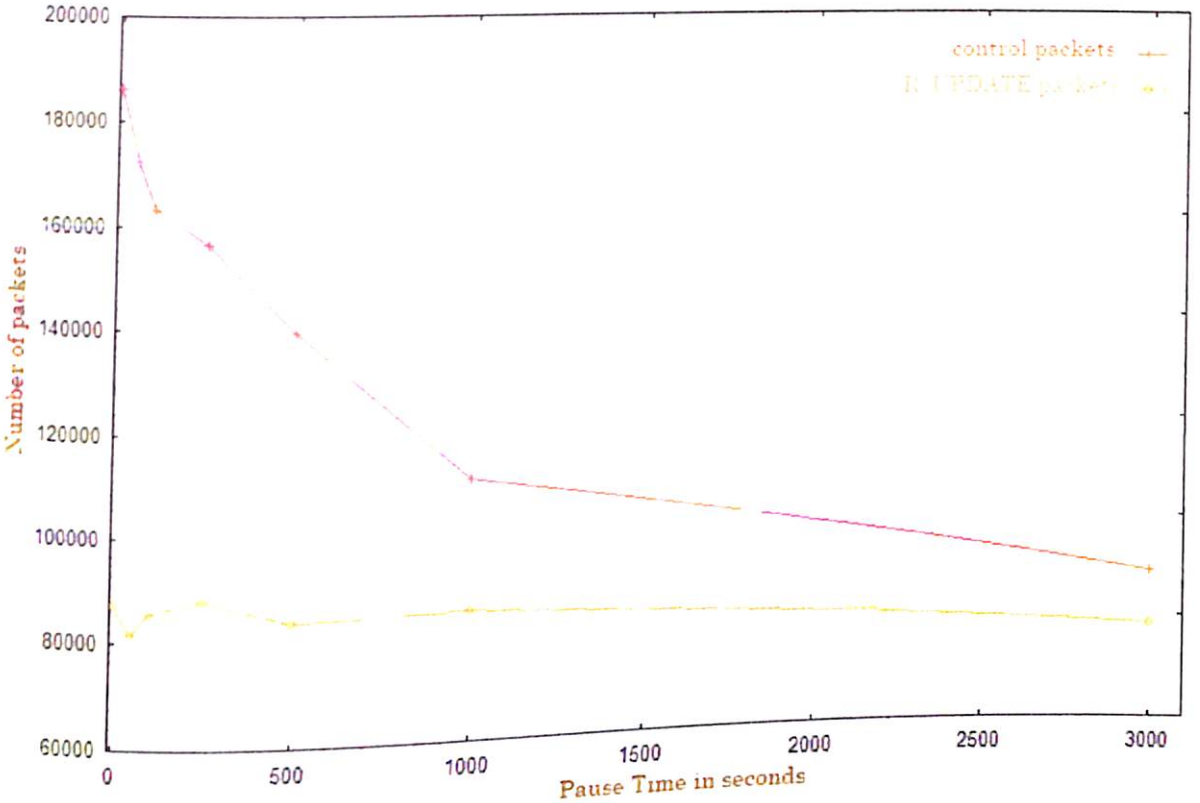


Fig 5.128 Number of R_UPDATE and Control Packets Vs Pause Time at a maximum node speed of 10m/s

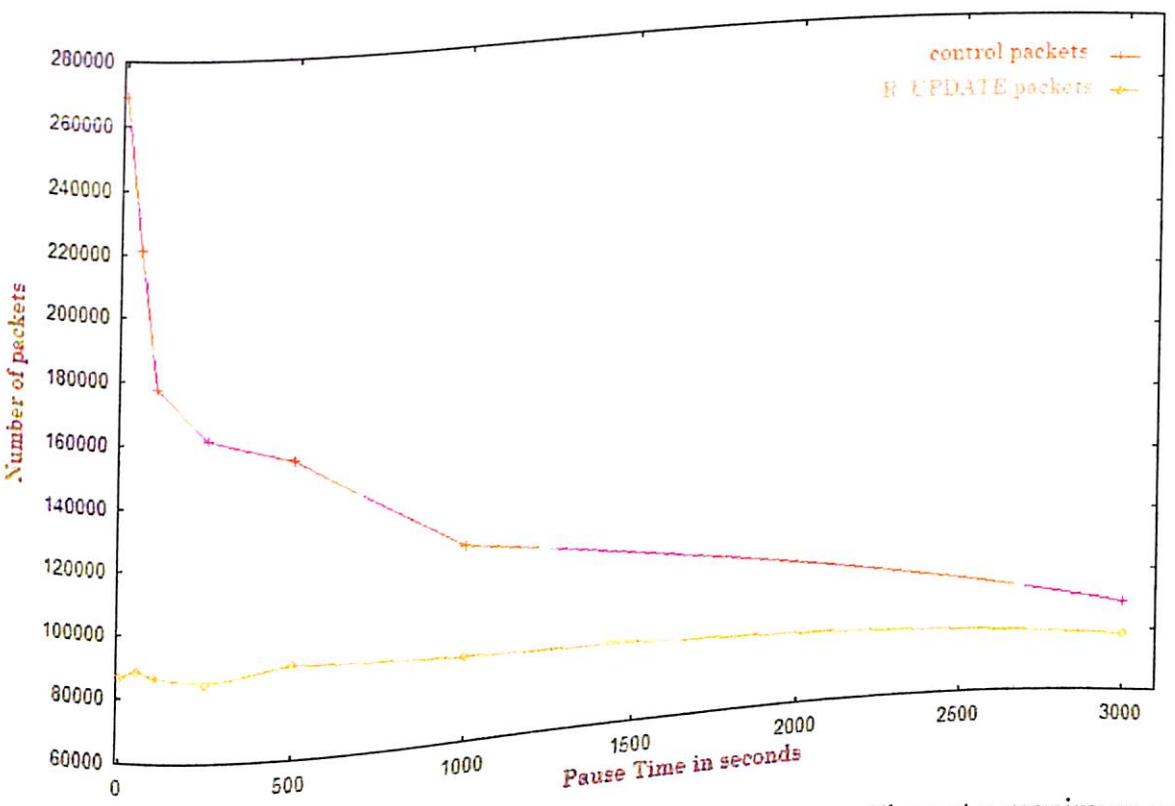


Fig 5.129 Number of R_UPDATE and Control Packets Vs Pause Time at a maximum node speed of 15m/s

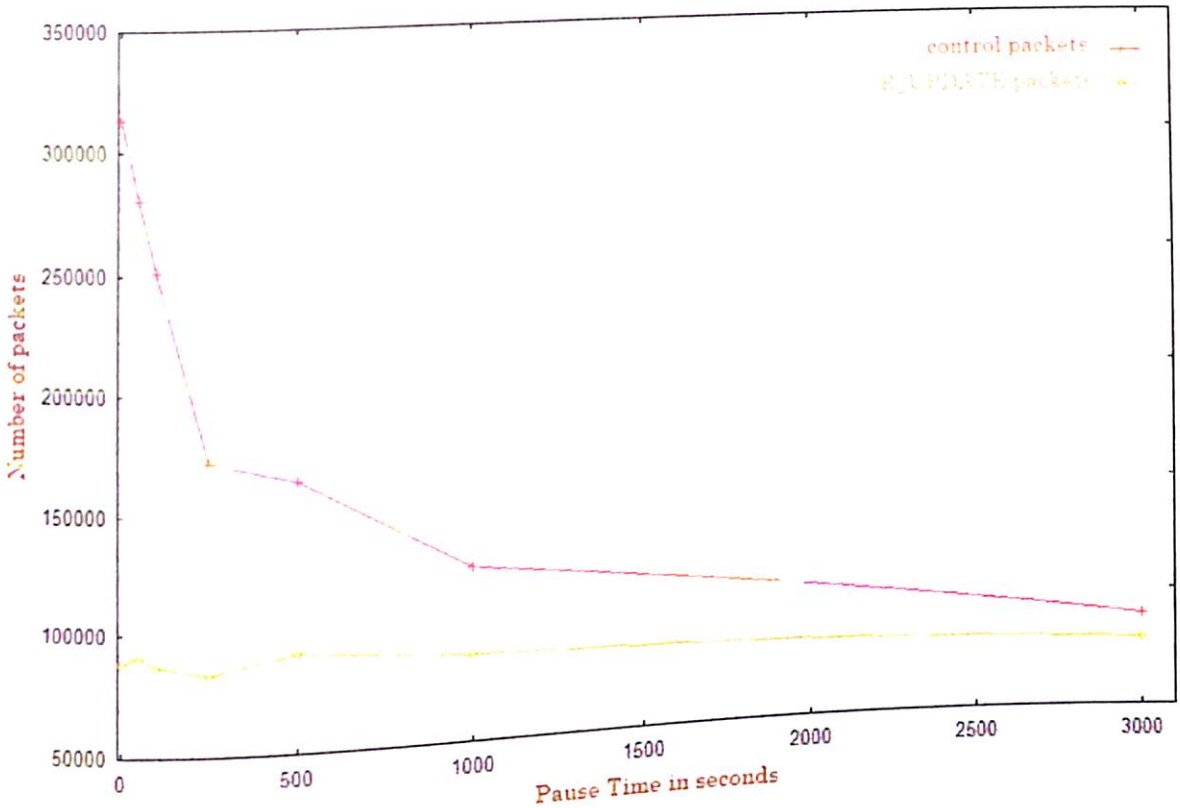


Fig 5.130 Number of R_UPDATE and Control Packets Vs Pause Time at a maximum node speed of 20m/s

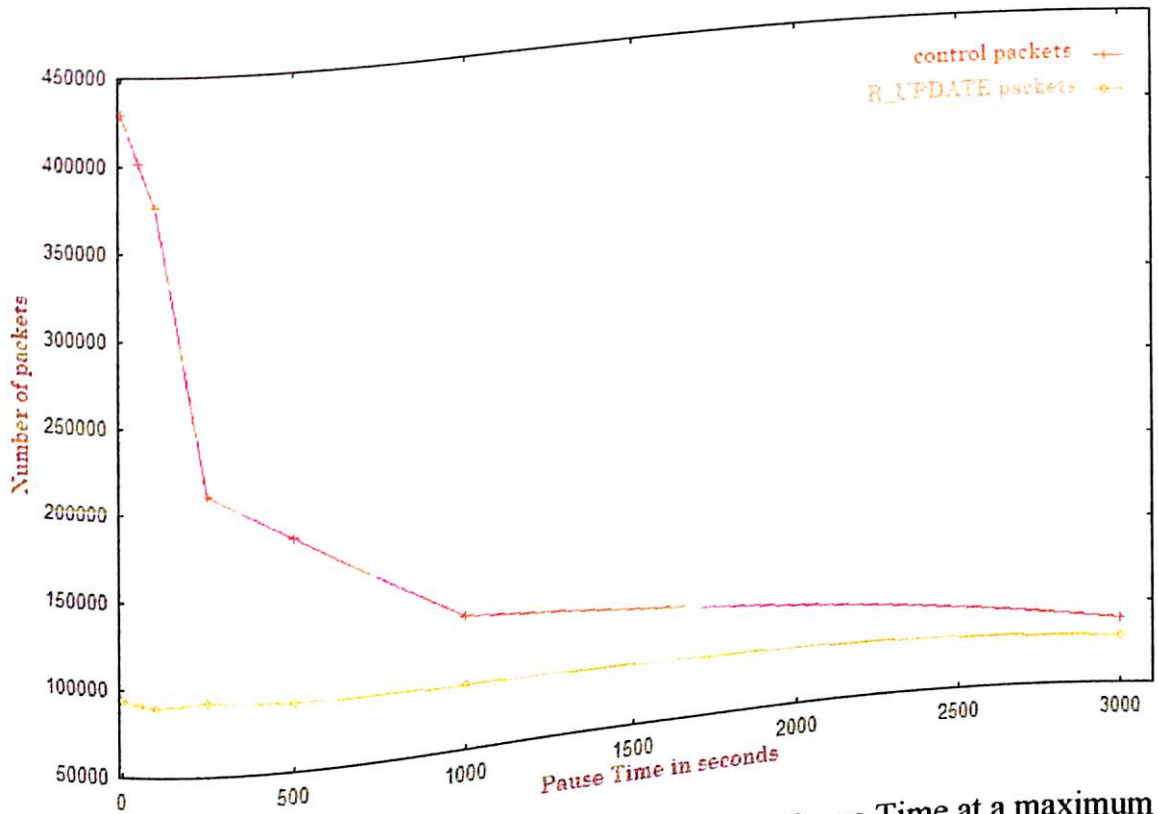


Fig 5.131 Number of R_UPDATE and Control Packets Vs Pause Time at a maximum node speed of 30m/s

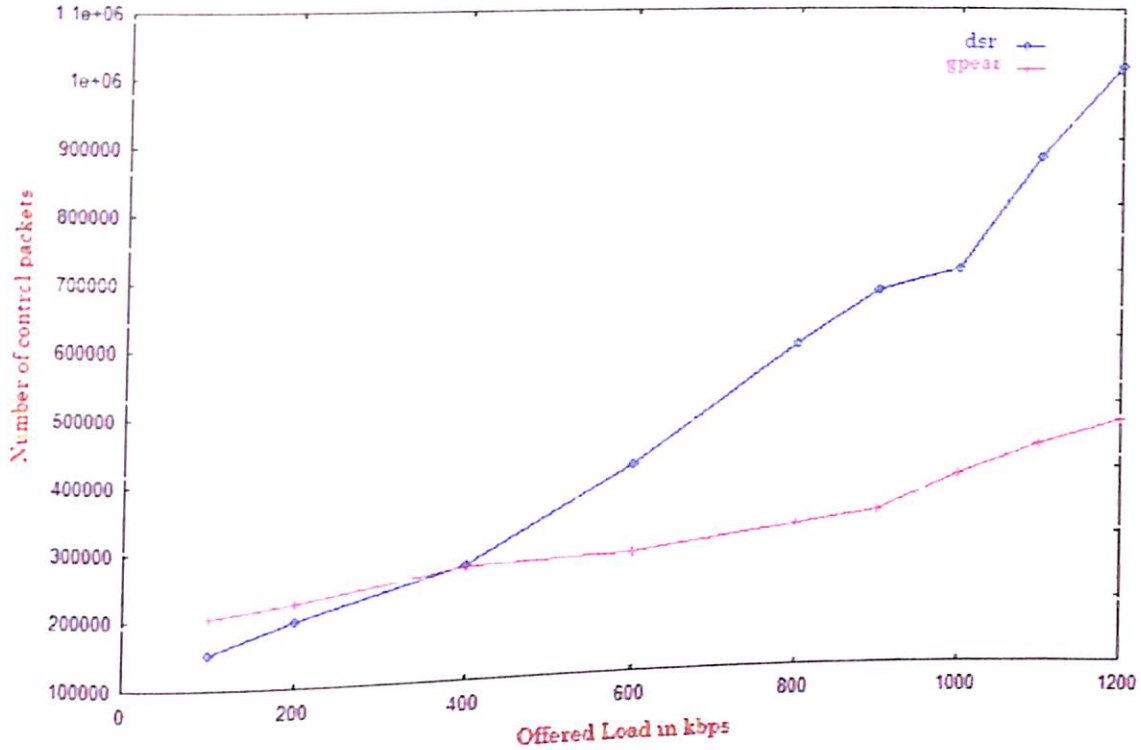


Fig 5.132 Number of control packets Vs Offered Load with 25 connections at a maximum node speed of 15m/s

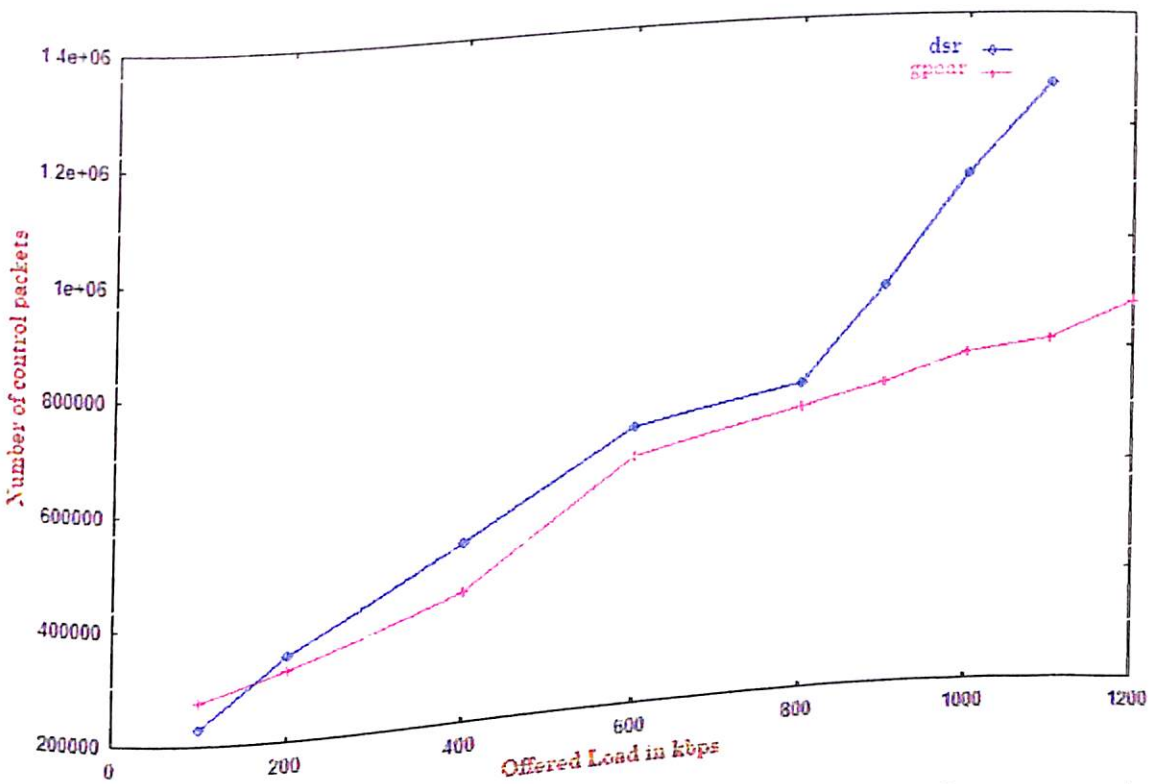


Fig 5.133 Number of control packets Vs Offered Load with 25 connections at a maximum node speed of 30m/s

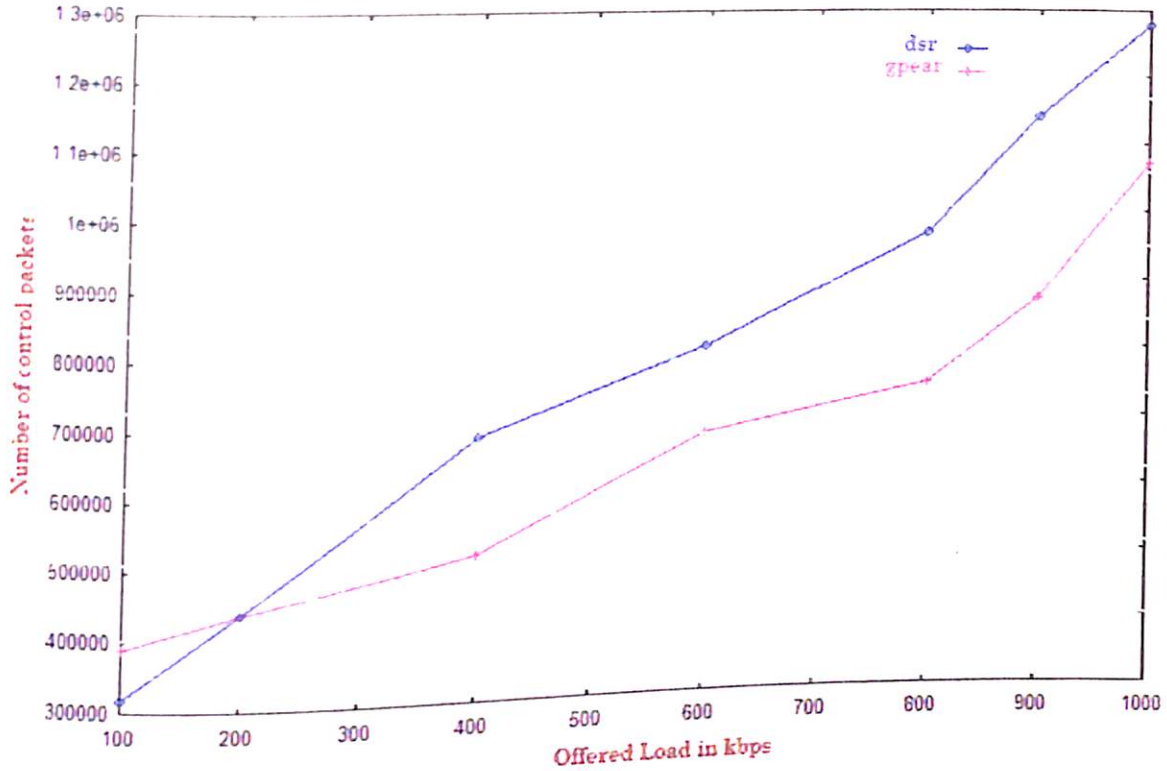


Fig 5.134 Number of control packets Vs Offered Load with 50 connections at a maximum node speed of 15m/s

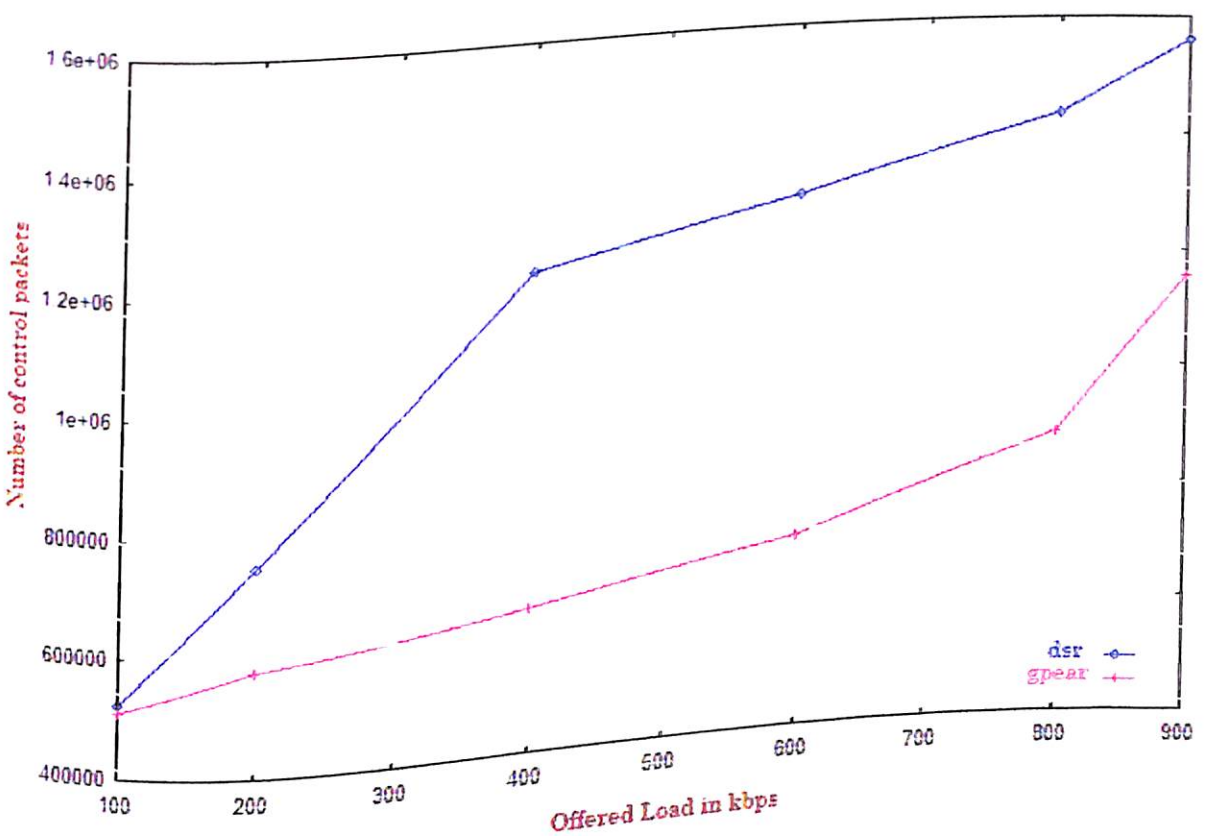


Fig 5.135 Number of control packets Vs Offered Load with 50 connections at a maximum node speed of 30m/s

Analysis of GPEAR Path Optimality (fig. 5.136 - fig. 5.141)

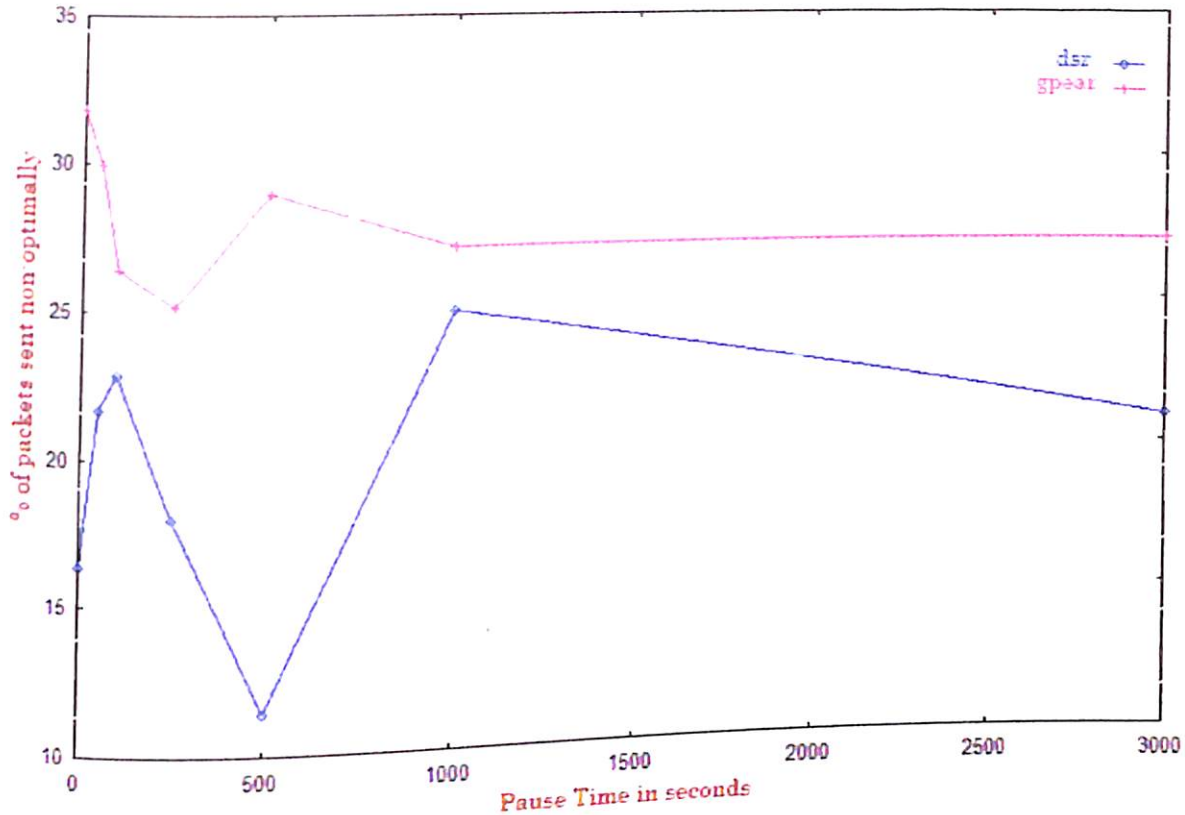


Fig 5.136 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 1m/s

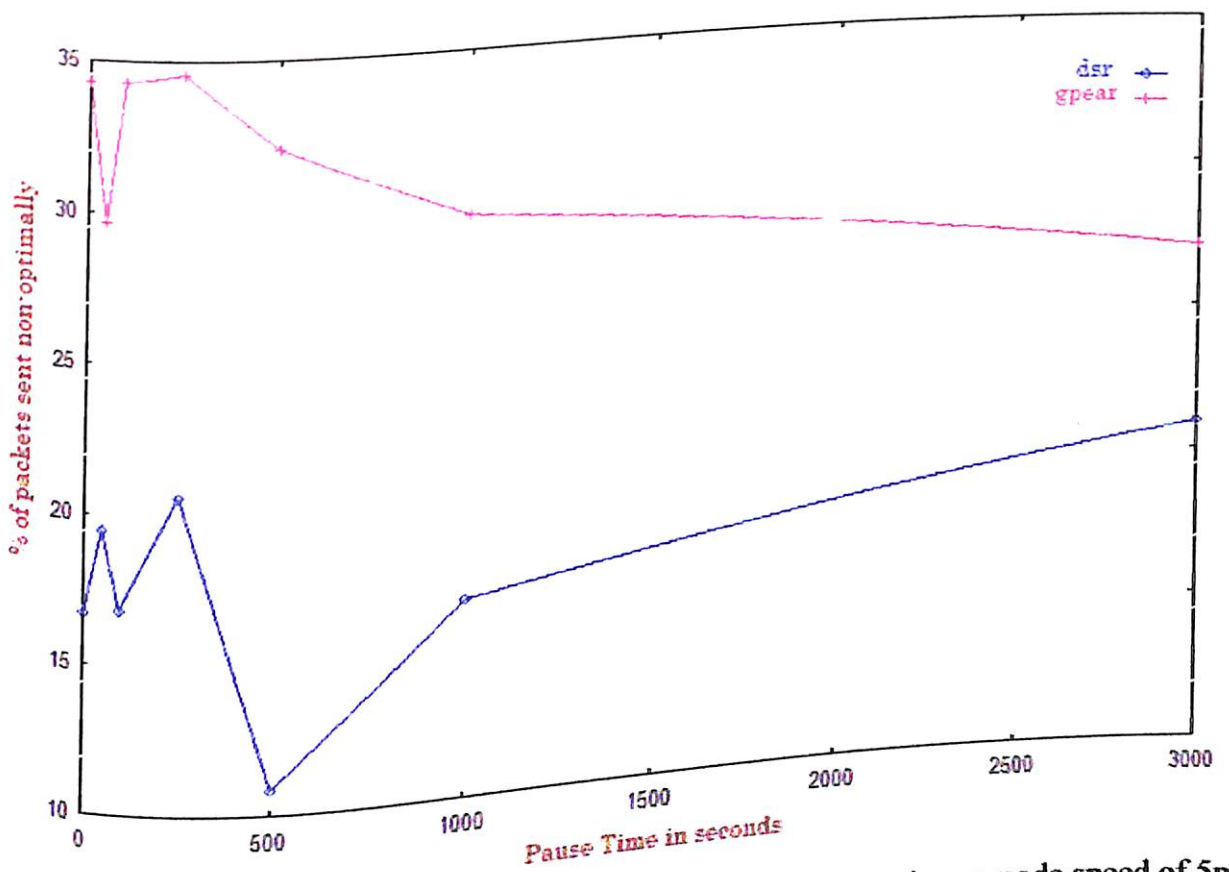


Fig 5.137 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 5m/s

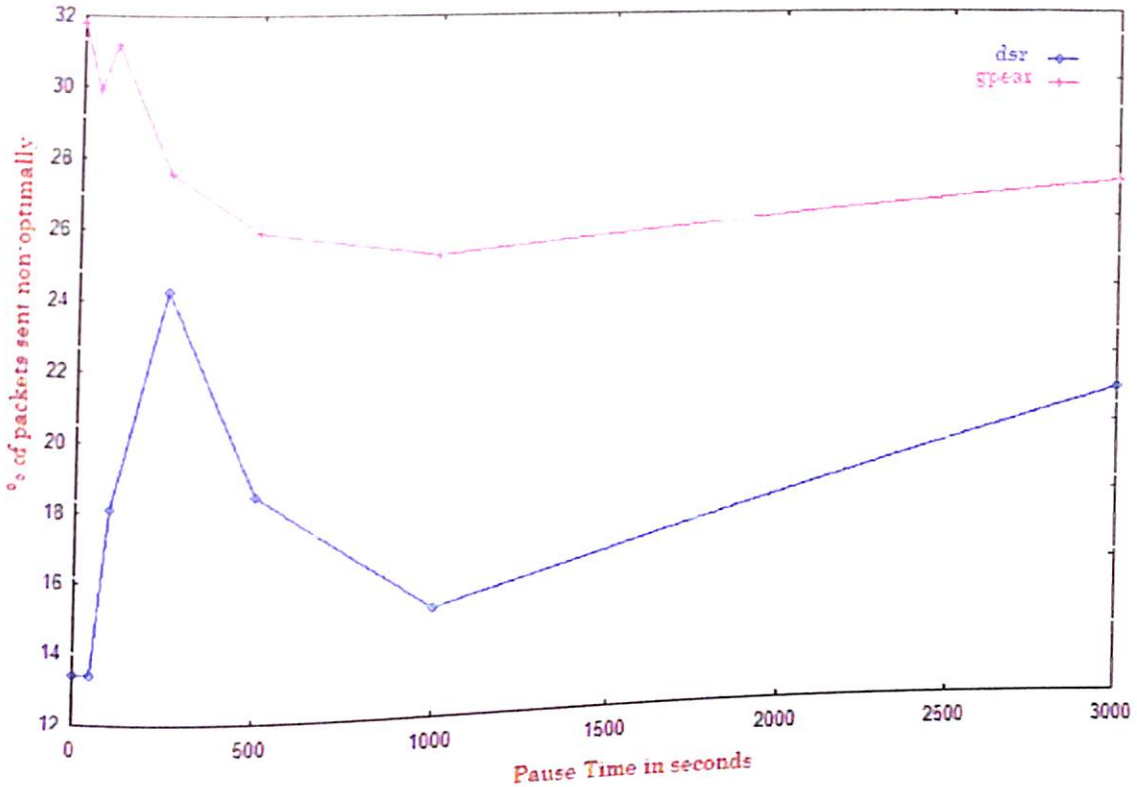


Fig 5.138 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 10m/s

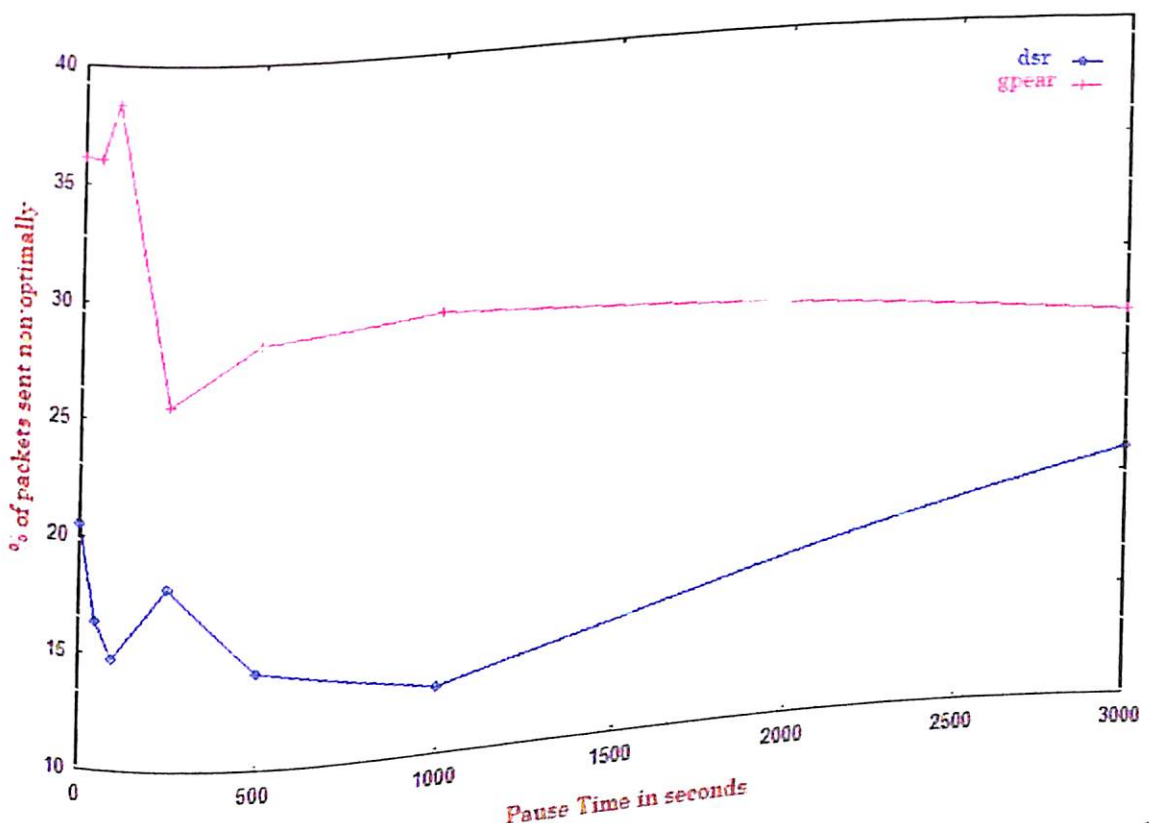


Fig 5.139 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 15m/s

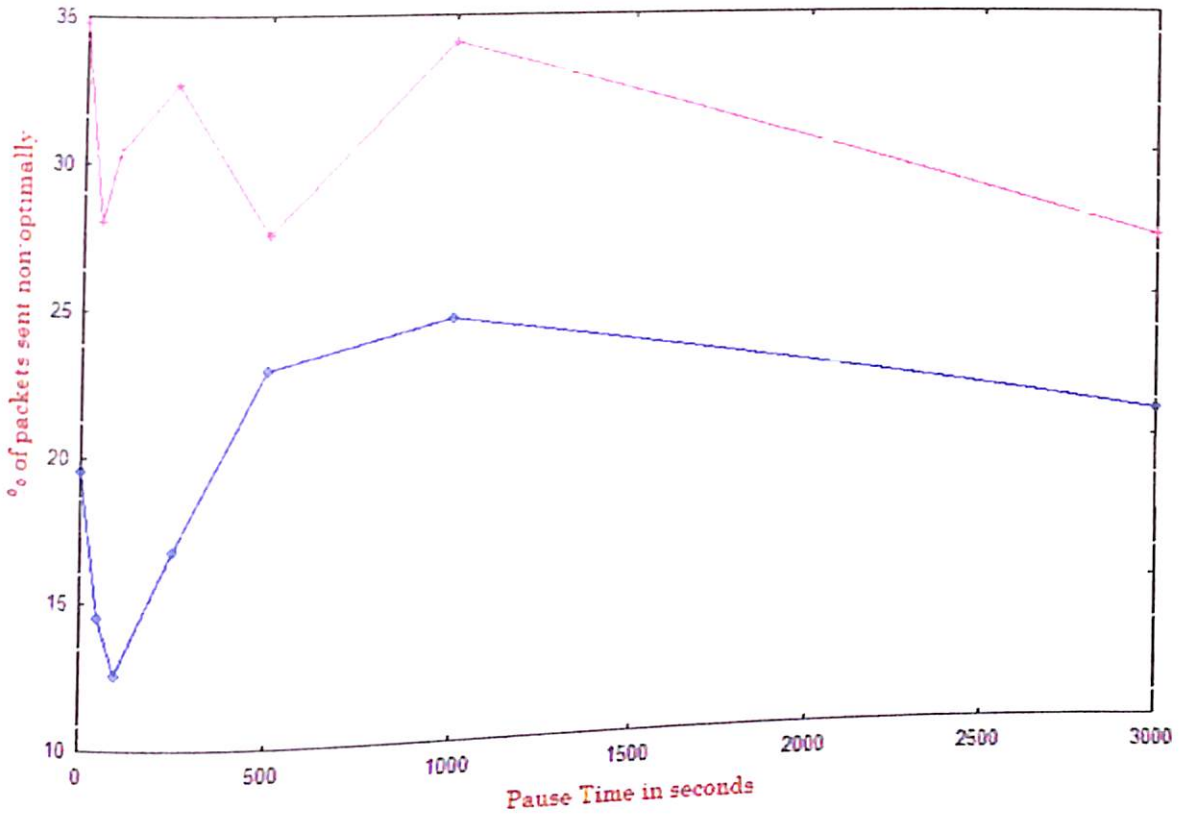


Fig 5.140 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 20m/s

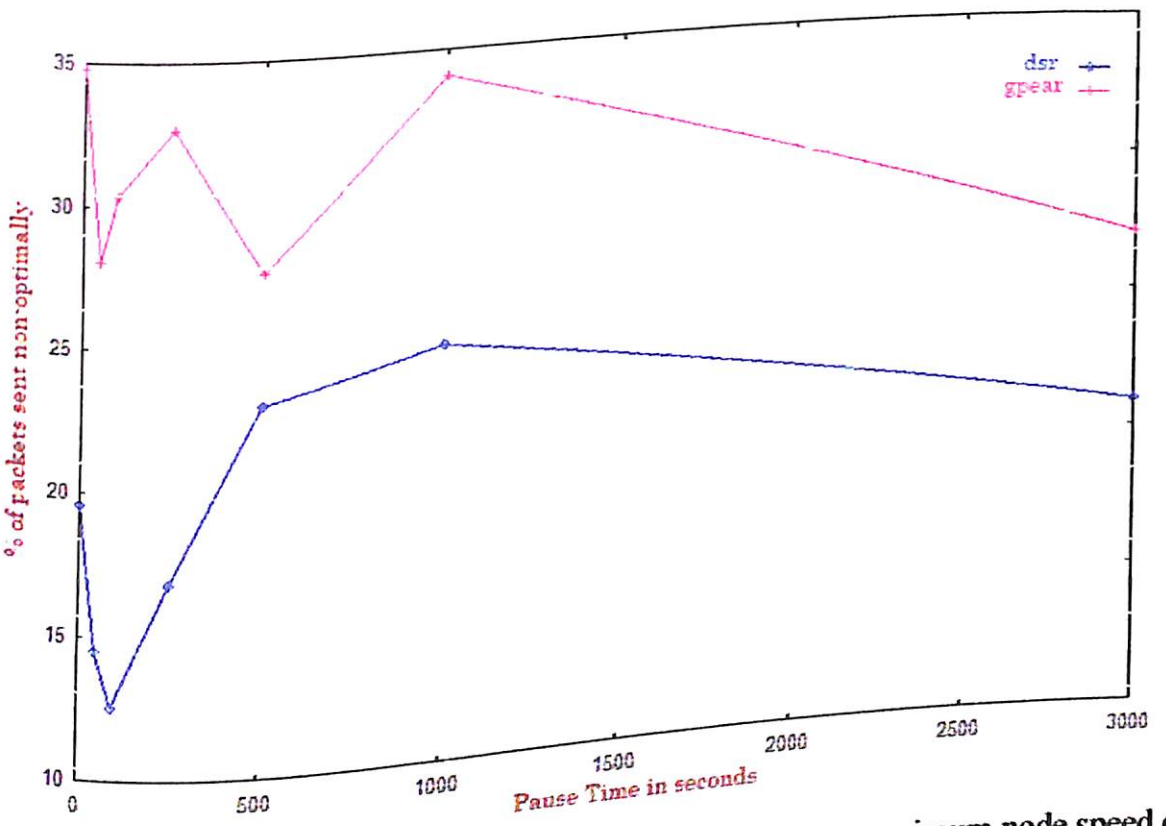


Fig 5.141 % of Packets sent non-optimally Vs Pause Time at a maximum node speed of 30m/s

Analysis of GPEAR Energy Consumption (fig. 5.142 - fig. 5.146)

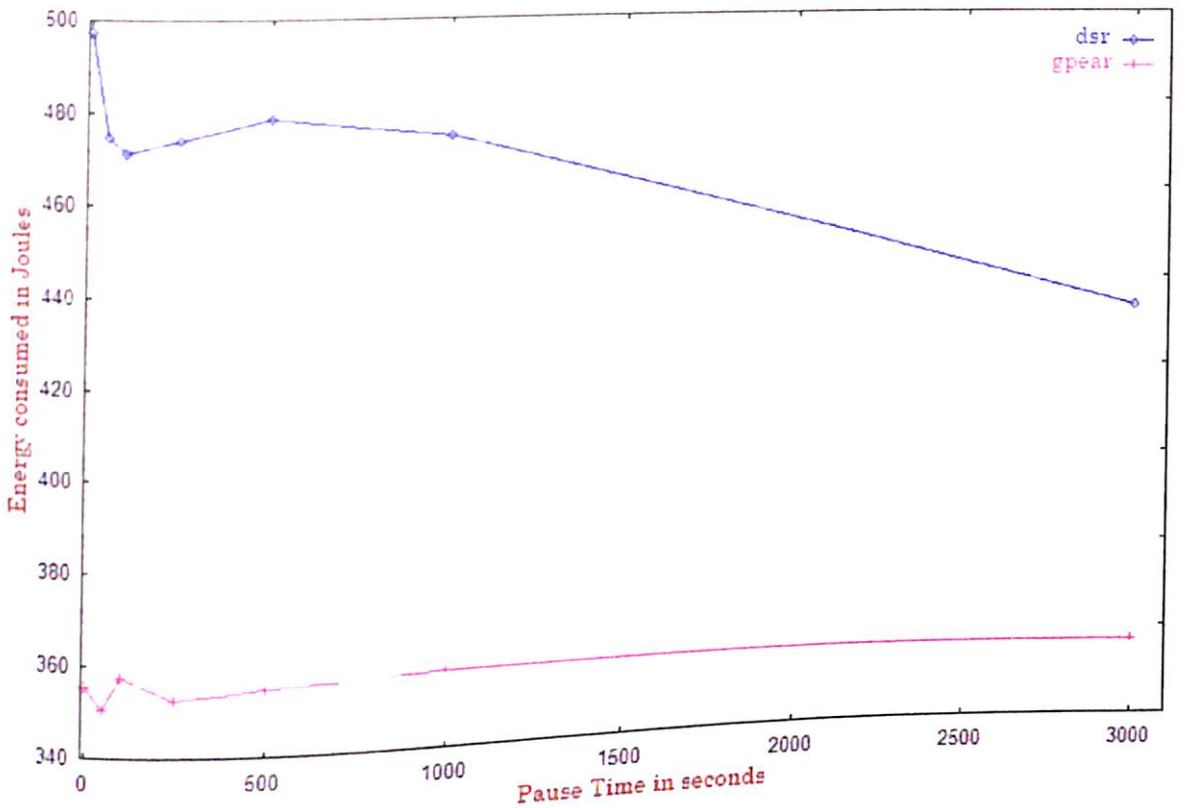


Fig 5.142 Energy consumed Vs Pause Time at a maximum node speed of 30m/s with 25 connections with a data rate of 4 packets/sec

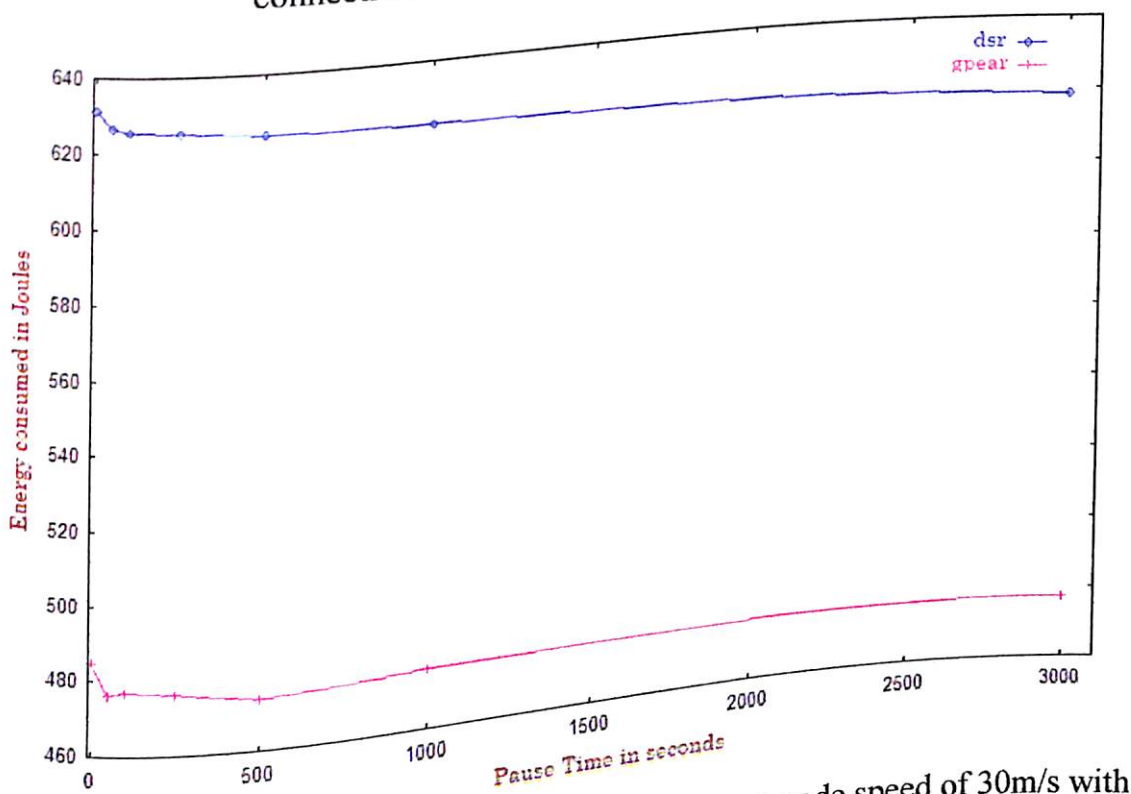


Fig 5.143 Energy consumed Vs Pause Time at a maximum node speed of 30m/s with 25 connections at a data rate of 6 packets/sec

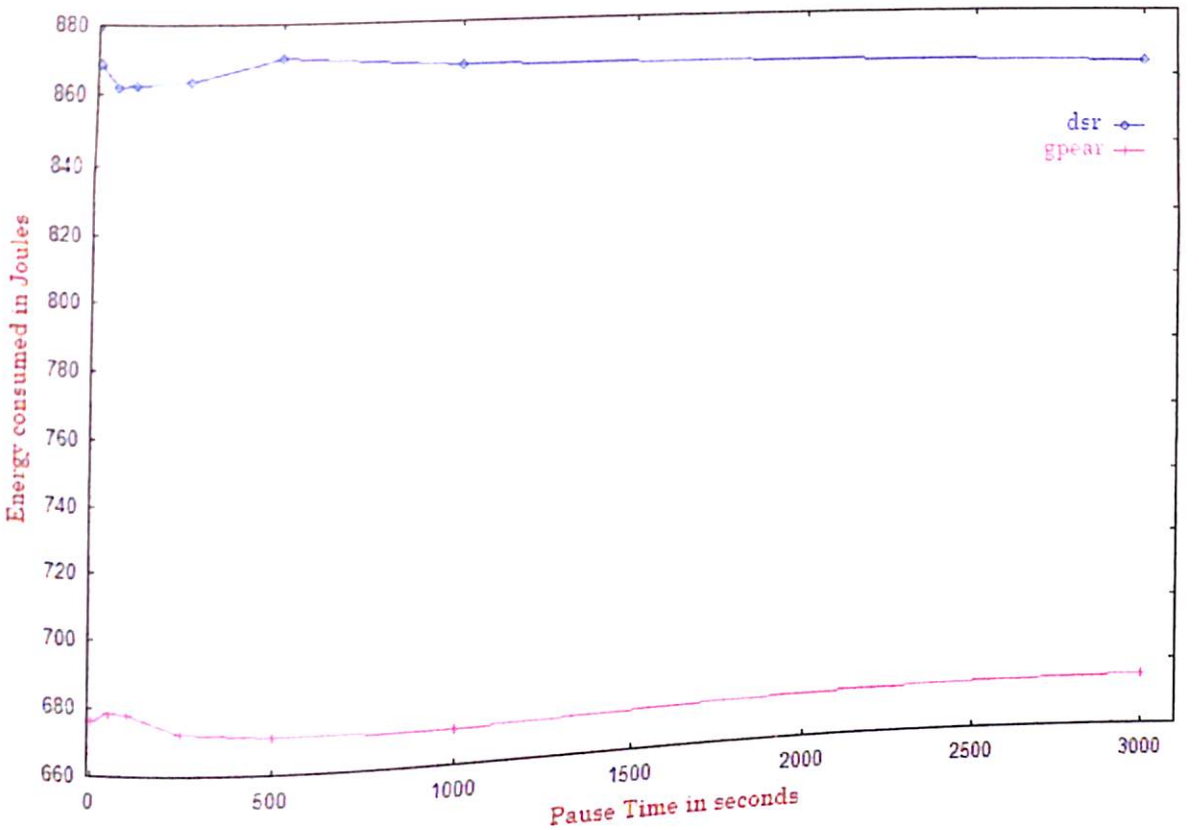


Fig 5.144 Energy consumed Vs Pause Time at a maximum node speed of 30m/s with 25 connections at a data rate of 8 packets/sec

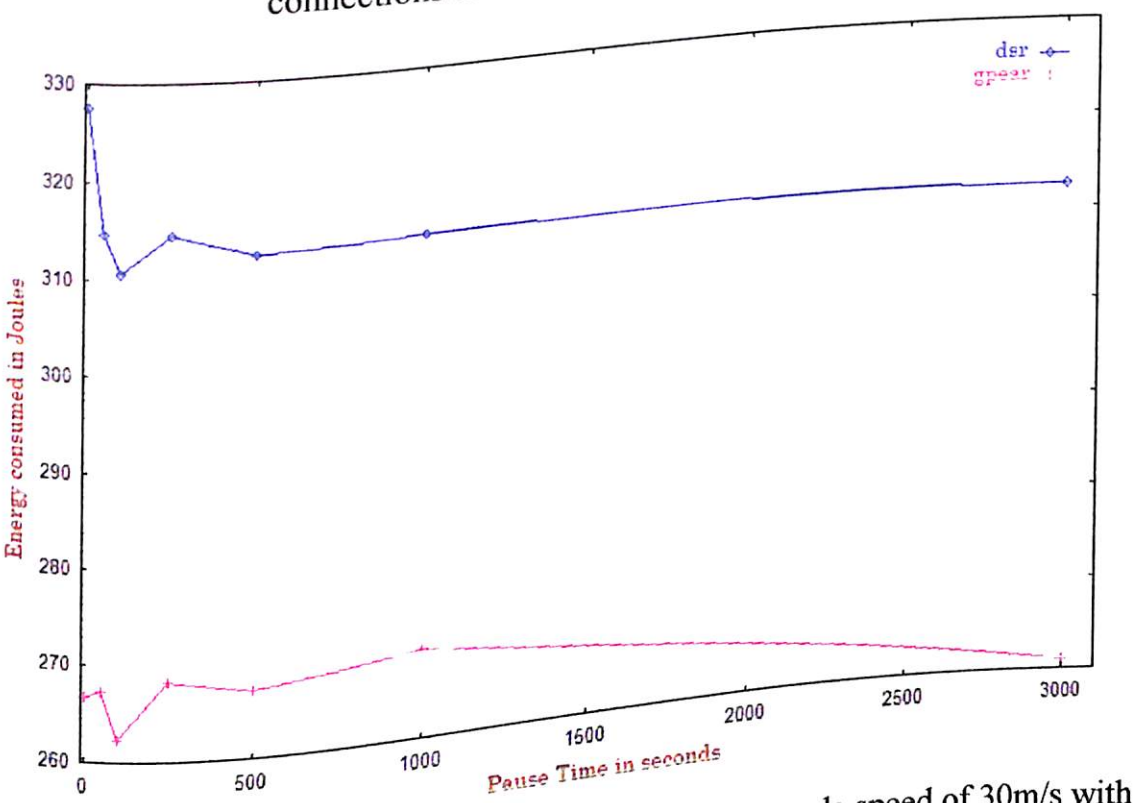


Fig 5.145 Energy consumed Vs Pause Time at a maximum node speed of 30m/s with 15 connections at a data rate of 4 packets/sec

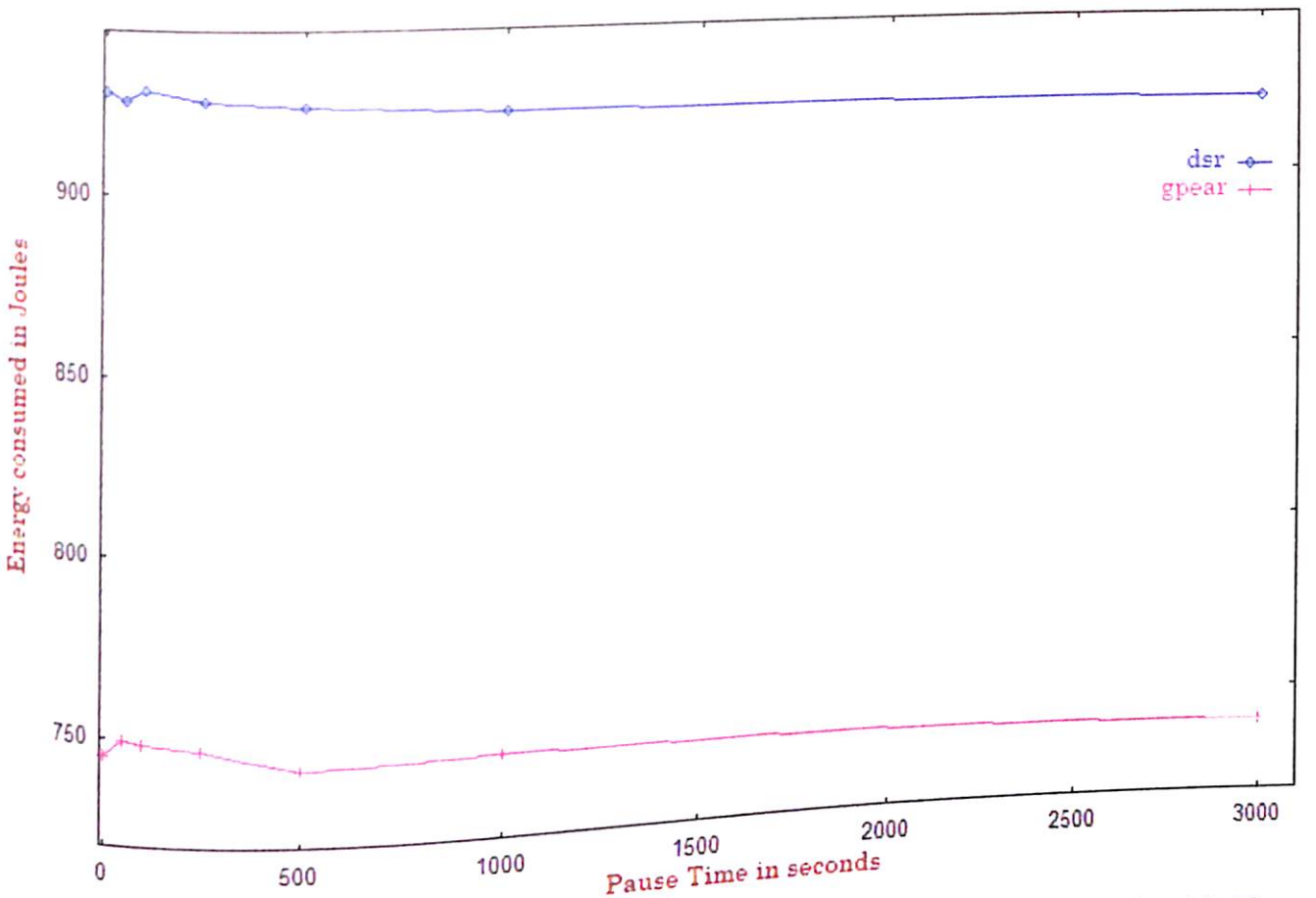


Fig 5.146 Energy consumed Vs Pause Time at a maximum node speed of 30m/s with 50 connections at a data rate of 4 packets/sec

Analysis of Optimum Node Density in a Mobile Adhoc Network using GPEAR as the Routing Protocol
(fig. 5.147 - fig. 5.160)

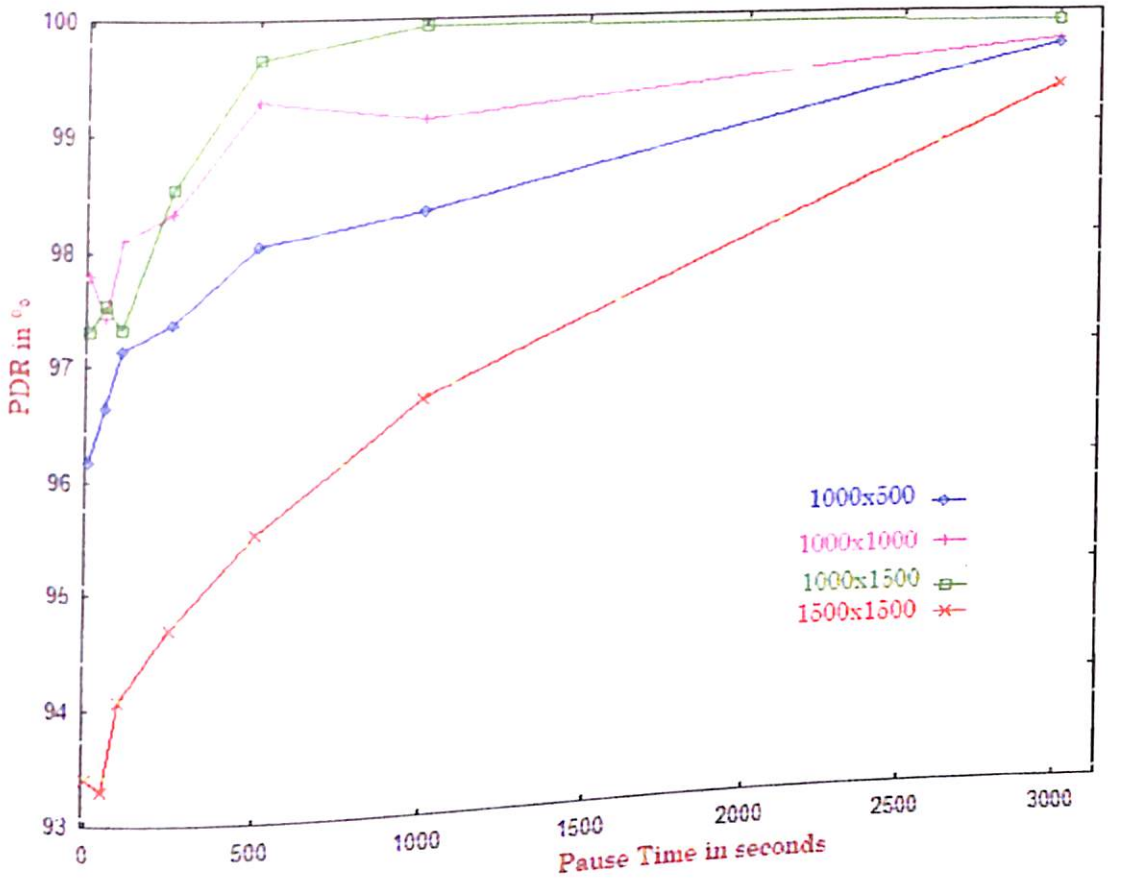


Fig 5.147 PDR Vs Pause Time at a maximum node speed of 10m/s for varying network area

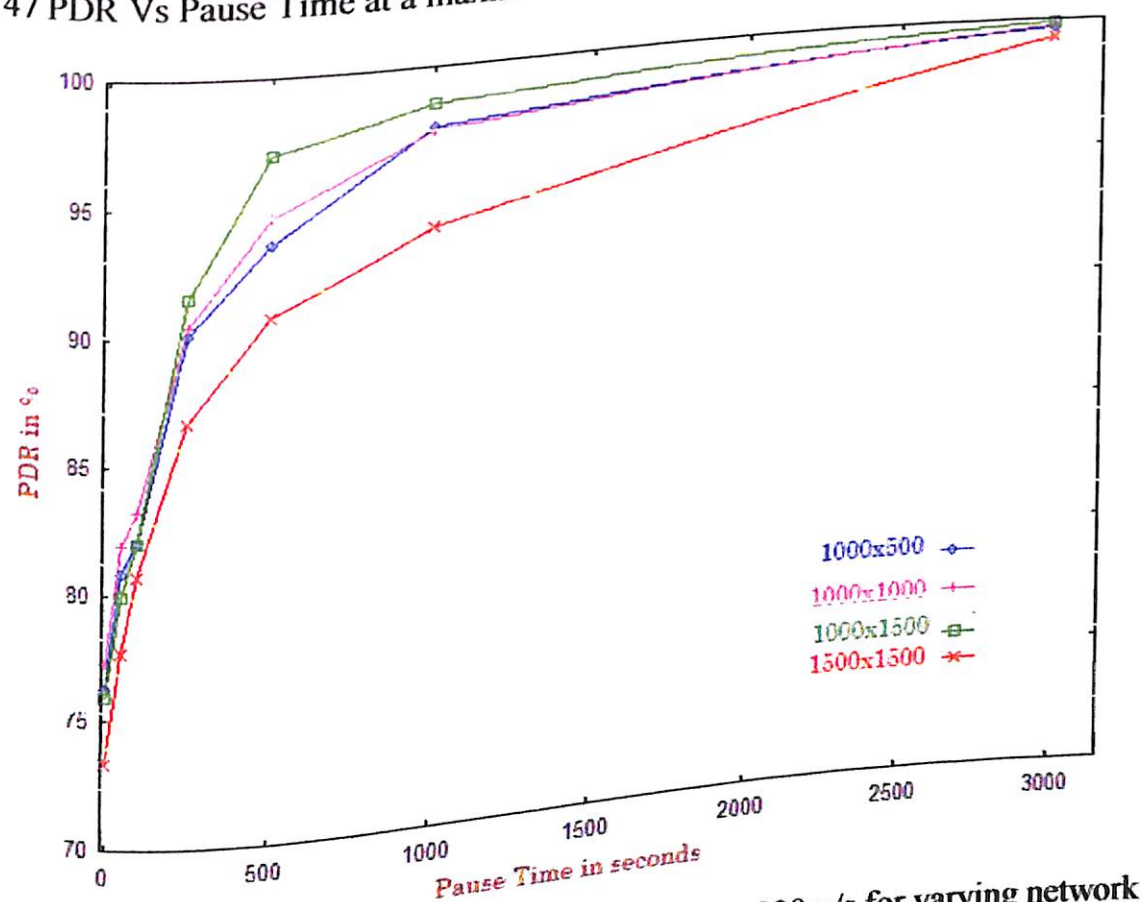


Fig 5.148 PDR Vs Pause Time at a maximum node speed of 20m/s for varying network area

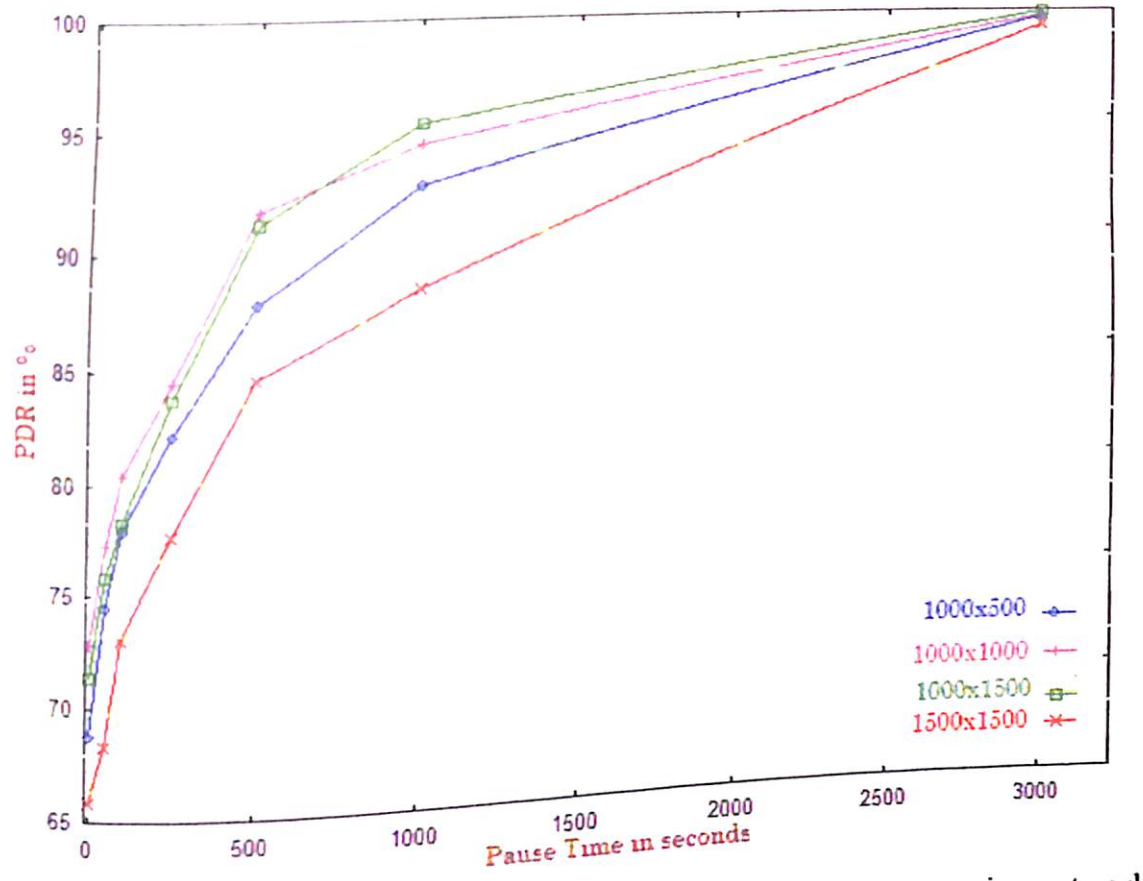


Fig 5.149 PDR Vs Pause Time at a maximum node speed of 30m/s for varying network area

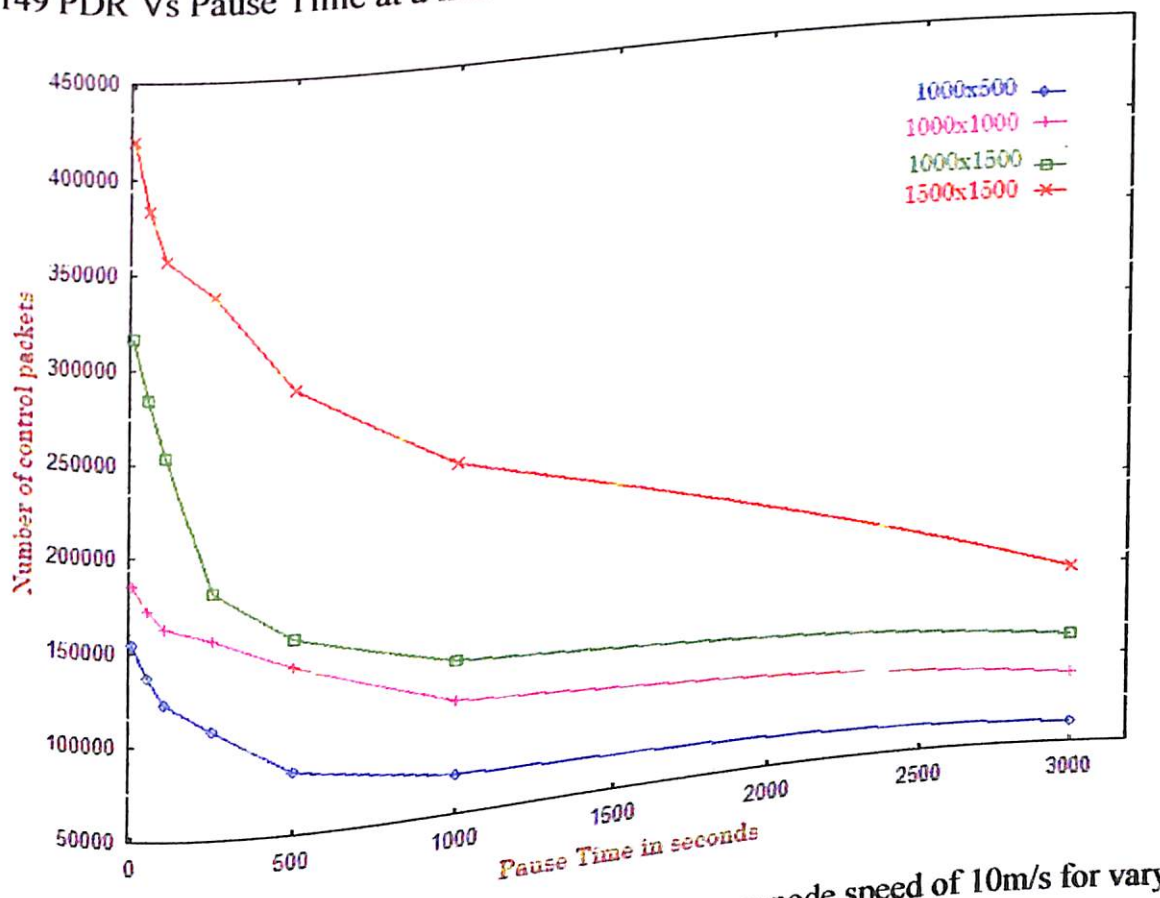


Fig 5.150 Control Overhead Vs Pause Time at a maximum node speed of 10m/s for varying network area

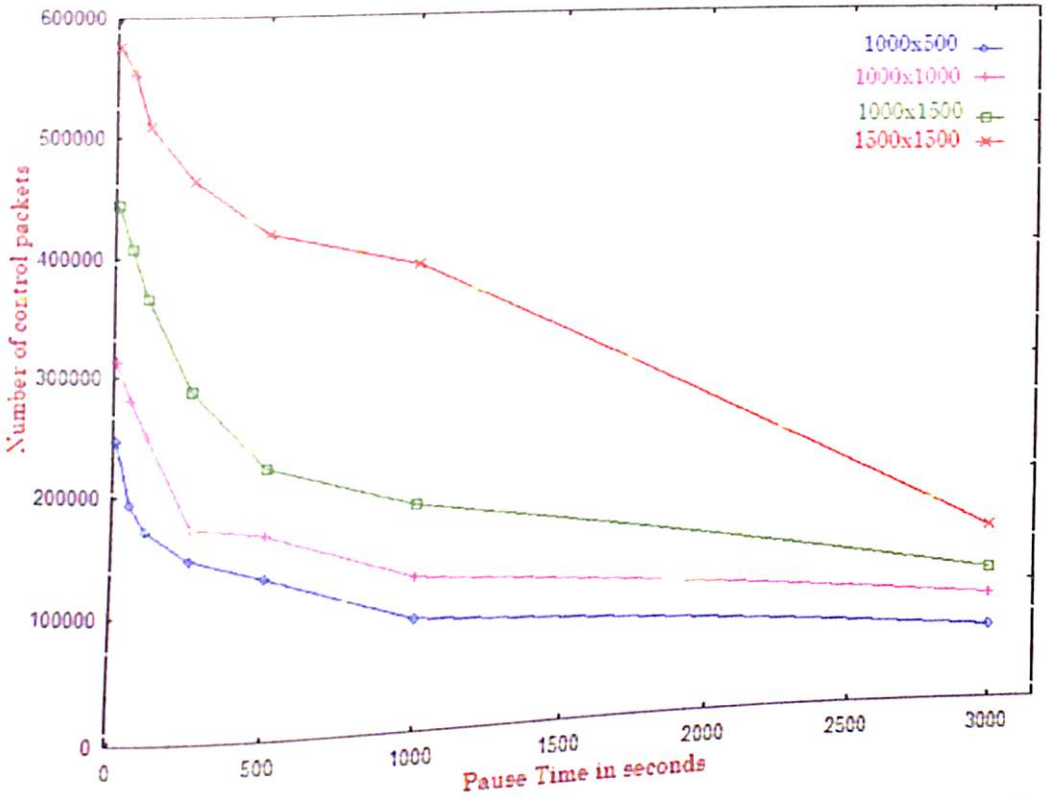


Fig 5.151 Control Overhead Vs Pause Time at a maximum node speed of 20m/s for varying network area

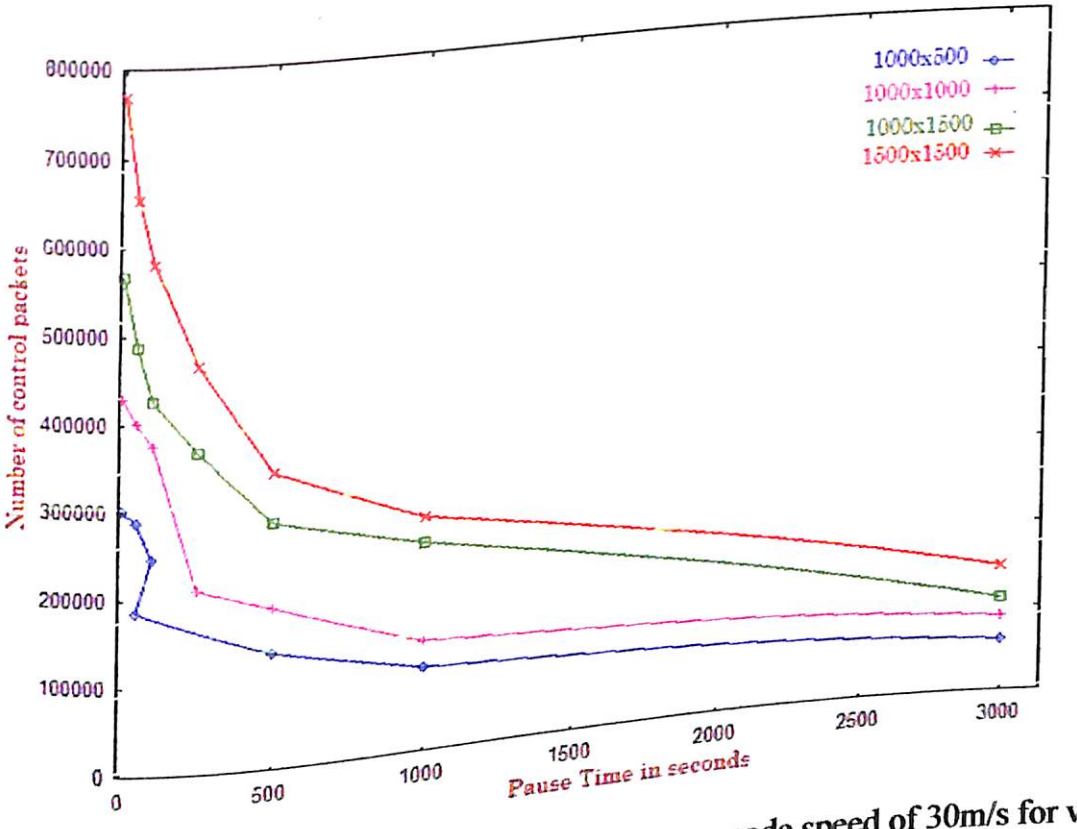
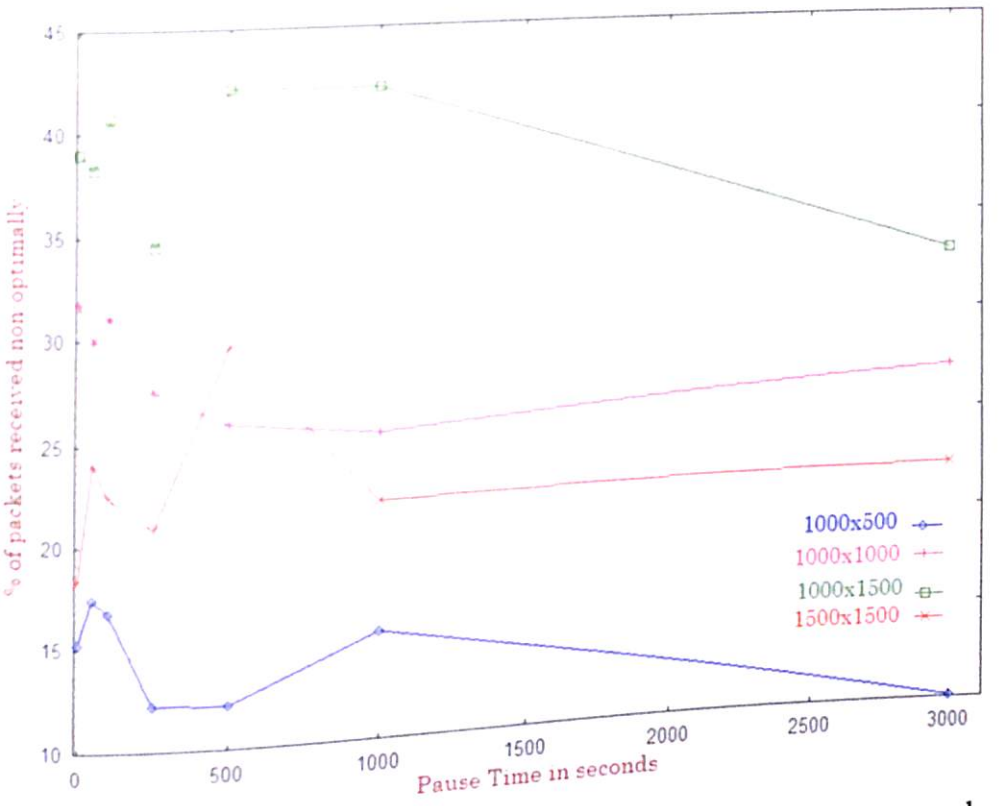
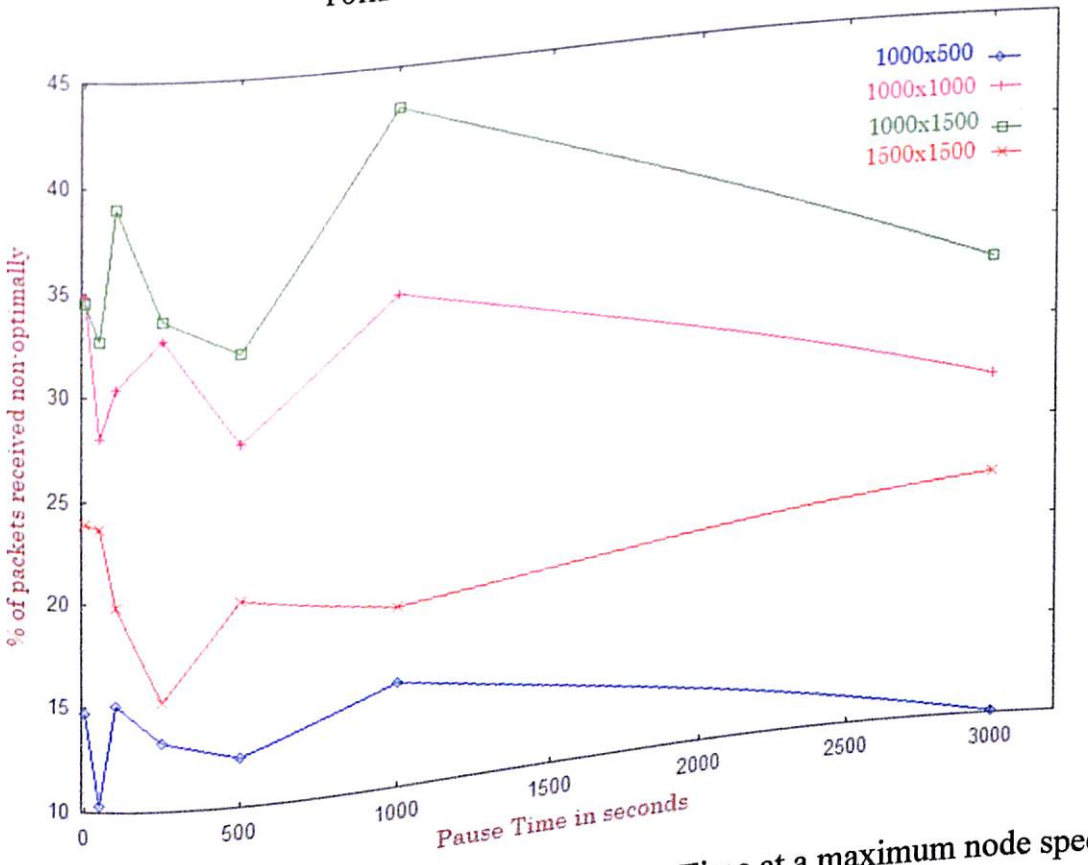


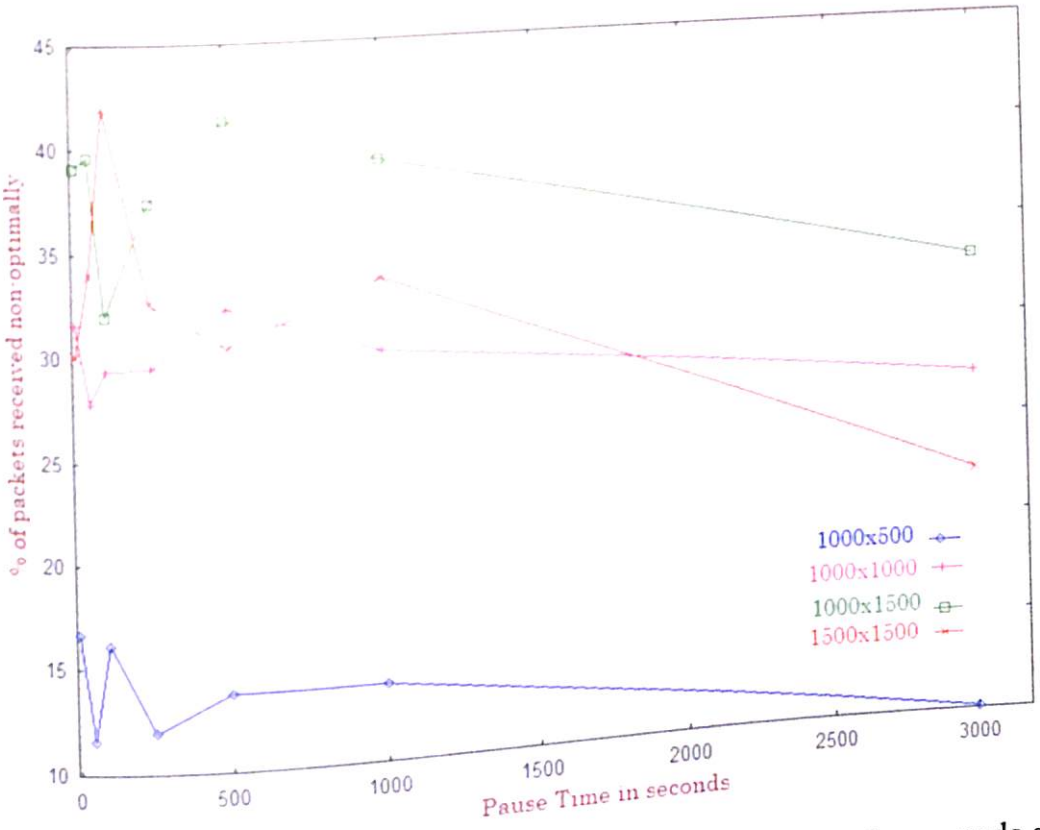
Fig 5.152 Control Overhead Vs Pause Time at a maximum node speed of 30m/s for varying network area



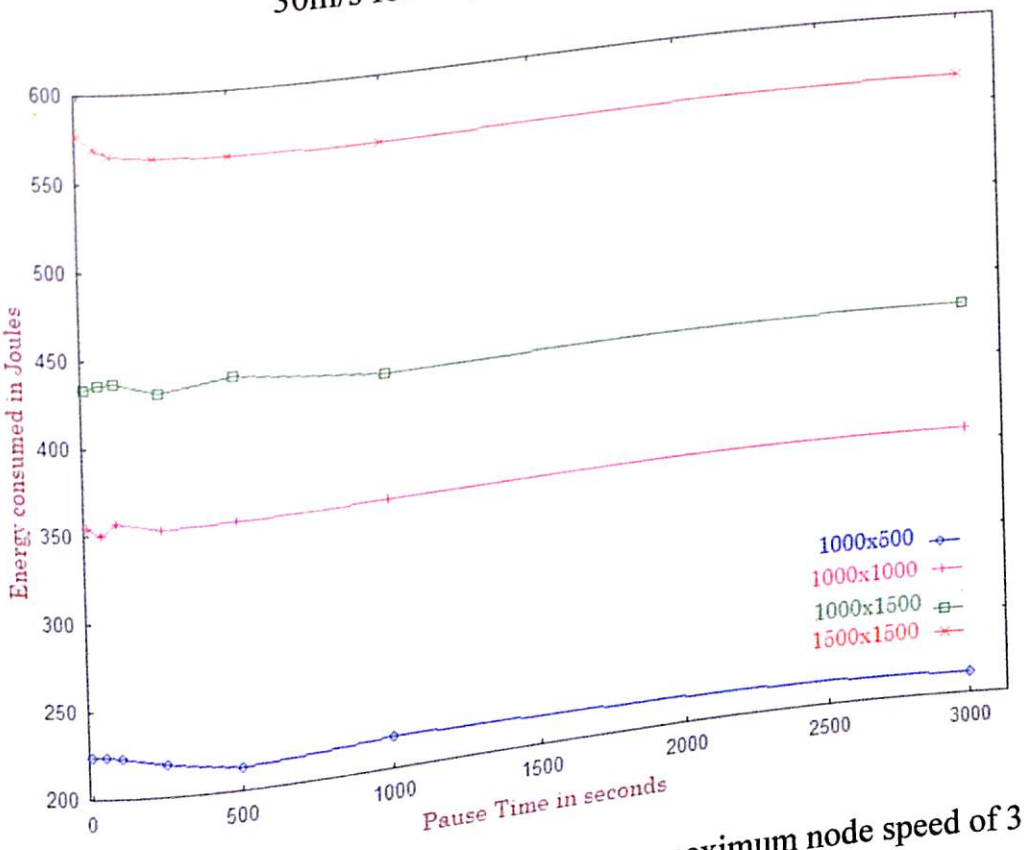
5.153 % of Packets received non-optimally Vs Pause Time at a maximum node speed of 10m/s for varying network area



5.154 % of Packets received non-optimally Vs Pause Time at a maximum node speed of 20m/s for varying network area



5.155 % of Packets received non-optimally Vs Pause Time at a maximum node speed of 30m/s for varying network area



5.156 Energy consumed per node Vs Pause Time at a maximum node speed of 30m/s for varying network area

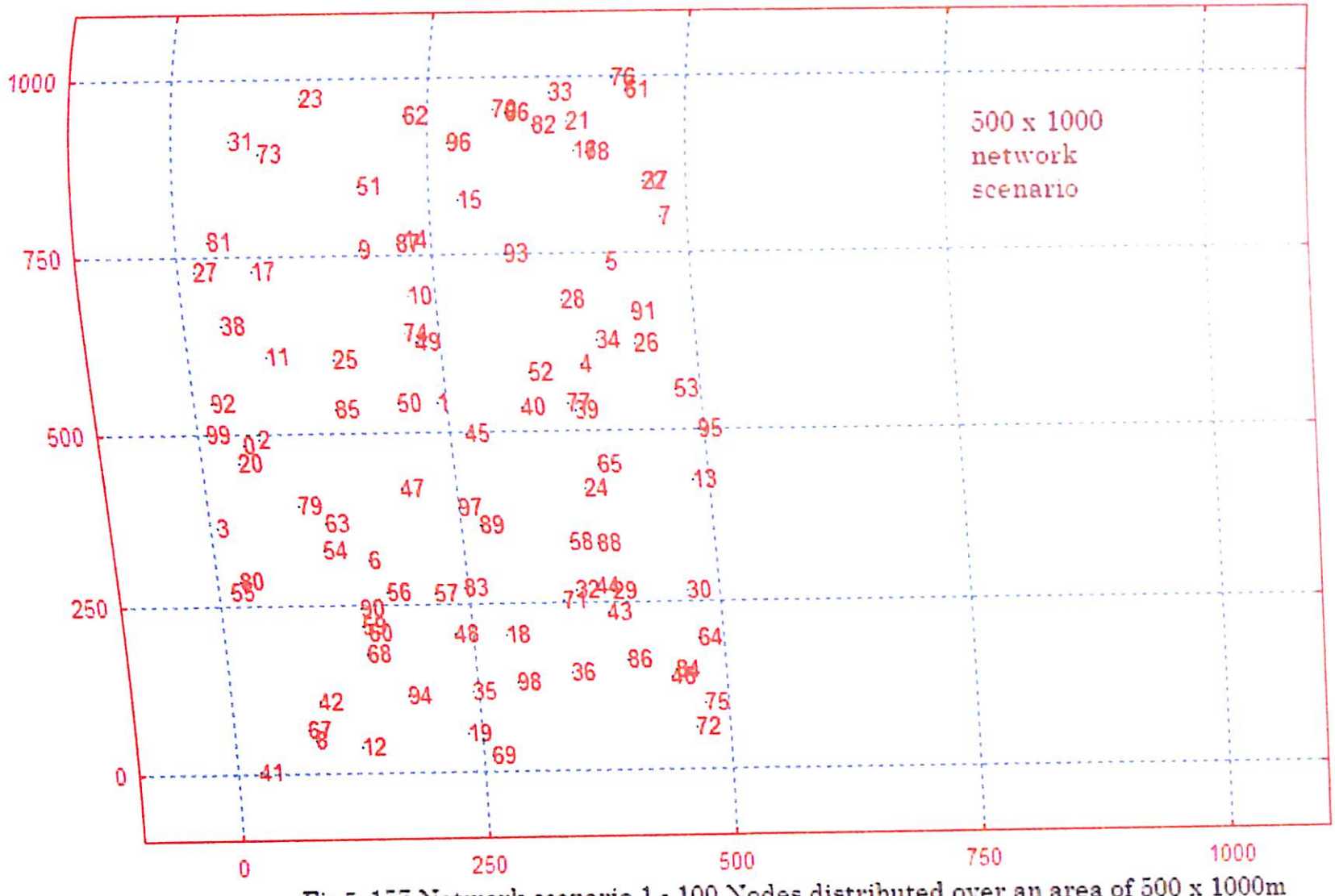


Fig5.157 Network scenario 1 - 100 Nodes distributed over an area of 500 x 1000m

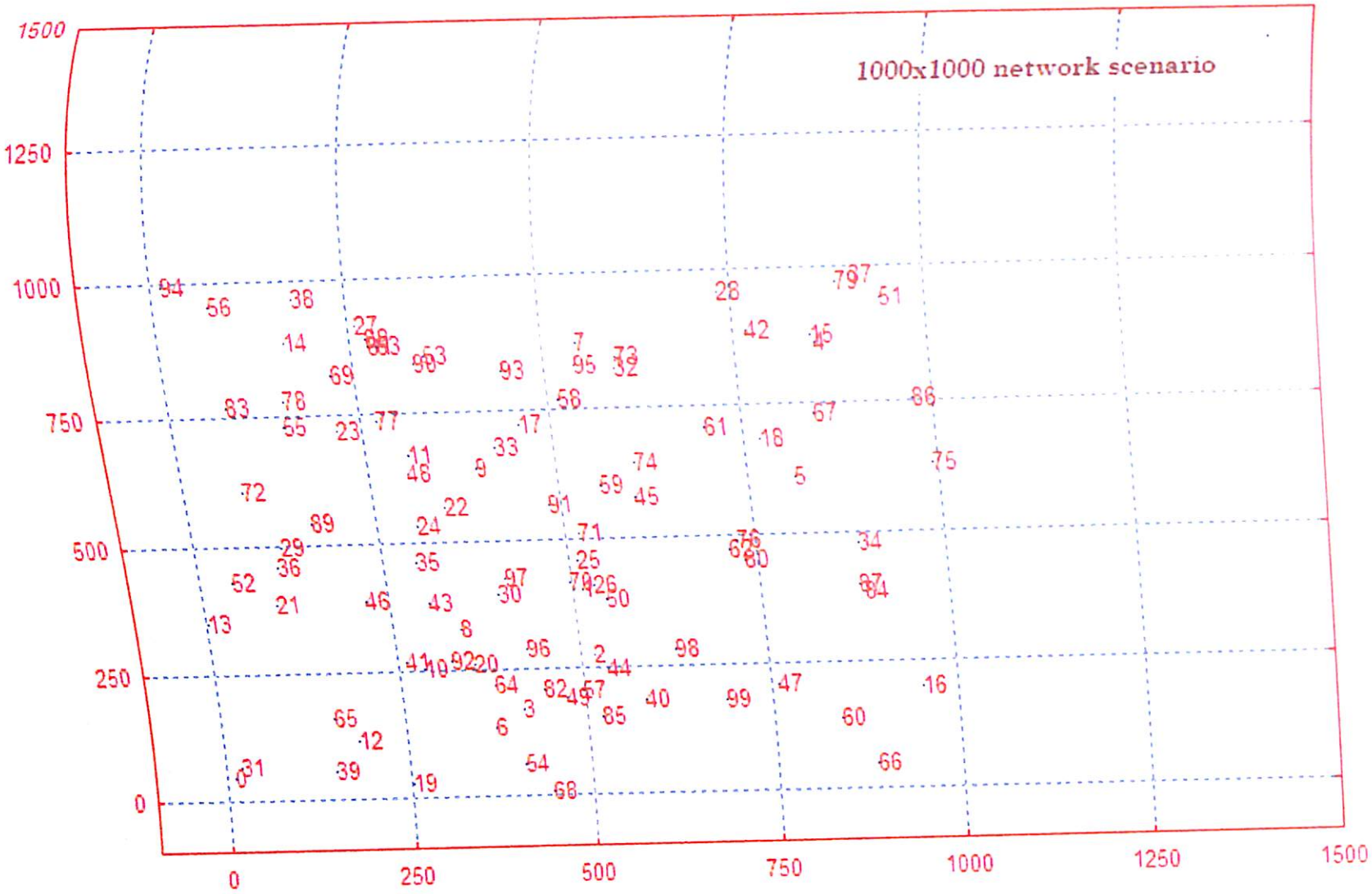


Fig5 .158 Network scenario 2 - 100 Nodes distribured over an area of 1000 x1000m

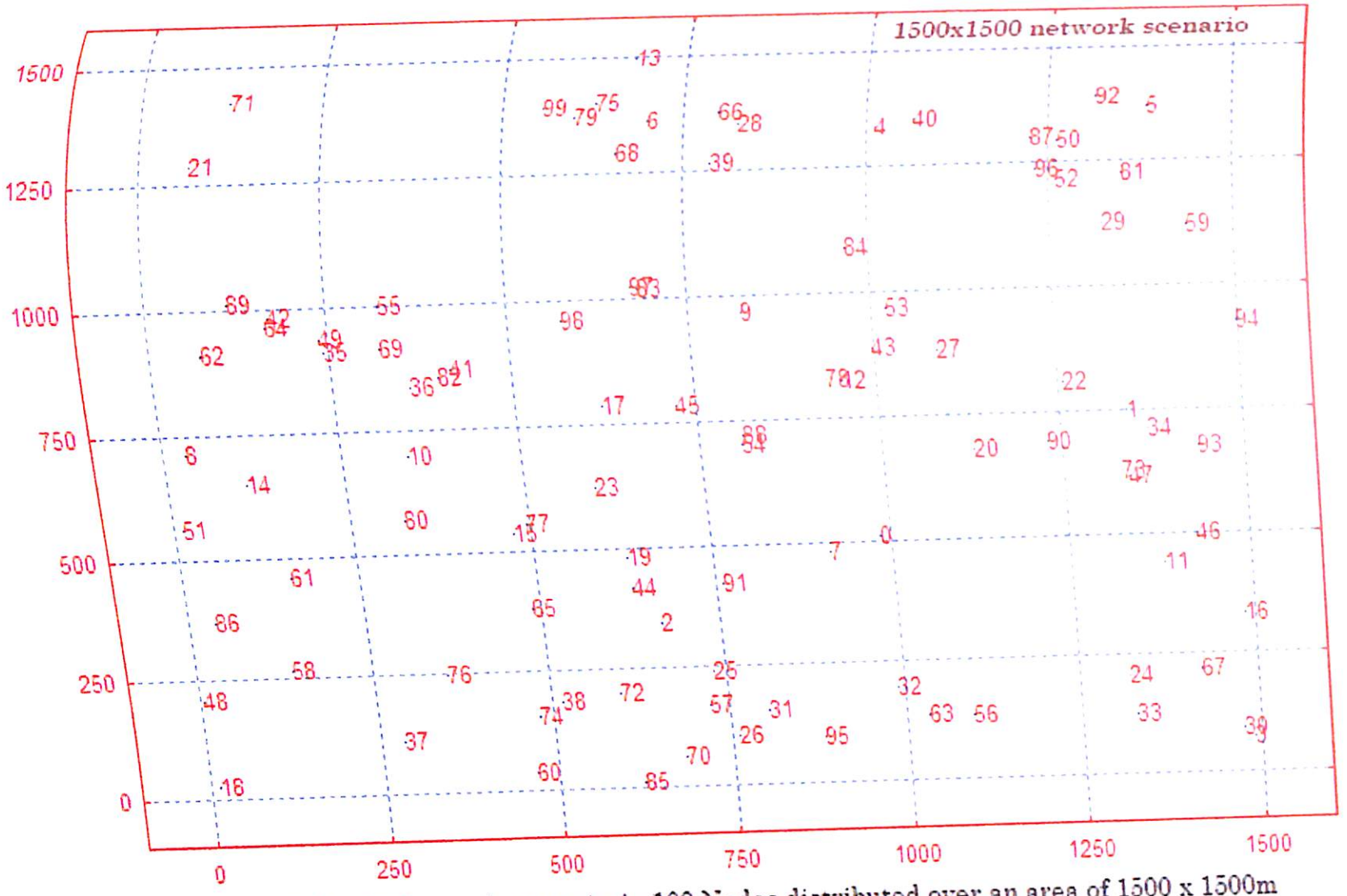
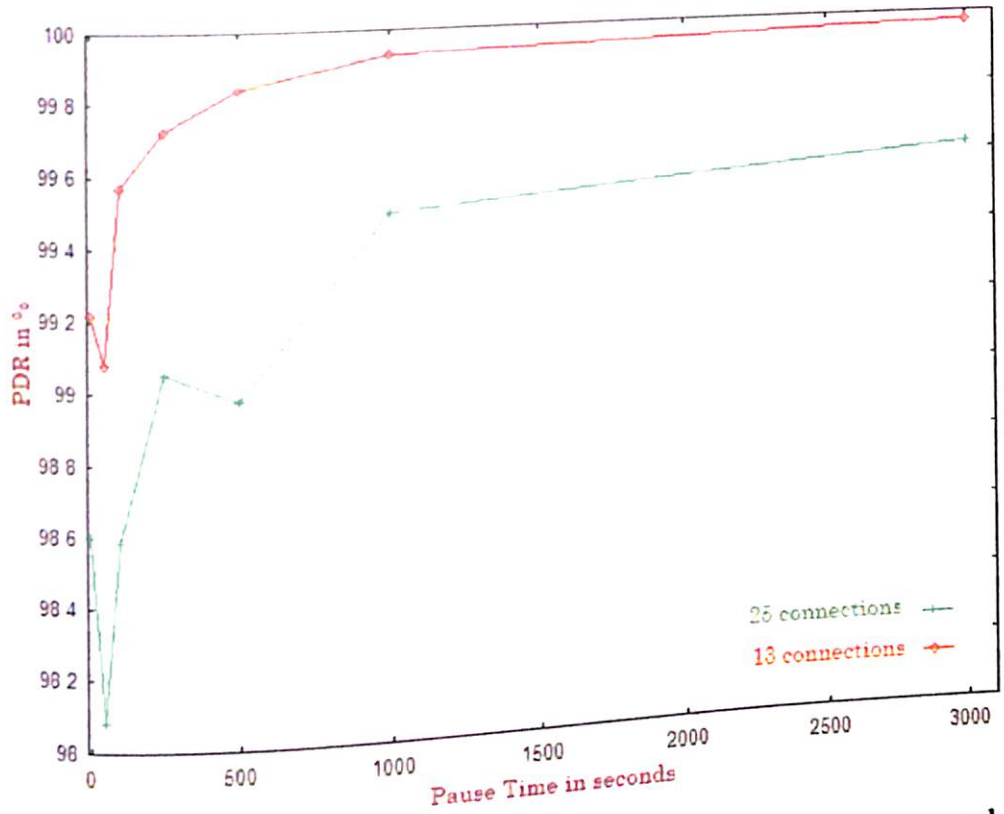
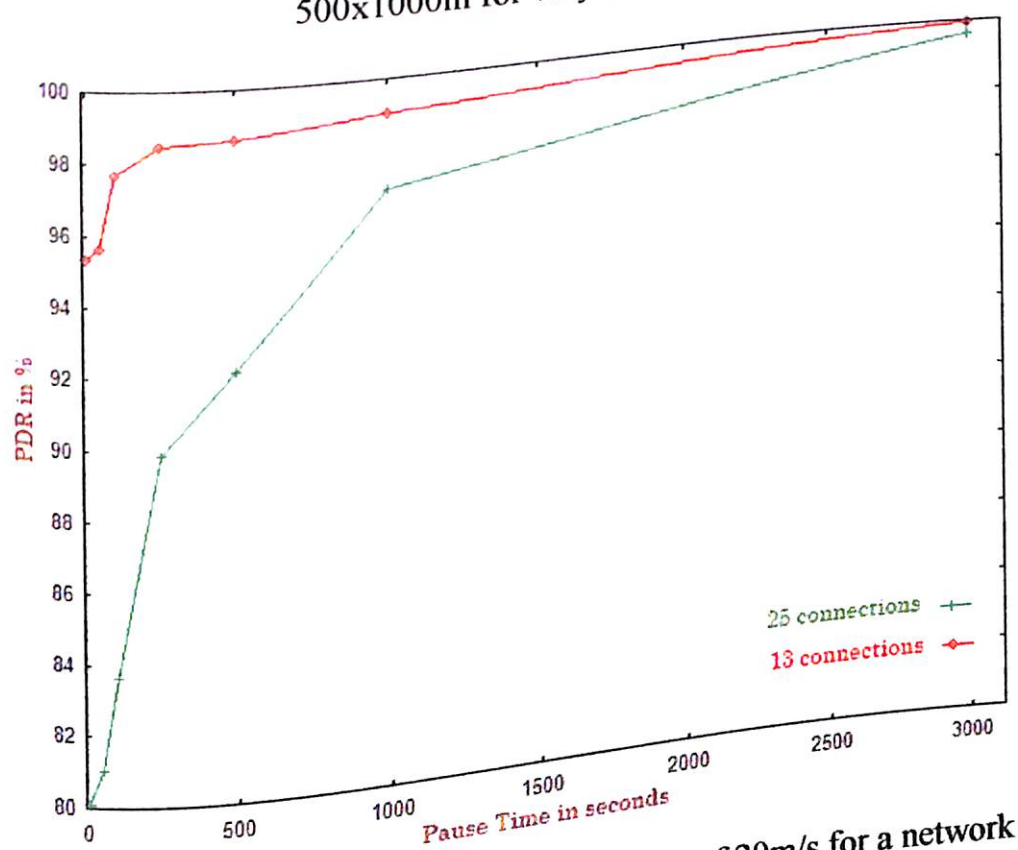


Fig 5.160 Network scenario 4 - 100 Nodes distributed over an area of 1500 x 1500m

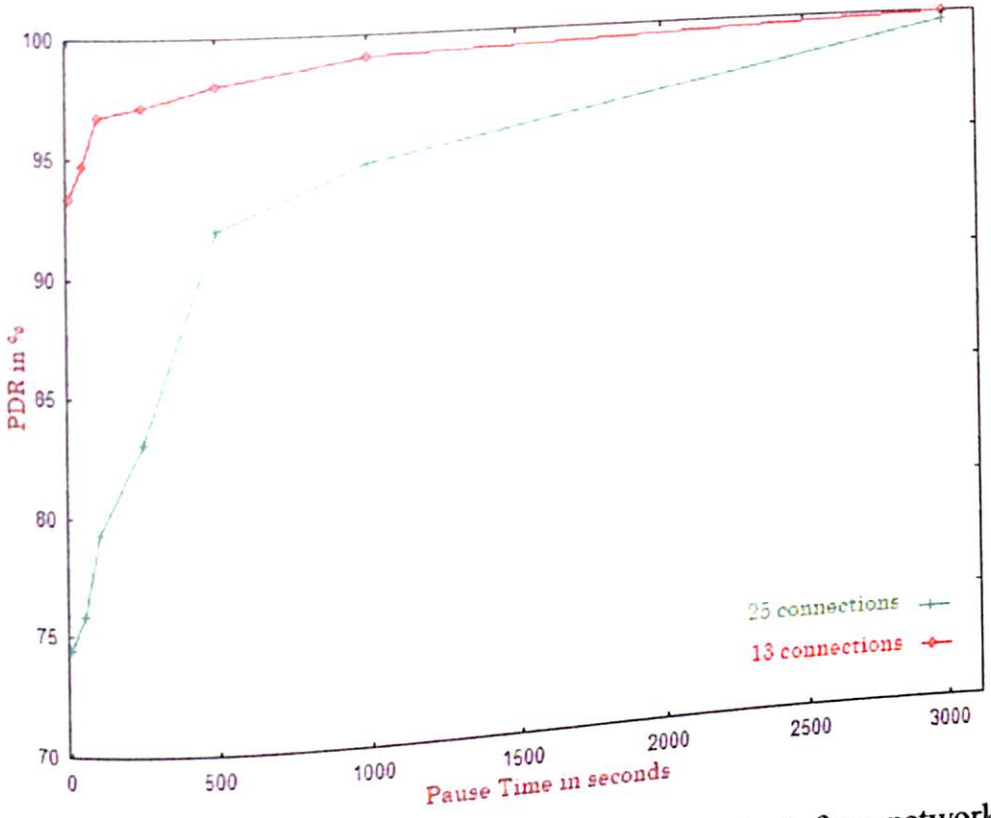
Scalability Analysis of GPEAR
(fig. 5.161 - fig. 5.168)



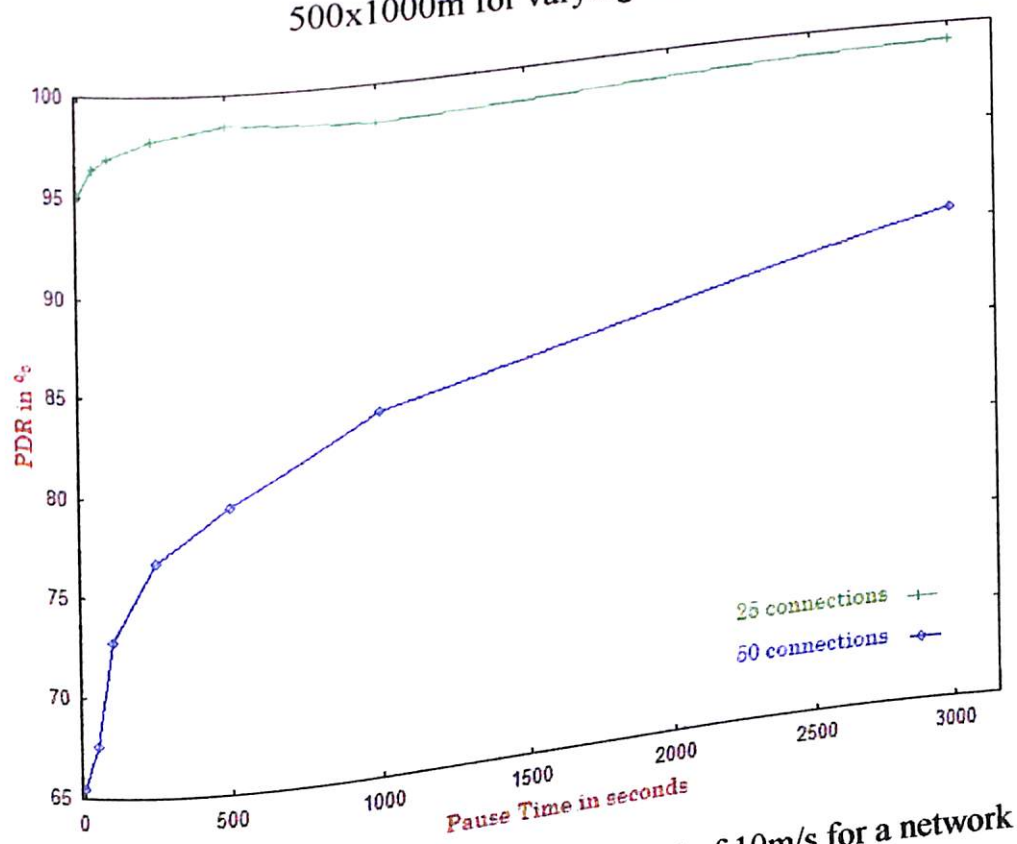
5.161 PDR Vs Pause Time at a maximum node speed of 10m/s for a network area of 500x1000m for varying traffic



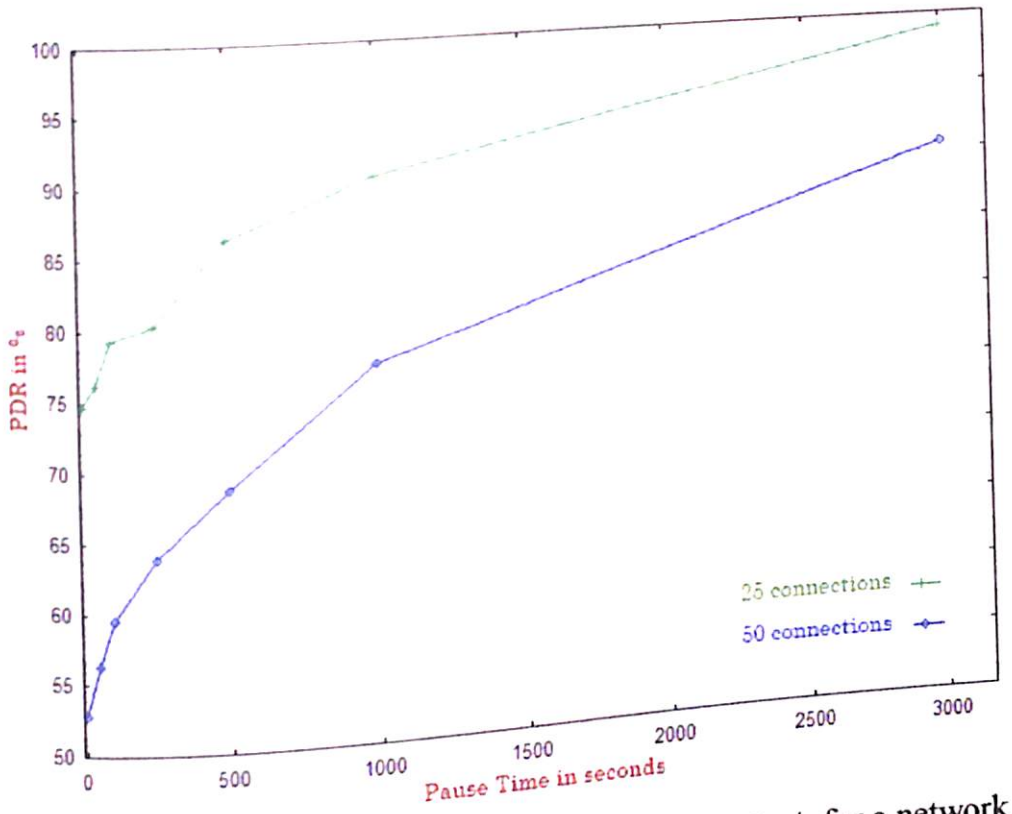
5.162 PDR Vs Pause Time at a maximum node speed of 20m/s for a network area of 500x1000m for varying traffic



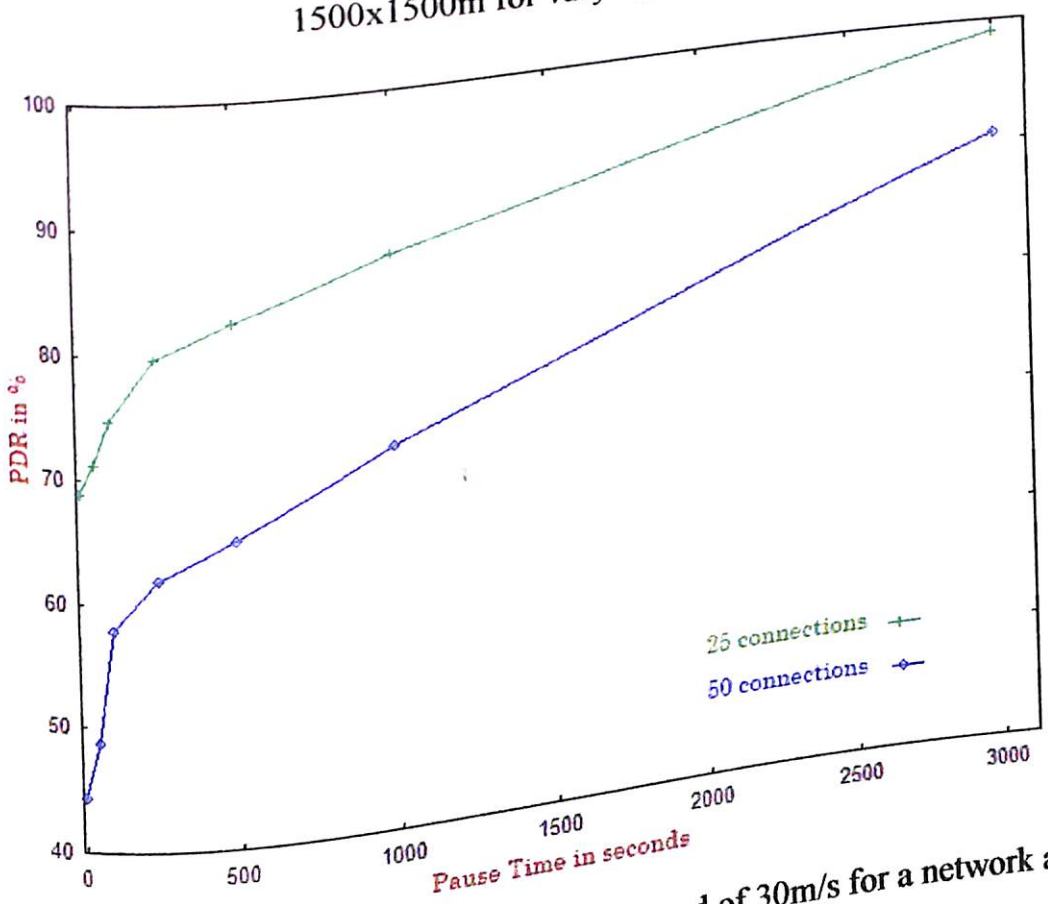
5.163 PDR Vs Pause Time at a maximum node speed of 30m/s for a network area of 500x1000m for varying traffic



5.164 PDR Vs Pause Time at a maximum node speed of 10m/s for a network area of 1500x1500m for varying traffic



5.165 PDR Vs Pause Time at a maximum node speed of 20m/s for a network area of 1500x1500m for varying traffic



5.166 PDR Vs Pause Time at a maximum node speed of 30m/s for a network area of 1500x1500m for varying traffic

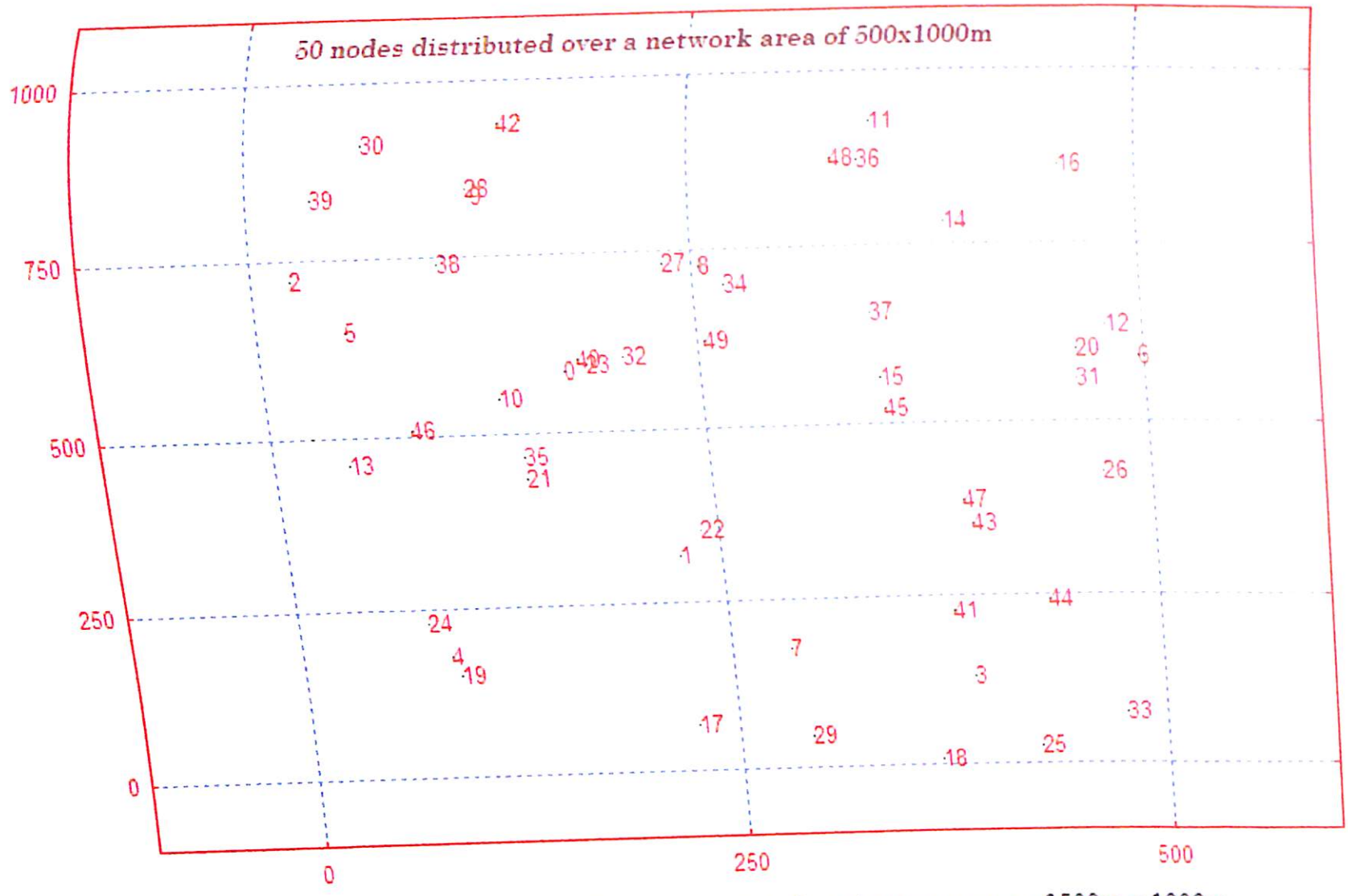


Fig 5.167 Network Scenario 5- 50 nodes distributed over an area of 500m x 1000m

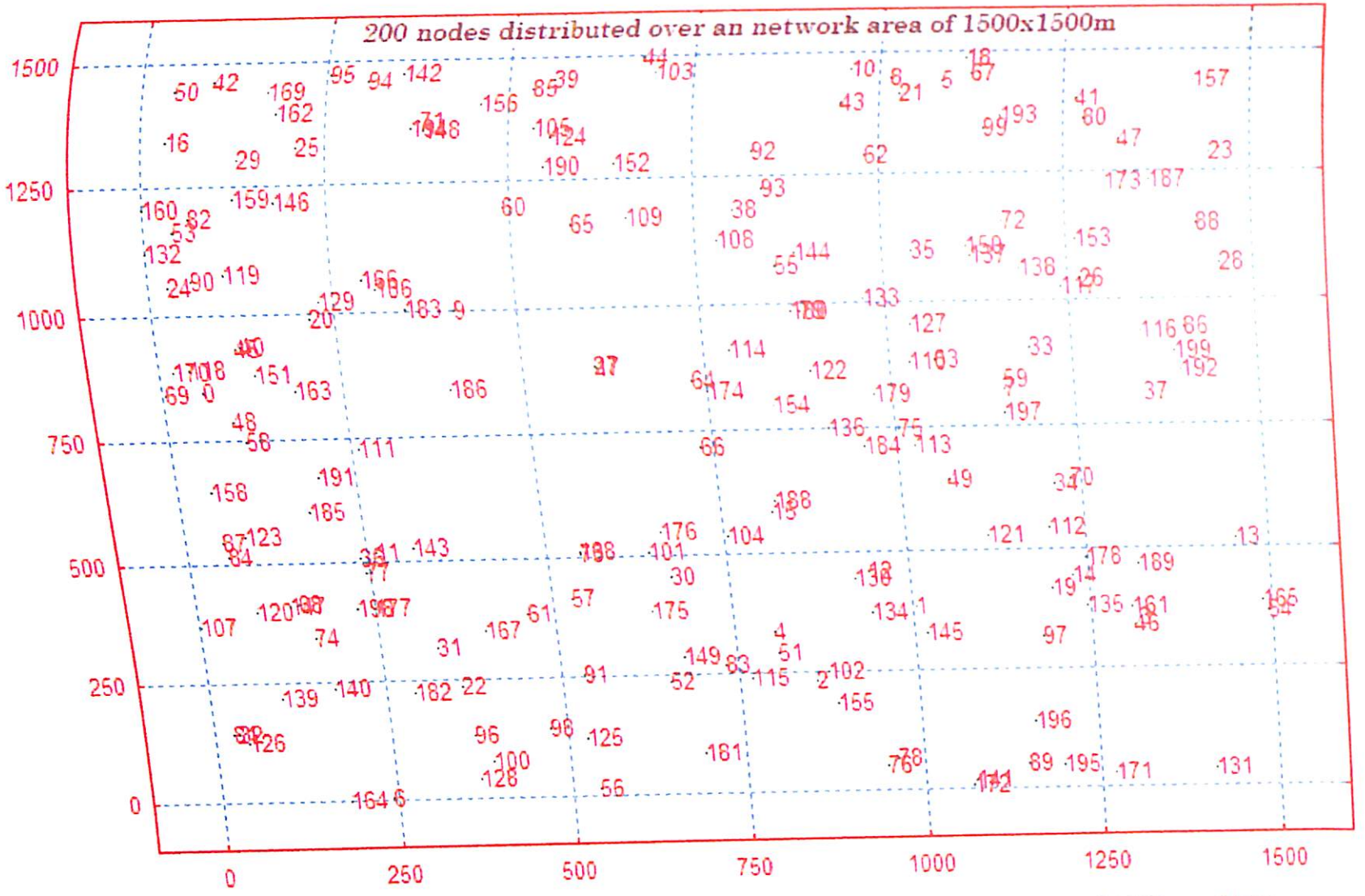


Fig5 .168 Network Scenario 6 - 200 nodes distributed over an area of 1500m x 1500m

Chapter 6 - MULTICAST ROUTING

A Multicast Routing Protocol Using Source Routing

6.1 Introduction

Multicast communication is very useful and efficient means of supporting group-oriented applications. This is especially the case in mobile/wireless environments where bandwidth is scarce and hosts have limited power. Example applications include audio- and video-conferencing as well as one-to-many data dissemination in critical situations such as disaster recovery and battlefield scenarios.

Many existing multicast routing approaches, used in fixed networks, rely on the state in the routers to keep track of multicast group members. This coupled with the high volume of routing information exchanges and slow convergence makes traditional multicast approaches untenable in highly dynamic networks composed of anaemic (low-power, low storage capacity) hosts. Therefore, new techniques that stress rapid and robust delivery must be developed.

The multicast protocol [49] described in this section enables dynamic self-starting multi-hop routing between participating mobile nodes wishing to join or participate in a multicast group within an adhoc network. The membership of these multi-cast groups is free to change during the lifetime of the network. The protocol enables mobile nodes to establish a tree connecting multicast groups. In the event of network partition, multi-cast trees established independently in each partition, are quickly connected if the network components merge.

One distinguishing feature of this protocol is that as trees are established they are simultaneously pruned. This protocol can be run on any underlying unicast protocol with minor modifications, as this protocol has its own set of control messages. The configuration parameters and various routing structures defined can be implemented on any unicast

protocol. As there existed no multicast protocol, which supported source routing; this generalized protocol was the initial step in the development of GPEAR.

6.2 Overview of the Protocol

6.2.1 Subscribing to a Multicast Group

A group formation cycle is initiated each time a node needs to find a route to or join a multicast group. A node may decide to initiate group formation because it would either like to subscribe to a new group, or because it would like to begin sending packets to a multicast group of which it is not already a member. The node initiates group formation by broadcasting a *Group Join (GJOIN)*, it then awaits a *Group Reply (GREP)* from a member of the multicast group. At the end of the formation period, the node unicasts a *Group Acknowledge (GACK)* message to its selected neighbor to activate the tree.

Terminologies

Upstream Neighbor: A node that has received GJOIN messages from a multicast node and in response, sends GREP messages to the originator that is acknowledged using GACK message, is the upstream neighbor of the originator.

Downstream Neighbor: A node, which has sent GJOIN messages and has received GREP messages in response, is the down stream neighbor of the node sending the GREP message.

Leaf Node: A node, which has no upstream neighbor.

Root Node: A node that has an upstream, but no downstream neighbor.

Group Join (GJOIN)

When a node desires to subscribe to a multicast group, it initiates group formation by broadcasting a GJOIN. The format of GJOIN message is shown in figure 6.1

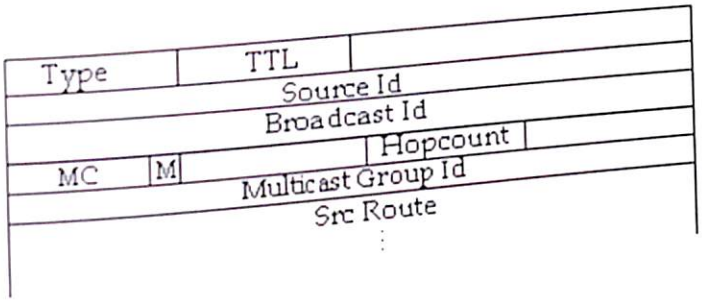


Fig 6.1 Format of GJOIN message

The GJOIN message format is same as RREQ format, with two additional fields, the one bit M Flag-which indicates whether the node forwarding the GJOIN packet is a multicast node and the 7-bit multicast count (MC) which indicates the number of multicast nodes that the source node can reach. A maximum of 128 nodes can be a part of a single multicast group when a 7-bit MC is used. Figure 6.2 shows the process of multicast tree formation.

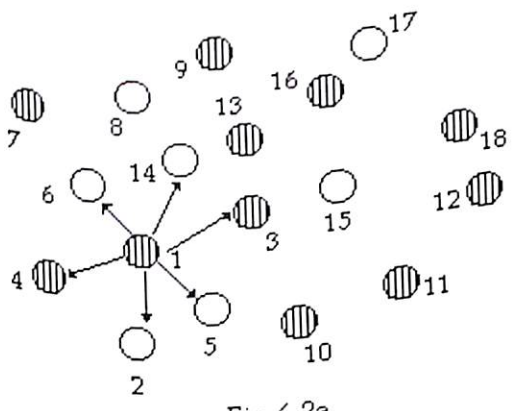


Fig 6.2a

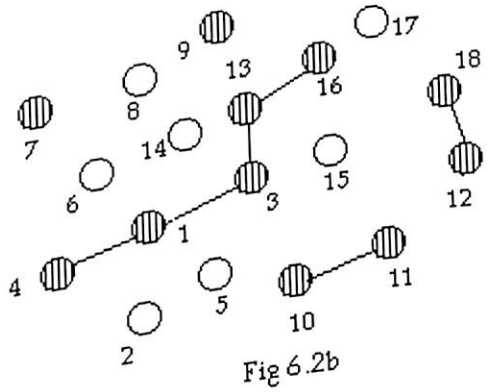


Fig 6.2b

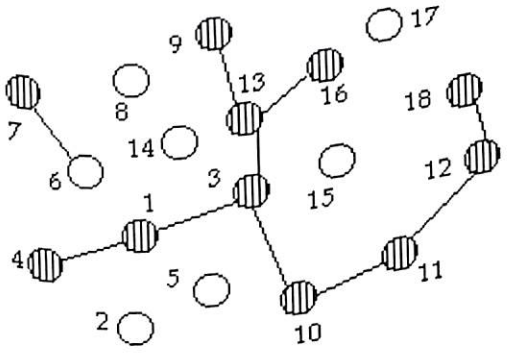


Fig 6.2c

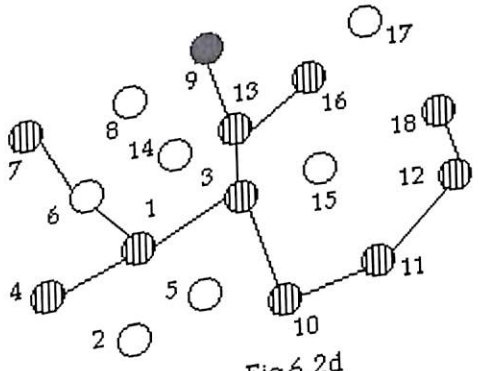


Fig 6.2d

Fig 6.2 Multicast Tree Formation

If node1 desires to join a multicast group it broadcasts a GJOIN message with the multicast ID as the target. Ring search is employed in multicast tree formation, so initially the TTL value set for GJOIN is 1; nodes 2,3,4,5,6 and 14, which are neighbors of node1, hence receive the GJOIN message.

Among the target nodes, Nodes 3 and 4 are either members of the multicast group already, or desire to join the multicast group, hence nodes 3 and 4 respond with **Group Reply** (GREP) messages. Nodes 2,4,6,14 will just drop the GJOIN message.

Before sending GREP messages in response, nodes3 and 4 will examine their respective *Multicast Tables*. The format of the Multicast Table is shown in fig 6.3

Node Id	MC	U/D	List of reachable multicast nodes

Figure 6.3 Multicast Table

A node stores the list of multicast nodes to which it is directly connected in its Multicast Table. It also stores the precursors to each multicast node and the precursor count is stored as the MC. Information regarding the relative position of the node in the multicast tree is also indicated (**upstream/downstream**). A node tries to match the ID of the node against the entries in the first and fourth columns of the multicast table (i.e. the list of directly reachable nodes and their precursors), before sending a GREP. It sends a GREP only if the source node is not listed in the multicast table. This prevents formation of any loops in the multicast tree and eliminates the necessity of pruning a tree once it has been established.

Since node1 is a new entrant into the multicast group node 3 and 4 respond with GREPs.

Group Reply (GREP)

The format of a GREP message is shown in fig 6.4. The GREP message in addition to the mandatory source header has information regarding the number of nodes reachable by the source of the GREP message and the ID of the nodes that can be reached.

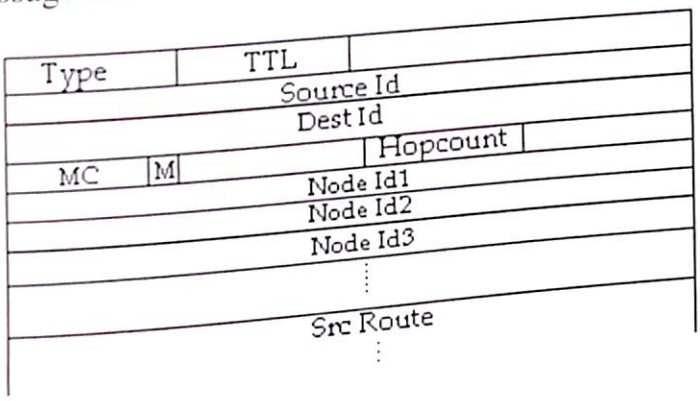


Fig 6.4 GREP Message Format

The Ids are presented in the order of connectivity. For e.g. from fig 5.2 b it can be observed if node 13 was to send a GREP then the Ids will be sent as linked list 3-1-4.This is to ensure that when 3 is removed from the multicast table its precursors are also removed from the table. Since the GREP messages are unicasted back to the source, this additional overhead in terms of the list of precursors is acceptable.

Nodes 3 and 4, which are still unconnected to the multicast tree, have a MC of zero. When Node 1 receives the GREP message, it waits for a time period of REQ_WAIT seconds. If there are no more GREPs within this period then node1 connects itself either to node 3 or 4. The decision is made on the basis of the following criteria:

- Number of Multicast Nodes (MC) that are reachable
- Number of Hops between node1 and node3/node4

In this case the MC of both nodes 3 and 4 are zero, so the decision is based on the number of hops. Since both 3 and 4 are neighbors of 1, node 1 responds to the node from which it has received the GPREP first.

Node 1 responds with a **Group Acknowledge** (GACK) message. Prior to transmitting the GACK message node 1 updates its multicast table and multicast count (MC

of node3 +1). The updated multicast table of node1 is shown in fig 6.5. Node 3 therefore is now node 1's designated upstream neighbor.

Node Id	MC	U/D	List of reachable multicast nodes
3	0	U	

Figure 6.5 Multicast Table of node 1

Group Acknowledge (GACK)

The format of GACK message is shown in fig 6.6

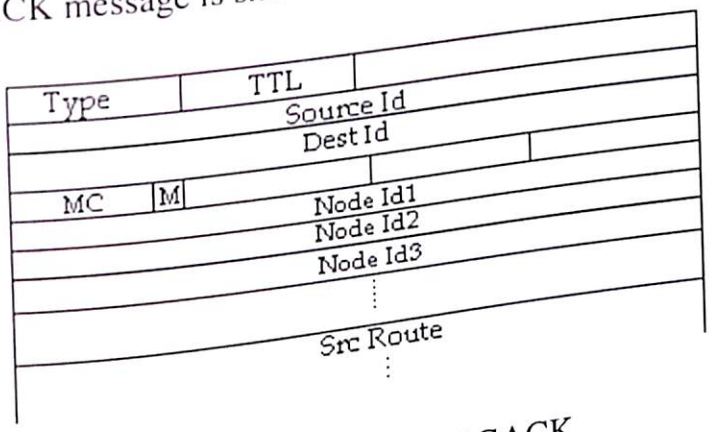


Figure 6.6 Format of GACK

When node 3 receives the GACK message it updates its multicast table and multicast count. It designates node 1 as its downstream neighbor. Node3 checks its multicast table for an upstream neighbor. If it has no upstream neighbor, it broadcasts a GJOIN message in its turn to its neighbors, in this case nodes 1,5,10,13,14,15 of which nodes 1,13,10 are multicast nodes. Node 1 does not respond to the GJOIN as it already has an entry for node 3 in its multicast table. As a result loop formation is prevented, and the tree is automatically pruned. Node 13 due to the tree formation process becomes node 3's upstream neighbor. The tree thus propagates as shown in figs 6.2b,c &d. Node 9 has no upstream neighbor hence it designates itself as the **leaf node**.

6.2.2 Group Maintenance

Group Leave

When any node desires to leave the multicast group, it clears its multicast table and sends a Group Leave (GLEAVE) message to its upstream and downstream neighbors. The format of GLEAVE message is shown in figure 6.7

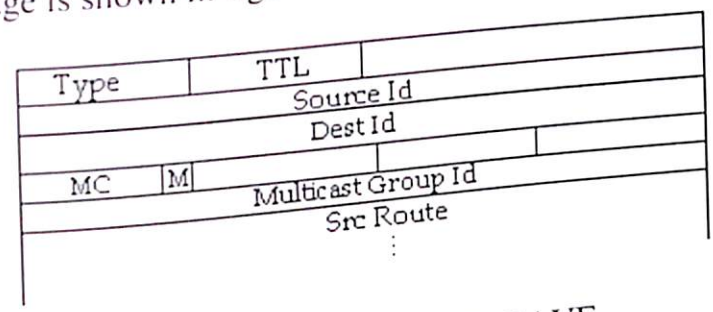


Figure 6.7 Format of GLEAVE

The process of tree maintenance in shown fig 6.8

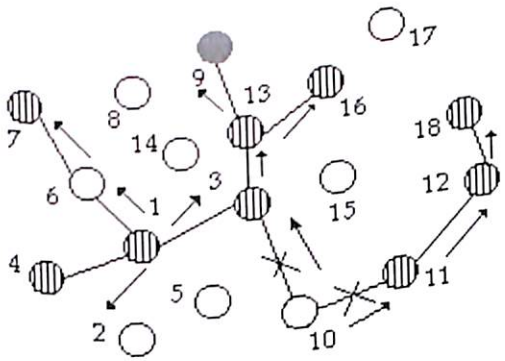


Fig 6.8 a
Propagation of GLEAVE

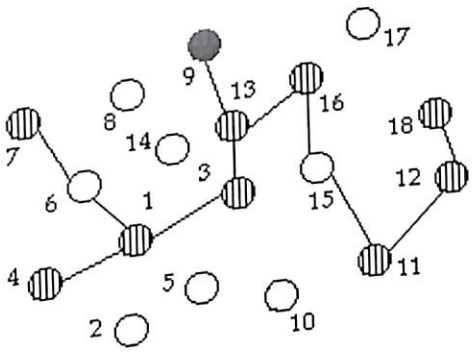


Fig 6.8 b
Tree Re-Formation

Fig 6.8 Tree Maintenance

From fig 6.8a, if node 10 desires to leave the multicast group it sends GLEAVE to its upstream (Node 3) and downstream (Node 11) neighbors.

When node 3 gets the GLEAVE message, it updates its multicast table, multicast count and forwards the GLEAVE message to the upstream and the downstream (except Node 10) branches. Node 3 will remove node 10 as well its precursors from the multicast table.

When node 11 gets the GLEAVE message, it updates its multicast table, multicast count, forwards the GLEAVE and attempts to find itself an upstream neighbor by means of Secondary GJOIN (SGJOIN) messages. The format of SGJOIN is same as GJOIN, except

that it is sent whenever a link break occurs. A node waits for a period of G_ROUNDTRIP seconds before responding to a SGJOIN. This is to ensure that all members of the multicast tree get the GLEAVE message; else node 11 may not be able to find an upstream neighbor for itself. The response thereafter proceeds along the same lines, as the response, to the GJOIN messages. Node 11 now connects itself to node 16 through node 15.

When a leaf node gets a GLEAVE, it updates its multicast structures, forwards the GLEAVE (if required) and tries to get itself a new upstream neighbor, it will be able to do so now since it has no connection to the tree whether upstream or downstream.

If a node receiving the GLEAVE message does not have the source of GLEAVE message as it's direct upstream or downstream neighbor, it checks whether the source is listed as a precursor to any of its upstream or downstream neighbors. If so, then it deletes the node and the entire branch from the precursor list and updates the multicast count.

Group Error (GERR)

Nodes keep moving in a mobile scenario, as the upstream and downstream neighbors of a node move, links break and hence the multicast tree has to be re-established. When a node forwarding a packet finds that it is unable to reach its neighbor, it propagates a Group Error (GERR) message to its upstream and downstream neighbors. The format of GERR message is shown in fig6.9.

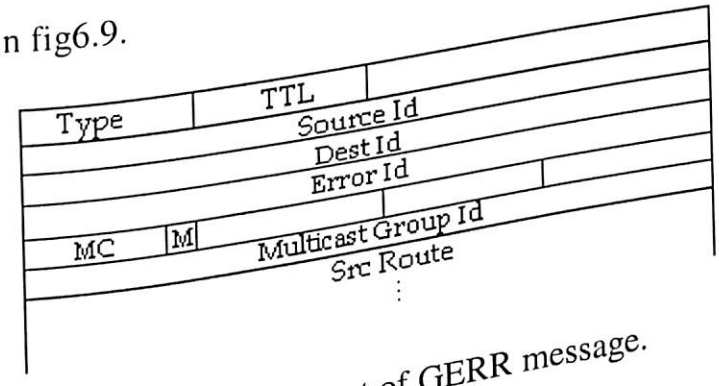


Figure 6.9 Format of GERR message.

The formats of GLEAVE and GERR messages are similar except in case of GERR the ID of the node that has failed is included.

On receiving the GERR message, the same sequence of actions will be executed by the recipient of the message, as in case of GLEAVE.

The process of a link break and the sequence of corrective actions taken as a result are shown in fig 6.10. In figure 6.10 a link break occurs due to the failure of node3.

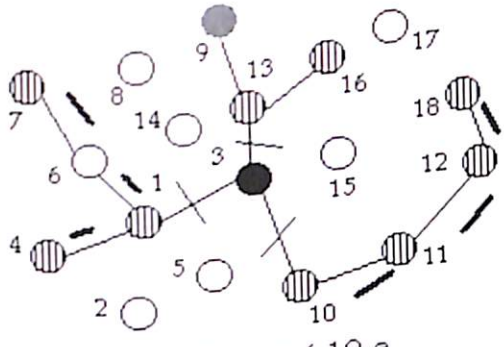


Figure 6.10 a

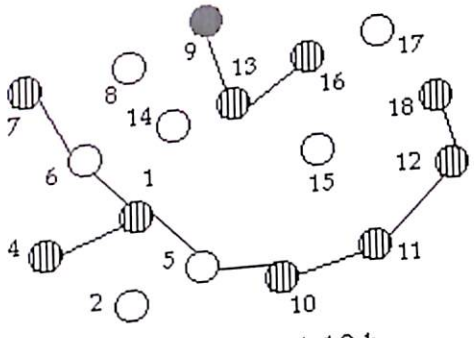


Figure 6.10 b

RERR Message

Figure 6.10 The Route Error Process

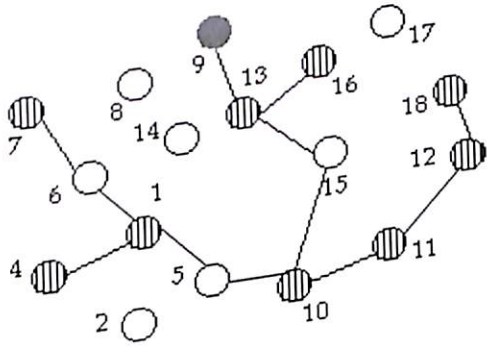


Figure 6.10 c

Figure 6.10 The Route Error Process

Node 13 that has lost its upstream neighbor, tries to re- establish a connection with the multicast tree. By use of the Group Join process node 13 connects itself to the multicast tree thro' node 10.

Hence the multicast tree is able to maintain itself in the face of network dynamics that are a special characteristic of MANETs.

Due to the absence of any multicast protocol, which supports source routing, this generalised protocol was developed as an envelope, using which a multicast protocol for GPEAR could be developed.

Chapter 7 - MULTICAST ROUTING

Multicast - GPS based Predictive Energy Aware Routing in MANETs (MGPEAR)

7.1 Introduction

The protocol presented in the previous chapter, for multicast routing in MANETs can be implemented on any underlying unicast protocol with minor changes. During an active communication all members of a multicast tree may be mobile. The tree structure therefore changes frequently. This poses a difficult challenge for multicast protocols since rapid reconstruction of the tree is crucial. Rather than reconstructing the tree after link breaks, MGPEAR, like its unicast counterpart GPEAR uses location information available via GPS, to predict expiry of links and reconstruct the multicast tree in advance.

MGPEAR's primary objectives are:

- To reduce multicast tree breaks by predicting link breakages in advance by the use of location information.
- To reduce energy consumed at each node by controlling the energy at which packets are transmitted or received by the use of location information.

MGPEAR attempts to provide the same kind of service for multicast routing that GPEAR offers for unicast.

7.2 Overview of the protocol

7.2.1 Subscribing to a Multicast Group

A group formation cycle is initiated each time a node would like to find a route to a multicast group. A node initiates group formation by broadcasting a Group Join (GJOIN). It then awaits a Group Reply (GREP). At the end of the tree formation period, the node

unicasts a Group Acknowledge (GACK) message to its selected neighbor to activate the tree. The process of subscription to a multicast group in MGPEAR is similar to the process used in the general Multicast protocol, except that any decision made about tree formation is position centric.

Group Join (GJOIN)

When a nodes wishes to subscribe to a multicast group, it initiates the process by broadcasting a GJOIN message; the format of the GJOIN message is shown in fig 7.1.

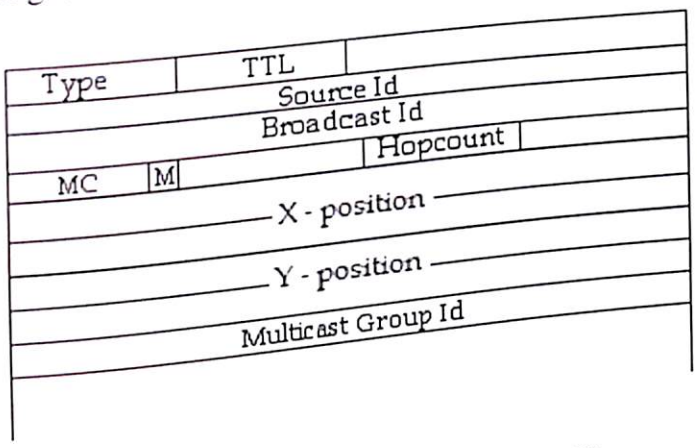


Fig 7.1 Format of GJOIN message

The format of GJOIN message is similar to that of GJOIN message in the general multicast protocol, the additional fields are the X and Y co-ordinates of the source.

The process of tree formation is shown in fig 7.3. Node 1 starts the process of tree formation by transmission of GJOIN message, nodes 3 and 4 respond with GREP. Nodes 2 and 5 drop the GJOIN message, *since the underlying unicast protocol GPEAR employs ring search* and as they are not members of the multicast group they need not respond to the GJOIN message.

Nodes 3 and 4 scan their multicast tables before responding to the GJOIN, the GREP is sent if node 1 is not listed as a neighbor or as a precursor to any of the neighbors. Before sending a GREP message, nodes 3 and 4 update their routing structures. The neighbor table (fig 7.2) is modified to accommodate multicast routing; an additional column (Mcast

ld) stores information regarding the multicast affiliation of each neighbor. The information is used by the node; for tree formation and maintenance.

Node Id	My Position				Neighbors Position				Time	Let	Mcast Id
	Xprev	Yprev	Xcur	Ycur	Xprev	Yprev	Xcur	Ycur			

Fig 7.2 Neighbor Table

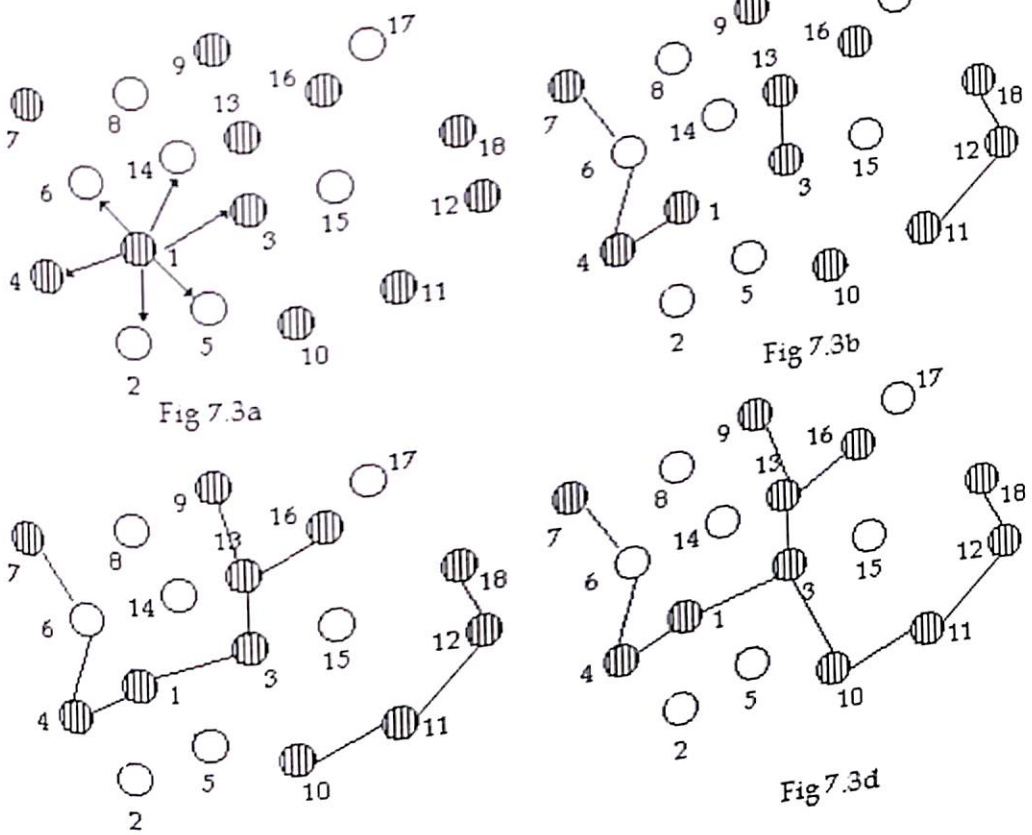


Fig 7.3 Multicast tree Formation

Nodes 3 and 4 will now respond with GREP messages. The format of GREP messages is as shown in fig 7.4.

The format of GREP message for MGPEAR is similar to the generalised multicast protocol, as is the case of the GJOIN messages the additional information is position related; the X-Y co-ordinates and the estimated RET (this RET information is static as it is calculated based only on distance between the neighboring nodes and average speed at which nodes are moving).

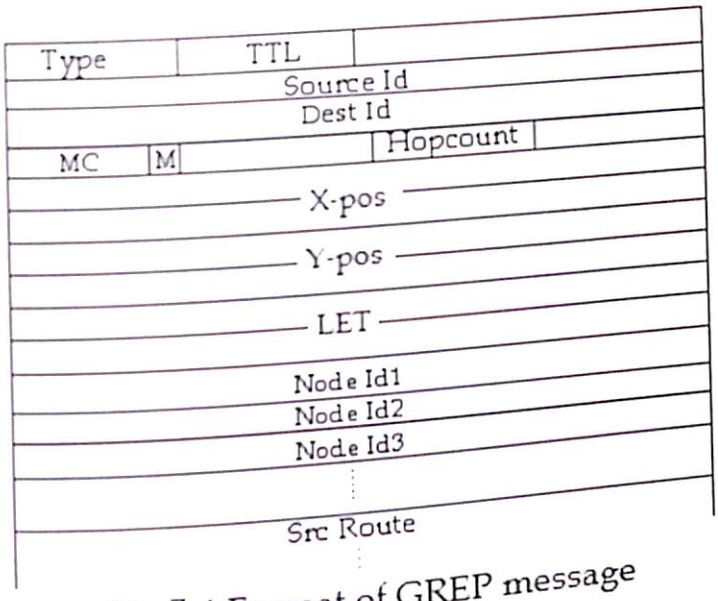


Fig 7.4 Format of GREP message

When node 1 receives the GREP message, it updates its routing structures. Node 1 receives GREP message from both node 3 and 4. In case of the generalised protocol, link decisions were made based on MC (multicast count) and minimum number of hops, in the case of MGPEAR, the decision is made based on RET. The link selection criteria is obtained using the relationship

$$W_1^* (RET) + W_2^* (MC) + W_3^* (MAX\_TTL - number\ of\ hops)$$

Where: RET – is the Route expiry time that is calculated based on position information.

MC – is the multicast count – this value is available in the GREP message.

MAX_TTL – The maximum number of hops through which a multicast packet is allowed to traverse.

The weights (W_1, W_2, W_3) allotted to the three selection criteria need not be evenly distributed. Allocation of weights is based on the mobility pattern, as the mobility increases the weightage allotted to RET is increased [27,28]. During initial tree formation, the weights are allotted statically; as the mobility pattern emerges, the protocol (MGPEAR) automatically updates weights. MGPEAR can allocate any value ranging from 0 –1, based on the estimated movement of nodes within the network. The movement pattern is obtained from the position information that is circulated in the GUPDATE messages.

Since both 3 and 4 are currently not connected to the multicast tree and are one hop away, the decision is based entirely on the RET (in this case static). Node 4 is closer to node 1 when compared to node 3, hence will be picked as the upstream neighbor, and is sent a GACK message. The format of GACK message is similar to the GACK message in multicast protocol. Update of position is available from the GPS every 2 seconds; the time involved in setting up a link in the tree is much lesser than 2 seconds, hence position information need not be appended with GACK messages.

The multicast tree formation progresses as shown in fig 7.3. The multicast tree formed in fig 6.2 using generalised multicast protocol is different from the multicast tree formed using MGPEAR since the multicast link selection criteria is different. In case of the generalised multicast protocol, link selections is based on the number of reachable multicast nodes, whereas in case of MGPEAR, the position of a node along the multicast tree is also taken as a selection criteria.

7.2.2 Group Maintenance

The group maintenance mechanism employed by MGPEAR is advanced when compared to the general multicast protocol as link breakages are predicted in advance using position information. Since the nodes are mobile, position information has to be constantly updated along the multicast links; this is done using Group Update (GUPDATE) messages. The leaf node starts the propagation of GUPDATE messages. Each multicast group has only one leaf node; hence at a time only a single set of GUPDATE message will be originated in a multicast group. The propagation of GUPDATE messages is shown in fig 7.5

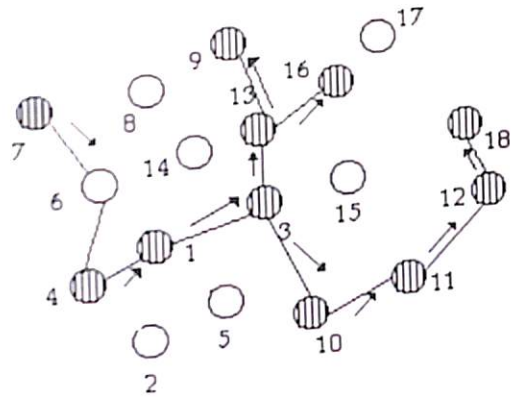


Fig 7.5 Propagation of GUPDATE messages

When a node receives a GUPDATE message it forwards the same to its various downstream neighbors. Each node has only one upstream neighbor; this will ensure that only one copy of a GUPDATE message reaches each node. The format of RUPDATE message is given in fig 7.6

It can be observed from fig 4.6 and fig 7.6 that the format of RUPDATE and GUPDATE messages are similar. The purpose of RUPDATE and GUPDATE messages are the same, to update neighbor position information. Position information storage and update are local in MGPEAR as is the case of GPEAR.

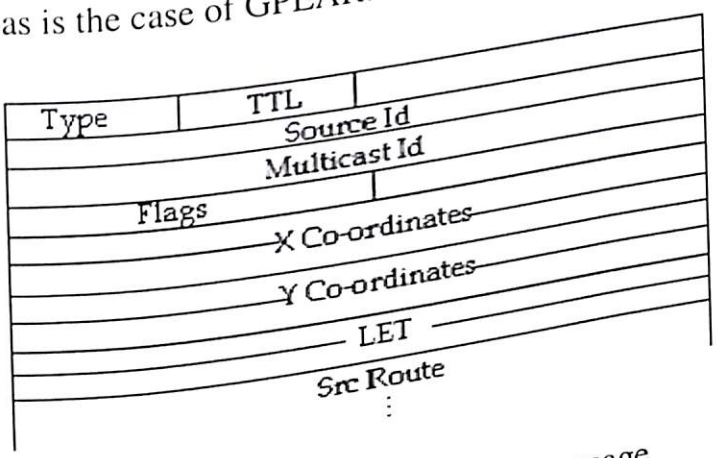


Fig 7.6 Format of GUPDATE message

From fig 7.5 it can be observed that node 7, which is the leaf node (for the network topology shown in fig 7.5) originates the GUPDATE message. It transmits its current X –Y co-ordinates, and the LET is set to MAX_RET. Unlike the case of route maintenance, LET information gives only the strength of multicast links, not the entire tree. When node 6 receives the GUPDATE message, it forwards the GUPDATE message to node 4. Node 6 is

not a multicast node but Node 6's position information is vital to calculate the strength of the multicast link between nodes 7 and 4. So node 6 treats a GUPDATE message like a RUPDATE message (i.e. replace position information with its own, calculate the LET, replace the minimum of the received LET and the calculated LET in the LET field of the source header of GUPDATE message). Node 4 on receiving the GUPDATE message updates its neighbor table, calculates the LET, and forwards the GUPDATE message to its downstream neighbor. Before forwarding the GUPDATE message, the node replaces the position information in the source header with its own, and sets the LET to MAX_RET. The LET field is used mainly when one or more unicast nodes connect two multicast nodes together. On calculation of LET if node 4 detects that its direction of motion is opposite to that of node 6 and that the LET calculated is less than T_OPT, then it sends out a Group Leave message (GLEAVE) message, after delay of T_Multi_Setup. The process of GLEAVE is temporary. The node can rejoin the multicast group, by sending out GJOIN messages. The process of GUPDATE continues till a *root node* is reached.

GLEAVE

The process of GLEAVE and the format of GLEAVE messages are similar to the generalised multicast protocol except that GLEAVE message is generated under two conditions:

(i) When a node desires to leave the multicast group

(ii) When a node detects a probability of a multicast link breakage.

GERR

Despite the fact that link breaks are detected in advance, GERR messages are also a part of the protocol. This is because link breaks may occur due to transmission failure or if a node's battery is down. *The format of GERR messages and the process of GERR remains the same as the generalised multicast protocol.*

It can be observed from the comparison of GPEAR and MGPEAR that, the process by which a route is formed is quite similar to multicast group formation (except for the process of GACK messages) and process of route maintenance and group maintenance is similar (except for the process of GLEAVE).

It can be argued that the same set of control messages can be used by both the unicast and multicast protocols. In such a case a series of flags have to be added to distinguish between unicast and multicast operations; as it has been done in the case of MAODV [50]. Instead of adding an additional field of flags and increasing the size of the already over-sized source header, a separate set of messages form a part of the multicast process.

Processing burden on a node is not increased due to the additional set of messages, since any node is given the option of enabling or disabling multicast functions. When a single set of set of control messages are used by the unicast and multicast protocols, the flags determine how each message has to be handled. As different actions have to be taken in case of multicast routing, when compared to unicast routing, the header fields have to be examined in detail. In case of a separate set of messages nodes, which do not wish to participate in any multicast activity, even forwarding, can always drop the control message after examining the type of message.

DSR uses broadcast mechanism for transmission of multicast messages, by piggybacking multicast messages on control message, which are broadcasted [51]. This leads to heavy broadcast storming, especially due to the large size of the control packets. Though the burden of tree formation and maintenance is added to a node, MGPEAR will be able to increase packet delivery ratio by reducing broadcast storms created by broadcast of multicast packets.

MGPEAR has been modified to support geocasting and the resultant protocol GGPEAR is presented in the next chapter.

Chapter 8 - GEOCAST ROUTING

Geocast - GPS based Predictive Energy Aware Routing in MANETs (GGPEAR)

8.1 Introduction

When an application needs to send the same information to more than one destination, multicasting is often used, because it is much more advantageous than multiple unicasts in terms of the communication costs. Cost considerations are all the more important for a mobile adhoc network (MANET) consisting of mobile hosts that communicate with each other over wireless links, in the absence of a fixed infrastructure. In MANET environments, the multicast problem is more complex because topology change of the network is extremely dynamic and relatively unpredictable. To do multicasting, some way is needed to define multicast groups.

In conventional multicasting algorithms, a multicast group is considered as a collection of hosts, which register with that group. It means that, if a host wants to receive a multicast message, it has to join a particular group first. When any host, has to send a message to a group of nodes, it simply multicasts the message, to the address of that group. All the group members then receive the message. A **geocast message** [52], on the other hand, is delivered to the set of nodes within a specified geographical area. Unlike the traditional multicast schemes, here, the multicast group (or Geocast group) is implicitly defined as the set of nodes within a specified area. The specified area is termed as the "**geocast region**", and the set of nodes in the geocast region as the **location-based geocast** group. If a host resides within the geocast region at a given time, it will automatically become a member of the corresponding geocast group at that time. Geocasting may be used for sending a message that is likely to be of interest to everyone in a specified area.

Two approaches may be used to implement location-based multicast *(geocast)*.

- Maintain a multicast tree, such that all nodes within the geocast region at any time belong to the multicast tree. The tree has to be updated whenever nodes enter or leave the geocast region.
- Do not maintain a multicast tree. In this case, the geocasting may be performed using some sort of "flooding" scheme.

Geocast schemes have been proposed using the concept of multicast flooding [53]. Two zones are used for this process, **GEOCAST ZONE** and **FORWARDING ZONE**. A packet destined for a particular geographical zone is flooded into the forwarding zone. The forwarding zone has a larger cross-section when compared to the geocast zone, which ensures that the packet reaches the entire geographical area termed as the geocast zone. The difference in cross-section between the geocast zone and the forwarding zone (δ) maybe either fixed or variable, depending on the protocol.

The disadvantage of such a scheme is that entire protocol becomes dependent on the size of the forwarding zone. If the forwarding zone is small, the PDR is reduced. If the size of the forwarding zone is large, then broadcast storms may cause a problem. Further the broadcast storming will increase exponentially with the geographical area of the geocast zone. Even if a scheme is envisaged that will be able to find the perfect δ value, a packet will be received multiple number of times by a single node.

Hence GGPEAR uses multicast trees for geocasting of packets. GGPEAR is modified version of MGPEAR, where multicast groups are defined based on their geographical location. *GGPEAR was developed to aid Inter-Zone Routing* (chapter9).

Some geocasting protocols go in for **geographical addressing** [54], where the address of the node is obtained on the basis of its physical location. The disadvantage of using geographical addressing is that a node can move only within a particular geographical area. Hence with such a scheme a node is portable not mobile. When a node leaves a

particular geographical area, its address has to be changed. The change of address will not create a problem as long geocasting is the only type of traffic in the network, but if unicast and multicast traffic patterns are also supported, the change in address will require a network protocol like Mobile IP to handle the change in address.

8.2 Overview of the Protocol

8.2.1 Subscribing to a Geocast Group

A group formation cycle is initiated each time a node would like to join a multicast group in case of multicasting. In geocasting when any node moves into a particular geographical area it automatically becomes a member of that multicast group. Fig 8.1 shows the division of network area into geocast groups. The zones can be of any geometrical shape. In fig 8.1 the zones are squares of area 250x250m. There are totally 16 zones in fig 8.1. Consequently there would be a total of 16 geocast groups *(g1-g16)*. The geocast zones may also overlap. Any node can be a member of one or more geocast groups, as in the case of multicast routing.

When a node enters a particular geographical zone (for e.g. 250x500) it automatically becomes a member of geocast zone *g5*, but if it desires to receive messages that are meant for the *g5* group, it has to connect itself to the geocast tree.

If a flooding scheme is used within a geocast group, all nodes within a particular geographical area receive and process multiple copies of the messages, irrespective of whether they actually want to be a member of the geocast group or not. In GGPEAR, a node is given a choice of not joining a geocast group, if it does not want to participate in any of the geocasting activities.

When a node finds itself in a particular geographical zone, and desires to join the geocast group, it transmits a GROUP JOIN (GJOIN) message, and the tree formation continues as it does in case of MGPEAR.

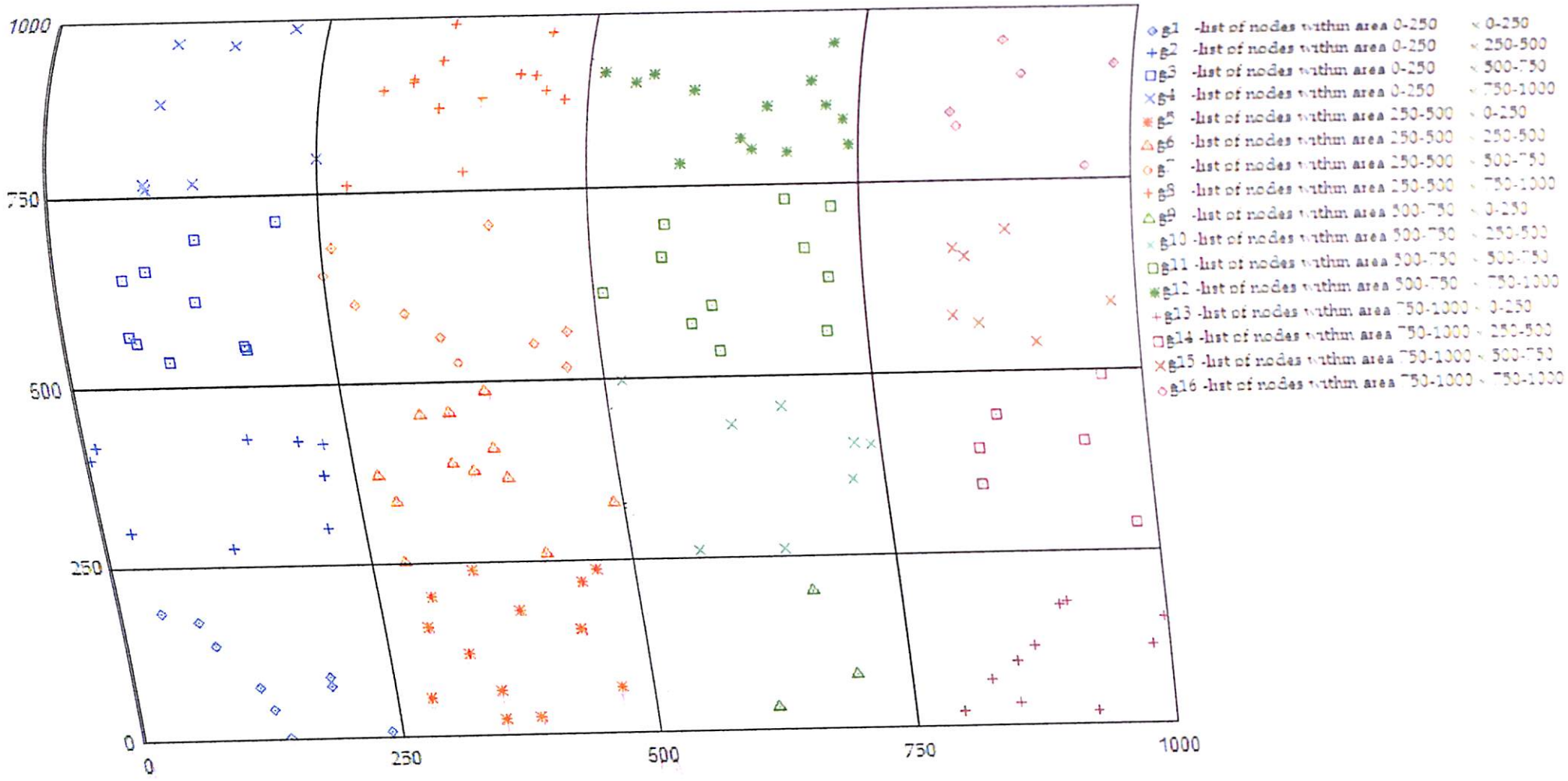


Fig 8.1 Geocast Zones- distribution of nodes within geocast areas

8.2.2 Geocast Group Maintenance

Geocast group maintenance is similar to group maintenance used in MGPEAR. In each group, the leaf node sends out Group Update (GUPDATE) messages at regular intervals. The format, purpose and response to a GUPDATE message are similar to MGPEAR. The only difference in group maintenance mechanisms, between the geocast and multicast routing protocols is that, in case of geocasting, a node automatically has to generate a GLEAVE message, when it realizes that it is moving out of the geographical zone, which constitutes the multicast zone.

Chapter 9 - SCALABLITY

GPS-based Predictive Inter-Zone Routing (GPIZR)

9.1 Introduction

A fundamental aspect of adhoc networking is that all nodes are "equal" and therefore any node can be used to forward packets between arbitrary sources and destinations. This aspect is realistic in military (battlefield, search and rescue) and civilian emergency situations.

The routing strategies are developed based on this aspect. The question is how well do these protocols scale in case of large populations, network area and node density? This problem is practically relevant since one can foresee that in the near future most of the commercial laptops and PDA's will be equipped with radios enabling them to form adhoc "virtual" wireless networks. The problem is particularly challenging, because of the presence of both large number and mobility. If the nodes are stationary, the large population can be handled with conventional hierarchical routing. In contrast, when nodes move, the hierarchical partitioning must be continuously updated - a significant challenge. Mobile IP [5] solutions work, if there is a fixed infrastructure supporting the concepts of "Home Agent" and "Foreign Agent". When all nodes move (including the agents) such a strategy cannot be directly applied.

Existing wireless routing schemes can be classified into three broad categories:

- **Global, pre-computed routing:** routes to all destinations are computed in advance and are maintained in the background via a periodic update process. Most of the conventional routing schemes, including Distance Vector and Link State (LS), fall in this category.
- **On demand routing:** the route to a specific destination is computed only when needed.

- **Flooding:** a packet is broadcast to all destinations, with the expectation that at least one copy of the packet will reach the intended destination. Directional scoping may be used to limit the overhead due to flooding.

Global, pre-computed routing schemes could be further subdivided into two further categories: flat and hierarchical [14]. In flat routing schemes each node maintains a routing table with entries for all the nodes in the network. This is acceptable if the user population is small. However, as the number of mobile hosts increases, so does the overhead. Thus, flat routing algorithms do not scale well to large networks. To permit scaling, hierarchical techniques can be used.

The major advantage of hierarchical routing is a drastic reduction in routing table storage and processing overhead. A hierarchical clustering (CBR) and routing approach specifically designed for large wireless network was recently proposed [55]. The proposal addresses the link and network layers only, and is independent of the physical/MAC layer. The network contains two kinds of nodes, endpoints and switches. Only endpoints can be sources and destinations for user data traffic, and only switches can perform routing functions. To form the lowest level partitions in the hierarchy, endpoints choose the most convenient switches to which they will associate by checking radio link quality. Autonomously, they group themselves into cells around those switches (cluster heads). This procedure is called "cell formation". Each endpoint is within one hop of the switch with which it is affiliated. The switches, in turn, organize themselves hierarchically into clusters, each of which functions as a multihop packet-radio network. First level cluster heads organize to form higher level clusters, and so on. This procedure is called *"hierarchical clustering"*. A switch is a 0th order cluster.

As nodes move, clusters may split or merge, altering cluster membership. To support data transfer between mobile nodes, it is necessary to keep track of their locations. To this end, both paging and query/response are used in conjunction with a location manager.

Each cluster has its location manager, which keeps track of nodes within the cluster and assists in locating nodes outside the cluster. Each node has a roaming level, which is specified with respect to the clustering hierarchy and which implicitly defines a roaming cluster at the corresponding level. Paging is used to locate a mobile node within its current roaming cluster. When a node moves outside of its current roaming cluster, it sends a location update to the location manager. This update propagates to the highest level from which inter-cluster movement is visible. By combining these hierarchical topology management and location management functions, hierarchical routing can be extended to the mobile environment. In this scheme there are several features, which are potentially complex to implement and hinder scalability. First, Cluster IDs are dynamically assigned. This assignment must be unique - not an easy task in multihop mobile environment, where the hierarchical topology is continuously reconfigured. Second, each cluster can dynamically merge and split, based on the number of nodes in the cluster. This feature causes frequent changes of cluster head, degrading the network performance significantly. Since the diameter of this cluster is variable, it is also difficult to predict how long it takes to propagate clustering control messages among nodes, and hence difficult to bound the convergence time of the clustering algorithm. Third, the paging and query/response approach used to locate mobile nodes may lead to control message overhead. Fourth, if the location manager leaves the current cluster, this function migrates to another location server. This requires a complex consistency management between the original server and new server.

On-demand routing does not scale well to large populations, as it does not maintain a permanent routing entry to each destination. Instead, as the name suggests, a route is computed only when there is a need. Thus, routing table storage is greatly reduced, if the traffic pattern is sparse. However, on-demand routing introduces the initial search latency, which may degrade the performance of interactive applications (e.g., distributed database queries). Besides in case of large area networks with high node density, the route length may

be large (longer the route, greater the great set-up time). As the route length increases, the probability of route errors also increases.

A recent proposal, which combines on demand routing and conventional routing, is Zone Routing [56,57,58,59]. For routing operation inside the local zone, any routing scheme, including DBF routing or LS routing, can be applied. For interzone routing, on demand routing is used.

The advantage of zone routing is its scalability, as a "global" routing table overhead is limited by its zone size. Yet, the benefits of global routing are preserved within each zone. In spite of all the obvious advantages, problems arise when different routing schemes are combined together.

(a) A node has to support various routing schemes. For e.g. if Zone1 uses AODV, Zone2 uses DSR, when a node moves from Zone1 to Zone2, it has to change its routing schemes.

(b) Even if all nodes where to use the same routing schemes, managers have to be built into each node, to support the concept of varied inter-zone and intra-zone routing protocols, besides the normal unicast, multicast and geocasting protocols. This would be a huge load on laptops and PDAs, forming the mobile nodes.

The **GPS-based Predictive Inter-Zone Routing Protocol (GPIZR)** proposed uses the underlying unicast, multicast and geocast protocol for routing within various zones. Flag bits in various packets and tables indicate whether a route traverses within a single zone or multiple zones.

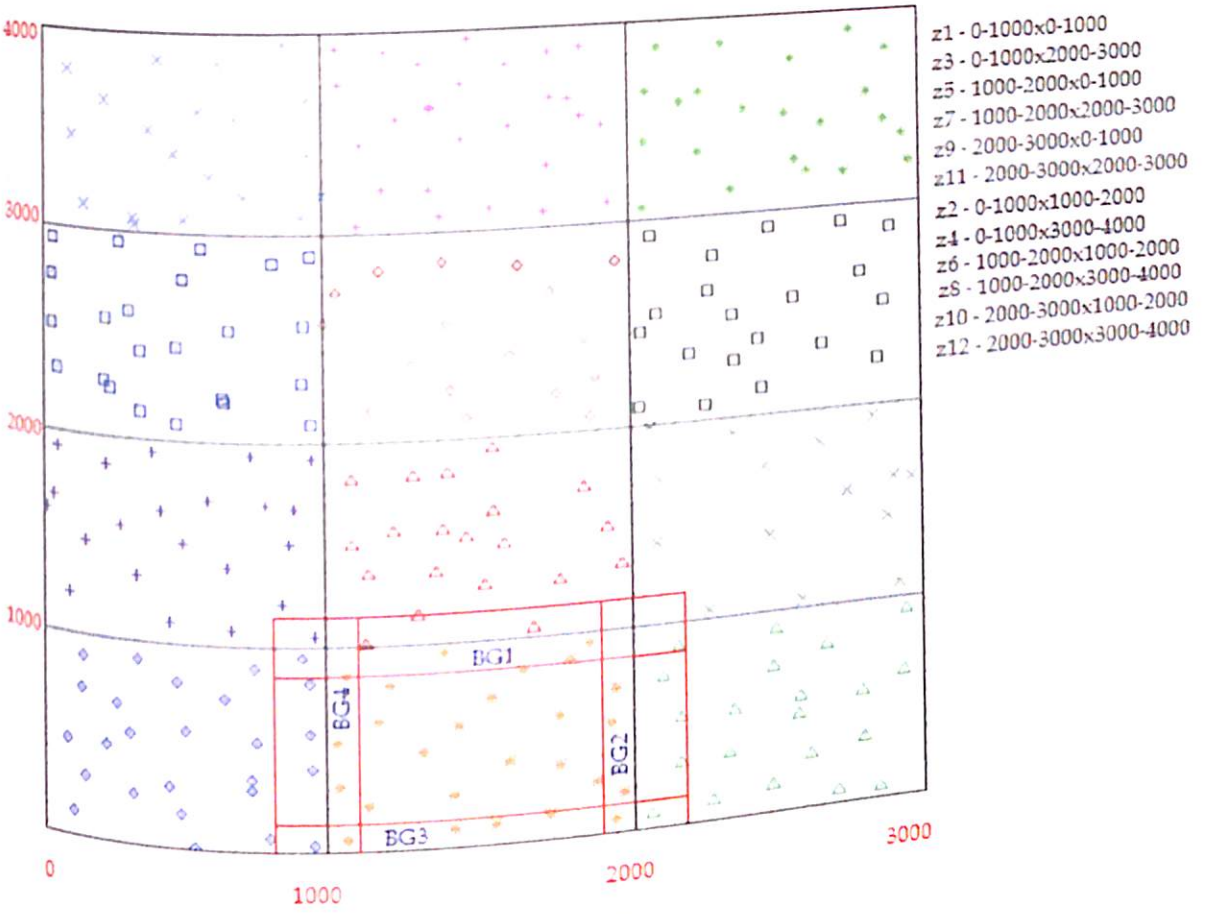


Fig 9.1 Inter-Zone Distribution of nodes

9.2 Overview of the protocol

In case of large areas as the one shown in fig 9.1 (area is 3000 X 4000), the network area is divided into smaller zones (polygons of area 1000 x1000). In fig 9.1, there are totally twelve zones of area 1000 x 1000.

Source – Destinations pairs within the same zone use GPEAR for handling unicast traffic. Multicast or Geocast groups can exist only within a particular zone (except for the geocast group that are on the border). Each zone has a minimum of four-geocast group: Border Group1 (BG1), Border Group 2 (BG2), Border Group 3 (BG3) and Border Group (BG4). Members of BG1, BG2, BG3, and BG4 are the nodes along the north, east, south and west borders of a geographical zone (as shown in fig 9.1 with Zone5). A node can be a member of more than one Geocast Group, as long as they are within the corresponding geographical area (this can be seen in fig 9.1).

Inter-zone routing takes over when a source node in one zone wants to transmit packets to a destination node in some other zone. For e.g. Node SR in Zone 5 wants to send a packet to Node DT in Zone 4. The inter-zone route formation process is shown from figs 9.2 to 9.4.

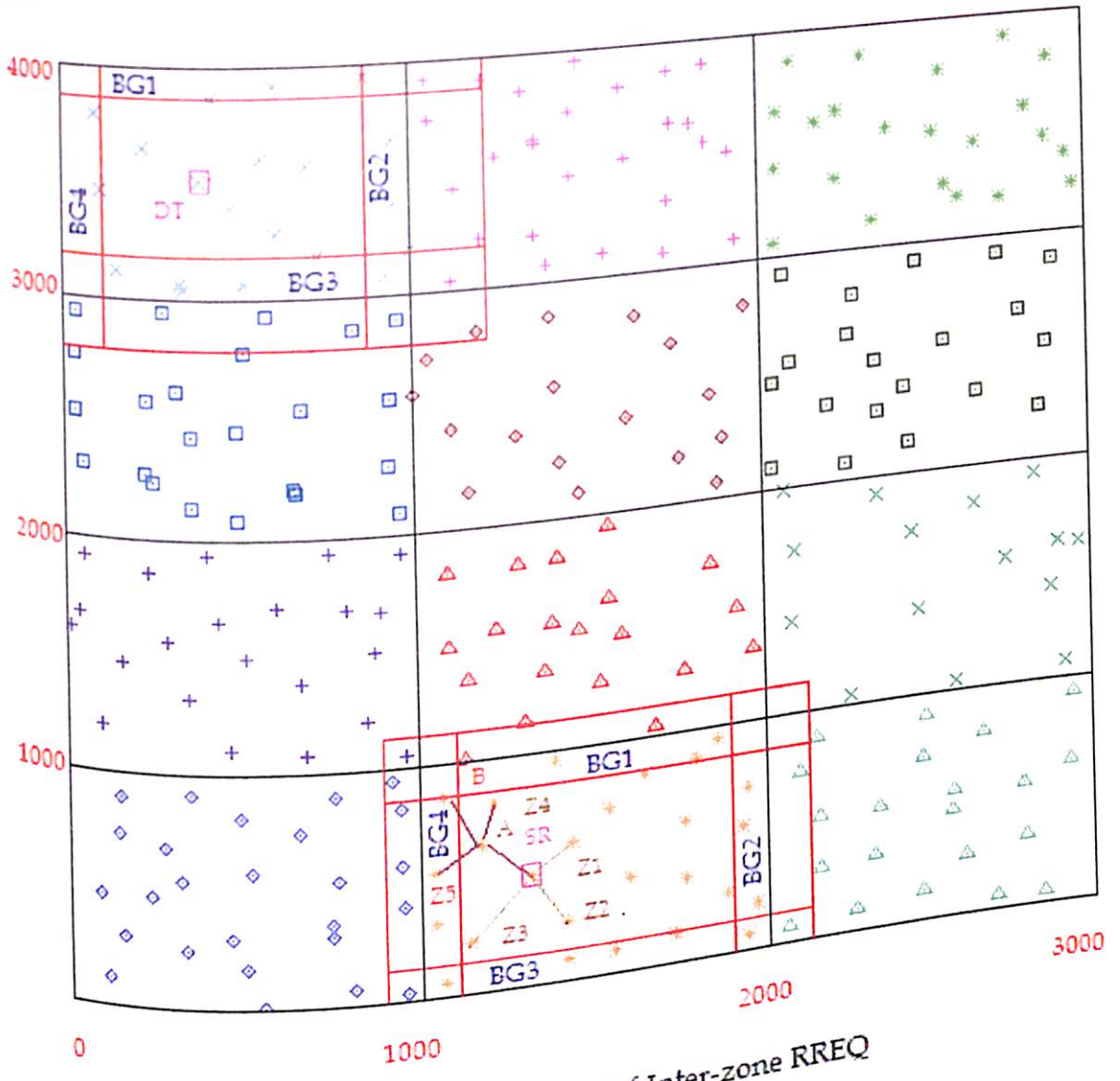


Fig 9.2 Propagation of Inter-zone RREQ

Node SR always starts with the assumption that Node DT is in its zone; hence it tries establishing a route to DT using GPEAR. On failure to establish a route within RREQ_TIMEOUT secs with TTL equal to MAX_TTL, SR assumes DT is in another zone, the protocol for Inter-Zone routing takes over.

9.2.1 Inter-Zone Route Discovery

Propagation of Inter-Zone Routes

Node SR on failure to establish an intra-zone route to Node DT, geocasts the Route Request to each of its Border Groups (BG1-4) in turn. The Route Request thus propagated is termed as Inter-Zone Route Request (IRREQ). IRREQ is an extension of the normal RREQ used by GPEAR, thus the format is similar.

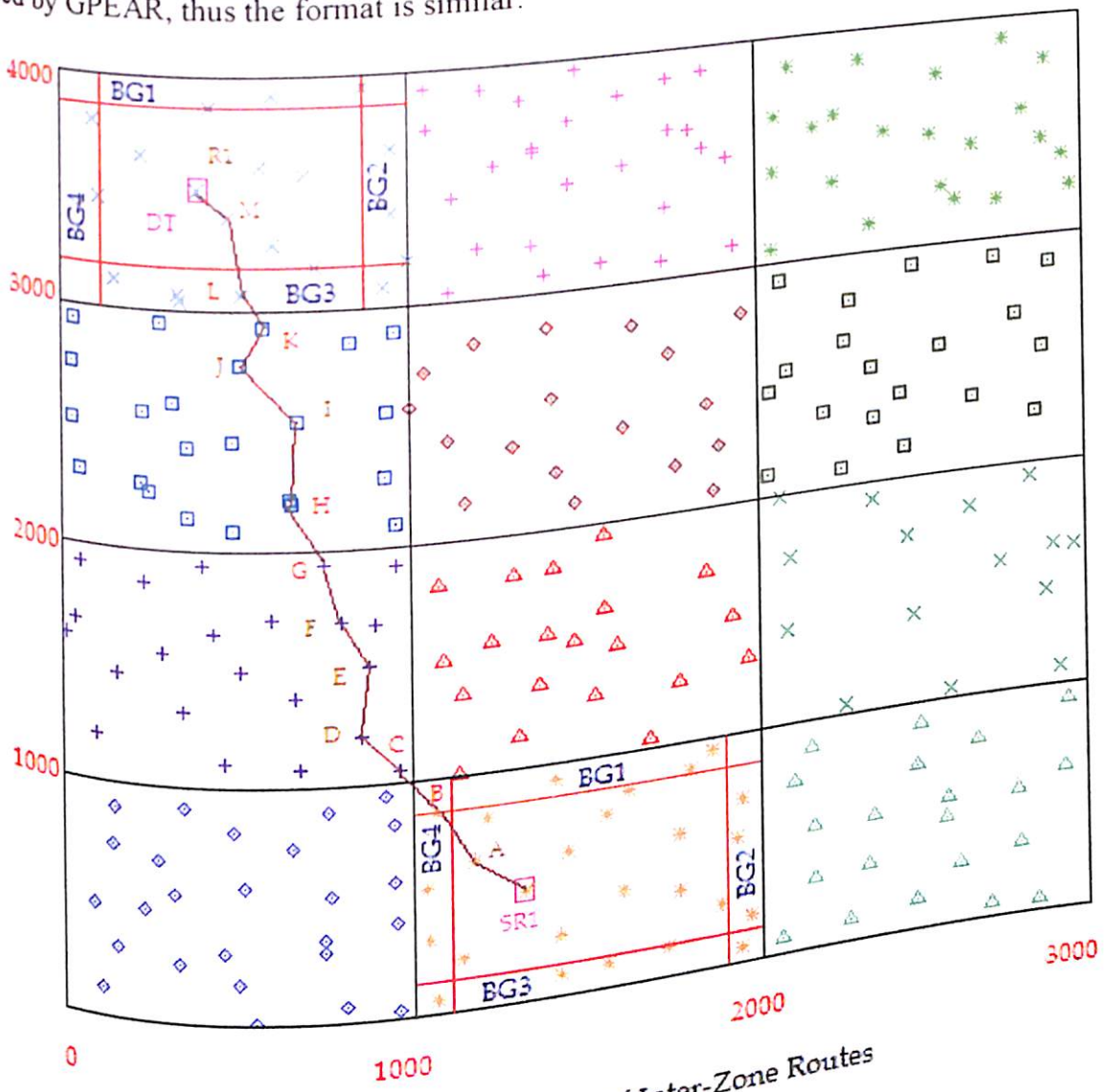


Fig 9.3 Formation of Inter-Zone Routes

It can be observed from fig 9.4, the additional fields in IRREQ are the flag bits, Zone No and the Zone Id. The Z bit indicates the nature of the route. In case of Inter-Zone routes this bit is set. The Destination Id in case of IRREQ propagated by the source

node is that of the Border Groups. Zone No. gives the number of zones through which the route has traversed. Zone Id gives the Id of the zone through which the packet is currently traveling.

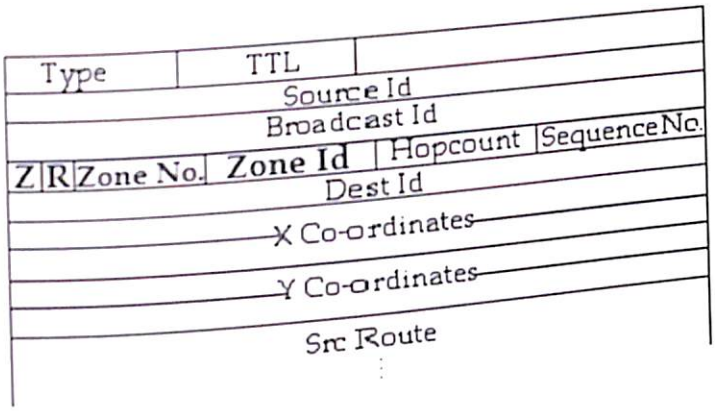


Fig 9.4 Format of IREQ

In case of nodes supporting inter-zone routing, all caching structures are subdivided into two groups - Inter - Zone and Intra - Zone structures. Instead of using the same caching structure with separate spaces, separate caching structures have been provided; the advantages being:

- As the size of the caching structures increases, the time taken to process its contents increases. Both GPEAR and GPIZR store every disjoint route cached, consequently the cache size will be quite large.
- Caches already have a number of flags that are used for multicast and geocast routing.

Hence without increasing the amount of memory taken up by routing structures, separate caches are maintained for inter-zone and intra-zone routes. The source node scans the intra-zone cache for the presence of a route, and then scans the inter-zone cache. If a route is not present then the route discovery process takes over.

On receiving the IRREQ any node, which is part of the geocast group (for e.g. BG4); forwards the IRREQ message upstream along the geocast tree. The responsibility of propagating an IRREQ further into another zone is left to the LEAF NODE alone. This is to prevent multiple copies of IRREQ propagating in a zone resulting in

BROADCAST STORMS. As the structure of a Geocast tree changes with node movement, the leaf node changes with the tree structure, hence a single node is not over-burdened.

In case of the scenario depicted in fig 9.3, the packet reaches Node B that is a member of BG4 via Node A. Node B forwards the packet up the geocast tree to the Node C, which is the leaf node. Node C checks its caching structures for a route to DT. In case a route is not present in both inter-zone as well as intra-zone routing structure, it attempts intra-zone route discovery. If the attempt is unsuccessful GPIZR is employed.

As is the case of GPEAR an aggressive caching scheme is also followed in case of GPIZR. Hence routes that are a part of Route Request as well as Route Reply will also be cached. The strategy used for caching an inter-zone route is different in the sense that the entire route is not cached. This is done to conserve memory space. Node B only caches the route within its zone. Hence the route SR-A-B is stored. Node C which is in a different zone as compared to the source Node SR, stores the route as the route as SR-B-C, since B already has the route to A. If C sends any packet to A, B just modifies the source route on the header and forwards the packet to A.

Since Node C and Node SR are in different zones, Node C updates the flag bits and Zone Id, increments the Zone Count, and then propagates the IRREQ for Node DT into its own zone. Since Node C and Node DT are in different geographical zones, Node C will not be able to find a route to DT in its own zone. Hence Node C geocasts the IRREQ to its Border Groups (BG1, BG2, BG3, BG4). The route formed within this zone will be C-D-E-F-G, where G is the leaf node. Since Node G is in the same zone as Node C, Node G cannot propagate the IRREQ into another zone. So it forwards the IRREQ message along a single downstream path with the R-bit set. The downstream neighbor is decided using position information. The R-bit being set, is an indication that the leaf node is handing over the responsibility of establishing a route to one of its downstream neighbors. In this case Node H is given the responsibility of propagating the IRREQ further into another zone.

The route cached by Node H will be SR-B-C-G. Node H updates the flag bits and Zone Id, increments the Zone count, and propagates the IRREQ into the new zone. The IRREQ is forwarded in the same manner into the next zone till it reaches the Node L, which caches the route as SR-B-C-G-H-K. Node L forwards the IRREQ into the destination Node DT's zone. When the IRREQ reaches DT. DT responds with an Inter-Zone Route Reply (IRREP). DT first caches the path obtained through the process of IRREQ. The route cached will be SR-K-L-M. DT appends itself to the route, reverses the path and sends out the IRREP. The format of the IRREP is as shown in fig 9.5

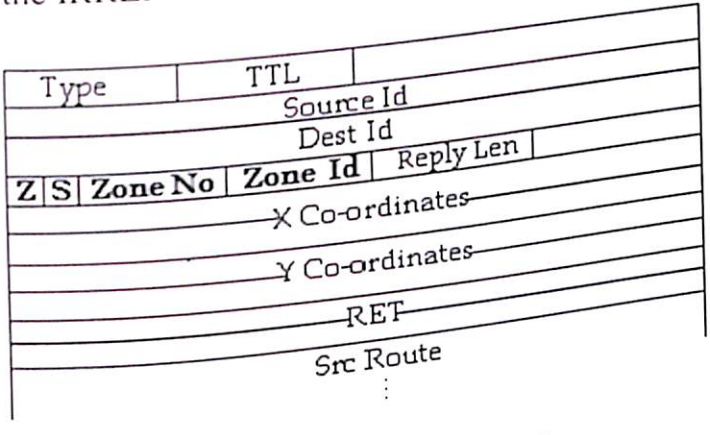


Fig 9.5 Format of IRREP

It can be observed from fig 9.5 that the format of IRREP is same as that of RREP of GPEAR, except for certain fields (flags). The additional flag bit (Z), Zone No and Zone Id fields serve the same purpose as in the case of IRREQ messages. The S flag bit is used for route maintenance.

The Z-bit is set, indicating an interzone route; a value of 4 is placed in the Zone No field. This indicates that the route passes through four different zones (this includes the source and destination zones). The destination zone's Id is placed in the Zone Id field. The source route used for unicasting the IRREP packet is DT-M-L-K–SR. The source route used within a zone carries only the Ids of nodes that are present in that particular zone and the Id of the contact node in the next zone (in this case Node K).

As is the case of GPEAR, as the packet is forwarded along the network, the caching structures at each node are updated. When the IRREP packet reaches Node K, the source

route is modified as DT-L-K-J-I-H-G-SR. The change in source route will be effected every time the zone through which the packet is forwarded, is changed. This new source route is retrieved from the reverse route that was cached when the IRREQ was forwarded along this zone. This is similar to Forward Path Setup scheme used by AODV [24]. Node K updates the source route and Zone Id fields and forwards the packet.

The current source route is used until the packet reaches Node G. Node G modifies the source route as DT-H-G-F-E-D-C-B-SR and forwards the packet into its zone. When the packet is received by Node B, the source route is modified to DT-C-B-A-SR. This is the route that will be finally cached by Node SR.

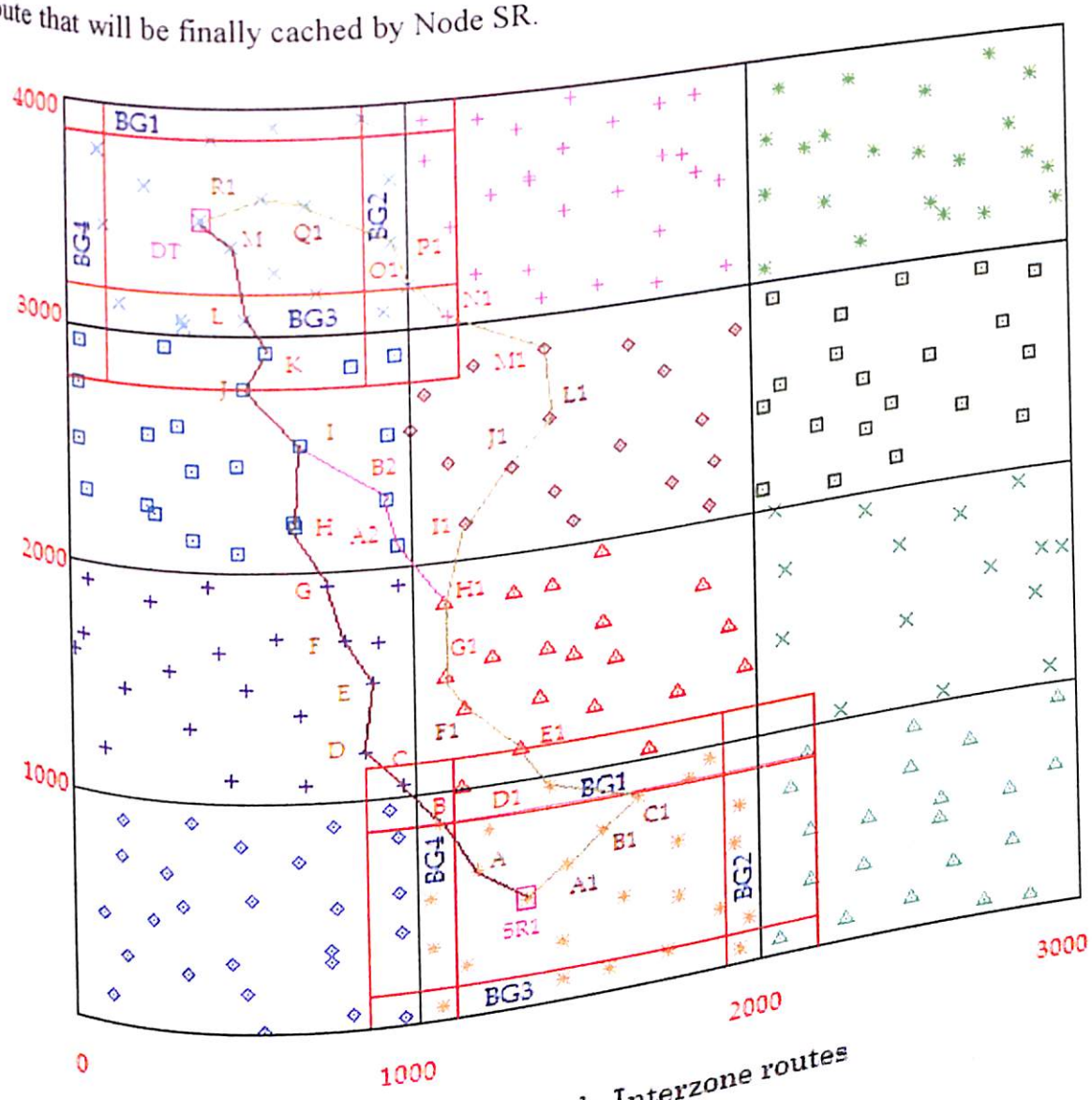


Fig 9.6 Multiple Interzone routes

From fig 9.6, it can be observed that the multiple inter-zones can be cache to a particular destination. The route used for unicasting a packet to the destination is selected using the following criteria:

- Number of zones through which a route traverses (Minimum)
- Link Expiry Time/ Minimum number of hops (to choose between two routes which have the same zone count)

9.2.2 Packet Transmission

When the source node (SR) needs to send a packet to a destination node (DT), it uses the freshly cached route available in its inter-zone primary route cache. The Data Packet Header is modified slightly, to include additional fields that are used for inter-zone routing.

The additional fields introduced are:

- The Z flag-bit to indicate the type of route.
- Zone count Field, this is decremented as it passes thro' different zones.
- Zone Id Field, which gives the Id of the zone thro' which the packet is currently being forwarded.

The source (SR) uses the route SR-A-B-C-DT to transmit the packet. When the packet reaches each intermediate node, it examines the Zone count and Zone Id fields. If the Zone count is greater than 1, then it indicates that the destination node is another zone. The intermediate node then examines the Zone Id field, if the Zone Id is same as its own, it forwards the packet without any modification to the source header.

When the packet reaches Node C, it modifies the source route on the header to SR-B-C-D-E-F-G-H-DT, using the route that it had cached. Node C can use any intra-zone route to Node H; if a shorter route is available other than that route which has been used. Node C also updates the Zone Id and the Zone Count fields, before forwarding the packet into its zone.

Node H modifies the source route when the packet enters the next zone. The source route now used is SR-G-H-I-J-K-L-DT. The final modification of the source route occurs when the packet enters the destination zone, the source route used is SR-K-L-M-DT.

9.2.3 Inter-Zone Route Maintenance

GPIZR, like GPEAR uses regular update (IUPDATE) messages for route maintenance. The messages are originated at the destination and sent along the currently active route to the source. The format of the IUPDATE message is shown in fig 9.7

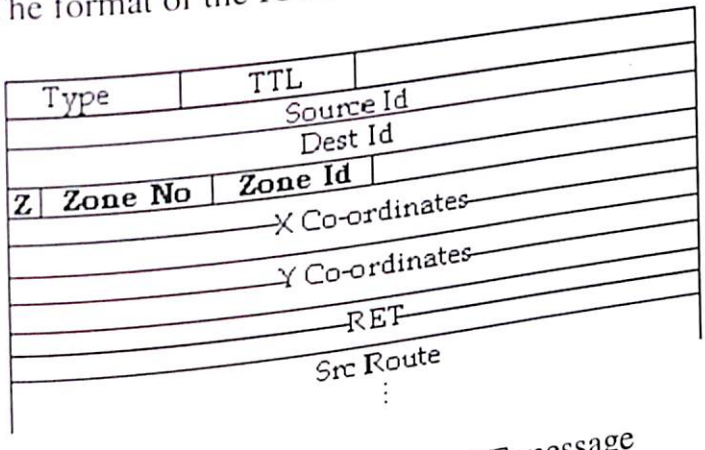


Fig 9.7 Format of IUPDATE message

The RUPDATE message used in GPEAR for route maintenance has been modified to support inter-zone routes. The additional fields introduced in an IUPDATE message are the Z flag, Zone Count and Zone Id.

The response of a node to an IUPDATE message is similar to its response to an RUPDATE message. When a node receives an IUPDATE message it updates the position information stored in its neighbor table; calculates the LET using the position information. If the LET calculated is greater than the time constant T_OPT then the node just forwards the IUPDATE packet after updating the RET.

If the LET is lesser than T_OPT then corrective action has to be employed. The corrective mechanism used is similar to the one employed by GPEAR. A node first attempts local repair, and if the local repair is successful, then the node sends an unsolicited IRREP, with the S-bit is set. The destination of the unsolicited-IRREP can be in the same zone or a

different zone. For e.g. if Node J finds that the LET of its link with Node H is lesser than T_OPT it attempts local repair. If successful then Node H sends an unsolicited IRREP with the S-bit set to both Node SR and Node DT. Node H on receiving the IRREP updates its caching structures based on the currently available route, and stops further propagation of the IRREP, because the information is local to the zone. Similarly when Node K receives the IRREP, it updates its caching structures and prevents further propagation of the IRREP.

If the faulty link is inter-zone in nature (for e.g. K-L link), the action taken is different. Node K attempts inter-zone repair by geocasting an IRREQ for Node L. If the repair is successful, then an unsolicited IRREP is sent in the direction of the source and destination nodes. The repaired source route that is sent towards the direction of the destination node will be modified further by Node L to have the source route between nodes L and DT. The repaired source route sent in the direction of the source will carry the source route between nodes L and G. Further propagation of this IRREP will be halted at Node H.

If the local repair mechanisms fail, the source node will re-initiate a route discovery process. This is possible since the propagation of IUPDATE will continue till the packet reaches the source node.

In case a node shuts down or all attempt at repair fails, then the Route Error process takes over. When a node detects a complete failure on one of its link, it removes any route that has the faulty link from its cache, and broadcasts an intra-zone RERR message in its zone, and then it unicasts an Inter-zone Route Error (IRERR) message in the direction of the source and destination nodes. The format of the IRERR message is same as that of the RERR message of GPEAR, except for the additional fields that are used for inter-zone routing. (Fig 9.8)

The RERR message of GPEAR carries the last known co-ordinates of the destination node. The intermediate nodes use this location information to limit the scope of the route re-discovery process.

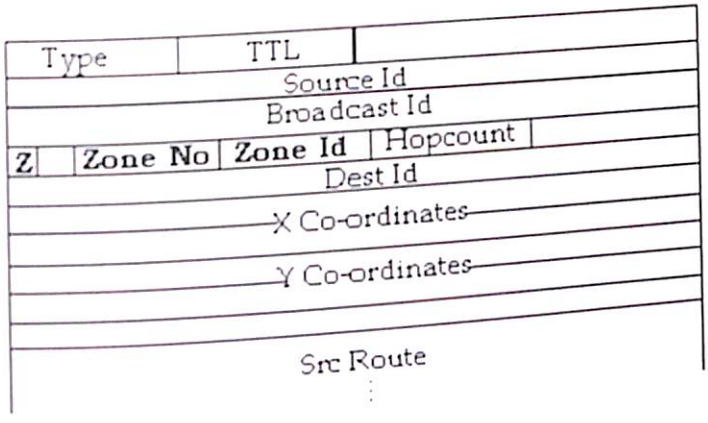


Fig 9.8 Format of IRERR message

The IRERR message does not carry the destination's co-ordinates for it will be of little use to the source. The source has to obtain the route using the help of its border group, irrespective of the destinations position in its zone.

9.3 Inter-Zone Caching and Routing Structures

GPIZR and GPEAR use similar route discovery and route maintenance mechanisms. Therefore GPIZR will require all the Caching and Routing structures employed by the GPEAR protocol.

9.3.1 Route Cache

GPIZR like its intra-zone counterpart has two route caches; the primary and secondary cache. The structure of these caches is similar to that of GPEAR. The caching policy of recording disjoint routes is confined to the local zone. The reasons that a disjoint cache policy is not applied to the entire inter-zone route are:

- A node at all times is not aware of the complete route that traverses through multiple zones.
- Even if the node was aware of the complete route, it is possible that there are only a few members in the geocast border group. If only disjoint nodes are to be cached, then only a single route may be available to the destination.

9.3.2 Neighbor Table

The neighbor table stores position and LET information of a node's neighbors. An inter-zone Neighbor table can exist only in a member of geocast border groups (fig 9.2). The format and the use of the inter-zone neighbor table is similar to that of GPEAR's Neighbor Table.

9.3.3 Energy Table, Srroute Table

GPIZR does not have a separate inter-zone energy table. To the physical layer, the significance of the Energy Table remains the same, whether the neighboring node is in the same zone or a different zone.

The Srroute Table is also not available in GPIZR, as it does not use ring search for obtaining an inter-zone route.

9.3.4 Dest Table, Request Table

GPIZR has separate inter-zone Dest and Request Tables. The Request Table is used to maintain information regarding IRREQs that have be originated or forwarded by the node. The format and function of the inter-zone request table is similar to that of GPEAR's [chapter 4].

The Dest Table is used by GPIZR to keep track of routes along which IUPDATE messages have to be sent.

9.4 Interaction between various protocols

GPEAR and GPIZR

When a node desires to establish a connection with a destination, it checks its intra-zone cache for a route. If a route is not available then it checks its inter-zone cache. If no previously cached route is available either in the intra-zone or the inter-zone cache then

GPEAR's route discovery mechanism takes over. If GPEAR fails to find a route within a certain time period (RREQ_TIMEOUT) then GPIZR takes over. Hence GPIZR takes over when GPEAR fails. This is the case when a source node is not aware of the zone in which the destination node is present. If the destination node's zone is known, either GPEAR or GPIZR will take over depending on the destination zone.

Any broadcast messages sent by GPEAR are restricted to the zone. All control messages broadcasted by GPEAR always carry position information of the node forwarding the message, when a node receives the broadcast message, it checks whether the position of the node forwarding the GPEAR packet is within its zone. If it is not, the forwarding of packets is halted. This ensures that any broadcast is local to a zone. If by any probability a RREQ broadcast reaches a node in a different zone and it has a route (inter-zone or intra-zone) it can send an unsolicited IRREP with the Z-bit set and then halt further broadcast of the RREQ.

Unicasting of data packets or control packets by GPEAR is completely independent of GPIZR and is always restricted within a zone.

GGPEAR and GPIZR

GGPEAR is an integral part of GPIZR route discovery mechanism. Any zone has a minimum of four geocast groups and these are the Border Groups. Other than these geocast groups, other Geocast groups can exist within any zone. In case any connection has to be established between two geocast groups which are in different zones, then the leaf node of the source geocast group has to set-up an inter-zone route with a member of the destination geocast group using GPIZR. When any member of the source geocast group wants to send a packet, it uses the cached inter-zone route; the Z-bit in the packet header is set to indicate an interzone route. The packet will then be forwarded to the destination geocast group, using the unicast route established.

Geocast groups can exist across zones, as long as the geographical zones are restricted. In fact the border groups themselves span more than one zone (fig 9.2). GGPEAR is employed within these zones and is independent of GPIZR.

If two members of a border group want to set up a unicast link then they have to use either GPEAR if they are in the same zone, or they have to use GPIZR.

MGPEAR and GPIZR

Unlike their geocast counterparts multicast groups can exist only within a zone. Any broadcasts by MGPEAR will carry position information, as is the case of GPEAR. Hence when these broadcasts reach a node that is in a different zone, the broadcasts will be halted.

If two multicast groups, that are in different zones, wish to communicate, then an inter-zone unicast route has to be set-up between these two multicast groups using GPIZR. This mechanism can be used for setting-up inter-zone multicast groups. Different Multicast sub-groups can be set-up within individual zones and they can be connected using inter-zone routes provided by GPIZR.

9.5 Theoretical Analysis of GPIZR

GPIZR is used for improving scalability in MANETS. GPIZR's main advantage is that it does not require any additional processing to support inter-zone routing. It uses the underlying unicast, multicast and geocast protocols' mechanism. The interaction between the protocols is seamless since all of them use similar mechanisms for route discovery and route maintenance. All the protocols use position information and the mechanisms of all these protocols are centered on the available position information. Hence they can work in tandem with each other.

GPIZR cannot be used with other available protocols such as DSR or AODV. Though DSR uses source routing GPIZR will not be able to interact with it, as position information is vital for its functioning. Even if position information was available, GPIZR has a pro-active route maintenance mechanism that is not supported by DSR. As position information is not available with DSR's control packets, there is no mechanism to limit broadcast messages to a particular zone.

GPIZR, MGPEAR or GGPEAR can be used only in a network that uses GPEAR as its unicast routing protocol.

Chapter 10 - CONCLUSION

An adhoc mobile network is a collection of nodes, each of which is capable of and is likely to be moving, resulting in continual changes in the topology of the network. These nodes communicate through wireless transmission, and each of them serves as a router for the other network nodes. Adhoc networks have unique characteristics that make network communication challenging. The mobility of nodes introduces the problem of discovering and maintaining paths over a dynamic network topology. Additionally, due to their mobile nature they often run out of battery and hence are power constrained. Furthermore, because of the wireless transmission, the network has limited bandwidth, and there are often high error rates. Because of these limitations, protocols designed for providing communication in wired networks are not suitable for wireless networks. Adhoc routing must be designed, with these limitations in mind, and hence must aim to minimize processing and transmission overhead, in addition to being able to find and maintain routes over a dynamic topology.

To serve this end GPS-based Predictive Energy Aware Routing protocol (GPEAR) has been designed. GPEAR provides routing in an on-demand reactive manner. Routes are established only when they are needed by the source node. As long as the source node requires the route, GPEAR maintains it. Unlike any other reactive protocol for MANET's GPEAR is predictive in nature, and is able to successfully detect link breakages in advance and set-up new routes to the destination thereby reducing control overhead in terms of error messages that are broadcasted around the network. GPEAR also conserves battery power by transmitting data packets with minimum possible energy, based on the distance between adjoining nodes. GPEAR protocol simulation studies have also led to the development of routing criteria for varying mobility conditions.

The operation of GPEAR protocol has been validated through extensive simulations. Simulation of GPEAR's unicast communication capability has been completed in a variety of network sizes and mobility rates. GPEAR efficiently establishes routes with minimum control overhead and minimum energy consumption. Currently, GPEAR is able to determine routes with bandwidth constraints and high mobility conditions. GPEAR's performance under high mobility conditions is better than benchmark protocols such as DSR and AODV.

The development of GPEAR protocol has led to the following:

- Prediction of link breakage
- Reduction of Control Overhead for high mobility scenarios.
- Reduction in Energy Consumption
- Satisfactory performance even under high mobility conditions (PDR, Energy Consumption etc.)
- Development of an alternate routing criterion for high mobility conditions.

In order to supplement GPEAR for varied traffic, multicast (MGPEAR) and geocast (GGPEAR) protocols have been developed. A generalized multicast protocol, which can be implemented on any underlying source-routing protocol, has also been developed; this protocol was the initial step for developing MGPEAR and GGPEAR, but it can also be used with DSR.

Scalability analysis carried out on GPEAR indicates a drop in various performance metrics. The GPS based Inter-Zone Routing protocol (GIZR) was developed to overcome the limitations of network size.

There is scope for further analysis being carried out in this area.

Chapter 11 – Future Scope of Research

A complete set of routing protocols for Mobile Adhoc Networks has been developed as part of this thesis. The unicast protocol- GPS based Predictive Energy Aware Routing (GPEAR) has implemented and analyzed in detail. A similar simulation based analysis of the multicast, geocast and inter-zone routing protocols has been envisaged as future work.

Another possible area of future research is defining the operation of wired or Mobile IP enabled networks. If one or more nodes are within range of wireless access point, either a network router or a foreign agent. The cloud of adhoc nodes should be able to take advantage of this connection to reach the Internet. This requires both the nodes being able to discover the existence of the access point, and determination of the addressing scheme whereby the nodes are reachable by the rest of the Internet.

The interaction between a routing protocol and other network parameters plays an important role in determining the performance of the protocol. It was shown that the node density and average nodal transmission range are key factors affecting network throughput and control overhead generated by the routing protocol.

There are numerous research problems remaining in the area of mobile networking. One of these is security. Wireless transmission is inherently insecure. Packets can be received by anyone. A method for the encryption of packet transmissions is needed so that users can have secure remote access to their files and transmit data across the channels. Additionally, adhoc routing protocols are particularly susceptible to denial of service attacks and the introduction of false routing information by malicious nodes. The prevention of each of these occurrences is crucial for preserving the integrity of the network.

Another diverse area of research work could be the real-time implementation and testing of the protocols suggested as a part of the thesis. Simulations are a valuable tool for

learning and comparing wireless protocols and techniques, but as Rodney Brooks observed "simulations are doomed to succeed". That is, it is always possible to find the right protocol tweaks and hacks that work well in a particular simulation environment. However, real-world systems face problems that do not occur in simulations. Unlike simulator experiments, test-bed experiments cannot be perfectly reproduced. Interference and radio propagation change between each experiment, and for all practical purposes, are out of the experimenter's control. Hence implementation of the protocols on a real-time adhoc test-bed could bring out various new facets of the protocols.

Adhoc networking is currently one of the most rapidly growing research areas. This thesis has made the contribution of routing protocols for providing connectivity within adhoc networks of mobile nodes. While the foundation for providing this connectivity has been laid, more research needs to be carried out to define a complete and comprehensive solution for mobile adhoc networking.

APPENDIX –A

Programs used for simulation studies

This Appendix has a set of sample programs and outputs that were a part of the simulation processes of GPEAR.The entire simulation process for validation of GPEAR from the generation of network scenario and traffic to the extraction of results has been sketched out in this appendix.

Scenario and Traffic Generation

Scenario Generation

Normally for large topologies, the node movement and traffic connection patterns are defined in separate files for convenience. A set of VC++ programs: **setdest.cpp, rng.cpp,** where used for generating the network scenario. Setdest.cpp generates the initial network scenario and keeps track of each node movement. The initial position the direction of movement and the speed of each node are generated randomly using the

program rng.cpp

SETDEST.H

```
#ifndef __setdest_h__
#define __setdest_h__

#ifndef LIST_FIRST
#define LIST_FIRST(head)        ((head)->lh_first)
#endif
#ifndef LIST_NEXT
#define LIST_NEXT(elm, field)((elm)->field.le_next)
#endif

void ReadInMovementPattern(void);

class vector {
public:
        vector(double x = 0.0, double y = 0.0, double z = 0.0) {
                X = x; Y = y; Z = z;
        }
```

```
        double length() {
                return sqrt(X*X + Y*Y + Z*Z);
        }

        inline void vector::operator=(const vector a) {
                X = a.X;
                Y = a.Y;
                Z = a.Z;
        }
        inline void vector::operator+=(const vector a) {
                X += a.X;
                Y += a.Y;
                Z += a.Z;
        }
        inline int vector::operator==(const vector a) {
                return (X == a.X && Y == a.Y && Z == a.Z);
        }
        inline int vector::operator!=(const vector a) {
                return (X != a.X || Y != a.Y || Z != a.Z);
        }
        inline vector operator-(const vector a) {
                return vector(X-a.X, Y-a.Y, Z-a.Z);
        }
        friend inline vector operator*(const double a, const vector b) {
                return vector(a*b.X, a*b.Y, a*b.Z);
        }
        friend inline vector operator/(const vector a, const double b) {
                return vector(a.X/b, a.Y/b, a.Z/b);
        }

        double X;
        double Y;
};      double Z;


class Neighbor {
public:
        u_int32_t       index;                  // index into NodeList
        u_int32_t       reachable;              // != 0 --> reachable.
        double          time_transition; // next change
};

struct setdest {
 double time;
 double X, Y, Z;
 double speed;
 LIST_ENTRY(setdest)   traj;
};
```

```
class Node {
 friend void ReadInMovementPattern(void);
public:
        Node(void);
        void    Update(void);
        void    UpdateNeighbors(void);
        void    Dump(void);

        double          time_arrival;           // time of arrival at dest
        double          time_transition; // min of all neighbor times


        // # of optimal route changes for this node
        int             route_changes;
        int         link_changes;

private:
        void    RandomPosition(void);
        void    RandomDestination(void);
        void    RandomSpeed(void);

        u_int32_t       index;          // unique node identifier

        vector          position;       // current position
        vector          destination;    // destination
        vector          direction;      // computed from pos and dest

        double          speed;
        double          time_update;    // when pos last updated

        static u_int32_t NodeIndex;

        LIST_HEAD(traj, setdest) traj;
public:
        // An array of NODES neighbors.
};      Neighbor        *neighbor;
#endif /* __setdest_h__ */
```

SETDEST.CPP

```
#include <io.h>
#include <assert.h>
#include <fcntl.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <iostream.h>
#include <fstream.h>

#if !defined(sun)
#include <errno.h>
#endif

#include "../../../rng1.h"
#include "../../../rng1.cpp"

#include "setdest1.h"

#define         SANITY_CHECKS

#define GOD_FORMAT      "$ns_ at %.12f \"$god_ set-dist %d %d %d\"\n"
#define GOD_FORMAT2     "$god_ set-dist %d %d %d\n"
#define NODE_FORMAT     "$ns_ at %.12f \"$node_(%d) setdest %.12f %.12f %.12f\"\n"
#define NODE_FORMAT2    "$node_(%d) setdest %.12f %.12f %.12f\n"
#define NODE_FORMAT3    "$node_(%d) set %c_ %.12f\n"

#define
#define         INFINITY        0x00ffffff
#define         min(x,y)        ((x) < (y) ? (x) : (y))
#define         max(x,y)        ((x) > (y) ? (x) : (y))
#define         ROUND_ERROR     1e-9

static int count = 0;
/*=============================================================================
  Function Prototypes
=============================================================================*/
void
void            usage(char**);
double          init(void);
void            uniform(void);
void            dumpall(void);
void            ComputeW(void);
void            floyd_warshall(void);
void            show_diffs(void);
void            show_routes(void);
void            show_counters(void);
```

```
/*==========================================================================
  Global Variables
==========================================================================*/
const double    RANGE = 250.0;          // transmitter range in meters
double          TIME = 0.0;             // my clock;
double          MAXTIME = 0.0;          // duration of simulation
double          MAXX = 0.0;
double          MAXY = 0.0;
double          MAXSPEED = 0.0;
double          PAUSE = 0.0;
u_int32_t       NODES = 0;
u_int32_t       RouteChangeCount = 0;
u_int32_t   LinkChangeCount = 0;
u_int32_t       DestUnreachableCount = 0;
Node            *NodeList = 0;
u_int32_t       *D1 = 0;
u_int32_t       *D2 = 0;

/*==========================================================================
  Random Number Generation
==========================================================================*/
#define M               2147483647L
#define INVERSE_M       ((double)4.656612875e-10)

char random_state[32];
RNG *rng;

double
uniform()
{
    count++;
    return rng->uniform_double() ;
}

/*==========================================================================
  Misc Functions...
==========================================================================*/
void
usage(char **argv)
{
        fprintf(stderr,
                "\nusage: %s\t-n <nodes> -p <pause time> -s <max speed>\n",
                argv[0]);
        fprintf(stderr,
                "\t\t-t <simulation time> -x <max X> -y <max Y>\n\n");
}


void
init()
{
```

```
/* Initialized the Random Number Generation*/
        rng = new RNG;
rng->set_seed(RNG::HEURISTIC_SEED_SOURCE);

init();
while(TIME <= MAXTIME) {
        double nexttime = 0.0;
        u_int32_t i;

        for(i = 0; i < NODES; i++) {
                NodeList[i].Update();
        }

        for(i = 0; i < NODES; i++) {
                NodeList[i].UpdateNeighbors();
        }

        for(i = 0; i < NODES; i++) {
                Node *n = &NodeList[i];

                if(n->time_transition > 0.0) {
                        if(nexttime == 0.0)
                                nexttime = n->time_transition;
                        else
                                nexttime = min(nexttime, n->time_transition);
                }

                if(n->time_arrival > 0.0) {
                        if(nexttime == 0.0)
                                nexttime = n->time_arrival;
                        else
                                nexttime = min(nexttime, n->time_arrival);
                                }
                }

        floyd_warshall();
#ifdef DEBUG
#endif          show_routes();


        show_diffs();
#ifdef DEBUG
#endif          dumpall();


        assert(nexttime > TIME + ROUND_ERROR);
        TIME = nexttime;
}

show_counters();
```

```
    int of;
    if ((of = open(".rand_state",O_WRONLY | O_TRUNC | O_CREAT, 0777)) < 0) {
     fprintf(stderr, "open rand state\n");
     exit(-1);
     }
    for (unsigned int i = 0; i < sizeof(random_state); i++)
  random_state[i] = 0xff & (int) (uniform() * 256);
    if (write(of,random_state, sizeof(random_state))< 0) {
     fprintf(stderr, "writing rand state\n");
     exit(-1);
     }
    close(of);
}

/*==========================================================================
  Node Class Functions
==========================================================================*/
u_int32_t Node::NodeIndex = 0;

Node::Node()
{
    u_int32_t i;

    index = NodeIndex++;

    //if(index == 0)
    //        return;

  route_changes = 0;
link_changes = 0;

/* For the first PAUSE seconds of the simulation, all nodes are stationary. */
    time_arrival = TIME + PAUSE;
    time_update = TIME;
    time_transition = 0.0;

    position.X = position.Y = position.Z = 0.0;
    destination.X = destination.Y = destination.Z = 0.0;
    direction.X = direction.Y = direction.Z = 0.0;
    speed = 0.0;

    RandomPosition();

    fprintf(stdout, NODE_FORMAT3, index, 'X', position.X);
    fprintf(stdout, NODE_FORMAT3, index, 'Y', position.Y);
    fprintf(stdout, NODE_FORMAT3, index, 'Z', position.Z);

    neighbor = new Neighbor[NODES];
    if(neighbor == 0) {
            perror("new");
```

```
        exit(1);
    }

    for(i = 0; i < NODES; i++) {
        neighbor[i].index = i;
        neighbor[i].reachable = (index == i) ? 1 : 0;
        neighbor[i].time_transition = 0.0;
    }       }
void
Node::RandomPosition()
{
    position.X = uniform() * MAXX;
    position.Y = uniform() * MAXY;
}       position.Z = 0.0;

void
Node::RandomDestination()
{
    destination.X = uniform() * MAXX;
    destination.Y = uniform() * MAXY;
        destination.Z = 0.0;
}       assert(destination != position);

void
Node::RandomSpeed()
{
    speed = uniform() * MAXSPEED;
}       assert(speed != 0.0);

void
Node::Update()
{
    position += (speed * (TIME - time_update)) * direction;

    if(TIME == time_arrival) {
        vector v;

        if(speed == 0.0 || PAUSE == 0.0) {

        RandomDestination();                                    }
            RandomSpeed();
            v = destination - position;
            direction = v / v.length();
            time_arrival = TIME + v.length() / speed;
        else {
            destination = position;
            speed = 0.0;
```

```
                time_arrival = TIME + PAUSE;
        }

        fprintf(stdout, NODE_FORMAT,
                TIME, index, destination.X, destination.Y, speed);
    }

    time_update = TIME;
    time_transition = 0.0;
}

void
Node::UpdateNeighbors()
{
    static Node *n2;
    static Neighbor *m1, *m2;
    static vector D, B, v1, v2;
    static double a, b, c, t1, t2, Q;
    static u_int32_t i, reachable;

    v1 = speed * direction;

    /* Only need to go from INDEX --> N for each one since links are symmetric. */

    for(i = index+1; i < NODES; i++) {

        m1 = &neighbor[i];
        n2 = &NodeList[i];
        m2 = &n2->neighbor[index];

        assert(i == m1->index);
        assert(m1->index == n2->index);
        assert(index == m2->index);
        assert(m1->reachable == m2->reachable);

        reachable = m1->reachable;

        /* ============================================================
                Determine Reachability
        ============================================================ */
        {       vector d = position - n2->position;

                if(d.length() < RANGE) {
#ifdef SANITY_CHECKS
                        if(TIME > 0.0 && m1->reachable == 0)
                                assert(RANGE - d.length() < ROUND_ERROR);
#endif
                        m1->reachable = m2->reachable = 1;
                }
                // Boundary condition handled below.
```

```
                    else {
#ifdef SANITY_CHECKS
                        if(TIME > 0.0 && m1->reachable == 1)
                            assert(d.length() - RANGE < ROUND_ERROR);
#endif
                        m1->reachable = m2->reachable = 0;
                    }

#ifdef DEBUG
            fprintf(stdout, "# %.6f (%d, %d) %.2fm\n",
                TIME, index, m1->index, d.length());
#endif
            }

            /* ============================================================
                                Determine Next Event Time
              ============================================================ */
            v2 = n2->speed * n2->direction;

            D = v2 - v1;
            B = n2->position - position;

            a = (D.X * D.X) + (D.Y * D.Y) + (D.Z * D.Z);
            b = 2 * ((D.X * B.X) + (D.Y * B.Y) + (D.Z * B.Z));
            c = (B.X * B.X) + (B.Y * B.Y) + (B.Z * B.Z) - (RANGE * RANGE);

            if(a == 0.0) {
                /*
                 *  No Finite Solution
                 */
                m1->time_transition= 0.0;
                m2->time_transition= 0.0;
                goto  next;
            }

            Q = b * b - 4 * a * c;
            if(Q < 0.0) {
/*  No real roots.*/
                m1->time_transition = 0.0;
                m2->time_transition = 0.0;
                goto next;
            }
            Q = sqrt(Q);

            t1 = (-b + Q) / (2 * a);
            t2 = (-b - Q) / (2 * a);

            if(t1 > 0.0 && t1 < ROUND_ERROR) t1 = 0.0;
            if(t1 < 0.0 && -t1 < ROUND_ERROR) t1 = 0.0;
            if(t2 > 0.0 && t2 < ROUND_ERROR) t2 = 0.0;
```

```
        if(t2 < 0.0 && -t2 < ROUND_ERROR) t2 = 0.0;

        if(t1 < 0.0 && t2 < 0.0) {
                m1->time_transition = 0.0;
                m2->time_transition = 0.0;
                goto next;
        }

        /*          Boundary conditions.                    */
        if((t1 == 0.0 && t2 > 0.0) || (t2 == 0.0 && t1 > 0.0)) {
                m1->reachable = m2->reachable = 1;
                m1->time_transition = m2->time_transition = TIME + max(t1, t2);
        }
        else if((t1 == 0.0 && t2 < 0.0) || (t2 == 0.0 && t1 < 0.0)) {
                m1->reachable = m2->reachable = 0;
                m1->time_transition = m2->time_transition = 0.0;
        }                                                   */

        /*              Non-boundary conditions.
        else if(t1 > 0.0 && t2 > 0.0) {
                m1->time_transition = TIME + min(t1, t2);
                m2->time_transition = TIME + min(t1, t2);
        }
        else if(t1 > 0.0) {
                m1->time_transition = TIME + t1;
                m2->time_transition = TIME + t1;
        }
        else {
                m1->time_transition = TIME + t2;
                m2->time_transition = TIME + t2;
        }

        /* ================================================
          Update the transition times for both NODEs.
          ================================================ */
        if(time_transition == 0.0 || (m1->time_transition &&
          time_transition > m1->time_transition)) {
                time_transition = m1->time_transition;
        }
        if(n2->time_transition == 0.0 || (m2->time_transition &&
          n2->time_transition > m2->time_transition)) {
                n2->time_transition = m2->time_transition;
        }
next:
    if(reachable != m1->reachable && TIME > 0.0) {
        LinkChangeCount++;                          }
        link_changes++;                     }
        n2->link_changes++;
```



```
void
Node::Dump()
```

```
{
        Neighbor *m;
        u_int32_t i;

        fprintf(stdout,
                "Node: %d\tpos: (%.2f, %.2f, %.2f) dst: (%.2f, %.2f, %.2f)\n",
                index, position.X, position.Y, position.Z,
                destination.X, destination.Y, destination.Z);
        fprintf(stdout, "\tdir: (%.2f, %.2f, %.2f) speed: %.2f\n",
                direction.X, direction.Y, direction.Z, speed);
        fprintf(stdout, "\tArrival: %.2f, Update: %.2f, Transition: %.2f\n",
                time_arrival, time_update, time_transition);

        for(i = 0; i < NODES; i++) {
                m = &neighbor[i];
                fprintf(stdout, "\tNeighbor: %d (%x), Reachable: %d, Transition Time: %.2f\n",
                        m->index, (int) m, m->reachable, m->time_transition);
        }
}


/*============================================================================
  Dijkstra's Shortest Path Algoritm
============================================================================*/
void dumpall()
{
        u_int32_t i;

        fprintf(stdout, "\nTime: %.2f\n", TIME);

        for(i = 0; i < NODES; i++) {
                NodeList[i].Dump();
        }
}

void
ComputeW()
{
        u_int32_t i, j;
        u_int32_t *W = D2;

        memset(W, '\xff', sizeof(int) * NODES * NODES);

        for(i = 0; i < NODES; i++) {
                for(j = i; j < NODES; j++) {
                        Neighbor *m = &NodeList[i].neighbor[j];
                        if(i == j)
                                W[i*NODES + j] = W[j*NODES + i] = 0;
                        else
```

```
                        W[i*NODES + j] = W[j*NODES + i] = m->reachable ? 1 :
INFINITY;
                }       }       }

void
floyd_warshall()
{
        u_int32_t i, j, k;

        ComputeW();   // the connectivity matrix

        for(i = 0; i < NODES; i++) {
                for(j = 0; j < NODES; j++) {
                        for(k = 0; k < NODES; k++) {
                                D2[j*NODES + k] = min(D2[j*NODES + k], D2[j*NODES + i]
+ D2[i*NODES + k]);
                        }       }       }

#ifdef SANITY_CHECKS
        for(i = 0; i < NODES; i++)
                for(j = 0; j < NODES; j++) {
                        assert(D2[i*NODES + j] == D2[j*NODES + i]);
                        assert(D2[i*NODES + j] <= INFINITY);
                }
#endif
}

/* Write the actual GOD entries to a TCL script. */
void
show_diffs()
{
        u_int32_t i, j;

        for(i = 0; i < NODES; i++) {
                for(j = i + 1; j < NODES; j++) {
                        if(D1[i*NODES + j] != D2[i*NODES + j]) {
                                if(D2[i*NODES + j] == INFINITY)
                                        DestUnreachableCount++;


                        if(TIME > 0.0) {
                                RouteChangeCount++;
                                NodeList[i].route_changes++;
                                NodeList[j].route_changes++;
                        }
                                if(TIME == 0.0) {
                                        fprintf(stdout, GOD_FORMAT2,
                                                i, j, D2[i*NODES + j]);

#ifdef SHOW_SYMMETRIC_PAIRS
```

```
                                        fprintf(stdout, GOD_FORMAT2,
                                                j, i, D2[j*NODES + i]);
#endif
                                }
                                else {
                                        fprintf(stdout, GOD_FORMAT,
                                                TIME, i, j, D2[i*NODES + j]);
#ifdef SHOW_SYMMETRIC_PAIRS
                                        fprintf(stdout, GOD_FORMAT,
                                                TIME, j, i, D2[j*NODES + i]);
#endif
                                }       }       }       }
        memcpy(D1, D2, sizeof(int) * NODES * NODES);
}

void
show_routes()
{
        u_int32_t i, j;

        fprintf(stdout, "#\n# TIME: %.12f\n#\n", TIME);
        for(i = 0; i < NODES; i++) {
                fprintf(stdout, "# %2d) ", i);
                for(j = 0; j < NODES; j++)
                        fprintf(stdout, "%3d ", D2[i*NODES + j] & 0xff);
                fprintf(stdout, "\n");
        }
        fprintf(stdout, "#\n");
}

void
show_counters()
{
        u_int32_t i;

        fprintf(stdout, "#\n# Destination Unreachables: %d\n#\n",
                DestUnreachableCount);
        fprintf(stdout, "# Route Changes: %d\n#\n", RouteChangeCount);
        fprintf(stdout, "# Link Changes: %d\n#\n", LinkChangeCount);
        fprintf(stdout, "# Node | Route Changes | Link Changes\n");
        for(i = 0; i < NODES; i++)
                fprintf(stdout, "# %4d |          %4d |          %4d\n",
                i, NodeList[i].route_changes,
                NodeList[i].link_changes);
        fprintf(stdout, "#\n");
}
```

RNG.H

```
/* New random number generator */

#ifndef _rng_h_
#define _rng_h_

// Define rng_test to build the test harness.
#define rng_test

#include <math.h>
#include <stdlib.h>

class RNGImplementation {
public:
        RNGImplementation(long seed = 1L) {
                seed_ = seed;
        };
        void set_seed(long seed) { seed_ = seed; }
        long seed() { return seed_; }
        long next();                    // return the next one
        double next_double();
private:
        long seed_;
};

/* Use class RNG in real programs. */

class RNG

        {
public:
        enum RNGSources { RAW_SEED_SOURCE, PREDEF_SEED_SOURCE,
HEURISTIC_SEED_SOURCE };

        RNG() : stream_(1L) {};
        RNG(RNGSources source, int seed = 1) { set_seed(source, seed); };

        void set_seed(RNGSources source, int seed = 1);
        inline int seed() { return stream_.seed(); }
        inline static RNG* defaultrng() { return (default_); }

        inline int uniform_positive_int() {  // range [0, MAXINT]
                return (int)(stream_.next());
        }
        inline double uniform_double() { // range [0.0, 1.0)
                return stream_.next_double();
        }

        inline int random() { return uniform_positive_int(); }
        inline double uniform() {return uniform_double();}
```

```
        inline int uniform(int k)
                { return (uniform_positive_int() % (unsigned)k); }
        inline double uniform(double r)
                { return (r * uniform()); }
        inline double uniform(double a, double b)
                { return (a + uniform(b - a)); }
        inline double exponential()
                { return (-log(uniform())); }
        inline double exponential(double r)
                { return (r * exponential()); }

        // See "Wide-area traffic: the failure of poisson modeling", Vern
        // Paxson and Sally Floyd, IEEE/ACM Transaction on Networking, 3(3),
        // pp. 226-244, June 1995, on characteristics of counting processes
        // with Pareto interarrivals.

        inline double pareto(double scale, double shape) {
                // When 1 < shape < 2, its mean is scale**shape, its
                // variance is infinite.
                return (scale * (1.0/pow(uniform(), 1.0/shape)));
        }

        inline double paretoII(double scale, double shape) {
                return (scale * ((1.0/pow(uniform(), 1.0/shape)) - 1));
        }
        double normal(double avg, double std);
        inline double lognormal(double avg, double std)
          { return (exp(normal(avg, std))); }
protected
                RNGImplementation stream_;
};      static RNG* default_;

#ifdef rng_test
class RNGTest {
public:
        RNGTest();
        void verbose();
};      void first_n(RNG::RNGSources source, long seed, int n);
#endif /* rng_test */

#endif /*       _rng_h_ */
```

RNG.CPP

```
/* new random number generator */

#ifndef WIN32                                   // for gettimeofday
#include <time.h>
#endif

#include <sys/timeb.h>
#include <stdio.h>
#include "c:\netsim\ns-2.1b8a-win\rng1.h"
 // for gettimeofday

#ifndef MAXINT
#define MAXINT          2147483647      // XX [for now]
#endif

/* RNGImplementation */

/*
 * Generate a periodic sequence of pseudo-random numbers with
 * a period of 2^31 - 2.  The generator is the "minimal standard"
 * multiplicative linear congruential generator of Park, S.K. and
 * Miller, K.W., "Random Number Generators: Good Ones are Hard to Find,"
 * CACM 31:10, Oct. 88, pp. 1192-1201.
 * The algorithm implemented is:  Sn = (a*s) mod m.
 *   The modulus m can be approximately factored as: m = a*q + r,
 *   where q = m div a and r = m mod a.

 * Then Sn = g(s) + m*d(s)
 *   where g(s) = a(s mod q) - r(s div q)
 *   and d(s) = (s div q) - ((a*s) div m)

 * Observations:
 *   - d(s) is either 0 or 1.
 *   - both terms of g(s) are in 0, 1, 2, . . ., m - 1.
 *   - |g(s)| <= m - 1.
 *   - if g(s) > 0, d(s) = 0, else d(s) = 1.
 *   - s mod q = s - k*q, where k = s div q.

 * Thus Sn = a(s - k*q) - r*k,
 *   if (Sn <= 0), then Sn += m.

 * To test an implementation for A = 16807, M = 2^31-1, you should
 *   get the following sequences for the given starting seeds:
 *
 *   s0, s1,    s2,       s3,          . . . , s10000,     . . . , s551246
 *    1, 16807, 282475249, 1622650073, . . . , 1043618065, . . . , 1003
 *    1973272912, 1207871363, 531082850, 967423018

 * It is important to check for s10000 and s551246 with s0=1, to guard
```

```
 * against overflow.
 */
/*
 * The sparc assembly code [no longer here] is based on Carta, D.G., "Two Fast
 * Implementations of the 'Minimal Standard' Random Number
 * Generator," CACM 33:1, Jan. 90, pp. 87-88.
 *
 * ASSUME that "the product of two [signed 32-bit] integers (a, sn)
 *      will occupy two [32-bit] registers (p, q)."
 * Thus: a*s = (2^31)p + q
 *
 * From the observation that: x = y mod z is but
 *   x = z * the fraction part of (y/z).
 * Let: sn = m * Frac(as/m)
 *
 * For m = 2^31 - 1,
 *   sn = (2^31 - 1) * Frac[as/(2^31 -1)]
 *      = (2^31 - 1) * Frac[as(2^-31 + 2^-2(31) + 2^-3(31) + . . .)]
 *      = (2^31 - 1) * Frac{[(2^31)p + q] [2^-31 + 2^-2(31) + 2^-3(31) + . .
.]}
 *
 *      = (2^31 - 1) * Frac[p+(p+q)2^-31+(p+q)2^-2(31)+(p+q)3^(-31)+ . . .]
 *
 * if p+q < 2^31:
 *   sn = (2^31 - 1) * Frac[p + a fraction + a fraction + a fraction + . . .]
 *      = (2^31 - 1) * [(p+q)2^-31 + (p+q)2^-2(31) + (p+q)3^(-31) + . . .]
 *      = p + q
 *
 * otherwise:
 *   sn = (2^31 - 1) * Frac[p + 1.frac . . .]
 *      = (2^31 - 1) * (-1 + 1.frac . . .)
 *      = (2^31 - 1) * [-1 + (p+q)2^-31 + (p+q)2^-2(31) + (p+q)3^(-31) + . . .]
 */    = p + q - 2^31 + 1


#define A       16807L          /* multiplier, 7**5 */
#define M       2147483647L     /* modulus, 2**31 - 1; both used in random */
#define INVERSE_M ((double)4.656612875e-10)     /* (1.0/(double)M) */

long
RNGImplementation::next()
{
        long L, H;
        L = A * (seed_ & 0xffff);
        H = A * (seed_ >> 16);

        seed_ = ((H & 0x7fff) << 16) + L;
        seed_ -= 0x7fffffff;
        seed_ += H >> 15;

        if (seed_ <= 0) {
```

```
            seed_ += 0x7fffffff;
        }
        return(seed_);
}

double
RNGImplementation::next_double()
{
        long i = next();
        return i * INVERSE_M;
}


/* RNG implements a nice front-end around RNGImplementation */

/* default RNG */

RNG* RNG::default_ = NULL;

double
RNG::normal(double avg, double std)
{
        static int parity = 0;
        static double nextresult;
        double sam1, sam2, rad;

        if (std == 0) return avg;
        if (parity == 0) {
                sam1 = 2*uniform() - 1;
                sam2 = 2*uniform() - 1;
                while ((rad = sam1*sam1 + sam2*sam2) >= 1) {
                        sam1 = 2*uniform() - 1;
                        sam2 = 2*uniform() - 1;
                }
                rad = sqrt((-2*log(rad))/rad);
                nextresult = sam2 * rad;
                parity = 1;
                return (sam1 * rad * std + avg);
        }
        else {
                parity = 0;
                return (nextresult * std + avg);
}       }


void
RNG::set_seed(RNGSources source, int seed)
{
        /* The following predefined seeds are evenly spaced around
         * the 2^31 cycle.  Each is approximately 33,000,000 elements
```

```
        * apart.
        */
#define N_SEEDS_ 64
        static long predef_seeds[N_SEEDS_] = {
                1973272912L, 188312339L, 1072664641L, 694388766L,
                2009044369L, 934100682L, 1972392646L, 1936856304L,
                1598189534L, 1822174485L, 1871883252L, 558746720L,
                605846893L, 1384311643L, 2081634991L, 1644999263L,
                773370613L, 358485174L, 1996632795L, 1000004583L,
                1769370802L, 1895218768L, 186872697L, 1859168769L,
                349544396L, 1996610406L, 222735214L, 1334983095L,
                144443207L, 720236707L, 762772169L, 437720306L,
                939612284L, 425414105L, 1998078925L, 981631283L,
                1024155645L, 822780843L, 701857417L, 960703545L,
                2101442385L, 2125204119L, 2041095833L, 89865291L,
                898723423L, 1859531344L, 764283187L, 1349341884L,
                678622600L, 778794064L, 1319566104L, 1277478588L,
                538474442L, 683102175L, 999157082L, 985046914L,
                722594620L, 1695858027L, 1700738670L, 1995749838L,
                1147024708L, 346983590L, 565528207L, 513791680L
        };
        static long heuristic_sequence = 0;

        switch (source) {
        case RAW_SEED_SOURCE:
                if (seed <= 0 || (unsigned int)seed >= MAXINT)   // Wei Ye
                        abort();
                // use it as it is
                break;
        case PREDEF_SEED_SOURCE:
                if (seed < 0 || seed >= N_SEEDS_)
                        abort();
                seed = predef_seeds[seed];
                break;
        case HEURISTIC_SEED_SOURCE:
                _timeb* ptr = (_timeb*) malloc( sizeof(_timeb));
                _ftime(ptr);
                heuristic_sequence++;  // Always make sure we're different than last time.
                seed = (ptr->time ^ ptr->millitm ^ (heuristic_sequence << 8)) & 0x7fffffff;
                break;
        };
        // set it
                if (seed < 0)
                seed = -seed;
        stream_.set_seed(seed);
                                        values of heuristic seed.
                                        heuristic seeds
```

```
        int i;
        for (i = 0; i < 128; i++) {
                stream_.next();
        };      };      }


/* RNGTest:
 * Make sure the RNG makes known values.
 * Optionally, print out some stuff.
 * Simple test program:
 * #include "rng.h"
 * void main() { RNGTest test; test.verbose(); }
 */

#ifdef rng_test
RNGTest::RNGTest()
{
        RNG rng(RNG::RAW_SEED_SOURCE, 1L);
        int i;
        long r;

        for (i = 0; i < 10000; i++)
                r = rng.uniform_positive_int();

        if (r != 1043618065L)
                abort();

        for (i = 10000; i < 551246; i++)
                r = rng.uniform_positive_int();

        if (r != 1003L)
}               abort();

void
RNGTest::first_n(RNG::RNGSources source, long seed, int n)
{
        RNG rng(source, seed);

        for (int i = 0; i < n; i++) {
                int r = rng.uniform_positive_int();
                printf("%10d ", r);
        };
}       printf("\n");

void
RNGTest::verbose()
{
        printf ("default: ");
        first_n(RNG::RAW_SEED_SOURCE, 1L, 5);
```

```
        int i;
        for (i = 0; i < 64; i++) {
                printf ("predef source %2d: ", i);
                first_n(RNG::PREDEF_SEED_SOURCE, i, 5);
        };

        printf("heuristic seeds should be different from each other and on each run.\n");
        for (i = 0; i < 64; i++) {
                printf ("heuristic source %2d: ", i);
                first_n(RNG::HEURISTIC_SEED_SOURCE, i, 5);
        };
}
#endif /* rng_test */
```

NETWORK SCENARIO

The user-defined inputs to the scenario generation program are: number of nodes, network area, maximum node speed, pause time between two motions and the simulation time. Using setdest.cpp and rng.cpp a network scenario generated was for 10 nodes distributed within an area of 150m x 100m moving at a maximum speed of 1m/s for a simulation period of 30 seconds with a pause time of 10 seconds. The format of the output scenario is shown below.

```
#
# nodes: 10, pause: 10.00, max speed: 1.00  max x = 150.00, max y: 100.00
#
$node_(0) set X_ 84.702086972585
$node_(0) set Y_ 58.650548920036
$node_(0) set Z_ 0.000000000000
$node_(1) set X_ 59.663626622828
$node_(1) set Y_ 11.048468532877
$node_(1) set Z_ 0.000000000000
$node_(2) set X_ 137.415962792858
$node_(2) set Y_ 0.057854317153
$node_(2) set Z_ 0.000000000000
$node_(3) set X_ 108.536262645495
$node_(3) set Y_ 12.644252745067
$node_(3) set Z_ 0.000000000000
$node_(4) set X_ 17.933846342331
$node_(4) set Y_ 42.770327645318
$node_(4) set Z_ 0.000000000000
$node_(5) set X_ 61.345159194614
$node_(5) set Y_ 52.060425535850
$node_(5) set Z_ 0.000000000000
$node_(6) set X_ 119.358040814208
$node_(6) set Y_ 67.061380184935
$node_(6) set Z_ 0.000000000000
$node_(7) set X_ 0.925241550813
$node_(7) set Y_ 67.023163549335
$node_(7) set Z_ 0.000000000000
$node_(8) set X_ 87.464749686211
$node_(8) set Y_ 13.365369155933
$node_(8) set Z_ 0.000000000000
$node_(9) set X_ 47.639123439900
$node_(9) set Y_ 80.498464438773
$node_(9) set Z_ 0.000000000000
$god_ set-dist 0 1 1
$god_ set-dist 0 2 1
```

```
$god_ set-dist 0 3 1
$god_ set-dist 0 4 1
$god_ set-dist 0 5 1
$god_ set-dist 0 6 1
$god_ set-dist 0 7 1
$god_ set-dist 0 8 1
$god_ set-dist 0 9 1
$god_ set-dist 1 2 1
$god_ set-dist 1 3 1
$god_ set-dist 1 4 1
$god_ set-dist 1 5 1
$god_ set-dist 1 6 1
$god_ set-dist 1 7 1
$god_ set-dist 1 8 1
$god_ set-dist 1 9 1
$god_ set-dist 2 3 1
$god_ set-dist 2 4 1
$god_ set-dist 2 5 1
$god_ set-dist 2 6 1
$god_ set-dist 2 7 1
$god_ set-dist 2 8 1
$god_ set-dist 2 9 1
$god_ set-dist 3 4 1
$god_ set-dist 3 5 1
$god_ set-dist 3 6 1
$god_ set-dist 3 7 1
$god_ set-dist 3 8 1
$god_ set-dist 3 9 1
$god_ set-dist 4 5 1
$god_ set-dist 4 6 1
$god_ set-dist 4 7 1
$god_ set-dist 4 8 1
$god_ set-dist 4 9 1
$god_ set-dist 5 6 1
$god_ set-dist 5 7 1
$god_ set-dist 5 8 1
$god_ set-dist 5 9 1
$god_ set-dist 6 7 1
$god_ set-dist 6 8 1
$god_ set-dist 6 9 1
$god_ set-dist 7 8 1
$god_ set-dist 7 9 1
$god_ set-dist 8 9 1
$ns_ at 10.000000000000 "$node_(0) setdest 56.537840794832 87.660192594735
0.048570174744"
$ns_ at 10.000000000000 "$node_(1) setdest 47.839044613026 20.548569046994
0.597999910605"
$ns_ at 10.000000000000 "$node_(2) setdest 87.674710912507 65.910922856275
0.648805038806"
$ns_ at 10.000000000000 "$node_(3) setdest 69.943167461089 89.877053713281
0.636418388488"
```

```
$ns_ at 10.000000000000 "$node_(4) setdest 42.578382109176 76.578764462796
0.592943941488"
$ns_ at 10.000000000000 "$node_(5) setdest 91.323766732110 52.365031720339
0.990881701875"
$ns_ at 10.000000000000 "$node_(6) setdest 112.314643338129 48.140455709157
0.966391465094"
$ns_ at 10.000000000000 "$node_(7) setdest 21.203204672225 74.840629923324
0.464671876939"
$ns_ at 10.000000000000 "$node_(8) setdest 111.035418928810 14.857356675516
0.075936585696"
$ns_ at 10.000000000000 "$node_(9) setdest 39.929379259890 95.384837586699
0.329654042745"
#
# Destination Unreachables: 0
#
# Route Changes: 0
#
# Link Changes: 0
#
# Node | Route Changes | Link Changes
#    0 |             0 |            0
#    1 |             0 |            0
#    2 |             0 |            0
#    3 |             0 |            0
#    4 |             0 |            0
#    5 |             0 |            0
#    6 |             0 |            0
#    7 |             0 |            0
#    8 |             0 |            0
#    9 |             0 |            0
#
```

Traffic Generation

The traffic is generated using the TCL script based program **CBRGEN.TCL;** the traffic generation program sets up connection between a randomly chosen source and destination pair. The traffic starts at a random time within 100 seconds from the start of the simulation process. UDP/CBR or TCP/FTP traffic can be generated using this program. The number of packets/sec, number of connection and the type of traffic are user-defined inputs to this program.

CBRGEN.TCL

```
# ======================================================================
# Default Script Options
# ======================================================================
set opt(nn)             0               ;# Number of Nodes
set opt(seed)           0.0             ;# Starting node number
set opt(mc)             0
set opt(pktsize)        512
set opt(rate)           0               ;# inverse of rate
set opt(interval)       0.0
set opt(type)           ""

proc usage { } {
   global argv0

   puts "\nusage: $argv0 \[-type cbr|tcp\] \[-nn nodes\] \[-seed seed\] \[-mc connections\] \[-rate
rate\]\n"
}

proc getopt {argc argv} {
                global opt
                lappend optlist nn seed mc rate type

                for {set i 0} {$i < $argc} {incr i} {
                    set arg [lindex $argv $i]
                    if {[string range $arg 0 0] != "-"} continue

                    set name [string range $arg 1 end]
                    set opt($name) [lindex $argv [expr $i+1]]

                }
}
```

```
proc create-cbr-connection { src dst } {
                global rng cbr_cnt opt

                set stime [$rng uniform 0.0 180.0]

                puts "#\n# $src connecting to $dst at time $stime\n#"

                ##puts "set cbr_($cbr_cnt) \[\$ns_ create-connection \
                    ##CBR \$node_($src) CBR \$node_($dst) 0\]";
                puts "set udp_($cbr_cnt) \[new Agent/UDP\]"
                puts "\$ns_ attach-agent \$node_($src) \$udp_($cbr_cnt)"
                puts "set null_($cbr_cnt) \[new Agent/Null\]"
                puts "\$ns_ attach-agent \$node_($dst) \$null_($cbr_cnt)"
                puts "set cbr_($cbr_cnt) \[new Application/Traffic/CBR\]"
                puts "\$cbr_($cbr_cnt) set packetSize_ $opt(pktsize)"
                puts "\$cbr_($cbr_cnt) set interval_ $opt(interval)"
                puts "\$cbr_($cbr_cnt) set random_ 1"
                puts "\$cbr_($cbr_cnt) set maxpkts_ 10000000"
                puts "\$cbr_($cbr_cnt) attach-agent \$udp_($cbr_cnt)"
                puts "\$ns_ connect \$udp_($cbr_cnt) \$null_($cbr_cnt)"

                puts "\$ns_ at $stime \"\$cbr_($cbr_cnt) start\""

                incr cbr_cnt
}

proc create-tcp-connection { src dst } {
                global rng cbr_cnt opt

                set stime [$rng uniform 0.0 180.0]

                puts "#\n# $src connecting to $dst at time $stime\n#"

                puts "set tcp_($cbr_cnt) \[\$ns_ create-connection \
                    TCP \$node_($src) TCPSink \$node_($dst) 0\]";
                puts "\$tcp_($cbr_cnt) set window_ 32"
                puts "\$tcp_($cbr_cnt) set packetSize_ $opt(pktsize)"

                puts "set ftp_($cbr_cnt) \[\$tcp_($cbr_cnt) attach-source FTP\]"


                puts "\$ns_ at $stime \"\$ftp_($cbr_cnt) start\""

                incr cbr_cnt
}
```

```
# ======================================================================
getopt $argc $argv

if { $opt(type) == "" } {
    usage
    exit
} elseif { $opt(type) == "cbr" } {
    if { $opt(nn) == 0 || $opt(seed) == 0.0 || $opt(mc) == 0 || $opt(rate) == 0 } {
                usage
                exit
    }

    set opt(interval) [expr 1 / $opt(rate)]
    if { $opt(interval) <= 0.0 } {
                puts "\ninvalid sending rate $opt(rate)\n"
                exit
    }
}

puts "#\n# nodes: $opt(nn), max conn: $opt(mc), send rate: $opt(interval), seed: $opt(seed)\n#"

set rng [new RNG]
$rng seed $opt(seed)

set u [new RandomVariable/Uniform]
$u set min_ 0
$u set max_ 100
$u use-rng $rng

set cbr_cnt 0
set src_cnt 0

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

                set x [$u value]

                if {$x < 50} {continue;}

                incr src_cnt

                set dst [expr ($i+1) % [expr $opt(nn) + 1] ]
                #if { $dst == 0 } {
                   #set dst [expr $dst + 1]
                   #}

                if { $opt(type) == "cbr" } {
                    create-cbr-connection $i $dst
                } else {
                    create-tcp-connection $i $dst

                }
```

```
        if { $cbr_cnt == $opt(mc) } {
            break
        }

        if {$x < 75} {continue;}

        set dst [expr ($i+2) % [expr $opt(nn) + 1] ]
        #if { $dst == 0 } {
            #set dst [expr $dst + 1]
        #}

        if { $opt(type) == "cbr" } {
            create-cbr-connection $i $dst
        } else {
            create-tcp-connection $i $dst
        }

        if { $cbr_cnt == $opt(mc) } {
            break
        }
}

puts "#\n#Total sources/connections: $src_cnt/$cbr_cnt\n#"
```

OUTPUT TRAFFIC FILE

The sample output traffic file shown below was created for a 3 connections, at a data rate of 4 packets/sec, in a 10-node scenario. The traffic generated was of type UDP/CBR.

```
# nodes: 10, max conn: 3, send rate: 0.25, seed: 1.0
# 2 connecting to 3 at time 82.557023746220864
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(2) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(3) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 0
$cbr_(0) set maxpkts_ 100000000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 82.557023746220864 "$cbr_(0) start"
# 2 connecting to 4 at time 95.898102734190459
set udp_(1) [new Agent/UDP]
$ns_ attach-agent $node_(2) $udp_(1)
set null_(1) [new Agent/Null]
$ns_ attach-agent $node_(4) $null_(1)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 512
$cbr_(1) set interval_ 0.25
$cbr_(1) set random_ 0
$cbr_(1) set maxpkts_ 100000000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(1)
$ns_ at 95.898102734190459 "$cbr_(1) start"
# 5 connecting to 6 at time 122.2733530505902
set udp_(2) [new Agent/UDP]
$ns_ attach-agent $node_(5) $udp_(2)
set null_(2) [new Agent/Null]
$ns_ attach-agent $node_(6) $null_(2)
set cbr_(2) [new Application/Traffic/CBR]
$cbr_(2) set packetSize_ 512
$cbr_(2) set interval_ 0.25
$cbr_(2) set random_ 0
$cbr_(2) set maxpkts_ 100000000
$cbr_(2) attach-agent $udp_(2)
$ns_ connect $udp_(2) $null_(2)
$ns_ at 122.2733530505902 "$cbr_(2) start"

#Total sources/connections: 2/3
```

Simulation Script

Once the scenario file and traffic file have been generated, the simulation is run using *ns2*. A TCL script is used for declaring the network parameters such as physical layer model, MAC model, Routing Protocol, Time for which the network is simulated etc. A sample simulation script for a 100-node network with the simulation run for 3000 seconds is shown below.

```
#===================================================================
# Define options
#===================================================================
set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhy
set val(mac)            Mac/802_11
set val(ifq)            Queue/DropTail/PriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(x)              1000.0  ;# X dimension of the topography
set val(y)              1000.0  ;# Y dimension of the topography
set val(ifqlen)         30      ;# max packet in ifq
set val(seed)           0.0
set val(adhocRouting)   GPEAR
set val(nn)             100     ;# how many nodes are simulated
set val(cp)             "c:/netsim1/ns-2.1b8a-win/indep-utils/cmu-scen-gen/cbr-25-100"
# path where traffic file is present/name of file
set val(sc) "c:/netsim1/ns-2.1b8a-win/indep-utils/cmu-scen-gen/setdest/debug/area-1000-1000-4"
#path where scenario file is present/name of file
set val(stop)           3000.00 ;# simulation time
set val(energyModel)    "EnergyModel"  ;#include energy model for tracing energy information
#===================================================================
#default parameters
#===================================================================
LL set mindelay_        50us
LL set delay_           25us
LL set bandwidth_       0       ;# not used

Agent/Null set sport_   0
Agent/Null set dport_   0

Agent/CBR set sport_    0
Agent/CBR set dport_    0

Agent/TCPSink set sport_ 0
Agent/TCPSink set dport_
```

```
Agent/TCP set sport_            0
Agent/TCP set dport_            0
Agent/TCP set packetSize_       1460

Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1
# enable Drop Tail Queue with Priority for control packets

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 2*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0

#===================================================================
# Main Program
#===================================================================

# Initialize Global Variables
# create simulator instance

set ns_         [new Simulator]

# setup topography object

set topo [new Topography]

# create trace object for ns output of simulation is sent to file wireless.tr
set tracefd    [open wireless.tr w]

$ns_ use-newtrace
$ns_ trace-all $tracefd

# define topology
$topo load_flatgrid $val(x) $val(y)

# Create God
set god_ [create-god $val(nn)]
```

```
# define how node should be created
#global node setting

$ns_ node-config -adhocRouting $val(adhocRouting) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan) \
                 -topoInstance $topo \
                 -energyModel $val(energyModel)\
                 -initialEnergy 1000\
                  -agentTrace ON \
                  -routerTrace ON\
                  -macTrace OFF

# Create the specified number of nodes [$val(nn)] and "attach" them  to the channel.

for {set i 0} {$i < $val(nn) } {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0          ;# disable random motion
    }

# Define node movement model
puts "Loading connection pattern..."
source $val(cp)

# Define traffic model
puts "Loading scenario file..."
source $val(sc)

# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i) reset";
}

proc finish {} {
        global ns_ tracefd
}       close $tracefd

$ns_ at 3000.0 "finish"

$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"

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

Output Trace

The up to date information of the complete activity in the network is stored in the output trace file some of the relevant information in the trace file are:

Event type: In the traces above, the first field describes the type of event taking place at the node and can be one of the four types:

s send

r receive]item[d] drop

f forward

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

-t time

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

-P gpear This denotes the adhoc routing protocol calledGPEAR. Information on GPEAR is represented by the following tags:

-Pn: how many nodes traversed

-Pq: routing request flag

-Pi: route request sequence number

-Pp: routing reply flag

-Pl: reply length

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

-Pw: error report flag

-Pm: number of errors

-Pc: report to whom

-Pb: link error from linka->linkb

-Pu: Route Update Flag

-Pr: Route Expiry Time

-Pmp: Maximum Propagation of RREQ (used in Ring Search)

-P cbr Constant bit rate. Information about the CBR application is represented by the following tags:

-Pi: sequence number

-Pf: how many times this pkt was forwarded

-Po: optimal number of forwards

Output Trace File

```
s -t 8.557023746 -Hs 2 -Hd -2 -Ni 2 -Nx 12.84 -Ny 718.83 -Nz 0.00 -Ne
1000.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.0 -Id 3.0 -It
cbr -Il 512 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 2
r -t 8.557023746 -Hs 2 -Hd -2 -Ni 2 -Nx 12.84 -Ny 718.83 -Nz 0.00 -Ne
1000.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.0 -Id 3.0 -It
cbr -Il 512 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 2
s -t 8.566366607 -Hs 2 -Hd -1 -Ni 2 -Nx 12.84 -Ny 718.83 -Nz 0.00 -Ne
1000.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.255 -Id 3.255
-It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps 1 -Pp 0 -
Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr 0.000000 -Pmp
1
r -t 8.566962897 -Hs 67 -Hd -1 -Ni 67 -Nx 13.79 -Ny 806.00 -Nz 0.00 -Ne
999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is 2.255 -
Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps 1
-Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566962940 -Hs 49 -Hd -1 -Ni 49 -Nx 90.25 -Ny 782.10 -Nz 0.00 -Ne
999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is 2.255 -
Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps 1
-Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963104 -Hs 55 -Hd -1 -Ni 55 -Nx 140.16 -Ny 641.05 -Nz 0.00 -
Ne 999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq
1 -Ps 1 -Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963152 -Hs 91 -Hd -1 -Ni 91 -Nx 57.23 -Ny 876.37 -Nz 0.00 -Ne
999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is 2.255 -
Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps 1
-Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963153 -Hs 6 -Hd -1 -Ni 6 -Nx 142.49 -Ny 819.19 -Nz 0.00 -Ne
999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is 2.255 -
Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps 1
-Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963235 -Hs 90 -Hd -1 -Ni 90 -Nx 147.79 -Ny 850.30 -Nz 0.00 -
Ne 999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq
1 -Ps 1 -Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963237 -Hs 7 -Hd -1 -Ni 7 -Nx 196.86 -Ny 675.02 -Nz 0.00 -Ne
999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is 2.255 -
Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps 1
-Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963242 -Hs 87 -Hd -1 -Ni 87 -Nx 203.35 -Ny 725.54 -Nz 0.00 -
Ne 999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq
1 -Ps 1 -Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963294 -Hs 56 -Hd -1 -Ni 56 -Nx 210.13 -Ny 658.56 -Nz 0.00 -
Ne 999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq
```

```
1 -Ps 1 -Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963324 -Hs 21 -Hd -1 -Ni 21 -Nx 224.30 -Ny 758.39 -Nz 0.00 -
Ne 999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq
1 -Ps 1 -Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
r -t 8.566963408 -Hs 47 -Hd -1 -Ni 47 -Nx 233.57 -Ny 623.46 -Nz 0.00 -
Ne 999.999804 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It GPEAR -Il 72 -If 0 -Ii 1 -Iv 32 -P gpear -Ph 1 -Pq
1 -Ps 1 -Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 1
s -t 8.807023746 -Hs 2 -Hd -2 -Ni 2 -Nx 12.84 -Ny 718.83 -Nz 0.00 -Ne
999.999673 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.0 -Id 3.0 -It
cbr -Il 512 -If 0 -Ii 7 -Iv 32 -Pn cbr -Pi 1 -Pf 0 -Po 2
s -t 10.567011549 -Hs 2 -Hd -1 -Ni 2 -Nx 12.84 -Ny 718.83 -Nz 0.00 -Ne
999.999673 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.255 -Id 3.255
-It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps 2 -Pp 0
-Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr 0.000000 -Pmp
3
r -t 10.567607840 -Hs 67 -Hd -1 -Ni 67 -Nx 13.79 -Ny 806.00 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
r -t 10.567607883 -Hs 49 -Hd -1 -Ni 49 -Nx 90.25 -Ny 782.10 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
r -t 10.567608047 -Hs 55 -Hd -1 -Ni 55 -Nx 140.16 -Ny 641.05 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
r -t 10.567608095 -Hs 91 -Hd -1 -Ni 91 -Nx 57.23 -Ny 876.37 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
r -t 10.567608096 -Hs 6 -Hd -1 -Ni 6 -Nx 142.49 -Ny 819.19 -Nz 0.00 -Ne
999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is 2.255 -
Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps
2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 3
r -t 10.567608177 -Hs 90 -Hd -1 -Ni 90 -Nx 147.79 -Ny 850.30 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
r -t 10.567608180 -Hs 7 -Hd -1 -Ni 7 -Nx 196.86 -Ny 675.02 -Nz 0.00 -Ne
999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is 2.255 -
Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -Pq 1 -Ps
2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 3
r -t 10.567608185 -Hs 87 -Hd -1 -Ni 87 -Nx 203.35 -Ny 725.54 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
```

```
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
r -t 10.567608237 -Hs 56 -Hd -1 -Ni 56 -Nx 210.13 -Ny 658.56 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
r -t 10.567608266 -Hs 21 -Hd -1 -Ni 21 -Nx 224.30 -Ny 758.39 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
r -t 10.567608351 -Hs 47 -Hd -1 -Ni 47 -Nx 233.57 -Ny 623.46 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 72 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 1 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
f -t 10.569568354 -Hs 47 -Hd -1 -Ni 47 -Nx 233.57 -Ny 623.46 -Nz 0.00 -
Ne 999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 128 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 2 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
f -t 10.569617917 -Hs 7 -Hd -1 -Ni 7 -Nx 196.86 -Ny 675.02 -Nz 0.00 -Ne
999.999608 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is 2.255 -
Id 3.255 -It gpear -Il 128 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 2 -Pq 1 -
Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 3
f -t 10.569893098 -Hs 67 -Hd -1 -Ni 67 -Nx 13.79 -Ny 806.00 -Nz 0.00 -
Ne 999.999324 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id 3.255 -It gpear -Il 128 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 2 -
Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0
-Pr 0.000000 -Pmp 3
s -t 10.572255074 -Hs 3 -Hd 47 -Ni 3 -Nx 314.54 -Ny 432.88 -Nz 0.00 -Ne
999.998755 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 3.255 -Id 2.255
-It gpear -Il 92 -If 0 -Ii 222 -Iv 255 -P gpear -Ph 3 -Pq 0 -Ps 2 -Pp 1
-Pn 2 -Pl 3 -Pe 2->3 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr 3000.000000 -
Pmp 0
s -t 10.580908474 -Hs 3 -Hd 44 -Ni 3 -Nx 314.54 -Ny 432.88 -Nz 0.00 -Ne
999.995687 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 3.255 -Id 2.255
-It gpear -Il 100 -If 0 -Ii 232 -Iv 255 -P gpear -Ph 4 -Pq 0 -Ps 2 -Pp
1 -Pn 2 -Pl 4 -Pe 2->3 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr 3000.000000
-Pmp 0
r -t 10.596143245 -Hs 44 -Hd 44 -Ni 44 -Nx 166.27 -Ny 415.86 -Nz 0.00 -
Ne 999.990253 -Nl RTR -Nw --- -Ma a2 -Md 2c -Ms 3 -Mt 800 -Is 3.255 -Id
2.255 -It gpear -Il 100 -If 0 -Ii 232 -Iv 255 -P gpear -Ph 4 -Pq 0 -Ps
2 -Pp 1 -Pn 2 -Pl 4 -Pe 2->3 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
3000.000000 -Pmp 0
f -t 10.596143245 -Hs 44 -Hd 47 -Ni 44 -Nx 166.27 -Ny 415.86 -Nz 0.00 -
Ne 999.990253 -Nl RTR -Nw --- -Ma a2 -Md 2c -Ms 3 -Mt 800 -Is 3.255 -Id
2.255 -It gpear -Il 100 -If 0 -Ii 232 -Iv 255 -P gpear -Ph 4 -Pq 0 -Ps
2 -Pp 1 -Pn 2 -Pl 4 -Pe 2->3 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
1.679233 -Pmp 0
r -t 10.597216755 -Hs 2 -Hd -1 -Ni 2 -Nx 12.84 -Ny 718.83 -Nz 0.00 -Ne
999.989528 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 38 -Mt 800 -Is 2.255
-Id 3.255 -It gpear -Il 128 -If 0 -Ii 221 -Iv 32 -P gpear -Ph 2 -Pq 1 -
```

```
Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->0 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr
0.000000 -Pmp 3
s -t 10.619415393 -Hs 2 -Hd 47 -Ni 2 -Nx 12.84 -Ny 718.83 -Nz 0.00 -Ne
999.982318 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.0 -Id 3.0 -It
cbr -Il 596 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 0 -Po 2
r -t 10.622909798 -Hs 47 -Hd 47 -Ni 47 -Nx 233.57 -Ny 623.46 -Nz 0.00 -
Ne 999.980194 -Nl RTR -Nw --- -Ma a2 -Md 2f -Ms 2 -Mt 800 -Is 2.0 -Id
3.0 -It cbr -Il 596 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 1 -Po 2
f -t 10.622909798 -Hs 47 -Hd 52 -Ni 47 -Nx 233.57 -Ny 623.46 -Nz 0.00 -
Ne 999.980194 -Nl RTR -Nw --- -Ma a2 -Md 2f -Ms 2 -Mt 800 -Is 2.0 -Id
3.0 -It cbr -Il 596 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 1 -Po 2
s -t 10.626639083 -Hs 3 -Hd 47 -Ni 3 -Nx 314.54 -Ny 432.88 -Nz 0.00 -Ne
999.979389 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 3.255 -Id 2.255
-It gpear -Il 92 -If 0 -Ii 247 -Iv 255 -P gpear -Ph 3 -Pq 0 -Ps 0 -Pp 1
-Pn 0 -Pl 3 -Pe 2->3 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0 -Pu 0 -Pr 4.000000 -Pmp
0
r -t 10.626664190 -Hs 52 -Hd 52 -Ni 52 -Nx 189.88 -Ny 388.29 -Nz 0.00 -
Ne 999.980044 -Nl RTR -Nw --- -Ma a2 -Md 34 -Ms 2f -Mt 800 -Is 2.0 -Id
3.0 -It cbr -Il 596 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 2 -Po 2
f -t 10.626664190 -Hs 52 -Hd 3 -Ni 52 -Nx 189.88 -Ny 388.29 -Nz 0.00 -
Ne 999.980044 -Nl RTR -Nw --- -Ma a2 -Md 34 -Ms 2f -Mt 800 -Is 2.0 -Id
3.0 -It cbr -Il 596 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 2 -Po 2
r -t 10.643354738 -Hs 3 -Hd 3 -Ni 3 -Nx 314.54 -Ny 432.88 -Nz 0.00 -Ne
999.972784 -Nl RTR -Nw --- -Ma a2 -Md 3 -Ms 34 -Mt 800 -Is 2.0 -Id 3.0
-It cbr -Il 596 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 3 -Po 2
r -t 10.643354738 -Hs 3 -Hd 3 -Ni 3 -Nx 314.54 -Ny 432.88 -Nz 0.00 -Ne
999.972784 -Nl AGT -Nw --- -Ma a2 -Md 3 -Ms 34 -Mt 800 -Is 2.0 -Id 3.0
-It cbr -Il 596 -If 0 -Ii 0 -Iv 32 -Pn cbr -Pi 0 -Pf 3 -Po 2
```

Statistics Extraction

Information is extracted from the trace file using Practical Extraction and Report Language (PERL) in order to calculate statistics such as Packet Deliver Ratio (PDR), Control Overhead, Non-optimality, Energy Consumption etc. The columns required for the statistics are extracted using the PERL program *column.pl*.

Column.pl

```
$line = <STDIN>;                    :name of trace file
$length = $#ARGV+1;                 : ARGV is the list of columns to be extracted
while ($line)
{
$i =0;
while($i <$length)
{
@word = split('\s+',$line);
print "@word[@ARGV[$i]]\t";
$i = $i+1;
}
$line = <STDIN>;
print "\n";
}
```

Once the relevant columns have been extracted the delivery ratio is calculated using the program *del_ratio.pl*, the control overhead is calculated using the program *overhead.pl*, the percentage of packets sent non-optimally using opt.pl and energy consumed using *energy.pl*. Various other PERL programs are used for generating information such as number of RERR packets, RUPDATE packets etc.

Del_ratio.pl

```
$line = <STDIN>;
$count = 0;
$count1 = 0;
while ($line)
{
@word = split('\s+',$line);
if(@word[2] eq "AGT")
{
if (@word[1] eq "r")
{
```

```
$count = $count+1;
}
if (@word[1] eq "s")
{
$count1 = $count1+1;
}
}
$line = <STDIN>;
}
$opt = $count/$count1;              :ratio of received to sent data packets
$optp = $opt*100;
print "$optp\n";
```

Overhead.pl

```
$line = <STDIN>;
$count = 0;
while ($line)
{
@word = split('\s+',$line);
if (@word[0] eq "DSR" and (@word[1] eq "s" or @word[1] eq "f"))
{
$count = $count +1;
 }
$line = <STDIN>;
}
print "$count \n";o
```

Opt.pl

```
$line = <STDIN>;
$count = 0;
$count1 = 0;
while ($line)
{
@word = split('\s+',$line);
if(@word[1] eq "r" and @word[2] eq "AGT")
{
$diff = @word[3]-@word[4];    :Actual number of hops transversed- Optimal number of hops
                              :between source and destination node

if ($diff > 0)
{
$count = $count+1;
}
$count1 = $count1+1;
}
$line = <STDIN>;
}
$opt = $count/$count1;
$optp = $opt*100;
print "$optp\n";
```

Energy.pl

```
$i =0;
while ($i <100)
{
@n[$i]=0;
$i = $i+1;
}
$i =0;
$line = <STDIN>;
while ($line)
{
@word = split('\s+',$line);
while ($i<100)
{
if (@word[0] eq $i)
{
@n[$i]=@word[1];        :updated energy for each node
}
$i =$i+1;
}
$i=0;
$line = <STDIN>;
}
$i =0;
while ($i <100)
{
print "N[$i]= @n[$i]\n";
$i = $i+1;
}
```

REFERENCES

1. C.Hendrick, "Routing Information Protocol (RIP)", Network Working Group, RFC 1058, June 1998.
2. J.Moy, "Open Shortest Path First (OSPF) Version 2", Network Working Group, RFC 2328, April 1998.
3. A.Ballardie "Core Based Tree (CBT) Multicast Routing Architecture", Network Working Group, RFC 2201, September 1997.
4. D. Estrin, D.Farinacci, A.Helmy, D.Thaler, S.Deering, M.Handley, V.Jacobson, C.Liu, and L.Wei "Protocol Independent Multicast", Network Working Group, RFC 2362, June 1998.
5. C.E.Perkins "IP Mobility Support", Network Working Group, RFC 2002, October 1996.
6. J.D.Solomon, "Mobile IP- The Internet Unplugged", PTR- Prentice Hall, New Jersey, 1998.
7. C.E.Perkins, "Mobile IP- Design Principles and Practices", Addison Wesley, Reading, Massachusetts, 1998.
8. C.E.Perkins, D.Johnson, "Mobility Support in Ipv6", draft-ietf-mobileip-ipv6-24.txt, June 2003.
9. C.E.Perkins, "Adhoc Networking", Addison Wesley Professional, Boston, 2001.
10. C.E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers", Computer Communication Review, pp.234-244, October 1994.

11. S. Murthy and J.J. Garcia-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks", ACM Mobile Networks and Applications Journal, Special Issue on Routing in Mobile Communication Networks, pp. 183-97,October 1996.

12. T-W.Chen and M.Gerla, "Global State Routing: A New Routing Scheme for Ad-hoc Wireless Networks", Proceedings of IEEE International Conference on Communication (ICC'98), Atlanta, June 1998.

13. M.Gerla, X.Hong, and G.Pei, "Fisheye State Routing Protocol (FSR) for Adhoc Networks", draft-ietf-manet-fsr-03.txt, June17, 2002

14. A. Iwata, C-C. Chiang, G. Pei, M. Gerla, and T-W. Chen, "Scalable Routing Strategies for Ad Hoc Wireless Networks", IEEE Journal on Selected Areas in Communications, Special Issue on Ad Hoc Networks, pp.1369-79, August 1999.

15. C-C. Chiang, "Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel", Proceedings of IEEE Singapore International Conference on Networks (SICON'97), Singapore, April 1997.

16. V. Park, S. Corson, "Temporally-Ordered Routing Algorithm (TORA) Version 1-Functional Specification", draft-ietf-manet-tora-spec-03.txt, 24 November 2000.

17. C.K.Toh, "Long-Lived Adhoc Routing based on the Concept of Associativity", draft-ietf-manet-longlived-adhoc-routing-00.txt, March 1999.

18. R. Dube, C.D. Rais, K. Wang and S.K.Tripathi , "Signal Stability Based Adaptive routing for Adhoc mobile networks", IEEE Personal Communication, February 1997.

19. Y.B. Ko and N.H. Vaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks," Proceedings of ACM/IEEE MOBICOM'98, Dallas, Texas, October 1998.

20. S. Basagni, I. Chlamtac, V.R. Syrotiuk, and B.A. Woodward, "A Distance Routing Effect Algorithm for Mobility (DREAM)," Proceedings of ACM/IEEE MOBICOM'98, Dallas, Texas, October 1998.

21. J. Gomez, A. T. Campbell, M. Naghshineh, and C. Bisdikian, "PARO: A Power-Aware Routing Optimization Scheme for Mobile Ad hoc Networks", draft-gomez-paro-manet-00.txt, February 2001.

22. Y-C. Hu, D.B. Johnson, and D.A. Maltz, "The Dynamic State Routing Protocol for Mobile Adhoc Networks (DSR)", draft-ietf-manet-dsr-09.txt, 23 April 2003.

23. Y-C. Hu, D.B. Johnson, and D.A. Maltz, "Flow State in the Dynamic Source Routing Protocol for Mobile Ad Hoc Networks", draft-ietf-manet-dsrflow-00.txt, 23 February 2001.

24. C.E. Perkins, E.M. Royer, and S.R. Das, " Ad hoc on demand distance vector (AODV) routing", draft-ietf-manet-aodv-06.txt, 14 July 2000.

25. C.E. Perkins, E.M. Royer, and S.R. Das, " Ad hoc on demand distance vector (AODV) routing", draft-ietf-manet-aodv-13.txt, 14 February 2003.

26. K.R. Anupama, and S.Balasubramanian, "GPS-based Predictive Energy Aware Routing Protocol (GPEAR)", Proceedings of the National Conference on Communication 2003, IIT Madras, pp 473-478, February 2003.

27. K.R. Anupama, and S.Balasubramanian, "GPS-based Predictive Energy Aware Routing Protocol (GPEAR)", Proceedings of the International Multi-Conference of Systematics, Informatics and Cybernetics 2003, Orlando, USA, July 2003.

28. K. Fall and K. Varadhan (Eds.). "*Ns* notes and documentation", available from http://www-mash.cs.berkeley.edu/ns/ , 1999.

29. S. Prata, "C++ Primer Plus", Galgotia Publications, New Delhi,1998.

30. J.K.Ousterhout, "Tcl and the Tk Toolkit", Addison Wesley Professional Computing Series, Boston, 1999.

31. "Otcl Tutorial Version.096", available from http://www-mash.cs.berkeley.edu/ns/, 1995.

32. D.B. Johnson and D.A. Maltz, "Dynamic source routing in adhoc wireless networks", In Mobile Computing, edited by T. Imielinski and H. Korth, chapter 5, pp.153–181. Kluwer Academic Publishers, Norvell, 1996.

33. T.S. Rappaport, "Wireless Communications: Principles and Practice", 2nd edition, Pearson Education, Singapore, 2002.

34. IEEE Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, The Institute of Electrical and Electronics Engineers, IEEE 802.11, 2001.

35. H.S.Chhaya, and S.Gupta, "Performance and Modeling of Asynchronous Data Transfer Methods of IEEE 802.11 MAC Protocol", ACM/Baltzer Wireless Networks Volume 3, pp 217-234, August 1997.

36. P. Karn, "MACA - A New Channel Access Protocol for Packet Radio", Proceedings of the ARRL/CRRL Amateur Radio Ninth Computer Networking Conference, Ontario, Canada, September 1990.

37. V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A Media Access Protocol for wireless LAN's", Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM '94) , London, U.K., August 1994.

38. D. C. Plummer, " An Ethernet Address Resolution Protocol: or Converting Network Protocol addresses to 48-bit Ethernet Addresses for Transmission on Ethernet Hardware", Network Working Group, RFC 826, November 1982.

39. G.R. Wright and W.R. Stevens, "TCP/IP Illustrated, Volume 2:The Implementation", Addison-Wesley, Boston, October1995.

40. S.R. Das, R. Castaneda, J. Yan, and R. Sengupta, "Comparative Performance Evaluation of routing Protocols for Mobile Adhoc Networks", Proceedings of the 7th International Conference on Computer Communications and Networks (IC3N), Lafayette, Louisiana, October 1998.

41. J. Broch, D. A. Maltz, D. B. Johnson, Y-C. Hu, and J. Jetcheva, " A Performance Comparison of Multi-hop Wireless Ad hoc Network Routing Protocols" Proceedings of the 4th International Conference on Mobile Computing and Networking (ACM MOBI-COM'98), October 1998.

42. S.R.Das, C.E. Perkins, E.M.Royer, and M.K.Marina, "Performance Comparison of Two On-Demand Routing Protocols for Adhoc Networks", IEEE Personal Communications, pp-16-28, February 2001.

43. M.Bansal, and G.Barua, "Performance Comparison of Two On-Demand Routing Protocols for Adhoc Networks", Proceedings of IEEE Conference on Personal and Wireless Communications (ICPWC-2002), New Delhi, India, pp196-200, December 2002.

44. A.S. Tannenbaum, "Computer Networks", PHI, 1999.

45. J.Postel, "User Datagram Protocol (UDP)", Network Working Group, RFC768, August 1980.

46. K.R. Anupama, and S.Balasubramanian, "Routing Criteria for Mobile Adhoc Networks", Proceedings of National Conference on Advances in Computer Communication Networks (CCN 2004), IIT Roorkee, February 2004.

47. S-J. Lee, and M. Gerla, "Dynamic Load-Aware Routing in Adhoc Networks", Proceedings of the Third IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET 2000), Boston, Massachusetts, 2000.

48. L. Klienrock, and J. Silvester, "Optimum Transmission Radii for Packet Radio Networks or Why Six is a Magic Number", Proceedings of IEEE national Telecommunications Conference, Alabama, December 1978.

49. K.R. Anupama, and S.Balasubramanian, "A Multicast Protocol for Mobile Adhoc Networks", Proceedings of IEEE Conference on Personal and Wireless Communications (ICPWC-2002), New Delhi, India, pp187-191, December 2002.

50. E.M. Royer, and C.E. Perkins, "Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing", draft-ietf-manet-maodv-00.txt, 15 July 2000.

51. J.G. Jetcheva, Y-C Hu, D.A. Maltz, David B. Johnson, " A Simple Protocol for Multicast and Broadcast in Mobile Ad Hoc Networks", draft-ietf-manet-simple-mbcast-00.txt, 17 November 2000.

52. T.Imielinski, and J.Navas, "GPS based Addressing and Routing", Network Working Group, RFC 2009, November 1996.

53. T.Imielinski , and J.Navas, "Geographical addressing and Routing", Proceedings of ACM/IEEE Mobicom97, Budapest, Hungary, September 1997.

54. Y.B.Ko and N.H.Vaidya, "Geocasting in MANET- Location based Multicast Algorithm", Proceedings of IEEE Workshop on Mobile Computing systems and Applications, New Orleans, Louisiana, February 1999.

55. M. Jiang, J. Li, and Y.C. Tay, "Cluster Based Routing Protocol", draft-ietf-manet-cbrp-spec-01.txt, August 1999.

56. Z. J. Haas, and M.R. Pearlman, "The Zone Routing Protocol (ZRP) for Adhoc Networks", draft-manet-zone-zrp-03.txt, March 2000.

57. Z. J. Haas, and M.R. Pearlman, "The Inter-zone Routing Protocol (IERP) for Adhoc Networks", draft-ietf-manet-zone-ierp-01.txt, June 2001.

58. Z. J. Haas, and M.R. Pearlman, "The Intra-zone Routing Protocol (IARP) for Adhoc Networks", draft-ietf-manet-zone-iarp-01.txt, June 2001.

59. J. Haas, and M.R. Pearlman, "The Bordercast Routing Protocol (BRP) for Adhoc Networks", draft-ietf-manet-zone-iarp-01.txt, June 2001.

List of Publications

1. K.R. Anupama, and S.Balasubramanian, "A Multicast Protocol for Mobile Adhoc Networks", Proceedings of IEEE Conference on Personal and Wireless Communications (ICPWC-2002), New Delhi, India, pp187-191, December 2002.
2. K.R. Anupama, and S.Balasubramanian, "GPS-based Predictive Energy Aware Routing Protocol (GPEAR)", Proceedings of the National Conference on Communication 2003, IIT Madras, pp 473-478, February 2003.
3. K.R. Anupama, and S.Balasubramanian, "GPS-based Predictive Energy Aware Routing Protocol (GPEAR)", Proceedings of the International Multi-Conference of Systematics, Informatics and Cybernetics 2003, Orlando, USA, July 2003.
4. K.R. Anupama, and S.Balasubramanian, "Routing Criteria for Mobile Adhoc Networks", Proceedings of National Conference on Advances in Computer Communication Networks (CCN 2004), IIT Roorkee, February 2004.
5. K.R. Anupama, and S.Balasubramanian, "GPS-based Predictive Energy Aware Routing Protocol (GPEAR)", Proceedings of the International Conference on Computing, Communications and Control Technologies (CCCT' 04), University of Texas, Austin, pp 295-299, August 2004.

Brief Biographical Sketch of the Supervisor

Prof. S. Balasubramanian completed his Bachelor's and Master's from the EEE Department of BITS, Pilani in the year 1970 and 1972 respectively. After his Post-graduation from BITS, Pilani in 1972, he joined the Microwave Division of the Indian Space Research Organisation (ISRO), SAC, Ahmedabad as a Microwave Engineer. He developed the space-qualified mixer/pre-amplifiers at 19,22 & 31 GHz as well as carried out the system Integration and environmental qualification of the Satellite Microwave Radiometer (SAMIR) Payload on-board BHASKARA – I & II satellites. SAMIR was India's First Microwave Payload to send remotely sensed data from the BHASKARA Satellites. He was member of the launch team for BHASKARA – I & II satellites from Volgograd, Russia.

In 1984 he went to ISAC, Bangalore where he was designated Project Manager for the checkout of INSAT-2 Series payloads. He developed the Automated Communication Transponder Checkout System for the Characterization of INSAT Communication Transponders. He was also the Head of the Data Reception & Transmisssion Section of the Space Ground Checkout Division of ISAC. He was member of the launch team for the launch of INSAT-2A from Kourou, French Guiana, in July 1992.

Subsequently, he joined academics and was with Bangalore University for a few years. He joined the EEE Group of BITS, Pilani in November 1998 as Associate Professor. He took over as the Group Leader of the EEE Group since April 2000. An M.E. (Communication Engg.) programme was started in July 2000. He teaches courses on communication systems (analog & digital), signals & systems, satellite communication, wireless and mobile Communication, mobile telecom networks, RF Micro-electronics etc. He has setup the communication laboratory, RF & Microwaves laboratory and the Digital Signal Processing laboratory.

He is one of the Research Advisors for the Research Projects undertaken by BITS, under the BITS-NOKIA Research Collaboration. The project involves the development of Source Routing Protocols for Mobile Ad hoc Networks and other QOS aspects in mobile and wireless networks.

He is Life Fellow of IETE (F108307) and Life Member of ISTE. He is member of the Screening-cum-Technical Evaluation Committee for the National Awards for R & D efforts in Industry for the year 2002, 2003 and 2004.

He is a member of the Specialist Review Committee for the Payloads of GSAT-3 and INSAT-4 Satellites and Member of the Specialist Review Committee for Electrical Ground Support Systems for GSAT-4.

Brief Biographical Sketch of the Student

Ms. K.R.Anupama completed her Bachelor's Degree in Electronics and Communication Engineering from Madras University in the year 1994 and her Master's Degree in Electronics and Control Engineering from BITS, Pilani in the year 1998. She joined EEE group of BITS, Pilani in May 1999. She commenced her doctoral work in August 2000.

She is one of the Research Fellows for the research project undertaken by BITS under the BITS-NOKIA Research Collaboration.