PROTOCOL FOR REDUCING COMMUNICATION DELAY IN MOBILE AD HOC NETWORKS BY PRE-FETCHING LOCATION INFORMATION

By

MUHAMMAD TANVIR ALAM

Bachelor of Science

North South University

Dhaka, Bangladesh

1999

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2002

PROTOCOL FOR REDUCING COMMUNICATION DELAY IN MOBILE AD HOC NETWORKS BY PRE-FETCHING LOCATION INFORMATION

Thesis Approved:

[illegible]

Thesis Advisor

J Chandler

[illegible]

Timothy J. Pettibone

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Johnson Thomas for his careful supervision, constructive guidance, inspiration and friendship. My sincere appreciation extends to Dr. J. P. Chandler and Dr. N. Park whose guidance, assistance, encouragement, and friendship are also invaluable.

I would also like to give my special appreciation to my father Eng. Mukarrmaul Bari and my mother Umme Fatema for their precious suggestions to my research, their strong encouragement at times of difficulty, love, and understanding throughout my research endeavor. Their blessings and strong eagerness to attend my convocation from far distance encouraged me to finish this paper soon.

Finally, I would like to thank the members of the Department of Computer Science for their support during the two years of my study.

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Johnson Thomas for his careful supervision, constructive guidance, inspiration and friendship. My sincere appreciation extends to Dr. J. P. Chandler and Dr. N. Park whose guidance, assistance, encouragement, and friendship are also invaluable.

I would also like to give my special appreciation to my father Eng. Mukarrmaul Bari and my mother Umme Fatema for their precious suggestions to my research, their strong encouragement at times of difficulty, love, and understanding throughout my research endeavor. Their blessings and strong eagerness to attend my convocation from far distance encouraged me to finish this paper soon.

Finally, I would like to thank the members of the Department of Computer Science for their support during the two years of my study.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

NOMENCLATURE

PFP	Pre-fetching Protocol
DSR	Dynamic Source Routing
DSDV	Distance-Sequenced Distance-Vector
MACT	Multicast Activation
LAR	Location Aided Routing
ZRP	Zone Routing Protocol

CHAPTER I

INTRODUCTION

Mobile and Ad hoc networks consist of mobile hosts that communicate with each other, in the absence of a fixed infrastructure. Route between two hosts in a Mobile Ad hoc Network (MANET) may consists of hops through other hosts in the network. In other words, a MANET is defined as a collection of mobile platforms or nodes where each node is free to move about arbitrarily. Each node logically consists of a router that may have multiple hosts and that also may have multiple wireless communications devices. Host mobility can cause frequent unpredictable topology changes. Therefore, the task of finding and maintaining routes in MANET is nontrivial. Many protocols have been proposed for mobile Ad hoc networks, with the goal of achieving efficient routing.

However, none of the existing protocols reduces the communication delay among nodes to zero. There is a trade off between delay and information overhead for those so far proposed protocols. In this research, we suggest an approach to decrease delay of route discovery almost to zero by utilizing pre fetching of location information for mobile hosts and deriving route tables for nodes in advance. We demonstrate how location information may be used for route generation in expense of overhead and memory. Our protocol allows the network to be completely self-organizing and self-configuring.

CHAPTER II

REVIEW OF LITERATURE

Packets are transmitted between the nodes of the network using route tables stored at each node. Each route table, at each of the nodes, lists all available destinations and the number of hops required. Each route table entry is tagged with a sequence number/installation time that is originated by the destination node. We may take a look at the different protocols and routing methods that have been proposed to perform this type of packet transmission.

2.1 Distance-Sequenced Distance-Vector protocol

The Distance-sequenced distance-vector (DSDV) protocol was first proposed in 1996 by Perkins and Bhagwat. The DSDV protocol requires each mobile node to advertise, to each of its current neighbors, its own route table (for instance, by broadcasting its entries). The entries in this list may change fairly dynamically over time, so the advertisement must be made often enough to ensure that every mobile computer can almost always locate every other mobile computer in the collection. In addition, each mobile computer agrees to relay data packets to other computers upon request. "The data broadcast by each mobile computer will contain its new sequence number and the following information for each new route:

- The destination's address
- The number of hops required to reach the destination

CHAPTER II

REVIEW OF LITERATURE

Packets are transmitted between the nodes of the network using route tables stored at each node. Each route table, at each of the nodes, lists all available destinations and the number of hops required. Each route table entry is tagged with a sequence number/installation time that is originated by the destination node. We may take a look at the different protocols and routing methods that have been proposed to perform this type of packet transmission.

2.1 Distance-Sequenced Distance-Vector protocol

The Distance-sequenced distance-vector (DSDV) protocol was first proposed in 1996 by Perkins and Bhagwat. The DSDV protocol requires each mobile node to advertise, to each of its current neighbors, its own route table (for instance, by broadcasting its entries). The entries in this list may change fairly dynamically over time, so the advertisement must be made often enough to ensure that every mobile computer can almost always locate every other mobile computer in the collection. In addition, each mobile computer agrees to relay data packets to other computers upon request. "The data broadcast by each mobile computer will contain its new sequence number and the following information for each new route:

- The destination's address
- The number of hops required to reach the destination

- The sequence number of the information received regarding that destination, as originally stamped by the destination." (Perkins and Bhagwat, 1996).

The addresses stored in the route tables will correspond to the layer at which the DSDV Ad hoc networking protocol is operated. That is, operation at layer 3 will use network layer addresses for the next hop and destination addresses; operation at layer-2 medium access control (MAC) addresses.

However, the delay is found because of the advertisement of each node to its neighboring nodes. Also, the nodes have to wait for a settling time before they communicate in order to take care of the damping fluctuations. A faster routing method is possible to come up with the help of the pre fetched route tables.

2.2 The Dynamic Source Routing Protocol

In 1998 Johnson, Broch and Maltz proposed the Dynamic Source Routing (DSR) protocol for multi hop wireless Ad hoc Networks. The protocol is composed of the two mechanisms of Route Discovery and Route Maintenance, which work together to allow nodes to discover and maintain source routes to arbitrary destinations in the Ad hoc Networks. The use of source routing allows packet routing to be trivially loop free, avoids the need for up-to-date routing information in the intermediate nodes through which packets are forwarded, and allows nodes that are forwarding or overhearing packets to cache the route information in them for their own future use. All aspects of the protocol operate entirely on demand, allowing the routing packet overhead of DSR to scale automatically to only that needed to react to changes in the routes currently in use.

Thus, DSR has very low routing overhead and is able to correctly deliver almost all originated data packets, even with continuous, rapid motion of all nodes in the network.

2.2.1 Route discovery

Figure 1 illustrates an example of Route Discovery, in which node A is attempting to discover a route to node E. To initiate the Route Discovery, A transmits a Route Request message as a single local broadcast packet, which is received by (approximately) all nodes currently within wireless transmission range of A. Each Route Request message identifies the initiator and target of the Route Discovery and also contains a unique request ID, determined by the initiator of the Request. Each Route Request also contains a record listing the address of each intermediate node through which this particular copy of the Route Request message has been forwarded. This route record is initialized to an empty list by the initiator of the Route Discovery.

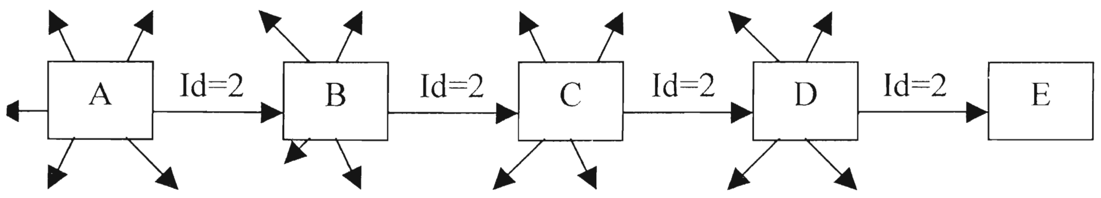

Figure 1: Route Discovery Example with node A as the initiator and node E as the target

When another node receives a Route Request, if it is the target of the Route Discovery it returns a Route Reply message to the Route Discovery initiator, giving a copy of the accumulated route record from the Route Request; when the initiator receives this Route

Reply, it catches this route in its Route Cache for use in sending subsequent packets to this destination. Otherwise, if the node receiving the Route Request recently saw another Route Request message from this initiator bearing this same request ID, or if it finds that its own address is already listed in the route record in the Route Request message, it discards the Request. If not, this node appends its own address to the route record in the Route Request message and propagates it by transmitting it as a local broadcast packet (with the same request ID).

2.2.1 Route Maintenance

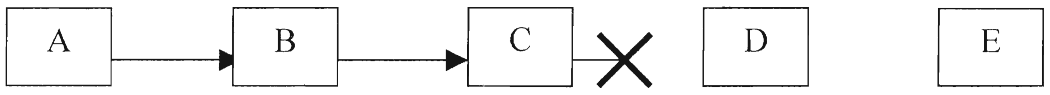

Figure 2: Route Maintenance Example Node C is unable to forward a packet from A to E over its link to the next hop, D

In case of route Maintenance, if C in Figure 2 is unable to deliver the packet to the next hop D, C returns a Route Error to A, stating that the link from C to D is currently "broken." Node A then removes this broken link from its cache, and any retransmission of the original packet is a function for upper-layer protocols such as TCP. For sending such a retransmission or other packets to this same destination E, if A has in its Route Cache another route to E, it can send the packet using the new route immediately. Otherwise, it may perform a new Route Discovery for this target.

So, we may observe that the Route discovery is on demand and the overhead is reduced almost to zero. Still, a node should delay sending its own Route Reply for a short period and listen to see if the initiating node begins using a shorter route first. Moreover, the Route Request and Reply process of DSR may give birth to a significant amount of routing delay.

2.3 The Ad Hoc On-Demand Distance-Vector Protocol

The Ad Hoc On-Demand Distance-Vector (AOVD) routing protocol discovered by Perkins and Royer in 1998, provides quick and efficient route establishment between mobile nodes desiring communication with minimal control overhead. The initial design of AODV was undertaken after the experience with the Destination-Sequenced Distance-Vector (DSDV) routing algorithm. Its goal is to reduce the need for system-wide broadcasts to the furthest extent possible. DSDV issues broadcasts to announce every change in the overall connectivity of the Ad hoc Network. Whenever two nodes enter communication range of each other, they become neighbors and change the network topology. This triggers a broadcast of the new connectivity information to the rest of the nodes in the Ad hoc Network. Similarly when two neighboring nodes drift out of direct communication range, the link break causes a broadcast-triggered update. With AODV it is no longer required that such changes initiate system wide broadcasts. In fact, if the link status does not affect ongoing communication or multicast tree maintenance, no broadcast occurs. This localizes the effects caused by local movements. In DSDV, local movements have global effects. In AODV, the only non-local effects result from a distant source trying to use a broken link. The triggered broadcast in DSDV is replaced by more careful bookkeeping that identifies the one or more nodes that had been using the broken link. Only those nodes have to be informed of the link's changed status. In the frequent case that a link was idle, no such indication is sent.

Another feature distinguishing AODV from DSDV and other Ad hoc protocols is its integrated handling of multicast routing. By modeling the multicast routing problem as an extension of AODV's distance-vector algorithm, protocol improvement may be applied to both unicast and multicast at the same time.

2.3.1 Route discovery

"The basic outline of the route discovery process of AODV for unicast routing is as follows:

- When a node needs a route to a destination, it broadcasts a Route Request.
- Any node with a current route to that destination can unicast a Route Reply back to the source node.
- Route information is maintained by each node in its route table.
- Information obtained through Route Request and Route Reply messages is kept with other information in the route table.
- Sequence numbers are used to eliminate stale routes.
- Routes with old sequence numbers are aged out of the system." (Perkins and Royer, 1998).

2.3.2 Route maintenance

When either the destination or some intermediate node moves, a Route Error message is sent to the affected source nodes. This Route Error is initiated by the node upstream of the break. It lists each of the destinations that are now unreachable because of the loss of the link. When the neighbors receive the Route Error, they mark their route to the destination as invalid by setting the distance to the destination equal to infinity and, when a source node receives the Route Error, is can reinitiate route discovery if the route is still needed.

Multicast route discovery follows directly from the unicast route discovery in that it utilizes the same two messages types (Route Request and Route Reply) for the route request/reply discovery cycle. Multicast group membership is dynamic; nodes are able to join and leave the group at any time.

Multicast Route activation/deactivation i.e. node joining/leaving a network is taken care by the Prune flag and multicast activation (MACT) message propagation among nodes. If a node wishes to revoke its member status or leave a particular network, unicasts a MACT message with the Prune flag set to its next hop. It then deletes the multicast group information from its multicast route table. When the next hop receives the prune message, it deletes the next hop information for the sending node and so on. However, the flag setting and reconfiguring the table may slightly slow down the routing method as the nodes move.

2.4 Zone Routing Protocol

The Zone Routing Protocol (ZRP) was introduced by Haas and Pearlman in 1998 for routing in re-configurable Ad hoc Networks (RWNs). ZRP dynamically adjusts itself to operational conditions by sizing a single network parameter – the zone radius. More specifically, ZRP reduces the cost of frequent updates to the constantly changing network topology by limiting the scope of the updates to the immediate neighborhood of the change. ZRP works between proactive and reactive routing methods. Routing protocols can be classified as either 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. In reactive protocols (that work opposite of proactive protocols), because route information may not be available at that time a Route Request is received, the delay before a route is determined can be quite significant. Furthermore, the reactive global search procedure requires significant control traffic.

ZRP limits the scope of the proactive procedure to the node's local neighborhood, but the search throughout the network, although global, is done by querying only a subset of the network nodes.

An example of the route discovery of ZRP is shown in Figure 3. Source node S sends a packet to destination D. To find a route within the network, S first checks whether D is within its routing zone. Because D does not lie within S's routing zone, S bordercasts a route request to all of its peripheral nodes - that is to nodes C, G and H. Nodes C, G and H then determine that D is not in their routing zones and therefore bordercast the request to their peripheral nodes. One of the H's peripheral nodes, B, recognizes D as being in its routing zone and responds to the Route Request, indicating the forwarding path S→H→B→D.

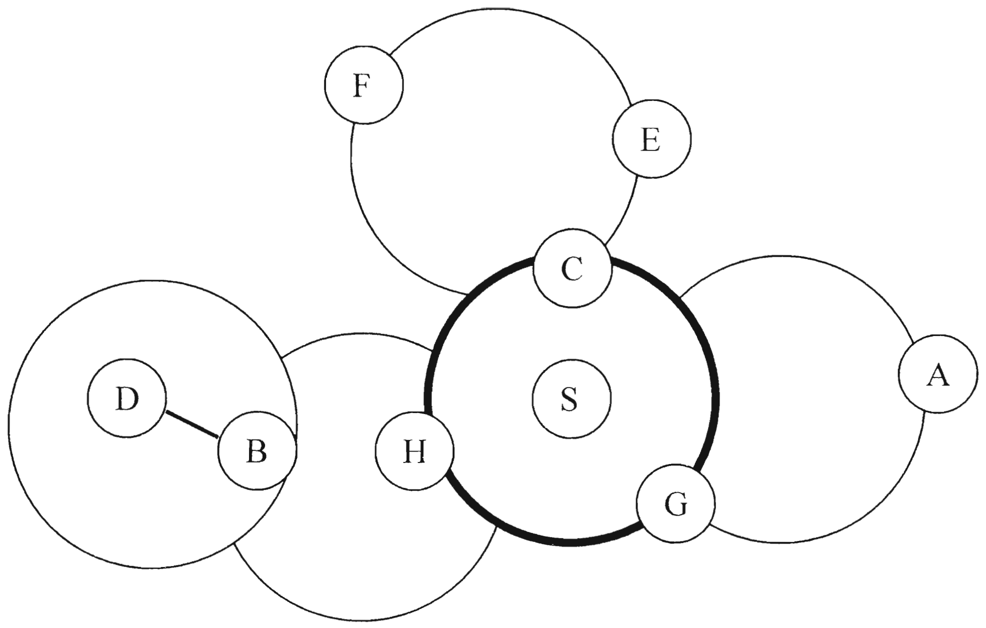

Figure 3: An example of route discovery operation

Thus, this protocol allows a node to recognize all its neighboring nodes within a fixed a radius. This process may reduce the communication delay compared to the DSR protocol since it operates at the peripheral nodes immediately when the destination does not lie within the sub-network. However, the Route Request and Reply process may still hold the delay period.

2.5 Location-Aided Routing

Vaidya and Ko came up with Location-Aided Routing (LAR) protocol to decrease overhead of the flooding algorithm for MANET. Flooding, basically is the forwarding of the Route Request for destination by a node to its neighbors. In LAR method, it is assumed that the average speed and position at a particular time of each node is known. With this information, the method utilizes the Expected zone and Request zone to find a destination node and reduce overhead.

2.5.1 Expected zone

Let us consider a node S that needs to find a route to node D. S knows that node D was at location L at time t_0, and the current time is at t_1 Then, the "expected zone" of D, from the viewpoint of node S at time t_1, is the region that node S expects to contain D at time t_1. Node S can determine the expected zone based on the knowledge that node D was at Location L at time t_0. For instance, if node S knows that node D travels with average speed v, then S may assume that the expected zone is the circular region of radius $v(t_1-t_0)$, centered at location L (refer to Figure 4a). If node S does not know a previous location of node D, then node S cannot determine expected zone and the entire region occupied by the MANET is assumed to be the expected zone. In this case, the algorithm reduces to the basic flooding algorithm. In general, having more information regarding mobility of a destination node, can result in a smaller expected zone. For example, if S knows that destination D is moving north/south, then the circular zone in Figure 4a can be reduced to a semi-circle (upper/lower), as in Figure 4b.

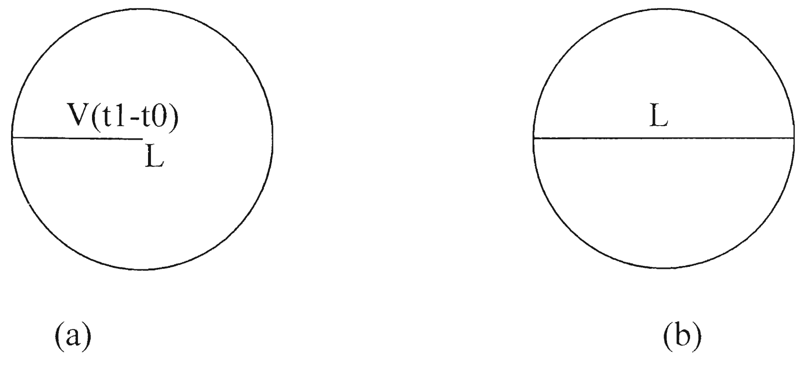

Figure 4: Examples of expected zone

2.5.2 Request Zone

The request zone is the smallest area that includes current location of source S, and the expected zone. The size of the request zone is proportional to average speed of movement v, and time elapsed since the last known location of the destination was recorded. The sender comes to know location of the destination only at the end of a route discovery. A source node forwards a route request to the neighboring node only if this neighboring node belongs to the request zone. In Figure 5, S will not send route request to A for destination D since A is outside the request zone.

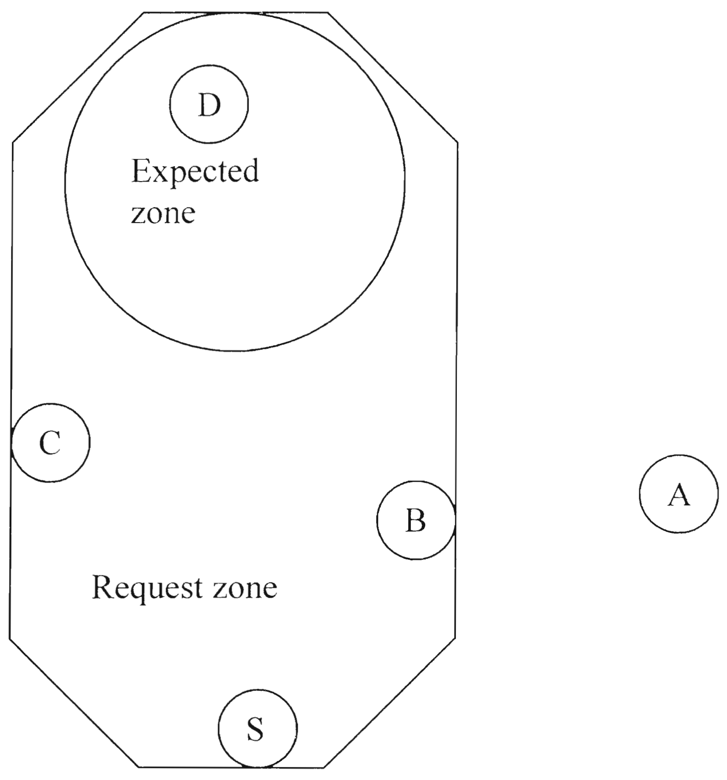

Figure 5: Request zone for source S and destination D

CHAPTER III

THESIS OBJECTIVES

Objectives of the thesis are:

1. To propose a new self-configuring protocol for MANET
2. To reduce the communication delay among nodes by pre-fetching of routing tables
3. To compare the protocol with other protocols

CHAPTER IV

DESIGN APPROACH

Our goal is to provide a protocol that would drop the delay sharply. We propose a scenario where our protocol would work smooth.

4.1 Assumption

The following are the assumptions for our protocol:

1. The average speed and location of each node are known.
2. The traveling direction of each node movement is known
3. Each node has cache memory to hold the table structure (similar to DSDV protocol).

The *table structure* of a node would consist of the following columns:

a) The destination node,

b) Number of hops/metric required to reach the destination.

c) Next hop/node,

d) Location of the destination,

e) Installation time of the table structure,

f) The shortest distance from the source to the destination and the

g) Sequence number of each message.

We use the fixed radius process of ZRP to calculate the number of hops for a destination node. If a destination node is far away from the source node and whose path is not known would be marked as $\propto$ for that particular destination node. Since we know the path and average speed of each node, we may calculate the position of a node, for instance X after every unit interval and check whether any node belongs to its radius. If such node(s) exist(s) and if the location/routing table of that node(s) can be pre-fetched by X, then it can enjoy zero latency in the next interval or near future to communicate them. The derivation of the tables of the probable pre-fetched nodes would follow the table of the nearer nodes of X that could communicate with the probable pre-fetching nodes.

4.2 Restrictions and suggested solutions

Our proposed pre-fetching protocol (PFP) may have several limitations. It can be observed that the protocol would work best if the distribution of the nodes is uniform over the network. The requirement of high call per mobility (CMR) ratio is also a disadvantage for the protocol. There is a lot of calculation involved that are to be carried out after each interval as X travels, producing high overhead. Thus, each node has to have a huge cache available. If a node does not have any fixed moving pattern and it takes the shape of loops after every while or if it moves back wards, the pre-fetched table structure may be repeated and the overhead would increase sharply. In this case, we would switch from our protocol to any of the above-mentioned protocols of the “Review of Literature” section. Moreover, if the node mobility rate is fairly high and a node leaves the area in a very short period, the usage of the derived table would not be a good choice. We may

In this system, the total cost of requesting a node for table is the sum of the system resource cost and the delay cost of waiting for the table to arrive. Therefore, if a node tries to pre-fetches a route table, the total cost of retrieving this table is

$c_1 = \alpha_B * s + \alpha_T * (t + t_0) = \alpha_B * s + \alpha_T * (s/b + t_0)$

where s is the table size in bytes, t is the transmission time, t_0 is the start-up time, and b is the capacity of the path to the server (packets/time unit). Assuming the cost of using a table on the local node is negligible, if the requested table had previously been pre-fetched and saved on the local cache of a node, then the cost associated with this request equals the system resource cost, i.e.

$c_2 = \alpha_B * s$

because the delay cost is zero.

We now investigate the situation in which a node has just started using a new table, and currently there are L distinct nodes in the network (whose route tables need to be pre-fetched) with access/pre-fetching probability (defined later) greater than zero. The average cost C of satisfying all nodes requests stemming from this new route table being used without any pre-fetching is the summation of the costs to retrieve each of the tables of L nodes times the access probability of the corresponding node, i.e.

$C = \sum_{i=1}^{L} \quad p_i * [\alpha_B * s_i + \alpha_T * (s_i/b_i + t_0)]$

where p_i (defined later) is the access probability of node i, t_{0i} is the start-up time for retrieving the routing table of ith node, and b_i is the capacity of the path to the server which contains the ith node.

If instead, m node-tables among all of the L candidate nodes are pre-fetched, say $i_1, i_2, \ldots, i_m$, this average cost becomes

$\sum_{j=1}^{m} \alpha_B * s_{ij} + \sum_{j=m+1}^{L} p_{ij} *[\alpha_B * s_{ij} + \propto_T *(t_{ij} + t_{0ij})]$

ere $i_j \in \{ 1,\ldots, L\}, j = 1..L$, and for any $j_1 \neq j_2, i_{j1} \neq i_{j2}$.

omparing the above two equations, we conclude that the average cost is minimized if nd only if we pre-fetch all and only those route tables of nodes which satisfy

$p_i > 1/((1+ s_0/s_i) * \alpha_T/(\alpha_B * b_i) +1)$

where $s_{0i} = b_i * t_{0i}$ and b_i is the capacity of the path to server i. Thus, we define the pre-fetch threshold

$H = 1/((1+ s_0/s_i) *\alpha_T/(\alpha_B * b_i) +1)$.

We came up with a prediction module that can be compared to the threshold probability to determine whether the information of the probable nodes would be pre-fetched. Each node would be pre-fetching the information of those nodes, whose access probably is greater than H. In this way, we may optimize cost and each node may enjoy zero latency to communicate among nodes that lie ahead of it.

Let the access probability of each probable node (whose location/routing table is a candidate to be pre-fetched/derived) is as follows:

Probability (of a probable pre-fetching node) = min (1, Z/Y), where Y is the number of nodes that lie in the probable pre-fetching range and not in the previous range where the moving node lies. Z is the number of those nodes that lie in the same range as the moving node lies and that are in the range of the particular probable pre-fetching node.

This algorithm may exceed the value, one often because, there may be more nodes in the current range of a moving node, than in the predicted range at near future. The more nodes from previous range of a moving node can communicate to any ır node of the predicted range, the higher the possibility of the moving node to nicate with that particular probable pre-fetching node in near future/next time ıl. We are varying Z in terms of nodes that have access to the probable pre-fetching ;. If Z/Y exceeds one, we take the minimum of 1 and Z/X, which simply implies that cular node-information would be pre-fetched. Thus a node would pre-fetch those es' information whose access probability is greater than H. This scenario is best ıted if we know the moving direction and average speed of all nodes in the MANET. owever, a node may change its direction all of a sudden, take a turn and it might not be n the place where we would expect it to be in the next time interval. The probable pre-fetching nodes would loose significance to be pre-fetched in such cases. In order to overcome this, we introduce a prediction probability P_d that would be multiplied with the access probability. Thus the actual access/pre-fetching probability is (P_d * Z/Y) where P_d is the predicted probability of changing direction of a moving node. If a moving node never changes speed and direction, the predicted probability of this node P_d is 100%. This ensures that it would be exactly at the position where it is expected to be in the near future. If each node in the MANET uses this technique, they may enjoy zero latency with optimized cost. The "Implementation approach" section describes a complete situation with an example.

CHAPTER V

IMPLEMENTATION APPROACH

We consider only one moving node, say X and other nodes to be stagnant for the rpose of implementation. We also assume that the traveling path of X is a straight line r the simplicity of the implementation. The initial position of the nodes is taken from he user. However, our pre-fetching strategy works for any MANET where all nodes are moving and even the nodes do not have any specific traveling pattern. A concrete example of seven nodes implementation is illustrated next. Each node needs to apply the same strategy to achieve the maximum outcome.

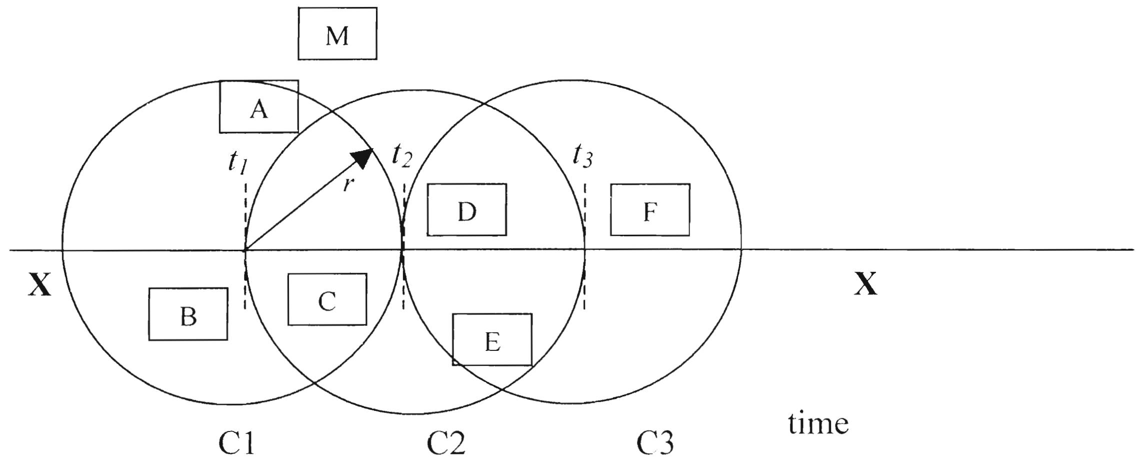

Figure 6: An example of the protocol

In Figure 6, let node X is traveling in a straight line from left to right. Let r be the communication radius range of X. At time t_1, X can communicate with nodes A, B and C.

CHAPTER V

IMPLEMENTATION APPROACH

We consider only one moving node, say X and other nodes to be stagnant for the rpose of implementation. We also assume that the traveling path of X is a straight line r the simplicity of the implementation. The initial position of the nodes is taken from he user. However, our pre-fetching strategy works for any MANET where all nodes are moving and even the nodes do not have any specific traveling pattern. A concrete example of seven nodes implementation is illustrated next. Each node needs to apply the same strategy to achieve the maximum outcome.

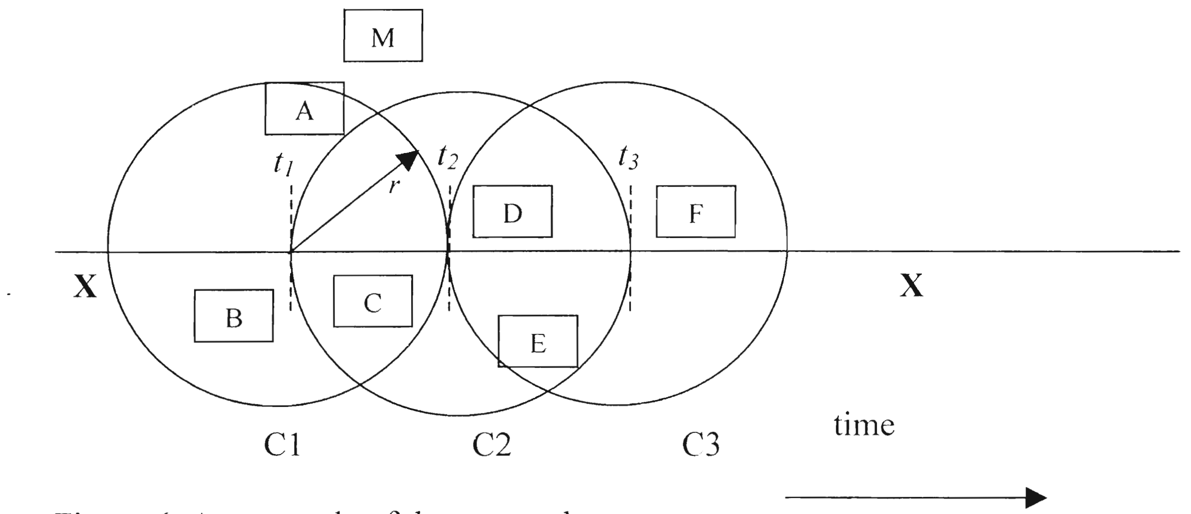

Figure 6: An example of the protocol

In Figure 6, let node X is traveling in a straight line from left to right. Let r be the communication radius range of X. At time t_1, X can communicate with nodes A, B and C.

ɪraveling in a straight line, at time t_2, X can communicate with nodes C, D and E. t_3, X can communicate with nodes D, E and F. let the location of node A is (x1, is (x2, y2), C is (x3, y3), D is (x4, y4), E is (x5, y5), F is (x6, y6), M is (x7, y7) ode X at time t_1 is (x8, y8) and at time t_2 is (x9, y9). When X is at position (x8, y8), robable pre-fetching range is the circle C2. Similarly, when X is at position (x9,y9), probable pre-fetching range is the circle C3.

In the normal protocols, X needs to have enough information at time t_1 to route ıckets through nodes A, B and C. This involves communications (for example, Route equest and Reply) with A, B and C, to derive a routing table for X, thereby incurring delay. Similarly, at time t_2, X needs to have enough information to route packets through nodes C, D and E. Again this involves communications (for instance, Route Request and Reply) with C, D and E, to derive a routing table for X, and thereby incurring delay.

However, in our pre-fetching protocol (PFP), X needs the following at time t_1:

- X needs to have enough information to route packets through nodes A, B and C.
- X pre-fetches addresses used by D and E so that when X gets to time t_2, X has enough information to route packets through nodes C, D and E immediately.

Similarly, X needs the following at time t_2:

- X needs to route packets through nodes D, E and F.
- X pre-fetches addresses used by F so that when X gets to time t_3, X has enough information to route packets through nodes D, E and F immediately.

As mentioned in the assumptions, we need to know which direction X is traveling. ır example, if X was heading North-East at time t_1, pre-fetching the addresses of M and ı would make more sense (rather than E). The routing cache may be divided into two ɔarts, current routing table and pre-fetch routing table. Some possible tables of different nodes at different times are furnished below:

Destination	Metric	Next hop	Location	Distance
B	1	B	x2, y2	AB<r
C	1	C	x3, y3	AC<r
D	1	D	x4, y4	AD<r
E	$\propto$		x5, y5	AE>r
F	2	D	x6, y6	AF>r
M	1	M	x7, y7	AM<r
X	1	X	x8, y8	AX<r

We assume that previously, node A communicated with node F via node D, but it did not communicate with node E.

Table 1: Current Routing table of A at time t_1

Destination	Metric	Next hop	Location	Distance
A	1	A	x1, y1	DA<r
C	1	C	x3, y3	DC<r

As mentioned in the assumptions, we need to know which direction X is traveling. r example, if X was heading North-East at time t_1, pre-fetching the addresses of M and would make more sense (rather than E). The routing cache may be divided into two arts, current routing table and pre-fetch routing table. Some possible tables of different nodes at different times are furnished below:

Destination	Metric	Next hop	Location	Distance
B	1	B	x2, y2	AB<r
C	1	C	x3, y3	AC<r
D	1	D	x4, y4	AD<r
E	∝		x5, y5	AE>r
F	2	D	x6, y6	AF>r
M	1	M	x7, y7	AM<r
X	1	X	x8, y8	AX<r

We assume that previously, node A communicated with node F via node D, but it did not communicate with node E.

Table 1: Current Routing table of A at time t_1

Destination	Metric	Next hop	Location	Distance
A	1	A	x1, y1	DA<r
C	1	C	x3, y3	DC<r

C	1	C	x3,y3	XC<r
D	$\propto$		x4,y	XD>r
E	$\propto$		x5,y5	XE>r
F	3	A, D	x6,y6	XF>r
M	$\propto$		x7,y7	XM>r

We assume that node X previously communicated with node F via node A and D, but it did not communicate with node E.

Table 4: Current routing table of X at time t_1

From the direction and location information, X can derive that it needs to pre-fetch the tables of D and E. However X cannot directly get the tables from E because at t_1 X cannot communicate directly with E. It can however derive at least part (if not the whole) of the table of D and E. For example, from A's table X knows that a packet to D is routed directly, and a packet to F is routed via node D. Similarly, from C's table X knows that a packet to E is routed directly. Therefore the pre-fetched table for X at t_1 would look like this:

Destination	Metric	Next hop	Location	Distance
A	2	D	x1,y1	XA>r
B	2	C	x2,y2	XB>r
C	1	C	x3,y3	XC<r
D	1	D	x4,y	XD<r
E	1	E	x5,y5	XE<r

F	2	D	x6,y6	XF>r
M	$\propto$		x7,y7	XM>r

Table 5: Pre-fetch routing table of X at time t_1

Therefore when X gets to t_2, it can immediately route a message to A, B, C, D and E – even though it has not got any tables or information from D or E. When X gets to t_2, a merging of the current and pre-fetch table stake place – this is the table updating process.

Destination	Metric	Next hop	Location	Distance
A	1	A	x1, y1	DA<r
C	1	C	x3, y3	DC<r
E	1	E	x5, y5	DE<r
F	1	F	x6, y6	DE<r
M	1	M	x7, y7	DM<r
B	$\propto$		x2, y2	DB>r
X	*1*	*X*	*X8, y8*	*DX<r*

Updated rows are italic

Table 6: Routing table of D At time t_2

Destination	Metric	Next hop	Location	Distance
A	*2*	*D*	*x1, y1*	*XA>r*

C	1	C	x3, y3	XC<r
E	*1*	*E*	*x5, y5*	*XE<r*
F	*2*	*D*	*x6, y6*	*XF>r*
M	*2*	*D*	*x7, y7*	*XM>r*
B	*2*	*C*	*x2, y2*	*XB>r*
D	*1*	*D*	*x4, y4*	*XD<r*

Updated rows are italic

Table 7: Current routing table of X at time t_2

The other nodes except node C need to be updated in the similar fashion since X has moved to circle C2. The pre-fetching table of node E by X and updating the table of node E would work in a similar fashion. Thus, now X can route packets through C, D and E without Route Request and Reply processes at time t_2. The pre-fetch routing table will now contain data related to node F (pre-fetched to be used at time t_3). However, we can see that the address of node F is already been pre-fetched in this case because of the previous communication. Each node in the network will have a current routing table and a pre-fetch routing table. Each node of the MANET may use the same strategy as X to justify the protocol.

We may now find the access probability of each of the probable pre-fetching nodes and compare them to the threshold probability H. Node D and E are the probable pre-fetching nodes when X is in circle C1 at time t_1. Node C can communicate with both D and E and A can only communicate with D and not E (i.e. D lies in the range of both A and C while E lies in the radius of C only).

Therefore, Probability (of pre-fetching E) = P_d * Z/Y = ½

Here, Y is the number of nodes that lie in circle C2 and not in C1, i.e. 2. Z is 1 because only C from circle C1 can communicate with E.

Similarly, the Probability (of pre-fetching D) = P_d * Z/Y= 2/2 = 1.

Two nodes from circle C1 i.e. A and C can communicate with node D. Thus Z is 2 here while (Y=2) is fixed for a particular pre-fetching range.

The node D has more chance to be pre-fetched than node E and rightly so because more nodes in circle C1 can communicate with D than E. It is highly likely that X would need to communicate with D when X moves in circle C2. Moreover, D has less chance to move out of C2 than E. Thus, X may enjoy zero latency with optimized network cost in circle C2 to communicate D and E (if both them are pre-fetched). All the other nodes may use the same strategy as X this way.

As mentioned earlier, P_d is the predicted probability of changing direction of a moving node. Since, we know the speed and traveling direction of node X, P_d is one for both the pre-fetching probabilities above. Nevertheless, if X does not maintain a specific traveling direction and it moves backward, then the value of P_d would drop sharply, i.e. close to zero. Alternatively, X may slow down rapidly indicating a possible change of direction. The value of P_d would therefore decrease. In this situation, X would not pre-fetch any node-tables of D or E since it would not reside in circle C2 at time t_2. If X takes the path of South-East after time t_1, P_d of node E would be much higher than that of node D.

The request message of DSR consists of 75 bits that includes the fields for source address, destination address, request ID number and type of message. On the other hand, the reply message of DSR consists of 115 bits that includes the fields for source address, destination address, request ID number and type of message, next hop address and the number of hops required/metric.

Source address	Destination address	Request ID	Message type
32 bits	32 bits	8 bits	3 bits

(a)

Source	Destination	Request ID	Metric	Next hop	Message type
32 bits	32 bits	8 bits	8 bits	32 bits	3 bits

(b)

Figure 7: Message format of DSR, (a) Request (b) Reply

The request message of PFP consists of 83 bits that includes the fields for destination address, source address (moving node X), sequence number, installation time and type of message. On the other hand, the reply message of PFP consists of 123 bits that includes the fields for return address (moving node X), destination address, sequence number, installation time, type of message, next hop address and the number of hops required/metric.

Destination address	Source address	Sequence no	Installation time	Message type
32 bits	32 bits	8 bits	8 bits	3 bits

(a)

Destination	Return	Seq. no	Installation time	Message type	Next hop	Metric
32 bits	32 bits	8 bits	8 bits	3 bits	32 bits	8 bits

(b)

Figure 8: Message format of PFP, (a) Request (b) Reply

Each advertisement of DSDV would contain a number of records. The number of bits transmitted in an advertisement can be expressed as:

Source address + (n – 1) * (Destination address + Next hop address + Metric + Sequence number + Installation time) = 32+(n-1)*(32+32+8+8+8) = 32+(n-1)*88, where 32 is the source address of a particular advertisement, n is the number nodes that advertise each other in a particular region at a particular time and 88 is the size of each record in an advertisement. Thus the bits transmitted in each advertisement would vary with n.

The total number of messages need to be sent to communicate with a probable pre-fetching node in DSR may be considered with the equation: (n-i)*2+i = 2n – i, where n is the number of necessary nodes required to communicate (both request and reply) with the destination and i is the number of extra nodes that receive the request messages only. The total number of messages need to be sent to pre-fetch a probable pre-fetching node in PFP is considered to be (n+1), where n is the number of nodes in the current circle of the moving node X. X needs to send n request messages in total for the information of one probable pre-fetching node. And there would be 1 reply message only containing the information, from a node among the requested nodes to X. The number of messages need to be advertised at a particular time in DSDV is considered to be n*(n-1), where n is the number of nodes that advertise each other in a region.

Therefore, the total number bits transmitted to communicate, pre-fetch or to advertise for a probable pre-fetching node would be: (size of a request message * number of request messages + size of a reply message * number of reply messages), for DSR and PFP; and (size of an advertisement * number of advertisements), for DSDV.

The processing overhead combines (a) the generation of extra messages, (b) moving data in cache and (c) number of records scanned in the cache of a node. The generation of extra messages for DSR would be the number of extra nodes that receive request messages i.e. i from the equation above. The generation of extra messages for PFP would be two less than the total number of messages sent to pre-fetch i.e. (n+1)-2. Here, 2 is number of necessary messages (one request and one reply) needed to pre-fetch information. However, DSDV does not generate any extra messages because every advertisement is used to generate a route. There is no cache adjustment for DSR and DSDV. On the other hand, each node needs to adjust the fields for destination address (32 bits), metric required (8bits) and the next hope address (32 bits) in PFP. Thus 72 bits of information is required to be adjusted in terms of moving data in the cache. It is considered that the number of records that need to be searched in the cache of a node is n on the average where n is the number of records in a table for DSDV and PFP. However, n is considered to be zero for DSR because n increases dynamically and it is zero if a node has not communicated with any node.

We also compare the overhead with the same parameters when the pre-fetching nodes leave the MANET. The number of messages sent for DSR and DSDV would be zero in this case because if there is no need, no messages would be generated by DSR, neither that node would be sent any advertisement table by other nodes in DSDV. But PFP still pre-fetches a probable pre-fetching node's information if it were in the range of being pre-fetched before leaving. Thus the overhead would be the number of messages sent to pre-fetch as before i.e. (n+1).

We now see the simulation comparison of pre-fetching time, communication time and advertisement time in Figure 9. The nodes are input as the order of non-decreasing x co-ordinates. The X-axis of the graph indicates the probable pre-fetching nodes in the order of non-decreasing x co-ordinates while Y-axis indicates the time required to pre-fetch, communicate or advertise for each probable node.

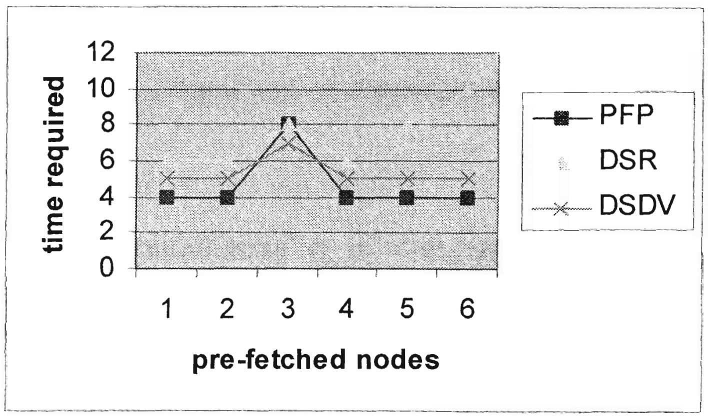

Figure 9. Comparison of pre-fetch, communication and advertise time

The graph shows that DSR takes maximum time to communicate. It takes more time to communicate with the far away nodes for DSR. The curve falls at the middle because X moves closer at time t_2 to the 4th probable pre-fetching node. The advertisement and pre-fetching take constant time for DSDV and PFP because the node distribution is uniform and equal for both the pre-fetching circle. We observe a notch for the third node because this node may not be accessed by any of the current circle nodes. The notch reaches DSR for PFP because PFP switches to DSR for such nodes to be communicated. In general, pre-fetching time is efficient than need based communication or advertisement.

Figure 10 illustrates the comparison of messages generated by the three protocols for each of the pre-fetching nodes. DSR generates more messages for far away nodes.

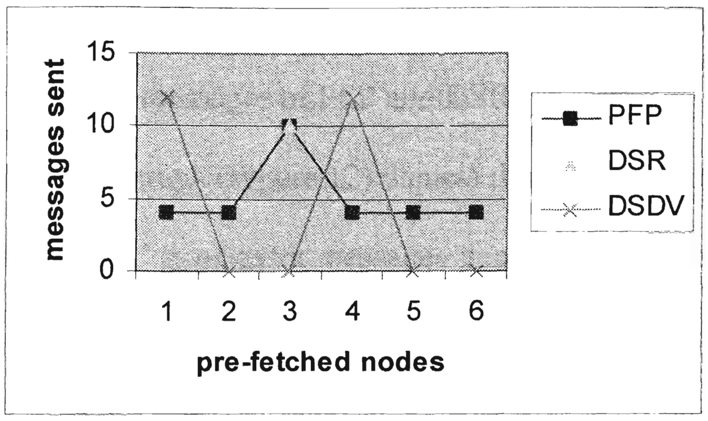

Figure 10. Comparison of messages sent by PFP, DSR and DSDV

PFP generates constant number of messages and switches to DSR if a probable pre-fetching node cannot be pre-fetched. DSDV advertises at time t_1 and t_2. Thus there is a sharp rise for the first and fourth node. The curve stays at the base line for the other nodes because no messages are generated in between time t_1 and t_2.

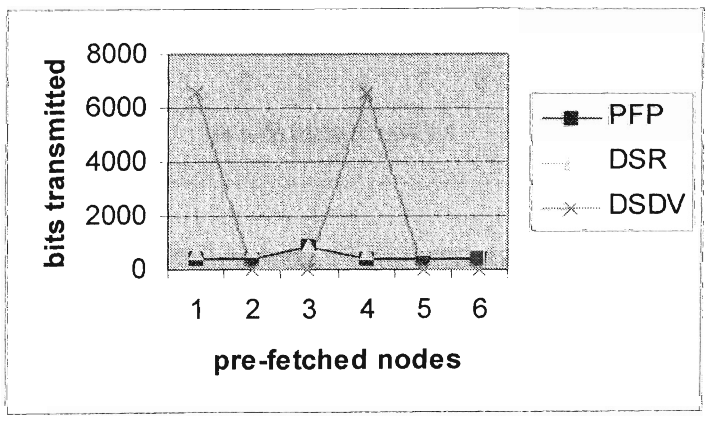

Figure 11. Comparison of bits transmitted by PFP, DSR and DSDV

The comparison of number of bits transmitted (Figure 11) follows the previous comparison. Because transmission of bits reflects the size of messages and the number of different messages sent i.e. request and reply. The DSDV curve takes a very high rise for node 1 and 4 because the size of single advertisement is much larger than the size of the request and reply messages of PFP and DSR.

The next graph (Figure 12) depicts the processing overhead. As mentioned earlier, it is the combination of extra messages generated, adjustment of cache and searching of records in cache. Since there is no adjustment of cache in DSDV and DSR, they outperform PFP. For DSR, the processing overhead is only the extra messages generated because average searching is considered to be zero. For DSDV, the processing overhead is only the scanning of average number of records that is equal here at both time t_1 and t_2. Thus the curve for DSDV forms a line here. PFP drops to DSR for the 3rd node because it switches to DSR for that node.

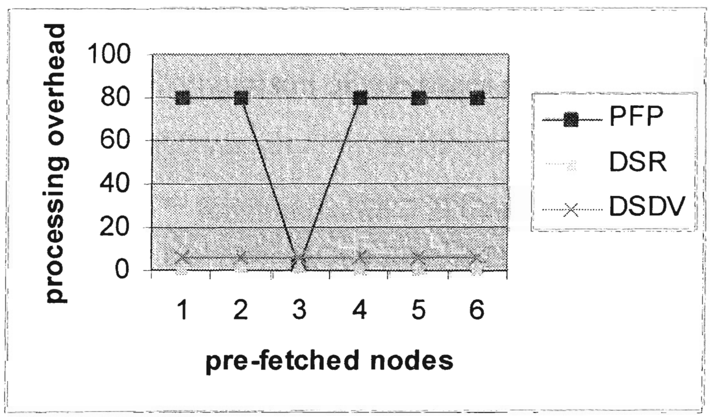

Figure 12. Comparison of processing overhead among PFP, DSR and DSDV

We now take a look at the overhead if the probable pre-fetching nodes move out of the region after they have been pre-fetched. The first comparison is the number of messages generated by the three protocols. In Figure 13, we can see that DSR does not generate any messages since it generates on the need basis. The DSDV advertisement may contain the record of a node that have quit, but that does not make the whole advertisement table invalid because an advertisement table contains information of other nodes. Moreover, DSDV waits a certain period of time before advertisement to see whether a node has left/joined and whether a short path has been established. Thus we consider number of messages generated for the leaving node to be zero for DSDV. Whatever messages are generated to pre-fetch a probable pre-fetching node (in Figure 10) is considered to be overhead for PFP.

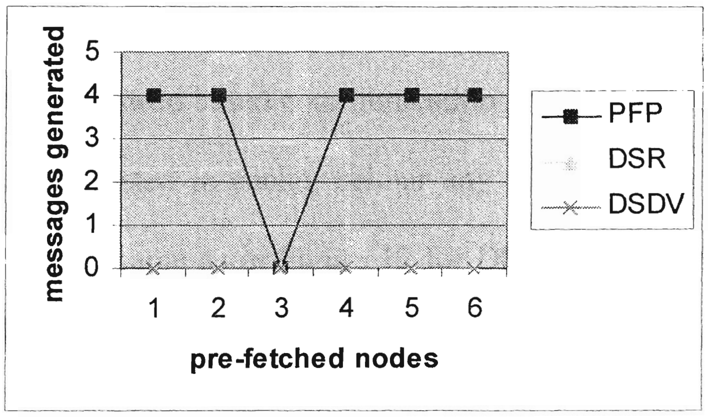

Figure 13. Comparison of messages sent if pre-fetching nodes move away

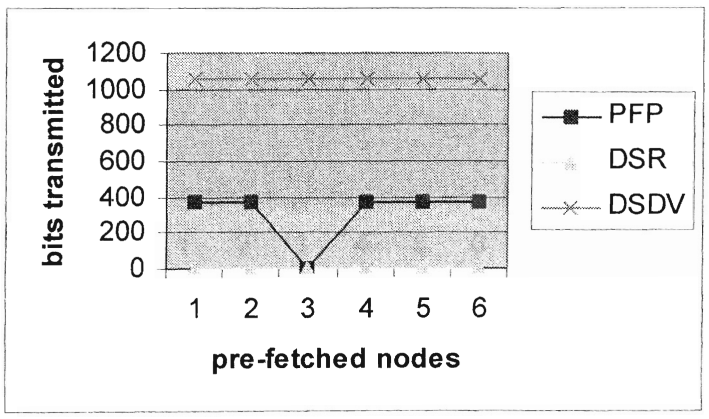

Figure 14. Comparison of bits transmitted if pre-fetching nodes move away

Figure 14 shows the overhead in terms of bits transmitted when the pre-fetching nodes move out of the communication range. There is no bit transferred for DSR. The bits transmitted for PFP is exactly the same as in Figure 11 except the notch is downward this time because DSR curve stays with the X-axis. As said earlier, the extra bits transmitted for DSDV is n*(n-1)*88 even though the extra messages generated is zero. Here, n is the number of nodes advertising in a region and n*(n-1) is the number of times (of advertisement) a record size (information of the node that quits) of 88 bits transferred. Since n is equal i.e. 3 in both the pre-fetching range the curve behaves a line for DSDV. Thus in terms of number of bits transferred, PFP works in between DSR and DSDV.

The processing overhead when the nodes move away is always zero for DSR. Since, DSR is a need based communication protocol, there is no extra message generated, no bits are adjusted in cache, neither any record is scanned in cache. This processing overhead is the same as in Figure 12 for DSDV. However, it is slightly higher this time than in Figure 12 for PFP, because whatever messages have been generated for the node that has quit would be extra.

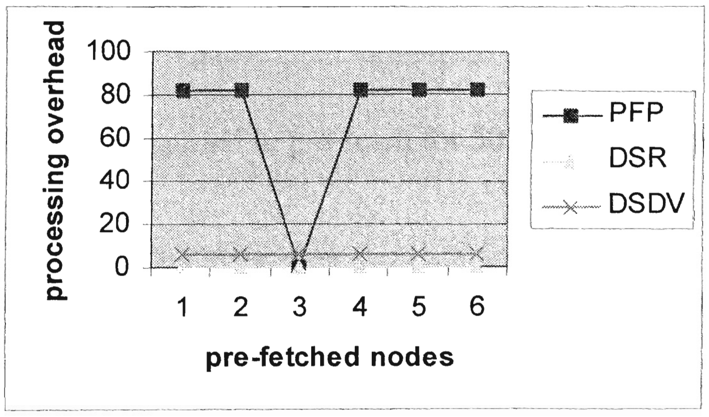

Figure 15. Comparison of processing overheads when pre-fetched nodes move

The graphs suggest that it is worth pre-fetching when the nodes are not highly mobile and the various overheads in such case are reasonably low except the processing overhead. However, if the predictions are wrong PFP suffers tremendously. So, nodes should be pre-fetched only when they exceed the pre-fetch threshold.

Next, we looked at the behavior of the PFP in terms of time required and messages sent to pre-fetch 4300 sets of randomly generated 100 nodes. 50 nodes were generated in the current range of the moving node X and the other 50 were generated in the pre-fetching range. The nodes were sorted according to their x-co ordinates after they were randomly generated.

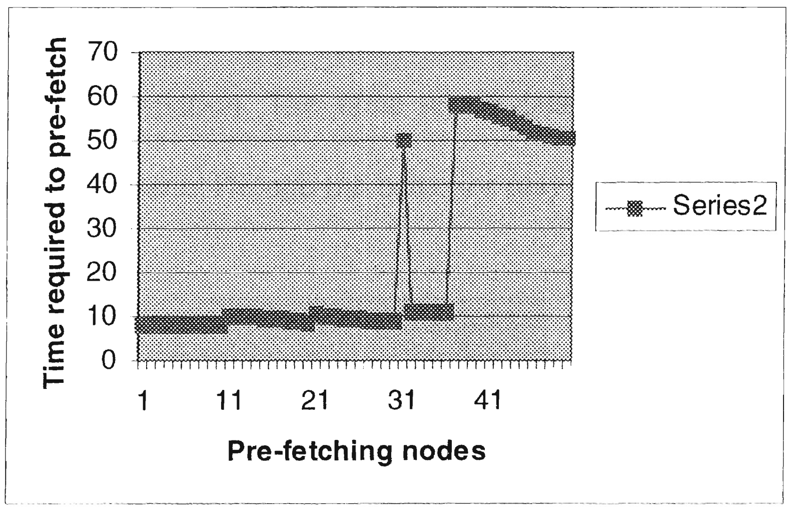

Figure 16. Time required to pre-fetch for 50 pre-fetching nodes

We see that the graph is pretty consistent for the nearer pre-fetching nodes. Since the nodes were sorted according to their x coordinates, time required to pre-fetch goes up very slowly for the later nodes. If the nodes are pre-fetched from the nodes that are within the current range of X, then the time required to pre-fetch is constant for all such nodes. Therefore, we see the curve to be a straight line for the first few nodes. At node 11, it goes slightly up and stays there because the pre-fetching nodes now fall little far away and the nodes information are pre-fetched via some other intermediate nodes' table by X. Thus, the time required to pre-fetch is somewhat higher there. The curve goes slightly up again at node 20 for the same reason. We see a skew for the node 30, which implies that this node information was not pre-fetched and the protocol needs to switch to other protocol to pre-fetch its information. The graph goes little higher for the later few nodes and forms a line again. And then for the further nodes, we see that curve goes up sharply which implies that there were some sets, among the 4300 sets whose node's information could not be pre-fetched in that range.

We now look at the behavior of the protocol in terms of messages sent to pre-fetch information. The same set of data was used as above to measure the number of messages sent. Though the behavior may seem similar, the shifts are much higher

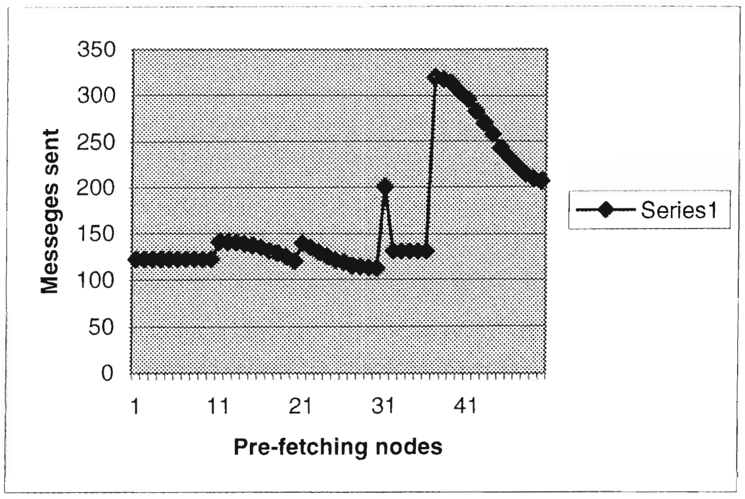

Figure 17. Overhead in terms for messages sent

here. The number of messages goes up as the range increases. It takes shifts after some nodes, because those nodes are located in such places where more messages need to be generated to pre-fetch their information. The number of messages sent goes down after the shift, because some nodes are located in the crowded area where their information could be pre-fetched with the generation of few messages. The spike at node 30 means that it could not be pre-fetched and PFP needs to shift to other protocols to pre-fetch the information of that node. The graph goes up sharply for the later nodes, because there were some sets among the 4300 sets whose information could not be pre-fetched in that range. Again the curve falls down because of the nodes residence in the crowded area.

Thus we see that both time and number of messages sent are low for the nodes that lie nearer to the moving node and high for the further nodes.

CHAPTER VII

CONCLUSIONS

In this paper, we have presented a pre-fetching scheme to reduce the communication delay in a MANET. To achieve high efficiency with pre-fetching, in the prediction module, we must devise an algorithm to predict the access probability of each probable pre-fetching node as accurately as possible. Nodes with access probability greater than its server's pre-fetch threshold are pre-fetched and if the pre-fetched nodes do not move out of bound, the protocol works faster than others. We also studied the performance of our protocol in terms of various overheads.

This work may be extended to see how PFP works when it is combined with Location Aided routing. Better result is expected if PFP switches to LAR when needed. The threshold with access probability needs to be applied to measure the improvement of performance. The test may also be performed when a node moves in a very irregular fashion. A real system is more desired to carry on the tests. Also, experiments should involve as many nodes as possible.

However, in summary we believe that pre-fetch is a good approach to reduce latency for Mobile Network applications.

REFERENCES

1. Stojmenovic, I. (Jul 2002). Position-based routing in ad hoc networks. *IEEE Communications Magazine* , Volume: 40 Issue: 7, Page(s): 128 -134

2. Ramanathan, R.; Redi, J. (May 2002). A brief overview of ad hoc networks: challenges and directions. *IEEE Communications Magazine* , Volume: 40 Issue: 5 Part: Anniversary , Page(s): 20 –22

3. Perkins, Charles E. (2001). *Ad Hoc Networking.* Addision-Wesley

4. Vaidya, Nitin H.; Ko, Young-Bae (2001). Location-Aided Routing (LAR*). IEEE Communications Magazine,* Volume: 35 Issue: 5, Page(s): 66 – 75

5. Johnson, D.; Maltz D.; Broch Josh (Mar 1998). The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, Volume: 30, Issue: 5, Page(s): 139 - 168

6. Kleinrock, L.; Jiang, Z (Apr 1998). An adaptive network prefetch scheme. *IEEE Journal on Selected Areas in Communications*, Volume: 16, No. 3, Page(s): 358 – 368

7. Perkins, C. E.; Royer, E. M. (Aug 1998). Ad hoc on demand distance vector (AODV) Routing. *IEEE Journal on Selected Areas in Communications*, Volume: 24, No. 4, Page(s): 173 - 207

8. Haas, Z. J.; Pearlman, M. R. (Aug 1998). The Zone Routing Protocol (ZRP) for Ad Hoc Networks. *IEEE Transactions on Networking*, Vol. 5, Page(s): 221 – 249

9. Akyildiz, I. F.; Ho, J. S. M.; Lin, Yi-Bing (Aug 1996). Movement-based location update and selective paging for PCS networks. *IEEE Transactions on Networking*, Vol. 4, Page(s): 629 – 638

10. Perkins, C. E.; Bhagwat, P. (1996). Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, Addison - Wesley

APPENDIX A

GLOSSARY

Damping Fluctuation	– The settling time that is used to prevent fluctuations of route table entry advertisements. This delay period allows a node to wait for the shortest path to the destination.
Route Discovery	– Used only when a node attempts to send a packet to a destination node and does not already know a route to it.
Route Maintenance	– Used to help detect a source node if the network topology has changed or whether a link to a destination node is broken. In such cases, Route Maintenance indicates the source node to use any other route it happens to know, or it can invoke Route Discovery again to find a new route.
MACT	– Multicast Activation message is used to send to a selected next hop to activate or deactivate a multicast route by setting the appropriate flag. MACT is also used when a node wishes to revoke its member status and leave the multicast group.

APPENDIX B

IMPLEMENTATION

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

//package Applications;
import java.awt.*;
import javax.swing.*;
import java.io.*;
import java.lang.*;
/*
 * Application_2.java
 *
 * Created on September 21, 2002, 4:40 PM
 */

/**
 *
 * @author  Muhammad T Alam
 */
public class Application_2 extends javax.swing.JFrame {

   public class nodes{
      private int x;
      private int y;
      /*public nodes(){
         x=0;
         y=0;
      }*/
      public nodes(int a, int b){
         x=a;
         y=b;
      }
      int getnodeX(){
         return x;
      }
      int getnodeY(){
            return y;
        }
        void setnodeX(int a){
            x = a;
        }
        void setnodeY(int b){
            y = b;
        }
    }

    //moving X;
    nodes totalnodes[] = new nodes[10];
    nodes currentnodes[] = new nodes[51];
    nodes prefetchnodes[] = new nodes[51];
    nodes mergedarray[] = new nodes[101];

    int t1[] = new int[11];

    int t2[] = new int[11];

    int y=0 ,r=0;
    public class moving{
        private int x;
        private int y;

        public moving(int a, int b){
            x = a;
            y = b;
        }
        int getnodeX(){
            return x;
        }
        int getnodeY(){
            return y;
        }
        void print(){
            System.out.println("The location of moving node X At time t1 is (0,"+y+")");
            System.out.println("The location of moving node X At time t2 is ("+r+","+y+")");
            System.out.println("The location of moving node X At time t3 is
("+2*r+","+y+")");
        }
    }

    moving X;
//  public class distance{
      public double dist(int x1, int y1, int x2, int y2){
         double a,b,c,d;
         a = (x1-x2)*(x1-x2) ;
         b = (y1-y2)*(y1-y2) ;
         d = a+b;
         c = Math.sqrt(d);
         return c;
      }
 //  }

      public boolean prefetch(int x[], int size){
        for (int i=1; i<=size;i++){
         if (x[i]!=0){
            return false;
//            break;
         }
        }
        return true;
      }

      public int firstnode(int x[], int size){
         int t_first=0;
         for (int a=1;a<10;a++){
            if  (x[a]!=0){
            t_first = a;
            break;
            }
          //return t_first;
         }return t_first;
      }

      public int lastnode(int x[], int size){
         int t_last = 0;
         for (int a=10;a>=1;a--){
            if  (x[a]!=0){
            t_last = a;
            break;
            }
          //return t_first;
         }return t_last;
      }

     public void result_t1(int x[]){
      int hold1=0, hold2=0;
      hold1 = firstnode(x,10);
      hold2 = lastnode(x,10);
```

```
System.out.println("*****************************************************
*************");

System.out.println("*****************************************************
*************");

       System.out.println("PF PROTOCOL: ");
       System.out.println("Request message size: 83 bits.");
       System.out.println("Reply message size: 123 bits.");

       //int check=0;
       for (int o = hold1; o <=hold2; o++){
         int check=0;
         for (int n=hold1-1; n>=1;n--)
             if ((dist(totalnodes[n].x, totalnodes[n].y, totalnodes[o].x, totalnodes[o].y)<=r)
&& (dist(0 , y, totalnodes[n].x, totalnodes[n].y)<=r)){

System.out.println("*****************************************************
*************");

               //System.out.println("OUR PROTOCOL: ");
               System.out.print("X can get location information of node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ) from node (");
               System.out.println(totalnodes[n].x+" , "+totalnodes[n].y+" ) which means
this node can be pre-fetched");
               System.out.println("and X can enjoy zero latency to communicate at time
t2");

               System.out.println("Number of Request messages: "+(hold1-1));
               System.out.println("Number of Reply messages: 1");
               System.out.println("Total messages sent to communicate is "+hold1);
               System.out.println("Total number of bits transmitted: "+((hold1-
1)*83+123));
               System.out.println("Time required to pre-fetch is 4");
               System.out.println("Generation of extra messages: "+(hold1-2));
               System.out.println("Number of bits adjusted in chache: 72");
               System.out.println("Average number of records scanned in cache: "+hold2);



System.out.println("*****************************************************
*************");
               System.out.println();
```

```
                System.out.println("If this node moves out of range:");
                System.out.println("Overhead - ");
                System.out.println("In terms of total number of messages: "+hold1);
                System.out.println("In terms of number of bits transmitted: "+((hold1-
1)*83+123));
                System.out.println("Processing Overhead - ");
                System.out.println("In terms of generation of extra messages: "+hold1);
                System.out.println("In terms of number of bits adjusted in cache: 72");
                System.out.println("In terms of average number of records scanned in cache:
"+hold2);



                check=1;
                break;
              }
              if (check == 0) {
                System.out.println();

                System.out.println("Cannot pre-fetch the information of node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ).");
                System.out.println("Switch to DSR.");
                System.out.println("Time required will be same as DSR.");
                System.out.println("Messages sent will be same as DSR.");
                System.out.println("Bits transmitted will be same as DSR.");
                System.out.println("Generation of extra messages will be same as DSR.");
                System.out.println("No bits adjusted in cache.");
                System.out.println("No records been scanned in cache.");


                System.out.println();

                System.out.println("If this node moves out of range, there would be no
overhead ");

                }
        }

        }

        public void result_t2(int x[], int t1first){
         int hold1=0, hold2=0;
         hold1 = firstnode(x,10);
         hold2 = lastnode(x,10);
```

```
System.out.println("********************************************************
*************");

System.out.println("********************************************************
*************");

            System.out.println("PF PROTOCOL: ");

            System.out.println("Request message size: 83 bits.");
            System.out.println("Reply message size: 123 bits.");


        if (t1first==0)
            System.out.println("Cannot Pre-fetch any of the nodes.");

        else{

        for (int o = hold1; o <=hold2; o++){
          int check =0;
          for (int n=hold1-1; n>=t1first; n--)
              if        ((dist(totalnodes[n].x,        totalnodes[n].y,        totalnodes[o].x,
totalnodes[o].y)<=r)&& (dist(r , y, totalnodes[n].x, totalnodes[n].y)<=r)){

System.out.println("********************************************************
*************");

                // System.out.println("OUR PROTOCOL: ");
                System.out.print("X    can    get    location    information    of    node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ) from node (");
                System.out.println(totalnodes[n].x+"  ,  "+totalnodes[n].y+"  )  which  means
this node can be pre-fetched");
                System.out.println("and X can enjoy zero latency at time t3");

                System.out.println("Number of Request messages: "+(hold1-t1first));
                System.out.println("Number of Reply messages: 1");
                System.out.println("Total   messages   sent   to   communicate   is   "+(hold1-
t1first+1));
                System.out.println("Total    number    of    bits    transmitted:    "+((hold1-
t1first)*83+123));
                System.out.println("Time required to pre-fetch is 4");
                System.out.println("Generation of extra messages: "+((hold1-t1first+1)-2));
                System.out.println("Number of bits adjusted in chache: 72");
                System.out.println("Average number of records scanned in cache: "+(hold2-
t1first+1));
System.out.println("*****************************************************
*************");
                    System.out.println();


                    System.out.println("If this node moves out of range:");
                    System.out.println("Overhead - ");
                    System.out.println("In terms of total number of messages: "+(hold1-
t1first+1));
                    System.out.println("In terms of number of bits transmitted: "+((hold1-
t1first)*83+123));
                    System.out.println("Processing Overhead - ");
                    System.out.println("In terms of generation of extra messages: "+(hold1-
t1first+1));
                    System.out.println("In terms of number of bits adjusted in cache: 72");
                    System.out.println("In terms of average number of records scanned in cache:
"+(hold2-t1first+1));



                    check = 1;
                    break;
                 }
                 if (check == 0) {
                    System.out.println();

                     System.out.println("Cannot pre-fetch the information of node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ).");
                    System.out.println("Switch to DSR.");
                    System.out.println("Time required will be same as DSR.");
                    System.out.println("Messages sent will be same as DSR.");
                    System.out.println("Bits transmitted will be same as DSR.");
                    System.out.println("Generation of extra messages will be same as DSR.");
                    System.out.println("No bits adjusted in cache.");
                    System.out.println("No records been scanned in cache.");

                    System.out.println();

                    System.out.println("If this node moves out of range, there would be no
overhead ");

                    }
         }
         }
        }
```

```
     public void DSR_t1(int x[]){
      int hold1=0, hold2=0;
      hold1 = firstnode(x,10);
      hold2 = lastnode(x,10);


System.out.println("*********************************************************
*************");

System.out.println("*********************************************************
*************");

     System.out.println("DSR PROTOCOL: ");
     System.out.println("Request message size: 75 bits.");
     System.out.println("Reply message size: 115 bits.");

     int count, flag = 0;
     //int check=0;
     for (int o = hold1; o <=hold2; o++){

System.out.println("*********************************************************
*************");

          // System.out.println("DSR PROTOCOL: ");
          System.out.println("At   time   t1,   X   can   communicate   with   node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ) ");
          count = 0;
          for( int i=1; i<= (o-1); i++){
             if (totalnodes[i].x==totalnodes[i+1].x){
                   count =count + 1;
                   flag = 1;
             }

          }

           if (flag==0){
             System.out.println("Number of Request messages: "+o);
             System.out.println("Number of Reply messages: "+o);
             System.out.println("Total messages sent to communicate is "+o*2);
             System.out.println("Total number of bits transmitted: "+(o*75+o*115));
             System.out.println("Time required to commucate is "+o*2);
             System.out.println("Generation of extra messages: 0");
             System.out.println("No bits adjusted in cache.");
             System.out.println("Average number of records scanned in cache: 0");
```

```
                    System.out.println();
                    System.out.println();

                    System.out.println("If this node moves out of range, there would be no
overhead ");

                    //System.out.println("Time required to commucate is "+o*2);

                }
                else
                    {
                        System.out.println("Number of Request messages: "+o);
                        System.out.println("Number of Reply messages: "+(o-count));
                        System.out.println("Total messages sent to communicate is "+(o*2-
count));
                        System.out.println("Total number of bits transmitted: "+(o*75+(o-
count)*115));

                        System.out.println("Time required to commucate is "+((o-count)*2));
                        System.out.println("Generation of extra messages: "+count);
                        System.out.println("No bits adjusted in cache.");
                        System.out.println("Average number of records scanned in cache: 0");

                        System.out.println();
                        System.out.println();

                        System.out.println("If this node moves out of range, there would be no
overhead ");

                    }

                }

        }

        public void DSR_t2(int x[], int t1first){
         int hold1=0, hold2=0;
         hold1 = firstnode(x,10);
         hold2 = lastnode(x,10);

System.out.println("***********************************************************
*************");

System.out.println("***********************************************************
*************");
```

```
            System.out.println("DSR PROTOCOL: ");
            System.out.println("Request message size: 75 bits.");
            System.out.println("Reply message size: 115 bits.");

        int count = 0, flag = 0;

        if (t1first==0){
          int f=1;
          for (int d=hold1; d<=hold2; d++){
            System.out.println("At    time    t2,    X    can    communicate    with    node
("+totalnodes[d].x+" , "+totalnodes[d].y+" ) ");
            System.out.println("Messages sent to communicate is "+f*2);
            System.out.println("Time required to communicate is "+f*2);
            f=f+1;
          }
        }
        else{


        for (int o = hold1; o <=hold2; o++){

System.out.println("*******************************************************
*************");

              // System.out.println("DSR PROTOCOL: ");
               System.out.println("At    time    t2,    X    can    communicate    with    node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ) ");
               count = 0;

               for( int i=t1first; i<= (o-1); i++){
                  if (totalnodes[i].x==totalnodes[i+1].x){
                        count =count + 1;
                        flag = 1;
                  }

               }



              if (flag==0){
                 System.out.println("Number of Request messages: "+(o-t1first+1));
                 System.out.println("Number of Reply messages: "+(o-t1first+1));
                 System.out.println("Total    messages    sent    to    communicate    is    "+(o-
t1first+1)*2);
                 System.out.println("Total    number    of    bits    transmitted:    "+((o-
t1first+1)*75+(o-t1first+1)*115));
```

```
                        System.out.println("Time required to commucate is "+(o-t1first+1)*2);
                        System.out.println("Generation of extra messages: 0");
                        System.out.println("No bits adjusted in cache.");
                        System.out.println("Average number of records scanned in cache: 0");

                        System.out.println();
                        System.out.println();

                        System.out.println("If this node moves out of range, there would be no
overhead ");

                    }
                    else
                    {
                        System.out.println("Number of Request messages: "+(o-t1first+1));
                        System.out.println("Number of Reply messages: "+(o-t1first+1-count));
                        System.out.println("Total messages sent to communicate is "+((o-
t1first+1)*2-count));
                        System.out.println("Total number of bits transmitted: "+((o-
t1first+1)*75+(o-t1first+1-count)*115));
                        System.out.println("Time required to commucate is "+((o-t1first+1-
count)*2));
                        System.out.println("Generation of extra messages: "+count);
                        System.out.println("No bits adjusted in cache.");
                        System.out.println("Average number of records scanned in cache: 0");

                        System.out.println();
                        System.out.println();

                        System.out.println("If this node moves out of range, there would be no
overhead ");

                    }



                }
            }
            }


            public void DSDV_t1(int x[]){
             int hold1=0, hold2=0;
             hold1 = firstnode(x,10);
             hold2 = lastnode(x,10);
```

```
System.out.println("*******************************************************
*************");

System.out.println("*******************************************************
*************");


        System.out.println("DSDV PROTOCOL: ");
        System.out.println("Advertisement will include source address and records of
information of all reachable destinations");
        System.out.println("Single record size: 88 bits.");

        //int count, flag = 0;
        //int check=0;
         for (int o = hold1; o <=hold2; o++){
           int check = 0;
           for (int n=hold1-1; n>=1;n--)
              if ((dist(totalnodes[n].x, totalnodes[n].y, totalnodes[o].x, totalnodes[o].y)<=r)
&& (dist(0 , y, totalnodes[n].x, totalnodes[n].y)<=r)){

System.out.println("*******************************************************
*************");

                System.out.println("At time t1, X can communicate with node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ) ");
                System.out.println("Time required to advertise is "+(hold1+1));
                System.out.println("Average number of records scanned in cache: "+hold2);


System.out.println("*******************************************************
*************");
                System.out.println();


                System.out.println("If this node moves out of range:");
                System.out.println("Overhead - ");
                System.out.println("In terms of total number of messages: 0");
                System.out.println("In terms of number of bits transmitted: "+((hold1-
1)*hold1*88));
                System.out.println("Processing Overhead - ");
                System.out.println("In terms of generation of extra messages: 0");
                System.out.println("In terms of number of bits adjusted in cache: 0");
                System.out.println("In terms of average number of records scanned in cache:
"+hold2);
```

```
                check =1;
                break;
            }
            if (check==0){

System.out.println("*****************************************************
*************");

                System.out.println("At time t1, X can communicate with node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ) ");

                System.out.println("Time required to advertise is "+(hold1+3));
                System.out.println("Average number of records scanned in cache:
"+hold2);

                System.out.println();
                System.out.println();

                System.out.println("If this node moves out of range:");
                System.out.println("Overhead - ");
                System.out.println("In terms of total number of messages: 0");
                System.out.println("In terms of number of bits transmitted: "+((hold1-
1)*hold1*88));
                System.out.println("Processing Overhead - ");
                System.out.println("In terms of generation of extra messages: 0");
                System.out.println("In terms of number of bits adjusted in cache: 0");
                System.out.println("In terms of average number of records scanned in
cache: "+hold2);

            }

        }

        System.out.println();
        System.out.println();

        System.out.println("Total advertisement messages sent at time t1 is "+((hold1-
1)*hold1));
        System.out.println("Total number of bits transmitted: "+ (((hold1-
1)*hold1*hold2)*88+32*(hold2+1)));
        System.out.println("Generation of extra messages: 0");
        System.out.println("No bits adjusted in cache.");

    }
```

```
        public void DSDV_t2(int x[], int t1first){
         int hold1=0, hold2=0;
         hold1 = firstnode(x,10);
         hold2 = lastnode(x,10);


System.out.println("*********************************************************
*************");

System.out.println("*********************************************************
*************");

         System.out.println("DSDV PROTOCOL: ");
         System.out.println("Advertisement will include source address and records of
information of all reachable destinations");
         System.out.println("Single record size: 88 bits.");

         //int count, flag = 0;
        // int check=0;
          for (int o = hold1; o <=hold2; o++){
            int check =0;
           for (int n=hold1-1; n>=t1first; n--)
              if        ((dist(totalnodes[n].x,        totalnodes[n].y,        totalnodes[o].x,
totalnodes[o].y)<=r)&& (dist(r , y, totalnodes[n].x, totalnodes[n].y)<=r)){

System.out.println("*********************************************************
*************");


                    System.out.println("At time t2, X can communicate with node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ) ");
                    System.out.println("Time required to advertise is "+(hold1-t1first+2));
                    System.out.println("Average number of records scanned in cache: "+(hold2-
t1first+1));


System.out.println("*********************************************************
*************");
                    System.out.println();

                    System.out.println("If this node moves out of range:");
                    System.out.println("Overhead - ");
                    System.out.println("In terms of total number of messages: 0");
                    System.out.println("In terms of number of bits transmitted: "+((hold1-
t1first+1)*(hold1-t1first)*88));
                    System.out.println("Processing Overhead - ");
```

```
                System.out.println("In terms of generation of extra messages: 0");
                System.out.println("In terms of number of bits adjusted in cache: 0");
                System.out.println("In terms of average number of records scanned in cache:
"+(hold2-t1first+1));


                check =1;
                break;
              }
               if (check==0){

System.out.println("*****************************************************
*************");

                  System.out.println("At time t2, X can communicate with node
("+totalnodes[o].x+" , "+totalnodes[o].y+" ) ");

                  System.out.println("Time required to advertise is "+(hold1-t1first+4));
                  System.out.println("Average number of records scanned in cache:
"+(hold2-t1first+1));
                  System.out.println();
                  System.out.println();

                  System.out.println("If this node moves out of range:");
                  System.out.println("Overhead - ");
                  System.out.println("In terms of total number of messages: 0");
                  System.out.println("In terms of number of bits transmitted: "+((hold1-
t1first+1)*(hold1-t1first)*88));
                  System.out.println("Processing Overhead - ");
                  System.out.println("In terms of generation of extra messages: 0");
                  System.out.println("In terms of number of bits adjusted in cache: 0");
                  System.out.println("In terms of average number of records scanned in
cache: "+(hold2-t1first+1));

              }

         }
         System.out.println();
         System.out.println();

         System.out.println("Total advertisement messages sent at time t2 is "+((hold1-
t1first+1)*(hold1-t1first)));
         System.out.println("Total number of bits transmitted: "+ ((hold1-t1first+1)*(hold1-
t1first)*(hold2-t1first+1)*88+32*(hold2-t1first+2)));
         System.out.println("Generation of extra messages: 0");
         System.out.println("No bits adjusted in cache.");
```

```
    }

  //distance temp;

  /** Creates new form Application_2 */
  public Application_2() {
     initComponents();
  }

  /** This method is called from within the constructor to
   * initialize the form.
   * WARNING: Do NOT modify this code. The content of this method is
   * always regenerated by the Form Editor.
   */
  private void initComponents() {//GEN-BEGIN:initComponents
     menuBar = new javax.swing.JMenuBar();
     fileMenu = new javax.swing.JMenu();
     openMenuItem = new javax.swing.JMenuItem();
     saveMenuItem = new javax.swing.JMenuItem();
     saveAsMenuItem = new javax.swing.JMenuItem();
     exitMenuItem = new javax.swing.JMenuItem();
     editMenu = new javax.swing.JMenu();
     cutMenuItem = new javax.swing.JMenuItem();
     copyMenuItem = new javax.swing.JMenuItem();
     pasteMenuItem = new javax.swing.JMenuItem();
     deleteMenuItem = new javax.swing.JMenuItem();
     helpMenu = new javax.swing.JMenu();
     contentsMenuItem = new javax.swing.JMenuItem();
     aboutMenuItem = new javax.swing.JMenuItem();
     jLabel1 = new javax.swing.JLabel();
     jButton1 = new javax.swing.JButton();
     jButton6 = new javax.swing.JButton();
     jSeparator2 = new javax.swing.JSeparator();
     jLabel2 = new javax.swing.JLabel();
     jLabel3 = new javax.swing.JLabel();
     jLabel6 = new javax.swing.JLabel();
     jLabel7 = new javax.swing.JLabel();
     jLabel8 = new javax.swing.JLabel();
     jLabel9 = new javax.swing.JLabel();
     jButton7 = new javax.swing.JButton();
     jTextField1 = new javax.swing.JTextField();
     jTextField2 = new javax.swing.JTextField();
     jLabel4 = new javax.swing.JLabel();
```

```
jLabel5 = new javax.swing.JLabel();
jLabel10 = new javax.swing.JLabel();
jLabel11 = new javax.swing.JLabel();
jButton2 = new javax.swing.JButton();

fileMenu.setText("File");
openMenuItem.setText("Open");
fileMenu.add(openMenuItem);
saveMenuItem.setText("Save");
fileMenu.add(saveMenuItem);
saveAsMenuItem.setText("Save As ...");
fileMenu.add(saveAsMenuItem);
exitMenuItem.setText("Exit");
exitMenuItem.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    exitMenuItemActionPerformed(evt);
  }
});

fileMenu.add(exitMenuItem);
menuBar.add(fileMenu);
editMenu.setText("Edit");
cutMenuItem.setText("Cut");
editMenu.add(cutMenuItem);
copyMenuItem.setText("Copy");
editMenu.add(copyMenuItem);
pasteMenuItem.setText("Paste");
editMenu.add(pasteMenuItem);
deleteMenuItem.setText("Delete");
editMenu.add(deleteMenuItem);
menuBar.add(editMenu);
helpMenu.setText("Help");
contentsMenuItem.setText("Contents");
helpMenu.add(contentsMenuItem);
aboutMenuItem.setText("About");
helpMenu.add(aboutMenuItem);
menuBar.add(helpMenu);

getContentPane().setLayout(null);

setFont(new java.awt.Font("Arial", 0, 14));
addWindowListener(new java.awt.event.WindowAdapter() {
  public void windowClosing(java.awt.event.WindowEvent evt) {
    exitForm(evt);
  }
});
```

```
    jLabel1.setFont(new java.awt.Font("Arial", 2, 24));
    jLabel1.setForeground(new java.awt.Color(102, 0, 102));
    jLabel1.setText("Protocol For Reducing Delay");
    getContentPane().add(jLabel1);
    jLabel1.setBounds(240, 0, 340, 50);

    jButton1.setBackground(java.awt.Color.blue);
    jButton1.setText("Exit");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
      }
    });

    getContentPane().add(jButton1);
    jButton1.setBounds(290, 410, 240, 50);

    jButton6.setBackground(java.awt.Color.cyan);
    jButton6.setText("10 Nodes Simulation");
    jButton6.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton6ActionPerformed(evt);
      }
    });

    getContentPane().add(jButton6);
    jButton6.setBounds(290, 310, 240, 50);

    getContentPane().add(jSeparator2);
    jSeparator2.setBounds(370, 120, 0, 2);

    jLabel2.setFont(new java.awt.Font("Arial", 1, 14));
    jLabel2.setForeground(new java.awt.Color(255, 0, 51));
    jLabel2.setText("Equation format: Y = C, Slope = 0");
    getContentPane().add(jLabel2);
    jLabel2.setBounds(290, 470, 350, 30);

    jLabel3.setText("t1 and t2, calculates the coordinates of X at t1 and t2, finds the
time and overhead incurred by our protocol to prefetch each node    ");
    getContentPane().add(jLabel3);
    jLabel3.setBounds(20, 160, 750, 16);

    jLabel6.setText("and finds time and overhead incurred by DSR and DSDV to
communicate/advertise the pre-fetching nodes at time t1 and t2.");
    getContentPane().add(jLabel6);
    jLabel6.setBounds(20, 190, 740, 20);

    jLabel7.setText("This software compares PFP with DSR and DSDV in terms of time
required and overhead. The moving node X is assumed to move in a ");
    getContentPane().add(jLabel7);
    jLabel7.setBounds(20, 70, 760, 30);

    jLabel8.setText("straight line with slope zero and the starting co-ordinate of x as
zero. Please enter node co-ordinates in non decreasing order of their");
    getContentPane().add(jLabel8);
    jLabel8.setBounds(20, 100, 750, 20);

    jLabel9.setText("x co-ordinates when you click the nodes button below. The
program calculates the probable pre-fetching nodes at two future times;  ");
    getContentPane().add(jLabel9);
    jLabel9.setBounds(20, 130, 750, 16);

    jButton7.setBackground(java.awt.Color.white);
    jButton7.setText("Enter the starting y co-ordinate of moving node X and the
communication range of each node first");
    jButton7.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton7ActionPerformed(evt);
      }
    });

    getContentPane().add(jButton7);
    jButton7.setBounds(80, 220, 610, 26);

    jTextField1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1ActionPerformed(evt);
      }
    });

    getContentPane().add(jTextField1);
    jTextField1.setBounds(310, 270, 20, 20);

    jTextField2.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextField2ActionPerformed(evt);
      }
    });

    getContentPane().add(jTextField2);
    jTextField2.setBounds(540, 270, 50, 20);
    jLabel4.setText("( 0 , ");
    getContentPane().add(jLabel4);
    jLabel4.setBounds(280, 270, 30, 16);

    jLabel5.setText(")");
    getContentPane().add(jLabel5);
    jLabel5.setBounds(340, 270, 10, 16);

    jLabel10.setText("( x , y ) =  ");
    getContentPane().add(jLabel10);
    jLabel10.setBounds(230, 270, 52, 16);

    jLabel11.setText("Communication radius =");
    getContentPane().add(jLabel11);
    jLabel11.setBounds(390, 270, 140, 16);

    jButton2.setBackground(java.awt.Color.cyan);
    jButton2.setText("100 Nodes Simulation");
    jButton2.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton2ActionPerformed(evt);
      }
    });

    getContentPane().add(jButton2);
    jButton2.setBounds(290, 360, 240, 50);

    setJMenuBar(menuBar);
    pack();
  }//GEN-END:initComponents

  private   void   jButton2ActionPerformed(java.awt.event.ActionEvent   evt)   {//GEN-
FIRST:event_jButton2ActionPerformed
   double time[] = new double[101];
   double message[] = new double[101];

  for (int run = 1; run <=200;run++)
   {
    //current range generation of random 50 nodes
    int hold_x, hold_y;
    for (int i=1;i<=50;i++)
    {
      hold_x = 1 + (int)(Math.random() * 50);
      hold_y = (int)(Math.random() * 101);
      //System.out.println(hold_x+" "+hold_y);
    //System.out.println(hold_y);
    currentnodes[i] = new nodes(hold_x, hold_y);
 }


 //sorting according to x co ordinates
 //System.out.println("***");
   // System.out.println("***");
 int tmpx, tmpy;
 for (int i=1; i<51; i++) {
  for (int j=1; j<51-i; j++)
     if (currentnodes[j+1].x < currentnodes[j].x) {  /* compare the two neighbors */
      tmpx = currentnodes[j].getnodeX();
      tmpy = currentnodes[j].getnodeY();
          /* swap a[j] and a[j+1]      */
      currentnodes[j].setnodeX(currentnodes[j+1].getnodeX());
      currentnodes[j].setnodeY(currentnodes[j+1].getnodeY());

      currentnodes[j+1].setnodeX(tmpx);
      currentnodes[j+1].setnodeY(tmpy);

     }
 }


 /*
 for (int j=1;j<=50;j++)
 {

    System.out.println(currentnodes[j].x+" "+currentnodes[j].y);
  }
 */
 //System.out.println("***");
  // System.out.println("***");
//random generation of 50 nodes of pre fecth range

for (int i=1;i<=50;i++)
 {
    hold_x = 51 + (int)(Math.random() * 100);
    hold_y = (int)(Math.random() * 101);
    //System.out.println(hold_x+" "+hold_y);
    //System.out.println(hold_y);
    prefetchnodes[i] = new nodes(hold_x, hold_y);
 }

 //System.out.println("***");
```

```
      // System.out.println("***");
    //int tmpx, tmpy;
    //sorting
     for (int i=1; i<51; i++) {
     for (int j=1; j<51-i; j++)
        if (prefetchnodes[j+1].x < prefetchnodes[j].x) {  /* compare the two neighbors */
         tmpx = prefetchnodes[j].getnodeX();
         tmpy = prefetchnodes[j].getnodeY();
             /* swap a[j] and a[j+1]      */
         prefetchnodes[j].setnodeX(prefetchnodes[j+1].getnodeX());
         prefetchnodes[j].setnodeY(prefetchnodes[j+1].getnodeY());

         prefetchnodes[j+1].setnodeX(tmpx);
         prefetchnodes[j+1].setnodeY(tmpy);

        }
    }


     /*
     for (int j=1;j<=50;j++)
     {

        System.out.println(prefetchnodes[j].x+" "+prefetchnodes[j].y);
      }
     */
    //merging both current and prefetch nodes
     for (int i=1;i<=50;i++)
     {
        mergedarray[i]          =         new           nodes(currentnodes[i].getnodeX(),
currentnodes[i].getnodeY());
     }
     int p=1;
     for (int i=51;i<=100;i++)
     {
        mergedarray[i]          =         new          nodes(prefetchnodes[p].getnodeX(),
prefetchnodes[p].getnodeY());
        p=p+1;
     }


    System.out.println("***");
        System.out.println("***");

     for (int j=1;j<=100;j++)
     {
        System.out.println(j+". "+mergedarray[j].x+" "+mergedarray[j].y);
      }

    System.out.println("***");
        System.out.println("Time required and messages sent to pre-fetch:");

     for (int i = 51; i <=60; i++){
        int flag = 0;
        for (int n=50; n>=1;n--){
            if       ((dist(mergedarray[n].x,         mergedarray[n].y,         mergedarray[i].x,
mergedarray[i].y)<=50) && (dist(0 , 50, mergedarray[n].x, mergedarray[n].y)<=50))
            {
                 System.out.println(i+". 3  51");
                 time[i] = time[i] + 3;
                 message[i] = message[i] + 51;
                 flag = 1;
                 break;
            }
        }
        if (flag == 0)
            System.out.println(i+". 5  71");
            time[i] = time[i] + 5;
            message[i] = message[i] + 71;


     }

     for (int i = 61; i <=70; i++){
     int flag = 0;
     for (int n=50; n>=1;n--){
            if       ((dist(mergedarray[n].x,         mergedarray[n].y,         mergedarray[i].x,
mergedarray[i].y)<=50) && (dist(0 , 50, mergedarray[n].x, mergedarray[n].y)<=50))
            {
                 System.out.println(i+". 3  51");
                 time[i] = time[i] + 3;
                 message[i] = message[i] + 51;

                 flag = 1;
                 break;
            }
        }
        if (flag == 0)
            System.out.println(i+". 7  91");
            time[i] = time[i] + 7;
            message[i] = message[i] + 91;
    }

    for (int i = 71; i <=80; i++){
     int flag = 0;
     for (int n=50; n>=1;n--){
         if      ((dist(mergedarray[n].x,        mergedarray[n].y,        mergedarray[i].x,
mergedarray[i].y)<=50) && (dist(0 , 50, mergedarray[n].x, mergedarray[n].y)<=50))
         {
             System.out.println(i+". 3  61");
             time[i] = time[i] + 3;
             message[i] = message[i] + 61;

             flag = 1;
             break;
         }
      }
      if (flag == 0)
         System.out.println(i+". 9  111");
         time[i] = time[i] + 9;
         message[i] = message[i] + 111;


    }

    for (int i = 81; i <=81; i++){
     int flag = 0;
     for (int n=50; n>=1;n--){
         if      ((dist(mergedarray[n].x,        mergedarray[n].y,        mergedarray[i].x,
mergedarray[i].y)<=50) && (dist(0 , 50, mergedarray[n].x, mergedarray[n].y)<=50))
         {
             System.out.println(i+". 3  71");
             time[i] = time[i] + 3;
             message[i] = message[i] + 71;

             flag = 1;
             break;
         }
      }
      if (flag == 0)
         System.out.println(i+". 50  200");
         time[i] = time[i] + 50;
         message[i] = message[i] + 200;
    }







    for (int i = 82; i <=86; i++){
    int flag = 0;
    for (int n=50; n>=1;n--){
        if ((dist(mergedarray[n].x, mergedarray[n].y, mergedarray[i].x,
mergedarray[i].y)<=50) && (dist(0 , 50, mergedarray[n].x, mergedarray[n].y)<=50))
        {
            System.out.println(i+". 3  71");
            time[i] = time[i] + 3;
            message[i] = message[i] + 71;

            flag = 1;
            break;
        }
     }
     if (flag == 0)
         System.out.println(i+". 11  131");
         time[i] = time[i] + 11;
         message[i] = message[i] + 131;


    }

    for (int i = 87; i <=100; i++){
    int flag = 0; int check = 0;
     for (int n=60; n>=51;n--){
        if (dist(mergedarray[n].x, mergedarray[n].y, mergedarray[i].x,
mergedarray[i].y)<=50)
        {
            System.out.println(i+". 7  111");
            time[i] = time[i] + 7;
            message[i] = message[i] + 111;

            flag = 1;
            break;
        }
     }
     if (flag == 0)
      for (int n=70; n>=61;n--){
            if        (dist(mergedarray[n].x,        mergedarray[n].y,        mergedarray[i].x,
mergedarray[i].y)<=50)
            {
                System.out.println(i+". 9  131");
                time[i] = time[i] + 9;
                message[i] = message[i] + 131;

                check = 1;
                break;
            }
         }

         if ((flag==0) && (check==0))
            System.out.println(i+". 50  200");
            time[i] = time[i] + 50;
            message[i] = message[i] + 200;


      }
    }

    //average of 100 runs
    for (int i=51;i<=100;i++){
       time[i]=time[i]/200;
       message[i]=message[i]/200;
    }

    for (int i=1;i<=100;i++)
        System.out.print(i+".  "+time[i]+"  ");

    System.out.println("");

     for (int i=1;i<=100;i++){

        System.out.print(i+".  "+message[i]+"  ");

     }

         // Add your handling code here:
   }//GEN-LAST:event_jButton2ActionPerformed

   private   void   jButton6ActionPerformed(java.awt.event.ActionEvent   evt)   {//GEN-
FIRST:event_jButton6ActionPerformed
       try{
      y = Integer.parseInt(jTextField1.getText());// Add your handling code here:*/
      r = Integer.parseInt(jTextField2.getText());// Add your handling code here:*/
    JOptionPane.showMessageDialog(null,"Node          residence          square->(0,"+(y-
r)+"),(0,"+(y+r)+"),("+3*r+","+(y-r)+"),("+3*r+","+(y+r)+").");//  Add  your  handling
code here:*/

    X = new moving(0,y);
    //X.print();

    int x_co,y_co;
    try {
    for (int i=1; i<=9; i++)
    {
      JOptionPane.showMessageDialog(null,"Node"+i);
      x_co = Integer.parseInt(JOptionPane.showInputDialog("Enter X co-ordinate"));//
Add your handling code here:*/
      y_co = Integer.parseInt(JOptionPane.showInputDialog("Enter Y co-ordinate"));//
Add your handling code here:*/

      totalnodes[i] = new nodes(x_co,y_co);
    }

    X.print();

    System.out.println("Nodes in the MANET are: ");

    for (int i=1; i<=9; i++)
    {

      System.out.print("-( "+totalnodes[i].x+" , "+totalnodes[i].y+" )-");

    }

    System.out.print("\n");

    System.out.println("The probable pre-fetching nodes at time t1 are: ");
   try {
    for (int j=1; j<=9;j++){

      if  ((dist(0,  y,  totalnodes[j].x,  totalnodes[j].y)>r)  &&  (totalnodes[j].x>r)  &&
(totalnodes[j].x<= (2*r)) /*&& (dist(4, 4, totalnodes[j].x, totalnodes[j].y)<=4.0)*/)
      {
       System.out.println("("+totalnodes[j].x+" , "+totalnodes[j].y+" )");
       t1[j] = j;
      }
    }
    if (prefetch(t1,10)==true)
     System.out.println("None.");
```

```
      else{
         result_t1(t1);
         DSR_t1(t1);
         DSDV_t1(t1);
      }


System.out.println("********************************************************
*************");

System.out.println("********************************************************
*************");

System.out.println("********************************************************
*************");

      System.out.println("The probable pre-fetching nodes at time t2 are: ");
      for (int k=1; k<=9;k++){

          if ((dist(r, y, totalnodes[k].x, totalnodes[k].y)>r) && (totalnodes[k].x> (2*r)) &&
(totalnodes[k].x<= (3*r)) /*&& (dist(8, 4, totalnodes[k].x, totalnodes[k].y)<=4.0)*/){
          System.out.println("("+totalnodes[k].x+" , "+totalnodes[k].y+" )");
          t2[k] = k;
       }
      }
      if (prefetch(t2,10)==true)
        System.out.println("None.");
      else{
         int q=0;
         //System.out.println("iiiiiiiiiiiiiiiiiiiiiii");
         q = firstnode(t1,10);
         //System.out.println(q);

        result_t2(t2, q);
        DSR_t2(t2, q);
        //int ql = lastnode(t1,10);
         //System.out.println(q);

        DSDV_t2(t2, q);

      }

     }catch (NumberFormatException f) {

      //System.out.println("hi:");
     }
```

```
    for (int l=1; l<=10;l++){
      System.out.print(t1[l]+" ");
    }

    System.out.print("\n");

    for (int m=1; m<=10;m++){

      System.out.print(t2[m]+" ");
    }

    }catch (NullPointerException e){
    //System.out.println("hiiiii");
    }
  }catch (NumberFormatException f){
  JOptionPane.showMessageDialog(null,"Input Error, Missing Y or r or node co-
ordinate");
  }

   //result();



             // Add your handling code here:
  }//GEN-LAST:event_jButton6ActionPerformed

  private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jTextField2ActionPerformed
     // Add your handling code here:
  }//GEN-LAST:event_jTextField2ActionPerformed

  private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jTextField1ActionPerformed
     // Add your handling code here:
  }//GEN-LAST:event_jTextField1ActionPerformed

  private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton7ActionPerformed
//        y = Integer.parseInt(JOptionPane.showInputDialog("Enter starting y co-ordinate
of X"));// Add your handling code here:*/
 //          r = Integer.parseInt(JOptionPane.showInputDialog("Enter communication
radius"));// Add your handling code here:*/
    //  JOptionPane.showMessageDialog(null,"Node residence square->(0,"+(y-
r)+"),(0,"+(y+r)+"),("+3*r+","+(y-r)+"),("+3*r+","+(y+r)+").");// Add your handling
code here:*/
```

```
        // Add your handling code here:
    }//GEN-LAST:event_jButton7ActionPerformed

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
System.exit(0);       // Add your handling code here:
    }//GEN-LAST:event_jButton1ActionPerformed

    private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_exitMenuItemActionPerformed
        System.exit(0);
    }//GEN-LAST:event_exitMenuItemActionPerformed

    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt) {//GEN-
FIRST:event_exitForm
        System.exit(0);
    }//GEN-LAST:event_exitForm

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {

        new Application_2().show();
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JMenuBar menuBar;
    private javax.swing.JMenu fileMenu;
    private javax.swing.JMenuItem openMenuItem;
    private javax.swing.JMenuItem saveMenuItem;
    private javax.swing.JMenuItem saveAsMenuItem;
    private javax.swing.JMenuItem exitMenuItem;
    private javax.swing.JMenu editMenu;
    private javax.swing.JMenuItem cutMenuItem;
    private javax.swing.JMenuItem copyMenuItem;
    private javax.swing.JMenuItem pasteMenuItem;
    private javax.swing.JMenuItem deleteMenuItem;
    private javax.swing.JMenu helpMenu;
    private javax.swing.JMenuItem contentsMenuItem;
    private javax.swing.JMenuItem aboutMenuItem;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton6;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JButton jButton7;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JButton jButton2;
    // End of variables declaration//GEN-END:variables

}
```

VITA 2

Muhammad Tanvir Alam

Candidate for the Degree of

Master of Science

Thesis: PROTOCOL FOR REDUCING COMMUNICATION DELAY IN MOBILE AD HOC NETWORKS BY PRE-FETCHING LOCATION INFORMATION

Major Field: Computer Science

Biographical:

Personal Data: Born in Dhaka, Bangladesh on December 31, 1977, the eldest son of Umme Fatema and Mukarramul Bari.

Education: Graduated from Computer Science Department, North South University, Dhaka, Bangladesh in May, 1999; received Bachelor of Science degree in Computer Science. Completed the requirements for the Master of Science degree at Oklahoma State University in December 2002.

Professional Experience: May 1999 – May 2000: Teaching Assistant North South University, Dhaka, Bangladesh
August 2001 – Present: Research Assistant (Web Application Developer), Department of Political Science, Oklahoma State University, Stillwater