

**Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596**

**ROUTING MESSAGES TO MIGRATING PROCESSES
IN LARGE DISTRIBUTED SYSTEMS**

Thirumalai Muppur Ravi

**August 1989
CSD-890049**

UNIVERSITY OF CALIFORNIA

Los Angeles

Routing Messages to Migrating Processes
in Large Distributed Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

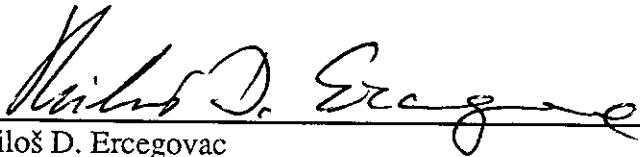
by

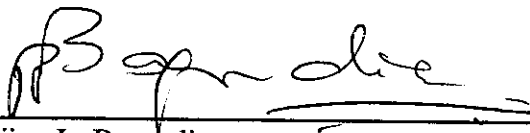
Thirumalai Muppur Ravi


1989

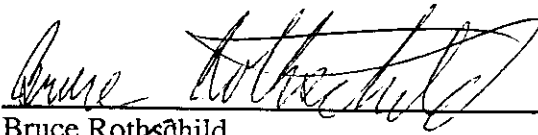
© Copyright by
Thirumalai Muppur Ravi
1989

The dissertation of Thirumalai Muppur Ravi is approved.


Miloš D. Ercegovac


Rajive L. Bagrodia


Kirby A. Baker


Bruce Rothschild


David R. Jefferson, Committee Chair

University of California, Los Angeles

1989

To my parents

Kumudini Srinivasan and T. M. Srinivasan

and my brother

Arun

Table of Contents

	page
I Introduction	1
1.1 Motivation	1
1.2 Problem Description	3
1.3 Related Research	5
1.4 Research Contributions	11
II The Basic Protocol	13
2.1 The Model	13
2.2 The Basic Protocol	14
2.3 Correctness Argument	29
2.4 Extended Protocol	37
III Routing Caches in Distributed Systems	39
3.1 Locality in Distributed Systems	39
3.2 Adaptive Routing with Caches	41
3.2.1 Introduction	41
3.2.2 Cache Structure	43
3.2.3 Process Caches	44
3.2.4 Node Caches	52
3.3 Analysis of Caching Schemes	54
3.3.1 Analytical Model	55
3.3.1.1 Modeling Temporal Locality	55
3.3.1.2 Performance Model for PCS (CM)	57
3.3.2 Simulation of Caching Schemes	61
IV Non-Uniform Updating of Routing Entries	73
4.1 Introduction	73
4.2 Shifting Neighborhood Protocol	76
4.3 Shifting Neighborhood Protocol with Dynamic Broadcast	83
4.4 Shifting Neighborhood Protocol with Adaptive Routing	87
4.5 Moving Home Node Protocol	90
4.6 Analysis and Performance Evaluation of the Shifting Neighborhood Protocol	

.....	94
4.6.1 Determination of the Optimal Neighborhood Structure	94
4.6.2 Stretch Ratios for the Shifting Neighborhood Protocol	112
4.7 Simulation of the Shifting Neighborhood Protocol and Related Protocols	127
4.7.1 Model of Message Communication between Processes	128
4.7.2 Model for Process Migration	130
4.7.3 Organization and Parameterization of the Simulation	132
4.7.4 Results of the Simulation of the Shifting Neighborhood Protocol and Related Protocols	137
4.7.5 Conclusions from the Simulation	169
V Other Issues	172
5.1 Locating Remote Routing Entries	172
5.1.1 Wave Searching	172
5.1.2 Hashing	177
5.2 Reliability Issues	178
5.3 Unbounded Migration-Count	180
VI Conclusions and Future Research	181
6.1 Conclusions	181
6.2 Future Work	186

ACKNOWLEDGEMENTS

The problem in this dissertation was proposed to me by Professor David Jefferson, my advisor. His dedication to the pursuit of innovative and original ideas, his scientific temperament, open-mindedness, intellectual rigor and emphasis on perfection have been a great source of inspiration. I am grateful to him for providing a very healthy and conducive environment for research and for treating me as a fellow colleague rather than a student. David has been actively involved in my research and this work should properly be viewed as a collaborative effort.

During my stay at UCLA, I have closely worked with Professors Milos Ercegovic and Tomas Lang who originally sparked my interest in computer architecture and distributed systems. The opportunity to interact with them has been a valuable one.

I would like to thank Professor Estrin for his interest in my research activities and his continuous encouragement of my endeavors. Professor Dick Muntz has always (particularly at odd hours and on weekends!) been available to provide honest and constructive criticism, to allow me to run ideas by him, and to pick his brain. Professor Jack Carlyle generously made available the computer resources needed to complete my computationally intensive simulations.

Unni Warriar, Balaji Narasimhan, Joe Bannister, Farshad Meshkinpour and Anne Lam have provided me with friendship, encouragement and support in numerous ways over the years. It is my particular pleasure to acknowledge their help. I would like to thank Unni, Anne and Balaji for giving me large amounts of computer time and Balaji for reading earlier drafts of my dissertation. I am grateful to Linda Keene for encouragement and for her pride in my accomplishments.

My colleagues in the UCLA Computer Science Department created an unique environment and offered the companionship which made life at UCLA an experience that I will always cherish. We enjoyed lively lunch time discussions, concerts at Royce Hall and countless parties (including our graduation party : do we know how to throw a party or what ?) together. In particular they are: Leon Alkalaj, Pak Chan, Maria Eugenia Fuenmayor, Hector Geffner, Art Goldberg, Miquel Huguet, Jeong-A Lee, Jaime Moreno, Martine Schlag, Frank Schaffa, Marc Tremblay and Paul Tu. In particular I would like to thank Miquel, who shared the same office with me for five years, for being a close friend, for his wonderful dinner parties, and for keeping me well informed about the steamier happenings in the department.

Doris Sublette and Verra Morgan have made the department a much nicer place to be by their warmth, affection and genuine concern for students.

I would like to acknowledge the following for their time and assistance: Bob Felderman, Chris Ferguson and Tom Verma for help with probability theory and other assorted problems, Lance Kurisaki for assistance with psfig and Greg and Tiffany Frazier for helping me track down bugs and solve problems related to SIMON.

The funding of this work by the JTF/JPL Contract No. 957523 and an IBM Graduate Fellowship is gratefully acknowledged.

I am specially grateful to Alison Baylor for her love and the wonderful times we had together.

Finally, I dedicate this dissertation to my parents Kumudini and T. M. Srinivasan and my brother Arun for their patience, faith, love and understanding during these long years.

Thank you all !

VITA

January 9, 1961 Born, Champaign, Illinois

1982 B.Tech., Electrical Engineering
Indian Institute of Technology
Kanpur, India

1982-1984 Research Assistant
University of California
Los Angeles, California

1984-1989 Post Graduate Research Engineer
University of California
Los Angeles, California

1986 M.S., Computer Science
University of California
Los Angeles, California

1987-1989 IBM Graduate Fellowship

PUBLICATIONS AND PRESENTATIONS

T. M. Ravi and David Jefferson., *A Basic Protocol for Routing Messages to Migrating Processes*, 1988 International Conference on Parallel Processing, August 15-19, 1988;

Unni Warriar, T. M. Ravi and B. Narasimhan., *A Classification of Processes in Distributed Discrete-Event Simulation*, Southeastern Simulation Conference, Orlando, Florida, October 17-18, 1988;

T. M. Ravi, M. D. Ercegovac, T. Lang and R. R. Muntz., *Static Allocation for a Data Flow Multiprocessor System*, Second International Conference on Supercomputing, Santa Clara, CA, May 1987;

M. D. Ercegovac, P. K. Chan and T. M. Ravi., *A dataflow multiprocessor architecture for high speed simulation of continuous systems*, Proc. International Workshop on High-Level Architecture, Los Angeles, CA, May, 1984;

M. D. Ercegovac, P. K. Chan, Z. Konstantinovic, T. M. Ravi and M. D. F. Schlag., *Task Partitioning, Allocation and Simulation for a Dataflow Multiprocessor System*, Proc. Summer Computer Simulation Conference, Boston, MA, 1984;

F. Meshkinpour and T. M. Ravi., *Instruction Set Partitioning for VLSI Architectures*, International Conference on Computers, Systems & Signal Processing, Bangalore, India, Dec. 1984;

T. M. Ravi, *Partitioning and Allocation of Functional Programs for Data Flow Processors*, M.S. Thesis, Report No. CSD-860063, Department of Computer Science, University of California, Los Angeles, April 1986;

T. M. Ravi and M. D. Ercegovac, *Allocation for the SANDAC Multiprocessor System*, Report No. CSD-860059, Department of Computer Science, University of California, Los Angeles, Feb. 1986.

ABSTRACT OF THE DISSERTATION

Routing Messages to Migrating Processes in Large Distributed Systems

by

Thirumalai Muppur Ravi
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 1989
Professor David R. Jefferson, Chair

We investigate process migration in large distributed systems with message passing. Our objective is to develop mechanisms for routing messages to and from moving processes, that can scale up to thousands of nodes. In particular we develop mechanisms where nodes have *out-of-date* routing tables, i.e. new routing updates indicating that a process has moved is not broadcast throughout the network; and *incomplete* routing tables, i.e. a node will generally have routing information about only some of the processes in the system.

We first present a Basic Routing and Migration (BRM) protocol which captures the fundamental synchronization and correctness issues associated with process migration and the routing of messages to processes. The BRM protocol is shown to be deadlock free and we show that under this protocol messages are eventually delivered to their destination processes. The BRM protocol serves as a basic building block which is extended to give protocols that address performance issues.

To take advantage of *temporal* localities in communication we propose schemes for caching routing entries. Caches help to reduce the time to locate a process (i.e. obtain a routing entry for a process) and provide a mechanism to keep routing entries up-to-date.

The main contribution of our work is the *Shifting Neighborhood Protocol (SNP)* for updating routing information in a large distributed system. Associated with a process, in SNP, are a family of nested neighborhoods consisting of nodes of the system. Routing entries located at nodes in different neighborhoods are updated with different frequency depending on the location of the process. The number of neighborhoods for a process corresponds to the number of levels in the protocol and the multi-level organization of SNP serves to reduce the update bandwidth.

We propose an analytical model to derive the optimal protocol parameters, i.e. the number of levels and the dimensions of the neighborhoods that results in minimum update bandwidth in SNP. From the analysis we determine that the average update bandwidth is a slowly increasing function of the size of the system while the average message trajectory stretch ratio (ratio of number of hops taken by message to the number of hops in the shortest path between source and destination) is independent of the size of the system.

We perform an extensive simulation of the Shifting Neighborhood Protocol running on a large distributed system and study the effect of the variation of protocol and system parameters on the performance and overhead of the protocol. The results of the simulation indicate that SNP is scalable with the size of the system and hence suitable for implementation in very large systems.

CHAPTER I

Introduction

1.1 Motivation

It is now common to have systems consisting of a large number of computers connected by a local area network. It is also becoming feasible to implement large scale Multiple-Instruction-Multiple-Data (MIMD) multiprocessors with thousands of nodes connected by an interconnection network. Nodes in such systems share no memory and communicate with each other by sending messages.

In order to effectively utilize the resources available in such large distributed systems and the potential they provide for improvement in performance, availability and reliability we have to implement *dynamic load management*. Dynamic load management is the attempt to optimize performance at run time by reallocating resources dynamically in order to adapt to changing system characteristics and demand of resources by applications. Dynamic load management is a complex problem with many different objectives and several approaches have been proposed to deal with different aspects of load management.

With a number of resources available it is possible to exploit large degrees of parallelism at the process or task level by distributing processes over different nodes. In a distributed system where processes belonging to different applications have been assigned to nodes statically, the load will vary from node to node during the course of execution. The load is a measure of the utilization of resources like the CPU, memory, I/O, communication links, special purpose accelerators etc. at a node. While some nodes are underutilized, others may be so heavily loaded as to become a critical performance bottleneck. By *dynamic load balancing* which is a component of load management, processes are moved from more heavily loaded nodes to less heavily loaded nodes, in order to balance the load on different nodes.

If the application consists of a number of communicating processes which cooperate with each other towards a common goal, then the frequency and amount of communication between processes can dynamically change over time. The objective of the *dynamic mapping problem* is to dynamically assign processes to nodes in order to minimize the overall communication between processes.

In order to minimize the execution time of an application consisting of communicating processes which cooperate with each other towards a common goal, load balancing is not adequate. Some processes may be more critical than others, and the degree to which a process is critical may change from time to time as the execution proceeds. Thus under dynamic load management, more resources need to be allocated to currently critical processes.

Another goal of dynamic load management is reliability. If a node is going down then all its processes can be migrated to other nodes. Or else if a node abruptly fails then the processes at that node can be restarted at other nodes if the process image is stored in stable storage or if a copy of the image is maintained at another site by checkpointing the process.

In a system where processes communicate with each other some processes that may be common to several applications (for example server processes) receive a large number of messages that have to be serviced. These processes can be replicated with copies spawned at different nodes and consistency messages sent between copies of a process if necessary. By replication the availability of a process is increased, the distance between other processes and a copy of the process is reduced and the traffic congestion and the length of queues at the process is decreased. In addition the copies of the process can execute concurrently.

We have identified two tools for dynamic load management of a distributed system - *process migration* and *process replication*. This work will focus on process migration and in particular migration in large distributed systems. Process cloning has been dealt with elsewhere [Goldberg 88].

Most current research in process migration is concerned with the *policy* of process migration, i.e. deciding which process is to be moved, when, and from which source node to which destination node ([Ni85], [Lin 87] & [Zhou 87]). Policy issues are concerned with what load index to use, algorithms for exchanging load information with

other nodes, algorithms for deciding when to move a process and where to move it and so on. Here we are concerned instead with the *mechanics* of process migration, i.e. protocols for moving a process from one node to another and the delivery of messages to processes that are moving. In particular, the emphasis of this work is to develop mechanisms for routing messages to and from migrating processes that can scale up to large distributed systems with thousands of nodes.

1.2 Problem Description

Consider a large network of nodes executing processes that send messages to each another asynchronously. A process may send a message to any other process at any time, addressing it by a globally unique name. We choose a general model of message communication. The results that we obtain for the general model of message communication can be applied to other more specific communication mechanisms.

The set of processes to which a process sends messages or from which it receives messages may be continuously changing. This is contrast to the model where processes communicate with a fixed set of other processes and the operating system has prior knowledge of which processes will communicate with each other.

Processes can migrate from one node to another dynamically based on load management policies. A process can be moved to any node, not just an adjacent node. We can not restrict the migration of a process to nodes belonging to a certain region in the network. We make no assumptions on the rate at which processes migrate - some processes may migrate frequently while others rarely, if at all, migrate.

Our objective is to route messages to migrating processes, so that messages are eventually delivered to their destination processes. The main technical issues arise from the fact that from the operating system's point of view both messages and processes are in motion, and messages must be routed to moving targets instead of fixed targets. As processes are moving, a node has to first determine the location of a process before sending messages to the node where it thinks the process is located. The location of a process is determined using routing tables containing entries that map process names to node addresses. Since in a distributed system with communicating processes it is impossible to predict at the time a process is to migrate whether or not there is a message in transit toward it, hence some messages will have to be *forwarded* to catch up with

their targets.

We are only interested in migration and routing mechanisms that *scale* up to thousands of nodes. In particular we are concerned with minimizing the bandwidth required to update routing tables and the storage required to store routing tables. Message routing must rely on possibly *out-of-date* routing information, i.e. new routing updates indicating that a process has moved may not be broadcast throughout the network. We also cannot assume that a node has enough memory to keep a complete routing table, mapping every process name to a node address. In our routing protocol, routing tables may have *incomplete* information, i.e. a node will generally have routing information about only some of the processes in the system.

Routing of messages to a process must be *efficient*, using paths to a process that are as short as possible. Since messages are routed based on out-of-date routing information and a process may be moving even while the message is enroute to the process, the path that a message takes before it is delivered to a process will be larger than the shortest distance between the source node and the destination process at the time of delivery. The performance metric of interest that is minimized is the *stretch ratio* - the ratio of the length of the path taken by the message to the shortest distance between the source node and the destination process at the time of delivery.

It is essential that the routing mechanism be *adaptive* and make routing decisions based on changes in the system. The routing mechanism should adapt to the specific pattern of communication between processes. Repeated communication between processes should result in an amortized cost of communication between them. The routing mechanism should also adapt to the migration of processes and not degrade in performance over time.

In designing any algorithm for distributed systems an objective is to develop *robust* algorithms. The failure of a few nodes and hence the routing tables contained in them should not make a large number of processes inaccessible. The routing mechanism should degrade slowly on the failure of nodes. All nodes should have the same function as far as the routing mechanism is concerned, containing similar amounts of routing information and receiving nearly the same number of requests for routing information. Solutions where select special nodes in a region are designated to maintain up-to-date and complete routing information are not acceptable solutions.

We will be interested only in forwarding messages to processes and will not consider the problem of open files of the process to be moved. Our work concerning the migration of processes is also applicable to the migration of files. Files can be considered to be a restricted type of process and messages to and responses from open files at remote nodes can be considered to be a special case of communication between processes.

1.3 Related Research

A *name server* [Terry 85] is a facility for providing global names for objects that allows for the access of information about objects. Objects can be active like processes or passive like files and devices. In a distributed system consisting of a collection of host computers all globally accessible objects should be given unique names that can be used by distributed applications.

Another important function of name servers is to bind an object name to its location. The name server has to dynamically maintain information on objects whose location is changing or which are being created and destroyed. However name servers provide many more services than just locating objects. They include interpretation of memory addresses, locking and access control of data objects, authentication, and information like size, creation time etc.

In this work we focus on one aspect of name servers, that is locating objects. In particular we are only interested in locating processes that may be moving from one node to another. Work on distributed name servers ([Birrell 82], [Oppen 83] & [Lampson 86]) concentrates on the conventions for naming objects and generally assume that the properties of a name change slowly. Issues such as what information to keep on local name servers, the location of local name servers and when to update local servers are not dealt with and left as open issues.

Process migration and forwarding of messages has been implemented in the DEMOS/MP system [Powell 83]. Our basic routing and migration (BRM) protocol can be considered to be a more formal specification and generalization of their migration procedure with particular concern for synchronization issues that arise when a process has to be moved from one node to another. Our model permits concurrent execution of several instantiations of the protocol on the same node. Our model also allows for the situation that a node may not have a routing entry for every process in the system. The

main emphasis of our work, however, is to develop routing protocols rather than migration protocols.

Fowler ([Fowler 85] & [Fowler 86]) proposed algorithms using forwarding addresses to locate moving objects in a distributed system. In particular he proposed three protocols for finding the address of a object that has moved. The most interesting protocol is based on the UNION_FIND path compression algorithm. The main feature of this approach is that routing entries used to forward a message to the destination are sent updates with the current location of the object when the message is delivered to the destination. In our protocol, we do not restrict ourselves to updating of routing entries when an object is accessed, but also consider schemes where maintenance messages are propagated as a side-effect of process migration. A problem with Fowler's scheme and in general with using forwarding addresses is that a single failure in an object's forwarding address chain makes the object inaccessible.

Gopal and Segall [Gopal 88] consider a very similar problem to ours in the context of providing a directory service for users to dynamically locate other users. They specify in detail two protocols - the first protocol defines a hierarchical relationship between directory servers that permits efficient querying and updating of remote user information. The essential idea is to partition the network into regions and select one node in each region to act as a regional directory server. Each regional server maintains knowledge of all local users in that region. Several logically related regions are grouped into super-regions and one node in each such super-region is designated a super-server and it maintains information on all users in that group of regions. The hierarchy can have a number of levels and super-regions can be grouped into super-super-regions and so on. Earlier in our research we considered this hierarchical approach and decided not to pursue it because of several problems associated with tree structured schemes. The main problem with tree based schemes is that different nodes have different functions in the routing scheme. The servers close to the root of the tree receive higher traffic and have to store information about a larger number of users. Moreover the reliability of tree based schemes is poor and the failure of servers particularly those close to the root have a catastrophic effect. Another problem is the partitioning of the network into regions. Nodes that are adjacent to each other in the network can be assigned to different regions and hence the overhead of locating some nodes can be very high compared to the actual distance between them. We develop protocols where all nodes have the same function in the routing scheme and are treated uniformly.

The second protocol proposed by Gopal and Segall is a non-hierarchical approach that requires all servers to maintain information on all users. Whenever there is a change in the location of a user this information is flooded throughout the network. The advantage of this scheme over ours is that message sequence numbers are not used but rather the Propagation of Information with Feedback (PIF) protocol [Segall 83] is used to guarantee the correctness of the protocol. However for the design of a practical scheme for large networks, we consider the cost of unbounded flooding every time a user moves prohibitive in terms of communication bandwidth.

Hwang [Hwang 87] proposes the replication of a location service to improve availability and efficiency. If one replica is inaccessible then a different one can be accessed and if several replicas are present, the client can choose the closest replica to send its query. The replicas are kept consistent using Liskov's [Liskov 86] multipart timestamp technique. The problem with using any scheme with a few location services is that the location services become a bottleneck because of the high message traffic requesting for service. Also with few location services the distance from any node to the location service is large resulting in a large message trajectory stretch ratio. The *message trajectory stretch ratio* is the ratio of the number of hops that a message takes to be delivered to its destination and the actual distance in hops between the source and the destination. The goal of routing schemes and in particular, the routing scheme that we are developing is to keep this ratio as close to one as possible. If we assume that the model of message communication is such that most nodes send messages to destinations very close to it, then Hwang's scheme with a few location services will have a very high stretch ratio. Hence, if we decide to have a large number of replicated services then because of communication bandwidth considerations we need a more sophisticated scheme to update the replicas. Keeping all the replicas up-to-date by spreading frequent gossip messages is unnecessary because nodes can use out-of-date information for routing and because some nodes rarely need information on some routing entries. In Chapter IV we develop a scheme to maintain a large set of location services or, as we prefer to call them routing tables.

Ahamad et. al. [Ahamad 87a] have proposed a scheme to locate objects when the underlying communication network has the multicast capability. The scheme handles object creation, migration and deletion. In our protocol we do not assume the existence of a broadcast medium and have a model of a network with point to point connections.

Mullender and Vitanyi [Mullen 85] studied the resource location problem for a *client-server* model in the context of the Amoeba operating system. The server posts its location to a set of nodes, and the client queries another set of nodes for the desired service. A *rendezvous* node is a node in the intersection of the two sets where the client node finds the location of the server node. They do not consider the routing of messages that may be in transit when a process moves, or messages that are sent based on old location information which has not yet been updated.

The Emerald system [Jul 88] has implemented mobility of process objects as well as data objects. To find an object it incorporates the forwarding address protocol proposed by Fowler ([Fowler 85] & [Fowler 86]). When a node is unreachable due to failure or the loss of a forwarding address, the Emerald system resorts to a system-wide broadcast to find the object.

The Eden system [Black 85] treats resources as objects that are location-independent. Currently the Eden system implements mobility for inactive objects, i.e. objects that have no messages in transit either being sent or received. An object is located by first checking a local cache for the address. If it is not found, a broadcast message requesting the location of the object is sent to all other hosts.

In both Locus [Popek 85] and the Sprite operating system [Douglass 87 & Ouster 88] processes use kernel calls instead of messages for interaction with other processes. Each process is assigned a special origin site or home node to which all remote calls are sent. The special node has a forwarding address to the current location of the process to which all calls are forwarded. However this strategy has several problems. There is a residual dependency of the process on the special node even after the process has migrated several times. In large distributed systems over a period of time, the additional path length for the call could become very large compared to the actual distance to a process. When all communication to a process is required to go through a special node, process migration is no longer effective in reducing communication distances. Moreover the failure of the special node makes the process inaccessible to other processes in the system. Our protocols for routing messages to migrating processes will be adaptive and not degrade in performance over time.

In the V-System [Theimer 86], when a message has to be routed to a process, a local cache is checked for the location of the destination process. If a cache entry is found then the message is sent to the location indicated. If there is no entry for the process at the local cache, or if the process is no longer at the location indicated by the cache entry then a query is broadcast system-wide to all nodes requesting the location of the process. The local cache is updated with the response to the query and the message is forwarded to the new location. We develop protocols that will not require a system-wide broadcast of queries or updates.

In the Accent system [Rashid 81] every time a migration takes place, the destination site broadcasts the change in location to all other sites. Like the scheme used in the V-system this scheme is not scalable.

Lu, Chen, and Liu [Lu 87] have proposed a migration mechanism that assumes that each process communicates with a finite number of processes called *adjacent* processes and each node has a table with the exact location of processes adjacent to resident processes. When a process migrates, it first blocks its adjacent processes from sending messages, migrates to a new node, sends the adjacent processes the new location of the process and re-enables the adjacent processes to send messages to the recently migrated process. This scheme ensures that there is no message for a process in transit when it migrates. In our protocol we tackle a more general problem assuming any process can communicate with any other process at any time, and that the operating system does not know which processes will communicate with each another.

Scheurich and Dubois [Scheur 87] deal with the problem of the migration and location of memory pages in distributed systems. The performance of their mechanism strongly depends on the geographical, spatial, and temporal locality of page references. It uses caches for hints, and uses a broadcast search when the page is not found. Our message routing scheme basically relies on routing entries and the forwarding of messages to a process that has moved. Extensions to our basic protocol proposed in Section 3.2 use caches to take advantage of localities.

Our problem is similar to the problem of routing to a mobile subscriber in packet radio systems with fixed stations [Kahn 78]. In packet radio networks, a flooding scheme is commonly used to initially locate a mobile subscriber. The weakness of this scheme, however, is the high channel bandwidth due to control messages. A modification of this scheme [Li 86] is to have a complete routing table at each station. To locate a mobile

subscriber a message is sent to the last known station to which the subscriber was affiliated. If the subscriber has moved out of range of that station then a flood broadcast is initiated at the last known station.

Westcott and Jubin [Westcott 82] propose an algorithm for routing in a packet radio network that is similar to the old ARPANET algorithm [McQuillan 78] where periodically a node exchanges its routing table with its neighbors and each node on receiving the updates computes its new table. Only that part of the routing table is sent which has changed since the last routing update. The main disadvantage of this approach is that the size of update messages increases linearly with the number of nodes in the network. We contend that it is not necessary to periodically exchange routing information in the entire system and have up-to-date information on the location of a packet radio (or a process in our case). We develop schemes that require less communication bandwidth based on the premise that nearby nodes need more up-to-date routing information while distant nodes can tolerate more out-of-date routing information. This is because our goal is to reduce the average stretch ratio and not to minimize the difference between the number of hops taken by a message to reach the destination and the actual distance in hops to the destination.

Several hierarchical routing algorithms have been proposed ([Amer 88], [Kamoun 76], [Kleinrock 77], [Lauer 86], [Peleg 87] & [Ramamoorthy 86]) that decompose the topology into hierarchically organized clusters. Clusters in turn can be organized into superclusters and so on. In several of the algorithms there is a special node associated with each cluster called the clusterhead. The algorithms proposed can be adapted for routing messages to nodes in a network where connectivity or link delays are dynamically changing. There are two advantages to the hierarchical routing algorithms in large distributed systems. Hierarchical schemes for routing to nodes have been shown to significantly reduce routing table space ([Kleinrock 77] & [Peleg 87]) without having to concentrate routing information in the clusterheads. The second advantage is that when a connectivity or other changes occur at a node, information about the change is spread only within the cluster that contains that node, thus restricting the number of update messages sent on every change.

A common feature of some ([Kleinrock 77] & [Peleg 87]) of the hierarchical schemes is that special names are assigned to nodes, and the name of a node indicates some information about its location. In our problem we are interested in routing messages to processes which do not have a fixed location but can migrate from cluster to

cluster. Thus from the name of a process a node can not determine which cluster it belongs to, and hence a similar saving in routing table space is not possible.

1.4 Research Contributions

In this work we have developed routing schemes with low storage and bandwidth requirements that route messages to moving processes. We assume the most general model of message communication, process migration and the underlying architecture. The schemes we have proposed can be adapted to systems that have a more restricted model.

We study the performance issues related to our schemes for routing messages to moving processes. We develop analytical models to study some performance metrics and simulation models for a more general and extensive evaluation of the schemes.

The main contributions of this research are summarized below.

- A basic routing and migration protocol that captures synchronization and correctness issues.
- Schemes for the use of caches to adapt to locality in distributed systems.
- The Shifting Neighborhood Protocol: A mechanism for updating routing information in large distributed systems.

The remainder of this dissertation is organized as follows. In Chapter II we present a formal specification of a Basic Routing and Migration (BRM) protocol and show it to be deadlock free and correct. The emphasis in this basic protocol is on synchronization and correctness issues associated with process migration and message routing. The BRM protocol is not intended to be an efficient protocol for either migrating processes or routing messages to processes, but rather a basic building block to which different extensions are added to improve the efficiency of routing messages.

In Chapter III, we propose the addition of caches to the BRM protocol to take advantage of localities present in distributed systems. By caching routing entries the time to locate a process (i.e. obtain a routing entry for a process) and the length of the path to the process can be significantly reduced in the presence of localities. We present three schemes for the addition of caches to the BRM protocol; two of them use caches associated with a process and the third scheme uses caches associated with a node. We describe analytic models for the performance of the caching schemes and simulate their performance.

We then present in Chapter IV the main result of our work - the Shifting Neighborhood Protocol for routing messages to moving processes. This protocol is a multiple level mechanism for updating routing entries for a process. Different routing entries for a process are updated with different frequency depending on the distance between their location and the process. The principal benefit is the saving in update bandwidth. The nodes of the system are partitioned into a family of nested neighborhoods for the process. The optimal number of levels and the dimensions of the neighborhoods that minimizes the update bandwidth for the Shifting Neighborhood Protocol and related protocols are analytically determined. We calculate the update bandwidth and the average message trajectory stretch ratio under certain assumptions of system. Extensive simulations of the Shifting Neighborhood Protocol executing on large distributed systems confirms that the protocol is scalable with the size of the system.

In Chapter V, we discuss how routing entries for a process can be obtained when nodes have incomplete routing tables. We discuss reliability issues related to our proposed protocol. Finally, in Chapter VI we summarize the contributions of our work and present related open problems.

CHAPTER II

The Basic Protocol

2.1 The Model

We envision a load managing operating system as consisting of five functional levels (Figure 2.1).

Application	cooperating processes
Load Management	migration policy
Mechanics of Routing & Migration	interprocess msg. routing migration protocol
Low Level Kernel	internode msg. routing process multiplexing
Architecture Level	many processors MIMD no shared memory

Figure 2.1 : Functional Levels in System

The lowest is the *Architecture* level consisting of the nodes and the network connecting them. The topology is assumed to be an arbitrary graph. The *Low Level Kernel* implements reliable node-to-node message routing. Communication between nodes is by asynchronous messages, takes finite time, and need not be order preserving. Each node may have multiple processes resident on it and this level also manages the multiplexing of processes on each node. The next layer handles the *Mechanics of Routing and*

Migration and includes protocols for process-to-process message routing, routing table maintenance, and migration of a process from one node to another. The *Load Management Policy Layer* makes decisions about how to redistribute the load and requests the *Mechanics of Routing and Migration* layer to migrate certain processes to specified destinations. Finally the *Application* layer consists of a collection of cooperating processes that communicate asynchronously by messages.

Each level of the system is assumed to be reliable. In this work we are interested in the implementation of the third layer, handling the *Mechanics of Routing and Migration*. Message routing between nodes is handled by the *Lower Level Kernel*, and we will not be concerned with how it is done.

2.2 The Basic Protocol

We now present a Basic Routing and Migration protocol, BRM, which forms a basis for the subsequent development of more complex protocols that optimize performance. The BRM is designed with careful attention to modularization to allow for different possible implementations of sub-modules and so that different features can be added on to the basic protocol for performance enhancement. This protocol assumes that buffers of unlimited size are available at each node to store messages while remote routing entries are being obtained.

Associated with each process, the operating system maintains a *migration-count* α which is incremented whenever the process migrates. Each node maintains a possibly incomplete *routing table* containing *entries* that among other things map process names to node addresses. Routing entries for a process are marked with a migration-count field that is equal to the migration-count of the process when it was located at the node indicated by the routing entry. The *status* of a routing entry indicates whether it is *permanent* or *temporary*, i.e. whether it remains in existence for the lifetime of the process or not. The *updates* field indicates the number of pending updates or update requests to remote nodes from the node where the entry is located. Before an update to remote nodes is started this field is incremented by one and when the update is complete it is decremented by one. A temporary routing entry can be deleted only when the value of this field is zero. A *lock* field controls access to the routing entry. The structure of a routing entry is shown in Figure 2.2. When a process becomes resident on a node, a new routing entry for that process is created at that node if one does not already exist.

Process Name	Lock	Location	Migration-count	Status	Updates
--------------	------	----------	-----------------	--------	---------

Figure 2.2 : Routing Entry

When a process is created, routing entries for the process are distributed to a subset of the nodes in the system and remain in existence for the lifetime of the process. It is not necessary that every node have a routing entry for every process. However each node U has access to a set $\&surrogate(U, q)$ of nearby nodes that have a routing entry for a process q . U itself is not considered a member of $\&surrogate(U, q)$. The set $\&surrogate(U, q)$ does not have to contain all the nodes in the system that have a routing entry for q . If U has a permanent routing entry for q then $\&surrogate(U, q)$ may be an empty set. However if U does not have a permanent routing entry for q then $\&surrogate(U, q)$ must be nonempty.

In brief the protocol for routing a message to a process is as follows :

- a. When node U must route a message to process q at the request of process p on U , the operating system first checks if a routing entry for q exists at node U . If so the message is either *delivered* locally or *routed* to the proper node depending on whether the routing entry points to U or not. However if no routing entry for the process is found at U then a *routing fault* occurs.
- b. When a *routing fault* for a message addressed to q occurs at node U , the message is buffered at U and a request is sent to one or more surrogates of U which have routing entries for q , i.e. nodes in $\&surrogate(U, q)$. When a routing entry for q is received in response, the message is routed to the node indicated by the routing entry.
- c. If a node V receives a message for a process q that is not resident on V , it *re-routes* or *forwards* the message. The procedure for *re-routing* or *forwarding* a message from V is identical to that for routing a message originating at that node.

Note that in some implementations of the BRM protocol, it is possible to receive several replies to a request for a particular routing entry from the surrogates. The first routing entry that is received is used to route the waiting message. If a routing entry is received and there are no pending messages then it is ignored.

The migration protocol for moving process q from node U to V is informally described below. Process migration is atomic with respect to message routing, i.e. a message cannot be delivered to a process while it is migrating.

- d. The temporary or permanent routing entry for q at node U is locked, and the execution of q is stopped at node U . Node U sends a MOVE system message, containing the process state (and program code if necessary) to node V .
- e. When node V receives the MOVE message, the migration-count of q is incremented, q is installed and restarted at V , and the routing entry for q at V is modified to indicate its current location and migration-count. If there is no routing entry for q at V then one is included in the routing table at V , marked as temporary, and assigned the new location and migration-count of q . A system message MOVE_CONFIRM is sent back to node U indicating that q is now located at V .
- f. Node U on receiving MOVE_CONFIRM, modifies the routing entry for q to indicate that q is at V , marks it with the incremented migration-count and unlocks the routing entry. Node U sends a routing update message to all surrogate nodes of U with routing entries for q .

Figure 2.3 illustrates the migration protocol. The choice of the surrogates and the method for the access and update of remote routing entries are left open in the BRM protocol. Obtaining a remote routing entry can be done in a number of ways such as bounded broadcasts, hashing etc., which we discuss in Section 5.1.

In the basic protocol temporary routing entries for a process are created at a node for two different reasons. If a process q is resident at a node U which does not normally have a routing entry for q , then a temporary routing entry for q is created at U to indicate that q is located at U . Another instance where a temporary routing entry is used is to facilitate the efficient atomic update of routing tables of surrogates. In order to reduce the synchronization delay during which a routing entry at U is unavailable, an atomic update can be achieved by creating a temporary routing entry at U , and sending asynchronous updates to the remote nodes that have to be updated. Creation of temporary routing entries can however lead to an increase in storage over time and hence these latter temporary entries must be deleted when the update is completed.

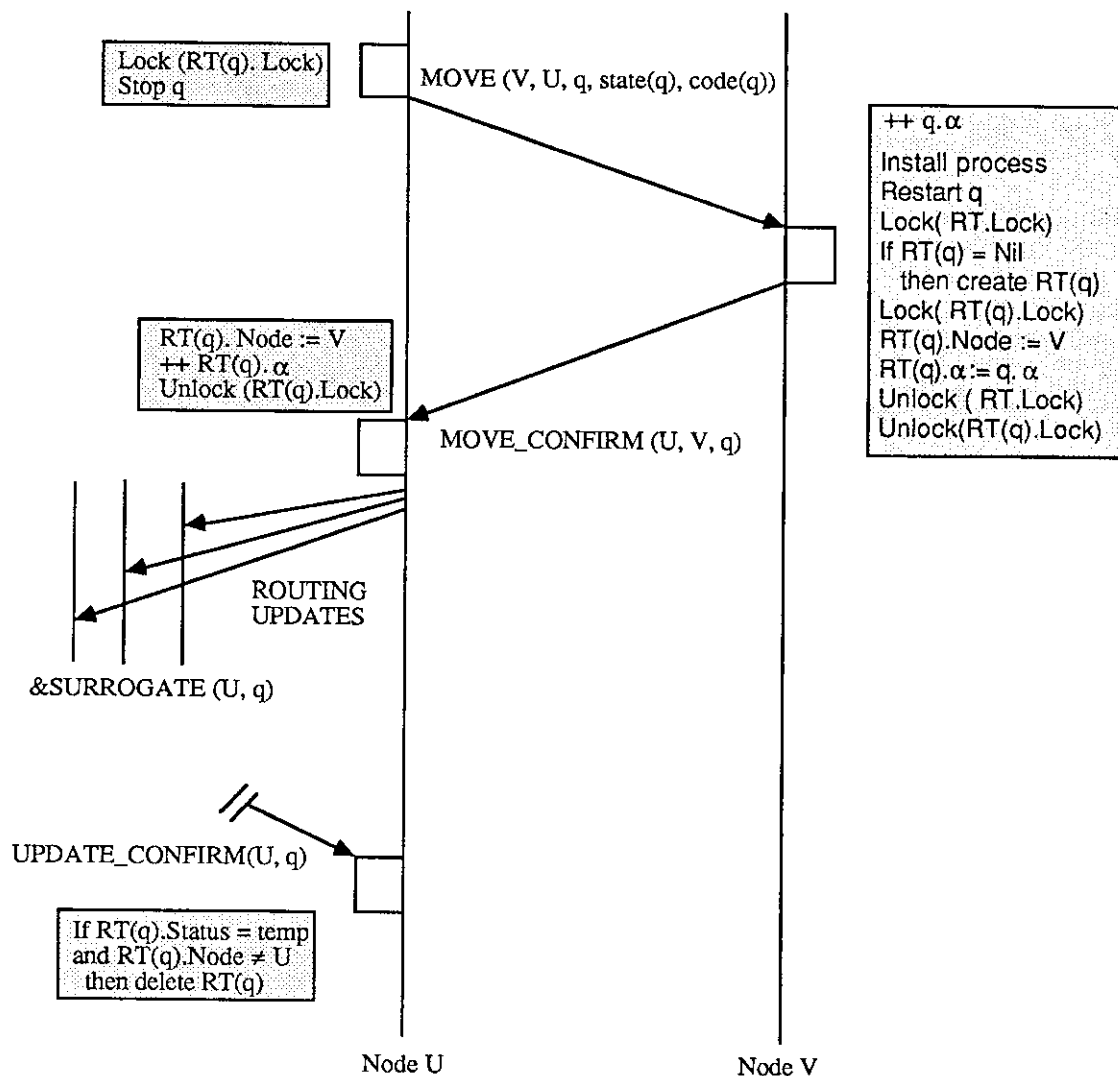


Figure 2.3 : Migration Protocol

For correctness, the routing table update procedure has to satisfy the following specifications :

1. When a routing table update is received at a node W indicating that process q is located at V with migration-count α , then the routing entry for q at W , if any, is updated only if α is greater than the migration-count of the routing entry for q .

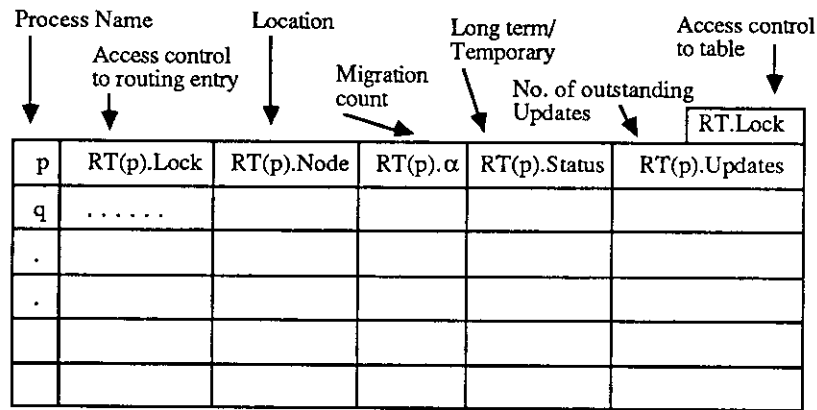


Figure 2.4 : Routing Table (RT)

2. The set of remote routing entries for q that can be queried when a routing-fault occurs at node U , must be a subset of those updated by U when q migrates away from U .
3. Any temporary entry for q at U can be deleted only after all members of $\&surrogate(U, q)$ have been updated.

If a process q migrates away from U , returns and migrates again, then it is possible that a new request for updating surrogate nodes of U with routing entries for q can be generated while a previous remote update for q is ongoing. The operating system therefore has to keep track of the number of requests for remote updates that are not yet complete. Only when all remote updates have completed can a temporary routing entry for q at node U be deleted.

Access to the tables is controlled by locks to ensure that possible concurrent access to a table at a node is synchronized. We design for the possibility that several instantiations of the BRM protocol may be running simultaneously at a node responding to different routing and migration activities. Associated with each table are two levels of locks, one lock to control insertion and deletion of entries from the table, and the other associated with each entry, to control read or write access to that entry.

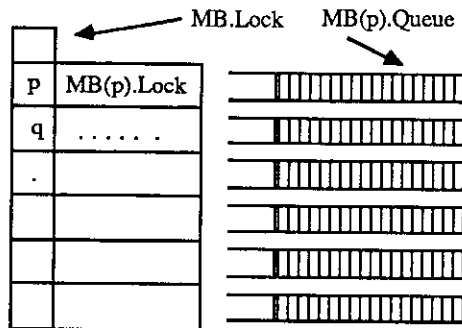


Figure 2.5 : Message Buffer Table (MB)

Figures 2.4 and 2.5 show the routing table (RT) and the message buffer table (MB). We refer to these tables at node U as RT_U and MB_U , but we drop the subscripts when the location of the table is clear from the context. Each entry in MB has a message queue to store messages. We will always assume that the buffer at node U is large enough to store those messages destined for a process q that arrive at U while q is migrating from U . It is a flow-control problem to guarantee that this assumption is true, but we do not deal with this issue here. **Enqueue** and **Dequeue** operations insert and remove messages from the queue. The **Empty** operation checks to see if a queue is empty. **Lock** (A.Lock) and **Unlock** (A.Lock) obtain and release A.Lock. A lock that is requested and is unavailable is retried and the algorithm does not proceed to the next step till the lock is obtained. **Insert&Lock** creates an entry that is initialized as locked. **Delete** unlocks an entry and then removes it.

"**Send**" is a kernel call to route a message to the node indicated in the message. "**When**" indicates the action to be taken on the receipt of a message from the kernel or a request from the higher levels of the operating system.

The algorithm for the routing and migration protocol follows. Each step of the algorithm is invoked asynchronously upon the arrival of a message or on receiving a command from the load management or application level.

BASIC ROUTING & MIGRATION PROTOCOL

Variable for each Process q

$q.\alpha$ migration-count for process q .
Incremented whenever q moves.

Variables at each Node U

RT routing table located at node U
containing routing entries.

RT.Lock lock associated with table RT that has
to be obtained before any routing
entries are inserted or deleted from
RT.

RT(q) routing entry for process q . By
convention we say it is equal to Nil if
there is no entry for q in the routing
table at U . RT(q) is deleted from RT by
the execution of Delete(q , RT). RT(q)
is created by executing Insert&Lock(q ,
RT, RT(q).Lock), that creates an entry
that is already locked.

RT(q).Node possibly out-of-date node address for
 q .

RT(q). α migration-count associated with the
above routing entry.

RT(q).Status indication of whether this routing
entry is permanent or temporary.
Permanent entries are marked
"permanent" and temporary entries are
marked "temporary".

RT(q).Updates number of updates to remote routing

entries whose completion has not yet been confirmed.

RT(q).Lock lock for routing entry that has to be obtained before the entry is read or written into.

MB message buffer table at node U consisting of a list of queues of messages destined for different processes and waiting for the resolution of routing-faults.

MB.Lock lock associated with MB that has to be obtained before MB(q) can be created or removed from MB.

MB(q) message buffer entry to indicate the existence of a queue of user messages whose destination is q. MB(q) = Nil if there are no messages whose destination is q. MB(q) is deleted from MB by the execution of Delete(q, MB). MB(q) is created by executing Insert&Lock(MB(q), MB(q).Lock), that creates an entry consisting of an empty queue that is already locked.

MB(q).Queue queue of user messages that are waiting for the resolution of routing-faults and whose destination is q.

MB(q).Lock lock for MB(q) that has to be obtained before a message can be enqueued or dequeued from MB(q).

Messages Sent & Received at each Node U

SMSG(U, q, text)	system message sent to node U containing user message text for process q.
MOVE(U, V, q, state, code)	system message for migration of q from node V to U, containing the process state and code.
MOVE_CONFIRM(V, U, q)	system message in response to a MOVE message; sent from node U to V indicating that process q is executing on U.
RE(U, q, W)	system message sent to U indicating that q is believed to be at W.
UPDATE_CONFIRM(U, q)	system message to confirm that updates for q sent to remote nodes have been received. It may be a message sent by U to itself.

Initialization Initially the migration-counts of all processes are zero and routing entries for each process are set up in the routing tables of some subset of nodes. The routing entries for a process consist of a node address which is the initial location of the process, and the entries are marked with a migration-count of zero and status permanent.

Procedures & Functions

Function ROUTE_LOCAL(q, text)

{ Attempts to deliver or route messages based on the local routing table. Delivers the message destined for q if q is located at node U. Else if there is a routing entry for process q at U then the message is forwarded to the node location given by the routing entry. }

begin

Lock(RT.Lock)

if RT(q) ≠ Nil **then begin**

{routing entry found}

Lock(RT(q).Lock)

Unlock(RT.Lock)

if RT(q).Node = U **then begin**

{locally delivered}

Place text in message queue for process q

result := delivered

end else begin

{re-routed}

send MSG(RT(q).Node, q, text)

result := re_routed

end

Unlock(RT(q).Lock)

end else begin

Unlock(RT.Lock)

result := failed

end

return(result)

end ROUTE_LOCAL

Procedure ROUTE_REMOTE(q, text)

{ Buffers the content of a message in the waiting table and requests for a remote routing entry for the destination process. The message with content "text" destined for q is buffered in the waiting table at node U and a remote routing entry for q is requested. }

begin

 Lock(MB.Lock)

if MB(q) = Nil **then begin**

 {no msg. buffer entry for q}

 Insert&Lock(q, MB, MB(q).Lock)

 Unlock(MB.Lock)

 {buffer message}

 Enqueue((q, text), MB(q).Queue)

 Unlock(MB(q).Lock)

 REQ_REMOTE_RE(q) {request remote routing entry}

end else begin

 {other msgs. for q waiting for routing entry}

 Lock(MB(q).Lock)

 Unlock(MB.Lock)

 {buffer message}

 Enqueue((q, text), MB(q).Queue)

 Unlock(MB(q).Lock)

end

end ROUTE_REMOTE

Procedure DELETE_RE(q)

{ Delete the routing entry for q from the local routing table if the entry is temporary, all updates to remote nodes have completed and the process is not resident at U. }

begin

 Lock(RT.Lock)

if RT(q) ≠ Nil **then begin**


```

Lock (RT (q) .Lock)
if RT (q) .Status = temporary
    {temporary routing entry}
and RT (q) .Node  $\neq$  U
    {process has not moved back}
and RT (q) .Updates = 0 then begin
    {remote updates complete }
    Delete (q, RT)
    {delete temp. routing entry}
end else begin
    Unlock (RT (q) .Lock)
end
end
Unlock (RT.Lock)
end DELETE_RE

```

Procedure REQ_REMOTE_RE (q)

{ This is a procedure to obtain a remotely located routing entry for q. In the Basic Routing & Migration Protocol the implementation of REQ_REMOTE_RE is left unspecified, as it can be done in a variety of ways. REQ_REMOTE_RE (q) executed at different nodes may return different values.

The set of remote routing entries that are queried are a subset of those updated.

In response to messages sent in this procedure, node U receives a message or messages RE(U, q, V) indicating that q is believed to be at V. }

Procedure UPDATE_REMOTE_RE (q, V, α)

{ Sends update messages to some subset of remote routing entries from U. The routing entries are updated to indicate

that process q is located at node V with migration-count α , but only if α is greater than the migration-count of the routing entry being updated. The implementation of UPDATE_REMOTE_RE is left unspecified. The subset contains at least those remote routing entries that can be queried from U .

If U has a temporary routing entry for q then in response to update messages sent in this procedure node U receives a UPDATE_CONFIRM (U, q) message indicating that all nodes in the subset have received the update. }

Algorithm at each Node U

```

<1> when a process on node  $U$  requests the sending
      of a message to process  $q$  begin
          {call from application layer}
      result := ROUTE_LOCAL ( $q, \text{text}$ )
      if result = failed then begin
          ROUTE_REMOTE ( $q, \text{text}$ )
      end
end

<2> when MSG( $U, q, \text{text}$ ) arrives begin
          {message from outside arrives}
      result := ROUTE_LOCAL ( $q, \text{text}$ )
      if result = failed then begin
          ROUTE_REMOTE ( $q, \text{text}$ )
      end
end

<3> when RE( $U, q, V$ ) arrives begin
      Lock(MB.Lock)
      if MB( $q$ )  $\neq$  Nil then begin
          {messages for  $q$  in buffer}
          Lock(MB( $q$ ).Lock)
  
```

```

        while not Empty (MB(q).Queue) do begin
            (q, text) := Dequeue(MB(q).Queue)
                {re-route messages in buffer}
            send MSG(V, q, text)
        end
        Delete(q, MB)
        Unlock(MB.Lock)
    end else begin
                {no messages for q in buffer}
        Unlock(MB.Lock)
    end
end
end

<4> when policy software
    requests migration of q from U to W begin
        {call from load management layer}
    Lock(RT (q).Lock)
        {block access to routing entry}
    Remove q from execution
    send MOVE(W, U, q, state, code)
end

<5> when MOVE(U, V, q, state, code) arrives begin
    q.alpha := q.alpha + 1
    Install process q with its state
    and code at node U
    Lock(RT.Lock)
    if RT(q) = Nil then begin
        {create temporary routing entry}
        Insert&Lock(q, RT, RT(q).Lock)
        Unlock(RT.Lock)
        RT(q).Status := temporary
        RT(q).Updates := 0
        RT(q).Node := U
        RT(q).alpha := q.alpha
        Unlock(RT (q).Lock)
    end else begin

```

```

                                {permanent routing entry exists}
Lock(RT (q).Lock)
Unlock(RT.Lock)
                                {update routing entry}
RT(q).Node := U
RT(q). $\alpha$  := q. $\alpha$ 
Unlock(RT (q).Lock)
end
send MOVE_CONFIRM(V, U, q)
end

<6> when MOVE_CONFIRM(U, W, q) arrives begin
                                {update local routing entry}
RT(q).Node := W
RT(q). $\alpha$  := RT(q). $\alpha$  + 1
                                {pending update to remote entries}
RT(q).Updates := RT(q).Updates + 1
 $\alpha$  := RT(q). $\alpha$ 
                                {unblock access to routing entry}
Unlock(RT(q).Lock)
                                {update remote routing entries}
UPDATE_REMOTE_RE(q, W,  $\alpha$ )
end

<7> when UPDATE_CONFIRM(U, q) arrives begin
                                {a remote update is complete}
Lock(RT.Lock)
if RT(q)  $\neq$  Nil then begin
    Lock(RT(q).Lock)
    Unlock(RT.Lock)
    RT(q).Updates := RT(q).Updates - 1
    Unlock(RT(q).Lock)
                                {attempt to delete entry}
    DELETE_RE(q)
end else begin
    Unlock(RT.Lock)
end
end

```

end

2.3 Correctness Argument

In this section we discuss the validation of the basic protocol by showing the absence of deadlocks and by proving that under appropriate conditions messages that are routed according to the protocol are eventually delivered to the destination process.

We first examine the possibility of local and distributed deadlock in the BRM protocol. Every step of the protocol has the following properties.

- (i) An attempt is made to obtain a lock for a table entry only when the entry exists and the entry is guarded from deletion while the protocol is waiting for its lock.
- (ii) The actions in each step other than waiting for locks take finite time.

Every step except <6> has the additional property.

- (iii) In each step table entries are locked before they are accessed.

Every step except <4> has the additional property.

- (iv) Before completion of a step all locks obtained in the step are released.

By assumption we have the following property.

- (v) Low level node-to-node message delivery takes finite time.

There are two levels of locks, one associated with an entry that has to be acquired before reading or writing to the entry, and the other associated with an entire table to control insertion, deletion and testing the existence of an entry. An entry lock and the corresponding table lock on the same node can be viewed as a *parent-child* lock pair. Obtaining the table lock does not prevent concurrent acquisition of an entry lock belonging to the table.

In the BRM protocol, the procedure for creating an entry is first to lock the table, test for the existence of the entry in the table, and if it doesn't exist then execute an Insert&Lock operation that creates an entry that is already locked. The table lock can now be released, the entry written into and the entry lock released. As the table lock has to be obtained to create an entry, two entries for a table cannot be created concurrently and hence there is no danger of creating duplicate entries.

An entry is deleted by first locking the table, then locking the entry and then executing a Delete operation that removes the entry. The table is then unlocked. As the entry has to be locked before it can be deleted hence in the BRM protocol it is not possible to delete an entry that is simultaneously being accessed.

To access an entry, only the entry lock has to be obtained. However, in the protocol whenever there is a possibility that an entry does not exist, the table is first locked and the existence of the entry in the table is checked. If the entry is in the table the entry is locked and the table lock is released. Hence in the BRM protocol there is no possibility of waiting forever in attempting to lock an entry that does not exist. Thus property <i> holds for all steps of the protocol.

A *causal path* is a sequence of dependent steps or events that can be on the same node or across nodes. Steps <4>, <5> and <6> form part of a causal path invoked when a process q migrates say from U to V , with steps <4> and <6> executing on node U and step <5> executing on V . The sequence of steps <4>, <5> and <6> take finite time because of properties (ii) and (v). The lock $RT_U(q).Lock$ obtained in step <4> is not released at the end of the step. Subsequently when step <6> is invoked at U the routing entry for q can be accessed because the lock is already obtained by an earlier step in the causal path. Finally in step <6> the lock $RT_U(q).Lock$ is released. We observe that even though properties (iii) and (iv) do not hold for all steps, for every causal path through the protocol entries are locked before they are accessed, and all locks obtained in the course of the path are released before the path terminates.

Theorem The Basic Routing and Migration Protocol is deadlock free.

Proof: When a causal path needs only one lock, it is obtained and released after finite time. For all causal paths in our protocol except the sequence of steps <4>, <5> and <6> invoked when a process migrates, at most two locks are held at any time. When two locks are held at a time, the two locks are always an entry lock and the corresponding

parent table lock and they are always obtained monotonically - first the parent table lock and then the entry lock. The concurrent execution of two paths of the protocol cannot result in a deadlock because the pair of locks have to be obtained in order. Moreover as only a corresponding parent-child lock pair can be held at the same time and not an entry-entry lock pair hence there is no possibility that three or more concurrent executions of the protocol can lead to a deadlock cycle. Thus there can be no deadlock when causal paths in the protocol other than the path related to migrating a process are executed concurrently on the same node or on different nodes.

Now consider the causal path that results when a process migrates. We assume that the Load Management level of the operating system is *well behaved* with the following properties:

- i. It does not request a process to be migrated from a node to itself.
- ii. It does not request a process to be moved from a node if it is not located at the node.

Because of assumption (i), steps <4> and <6>, and <5> that belong to a causal path cannot be invoked on the same node. Let us consider the migration of a process q from U to V . Steps <4> and <6> are invoked on U and <5> is invoked on V . A routing entry for q at U and a parent-child pair of the routing table and routing entry at V can be locked at the same time. Because the pair of locks at V is obtained in parent-child order hence a causal path with steps <4> , <5> and <6> cannot be involved in a deadlock with any other causal path. From assumption (ii) and the fact that every process has an unique name, simultaneous migrations from a node or between a pair of nodes has to involve locking routing entries for different processes which can not result in deadlock. Hence any two causal paths of the protocol can not be involved in a deadlock.

We have thus shown that the protocol is deadlock free. ■

Figure 2.6 illustrates the absence of deadlock in the scenario when two processes q and p migrate simultaneously from U to V and V to U respectively.

Before we proceed with the verification of the basic protocol we introduce a few definitions. A *process trajectory* at time t is defined as the sequence of node locations where the process has been resident up to time t . Since initialization, if a process q has been resident on nodes X_0, X_1, \dots and at time t is resident at X_η , then the process

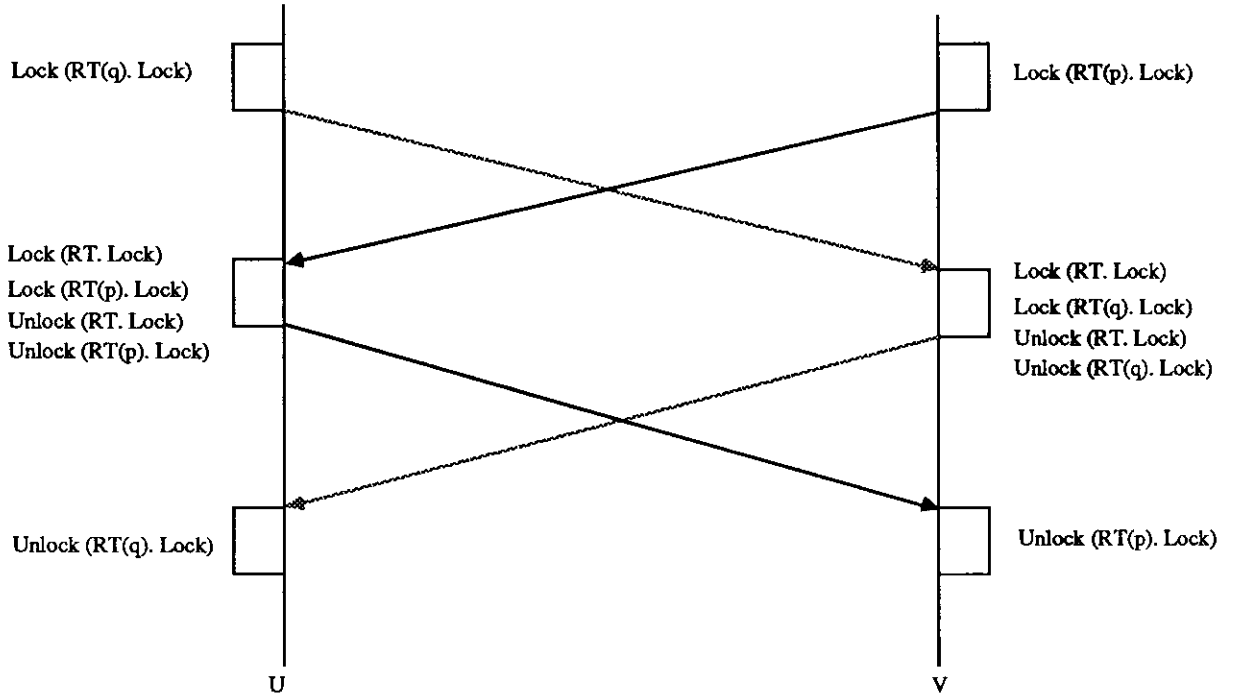


Figure 2.6 : Simultaneous process migration between U and V

trajectory is $trajectory(q, t) = X_0 X_1 X_2 \cdots X_\eta$. If the process q moves from X_η to $X_{\eta+1}$ at time t' , where $t' > t$ then $trajectory(q, t') = X_0 X_1 X_2 X_3 \cdots X_\eta X_{\eta+1}$.

Similarly let a *message trajectory* at time t for a message be the sequence of node locations to which the message has been routed up to time t . If a message M_q whose destination is process q , originates at node Y_0 and is initially routed to Y_1 , then re-routed to Y_2, Y_3, \dots and is last routed to Y_τ , then the message trajectory at time t is $trajectory(M_q, t) = Y_1 Y_2 Y_3 \cdots Y_\tau$. For technical reasons we leave the origin of the message Y_0 out of the message trajectory. When the message is re-routed from node Y_τ to $Y_{\tau+1}$ then the old message trajectory is appended with $Y_{\tau+1}$ to obtain the new message trajectory.

We define the functions *delegate* and *toward*. Let q be a process and U and X be nodes. The nondeterministic function *surrogate* was earlier defined as:

$$surrogate(U, q) = \text{Some node } X \text{ such that } X \neq U \text{ and } RT_X(q) \neq nil$$

The function *surrogate* need not return the same node value every time it is invoked. We define $\&surrogate(U, q)$ as the set of all possible nodes that can be returned by *surrogate* (U, q).

The node with the routing entry used to route messages to process q from node U is given by function *delegate*(U, q).

$$\begin{aligned}
 \textit{delegate}(U, q) = & \text{If } RT_U(q) \neq \textit{nil} \text{ then} \\
 & U \\
 & \text{else} \\
 & \textit{surrogate}(U, q)
 \end{aligned}$$

The function *delegate*(U, q) can never be *nil*. In accordance with procedure ROUTE_MSG, if there is an entry for q at U then it is chosen for routing messages for q at node U , else an entry for q located at *surrogate* (U, q) is used.

The function *toward*(U, q) computes the address of a node to which messages for a process q are routed (or re-routed) from node U , i.e. the node that *delegate*(U, q) believes is the location of q .

$$\textit{toward}(U, q) = RT_{\textit{delegate}(U, q)}(q).Node$$

In order to show the correctness of the Basic Routing and Migration Protocol we prove the following theorems:

First we prove that while routing entries are not necessarily up to date, they always point to a past or the present location of the process and have a migration-count equal to the number of the times the process had (has) migrated when it was (is) at that location.

Theorem 1 Let the process trajectory for process q at time t be *trajectory* (q, t) = $X_0 X_1 \cdots X_\gamma \cdots X_\eta$. For any node U at any time $RT_U(q)$ is unlocked, if $\gamma = RT_U(q).\alpha$ then $RT_U(q).Node = X_\gamma$.

Proof:

Initially, all routing entries for q point to the original location of q , i.e. X_0 , and all routing entries for q have migration-count zero. Also initially a process has a migration-count equal to zero. A routing entry can be modified only in step <6> and by the receipt of updates. Steps <4> to <6> and the remote update procedure are invoked whenever a process migrates to a new location. Every time a process migrates, its migration-count is incremented (step <5>). We observe in steps <6> and from the specifications of the update procedure that the routing entries that are modified when a process migrates to its $\gamma+1$ th location, point to the new location of the process and are assigned a migration-count γ . Thus any routing entry for a process always points to the present or past locations of the process, and a routing entry with migration-count γ points to the $\gamma+1$ th location of the process. ■

We now show that messages to a process from a node that is a previous location of the process, will be routed to a more recent location of the process.

Theorem 2 At any time after a process q has left its μ th location U , we have $RT_{delegate(U, q)}(q).\alpha \geq \mu$.

Proof: Let us assume that the process q with migration count $\mu-1$ resides on node U which is the μ th location of q . If process q moves from U to V , the routing entry for q at U if non null is updated to point to V and it is assigned the latest migration-count of q , i.e. μ . If the routing entry for q at U is null, then all nodes belonging to $\&surrogate(U, q)$, are updated (<6> and specification of remote update procedure) to point to V and assigned the migration-count μ . During the update of nodes $surrogate(U, q)$, the routing entries for q at $surrogate(U, q)$ are inaccessible from U because a temporary local routing entry for q is created at U and remains in place till all the surrogates have been updated (<6> & <7>). Thus at the moment process q completes its migration away from its μ th location U , a routing entry for q at all values of $delegate(U, q)$ will have a migration-count μ .

We now show that the migration-count field of a given routing entry on a given node never decreases. From the update procedure which describes the action taken by a node on receiving an update message, we observe that the routing entry for a process q is left unchanged unless the migration-count of the update is strictly greater than the migration-count of the routing entry. If the process q revisits U then the migration-count of its routing entry at U can only increase when q leaves U because the migration-count of a process is monotonically increasing on every move. Therefore the routing entry for q at any $delegate(U, q)$ will have migration-count $\geq \mu$, where U is the μ th location of q . ■

In Theorem 3 we prove that if messages for a process are routed to a new node from another node then either the process is resident on the new node or else the new node has a more recent routing entry for the process.

Theorem 3 At all times, either $RT_{toward(U, q)}(q).Node = toward(U, q)$ or $RT_{delegate(toward(U, q), q)}(q).\alpha > RT_{delegate(U, q)}(q).\alpha$.

Proof: Let $\gamma = RT_{delegate(U, q)}(q).\alpha$. From Theorem 1 we know that any routing entry for process q with migration-count γ points to the $\gamma+1$ th node in the process trajectory of q . Therefore $toward(U, q)$ is the $\gamma+1$ th node in the process trajectory of q . Suppose q is not at node $toward(U, q)$, i.e. $RT_{toward(U, q)}(q).Node \neq toward(U, q)$. From Theorem 2 we know that if node $toward(U, q)$ is the $\gamma+1$ th location of q , then $RT_{delegate(toward(U, q), q)}(q).\alpha \geq \gamma+1$. Therefore $RT_{delegate(toward(U, q), q)}(q).\alpha > RT_{delegate(U, q)}(q).\alpha$. ■

Finally, we show that a message to a process follows the trajectory taken by the process.

Theorem 4 The trajectory of a message is a subsequence (not necessarily contiguous) of the trajectory of its target process.

Proof: From the definition of the function *delegate* we know that either a node U has a routing entry for q , or else $surrogate(U, q)$ has a routing entry for q . From procedure ROUTE_MSG we observe that messages for a process q are routed from any node U to $toward(U, q)$. If a message is routed to a node $toward(U, q)$ where the process is not resident, then the message is re-routed to $toward(toward(U, q), q)$. Suppose the message is routed from say Y_0 to Y_1 , from Y_1 to Y_2 and so on, and arrives at Y_τ at time t . At time t , the message trajectory is $trajectory(M_q, t) = Y_1 Y_2 Y_3 \cdots Y_\tau$.

Let the process trajectory be $trajectory(q, t) = X_0 X_1 \cdots X_\gamma \cdots X_\eta$. From Theorem 1 we know that any routing entry for a process points to a node on the process trajectory. Therefore at any time t the j th node in $trajectory(q, t)$ maps to the μ th node of $trajectory(M_q, t)$ such that $X_{j+1} = Y_\mu$. From Theorem 3 we know that this mapping is strictly monotonic i.e. if $X_{j+1} = Y_\mu$ and $X_{\gamma+1} = Y_\gamma$ and if $j > \gamma$ then we have $\mu > \gamma$.

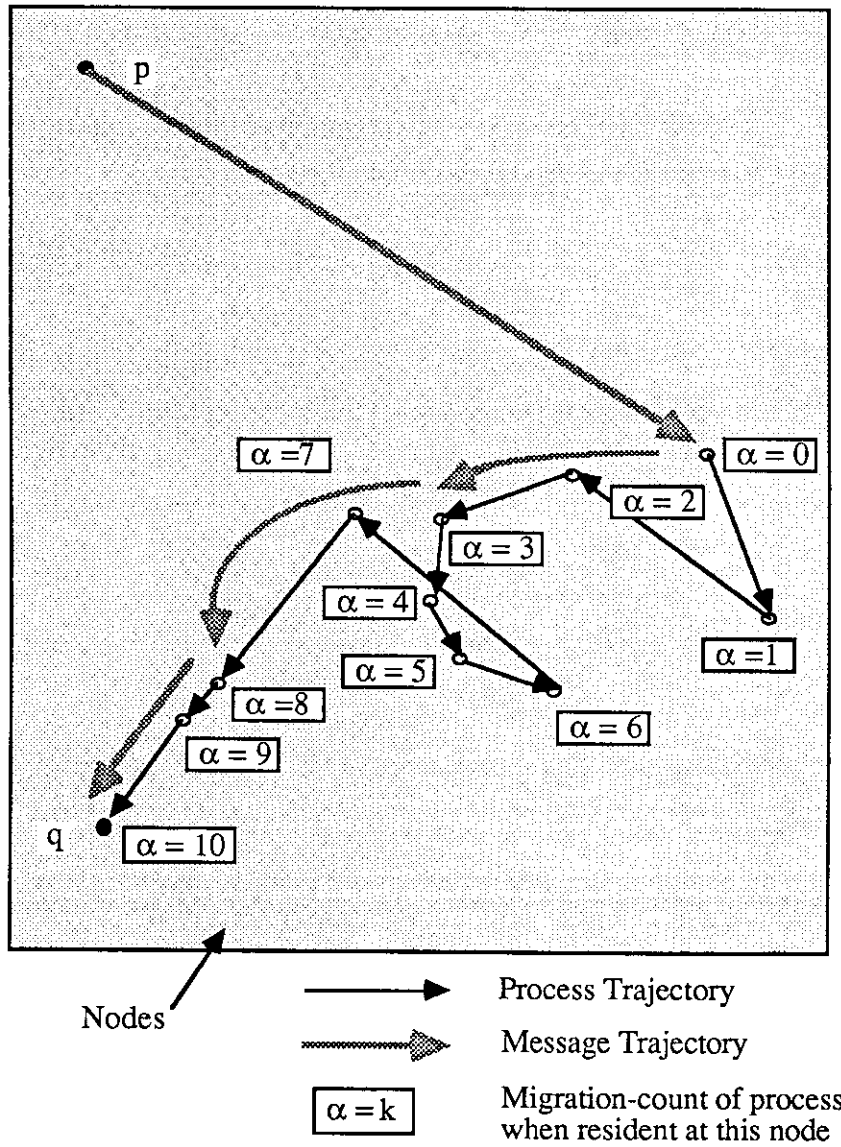


Figure 2.7 : Route taken by a message from process p to q

Therefore a message for a process follows a trajectory that is a subsequence of the trajectory taken by its target in such a way that every time a message is re-routed it is sent to a more recent location of the process. ■

We now show that under certain conditions for the velocity of messages and processes the BRM protocol does correctly deliver all messages to their target processes. The velocity of a message takes into account the time that a message is waiting for the resolution of routing faults.

Proposition If a message always travels a shortest physical distance route between stops along its trajectory, and if its average velocity is strictly greater than the average velocity of its target process, then the message will be delivered to its target.

Proof: By Theorem 4 the trajectory of the message is a subsequence of the trajectory of the target process. By the triangle inequality the message travels a physical distance between any two stops along its trajectory that is less than or equal to the physical distance the process traveled between the same two stops. Hence, if the average velocity of a message is higher, and the average distance traveled between points is equal or shorter, the message will *catch up with* the process. ■

Figure 2.7 illustrates a message from process p following the trajectory of process q . The migration-count of q is originally zero and every time q migrates, its migration-count is incremented by one. The message is delivered to q when the migration-count of q is ten.

2.4 Extended Protocol

The basic protocol is designed to address the synchronization and correctness issues associated with the migration of processes. It is meant to be a basic building block to which extensions can be added to improve performance.

The basic protocol by itself is impractical because in the worst case a message for a process will have to retrace the entire trajectory of the process since its creation. This is because when a process moves from a source node to a destination node, only the source node and its surrogates are informed of the new process location. If we ignore the movement of a process while a message is enroute to it then in the worst case a message has to be re-routed $(N - 1)$ times before it is delivered, where N is the number of nodes in the system. On the other hand the update bandwidth due to the BRM protocol is zero, if we ignore the updates sent to surrogates.

A simple extension to the BRM protocol that we call the *Complete Broadcast on Migration* (CBM) protocol and which is used by several systems is as follows: When process q moves from node U to V , an asynchronous UPDATE message is sent to *all* nodes in the system that have a routing entry for q . Any node that receives an UPDATE message modifies its own routing entry for q if the migration-count of the entry is less than the migration-count of the routing update.

In the Complete Broadcast on Migration protocol, the number of times a message is forwarded is zero, if we ignore the movement of a process while a message is enroute to it and if we assume that the time for updates to propagate is negligible when compared to the time between process migration. However the number of update messages sent every time a process moves is $O(N)$ update_msg-hops, where N is the number of nodes in the system. If we assume a process migrates at the rate of ν moves/sec, and there are P processes in the system then the total update bandwidth is $\nu \times P \times N$ update_msg-hops/sec. If the number of processes per node and the migration rate of a process are constants then the total bandwidth is $O(N^2)$.

In the BRM protocol while the update bandwidth is the minimum possible the messages latency is unacceptable, particularly when processes are fairly mobile, because the latency increases with the time for which the process has been in existence. On the other hand the CBM protocol demonstrates the lowest possible message latency, with the message trajectory stretch ratio close to one. However the update bandwidth is high if the number of nodes and the migration rate of processes is high. Thus the two protocols provide lower and upper bounds for update bandwidth and message latency - the message latency in the CBM protocol serves as a lower bound for message latency and the bandwidth to update routing tables in CBM serves as an upper bound for the update bandwidth for any protocol to route messages to moving processes. In this work we propose extensions to the BRM protocol with both reasonable update bandwidth and message latency, by a tradeoff of increased bandwidth to update routing tables for improved message latency.

CHAPTER III

Routing Caches in Distributed Systems

3.1 Locality in Distributed Systems

Communication between processes in distributed systems is not random but tends to follow certain patterns. When the probability that a process communicates with other processes in the system is not equal, and instead a process tends to favor a subset of the set of processes to communicate with, then the communication between distributed cooperating processes is said to have the *locality* property. Locality in distributed systems consists of two components - *temporal* and *spatial* locality.

Temporal locality in a distributed system is inherent to the application and has the following properties:

- i. A process that has sent a message to another process in the recent past is likely to send messages to it in the near future.
- ii. The set of processes with which a process communicates is small and changes slowly.

Temporal locality results from the way applications are structured by the user.

Spatial locality on the other hand results due to the initial placement of processes and due to the migration of processes due to dynamic load management. Its manifestation is that a process is more likely to communicate with another process on a node close to it than a process located at a distant node.

We can define the *logical distance* between two processes to be indicative of the degree of communication between the two processes in the recent past. If the frequency of communication between two processes is high then the logical distance between them is small. If two processes have a small logical distance between them at a particular time, then by the principle of temporal locality they are likely to have a small logical distance between them in the near future. When the communication pattern between processes is changing dynamically one of the objectives of dynamic load management is to translate temporal locality in an application to spatial locality in the system. Dynamic mapping which is a component of dynamic load management maps processes that have a small *logical distance* between them to nodes that have a small *real distance* between them, i.e. nodes that are geographically close to each other. The effectiveness of the transformation of the logical distances between processes to real distances between them is constrained by CPU and memory bounds. Thus spatial locality is not an inherent property of the distributed application but results due to the mapping of processes to nodes.

There is a locality principle that we implicitly assume in developing our routing schemes: when a process migrates it moves a short real distance from its previous location. The principle of forwarding a message from an older location of the destination process to a more recent location of the destination process is an efficient strategy only if the message on being forwarded gets closer to the destination process.

There is an obvious analogy between locality of communication in distributed systems and the locality of memory accesses in memory management systems. Just as virtual memory takes advantage of locality to improve the memory access time, similarly we introduce *routing caches* to improve the time to access routing entries. In this chapter we study the addition of caches to the Basic Routing and Migration (BRM) protocol to take advantage of *temporal localities* in communication between distributed cooperating processes. In the following chapter we study schemes for the exploitation of *spatial localities* in the communication between distributed cooperating processes.

3.2 Adaptive Routing with Caches

3.2.1 Introduction

Caches containing routing entries track temporal localities in the communication between processes. When a source process has to send a message to a destination process, the source process has to first obtain a routing entry for the destination process. Analogous to the memory hierarchy organized to optimize the cost of memory references given the constraints on the size and speed of memory, we propose a *routing table hierarchy* to optimize the cost for a process to obtain routing entries. The routing table hierarchy, shown in Figure 3.1, consists of three stages. First the process checks a local cache for the particular routing entry. If the local cache does not have the routing entry then the local routing table is checked, and if the local routing table does not have the entry then remote routing tables are checked till the entry is found. As we observed earlier, in large distributed systems it is not possible because of size constraints for local routing tables to contain routing entries for all processes. Therefore when a *routing entry request miss* occurs at the local routing table then the operating system has to search for the routing entry in the routing tables of remote nodes using time consuming and bandwidth intensive mechanisms described in Chapter V. If the required routing entry is not found locally, then we say a *routing fault* has occurred. The cost of obtaining a routing entry from a remote node is orders of magnitude greater than the cost of obtaining a local entry. Access of remote routing entries in the routing hierarchy is thus analogous to disk access in memory hierarchy.

When a process is created, routing entries for that process are distributed to select routing tables in the system. Subsequently new routing entries for that process are not added to any routing tables except for the purpose of synchronization when process migration occurs, in which case they are deleted once their synchronization purpose has been served. Thus entries in routing tables are of a permanent nature. The contents of local caches however dynamically adapt to the pattern of communication between processes. Routing entries that are repeatedly used are retained in the cache while those that are no longer used are dropped from the cache. The purpose of routing caches is to reduce requests for remote routing entries for processes.

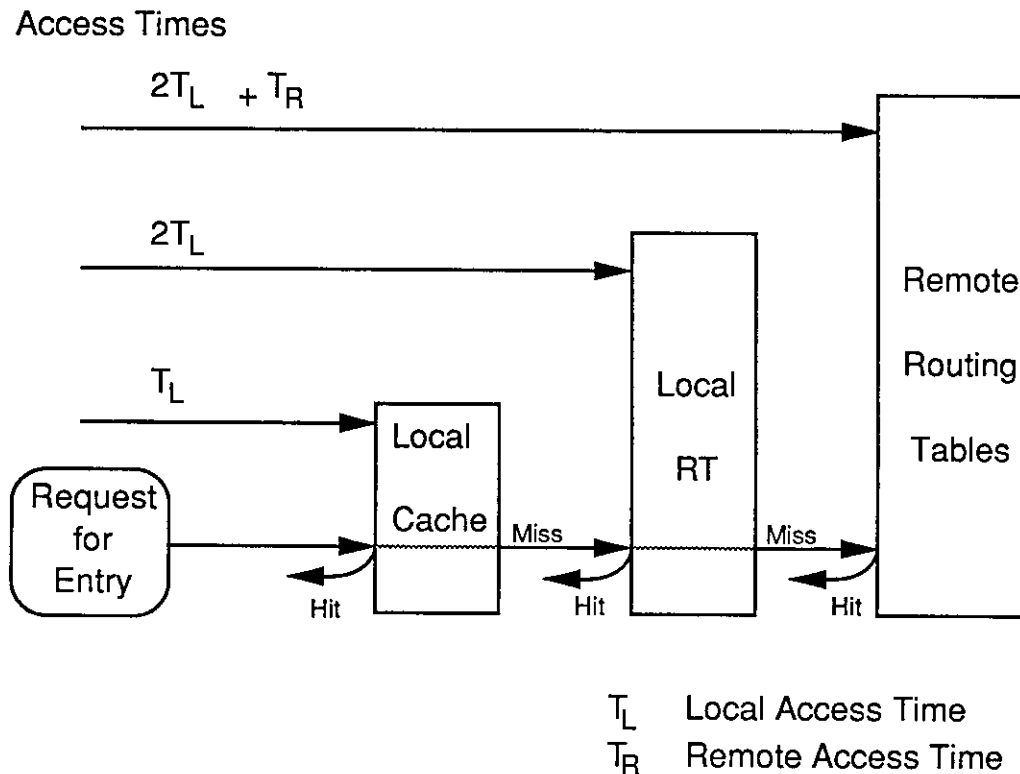


Figure 3.1 : Routing Table Hierarchy

Another purpose of having caches is to keep local entries updated. When a message reaches its destination process the sending process is sent a cache update consisting of the latest location and migration-count of the destination process. The entries in the routing cache of the sending process are kept updated by cache update messages. By using more up-to-date entries, the number of times a message is forwarded and consequently the delay for the message to reach the destination is reduced.

Unlike memory caches used in CPU architectures the time to access a local routing cache is not significantly different from the time to access the local routing table. The difference between the local cache and routing table and the reason for accessing the local cache first is because the cache adapts dynamically to the pattern of communication while the local routing table does not.

We can associate a cache with each process on a node or have one common cache per node [Terry85]. The advantage of *Process Caches* is that when a process migrates the Process Cache can be carried and installed at the new location of the process thus avoiding a larger frequency of routing faults soon after migration. *Node Caches* on the other hand can be shared by all the processes resident on the node, thus avoiding duplication of entries and permitting more effective sharing of memory space. We first discuss schemes that incorporate Process Caches in the BRM protocol and later discuss the BRM protocol with Node Caches.

3.2.2 Cache Structure

Process Name	Location	Migration-count
--------------	----------	-----------------

Figure 3.2 : Cache Entry

In the BRM protocol we add fixed size caches in addition to the routing tables at each node. A cache entry consists of a process name, its node address and the migration-count of the process. The structure of a cache entry is shown in Figure 3.2. Cache entries are accessed, added or modified by giving commands to the *cache manager*. Caches are of fixed size and when a new entry is added to an already full cache then a suitable entry is chosen and dropped based on a *replacement policy* run by the cache manager. Least Recently Used (LRU) or other schemes can be used as a replacement policy. Associated with each cache entry is a *reference field* maintained by the cache manager to indicate how recently an entry was accessed. The cache manager makes decisions on which entry to replace when the cache is full, based on the value of the reference fields of the different entries.

The operating system interacts with the cache through the cache manager. The commands to the cache manager are given in Figure 3.3.

The command **Lookup** checks the cache for an entry for the given process and if one is found, the location and the migration-count of the process are returned. If the entry is not found then (*Nil, Nil*) is returned. If the parameter *mark* is specified in the *Lookup* command then the reference field of the accessed cache entry is updated, else the reference field is not modified. The **Refresh** command replaces the cache entry for a

particular process, if one exists, with the specified location and migration-count for the process. The *mark* parameter serves the same purpose as before. **Remove** command removes an entry for the specified process from the cache if it is in the cache. The **Add** command adds a new entry to the cache and returns the entry that is being replaced. If no entry is replaced then *(Nil, Nil, Nil)* is returned. The *Enumerate* command returns a list of all the cache entries.

```

(Location, Migration-count) := Lookup (Cache name, Process name, mark/no_mark)

Refresh (Cache name, (Process name, Location, Migration-count), mark/no_mark)

Remove (Cache name, Process name)

(Process name, Location, Migration-count) :=
    Add (Cache name, (Process name, Location, Migration-count))

((Process name, Location, Migration-count), (. . ), .. ) := Enumerate (cache name)

```

Figure 3.3 : Commands to Cache Manager

We first discuss schemes that incorporate Process Caches in the BRM protocol and later discuss the BRM protocol with Node Caches.

3.2.3 Process Caches

We associate two caches with each process p - a *Send Cache CS(p)* and a *Receive Cache CR(p)*. The Send Cache stores the location and associated migration-count of processes to which process p sent messages recently. The Receive Cache on the other hand stores the location and the associated migration-count of processes that sent messages to p recently. When process p sends messages to q , an entry for q is kept in the Send Cache $CS(p)$ to avoid expensive routing faults. The purpose of the Receive Cache $CR(p)$ of process p is so that when p migrates to a new location the operating system can notify all processes that recently sent messages to p , of the new location of p . Thus when

temporal localities are present, Send Caches amortize the time to locate a process when routing tables are incomplete, while Receive Caches reduce the number of times a message is forwarded before it is delivered to a moving process. Both the Send and Receive Caches of a process are carried along with a process when the process is migrated and installed at the new location, avoiding a *cold start* of the process with empty caches.

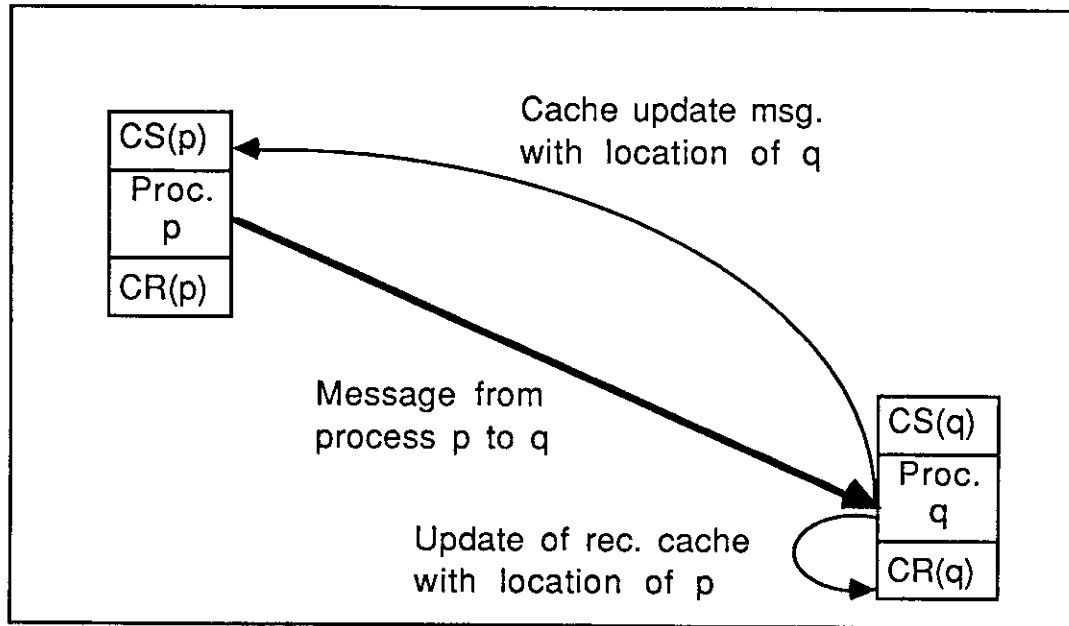


Figure 3.4 : Update of Caches on Receipt of Message

We propose two alternate schemes for the addition of caches associated with processes to the BRM protocol. The Process Caching Scheme with placement on Routing Fault, *PCS (RF)*, is informally described below. Note that when a cache or routing entry is updated, any modification to the entry is made only if the update has a higher migration-count.

- i. When process p located at node U requests the sending of a user message to process q , the operating system first checks the Send Cache $CS(p)$ for an entry for q . If no entry is found then the local routing table at node U is checked. However if there is no entry for q in the local routing table also, then a routing fault occurs and a request for a routing entry for q is sent to a remote node.

- ii. On obtaining an entry for q either from the local cache, local routing table or a remote node, the user message from p is routed to the node location indicated by the entry. If the entry is obtained from the remote routing tables it is added to the Send Cache $CS(p)$ at U .
- iii. When process q receives a user message from process p , the entry for p in $CR(q)$ if any is updated or else a new one is added to the Receive cache $CR(q)$ indicating the latest location and migration-count of p . Process q sends a cache update message containing the latest location and migration-count of q to process p . This is illustrated in Figure 3.4.
- iv. When a process p at node U receives a cache update message for an entry for q , the operating system checks the local Send Cache $CS(p)$ and the local routing table for an entry for q . If an entry for q is found in either the cache or the routing table it is updated.
- v. When node U receives a routing update message for an entry for q , the operating system checks if there is a entry for q in the local routing table and if so updates it.
- vi. When process q migrates the caches $CS(q)$ and $CR(q)$ are removed from the old location of q and installed at the new location of q . All entries in $CS(q)$ that are common to the entries at the routing table at the new location are deleted from $CS(q)$. All processes for which there is an entry in $CR(q)$ are sent a cache update message containing the new location and migration-count of q .
- vii. A node that receives a cache update message routes the message similarly to user messages.

When a cache update message is received by a process the local routing table has to be updated in addition to the Send Cache of the process because entries that belong to the routing table are not kept in the cache.

Figure 3.5 illustrates the action taken when process q migrates. The Receive Cache $CR(q)$ associated with process q has entries for processes p , r and s due to receipt of messages from these processes. On migration, process q carries its Send and Receive Cache to the new location and sends cache update messages to processes p , q and r .

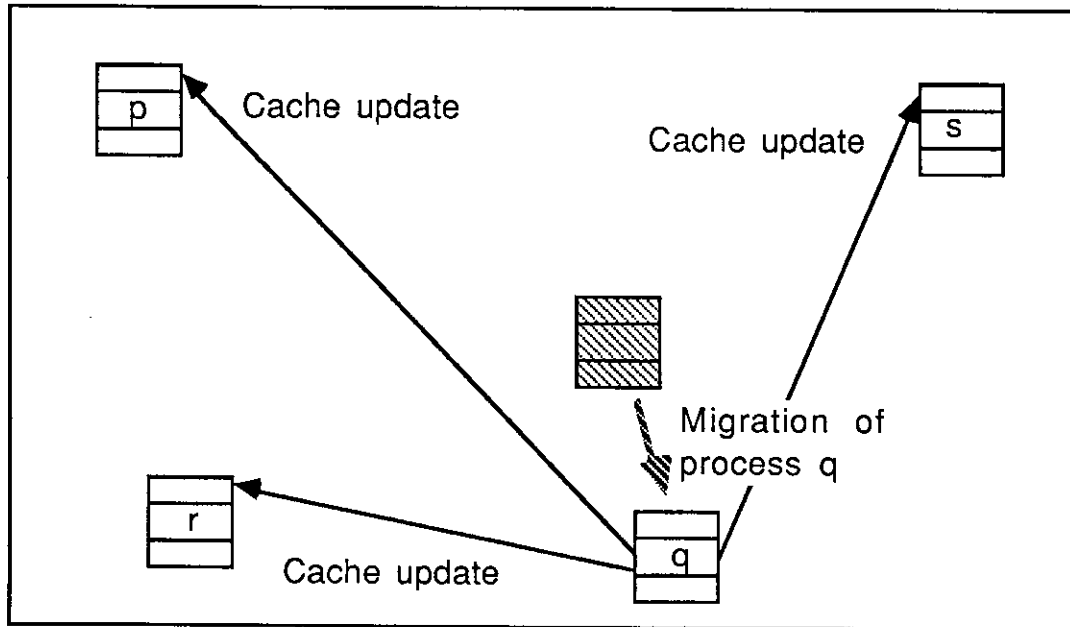
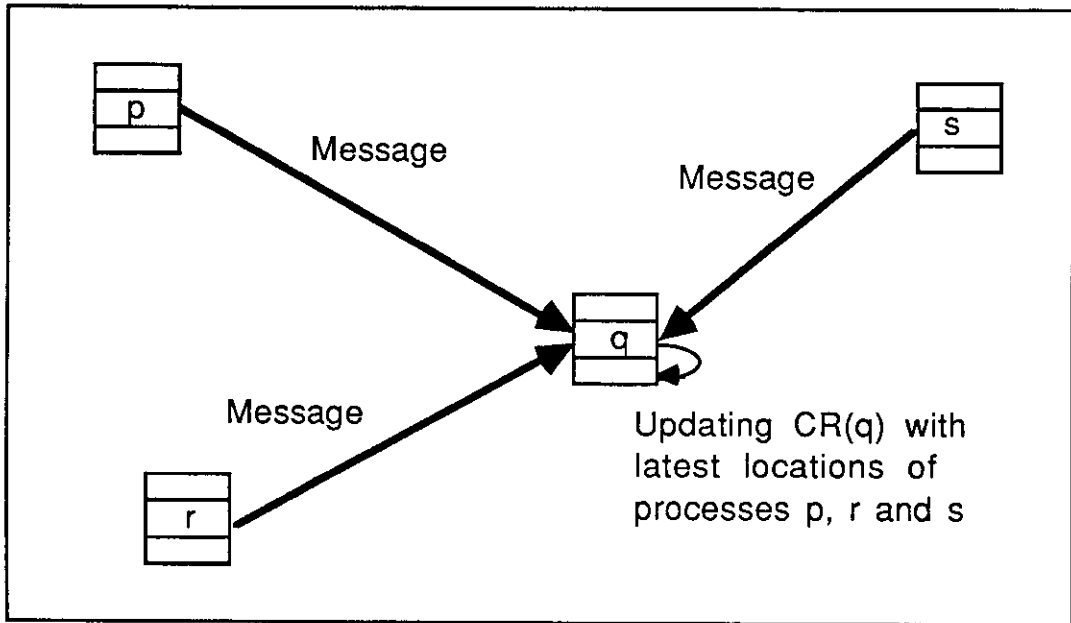


Figure 3.5 : Update of Caches on Process Migration

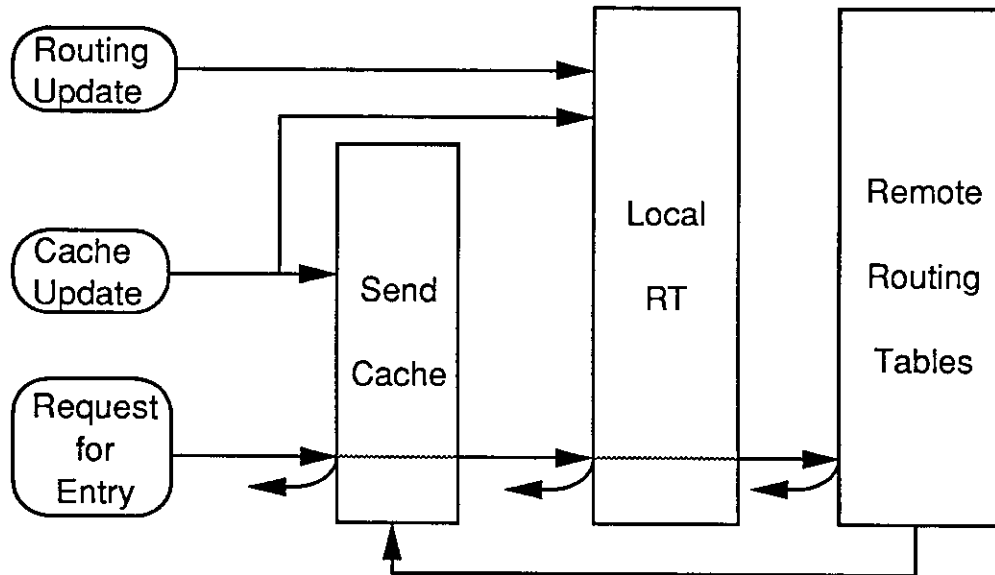


Figure 3.6 : Routing Hierarchy for Process Caching Scheme (RF)

Receive Caches are updated by user messages while Send Caches are updated by cache update messages in response to user messages. The updating of caches results in negligible communication bandwidth overhead. Cache update messages need be sent only if the user message has been forwarded before its delivery to the destination process, i.e. if the cache entry at the source process is known to be out-of-date. Cache update messages can be piggybacked onto other user messages or acknowledgements to user messages.

The routing table hierarchy for Process Caching Scheme (RF) is illustrated in Figure 3.6.

At each node there is a set of Send and Receive Caches for all processes resident on that node. The set of Send Caches at a node are illustrated in Figure 3.7. Access to the set of Send Caches *CS* at a node is controlled by a lock. A new Send Cache for a process can be added to the set of Send Caches at a node by executing **Insert&Lock** which creates a Send Cache for the process that is already locked. **Delete** unlocks the Send Cache of a process and removes it from the set of Send Caches on the node. The implementation of the set of Receive Caches is similar to the set of Send Caches.

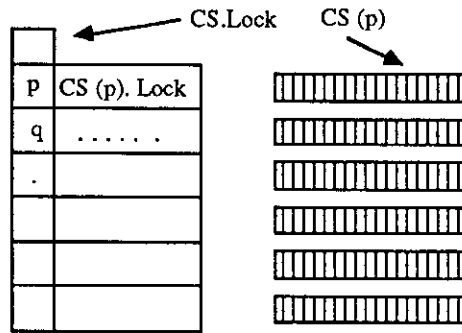


Figure 3.7 : Send Caches (CS) at a Node

The alternative mechanism - the Process Caching Scheme with placement on Cache Miss *PCS (CM)* differs from the previous mechanism in that an entry is added to the cache when there is a *cache miss* instead of a routing fault. Thus an entry is added to the cache if a request for that entry is made and the entry is not found in the cache.

The Process Caching Scheme with placement on Cache Miss *PCS (CM)* is informally described below. The main difference between the two schemes is in the policy of addition of entries to the cache that can result in duplication of entries in the cache and routing table. A consequence of the duplication of entries is that in *PCS(CM)* if an entry is dropped from the Send Cache because the Send Cache is full, the local routing table has to be updated prior to dropping the cache entry. There are also differences in the way cache and routing updates are handled. Note as before that when any cache or routing entry is updated, any modification to the entry is made only if the update has a higher migration-count.

- i. When process p located at node U requests the sending of a user message to process q , the operating system first checks the Send Cache $CS(p)$ for an entry for q . If no entry is found then a *cache miss* occurs and the local routing table at node U is checked. However if there is no entry for q in the local routing table also, then a request for a routing entry for q is sent to a remote node.
- ii. On obtaining an entry for q either from the local cache, local routing table or a remote node, the user message from p is routed to the node location indicated by the entry. If the entry is obtained either from the local or remote routing tables it

is added to the Send Cache $CS(p)$ at U .

- iii. When process q receives a user message from process p , the entry for p in $CR(q)$ if any is updated or else a new one is added to the Receive cache $CR(q)$ indicating the latest location and migration-count of p . Process q sends a cache update message containing the latest location and migration-count of q to process p .
- iv. When a process p at node U receives a cache update message for an entry for q , the operating system checks if there is a entry for q in the local Send Cache $CS(p)$. If there is an entry for q in $CS(p)$ then it is updated, else the routing table at U is checked for an entry for q if any and updated.
- v. When node U receives a routing update message for an entry for q , the operating system checks each of the Send Caches belonging to the processes resident at U and the routing table on node U for an entry for q and if an entry for q is found it is updated.
- vi. When process q migrates, entries in the routing table at the old location of q that are in common with entries in $CS(q)$ are updated with the cache entries. The caches $CS(q)$ and $CR(q)$ are removed from the old location of q and installed at the new location of q . All processes for which there is an entry in $CR(q)$ are sent a cache update message containing the new location and migration-count of q .
- vii. A node that receives a cache update message routes the message similarly to user messages.
- viii. When a cache entry for q has to be discarded from the Send Cache of process p because the Send Cache is full, then the routing entry for q , if any in the local routing table is updated with the location and migration-count for q in the cache entry.

The routing table hierarchy for the Process Caching Scheme (CM) is illustrated in Figure 3.8. The reason that when a routing update is received at a node all the caches at that node have also to be updated is because an entry for a particular process could be present in each of the Send Caches at the node in addition to the routing table.

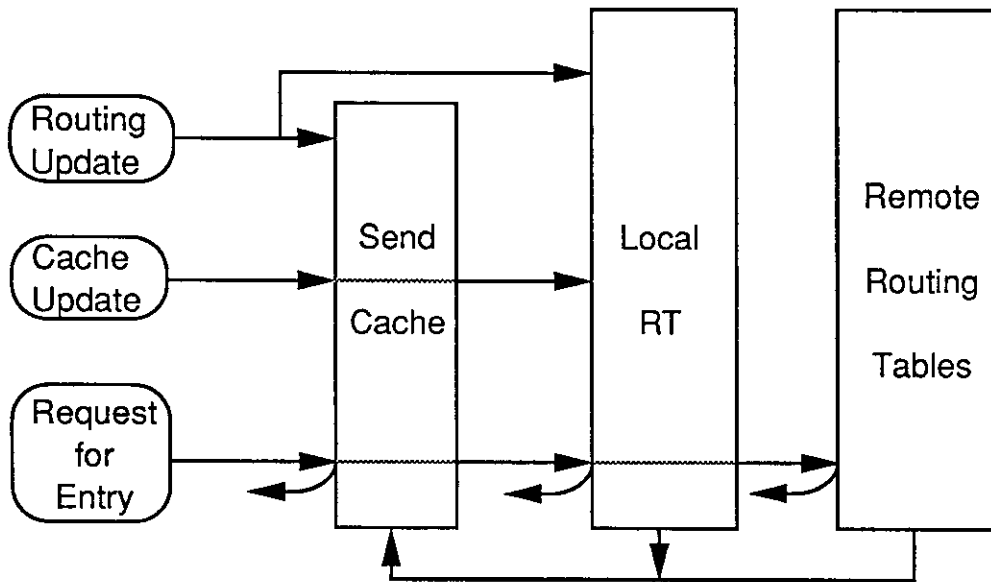


Figure 3.8 : Routing Hierarchy for Process Caching Scheme (CM)

The difference between the two schemes for adding processes to the BRM protocol is that in PCS (RF) entries are added to the cache only when there is a routing fault in accessing an entry, whereas in PCS (CM) entries are added to the cache whenever any entry, local or remote, is accessed. Thus in PCS (CM) there is no change in the rate of routing faults when a process migrates, because the Send cache contains entries for *all* processes to which the process has sent messages recently. However in PCS (RF) there will be an increase in the rate of routing faults when a process migrates because the Send Cache does not have entries for processes to which messages were sent recently if these routing entries were in the routing table at the previous location. In PCS (CM) many entries in the Send Cache may be redundant because of duplication of entries in the local routing table. Thus, with fixed size process caches the superiority of one scheme over another depends on the pattern of communication and the size of the routing tables.

In PCS (RF), memory space is effectively utilized by avoiding duplication of entries in a process cache and the local routing table. However in both PCS (RF) and PCS (CM) there is no sharing of process caches between different processes located on the same node. The individual Send Caches of a process resident at a node and the local routing table they share are shown in Figure 3.9. We now introduce a scheme that uses

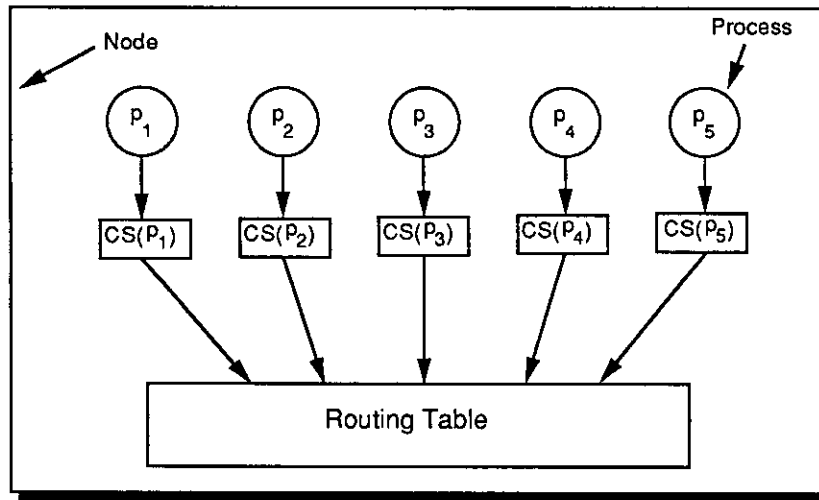


Figure 3.9 : Request of Entries by Process in Process Caching Scheme

node caches that are shared by all processes resident on the node thus avoiding duplication of entries among different process caches resident on the same node.

3.2.4 Node Caches

Node caches like local routing tables are associated with individual nodes. An entry is added to the local node cache when a process resident on the node has to send a message that results in a routing fault. The advantage of node caches is that there is no duplication of entries at a node and the available memory space is effectively utilized. The sharing of a Node Cache by processes resident on a node is shown in Figure 3.10. However there are also many disadvantages. It is difficult to maintain fairness in the use of the node cache by processes resident on a node, particularly when some of the processes are sending out many more messages to different processes than others. Moreover, when a process migrates to a new location it will have no cache entries at the node cache in the new location and hence there will be a large number of routing faults after the move.

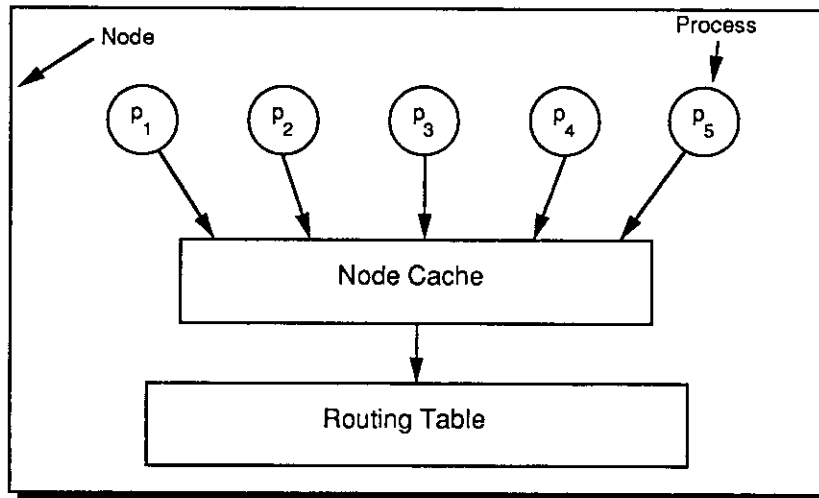


Figure 3.10 : Request of Entries by Process in Node Caching Scheme

The Node Caching Scheme *NCS* is informally described below. As usual when any cache or routing entry is updated, a modification to the entry is made only if the update has a higher migration-count.

- i. When process p located at node U requests the sending of a user message to process q , the operating system first checks the Node Cache CN for an entry for q . If no entry is found then the local routing table at node U is checked. However if there is no entry for q in the local routing table also, then a routing fault occurs and a request for a routing entry for q is sent to a remote node.
- ii. On obtaining an entry for q either from the local node cache, local routing table or a remote node, the user message from p is routed to the node location indicated by the entry. If the entry is obtained from the remote routing tables it is added to the Node Cache NC at U .
- iii. When process q receives a user message from process p , q sends a cache update message containing the latest location and migration-count of q to process p .
- iv. When a process p at node U receives a cache update message for an entry for q , the operating system checks the local Node Cache NC and the local routing table for an entry for q . If an entry for q is found in either the cache or the routing table

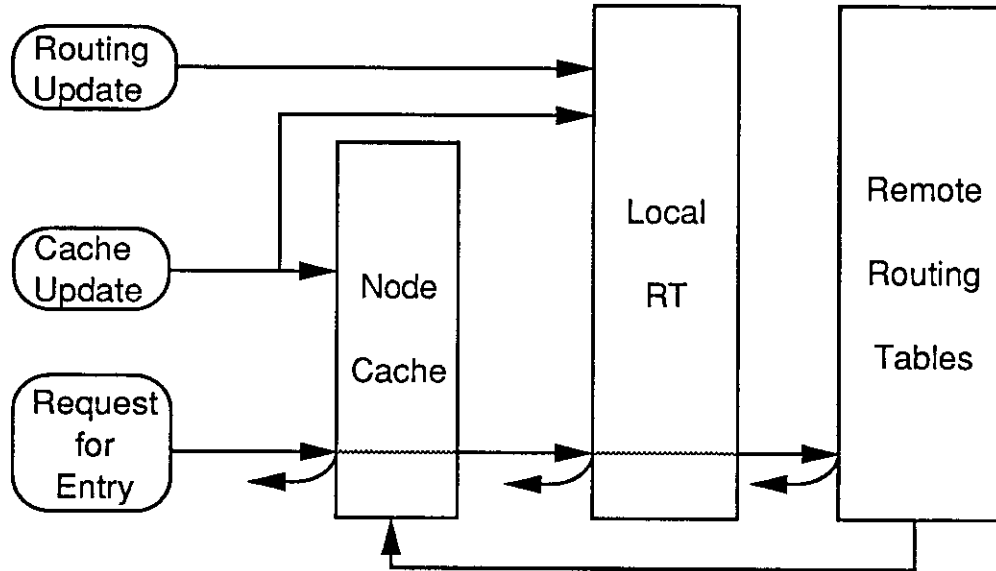


Figure 3.11 : Routing Hierarchy for Node Caching Scheme

it is updated.

- v. When node U receives a routing update message for an entry for q , the operating system checks if there is a entry for q in the local routing table and if so updates it.
- vi. When process q migrates no cache entries are taken with the process to the new location of q .
- vii. A node that receives a cache update message routes the message similarly to user messages.

The routing table hierarchy for the Node Caching Scheme is illustrated in Figure 3.11.

3.3 Analysis of Caching Schemes

In this section we study the effectiveness of the routing table hierarchy and in particular compare the relative merits and demerits of the different caching schemes. As discussed earlier (Section 3.2.1) one of the main motivations for introducing caches in the routing table hierarchy is to reduce the time to obtain a routing entry for a particular

process. The cost of obtaining a routing entry for a particular process is very high if there is a routing fault and the routing entry has to be obtained from remote nodes. Our objective in this section is to study how different parameters like the choice of caching scheme, the size of the local Send Cache and routing table, the rate of process migration etc. affect the cost of obtaining a routing entry.

Message communication between processes in the distributed system is assumed to have temporal locality, i.e. the destination process to which a process sends a message is likely to be one to which the process has sent a message in the recent past. Every time a process has to send a message it requests the operating system for a routing entry and hence the requests for routing entries follows a certain distribution. We propose a model for temporal locality of communication between processes and given the resulting distribution of requests for routing entries we determine the average time spent to obtain routing entries.

While the main motivation for introducing caches is to reduce the time to obtain a routing entry, the caches are kept updated in order to reduce the time for a message to reach the destination process. If the entries in the cache are out-of-date then the message has to be forwarded several times before it is delivered to its destination. In the previous section we discussed how Send Caches are updated by acknowledgements to user messages and by the mechanism of Receive Caches at the destination processes. These mechanisms maintain most of the Send Cache entries up-to-date when temporal locality holds and hence we did not study the performance of the updating of caches.

3.3.1 Analytical Model

3.3.1.1 Modeling Temporal Locality

Our model for temporal locality in the communication between processes is based on the Least Recently Used Stack Model (LRUSM) for generation of memory page references to model program behavior [Coffman 73]. Reed et. al. [Reed 87] adopted the LRUSM model to model network traffic, i.e. communication between nodes in a distributed system.

We associate a *stack* of size *NoProc* with each process, where *NoProc* is the total number of processes in the system. Each stack location contains a process name and has a probability associated with it. The probability that a process p sends a message to another process q is equal to the probability associated with the stack location that contains q in p 's process stack.

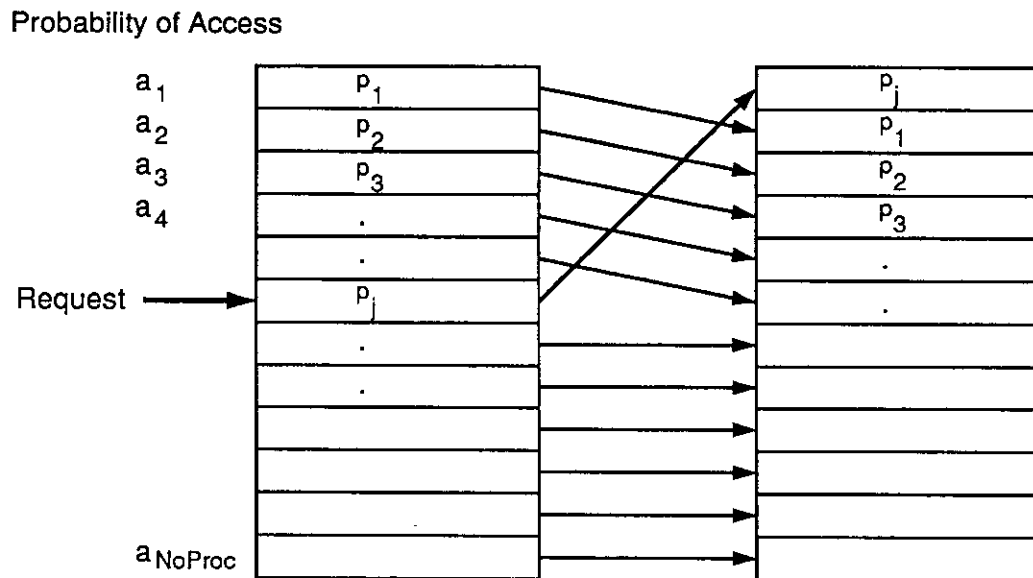


Figure 3.12 : Updating of the Process Stack After a Message is Sent

The stack is not fixed over time but rather the position of process names in the stack changes on the sending of a message by the process. Figure 3.12 illustrates the updating of the stack after a message is sent. On the left is the stack at time t and a message is sent at time $t + 1$ to the process whose name is in stack position j . As a result the process name in location j is moved to the top of the stack and all the process names in locations $1, 2, \dots, j - 1$ are pushed down one location. All other positions below j remain the same.

We define the *stack distance* d_t as the position of the destination process that is sent the t -th message in the stack of the source process. According to the LRU Stack Model the stack distances are independent random variables such that

$$Probability(d_t = i) = a_i, \text{ for all } t \text{ and } 1 \leq i \leq NoProc$$

where $\sum_{i=1}^{NoProc} a_i = 1$.

Thus at time t if a source process is sending a message, the probability of sending it to the process listed in location 1 of its stack is a_1 , to the process listed in location 2 is a_2 and so on. The *distance probabilities* a_i are stationary, i.e. they do not change over time.

The parameter a_i can be chosen to model different communication patterns. To model temporal locality as we defined in the previous section the distance distribution is biased towards shorter stack distances, i.e.

$$a_1 \geq a_2 \geq \dots \geq a_{NoProc}$$

Requiring $a_{NoProc} > 0$ is necessary and sufficient to guarantee that the source process will send messages to all processes in the system eventually. If the source process will never send messages to some processes then these processes are placed in the bottom positions of the initial stack and assigned $a_i = 0$.

In our simulation experiments we use this model for generation of messages by a process. We choose the distance probability $a_i = b \times LocalityBase^{-i}$, where $b = LocalityBase - 1$ and $LocalityBase > 1$. For large $NoProc$ we have $\sum_{i=1}^{NoProc} b \times LocalityBase^{-i} \approx 1$.

3.3.1.2 Performance Model for PCS (CM)

We evaluate the average time T_{Find_RE} to obtain a routing entry for a particular process for the routing hierarchy where the model for generation of messages and thereby the model for the request of routing entries by a process is the LRU Stack Model.

The policy of replacement of cache entries when a cache is full is chosen to be the Least Recently Used (LRU) scheme. The cache is ordered like a stack and is reordered each time an entry is accessed. Unlike a conventional stack however any entry in the cache can be accessed at any time. The cache entries are ordered by decreasing recency of reference. If a new entry has to be added to a cache that is full, the entry at the bottom of the cache which has not been accessed for the longest time is chosen to be discarded.

The new entry is added to the top of the cache and the other entries in the cache are pushed down to the next location.

The Process Caching Scheme (CM) is the most analytically tractable of the three schemes because its performance is independent of the rate of process migration. This is because in PCS(CM) an entry is added to the Send Cache whenever there is a *cache miss* and a Send Cache of size C contains the C most recently accessed entries for distinct processes. When a process migrates to a new location, the Send Cache for that process is also migrated along with the process. Hence after migration the process has access to the C most recently accessed distinct entries. The *cache miss ratio* or the fraction of requests that cannot be satisfied by the cache in this case is called a *warm start miss ratio*.

This is in contrast to the Process Caching Scheme (RF) where an entry is added to the Send Cache only if there is a routing fault, i.e. a cache miss occurs and the cache entry is not found in the local routing table either. In PCS (RF), soon after a process migrates the Send Cache of the process at the new location does not have entries that happened to be in the local routing table at the node where the process was previously located. Hence in PCS (RF) there will be an increase in the rate of cache misses soon after migration because the Send Cache of size C belonging to the process may not have all of the C most recently accessed distinct entries. In this case we call the cache miss ratio a *lukewarm start miss ratio*. For PCS(RF) the *hit ratio* or the fraction of requested entries found in the cache will be inversely proportional to the rate of migration.

In the Node Caching Scheme the Node Cache is shared by different processes resident on the same node. The interaction between the locality sets of different processes is difficult to model. In NCS when a process migrates to a new location no cache entries are carried to the new location and hence we call the associated miss ratio a *cold start miss ratio*.

In this section we study the performance of the routing hierarchy when the caching scheme is the Process Caching Scheme with placement on Cache Miss. The parameters of the model are given in Table 3.1.

<i>NoProc</i>	Total Number of Processes
<i>CacheSize</i>	Size of Send Cache
<i>RTSize</i>	Size of Local Routing Table
a_i	Distance Probabilities in the LRUSM model
T_{Cache}	Time to check for cache entry
T_{RT}	Time to check for Routing Table entry
T_{Remote_RT}	Time to access Remote Routing Table entry

Table 3.1 : Parameters for the Routing Hierarchy Model

We assume that the time to check the cache for a particular entry is the same as the time to check and return the entry. Let P_{Cache_Hit} be the cache hit ratio or the fraction of entries requested by the process that are found in the cache; and P_{RT_Hit} be the fraction of entries requested from and found in the routing table. The average time to obtain a routing entry is

$$\begin{aligned}
T_{Find_RE} &= P_{Cache_Hit} \times T_{Cache} \\
&\quad + (1 - P_{Cache_Hit}) \times P_{RT_Hit} \times (T_{Cache} + T_{RT}) \\
&\quad + (1 - P_{Cache_Hit} - (1 - P_{Cache_Hit}) \times P_{RT_Hit}) \times (T_{Cache} + T_{RT} + T_{Remote_RT}) \\
&= P_{Cache_Hit} \times T_{Cache} \\
&\quad + (1 - P_{Cache_Hit}) \times P_{RT_Hit} \times (T_{Cache} + T_{RT}) \\
&\quad + (1 - P_{Cache_Hit}) (1 - P_{RT_Hit}) \times (T_{Cache} + T_{RT} + T_{Remote_RT})
\end{aligned}$$

The explanation for the above expression is as follows. The operating system receives requests for routing entries from a process. The request of routing entries by a process is modeled by a LRU Stack for that process. The operating system first checks the Send Cache of the process for the particular routing entry and the probability of finding that entry is P_{Cache_Hit} . If the entry is found in the cache then the time taken is T_{Cache} . If the entry is not found in the cache, the operating system next checks the local routing table for the entry. The probability of requesting an entry from the routing table is $(1 - P_{Cache_Hit})$, and the probability of finding the entry in the routing table is P_{RT_Hit} . In the PCS(CM) the presence of a particular entry in the routing table is independent of the

presence of an entry in the cache and hence P_{RT_Hit} is independent of P_{Cache_Hit} . If the entry is found in the routing table then the total time to find the entry includes the time spent to check the cache before the routing table. If the entry is not found in the local routing table either then it is guaranteed to be found in one of the remote routing tables in the system. The probability of a routing fault is

$$P_{Routing_Fault} = (1 - P_{Cache_Hit}) (1 - P_{RT_Hit})$$

We assume an average time T_{Remote_Rt} to obtain a routing entry from a remote node. The methods for obtaining a remote routing entry will be discussed in Chapter V.

If the distance probabilities a_i are stationary, i.e. constant in time, then at any time after the cache has filled up the process names in the top $CacheSize$ locations of the LRU Stack of the process are identical to the entries of the cache. The average long term cache hit ratio is independent of the initial stack contents and is given by

$$P_{Cache_Hit} = \sum_{i=1}^{CacheSize} a_i$$

Since the routing table has $RTSize$ entries out of a total of $NoProc$ possible entries, and the selection of which routing entries to keep in a particular routing table is done in random, the routing table hit ratio is

$$P_{RT_Hit} = \frac{RTSize}{NoProc}$$

Therefore the average time to find a routing entry is

$$\begin{aligned} T_{Find_RE} = & \sum_{i=1}^{CacheSize} a_i \times T_{Cache} \\ & + \left(1 - \sum_{i=1}^{CacheSize} a_i \right) \times \frac{RTSize}{NoProc} \times (T_{Cache} + T_{RT}) \\ & + \left(1 - \frac{RTSize}{NoProc} \right) \left(1 - \sum_{i=1}^{CacheSize} a_i \right) \times (T_{Cache} + T_{RT} + T_{Remote_RT}) \end{aligned}$$

3.3.2 Simulation of Caching Schemes

We compare the performance of the caching schemes by simulating the routing hierarchy for the three different caching schemes. The simulation we have developed is based on SIMON II [Swope 86]. SIMON is an object oriented discrete event simulator with events represented by messages. Objects may be defined hierarchically in terms of other objects and can be created, deleted or moved. Explicit well defined connections between objects allow objects to exchange messages with each other. Messages carry a time stamp corresponding to the time at which the event modeled by the message occurs in the real system. SIMON manages the scheduling of events so that parallel execution between different components of a real system can be faithfully modeled. The main advantage of SIMON is that it is general purpose and flexible enough to permit the simulation of a wide range of real systems without being restricted by the simulator.

We consider a single source process that is requesting routing entries in order to send messages to other processes in the system. Message communication between processes is assumed to have temporal locality and the generation of requests by a process is modeled by the LRU Stack Model. The request for a routing entry is resolved by the routing table hierarchy consisting of the cache, local routing table and the remote routing table. The source process, i.e. the process requesting routing entries, can migrate from one node to another. The three caching schemes differ in when an entry is placed in the cache and in the effect of the migration of the process on the cache hit ratio.

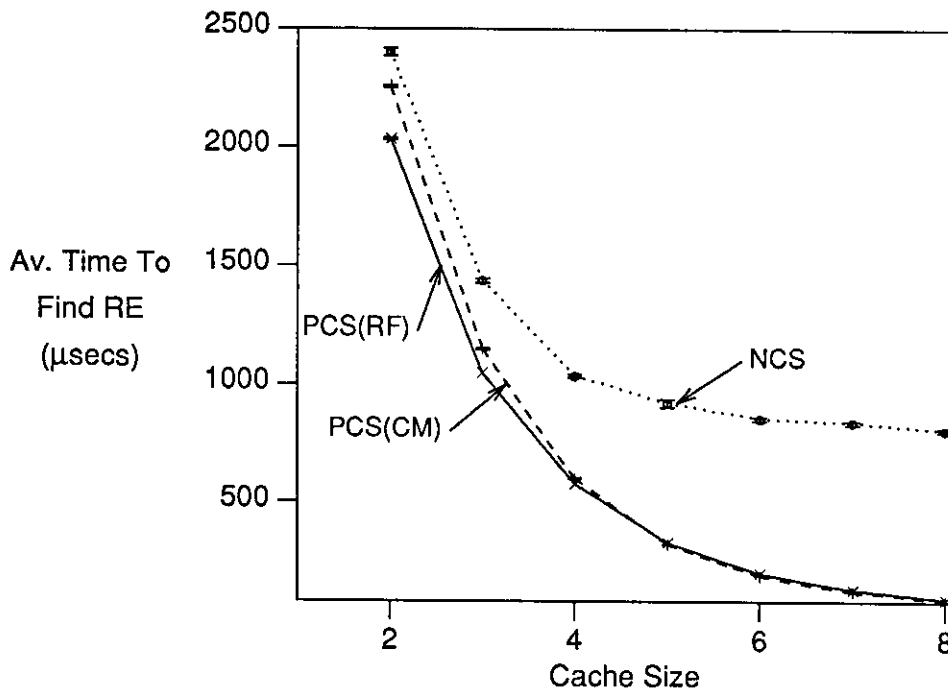
The simulation experiments compare the performance of the three schemes when different system parameters such as the rate of process migration, the size of caches, routing tables, etc. are varied. The parameters for the simulation are given in Table 3.2. All times are in microseconds. The residence time of a process at a node is chosen from a geometric distribution. In the LRU Stack model for the generation of requests for routing entries by a process, the distance probabilities a_i are chosen such that $a_i = b \times \text{LocalityBase}^{-i}$, where $b = \text{LocalityBase} - 1$ and $\text{LocalityBase} > 1$. Note that $a_1 > a_2 > \dots > a_{\text{NoProc}}$ and for large NoProc we have $\sum_{i=1}^{\text{NoProc}} b \times \text{LocalityBase}^{-i} \approx 1$. In our experiments we have generally chosen LocalityBase to be between 1.0 and 2.0. As the LocalityBase increases the degree of locality increases, i.e. the set of destination processes to which a message is most likely to be sent becomes smaller.

<i>MsgInterval</i>	Constant interval between requests for entries
<i>AvProcResTime</i>	Mean time a process is resident at a node
<i>DistResTime</i>	Distribution of residence time of a process at a node
<i>NoProc</i>	Total number of processes
<i>ProcPerNode</i>	Average number of processes resident on a node
<i>LocalityBase</i>	Degree of locality in the LRU Stack Model of process
<i>RTSize</i>	Size of a local routing table
<i>CacheSize</i>	Size of the cache
T_{Cache}	Expected time to access cache entry
T_{RT}	Expected time to access routing table entry
T_{Remote_RT}	Expected time to access remote routing table entry

Table 3.2 : Parameters for the Simulation

The routing table and the cache are of fixed size. The entries in the routing table at each node are chosen independently and randomly from the total possible set of routing entries. Creation of entries in the routing tables is done at the time of process creation. When a process moves to a new location the routing table at that node is different from the table at its previous location. Initially at the time the system is started up all caches are empty. Without loss of generality we assume the LRU stack of the process to be initialized to 1, 2, 3, . . . *NoProc* , i.e. the top of the stack contains 1 corresponding to process 1, next 2 and so on.

The average number of processes resident on a node (*ProcPerNode*) is not relevant for the Process Caching Schemes but is an important parameter for the Node Caching Scheme. The Node Cache is shared by all the processes resident on the node and entries of the Node Cache are replaced based on the LRU scheme. We assume that all processes on a node have the same rate of generation of routing requests. We measure the time to find a routing entry and collect statistics on the rate of cache misses and routing faults for a chosen process on the node. The LRU Stacks of the other processes on the same node are initialized so that process names in the stacks are in random order. When a process migrates to a new location, the node cache at the new location is initially filled with random entries. When we compare the performance of one of the Process Caching



MsgInterval	5 μsecs	ProcPerNode	2	T_{Cache}	50 μsecs
AvProcResTime	200 μsecs	LocalityBase	2.0	T_{RT}	150 μsecs
NoProc	60	RTSize	8	T_{Remote_RT}	10^4 μsecs

Figure 3.13 : Average Time to find Routing Entry (RE) vs. Cache Size

Schemes with the Node Caching Scheme we make sure that the sum of the cache sizes at a node are the same for the two schemes being compared. This implies that in the comparison of Process Caching Schemes and the Node Caching Scheme if the Send Cache in the Process Caching Scheme is of size C then the Node Cache has size $ProcPerNode \times C$.

For our experiments we choose $T_{Cache} = 50 \mu secs$, $T_{RT} = 150 \mu secs$ and $T_{Remote_RT} = 10^4 \mu secs$. Note that the expected time for a remote access is more than two orders of magnitude greater than the time for a cache access.

For the experiments whose results are presented here each simulation run consisted of 300,000 requests for routing entries by the process under observation. To eliminate from consideration start-up transient effects, the measurements were started after 400 requests had been resolved allowing sufficient time for the caches to fill up.

The performance measures of interest are the average time to obtain a routing entry and the fraction of routing faults. Every data point in the performance graphs is a result obtained from 5 experiments. We compute the 95 % confidence intervals using the *t distribution*. In the performance graphs the average is plotted as a point and vertical bars indicate the confidence intervals.

The surprising observation from our simulation experiments was that the Process Caching Scheme with placement on Cache Miss did not perform better than the Process Caching Scheme with placement on Routing Fault as often as we had expected it to. In fact PCS (RF) almost always performed better than PCS(CM). Figure 3.13 shows the average time to find the routing entry and its variation with the size of the cache for the three different schemes. The rate of migration of the process is rapid with the process migrating after having sent an average of 40 messages. In NCS, a process does not carry any cache entries when it moves to a new location and therefore a number of routing faults occur before entries for the favored set of destination processes are placed in the cache. Therefore when the cache size increases, the decrease in the time to find a routing entry for NCS is much less than the decrease for the Process Caching Schemes. Moreover note that for the Process Caching Schemes T_{Find_RE} continues to decrease as cache size is increased beyond $CacheSize = 5$ while for NCS the curve levels off as the cache size is increased beyond a total Node Cache size of $ProcPerNode \times CacheSize$ i.e. 10. For NCS when process migration is rapid and the degree of locality is high, the process migrates to a new location before its cache can be filled up and utilized.

The interesting observation from the graph of Figure 3.13 is that PCS(RF) performs better than PCS(CM) when the cache size is less than 5 even though process migration is rapid. The reason for this is that when the cache size is small, effective use of the memory space on a node is critical. PCS(RF) effectively utilizes the small memory space available by avoiding the replication of entries in the cache and the routing table. In PCS(RF) if an entry is found in the local routing table it is not added to the Send Cache thus making more cache space available to store entries that are likely to be used. Therefore the effective cache size for PCS(RF) is larger than for PCS(CM). In PCS(CM) the advantage is that all recently accessed entries are moved along with the process to the

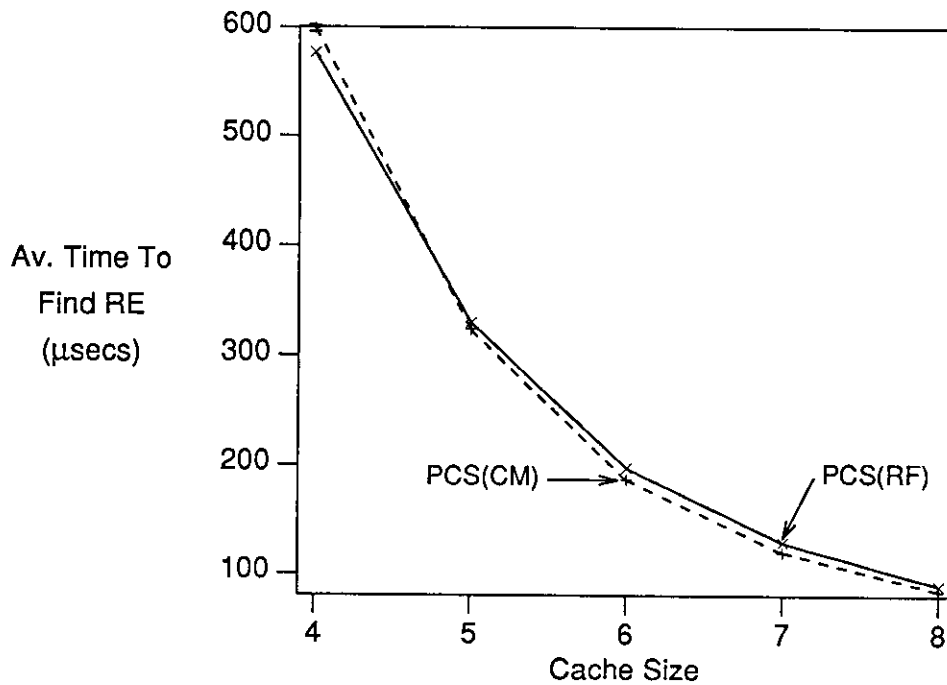
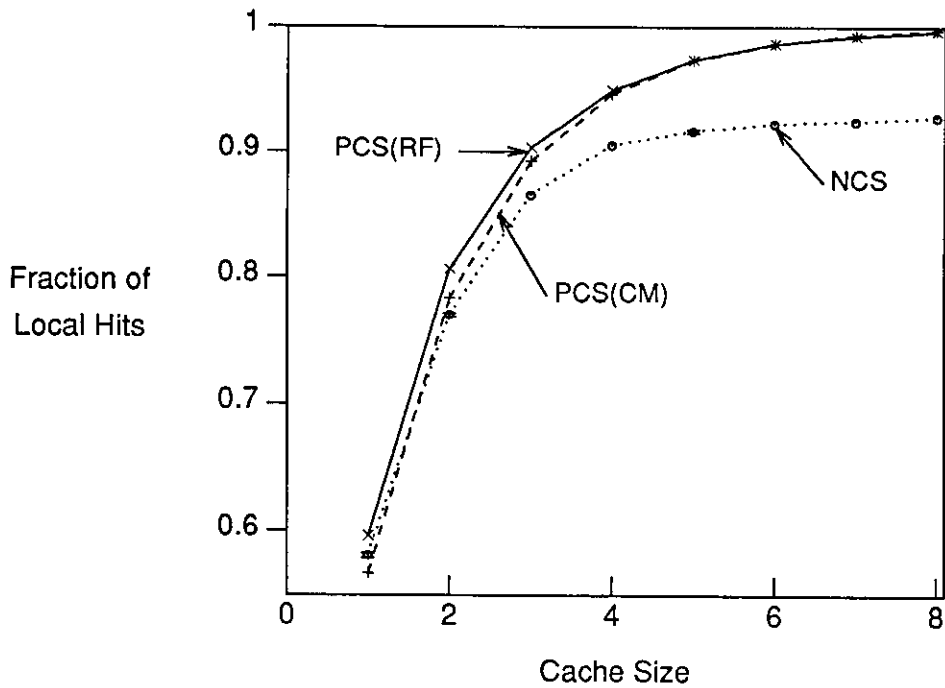


Figure 3.14 : Magnified View of Figure 3.13

new location thus avoiding routing faults that occur in PCS(RF) soon after process migration in order to obtain entries that were present in the routing table at the previous location. However from our simulations we observed that this has only a secondary effect on the performance and a single routing fault is all that is required to obtain and place in the cache an entry that was recently accessed but was located in the routing table of the previous location. In PCS(CM) the effective cache space decreases due to replication of entries in the Send Cache and the local routing table, resulting in repeated routing faults when the degree of locality is moderate.

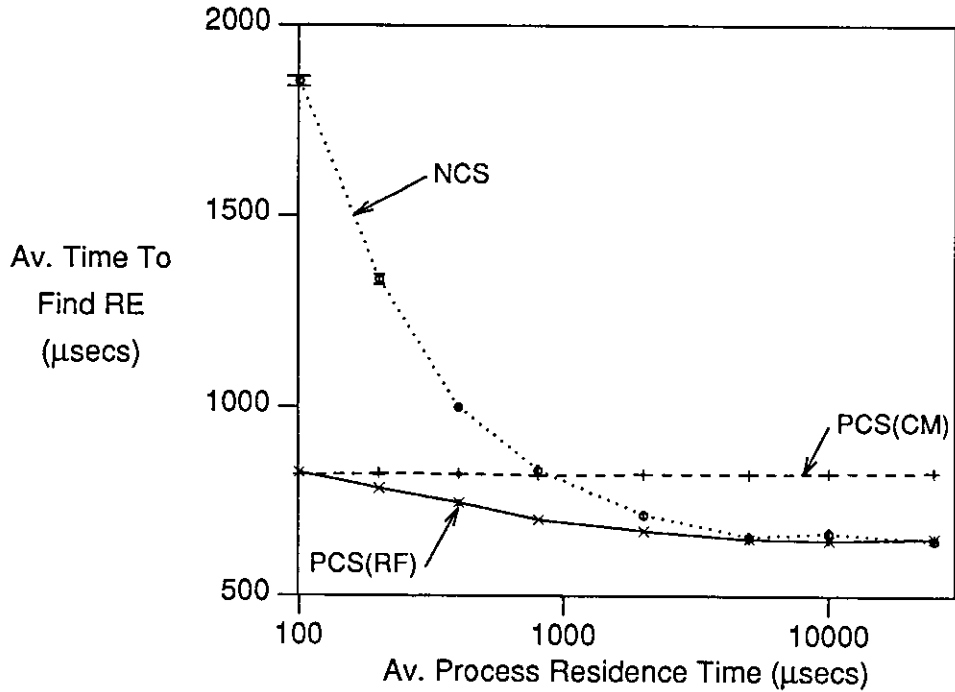
As the Send Cache size is increased beyond 5, PCS(CM) performs slightly better than PCS(RF). This is illustrated in Figure 3.14 which shows the region of the previous graph where the time to find a routing entry for PCS(CM) is smaller than the time for PCS(RF). For cache size greater than 5, the cache size is large enough to hold enough entries so that the probability of cache hit is very high. In this case it becomes more important to reduce the rate of routing faults after process migration instead of using the available cache space effectively.



MsgInterval	5 μ secs	ProcPerNode	2	T_{Cache}	50 μ secs
AvProcResTime	200 μ secs	LocalityBase	2.0	T_{RT}	150 μ secs
NoProc	60	RTSize	8	T_{Remote_RT}	10^4 μ secs

Figure 3.15 : Fraction of Local Hits vs. Cache Size

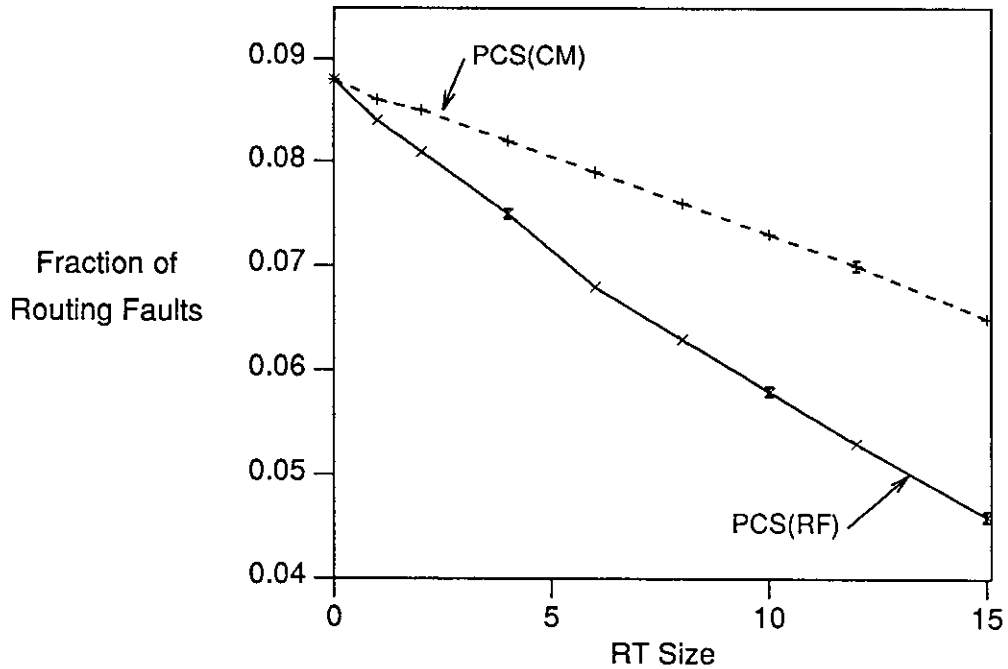
For the same experiment as Figure 3.13, Figure 3.15 plots the fraction of local accesses, i.e. accesses to the local cache and routing table, as the cache size is increased. Note that when the cache size is very small, the fraction of local hits is best for PCS(RF), then NCS and finally PCS(CM). When the cache size is very small schemes that effectively utilize local memory have a higher fraction of local hits. As the cache size increases the fraction of local hits for the Process Caching Schemes is much higher than for the Node Caching Scheme. Note as before that when the cache size increases there is no significant difference in the local hit ratio for the two Process Caching Schemes.



MsgInterval	5 μsecs	LocalityBase	1.5	T _{Cache}	50 μsecs
NoProc	60	CacheSize	6	T _{RT}	150 μsecs
ProcPerNode	2	RTSize	8	T _{Remote_RT}	10 ⁴ μsecs

Figure 3.16 : Average Time to Find Routing Entry (RE) vs. Average Process Residence Time

Figure 3.16 illustrates the effect of the variation of the rate of process migration on the average time to obtain a routing entry. As expected the average time to find a routing entry for PCS(CM) is independent of the rate of process migration. The change of the rate of process migration has the most pronounced effect on NCS where the average time to find a routing entry increases rapidly as the rate of migration increases. When process migration is very rapid the Process Caching Schemes perform significantly better than the Node Caching Scheme, because caches are not moved when the process moves in NCS resulting in a large number of routing faults after every migration. In the case when process migration is so frequent that all the cache entries are not accessed before the process is moved, then the performance of PCS(CM) is slightly better than PCS(RF). In PCS(RF) when the rate of process migration decreases the rate of routing faults

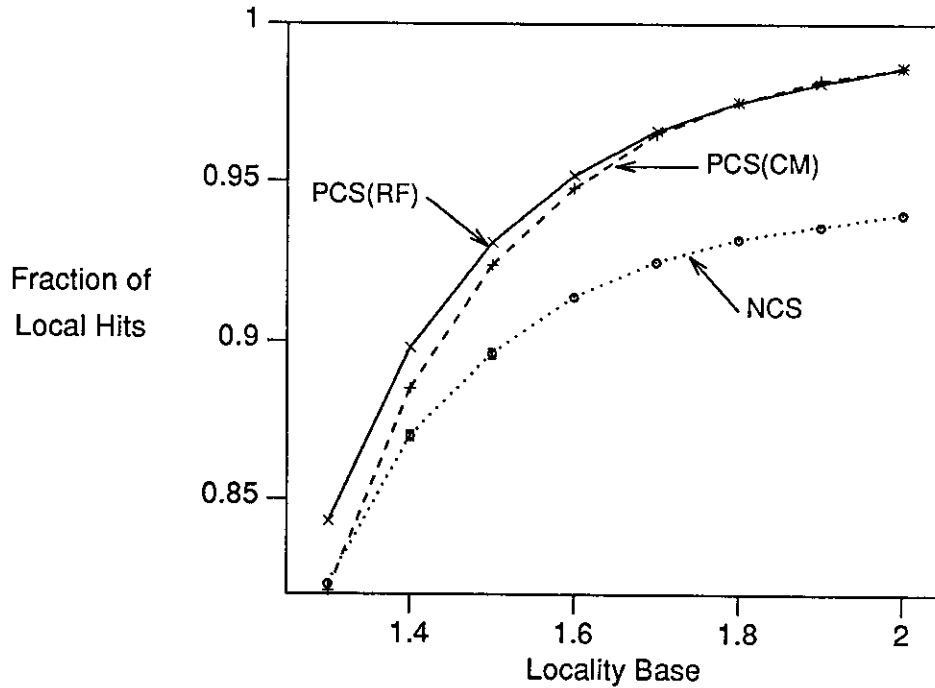


MsgInterval	5 μ secs	ProcPerNode	2	T_{Cache}	50 μ secs
AvProcResTime	800 μ secs	LocalityBase	1.5	T_{RT}	150 μ secs
NoProc	60	CacheSize	6	T_{Remote_RT}	10^4 μ secs

Figure 3.17 : Fraction of Routing Faults vs. RT Size

decreases, resulting in a lower average time to find a routing entry. When processes migrate relatively infrequently we observe that the performance of the Node Caching Scheme and the PCS(RF) schemes are similar and the choice would be to implement NCS because NCS is relatively easy to implement and the procedures for routing table and cache updating are simple.

In Figure 3.17 we keep the size of the cache fixed and observe the fraction of routing faults as the size of the routing table is increased. When *RTSize* is zero then PCS(RF) and PCS(CM) are identical. When the routing table size is increased the fraction of routing faults for both Process Caching Schemes decreases. The degree of improvement in performance of PCS(RF) is much greater than for PCS(CM) when the routing table size increases. This is because in PCS(RF) there is no duplication of entries



MsgInterval	5 μ secs	ProcPerNode	2	T_{Cache}	50 μ secs
AvProcResTime	300 μ secs	CacheSize	6	T_{RT}	150 μ secs
NoProc	60	RTSize	8	T_{Remote_RT}	10^4 μ secs

Figure 3.18 : Fraction of Local Hits vs. Locality Base

between the local routing table and cache and hence PCS(RF) is able to more effectively exploit the increased probability of routing table hits to increase the fraction of local hits.

In Figure 3.18 we compare the the performance of the three caching schemes as the Locality Base increases. We fix the cache size to be 6 and the average time for which a process is resident at a node to be 300 μ secs. If we define the degree of locality to be the probability that the next message sent by a process is one of the last 6 distinct destinations to which a message was sent, then

$$Degree\ of\ Locality = \sum_{i=1}^6 b \times LocalityBase^{-i}$$

where $b = LocalityBase - 1$.

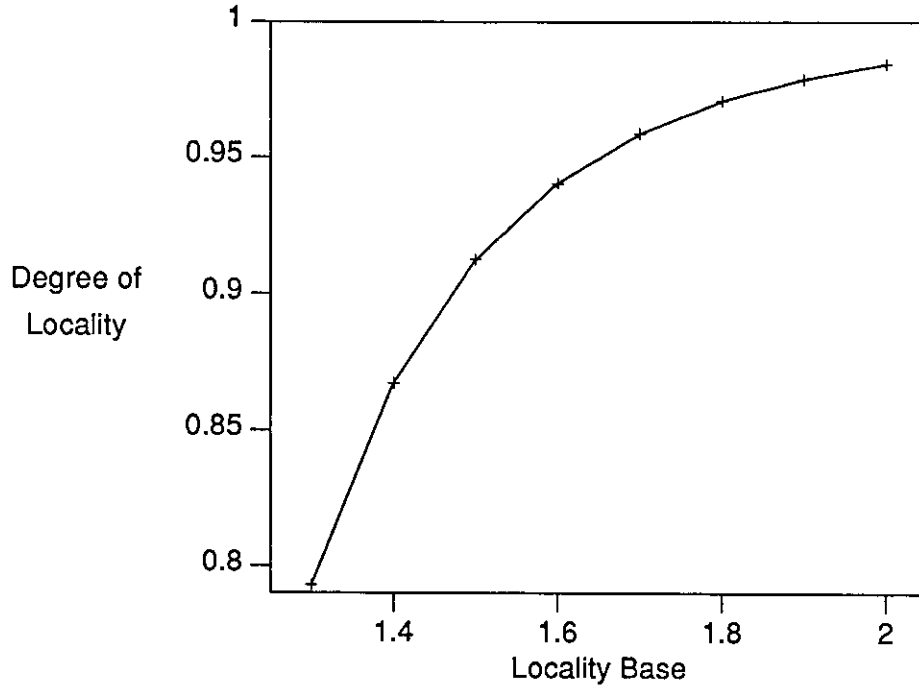


Figure 3.19 : Degree of Locality vs. Locality Base

In Figure 3.19 we plot the change in the Degree of Locality with the change in the Locality Base parameter.

We observe in Figure 3.18 that the behavior of the caching schemes as the locality base is varied is similar to their behavior (Figure 3.15) when the size of the cache is varied. This is because the size of the cache required is directly related to the degree of temporal locality in the communication. At relatively low degrees of locality (LocalityBase = 1.3), PCS(RF) performs significantly better than the other two schemes. More significantly at low degrees of locality and with fairly rapid rates of migration PCS(CM) and NCS have similar performance. However as the degree of locality increases the Process Caching Schemes do significantly better than the Node Caching Scheme. At high degrees of locality the two Process Caching Schemes perform similarly.

<i>NoProc</i>	60
<i>LocalityBase</i>	2.0
<i>RTSize</i>	8
<i>T_{Cache}</i>	50 μ secs
<i>T_{RT}</i>	150 μ secs
<i>T_{Remote_RT}</i>	10,000 μ secs

Table 3.3 : Parameters for the Analytical Model of PCS(CM)

Finally we compare the results from the analytical model for the performance of PCS(CM) with results obtained from the simulation of PCS(CM). The parameters for the analytical model are given in Table 3.3. In addition in the simulation we assume *MsgInterval* = 5 μ secs and *AvProcResTime* = 200 μ secs. The average time to find a routing entry as determined from the analytical and simulation models is given in Table 3.4.

<i>CacheSize</i>	<i>T_{Analytical}</i> (μ secs)	<i>T_{Simulation}</i> (μ secs)
1	4458.3	4455.0 \pm 12.0
2	2254.2	2255.8 \pm 7.1
3	1152.1	1148.6 \pm 6.6
4	601.0	600.0 \pm 5.3
5	325.5	323.6 \pm 2.5
6	187.8	187.0 \pm 1.3
7	118.9	120.0 \pm 2.2
8	84.4	84.2 \pm 0.9

Table 3.4 : Comparison of the Analytical and Simulation Models

The average time to find a routing entry calculated from the analytical model is found to be almost identical to the values obtained from the simulation model.

From the simulations we conclude that in a majority of the cases

$$T_{PCS(RF)} < T_{PCS(CM)} < T_{NCS}$$

where $T_{PCS(RF)}$ is the average time to find a routing entry for PCS(RF). We observe from the simulations that the effective use of memory space by avoiding duplication of entries between the cache and local routing table in PCS(RF) has a greater impact on performance than the advantage in PCS(CM) of moving *all* the recently used entries along with the process to the new location of the process thus avoiding an increase in the rate of routing faults soon after migration. We observe that on the average the performance of PCS(RF) is significantly better than PCS(CM) except when cache sizes are very large, the rate of process migration is very rapid or the degree of locality in communication is very large. In these cases PCS(CM) performs slightly better than PCS(RF).

CHAPTER IV

Non-Uniform Updating of Routing Entries

4.1 Introduction

In the Basic Routing and Migration (BRM) protocol, when a process moves from a source node to a destination node the only routing entries that are updated are the entries for the process at the destination node and at the source node and its surrogates. Thus in the worst case in the BRM protocol, a message for a process can retrace the entire trajectory of the process since its creation.

The addition of caches to the BRM protocol provides a mechanism to locally maintain up-to-date entries for processes to which messages had been sent in the near past. The caching schemes thus help in reducing the cost of repeated communication between a source process and a destination process after an initial message has been sent between them.

In this chapter we introduce protocols for updating routing entries for a process when the process moves from one node location to another. The objective is to keep routing entries in the system up-to-date in order to reduce the number of times a message is forwarded when the message is the first one sent in the recent past from a source process to a destination process. Repeated communication between two processes is indicative of the presence of temporal locality and is handled by the caching schemes proposed in the previous chapter. Here we develop protocols to prevent the first message from a source to the destination process from retracing the path of the destination process since its creation.

The efficiency of routing schemes is measured in terms of the *Message Trajectory Stretch Ratio* and the *Time Stretch Ratio*.

The *Message Trajectory Stretch Ratio* (MTSR) is the ratio of the actual length of a message trajectory in hops to the shortest distance between the source of the message at the time of sending and the destination at the time of the delivery. The stretch ratio as a measure of the performance of routing schemes has been used previously by Kleinrock and Kamoun [Kleinrock 77] and Peleg and Upfal [Peleg 87].

The *Time Stretch Ratio* (TSR) is the ratio of actual time taken by a message to be delivered to the shortest time to send a message from the source of the message at the time of sending to the destination at the time of the delivery. The Time Stretch Ratio is not equivalent to the Message Trajectory Stretch Ratio because the time taken to route messages and the time spent by messages at intermediate buffers are taken into account in calculating the Time Stretch Ratio.

The *Update Bandwidth* is a measure of the message overhead for updating routing tables. Update Bandwidth is measured in terms of message-hops/sec. If an update message traverses h hops then the contribution by this update message to the number of updates is h message-hops.

The principal protocol that we present here is the *Shifting Neighborhood Protocol* (SNP). The other protocols that are introduced here are variations of the SNP. In terms of performance the Shifting Neighborhood Protocol simultaneously achieves stretch ratios close to 1 while keeping the update bandwidth low. The stretch ratio achieved by SNP is significantly lower than the BRM protocol where only forwarding addresses are left when a process moves. The update bandwidth incurred by SNP is significantly smaller than CBM (Section 2.4) where a complete broadcast results when a process moves to a new location. The Shifting Neighborhood Protocol is designed to be scalable with the size of the system unlike the BRM and CBM protocols. In SNP the stretch ratio (an indication of message latency) and the update bandwidth can be traded off by appropriately choosing the parameters of the protocol, based on the stretch ratio requirement and the bandwidth available.

The motivation for the Shifting Neighborhood Protocol is that distant nodes need not know about small perturbations in the location of a process. The reason for this is two fold. The performance goal of the routing scheme is to minimize the maximum stretch ratio between any two processes in the system. We do not minimize the maximum difference between the length of the message trajectory and the shortest distance between the location of the source and the location of the destination. Therefore distant nodes can be updated less frequently than nearby nodes, because the relative increase in the path length that a message from a distant process has to take will be small compared to the actual distance to the destination process. The second reason is because of our assumption of spatial locality in the communication between processes. In a system where processes are mapped to nodes to minimize communication costs, a process is more likely to send messages to nearby processes than distant processes. This is another justification for keeping nearby nodes more up-to-date than distant nodes.

In the *multi-level* schemes that we propose here, routing entries for a process are uniformly distributed throughout the system but entries in different levels are updated with different frequencies. This is in contrast to schemes where more routing entries for a process are concentrated close to that process. The problem with this approach is that when a process moves, routing entries for that process also have to be moved to maintain the same distribution of entries around the process.

The multi-level protocol we propose here is different from the hierarchical protocol proposed by Kleinrock and Kamoun [Kleinrock 77] in a number of ways. Their mechanism is for message routing to nodes whose location is fixed. In their mechanism the network is statically partitioned into clusters and node addresses inherently indicate the cluster they belong to. We on the other hand are interested in routing messages to processes that don't have fixed locations but migrate. Instead of partitioning the network statically, we utilize the concept of the *neighborhood* of a process that is not fixed but can change dynamically depending on the process location. While in Kleinrock and Kamoun's mechanism different levels have different degrees of information about the location of a node, in our mechanism routing entries are distributed uniformly over all levels but entries at different levels are updated with varying frequency. Our scheme is a *flat* scheme and not hierarchical. Thus the multi-level organization serves to reduce bandwidth in our scheme, while it serves to reduce storage in Kleinrock and Kamoun's mechanism.

In an m -level Shifting Neighborhood Protocol, neighborhoods $\{N_1, N_2, N_3 \dots N_m\}$ are associated with a process, where a neighborhood is a set of nodes such that $N_1 \subset N_2 \subset N_3 \dots \subset N_m$. A node may belong to several neighborhoods. The neighborhood N_m contains all the nodes in the system. Figure 4.1a illustrates a process and its neighborhoods in a 3-level SNP. Nodes in N_1 are in the immediate proximity of the location of the process while nodes in $N_2 - N_1$ are further away from the process. Based on the assumption that transmission costs are small in comparison to the rest of the communication costs, all distances are measured in terms of number of hops and not hop lengths.

The Shifting Neighborhood Protocol can be implemented for systems with a large range of topologies. The only requirement is that the diameter of the connected graph associated with the topology should grow with the number of nodes in the graph. This excludes topologies with trivially small diameters like star graphs and completely connected graphs.

4.2 Shifting Neighborhood Protocol

Consider a large system with a particular process q that is migrating in the system. Without loss of generality and for ease of description let us assume all nodes have a routing entry for q . The protocol can be applied similarly to systems where nodes have incomplete routing tables. Figures 4.1 and 4.2 illustrate the operation of a 3-level SNP. In Figure 4.1a, initially process q is located at node U and associated with the process is a set of neighborhoods $\{N_1, N_2, N_3\}$ with radii $\{r_1, r_2, r_3\}$ respectively. Neighborhood N_1 covers a region in the immediate proximity of U , N_2 covers a larger region around U and N_3 covers the entire system. The radius of the system is r_3 . Initially the routing entries for q at all nodes indicate that q is located at U and the entries have a migration-count of zero.

If process q moves from node U to node V that is within N_1 , by the BRM protocol the entry for q at U is updated to indicate that q is now located at V . In addition, in SNP routing updates indicating the new location of q are broadcast to all nodes that are in N_1 . This is illustrated in Figure 4.1b. On receiving updates the routing entries at these nodes are modified to indicate that q is located at V .

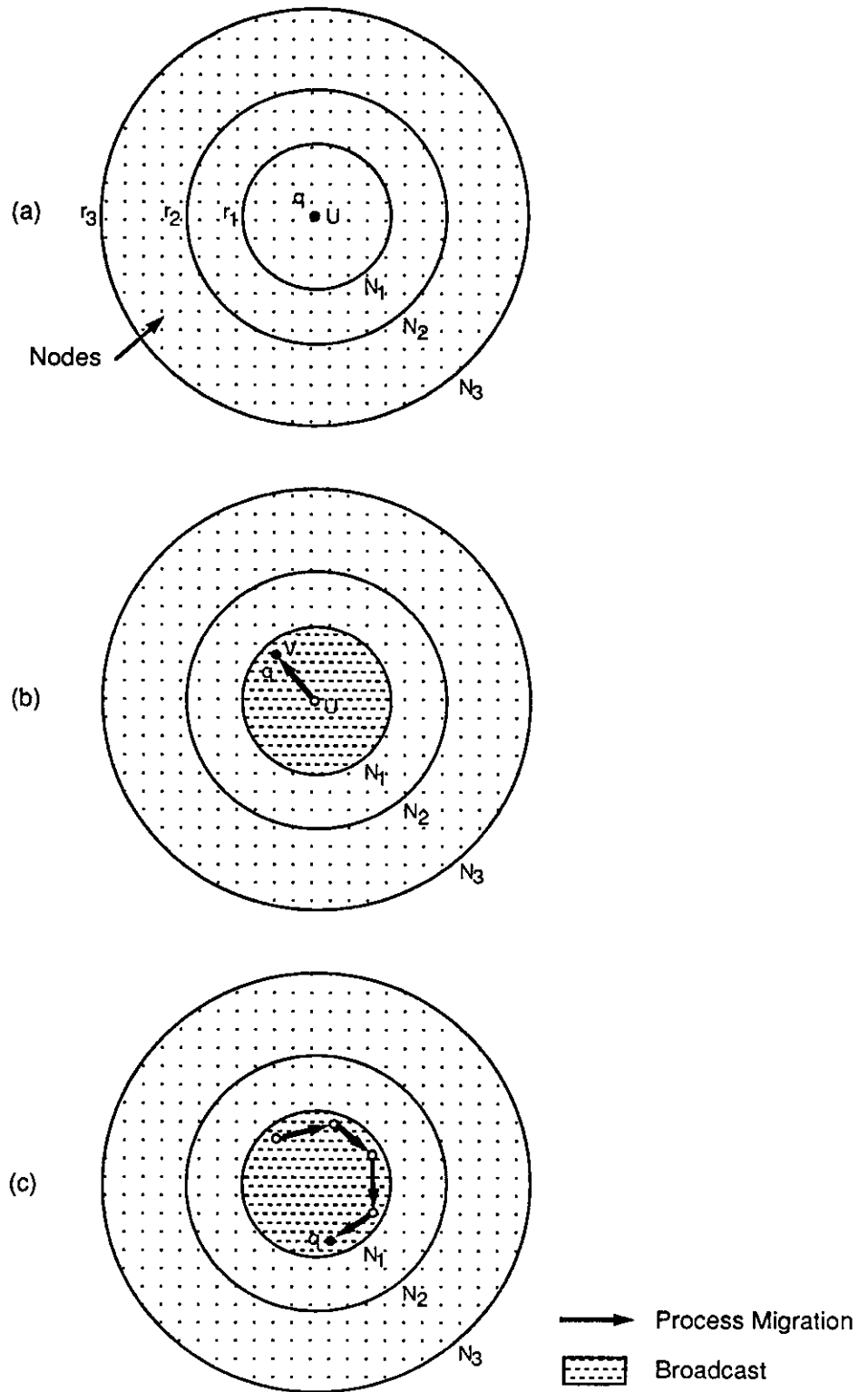


Figure 4.1 : Process Migration within the Lowest Level Neighborhood in SNP

Subsequently if q migrates to locations that are in N_1 (Figure 4.1c), then after every move routing updates are broadcast to the nodes in the region N_1 . Thus as long as the process moves to a location within the lowest level neighborhood routing updates are broadcast to all nodes belonging to the lowest level neighborhood. Note that as in previous protocols there is a migration-count associated with a process which is incremented whenever the process is moved, and when a routing table is updated, any modification to the entry is made only if the update has a higher migration-count.

Now if the process at node W migrates to node X (Figure 4.2a) that is in N_2 but outside N_1 , then routing updates are broadcast to all the nodes in N_2 which includes nodes of N_1 . Furthermore N_1 is reconstituted (Figure 4.2b) to consist of those nodes in N_2 that are within distance r_1 of X . Note that N_1 is constructed to be a proper subset of N_2 . Thus when the process moves out of the lowest level neighborhood but within the second level neighborhood routing updates are broadcast to all nodes in the second level neighborhood, and the lowest level neighborhood is reconstituted.

Subsequently the process q can move to locations within N_1 resulting in the broadcast of updates to the nodes in N_1 . This is shown in Figure 4.2c.

Similarly when q moves to a location that is outside N_2 then routing updates are broadcast to the neighborhood at the next higher level which in this case is N_3 the entire system. As a consequence neighborhoods N_1 and N_2 are reconstituted.

The main feature of SNP is that nodes in the lowest level neighborhood of a process have up-to-date information about the exact location of the process while nodes belonging to the second level neighborhood but not the first level neighborhood contain more out-of-date and more approximate information about the process location. Thus the multi-level scheme provides a mechanism to update nodes at different levels with different frequency.

The trajectory of messages to q from processes located at different nodes of the system is illustrated in Figure 4.3. For the purpose of this illustration we assume that the time for update messages to be broadcast is small compared to the time between consecutive migrations of a process. Let us consider the case when a process p sends a message to q , where p is located at a node in the neighborhood N_3 of process q but outside N_2 . When a message is sent from process p to q , the message is first routed to X the *center* of N_2 . The last time the routing entry at the node, where p is located, was

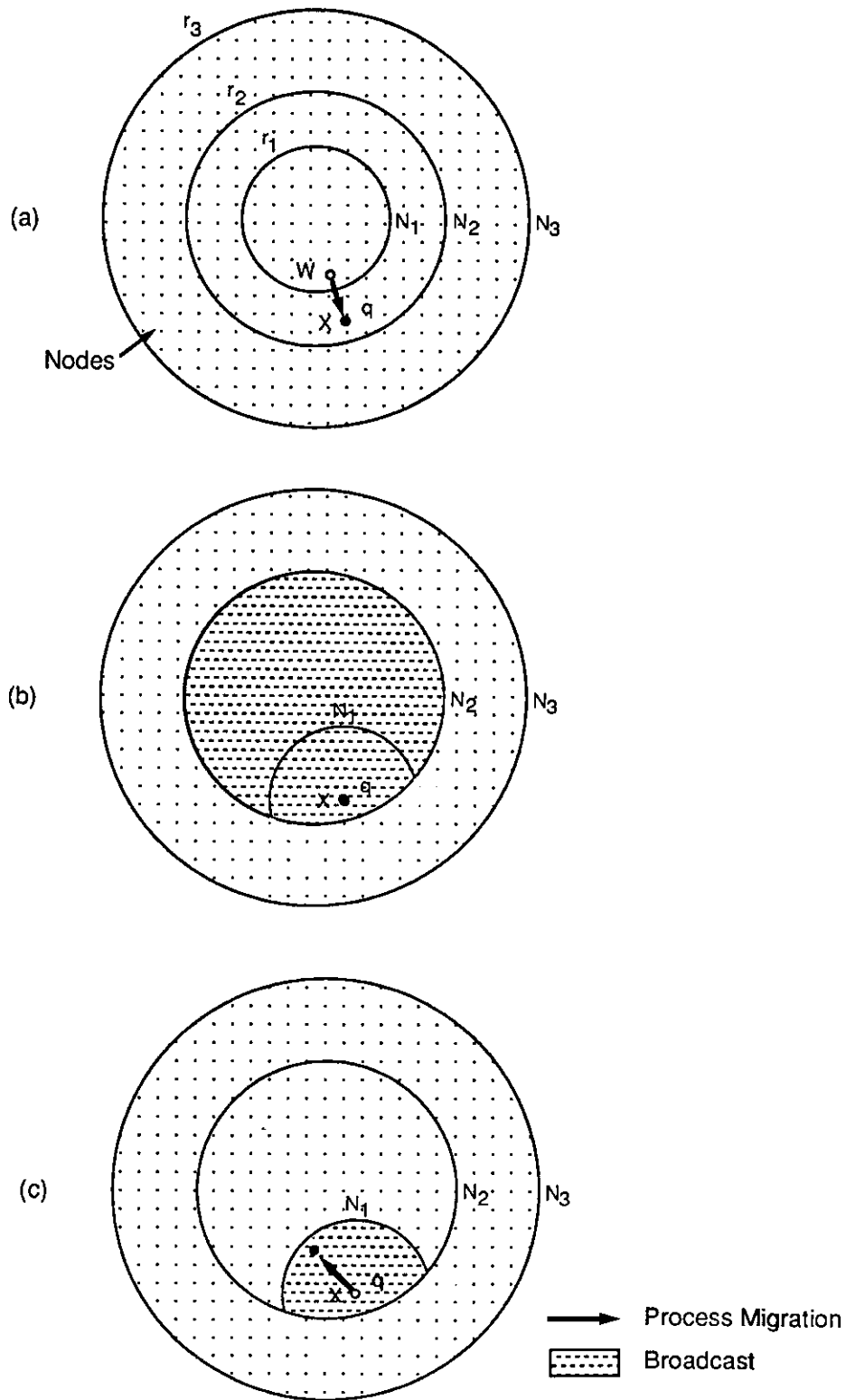


Figure 4.2 : Migration of a Process outside its Lowest Level Neighborhood in SNP

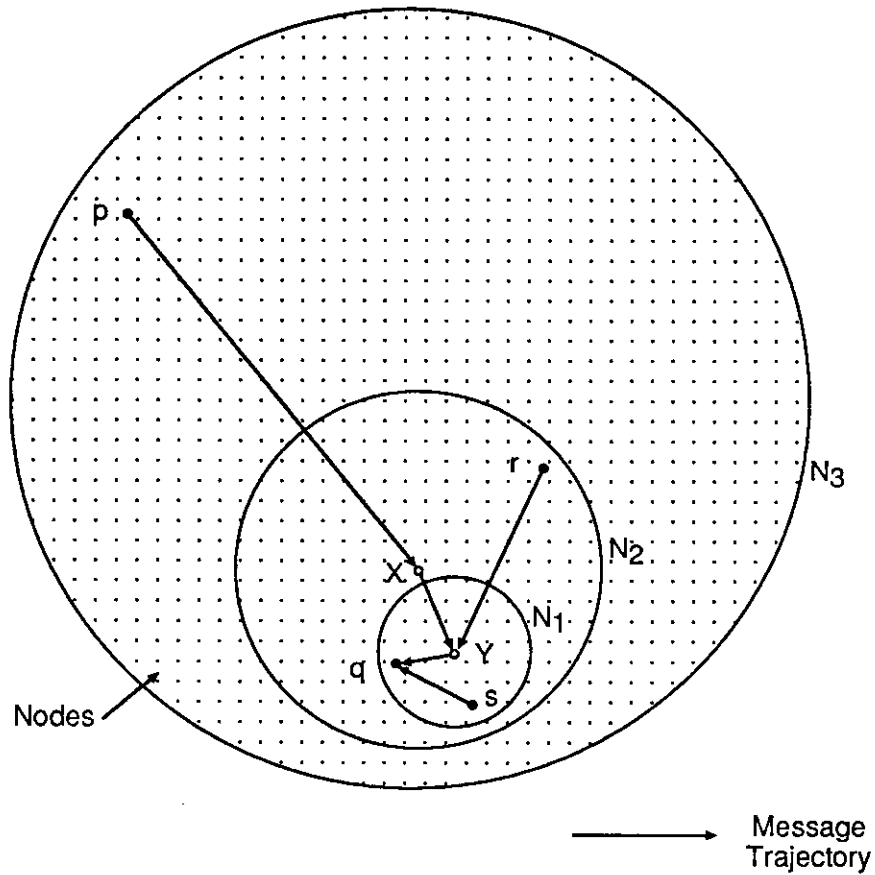


Figure 4.3 : Trajectory of Messages to Process q in SNP

updated the process q was located at X . Since the process is no longer located at X the message is forwarded to Y the center of N_1 . The last time the routing entry at X was updated the process q was located at Y . Assuming that the time for update messages to be broadcast is small compared to the time between consecutive migrations of a process all nodes in N_1 know the current location of q . Hence node Y that belongs to N_1 forwards the message to the current location of q where it is delivered to q . Thus if we assume that the process q does not move while a message from p is enroute to it, then the message from p to q takes three message trajectory hops to be delivered to the destination.

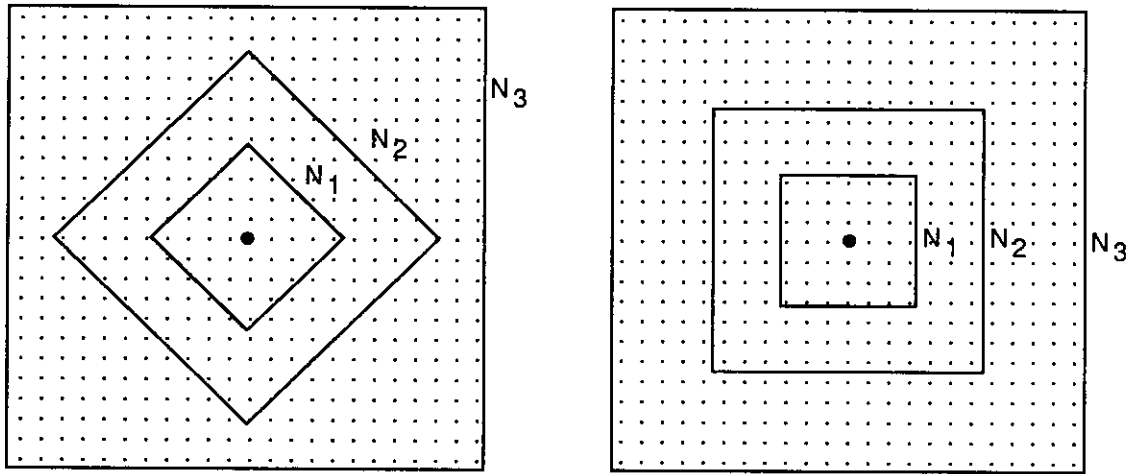


Figure 4.4 : Alternate Decompositions of a 2-d Toroidal Mesh into Neighborhoods

Similarly a message from process r is routed to node Y the center of neighborhood N_1 of the process q . From node Y the message is forwarded to the current location of the process and delivered to the process.

A message from a process s located in neighborhood N_1 of process q is directly routed without being forwarded to p .

Thus we observe that the number of times a message is forwarded is related to the neighborhood structure and the distance of the originating process from the destination process. If a message originates from a process located at a node that is outside N_{i-1} but in N_i then the message is at most forwarded $i - 1$ times before it is delivered, assuming that the time for update messages to be broadcast is small compared to the time between consecutive migrations of a process and the process does not move while a message is enroute to it. Thus we can establish a bound of $m - 1$ on the maximum number of times a message can be forwarded before its delivery for a m -level SNP, if the destination process does not move while the message is in transit.

The region covered by a neighborhood need not consist of all the nodes within a distance r from the center of the neighborhood. The region of the topology covered by a neighborhood N_i can have an arbitrary shape with the constraint that $N_{i-1} \subset N_i \subset N_{i+1}$. Figure 4.4 shows a toroidal mesh with two possible ways of decomposing the system into square regions. In the rest of this section we choose a neighborhood of radius r_i consisting of all nodes that are within distance r_i of the center of the neighborhood.

For the implementation of SNP we associate a data structure with each process that is stored in the process control block. The structure is an array *Nbhd_Center* to keep track of the centers of the neighborhoods of the process. *Nbhd_Center*[1] contains the location of the center of neighborhood N_1 , *Nbhd_Center*[2] contains the location of the center of neighborhood N_2 and so on. It is not necessary to represent *Nbhd_Center*[0] - the current location of the process as this can be trivially determined by the process. The array index ranges from 1 to *NoLevels* - 1, where *NoLevels* is the number of levels in the protocol.

In addition each process needs to know the radii of its neighborhoods in order to determine if the process has moved out of one of its neighborhoods. In case the neighborhood is not the set of all nodes within a certain radius from the center, a descriptor of the neighborhood has to be kept in order to determine if an arbitrary node belongs to a neighborhood. With each process we associate an array *Nbhd_Radius* of constants, with index from 1 to *NoLevels*, where *Nbhd_Radius*[*NoLevels*] is the radius of the system. The optimal radii for processes depends on their average move size and hence we associate the data structure containing the neighborhood radii with a particular process instead of a node so that different processes at a node may choose to have different numbers of levels and different radii of neighborhoods.

The m -level SNP is informally described below. We assume that array *Nbhd_Radius* contains the radii of levels 1 to m .

1. Initially when a process q is created at a node U the routing entry for q indicating its location and migration-count is broadcast to all the nodes in the system. *Nbhd_Center*[1], *Nbhd_Center*[2], *Nbhd_Center*[$m-1$] are all initialized to U .
2. When process q moves from node V to W , the operating system at W determines the maximum j ($1 \leq j \leq m-1$) such that $Distance(W, Nbhd_Center[j]) > Nbhd_Radius[j]$, where $Distance(X, Y)$ is the length of the shortest path from

node X to Y . If there is no j for which the above inequality is true then we set j equal to zero. Let $i := j + 1$.

- (i) The operating system at W sends a REQ_BROADCAST_UPDATE message containing parameter $Nbhd_Radius[i]$ and the location and migration-count of q to the node $Nbhd_Center[i]$.

The operating system at node $Nbhd_Center[i]$ on receiving the REQ_BROADCAST_UPDATE message, updates the local routing entry for q and broadcasts routing updates containing the latest location W and migration-count of q to all nodes within a distance $Nbhd_Radius[i]$ of the node $Nbhd_Center[i]$.

- (ii) If $i > 1$ then operating system at W reassigns the values of $Nbhd_Center[1] \dots Nbhd_Center[i-1]$ to be W .

Note that the degenerate 1 level SNP is equivalent to CBM (Section 2.4) where a complete broadcast results when a process moves to a new location.

4.3 Shifting Neighborhood Protocol with Dynamic Broadcast

The average message trajectory stretch ratio for the Shifting Neighborhood Protocol (SNP) for arbitrary source and destination process locations is calculated in Section 4.6.2 and observed to be close to one. However the worst case stretch ratio for the SNP can be arbitrarily large in some anomalous cases. Consider the situation in Figure 4.5 where process p at node U sends a message to process q at node V . Figure 4.5 illustrates the neighborhoods $\{N_1, N_2, N_3\}$ of process q with radii $\{r_1, r_2, r_3\}$. The message from process q that is located at a node outside N_2 is first routed to node X , the center of N_2 , then to node Y , the center of N_1 , and finally to node V where it is delivered to process q . The message trajectory stretch ratio for this message is

$$\begin{aligned}
 \text{Msg. Traj. Stretch Ratio} &= \frac{\text{Distance}(U,X) + \text{Distance}(X,Y) + \text{Distance}(Y,V)}{\text{Distance}(U,V)} \\
 &\approx \frac{2r_2}{\delta}, \text{ where } \delta = \text{Distance}(U,V)
 \end{aligned}$$

If δ is 1 hop length, then the resulting stretch ratio of the message is approximately $2r_2$, which can grow arbitrarily large in large systems.

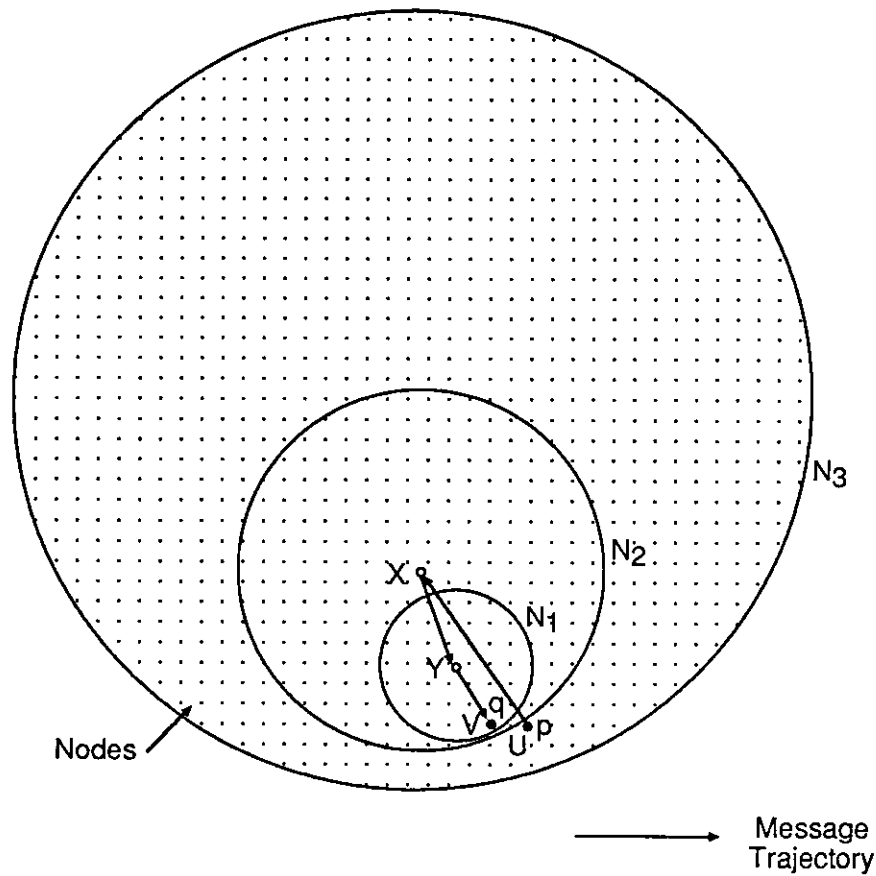


Figure 4.5 : Anomaly Resulting In Large Stretch Ratio in SNP

Thus there are anomalies in the performance of SNP when the originating process of a message and the destination process are located close to each other but on opposite sides of the boundary of the neighborhood of the destination process. The situation is further worsened in the above situation when there is spatial locality because the probability that those two nearby processes communicate with each other is very high. We propose two protocols that are modifications of SNP to resolve this boundary effect - the Shifting Neighborhood Protocol with Dynamic Broadcast (SNP(DB)) and the Shifting Neighborhood Protocol with Adaptive Routing (SNP(AR)). In this section we present SNP(DB) and in Section 4.5 we will present SNP(AR).

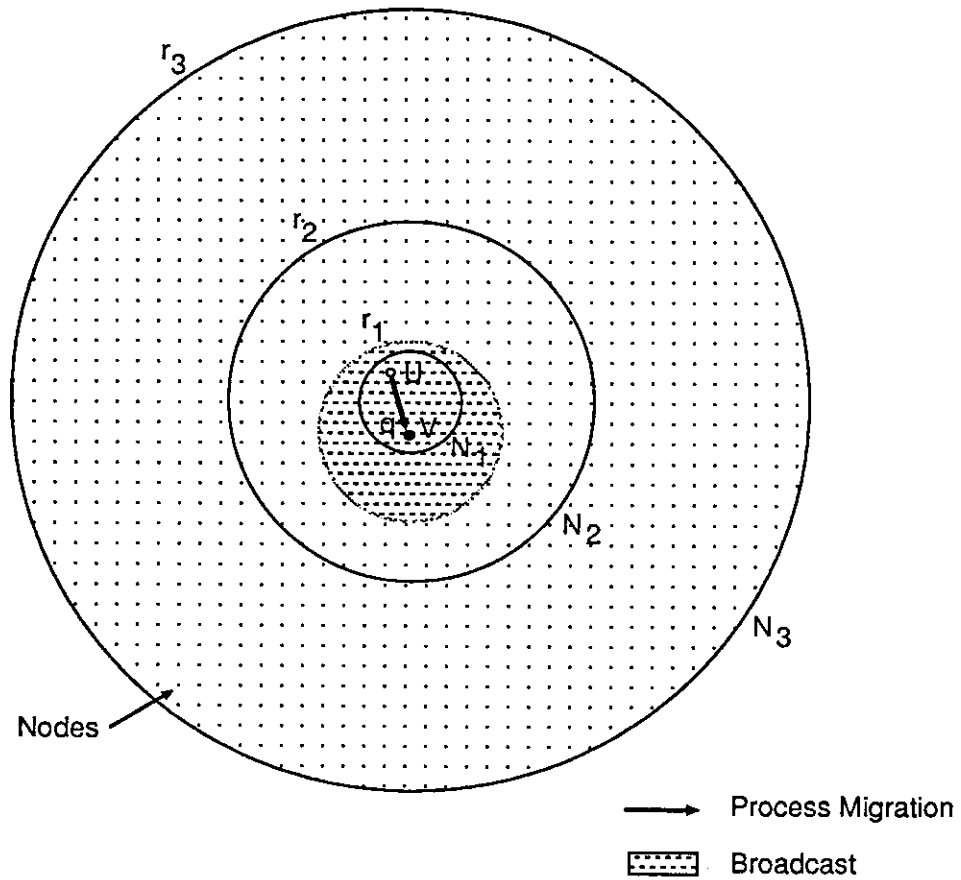


Figure 4.6 : Process Migration in SNP(DB)

In the Shifting Neighborhood Protocol with Dynamic Broadcast (SNP(DB)), the source of an update broadcast after the migration of a process is the current location of the process rather than the center of the neighborhood of the process as in the SNP. This is illustrated in Figure 4.6 where process q moves from node U to V . $\{r_1, r_2, r_3\}$ are the radii of neighborhoods $\{N_1, N_2, N_3\}$ of process q . When q moves to node V that is in N_1 routing updates are broadcast to all nodes within radius $2r_1$ of the node V . This is in contrast to SNP where routing updates would be broadcast to all nodes within radius r_1 from the center of N_1 .

If the process moves to a location outside N_1 but within N_2 then routing updates are broadcast to all nodes within radius r_2 of the current location of the process and the neighborhood N_1 of process q is reconstituted.

The m -level SNP(DB) is informally described below. We assume that array $Nbhd_Radius$ contains the radii of neighborhoods corresponding to levels 1 to m .

1. Initially when a process q is created at a node U the routing entry for q indicating its location and migration-count is broadcast to all the nodes in the system. $Nbhd_Center[1]$, $Nbhd_Center[2]$, $Nbhd_Center[m-1]$ are all initialized to U .
2. When process q moves from node V to W , the operating system at W determines the maximum j ($1 \leq j \leq m-1$) such that $\text{Distance}(W, Nbhd_Center[j]) > Nbhd_Radius[j]$. If there is no j for which the above inequality is true then we set j equal to zero. Let $i := j + 1$.

(i) The operating system at W broadcasts routing updates containing the latest location W and migration-count of q to all nodes within a distance $2 \times Nbhd_Radius[i]$ of the node W .

(ii) If $i > 1$ then the operating system at W reassigns the values of $Nbhd_Center[1] \dots Nbhd_Center[i-1]$ to be W .

We observe in SNP that the source of the routing update broadcast is the current location of the process and nodes in the proximity of the process know the correct location of the process. Thus in SNP(DB) boundary effects such as those observed in SNP are not present.

In SNP(DB) the radius of the update broadcast after every move is twice the radius of the broadcast in SNP. The reason for this is as follows. In Figure 4.6 if process q migrates to a location within N_1 and as a consequence broadcasts routing entries for q only to nodes that are within a distance r_1 instead of $2r_1$ from the new location of q , then not all the nodes in the neighborhood N_1 of process q will receive a routing update regarding the latest location of q . However the routing entry in the node at the center of the neighborhood will always point to the latest location of q , assuming updates are instantaneous. In the resulting modified scheme, messages from processes that are located at nodes in the neighborhood N_1 of process q may in the

worst case be forwarded a large number of times before being delivered to the destination process. Hence the radii of broadcast is chosen to be $2r_1$ instead of r_1 .

The update broadcast radius specified by us for SNP(DB) is conservative and in actual implementations of SNP the radius of broadcasts can be reduced appropriately depending on the new location of the process in order to just cover the neighborhood.

The maximum as well as average message trajectory stretch ratios for SNP(DB) are smaller than for SNP as will be observed from the simulations of Section 4.7. However the tradeoff is that the resulting update bandwidth for SNP(DB) is higher than for SNP. If the number of levels and the radii of the neighborhoods are chosen to be the same for SNP and SNP(DB) then the total update bandwidth for SNP(DB) could be up to four times the bandwidth due to SNP. However in practice, as we will observe in Section 4.6.1, the radii of neighborhoods chosen for the two protocols are different and hence the difference in the update bandwidths due to the two schemes is much smaller.

4.4 Shifting Neighborhood Protocol with Adaptive Routing

An important property of the Shifting Neighborhood Protocol and its other variations is that routing entries for a process are more up-to-date at nodes near the location of the process than at distant nodes. We take advantage of this property to adaptively route a message towards a process.

When a process p at node U has to send a message to process q then the Mechanics of Routing and Migration Level of the operating system at U obtains a routing entry for q indicating that q was located at node V with the migration-count of α_1 . In SNP and the other protocols discussed in this chapter the text of the message along with the destination process q and its last known location V are passed to the Low Level Kernel at node U . The Low Level Kernel passes the text of the message and its destination process name and node address to an adjacent node. The message is routed by the Low Level Kernel of the intermediate nodes till the message reaches node V where it is passed up to the next higher level, viz., the Mechanics Level which either re-routes the message or delivers it to the process.

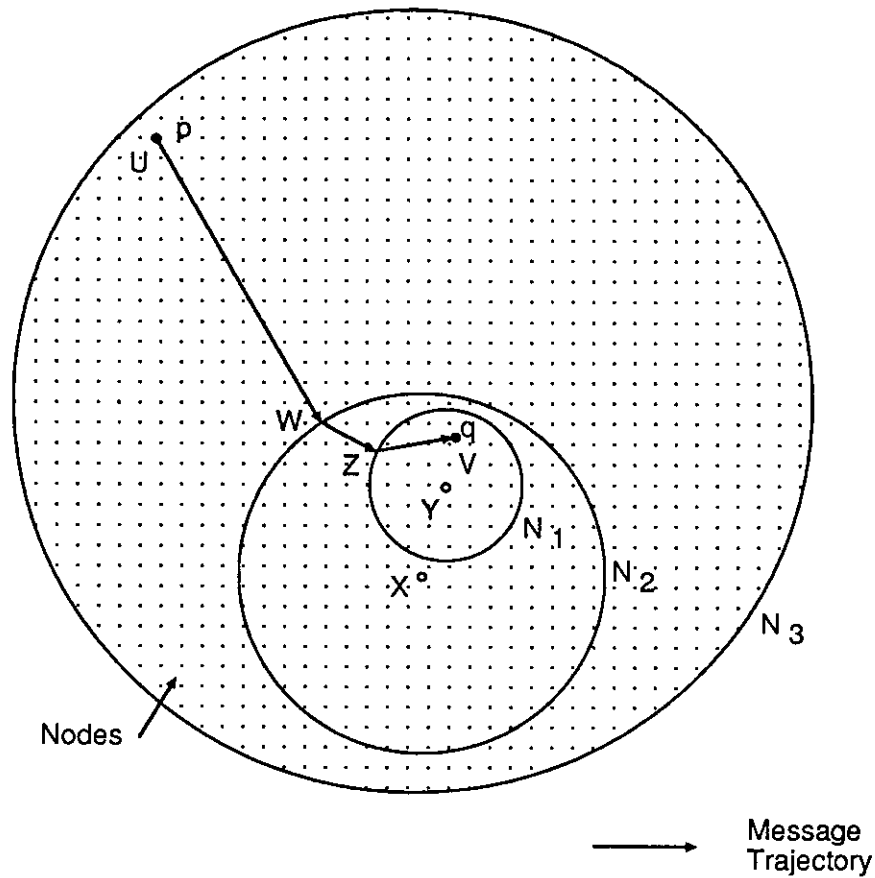


Figure 4.7 : Adaptive Routing of a Message to Process q in SNP(AR)

In the *adaptive message routing* scheme the migration-count α_1 is included in the message along with the destination process name q , its location V and the text of the message. The message that is routed from U to V is passed to the Mechanics Level at *select* intermediate nodes on the path of the message. At the Mechanics Level of an intermediate node the migration-count α_2 of the local routing entry for q is compared with migration-count α_1 in the message. If $\alpha_2 > \alpha_1$ then the message is re-routed to the new node location contained in the routing entry for q at the intermediate node.

The adaptive routing scheme can be applied to any of the three protocols discussed in this chapter. The Shifting Neighborhood Protocol with Adaptive Routing (SNP(AR)) consists of the SNP protocol for broadcasting routing updates and the adaptive routing technique for routing messages to a process. Figure 4.7 illustrates the

working of adaptive routing in the 3-level SNP(AR). A message is sent from process p at node U to q . The neighborhoods of q are $\{N_1, N_2, N_3\}$. At node U in N_3 the message for q is routed to node X the center of N_2 . However at node W , the first intermediate node in the path of the message that lies in N_2 , the message is re-routed to node Y the center of N_1 . All nodes in N_2 , and hence node W , have routing information indicating that the center of the neighborhood N_1 of process q is node Y . At node Z , the first intermediate node in the path of the message from W to Y that lies in N_1 , the message is re-routed to node V where the process is located. Thus the message trajectory is significantly reduced with adaptive routing.

An alternative to routing messages towards the center of a lower level neighborhood of the destination process is to route the message to the closest node in the lower level neighborhood. In the topology of Figure 4.7 the two schemes are equivalent but for many topologies like, for example, the toroidal mesh, routing in the two schemes is different. The reduction in the length of the paths taken by messages will be even larger when messages are routed to the closest node of the lowest level neighborhood instead of the center of the lowest level neighborhood.

The select intermediate nodes at which messages are passed up to the Mechanics level are chosen to be the nodes at the boundaries of the neighborhoods of the destination process. These nodes are the first in the message trajectory that are likely to have more recent information about the location of the process. In order to be able to route messages to the nodes on the boundaries of the neighborhood of a process, information on the radii of the neighborhoods of a process has to be stored along with the location information and migration-count in the routing entry.

By using adaptive routing the message trajectory lengths and hence message trajectory stretch ratios are substantially reduced. The additional cost however is in the additional time for a message to go up to the Mechanics Level of the operating system at select intermediate nodes. In Section 4.7 we compare the time stretch ratio of SNP and SNP(AR) and will observe that the time stretch ratio for SNP(AR) is also significantly lower than for SNP.

4.5 Moving Home Node Protocol

A protocol that is traditionally used to route messages or remote calls to moving objects is the *Home Node Protocol*. Associated with each process is a *home node*, usually the node where the process was created, to which all messages for the process are routed. The home node forwards messages for the process to its current location. Every time a process moves the home node is sent an update indicating the new location of the process.

The Home Node Protocol is used in the Locus [Popek 85] and Sprite [Douglass 87 & Ouster 88] operating systems to route remote calls to moving processes. One of the problems with the Home Node Protocol is that it is not scalable and the stretch ratio degrades with the increase in the lifetime of a process. In the course of time a process could move far away from its home node, and hence the additional path length for all messages to that process will be large even though the actual distance between the process that originates a message and the destination process may be very small.

We propose a modification of the Home Node Protocol based on the mechanisms used in SNP, called the the Moving Home Node Protocol (MHNP) The operation of the 3-level MHNP is shown in Figure 4.8. The process q is initially located at node H_1 and all routing entries for q in the system are initialized to indicate that q is at H_1 . The neighborhoods of q are $\{N_1, N_2, N_3\}$. If q moves to a location within N_1 (Figure 4.8a) then a routing update is sent only to the home node. The home node is the center of N_1 , and has a routing entry that indicates the current location and migration-count of q . In Figure 4.8b when q moves to a new location that is outside N_1 but in N_2 , routing updates are broadcast to all nodes in N_2 . Just as in SNP the neighborhood N_1 is reconstituted to consist of those nodes in N_2 that are within distance r_1 of the new location of the process. The new home node H_2 for process q is the current location of the process.

Thus in MHNP as long as the the process migrates to a location in the lowest level neighborhood of the process the updating protocol is the same as the Home Node Protocol. However if the process moves to a higher level neighborhood then the updating protocol is the same as SNP.

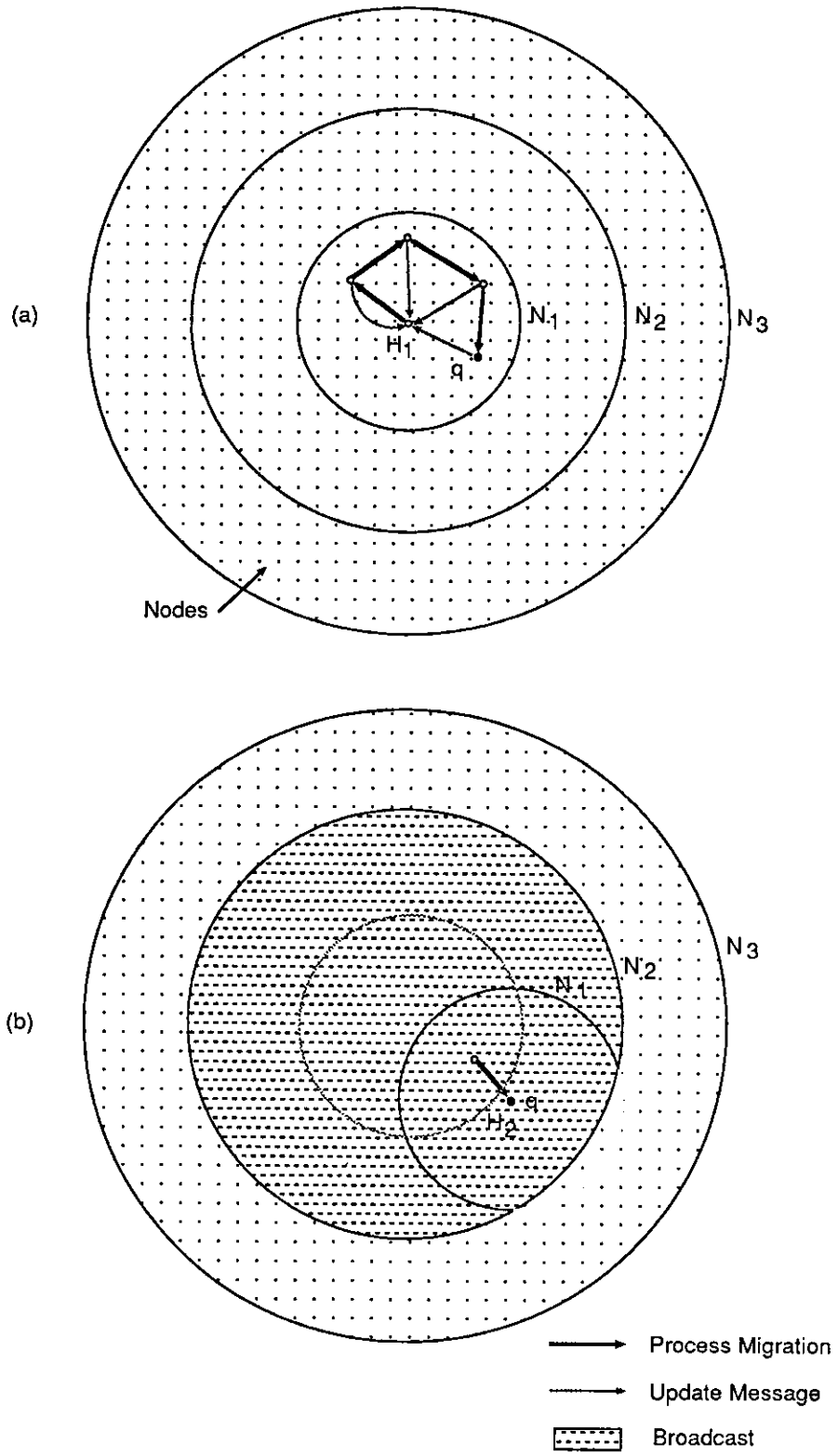


Figure 4.8 : Process Migration in MHNP

The m -level MHNP is informally described below. We assume that array $Nbhd_Radius$ contains the radii of neighborhoods corresponding to levels 1 to m .

1. Initially when a process q is created at a node H_1 the routing entry for q indicating its location and migration-count is broadcast to all the nodes in the system. $Nbhd_Center[1]$, $Nbhd_Center[2]$, ..., $Nbhd_Center[m-1]$ are all initialized to H_1 .
2. When process q moves from node V to W , the operating system at W determines the maximum j ($1 \leq j \leq m-1$) such that $Distance(W, Nbhd_Center[j]) \leq Nbhd_Radius[j]$. If there is no j for which the above inequality is true then we set j equal to zero. Let $i := j + 1$.

(i) If $i = 1$ then the operating system at node W sends an UPDATE message containing the location and migration-count of q to the node $Nbhd_Center[1]$.

If $i > 1$ then operating system at W sends a REQ_BROADCAST_UPDATE message containing parameter $Nbhd_Radius[i]$ and the location and migration-count of q to the node $Nbhd_Center[i]$.

The operating system at node $Nbhd_Center[i]$ on receiving the REQ_BROADCAST_UPDATE message, updates the local routing entry for q and broadcasts routing updates containing the latest location W and migration-count of q to all nodes within a distance $Nbhd_Radius[i]$ of the node $Nbhd_Center[i]$.

(ii) The operating system at W reassigns the values of $Nbhd_Center[1]$... $Nbhd_Center[i-1]$ to be W .

As we will observe from the simulation results described in Section 4.7 the update bandwidth for MHNP is significantly smaller than SPN because no update messages are broadcast while the process migrates to locations that are within the lowest level neighborhood. However the average stretch ratio for MHNP is greater than the average stretch ratio observed for SNP. All messages originating from a process located at a node in neighborhood N_1 of process q (Figure 4.9) are sent to H_1 and then forwarded to the current location of q . Thus, the number of message

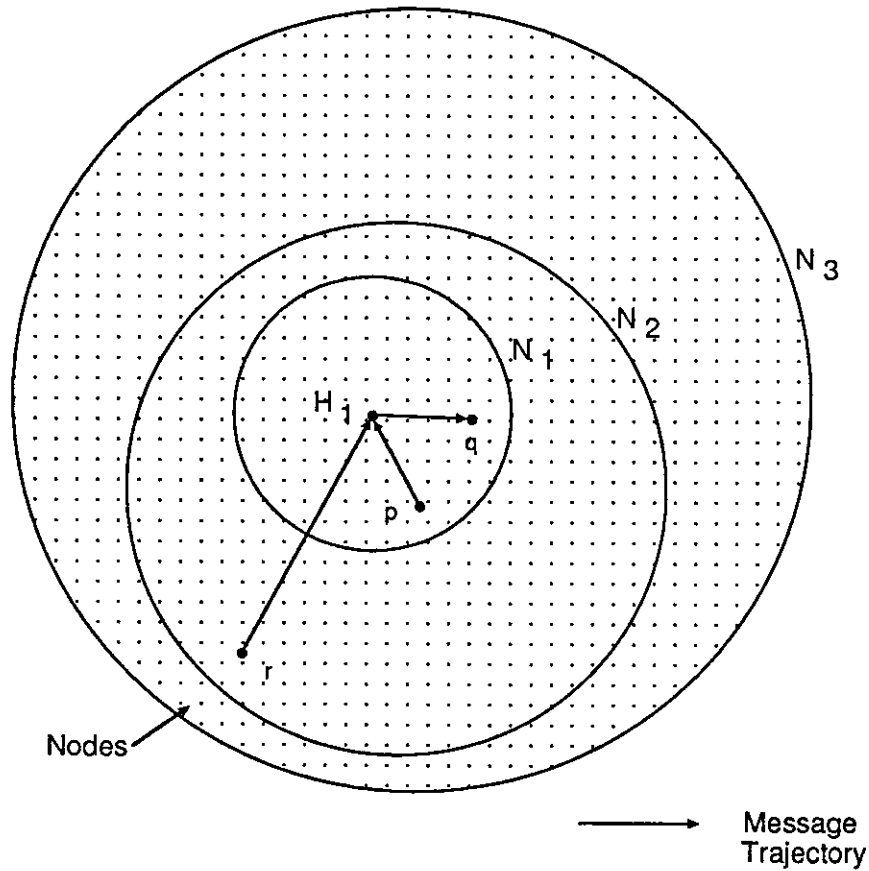


Figure 4.9 : Trajectory of Messages to Process q in MHNP

trajectory hops traversed by a message from a process located in the lowest level neighborhood of the destination process is 2. This is assuming that the time for updates to be delivered is small compared to the time between successive migrations and that the destination process does not migrate while a message is in transit toward it. The maximum stretch ratio for a message originating from a process located in the lowest level neighborhood of the destination process is approximately $2r_1$. However for messages that originate from nodes that are not in the lowest level neighborhood of the destination process the number of times a message is forwarded is the same as in SNP, i.e. the number of times that a message is forwarded is equal to the index of the lowest level neighborhood to which the node, where the process sending the message is resident, belongs.

4.6 Analysis and Performance Evaluation of the Shifting Neighborhood Protocol

In this section we discuss the analysis and evaluation of the Shifting Neighborhood Protocol and related protocols that update routing information in large distributed systems in the presence of process migration. The Shifting Neighborhood Protocol is modeled and the optimal neighborhood structure, i.e. the number of levels and the size of the corresponding neighborhoods, that minimizes the bandwidth for updating routing tables is analytically determined. The update bandwidth of SNP with the optimal neighborhood structure is evaluated and compared to the update bandwidth for the other related protocols. The performance measures other than the update bandwidth that are evaluated are the number of times a message is forwarded and the stretch ratio of messages. The main result from the analysis is that SNP is scalable over a large distributed system. In particular under reasonable assumptions the mean communication bandwidth to update routing entries due to the migration of a single process is $O(\log R)$ message-hops/sec, where R is the radius of the underlying topology.

4.6.1 Determination of the Optimal Neighborhood Structure

For the analysis we choose the 2-dimensional square toroidal mesh as the specific topology underlying the system executing the Shifting Neighborhood Protocol. The 2-dimensional toroidal mesh belongs to the more general category of multidimensional meshes. The advantage of choosing a toroidal mesh structure is that edge effects due to the migration of a process to the boundary of the topology are avoided. Another advantage is that a sub-section of the mesh consisting of a square sub-mesh with boundaries has topological properties similar to the larger square toroidal mesh.

The migration of a process is modeled as an *unrestricted random walk*. The key result from random walk theory that we use is that if a process on every migration moves a distance that is constant or normally distributed and in a direction that is uniformly distributed, then the process will take $\frac{r_i^2}{l^2}$ moves to cross a boundary of radius r_i around the initial position of the process, where l is a constant related to the average size of a move. In Appendix 1 we discuss this result in more detail. We assume this model for the migration of a process in the forthcoming analysis. However sometimes there may be phase changes in the system, and in between phase changes the process motion can not be represented as an unrestricted random walk but rather as motion that is *directed*. In this

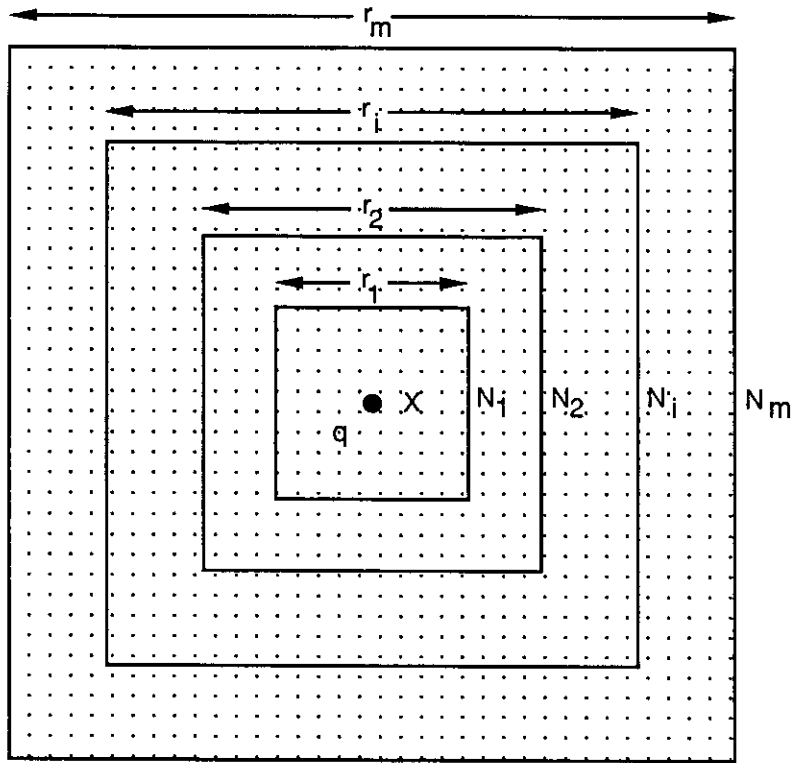


Figure 4.10 : Neighborhoods of a Process in a Square Toroidal Mesh

case a process takes $b \times r_i$ moves to cross a boundary of radius r_i around the initial position of the process, where b is some constant. We also study the effect of directed process motion on the update bandwidth.

In order to minimize the update bandwidth and obtain the optimal neighborhood structure, i.e. the optimal number of levels and sizes of neighborhoods, from the analytical expressions for the update bandwidth, we assume that the number of levels and the dimension of neighborhoods are real valued. Discrete approximations to the optimal values of the number of levels and the dimensions of the corresponding neighborhoods obtained from the solution are used for the actual implementation of SNP.

The notations for the parameters and performance measures for the model of SNP are given in Table 4.1.

C	No. of update-msg-hops/sec due to a single process
SR	Stretch Ratio
N_i	Square neighborhood N_i
m	No. of levels or neighborhood for a process
n_i	No. of nodes in neighborhood N_i
N	No. of nodes in the system
r_i	Dimension (side) of neighborhood N_i
R	Dimension of the entire square toroidal mesh
v	Rate of migration in terms of no. of moves/sec
l	Constant related to the average size of a move

Table 4.1 : Parameters and Performance Measures for the Shifting Neighborhood Protocol

Figure 4.10 illustrates process q initially located at node X of an $R \times R$ toroidal mesh. The neighborhoods $\{N_1, N_2, \dots, N_m\}$ of process q are square meshes, with the corresponding sides of the square mesh $r = \{r_1, r_2, \dots, r_m\}$, where $r_1 < r_2 < r_3 \dots < r_m$. The neighborhood N_i has dimensions $r_i \times r_i$. We will call the node at the intersection of the diagonals of a square mesh its *center*. In an $R \times R$ square toroidal mesh because of the symmetry any node can be considered the center. Note that the neighborhood N_m contains all the nodes in the system. Hence

$$r_m = R$$

$$n_m = N$$

The model of process migration is as follows. Let a process be initially located at the center of an $r_i \times r_i$ neighborhood N_i . We assume that the size of a move is constant or normally distributed. In our model of process migration the process takes $\frac{r_i^2}{l^2}$ moves to cross the boundary of neighborhood N_i , where l is a constant and depends on the average move size. In Appendix 1 we show how this property holds when the migration of the process is modeled as an unrestricted random walk.

From the description and specification of SNP in the previous section we know that every time the process moves to a new location that is within N_1 , updates are broadcast to all nodes in N_1 . Generally, when the process moves and crosses out of the boundary of N_{i-1} to a node in N_i then updates are broadcast to all nodes in N_i .

In a mesh the optimal number of update messages-hops required for broadcasting updates to N_i is equal to the number of nodes in neighborhood N_i minus one i.e. $n_i - 1$. Therefore

For every move of the process n_1 update message-hops are sent.

For every $\frac{r_1^2}{l^2}$ moves of the process, n_2 update message-hops are sent.

For every $\frac{r_{m-1}^2}{l^2}$ moves of the process, n_m update message-hops are sent.

The above estimate is slightly conservative because if we assume that every time a process moves n_1 update message-hops are sent, then for every $\frac{r_1^2}{l^2}$ moves actually only $(n_2 - n_1)$ additional update message-hops will be sent and for every $\frac{r_{i-1}^2}{l^2}$ moves actually only $(n_i - n_{i-1} \cdots - n_2 - n_1)$ update message-hops will be sent. In order to simplify the analysis for the update bandwidth we ignore lower order difference terms and adopt the above accounting procedure for counting update messages. The update bandwidth estimate that we obtain will be slightly higher than the actual update bandwidth. The update bandwidth due to the migration of a single process can be expressed as

$$C = v \left[n_1 + \frac{n_2 l^2}{r_1^2} + \dots + \frac{n_i l^2}{r_{i-1}^2} + \dots + \frac{n_m l^2}{r_{m-1}^2} \right] \text{update_msg-hops/sec} \quad (4.1)$$

From the properties of a square mesh we know $n_i = r_i^2$, and thus

$$C = v \left[r_1^2 + \frac{r_2^2 l^2}{r_1^2} + \dots + \frac{r_i^2 l^2}{r_{i-1}^2} + \dots + \frac{r_m^2 l^2}{r_{m-1}^2} \right] \text{update_msg-hops/sec} \quad (4.2)$$

We now state the first optimization problem as follows:

Minimize $C = C(m, r_1, r_2 \cdots r_m, v)$
 given fixed m
 over $(r_1, r_2 \cdots r_{m-1})$

We determine the point $\mathbf{r} = \mathbf{r}_a$ at which $C(\mathbf{r})$ is a minima by letting

$$\frac{\partial C(\mathbf{r})}{\partial r_i} = 0 \quad \text{for } i = 1, 2, \dots, m-1$$

$$\frac{\partial C}{\partial r_1} = v \left[2r_1 - \frac{2r_2^2 l^2}{r_1^3} \right] = 0$$

$$\Rightarrow r_2 = \frac{r_1^2}{l}$$

$$\frac{\partial C}{\partial r_{i-1}} = v \left[\frac{2r_{i-1} l^2}{r_{i-2}^2} - \frac{2r_i^2 l^2}{r_{i-1}^3} \right] = 0$$

$$\Rightarrow r_i = \frac{r_{i-1}^2}{r_{i-2}} \quad \text{for } i = 3 \dots m$$

By induction we can easily show that

$$r_i = \frac{r_1^i}{l^{i-1}} \quad \text{for } i = 2 \dots m \quad (4.3)$$

From Equation 4.3 we obtain

$$\frac{r_i}{r_{i-1}} = \frac{r_1}{l} \quad \text{for } i > 1 \quad (4.4)$$

and

$$r_1 = r_m \frac{1}{l^m} \frac{1}{l^{m-1}} = R \frac{1}{l^m} \frac{1}{l^{m-1}}$$

$$\therefore r_1 = \left(\frac{R}{l} \right)^{\frac{1}{m}} l \quad (4.5)$$

From Equations 4.3 and 4.5 we obtain the optimal dimension r_i of neighborhood N_i

$$r_i = \left(\frac{R}{l} \right)^{\frac{i}{m}} l \quad \text{for } 1 \leq i \leq m \quad (4.6)$$

In Appendix 2 we show that at the point $\mathbf{r} = \mathbf{r}_a$ the update bandwidth C is a minimum. Thus we have obtained the optimal dimension $r_i = O(R^{1/m})$ of neighborhood N_i , given that there are m neighborhoods. By substituting Equations 4.4 and 4.6 in Equation 4.2 we obtain the update bandwidth

$$C = vmr_1^2 \quad (4.7)$$

$$= vm \left(\frac{R}{l}\right)^{\frac{2}{m}} l^2 \quad \text{update_msg-hops/sec} \quad (4.8)$$

Having obtained an expression for the optimal radii of the neighborhoods given m we now find the optimal number of neighborhoods to minimize the update bandwidth. The optimization problem that we solve is

$$\begin{aligned} &\text{Minimize } C = C(m, r_1, r_2 \cdots r_m, v) \\ &\text{given } (r_1, r_2 \cdots r_m) \text{ as in Equation 4.6} \\ &\text{over } m \end{aligned}$$

In Equation 4.8 let

$$\begin{aligned} \frac{dC}{dm} &= 0 \\ \Rightarrow v \left[\left(\frac{R}{l}\right)^{\frac{2}{m}} l^2 + m \left(\frac{R}{l}\right)^{\frac{2}{m}} l^2 \ln\left(\frac{R}{l}\right) \frac{d}{dm}\left(\frac{2}{m}\right) \right] &= 0 \\ \Rightarrow 1 - \frac{2}{m} \ln\frac{R}{l} &= 0 \\ \Rightarrow m = 2 \ln\frac{R}{l} \end{aligned} \quad (4.9)$$

To verify that $m = 2 \ln\frac{R}{l}$ leads to a minima for C we show below that $\frac{d^2C}{dm^2} > 0$ at $m = 2 \ln\frac{R}{l}$.

$$\begin{aligned} \frac{d^2C}{dm^2} &= \\ vl^2 \left[\left[\left(\frac{R}{l}\right)^{\frac{2}{m}} \ln\left(\frac{R}{l}\right) \frac{d}{dm}\left(\frac{2}{m}\right) \right] \left[1 - \frac{2}{m} \ln\frac{R}{l} \right] - \left(\frac{R}{l}\right)^{\frac{2}{m}} \frac{d}{dm}\left(\frac{2}{m}\right) \ln\left(\frac{R}{l}\right) \right] \end{aligned}$$

$$\begin{aligned}
&= vl^2 \left(\frac{R}{l}\right)^{\frac{2}{m}} \ln\left(\frac{R}{l}\right) \left[-\frac{2}{m^2} \left[1 - \frac{2}{m} \ln\frac{R}{l} \right] + \frac{2}{m^2} \right] \\
&= \frac{4vl^2 \left(\frac{R}{l}\right)^{\frac{2}{m}} \left[\ln\left(\frac{R}{l}\right) \right]^2}{m^3} > 0
\end{aligned}$$

Thus we determine that the optimal number of levels in the SNP protocol is $O(\log R)$. The update bandwidth for the optimal neighborhood structure is given by

$$\begin{aligned}
C &= 2v \ln\frac{R}{l} r_1^2 \\
&= 2v \ln\frac{R}{l} \left(\frac{R}{l}\right)^{\frac{1}{\ln\frac{R}{l}}} l^2
\end{aligned}$$

Since $e^{\ln\frac{R}{l}} = \frac{R}{l}$, where $e = 2.718\dots$, we have

$$C = 2ve l^2 \ln\frac{R}{l} \text{ update_msg-hops/sec} \quad (4.10)$$

The total update communication bandwidth due to P processes in the system is

$$2veP l^2 \ln\left(\frac{R}{l}\right) \text{ update_msg-hops/sec} \quad (4.11)$$

We now compare the update bandwidth due to the migration of a single process for SNP and the CBM protocol where there is a complete broadcast on every move. As we noted in Section 2.4, for BRM with complete broadcast on every move the update bandwidth is

$$C_{CBM} = O(N) = O(R^2) \text{ update_msg-hops/sec}$$

However for SNP, from Equation 4.10 we have

$$C_{SNP} = O(\log R) \text{ update_msg-hops/sec}$$

The update bandwidth for SNP is a vast improvement over the update bandwidth for CBM. For the Shifting Neighborhood Protocol executing on a square toroidal mesh, the

bandwidth overhead resulting from the updating of routing tables due to process migration has only a logarithmic dependency on the dimension of the system. Since the logarithmic function of the dimension of the system is a very slowly increasing function of the size of the system, hence we conclude that the Shifting Neighborhood Protocol is scalable with the size of the system.

As we noted earlier our estimate of the update bandwidth is slightly conservative because in order to simplify the expression for the update bandwidth C we drop some lower order terms. The exact expression for C is given by

$$C = v \left[n_1 + \frac{(n_2 - n_1)l^2}{r_1^2} + \frac{(n_3 - n_2 - n_1)l^2}{r_2^2} + \dots + \frac{(n_i - n_{i-1} \dots - n_2 - n_1)l^2}{r_{i-1}^2} + \dots + \frac{(n_m - n_{m-1} \dots - n_2 - n_1)l^2}{r_{m-1}^2} \right] \quad (4.12)$$

$$= v \left[r_1^2 + \sum_{i=2}^m \left[\frac{r_i^2 - \sum_{j=1}^{i-1} r_j^2}{r_{i-1}^2} \right] \right] \text{ update_msg-hops/sec} \quad (4.13)$$

The analysis of the optimal neighborhood structure and the corresponding update bandwidth for the Shifting Neighborhood Protocol with Adaptive Routing SNP(AR) is identical to the analysis of the Shifting Neighborhood Protocol because the difference between the two protocols is in the routing of messages and not in the update of routing entries when a process migrates.

Optimal Neighborhood Structure for SNP(DB)

The analysis of the Shifting Neighborhood Protocol with Dynamic Broadcast follows along the lines of the analysis of SNP and therefore we will provide only an outline of the analysis of SNP(DB).

Let us associate neighborhoods $\{N_1, N_2, \dots, N_m\}$ with process q and let q be initially located at node X of an $R \times R$ toroidal mesh as illustrated in Figure 4.10. The neighborhood N_i is a square mesh with side r_i . The updating mechanism for the Shifting Neighborhood Protocol with Dynamic Broadcast is specified as follows. Every time the

process moves to a location that is within N_1 , updates are broadcast to all nodes within distance $2r_1$ from the new location of the process. When the process moves and crosses the boundary of N_1 to a node that is outside N_1 but still in N_2 then updates are broadcast to all nodes within distance $2r_2$ from the new location of the process. Similarly when the process moves and crosses the boundary of N_{i-1} to a node that is outside N_{i-1} but still in N_i then updates are broadcast to all nodes within distance $2r_i$ from the new location of the process. Finally when the process moves and crosses the boundary of N_{m-1} to a node outside N_{m-1} but still in N_m then updates are broadcast to all nodes in N_m .

In a square toroidal mesh the number of update message-hops required for broadcasting updates to all nodes within distance $2r_i$ from the source is approximately equal to $4r_i^2$. Note however that when the process crosses the boundary of its neighborhood N_{m-1} , it is sufficient to broadcast messages to r_i^2 nodes that will cover the entire system. Therefore

For every move of the process $4r_1^2$ update message-hops are sent.

For every $\frac{r_i^2}{l^2}$ moves of the process, $4r_{i+1}^2$ update message-hops are sent, where $i = 1, 2 \dots m-2$

For every $\frac{r_{m-1}^2}{l^2}$ moves of the process, r_m^2 update message-hops are sent.

The above estimate as in the analysis of SNP is slightly conservative because a few updates are accounted for twice in order to simplify the analysis. The update bandwidth due to the migration of a single process for SNP(DB) can be expressed as

$$C = 4v \left[r_1^2 + \frac{r_2^2 l^2}{r_1^2} + \dots + \frac{r_i^2 l^2}{r_{i-1}^2} + \dots + \frac{r_m^2 l^2}{4r_{m-1}^2} \right] \text{ update_msg-hops/sec} \quad (4.14)$$

To determine the point $\mathbf{r} = \mathbf{r}_a$ at which $C(\mathbf{r})$ is an minima we keep m fixed and let

$$\frac{\partial C(\mathbf{r})}{\partial r_i} = 0 \quad \text{for } i = 1, 2, \dots, m-1$$

For $i = 2, 3 \dots m-1$

$$\frac{\partial C}{\partial r_{i-1}} = 0 \Rightarrow r_i = \frac{r_1^i}{l^{i-1}} \quad (4.15)$$

$$\frac{\partial C}{\partial r_{m-1}} = 0 \Rightarrow r_m = \frac{2r_{m-1}^2}{r_{m-2}} = \frac{2r_1^m}{l^{m-1}}$$

Since $r_m = R$ we have

$$r_1 = \left(\frac{R}{2l}\right)^{\frac{1}{m}} l \quad (4.16)$$

From Equations 4.15 and 4.16 we obtain

$$r_i = \left(\frac{R}{2l}\right)^{\frac{i}{m}} l \quad \text{for } i = 1, 2, \dots, m-1 \quad (4.17)$$

and

$$r_m = R$$

Similarly as in Appendix 2 we can show that at the point $\mathbf{r} = \mathbf{r}_a$ the update bandwidth C is a minimum. Thus if the number of levels m is fixed, the optimal dimension r_i of neighborhood N_i for SNP(DB) is $\left(\frac{1}{2}\right)^{\frac{i}{m}}$ of the dimension of the corresponding neighborhood in SNP, where i ranges from 1 to $m-1$. By substituting Equation 4.16 in Equation 4.14 we obtain the update bandwidth for SNP(DB) as

$$C = 4vmr_1^2 \quad (4.18)$$

$$= 4vm \left(\frac{R}{2l}\right)^{\frac{2}{m}} l^2 \quad \text{update_msg-hops/sec} \quad (4.19)$$

Now that we have obtained an expression for the optimal radii of the neighborhoods of a process in SNP(DB), we can find the optimal number of neighborhoods that minimizes the update bandwidth.

In Equation 4.16 let

$$\begin{aligned} \frac{dC}{dm} &= 0 \\ \Rightarrow 4v \left(\left(\frac{R}{2l}\right)^{\frac{2}{m}} l^2 + m \left(\frac{R}{2l}\right)^{\frac{2}{m}} l^2 \ln\left(\frac{R}{2l}\right) \frac{d}{dm}\left(\frac{2}{m}\right) \right) &= 0 \\ \Rightarrow 1 - \frac{2}{m} \ln\frac{R}{2l} &= 0 \\ \Rightarrow m &= 2 \ln\frac{R}{2l} \end{aligned} \quad (4.20)$$

To verify that $m = 2 \ln \frac{R}{2l}$ leads to a minimum for C we can show that $\frac{d^2C}{dm^2} > 0$ at $m = 2 \ln \frac{R}{2l}$.

The update bandwidth for SNP(DB) when the optimal neighborhood structure is chosen is given by

$$\begin{aligned} C &= 8v \ln \frac{R}{2l} r_1^2 \\ &= 8v \ln \frac{R}{2l} \left(\frac{R}{2l} \right)^{\frac{1}{\ln \frac{R}{2l}}} l^2 \end{aligned}$$

$$\therefore C = 8ve l^2 \ln \frac{R}{2l} \text{ update_msg-hops/sec} \quad (4.21)$$

The total update communication bandwidth for SNP(DB) due to P processes in the system is

$$8veP l^2 \ln \left(\frac{R}{2l} \right) \text{ update_msg-hops/sec} \quad (4.22)$$

The exact bandwidth required to update routing tables on the migration of a single process is given by

$$C = 4v \left[r_1^2 + \sum_{i=2}^{m-1} \left[\frac{r_i^2 - \sum_{j=1}^{i-1} r_j^2}{r_{i-1}^2} \right] + \frac{r_m^2 - \sum_{j=1}^{m-1} r_j^2}{4r_{m-1}^2} \right] \text{ update_msg-hops/sec} \quad (4.23)$$

Optimal Neighborhood Structure for MHNP

As before we consider the situation illustrated in Figure 4.10. The updating mechanism for the Moving Home Node Protocol is specified as follows. Every time the process moves to a location that is within N_1 , 1 update message is sent. When the process moves to a location that is outside N_1 but in N_2 then updates are broadcast to all nodes in N_2 . Similarly when the process moves to a location outside N_{m-1} then updates are broadcast to all nodes in N_m .

The update bandwidth due to the migration of a single process can be expressed as

$$\begin{aligned}
C &= v \left[1 + \frac{n_2 l^2}{r_1^2} + \dots + \frac{n_i l^2}{r_{i-1}^2} + \dots + \frac{n_m l^2}{r_{m-1}^2} \right] \\
&= v \left[1 + \frac{r_2^2 l^2}{r_1^2} + \dots + \frac{r_i^2 l^2}{r_{i-1}^2} + \dots + \frac{r_m^2 l^2}{r_{m-1}^2} \right] \text{ update_msg-hops/sec} \quad (4.24)
\end{aligned}$$

In MHNP the updating mechanism for migration to a location within the lowest level neighborhood is different from the updating mechanism when a process crosses neighborhood boundaries. When the process migrates to a location within the lowest level neighborhood, then no broadcast of updates takes place, but instead a single update message is sent to the home node. In Equation 4.24 we observe that the update bandwidth C as a function of r_1 is not concave, but decreases as r_1 approaches r_2 . Therefore the dimension of the lowest level neighborhood r_1 is chosen based on the worst case stretch ratio that can be tolerated. Here we assume r_1 to be a constant.

To determine the point at which the update bandwidth is a minimum, we let

$$\begin{aligned}
\frac{\partial C}{\partial r_i} &= 0 \quad \text{for } i = 2, \dots, m-1 \\
\frac{\partial C}{\partial r_2} &= v \left[\frac{2r_2 l^2}{r_1^2} - \frac{2r_3^2 l^2}{r_2^3} \right] = 0 \\
\Rightarrow r_3 &= \frac{r_2^2}{r_1}
\end{aligned}$$

For $i = 3, 4, \dots, m$ we have

$$\begin{aligned}
\frac{\partial C}{\partial r_{i-1}} &= v \left[\frac{2r_{i-1} l^2}{r_{i-2}^2} - \frac{2r_i^2 l^2}{r_{i-1}^3} \right] = 0 \\
\Rightarrow r_i &= \frac{r_{i-1}^2}{r_{i-2}} = \frac{r_2^{i-1}}{r_1^{i-2}} \quad (4.25)
\end{aligned}$$

From Equation 4.25 for $i = 2, 3, \dots, m$ we obtain

$$\frac{r_i}{r_{i-1}} = \frac{r_2}{r_1} \quad (4.26)$$

and

$$r_2 = \left(\frac{R}{r_1}\right)^{\frac{1}{m-1}} r_1 \quad (4.27)$$

From Equations 4.25 and 4.27 we obtain the optimal dimension r_i of neighborhood N_i

$$r_i = \left(\frac{R}{r_1}\right)^{\frac{i-1}{m-1}} r_1 \quad \text{for } i = 1, 2, \dots, m \quad (4.28)$$

As in Appendix 2 we can similarly show that at the inflection point the update bandwidth C is a minimum.

By substituting Equations 4.28 in Equation 4.24 we obtain the update bandwidth

$$\begin{aligned} C &= v \left(1 + \frac{r_2^2 l^2 (m-1)}{r_1^2} \right) \\ &= v \left(1 + (m-1) \left(\frac{R}{r_1}\right)^{\frac{2}{m-1}} l^2 \right) \end{aligned} \quad (4.29)$$

We have obtained an expression for the update bandwidth due to process migration in the Moving Home Node Protocol given a fixed number of levels or neighborhoods in the protocol. Now if we let m the number of levels vary then we can determine the optimal number of neighborhoods to minimize the update bandwidth. In Equation 4.29 let

$$\begin{aligned} \frac{dC}{dm} &= 0 \\ \Rightarrow v \left[\left(\frac{R}{r_1}\right)^{\frac{2}{m-1}} l^2 + (m-1) \left(\frac{R}{r_1}\right)^{\frac{2}{m-1}} l^2 \ln\left(\frac{R}{r_1}\right) \frac{d}{dm} \left(\frac{2}{m-1}\right) \right] &= 0 \\ \Rightarrow 1 - \frac{2}{m-1} \ln \frac{R}{r_1} &= 0 \\ \Rightarrow m = 1 + 2 \ln \frac{R}{r_1} \end{aligned} \quad (4.30)$$

As before we can show that $\frac{d^2C}{dm^2} > 0$ at $m = 1 + 2 \ln \frac{R}{r_1}$ implying that C is minimum at this point.

Thus we have obtained the optimal neighborhood structure and the optimal number of levels for MHNP. The update bandwidth for MHNP when the optimal neighborhood structure is chosen is given by

$$C = v \left(1 + 2 \ln \frac{R}{r_1} \left(\frac{R}{r_1} \right)^{\frac{1}{\ln \frac{R}{r_1}}} l^2 \right)$$

$$\therefore C = v \left(1 + 2 e l^2 \ln \frac{R}{r_1} \right) \text{ update_msg-hops/sec} \quad (4.31)$$

Protocol	Optimal Neighborhood Dimensions (r_i)	Optimal No. of Levels (m)
SNP and SNP(AR)	$r_i = \left(\frac{R}{l} \right)^{\frac{i}{m}} l$ for $1 \leq i \leq m$	$2 \ln \frac{R}{l}$
SNP(DB)	$r_i = \left(\frac{R}{2l} \right)^{\frac{i}{m}} l$ for $1 \leq i \leq m-1$	$2 \ln \frac{R}{2l}$
MHNP	$r_i = \left(\frac{R}{r_1} \right)^{\frac{i-1}{m-1}} r_1$ for $1 \leq i \leq m$	$1 + 2 \ln \frac{R}{r_1}$

Table 4.2 : Optimal Neighborhood Structure for SNP and related protocols

The total update communication bandwidth for MHNP due to P processes in the system is

$$vP \left(1 + 2 e l^2 \ln \frac{R}{r_1} \right) \text{ update_msg-hops/sec} \quad (4.32)$$

The exact bandwidth required to update routing tables on the migration of a single process is given by

$$C = v \left[1 + \sum_{i=2}^m \left[\frac{r_i^2 - \sum_{j=1}^{i-1} r_j^2}{r_{i-1}^2} \right] \right] \text{ update_msg-hops/sec} \quad (4.33)$$

Tables 4.2 and 4.3 summarize our results for the optimal size of neighborhoods, the optimal number of neighborhoods, the average update bandwidth when the number of levels m is given and the neighborhood sizes are chosen to be optimal, and the average update bandwidth when an optimal neighborhood structure, i.e. optimal neighborhood sizes and optimal number of neighborhoods, is chosen. The constant l is related to the average size of a process move. The update bandwidth is given in terms of the number of update_msg-hops/sec. The update bandwidth given in Table 4.3 is due to the migration of a single process.

Protocol	Update Bandwidth Optimal r_i Fixed m	Update Bandwidth Optimal r_i Optimal m
SNP and SNP(AR)	$v m \left(\frac{R}{l}\right)^{\frac{2}{m}} l^2$	$2 v e l^2 \ln \frac{R}{l}$
SNP(DB)	$4 v m \left(\frac{R}{2l}\right)^{\frac{2}{m}} l^2$	$8 v e l^2 \ln \frac{R}{2l}$
MHNP	$v [1 + (m-1) \left(\frac{R}{r_1}\right)^{\frac{2}{m-1}} l^2]$	$v [1 + 2 e l^2 \ln \frac{R}{r_1}]$

Table 4.3 : Update Bandwidth (update_msg-hops/sec) for SNP and related protocols

The size of the neighborhoods and the number of levels in the expressions given in the Tables are real valued and therefore actual implementations have to use discrete approximations to the values obtained. For SNP(DB) note that the expression for r_i is not applicable when i is equal to m and hence can not be used to obtain $r_m = R$. The reason for this is that in SNP(DB) the broadcast of updates to nodes in the m th level neighborhood is done differently than to nodes in lower level neighborhoods. Note that for MHNP the value of r_1 is chosen based on other criterion with the constraint that $r_1 < r_2$.

From Table 4.2 we observe that the optimal number of levels for SNP and SNP(AR) is $2 \ln \frac{R}{l}$ which is $O(\log R)$. For SNP(DB) the optimal number of levels is $2 \ln \frac{R}{2l}$ which is $2 \ln 2$ or 1.39 less than the optimal number of levels for SNP and SNP(AR). The optimal dimension of neighborhood N_i for a m -level SNP or SNP(AR) protocol is $O(R^{\frac{i}{m}})$. If the number of levels m is fixed, the optimal dimension r_i of neighborhood N_i for SNP(DB) is $(\frac{1}{2})^{\frac{i}{m}}$ of the dimension of the corresponding neighborhood in SNP, where i ranges from 1 to $m-1$. The behavior of MHNP is similar to SNP except at the lowest level of the protocol where MHNP behaves differently. The expressions for the optimal neighborhood structure and update bandwidth for MHNP indicate the similarity between MHNP and SNP for all levels of the protocol except the lowest level.

The update bandwidth for SNP and related protocols (Table 4.3) due to the migration of a single process is $O(\log R)$ update_msg-hops/sec when the rate of process migration is fixed, which is a vast improvement over the update bandwidth of $O(R^2)$ update_msg-hops/sec for the protocol with Complete Broadcast on Migration. Note that the update bandwidth for SNP and related protocols is proportional to the rate of process migration. Observe that the update bandwidth for SNP(DB) with an optimal neighborhood structure is $(4 - \frac{4 \ln 2}{\ln(R/l)})$ times the update bandwidth for SNP with an optimal neighborhood structure. The increased update bandwidth for SNP(DB) is accounted for by the fact that when a process crosses the boundary of neighborhood N_{i-1} in SNP(DB), where $i < m$, there is a broadcast to a square mesh of side $2r_i$ while in SNP the broadcast is to a square mesh of side r_i . The reason that the update bandwidth for SNP(DB) is less than four times that of SNP is because the optimal neighborhood

structures for the two protocols are different and in SNP(DB) when the process crosses its neighborhood N_{m-1} a broadcast of size $2r_m \times 2r_m$ is not necessary since a broadcast of size $r_m \times r_m$ is sufficient to cover the entire system.

Effect of Directed Motion of a Process between Phase Changes on Update Bandwidth

Let us consider that the Shifting Neighborhood Protocol with an optimal neighborhood structure is chosen for processes executing on a square toroidal mesh. The optimal neighborhood structure was derived earlier with the assumption that process migration can be modeled as an unrestricted random walk. Let us now assume that between phase changes the migration of a process is no longer an unrestricted random walk but is directed in a certain direction. We examine the effect in SNP of this change in the motion of a process on the update bandwidth.

In the case of directed process motion a process takes $b \times r_i$ moves to cross the boundary of a square mesh of side r_i centered at the initial location of the process, where b is a constant related to the average size of a move. Therefore for the period between phase changes we can express the update bandwidth as

$$C = v \left[r_1^2 + \frac{r_2^2}{r_1 b} + \dots + \frac{r_i^2}{r_{i-1} b} + \dots + \frac{r_m^2}{r_{m-1} b} \right] \text{ update_msg-hops/sec (4.34)}$$

where from Equations 4.5 and 4.6 we have

$$r_1 = \left(\frac{R}{l}\right)^{\frac{1}{m}} l$$

and $r_i = \left(\frac{R}{l}\right)^{\frac{i}{m}} l$ for $1 \leq i \leq m$

For $2 \leq i \leq m$ we obtain

$$\frac{r_i^2}{r_{i-1}} = \left(\frac{R}{l}\right)^{\frac{i+1}{m}} l$$

We therefore have the update bandwidth

$$\begin{aligned}
C &= v \left[\left(\frac{R}{l}\right)^{\frac{2}{m}} l^2 + \frac{l}{b} \sum_{i=2}^m \left(\frac{R}{l}\right)^{\frac{i+1}{m}} \right] \\
&= v \left[\left(\frac{R}{l}\right)^{\frac{2}{m}} l^2 + \frac{l}{b} \left(\frac{R}{l}\right)^{\frac{1}{m}} \left(\sum_{i=0}^m \left(\frac{R}{l}\right)^{\frac{i}{m}} - 1 - \frac{R}{l} \right) \right] \\
&= v \left[\left(\frac{R}{l}\right)^{2/m} l^2 + \frac{l}{b} \left(\frac{R}{l}\right)^{1/m} \left[\frac{\left(\frac{R}{l}\right)^{(m+1)/m} - 1}{\left(\frac{R}{l}\right)^{1/m} - 1} - 1 - \frac{R}{l} \right] \right]
\end{aligned}$$

From Equation 4.9 we have $m = 2 \ln \frac{R}{l}$ and hence

$$\left(\frac{R}{l}\right)^{\frac{1}{m}} = e^{\frac{1}{2}}, \text{ where } e = 2.718\dots$$

$$\therefore C = v \left[e l^2 + \frac{l}{b} \sqrt{e} \left(\frac{\left(\frac{R}{l}\right)^{\sqrt{e}} - 1}{\sqrt{e} - 1} - 1 - \frac{R}{l} \right) \right] \text{ update_msg-hops/sec} \quad (4.35)$$

Hence the update bandwidth for the duration between the phase shift in the migration of a single process is $O(Rv)$, where the neighborhood structure for SNP is chosen by optimizing the update bandwidth assuming process migration can be modeled as an unrestricted random walk. Thus the update bandwidth increases from $O(\log(R))$ to $O(R)$ during a phase change, assuming the rate of process migration is fixed. If the typical mode of the migration of a particular process is characterized by directed motion and not an unrestricted random walk then the neighborhood structure to be used in SNP can be chosen by minimizing the update bandwidth assuming the process motion is modeled as directed motion. The analysis for the choice of the optimal neighborhood structure assuming directed process motion would then follow along the lines of the analysis for SNP for unrestricted random walk.

4.6.2 Stretch Ratios for the Shifting Neighborhood Protocol

In the previous section we obtained the optimal neighborhood structure for the Shifting Neighborhood Protocol and other related protocols. The optimal neighborhood structure that consists of the optimal size of neighborhoods and the number of neighborhoods is derived by minimizing the update bandwidth resulting from updating routing entries due to process migration. In this section we study the efficiency of the routing schemes in terms of the increase in the path length of messages compared to the shortest distance between the source and the destination of the messages. In particular we determine the Message Trajectory Stretch Ratio for messages averaged for all possible source process and destination process locations. We will show that despite the large reductions in update bandwidth observed for SNP over the protocol with Complete Broadcast on Migration, the increase in the stretch ratio for SNP is very small and the stretch ratio is close to one.

The main assumption we make for the evaluation of the average message trajectory length is that a process sends a message to any process in the system with equal probability. This is the *uniform probability model* of communication between processes. This is in contrast to the *spatial locality model* where a process is more likely to send a message to a nearby process than to a distant process. The reason for assuming the uniform probability model is because it simplifies the analysis of the message trajectory stretch ratio in the Shifting Neighborhood Protocol. In fact the stretch ratio for the Shifting Neighborhood Protocol will be even smaller when there is spatial locality in the communication between processes. In the simulation of SNP and related protocols that follows in the next section, we study the performance of the protocols assuming the spatial locality model.

To calculate the stretch ratio for a particular message we assume that the destination process does not move while the message is enroute to it. In addition the time for updates to be broadcast is assumed to be small in comparison to the time between successive migrations of a process.

Let

$\overline{h_r}$ = The length of the trajectory of a message in hops averaged over all possible source process and destination process locations

\bar{h} = The shortest distance in hops between the location of any two processes averaged over all possible process locations

We calculate the the Message Trajectory Stretch Ratio (MTSR) for a particular protocol as

$$MTSR = \frac{\bar{h}_r}{\bar{h}}$$

We define the relative increase in the path length (*RPL*) as

$$RPL = \frac{\bar{h}_r}{\bar{h}} - 1$$

The topology chosen for this analysis is a $p \times p$ square toroidal mesh and without loss of generality we assume that p is odd. In Appendix 3 to 7, we derive the following properties of a square mesh:

- The average distance between any two random distinct nodes in a $q \times q$ mesh is $\frac{2q}{3}$ (4.36)
- The average distance between the center of a $q \times q$ mesh and any other random node in the mesh is $\frac{q}{2}$ (4.37)
- The average distance between a random node in a $q \times q$ mesh and a random node on the edge of the mesh is $\frac{5q}{6} - \frac{2}{3}$ (4.38)
- The average distance between the center of a $q \times q$ mesh and nodes on the edge is $\frac{3}{4}(q-1)$ (4.39)
- The average distance between two nodes U and Y (Figure 4.11), such that U and Y belong to a $r_2 \times r_2$ mesh, Y is the center of a $r_1 \times r_1$ sub-mesh, U is outside the $r_1 \times r_1$ sub-mesh and r_1 is $O(\sqrt{r_2})$ is approximately $\frac{2}{3}(r_2 + \frac{r_1^2}{r_2} - \frac{1}{r_2})$ (4.40)

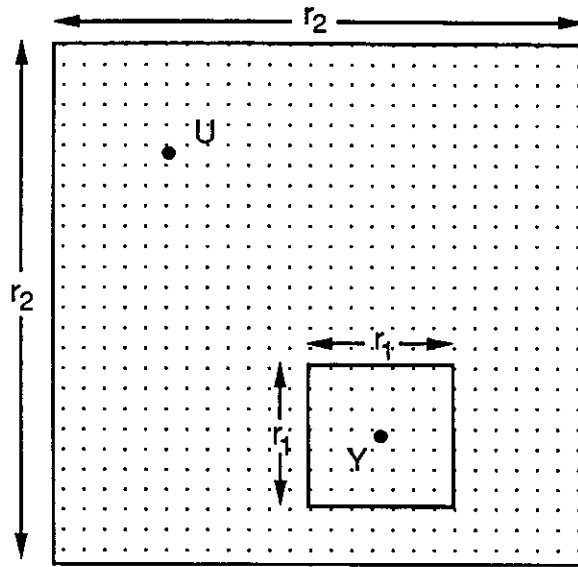


Figure 4.11 : Two Level Neighborhood

Stretch Ratio for 2 level Shifting Neighborhood Protocol

We first evaluate the average message trajectory stretch ratio for the two level SNP. The size of the neighborhoods are chosen according to the analysis of the previous sub-section to minimize the update bandwidth for the 2 - level protocol. Consider a message that is sent from a source process s to a destination process d . The neighborhoods of process d are $\{N_1, N_2\}$ with corresponding sizes $\{r_1, R\}$. We consider two cases - when process s is in and neighborhood N_1 and outside neighborhood N_1 of process d . Let

$$\bar{L}_i = \text{Average length of message trajectory from } s \text{ to } d \text{ for case } i$$

- (1) Source Process s located in N_1

This case is illustrated in Figure 4.12 where a message is being sent from source process s to destination process d . Since we assume that the time to broadcast update messages is small compared to the elapsed time between successive process migrations hence according to the Shifting Neighborhood Protocol the node at which s is located will have an up-to-date routing entry for the location of process d . Therefore a message from process s to d is directly routed from s to the location of process d . We assume that

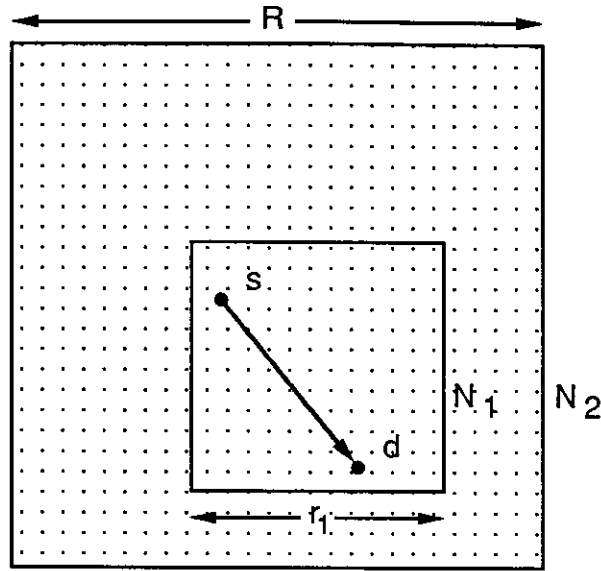


Figure 4.12 : 2 Level SNP - Source of Message Located in the Lowest Level Neighborhood of Destination Process

the destination process d does not move while a message is enroute to it. From Equation 4.36 we have

$$\overline{L}_1 = \frac{2}{3} r_1$$

(2) Source Process s located outside N_1

This case is illustrated in Figure 4.13. According to the Shifting Neighborhood Protocol, a message from process s is first routed to Y , the center of N_1 , and then re-routed to process d .

$$\therefore \overline{L}_2 = \overline{d}_1 + \overline{d}_2$$

From Equation 4.37 we have

$$\overline{d}_1 = \frac{r_1}{2}$$

From Equation 4.40 we have

$$\overline{d}_2 = \frac{2}{3} \left(R + \frac{r_1^2}{R} - \frac{1}{R} \right)$$

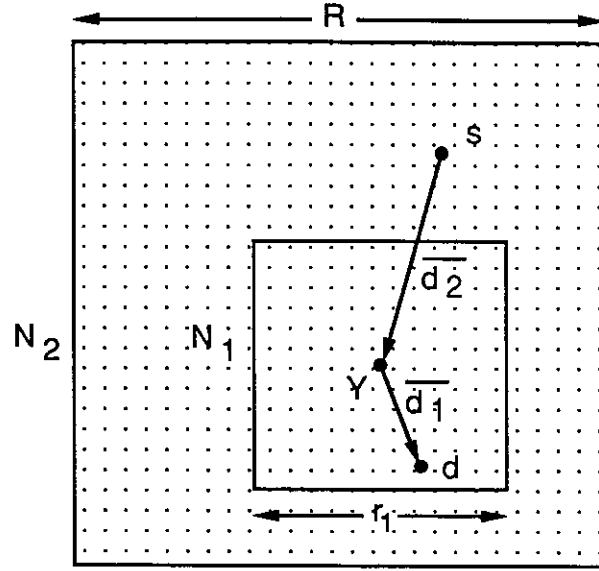


Figure 4.13 : 2 Level SNP - Source of Message Located outside the Lowest Level Neighborhood of Destination Process

$$\therefore \bar{L}_2 = \frac{r_1}{2} + \frac{2}{3} \left(R + \frac{r_1^2}{R} - \frac{1}{R} \right)$$

To evaluate the average trajectory length of a message sent from an arbitrary source to an arbitrary destination the trajectory lengths obtained for the two cases are weighted with the probability of the occurrence of each case, i.e.

$$\bar{h}_r = \frac{1}{R^2} \left[\frac{2}{3} r_1^3 + (R^2 - r_1^2) \left[\frac{r_1}{2} + \frac{2}{3} \left(R + \frac{r_1^2}{R} - \frac{1}{R} \right) \right] \right]$$

From Equation 4.36 we have the average shortest distance between any two processes in the system

$$\bar{h} = \frac{2}{3} R$$

The choice of r_1 for a 2-level SNP to minimize the update bandwidth is obtained from Equation 4.6 as

$$r_1 = \sqrt{R} \sqrt{l}$$

$$\therefore \bar{h}_r = \frac{2l\sqrt{l}}{3\sqrt{R}} + \frac{(R-l)}{R} \left[\frac{\sqrt{Rl}}{2} + \frac{2}{3} R + \frac{2}{3} l - \frac{2}{3R} \right]$$

$$\begin{aligned}
&= \frac{2}{3} R + \frac{\sqrt{Rl}}{2} + \frac{l\sqrt{l}}{6\sqrt{R}} - \frac{2(1+l^2)}{3R} + \frac{2l}{3R^2} \\
\therefore \quad MTSR &= \frac{\bar{h}_r}{\bar{h}} = 1 + \frac{3\sqrt{l}}{4\sqrt{R}} + \frac{l\sqrt{l}}{4R\sqrt{R}} - \frac{(1+l^2)}{R^2} + \frac{l}{R^3} \\
&= 1 + \frac{3\sqrt{l}}{4\sqrt{R}} \\
RPL &= \frac{\bar{h}_r}{\bar{h}} - 1 = \frac{3\sqrt{l}}{4\sqrt{R}} + \frac{l\sqrt{l}}{4R\sqrt{R}} - \frac{(1+l^2)}{R^2} + \frac{l}{R^3} \\
&= O\left(\frac{1}{\sqrt{R}}\right)
\end{aligned}$$

Thus for a 2-level Shifting Neighborhood Protocol the relative increase in the path length is $O\left(\frac{1}{\sqrt{R}}\right)$. This result is based on the assumption that update and user message traversal times are small compared to the time between successive migrations of a process.

Stretch Ratio for SNP with Optimal Neighborhood Structure

We extend the analysis for the 2 level SNP to the more general case when an optimal neighborhood structure is chosen for SNP. Let m be the optimal number of levels of SNP and $(r_1, r_2 \dots r_m)$ be the optimal radii of the m neighborhoods of the process, where $r_m = R$. Figure 4.14 illustrates a particular message trajectory for a 3 level SNP.

From Equation 4.36 we have

$$\bar{h} = \frac{2}{3} R$$

The optimal size of neighborhoods to minimize the update bandwidth is given by Equation 4.6 as

$$r_i = \left(\frac{R}{l}\right)^{\frac{i}{m}} l \quad \text{for } 1 \leq i \leq m-1$$

From Equation 4.3 we have

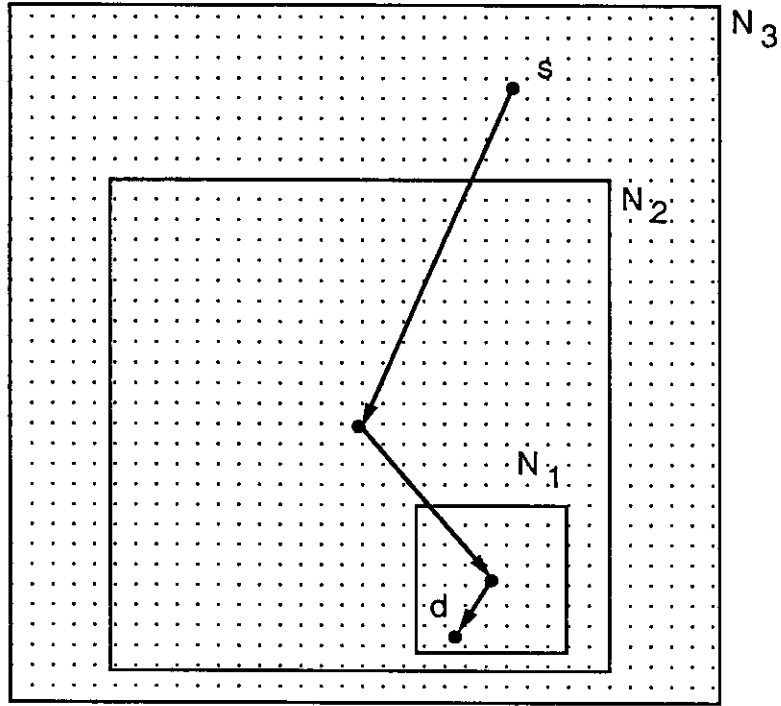


Figure 4.14 : Message Trajectory for 3 Level SNP

$$r_i = \left(\frac{r_1}{l}\right)^i l \quad \text{for } 1 \leq i \leq m-1$$

In a manner similar to the two level protocol we can express \bar{h}_r for the m level protocol as

$$\bar{h}_r = \frac{1}{R^2} \left[\frac{2}{3} r_1 r_1^2 + \sum_{i=1}^{m-1} \left((r_{i+1}^2 - r_i^2) \left[\sum_{j=1}^i \frac{r_j}{2} + \frac{2}{3} \left(r_{i+1} + \frac{r_i^2}{r_{i+1}} - \frac{1}{r_{i+1}} \right) \right] \right) \right]$$

If we assume the constant $l = 1$ then

$$r_i = R^{i/m} = r_1^i$$

$$\therefore \sum_{i=1}^{m-1} \left[(r_{i+1}^2 - r_i^2) \left[\sum_{j=1}^i \frac{r_j}{2} + \frac{2}{3} \left(r_{i+1} + \frac{r_i^2}{r_{i+1}} - \frac{1}{r_{i+1}} \right) \right] \right]$$

$$\begin{aligned}
&= \sum_{i=1}^{m-1} \left[(r_1^{2(i+1)} - r_1^{2i}) \left(\frac{r_1^{i+1} - r_1}{2(r_1 - 1)} + \frac{2}{3} r_1^{i+1} + \frac{2}{3} r_1^{i-1} - \frac{2}{3 r_1^{i+1}} \right) \right] \\
&= (r_1^2 - 1) \left[\frac{r_1}{2(r_1 - 1)} \left[\left(\frac{r_1^{3m} - r_1^3}{r_1^3 - 1} \right) - \left(\frac{r_1^{2m} - r_1^2}{r_1^2 - 1} \right) \right] + \frac{2}{3} r_1 \left(\frac{r_1^{3m} - r_1^3}{r_1^3 - 1} \right) \right. \\
&\quad \left. + \frac{2}{3 r_1} \left(\frac{r_1^{3m} - r_1^3}{r_1^3 - 1} \right) - \frac{2}{3 r_1} \left(\frac{r_1^m - r_1}{r_1 - 1} \right) \right] \\
\bar{h}_r &= \frac{(r_1 + 1)(r_1^{3m} - r_1^3)(4r_1^3 - r_1^2 + 4r_1 - 4)}{6(r_1^{2m+4} - r_1^{2m+1})} - \frac{2(r_1^{m+1} - r_1^2 + r_1^m - r_1)}{3r_1^{2m+1}} \\
&\quad - \frac{r_1}{2(r_1 - 1)} \left(1 - \frac{1}{r_1^{2m-2}} \right) + \frac{2}{3r_1^{2m-3}}
\end{aligned}$$

For large m ($m > 10$), ignoring the lower order terms of m we have

$$\begin{aligned}
\bar{h}_r &= \frac{2}{3} R + \frac{1}{2} R^{\frac{m-1}{m}} + \frac{1}{2} R^{\frac{m-2}{m}} + \frac{2}{3} R^{\frac{m-3}{m}} - \frac{1}{6} R^{\frac{m-4}{m}} + \frac{1}{2} R^{\frac{m-5}{m}} \dots \\
\therefore MTSR &= \frac{\bar{h}_r}{h} = 1 + \frac{3}{4R^{\frac{1}{m}}} + \frac{3}{4R^{\frac{2}{m}}} + \frac{1}{R^{\frac{3}{m}}} - \frac{1}{4R^{\frac{4}{m}}} + \frac{3}{4R^{\frac{5}{m}}} \dots
\end{aligned}$$

From Equation 4.9 the optimal number of levels for SNP is given by

$$m = 2 \ln \frac{R}{l}$$

Since $R^{\frac{1}{m}} = R^{\frac{1}{2 \ln R}} = e^{\frac{1}{2}}$, therefore

$$\begin{aligned}
MTSR &= 1 + \frac{3}{4e^{1/2}} + \frac{3}{4e} + \frac{1}{e^{3/2}} - \frac{1}{4e^2} + \frac{3}{4e^{5/2}} \dots \\
&\approx 2.1
\end{aligned}$$

For an optimal SNP we have

$$RPL \approx 1.1$$

Assuming that the constant l is equal to one and the probability for a process to send a message to any process is equal irrespective of its location, we determine that the average message trajectory stretch ratio for SNP with an optimal neighborhood structure is given by a converging series which when summed to 12 terms is equal to 2.05. The average message trajectory stretch ratio in general will depend on l , the constant related to the average size of a move.

We thus show an important property of the Shifting Neighborhood Protocol, i.e. for an optimal neighborhood structure the average message trajectory stretch ratio is *independent of the size of the system*. The optimal neighborhood structure minimizes the update bandwidth. From Equation 4.10 we observe that the update bandwidth due to the migration of a single process in the Shifting Neighborhood Protocol is $O(\log R)$. This shows that the Shifting Neighborhood Protocol is *scalable* with the size of the system.

The assumptions that we make in showing SNP is scalable is that the time for updates to be broadcast is small compared to the time between successive migrations of a process. In addition we assume in the calculation of the stretch ratio for a message that the process does not move while the message is enroute to it. The motion of a process is assumed to be an unrestricted random walk. The target topology is assumed to be a 2 dimensional square toroidal mesh.

Comparison of the Performance of Optimal SNP with 1-level SNP

From the preceding analysis the Shifting Neighborhood Protocol is shown to be scalable with the size of the system. The optimal neighborhood structure, the average update bandwidth and the average message trajectory stretch ratio for the Shifting Neighborhood Protocol is summarized in Table 4.4. The optimal neighborhood structure is the neighborhood structure chosen to minimize update bandwidth. We compare the performance of SNP operating with an optimal neighborhood structure to the performance of a 1-level SNP executing on a square toroidal mesh. The 1-level SNP is equivalent to the protocol with Complete Broadcast on Migration (CBM). From Section 2.4 and Equation 4.8 we obtain the update bandwidth for a 1-level SNP as

$$C_{CBM} = v R^2 \text{ update_msg-hops/sec}$$

Protocol	Optimal Nhbd. Dimensions	Optimal No. of Levels	Update Bandwidth	Message Trajectory Stretch Ratio
SNP	$r_i = \left(\frac{R}{l}\right)^{\frac{i}{m}} l$ for $1 \leq i \leq m$	$2 \ln \frac{R}{l}$	$2 v e l^2 \ln \frac{R}{l}$	Constant ≈ 2.1 (when $l = 1$)

Table 4.4 : Optimal Protocol Parameters, Cost and Performance of SNP

From Table 4.4 we note that the update bandwidth for SNP with an optimal neighborhood structure is

$$C_{SNP} = 2 v e l^2 \ln \frac{R}{l} \text{ update_msg-hops/sec}$$

$$\therefore \frac{C_{SNP}}{C_{CBM}} = 2 e l^2 \frac{\ln \frac{R}{l}}{R^2}$$

The message trajectory stretch ratio for the CBM protocol is equal to one assuming that the time between successive migrations of a process is large compared to the time to broadcast updates in a neighborhood and that the process does not migrate while a message is enroute to it. Since we have determined that the average message trajectory ratio for SNP is a constant hence the ratio of the message trajectory stretch ratios for the two protocols is

$$\frac{MTSR_{SNP}}{MTSR_{CBM}} = \text{Constant}$$

Hence SNP has bandwidth that is significantly lower than CBM while the message trajectory stretch ratios for the two schemes are comparable.

Stretch Ratio for the 2-level Shifting Neighborhood Protocol with Adaptive Routing

Message routing in the SNP(AR) protocol is different from routing in SNP because of the use of adaptive message routing in the SNP(AR). In SNP(AR) a message is re-routed at intermediate nodes on the message trajectory, if the intermediate nodes have more recent routing information about the destination process than the message has. We evaluate the stretch ratio for the simplest version of SNP(AR) described in Section 4.5 where messages are routed towards the center of a lower level neighborhood of the destination process.

The neighborhood dimensions are chosen in order to minimize the update bandwidth for the 2 - level protocol. Consider a message that is sent from a source process s to a destination process d . The neighborhoods of process d are $\{N_1, N_2\}$ with corresponding sizes $\{r_1, R\}$. As in the analysis of the 2-level SNP we consider two cases - when process s is in and outside neighborhood N_1 of process d . Let

$\overline{h_{ar}}$ = The length of the trajectory of a message in hops averaged over all possible source and destination process locations when messages are adaptively routed

\overline{L}_i = Average length of message trajectory from s to d for case i

(1) Source Process s located in N_1

This case is illustrated in Figure 4.15 where a message is sent from source process s to destination process d . If we assume that the time for updates to be delivered is small compared to the time for which a process is resident at a node then according to SNP(AR) nodes in N_1 will have an up-to-date routing entry for process d . Hence a message from process s is directly routed to the location of process d . Hence for case (i) the average length of a message trajectory for SNP and SNP(AR) is identical. From Equation 4.36 we have

$$\overline{L}_1 = \frac{2}{3} r_1$$

(2) Source Process s located outside N_1

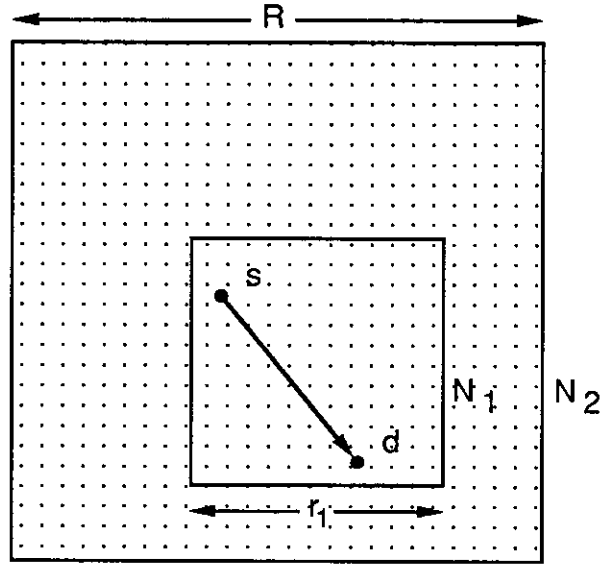


Figure 4.15 : 2 Level SNP with Adaptive Routing - Source of Message Located in the Lowest Level Neighborhood of Destination Process

This case is illustrated in Figure 4.16. In the Shifting Neighborhood Protocol with Adaptive Routing, a message from process s is initially routed towards node Y the center of N_1 , but at intermediate node Z the message is re-routed to the current location of process d .

$$\therefore \bar{L}_2 = \bar{d}_1 + \bar{d}_2$$

From Equations 4.38 we have

$$\bar{d}_1 = \frac{5r_1}{6} - \frac{2}{3}$$

From Equations 4.39 and 4.40 we have

$$\bar{d}_2 = \left[\frac{2}{3} \left(R + \frac{r_1^2}{R} - \frac{1}{R} \right) - \frac{3}{4} (r_1 - 1) \right]$$

$$\bar{L}_2 = \frac{2}{3} R + \frac{r_1}{12} + \frac{2r_1^2}{3R} + \frac{1}{12} - \frac{2}{3R}$$

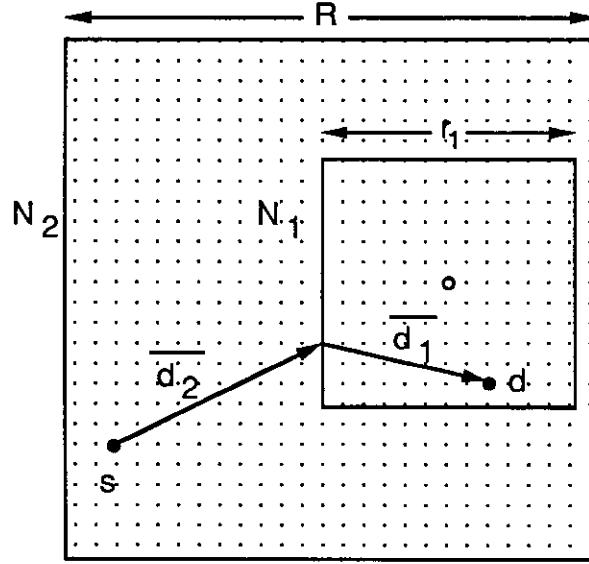


Figure 4.16 : 2 Level SNP with Adaptive Routing - Source of Message Located outside the Lowest Level Neighborhood of Destination Process

To evaluate the average trajectory length of a message sent from an arbitrary source to an arbitrary destination the trajectory lengths obtained for the two cases are weighted with the probability of the occurrence of each case, i.e.

$$\overline{h_{ar}} = \frac{1}{R^2} \left[\frac{2}{3} r_1^3 + (R^2 - r_1^2) \left(\frac{2}{3} R + \frac{r_1}{12} + \frac{2 r_1^2}{3 R} + \frac{1}{12} - \frac{2}{3 R} \right) \right]$$

The choice of the size of the lowest level neighborhood that minimizes the update bandwidth for a 2-level SNP(AR) is identical to that for SNP and is obtained from Equation 4.6 as

$$r_1 = \sqrt{R} \sqrt{l}$$

$$\therefore \overline{h_{ar}} = \frac{2}{3} R + \frac{\sqrt{R l}}{12} + \frac{1}{12} + \frac{1}{\sqrt{R}} \left(\frac{7 l \sqrt{l}}{12} \right) - \frac{1}{R} \left(\frac{2 l^2}{3} + \frac{l}{12} + \frac{2}{3} \right) + \frac{2 l}{3 R^2}$$

From Equation 4.36 we have

$$\bar{h} = \frac{2}{3} R$$

$$\therefore MTSR = \frac{\bar{h}_{ar}}{\bar{h}} = 1 + \frac{\sqrt{l}}{8\sqrt{R}} + \frac{1}{8R} + \frac{7l\sqrt{l}}{8R\sqrt{R}} - \frac{1}{R^2} \left(l^2 + \frac{l}{8} + 1 \right) + \frac{l}{R^3}$$

$$\approx 1 + \frac{\sqrt{l}}{8\sqrt{R}}$$

$$RPL = \frac{\bar{h}_r}{\bar{h}} - 1 = \frac{\sqrt{l}}{8\sqrt{R}} + \frac{1}{8R} + \frac{7l\sqrt{l}}{8R\sqrt{R}} - \frac{1}{R^2} \times \left(l^2 + \frac{l}{8} + 1 \right) + \frac{l}{R^3}$$

$$= O\left(\frac{1}{\sqrt{R}}\right)$$

The relative increase in the path length for the 2-level Shifting Neighborhood with Adaptive Routing is also of order $O\left(\frac{1}{\sqrt{R}}\right)$. However the stretch ratio and relative increase in path length for SNP(AR) is smaller by a constant factor than the corresponding measures for SNP. The difference in the message trajectory stretch ratio for a 2-level SNP with and without adaptive routing is given by

$$MTSR_{SNP} - MTSR_{SNP(AR)} = \frac{5\sqrt{l}}{8\sqrt{R}} - \frac{1}{8R} - \frac{1}{R\sqrt{R}} \left(\frac{5l\sqrt{l}}{8} \right) + \frac{l}{8R^2}$$

Thus we observe that the message trajectory stretch ratio is reduced when messages are routed adaptively in the Shifting Neighborhood Protocol. The ratio of the message trajectory stretch ratios for SNP and SNP(AR) is

$$\frac{MTSR_{SNP}}{MTSR_{SNP(AR)}} = 1 + \frac{5\sqrt{l}}{8\sqrt{R}} + \dots \text{lower order terms}$$

Summary of Stretch Ratio results for SNP and SNP(AR)

In this sub-section we have calculated the average message trajectory stretch ratio for SNP and SNP(AR). The underlying topology is assumed to be a 2-dimensional square toroidal mesh. To calculate the stretch ratio we assume the uniform probability model of communication, i.e. a process sends a message to any process with equal probability. We also assume that the time interval between successive migrations of a process is large compared to the time for updates to be broadcast in a neighborhood. In addition as before we assume that process motion can be modeled as an unrestricted random walk. The dimensions of neighborhoods are assumed to be real valued.

Protocol	No. of Levels	Neighborhood Dimensions	Message Trajectory Stretch Ratio
SNP	2	$r_1 = \sqrt{R l}$ $r_2 = R$ for min. update bw.	$1 + \frac{3\sqrt{l}}{4\sqrt{R}} + \dots$
SNP(AR)	2	$r_1 = \sqrt{R l}$ $r_2 = R$ for min. update bw.	$1 + \frac{5\sqrt{l}}{8\sqrt{R}} + \dots$
SNP	$2 \ln \frac{R}{l}$ for min. update bw.	$r_i = \left(\frac{R}{l}\right)^{\frac{i}{m}} l$ for $1 \leq i \leq m$ for min. update bw.	Constant ≈ 2.1 (when $l = 1$)

Table 4.5 : Stretch Ratio results for SNP and SNP(AR)

Table 4.5 summarizes the stretch ratio results for a 2-level SNP, 2-levels SNP(AR) and SNP with optimal number of levels. For the 2-level schemes the neighborhood dimensions are chosen such that the update bandwidth is minimized. For SNP with optimal number of levels the neighborhood dimensions and the number of

levels are chosen such that the update bandwidth is minimized.

For the 2-level SNP we find that the average message trajectory stretch ratio is close to 1 and the relative increase in path length is of $O(R^{-1/2})$. For the 2-level SNP(AR) we determine that the average message trajectory stretch ratio is lower than the ratio for a 2-level SNP. However the relative increase in path length for SNP(AR) is lower than SNP only by a constant factor assuming that the pattern of message communication can be modeled by the uniform communication model. The important result that we derive in this sub-section is that for SNP with optimal number of levels the message trajectory stretch ratio only depends on the average size of a move and is independent of the size of the system. However while the performance of SNP is independent of the size of the system, the cost in terms of update bandwidth increases as $O(\log R)$, where R is the dimension of the underlying toroidal mesh.

4.7 Simulation of the Shifting Neighborhood Protocol and Related Protocols

We simulate the Shifting Neighborhood Protocol operating in a large distributed system and study the effect of the variation of protocol and system parameters on the performance of the protocol. We compare the performance of the four proposed protocols, i.e. Shifting Neighborhood Protocol (SNP), Shifting Neighborhood Protocol with Dynamic Broadcast (SNP(DB)), Shifting Neighborhood Protocol with Adaptive Routing (SNP(AR)), and Moving Home Node Protocol (MHNP), and investigate the circumstances under which one protocol performs better than another.

As in the simulation of the caching schemes (Section 3.3.2) the simulator developed for the simulation of SNP and related protocols is based on SIMON [Swope 86]. SIMON provides mechanisms for the definition of objects that may send messages to each other which makes it convenient to model message passing multiprocessor systems. In particular one of the features of SIMON that was very useful for our simulation was the ability to move objects and to dynamically make connections between objects. This feature facilitated the implementation of process migration in the simulation.

In the simulation we consider a particular process q that is migrating in a large distributed system. Processes located at the different nodes of the system send messages at different times to process q . We consider a single destination process receiving messages and a number of source processes sending messages. Since process q is migrating, messages sent to q may have to be re-routed before they are delivered to the process. Every node in the system is assumed to have a routing entry for process q . In other words we assume that the routing table at any node is complete.

In the simulation model we assume that processes are uniformly distributed over nodes so that there are an equal number of processes per node at any time. We model a number of source processes located at a node and that send messages to process q by implementing a single message source per node.

The topology of the underlying system chosen for the simulation is the 2-dimensional square toroidal mesh. By choosing the square toroidal mesh we take advantage of the symmetry and the lack of edges in the toroidal mesh.

4.7.1 Model of Message Communication between Processes

In the analytical model for the evaluation of the average message stretch ratio in Section 4.6.2 we assumed the *uniform probability model* of communication between processes where each process irrespective of its location has an equal probability of sending a message to a particular process q . We now consider a more general model of communication between processes - the *spatial locality model* that captures a range of communication patterns between processes including the uniform probability model.

Let the probability of sending a message to a process located l hops away from the source be

$$c \times a^l \quad \text{where } 0 < a < 1$$

The parameter a called the *Locality Base* controls the degree of locality in the communication, while c is a normalizing constant. When a approaches 1 the probability of sending a message from a source process at any node location to the destination process q is almost equal and therefore approximates uniform routing. If a approaches 0 we have a traffic pattern of high spatial locality. The spatial locality model is similar to

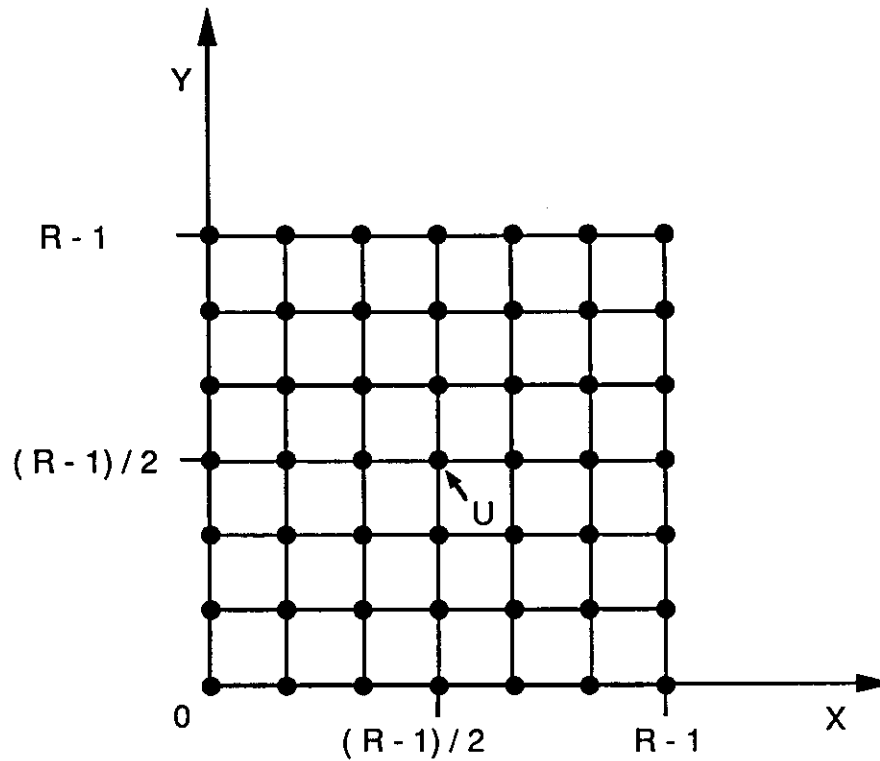


Figure 4.17 : Square Mesh

one adopted by [Reed 87].

The underlying square toroidal mesh topology is shown in Figure 4.17 with end to end connections between the edge nodes. In the square toroidal mesh due to symmetry any node U can be considered to be the center of the mesh. Without loss of generality and for the convenience of the analysis we assume that the side R of the square toroidal mesh is odd. The normalizing constant c is chosen so that the probability of sending a message to the process is 1. That is

$$\sum_{i=0}^{R-1} c \times a^i \times N_i = 1$$

where N_i = No. of nodes at distance i from a process located at node U

a = Locality Base

c = Normalizing constant

R = Side of toroidal mesh

From Figure 4.17 we observe

$$N_i = \begin{cases} 1 & i = 0 \\ 4i & 1 \leq i \leq (R-1)/2 \\ 4(R-i) & (R+1)/2 \leq i \leq R-i \end{cases}$$

$$\therefore 1 = c \left[1 + \sum_{i=1}^{(R-1)/2} 4i a^i + \sum_{i=(R+1)/2}^{R-1} 4(R-i) a^i \right]$$

$$\Rightarrow c = \frac{(1-a)^2}{(1-a)^2 + 4a(1-a^{(R-1)/2})(1-a^{(R+1)/2})}$$

We can also calculate the average distance between the source and destination when a message is sent as

$$\frac{1}{\sum_{i=0}^{R-1} N_i} \times c \sum_{i=0}^{R-1} a^i \times N_i \times i$$

4.7.2 Model for Process Migration

The model of process migration chosen for the simulation of the Shifting Neighborhood Protocol is similar to the model of process migration used for the analysis. In the analysis we assumed that the size of a move of a process is constant. In the simulation of SNP the size of the move is geometrically distributed. The motion of the process is modeled as a random walk.

Consider a process executing at a particular node. After every load management interval τ , the load management policy decides on whether to let the process execute at the same node or to migrate it. In our model of the load management policy the load management interval τ is assumed to be constant.

Let

p = Probability that the process is not moved at the load management decision time
and

X = Number of hops that the process is moved at a load management decision time

Then the probability that a process is moved i hops at a load management interval is

$$P [X = i] = p (1 - p)^i \quad \text{where } i = 0, 1, 2, \dots, R-1$$

The average size of moves including moves of size 0 is

$$\begin{aligned} E[X] &= \sum_{i=0}^{R-1} p (1 - p)^i i \\ &= \frac{p (1 - p) \left[1 - R (1 - p)^{R-1} + (R - 1) (1 - p)^R \right]}{p^2} \end{aligned}$$

If the system is large, i.e. R is large then

$$E[X] \approx \frac{1-p}{p}$$

The average size of a move excluding moves of size 0 is

$$\begin{aligned} E[X \mid X > 0] &= \sum_{i=0}^{R-1} i P[X = i \mid X > 0] \\ &= \sum_{i=0}^{R-1} \frac{i p (1 - p)^i}{1 - p} \\ &= p \sum_{i=0}^{R-1} i (1 - p)^{i-1} \end{aligned}$$

If $R \rightarrow \infty$ then

$$E[X \mid X > 0] \approx \frac{1}{p}$$

We now calculate the average residence time of a process at a node. Let

Z = Number of load management policy decisions including the first time the load management policy decides to move the process from the node on which it is resident.

The probability that the random variable Z has value k is

$$P [Z = k] = p^{k-1} (1 - p) \quad \text{where } k = 1, 2, \dots$$

The expected value of Z is given by

$$E[Z] = \sum_{i=0}^{\infty} k p^{k-1} (1-p)$$

$$= \frac{1}{1-p}$$

The process remains at a node for the duration of $E[Z]$ load management intervals including the interval when the process first arrives at a node. If the load management interval is τ secs then the average residence time of a process at a node is

$$\frac{\tau}{1-p} \text{ secs}$$

The rate of migration of a process is

$$\frac{1-p}{\tau} \text{ moves/sec}$$

4.7.3 Organization and Parameterization of the Simulation

The target topology we consider is an $R \times R$ toroidal mesh. The neighborhoods of a process are chosen to be square regions of the mesh as illustrated in Figure 4.10. In the simulation the system consists of nodes connected together by communication links. A node as implemented in the simulation is shown in Figure 4.18. The *source* models multiple source processes at a node sending messages to different processes in the system including to the process q under observation. Figure 4.19 represents the node where the process q is currently resident. If the destination process of the message generated by the source at a node is the process q then the message is passed to the Communication Co-Processor (CCP) else no action is taken on the message since we are only interested in monitoring those messages destined for the particular process q . The CCP handles the interaction of a node with other nodes. It is responsible for routing user messages, broadcasting update messages, updating routing tables based on information in update messages received, and delivering user messages to the destination process located at that node. The CCP is connected by four incoming and four outgoing links to the neighbors of that node. The process shown in Figure 4.19 is the destination process q that receives messages sent from different nodes in the system.

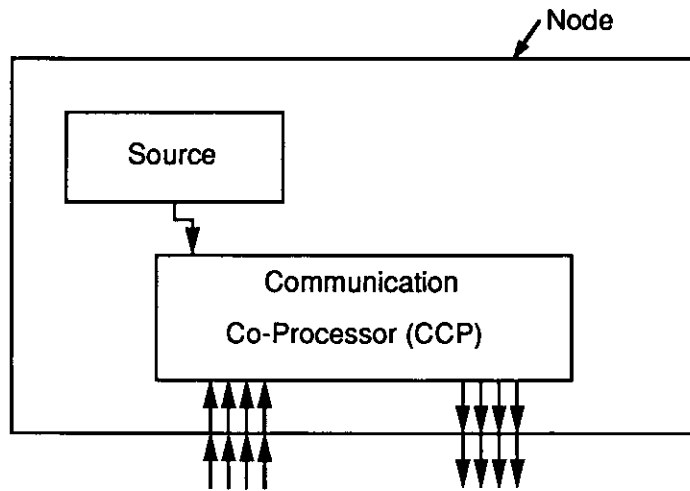


Figure 4.18 : Representation of a Node in the Simulation

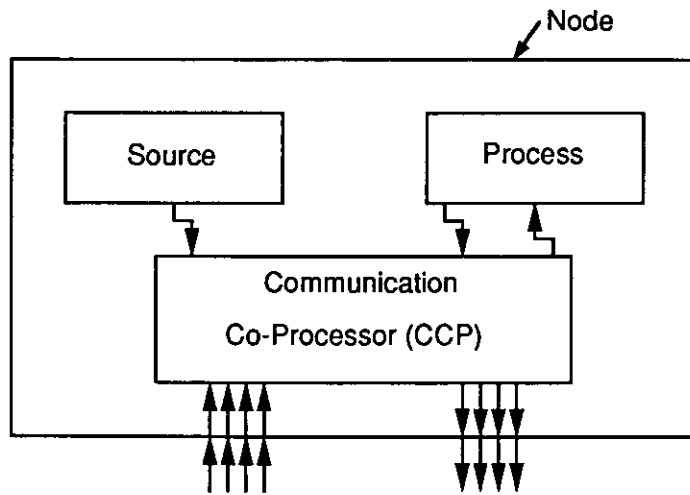


Figure 4.19 : Representation of Node containing Destination Process in the Simulation

Broadcast of update messages to a square mesh region around a node is done by *tree broadcasting*. Broadcast from a node U to a 7×7 mesh around U is shown in Figure 4.20. An update message carries a counter that is used to constrain the broadcast to the 7×7 sub-mesh. The initial direction of broadcast, i.e. along the x or y axis is chosen randomly. The number of update messages generated by this broadcast mechanism is optimal and any node in the broadcast region receives the update in the shortest possible time.

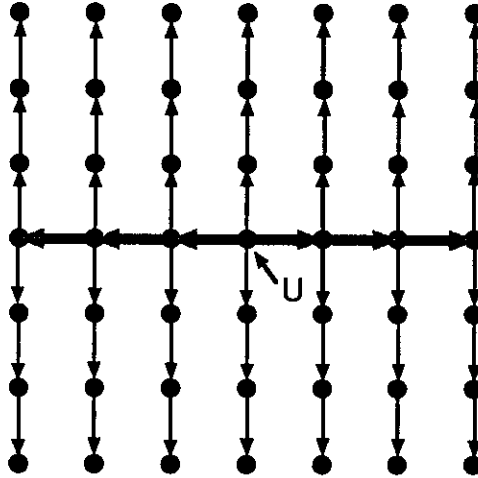


Figure 4.20 : Broadcast in a Mesh

The parameters for the simulation of SNP and related protocols are given in Table 4.6.

At every *MsgInterval* each node independently sends a message to the destination process q with probability $c \times a^l$ where a is the locality base and l is the distance of that node from the location of the process q . Because of the way the normalization constant c is calculated, on the average the process receives 1 message per *MsgInterval*.

The time taken to migrate a process from one node to another is

$$T_{Start} + m \times T_{MoveProcess} + T_{StopProcess}$$

where m is the number of hops that the process is moved.

R	Dimension (side) of the entire square toroidal mesh
m	No. of levels or neighborhoods for a process
r_i	Dimension (side) of neighborhood N_i
$MsgInterval$	Mean interval between messages sent by a process
$DistMsgInterval$	Distribution of msg. interval ("geometric" or "constant")
$LoadMgmtInterval$	Constant period between decision to move process or not
p	Prob. that process is not moved at load mgmt. decision time
$LocalityBase$	Parameter to control degree of locality in communication
$T_{UserMsg}$	Expected time for user message to traverse a hop
$T_{UpdateMsg}$	Expected time for update message to traverse a hop
$T_{MoveProcess}$	Expected time for a process to be sent as a msg. over a hop
$T_{StopProcess}$	Expected time to stop a process and collect its state
$T_{StartProcess}$	Expected time to install a process at a node and start it
T_{RT}	Expected time to access (read or write) a routing table

Table 4.6 : Parameters for the Simulation of the Shifting Neighborhood Protocol

The time for a user message to traverse a hop is $T_{UserMsg}$ which includes the protocol overhead, software overhead and queuing delays. Similarly the time for an update message to traverse a hop is $T_{UpdateMsg}$. The time to read or modify a routing entry is T_{RT} . Consider a message that is sent from a process located at node U to the process q located at node V , where the distance between nodes U and V is m hops. At the time the message is sent from node U let us assume that U has an up-to-date routing entry for process q . Let us also assume that the process q does not migrate away from V while the message is enroute to it. Then for SNP the total time taken to deliver the message to q will be

$$m \times T_{UserMsg} + 2 \times T_{RT}$$

The routing table is checked at nodes U and V .

If however we consider the Shifting Neighborhood Protocol with Adaptive Routing and assume that routing tables are checked at all intermediate nodes, then the total time taken to deliver the message to q will be

$$m \times T_{UserMsg} + (m + 1) \times T_{RT}$$

Whenever a message is forwarded or re-routed the routing table is accessed and hence a delay of T_{RT} is incurred. If a message for process q is sent to a node V and the routing entry for q at node V is locked because the process q is currently being migrated away from V then the message is buffered at node V . Therefore some messages will incur additional delays due to time spent in intermediate buffers.

We assume the capacity of links to be large and do not take into account the additional delay for user and update messages due to congestion of communication links.

The message types defined in the simulation are shown in Table 4.7. *User* messages are application messages sent from one process to another. *Update_Broadcast* messages are routing update messages broadcast to nodes in a square mesh neighborhood. *Update* messages on the other hand are sent from one node to another to update a routing entry at the destination node. *Move* messages carry the process control block, state, code etc. of the process being migrated. A *Move_Confirm* message sent to the previous location of a process confirms that the process has been installed at its new location. Finally *Simulation* messages are messages used for the internal operation of the simulation.

User
Update_Broadcast
Update
Move
Move_Confirm
Simulation

Table 4.7 : Message Types in the Simulation of the Shifting Neighborhood Protocol

For each performance metric we computed the 95 % confidence intervals using the *t distribution*. In the performance graphs the average is plotted as a point and vertical bars indicate the confidence intervals. To eliminate from consideration start-up transient effects, the performance measurements are monitored after the simulation has stabilized.

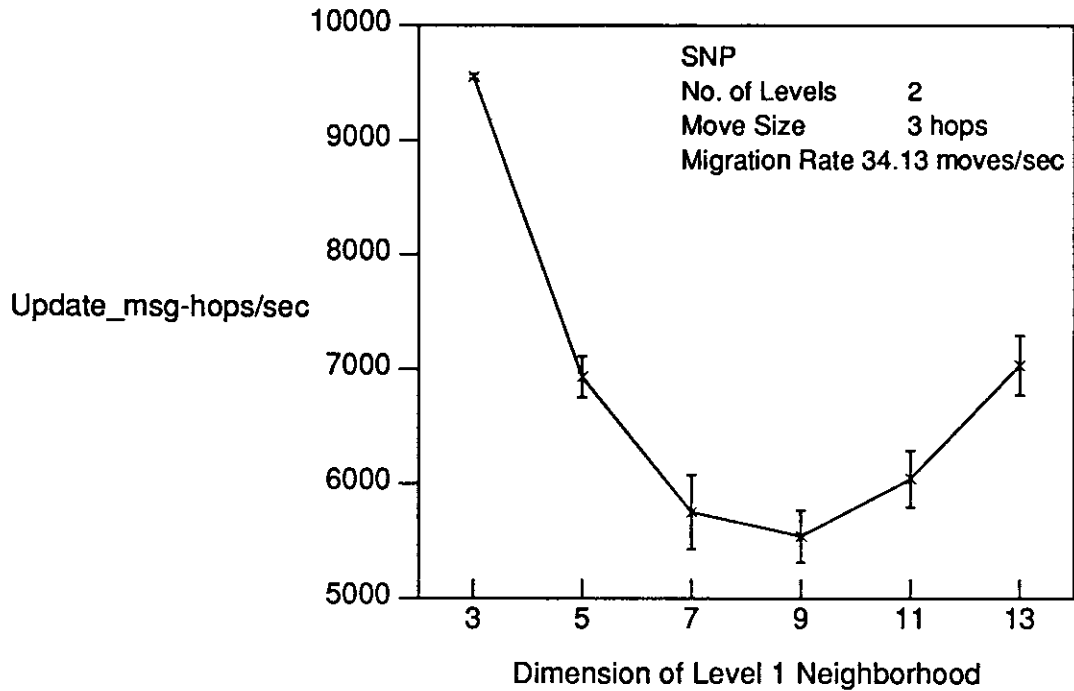
4.7.4 Results of the Simulation of the Shifting Neighborhood Protocol and Related Protocols

We first study the performance of the Shifting Neighborhood Protocol and the effect of the change in protocol parameters on the performance of the protocol. The parameters for these set of simulations are given in Table 4.8. The system parameters are chosen so that process migration occurs at a very rapid rate in order to test the performance of SNP at its limit.

R	21	$T_{UserMsg}$	25 μ secs
MsgInterval	250 μ secs	$T_{UpdateMsg}$	5 μ secs
DistMsgInterval	Constant	$T_{MoveProc}$	500 μ secs
LoadMgmtInterval	18750 μ secs	$T_{StopProc}$	100 μ secs
p	0.33	$T_{StartProc}$	100 μ secs
LocalityBase	0.6	T_{RT}	5 μ secs

Table 4.8 : System Parameters Chosen for the Study of the Effect of Variation in Protocol Parameters on the Performance of the Shifting Neighborhood Protocol

The parameter values are chosen in order to test the performance of the protocols under the most extreme conditions. The load management interval is chosen so that the process receives on the average 75 messages in every load management interval. After every load management interval the process is moved to a new location with probability $\frac{2}{3}$. Thus the rate of process migration is extremely rapid for the current state of operating systems technology. The average size of a move excluding moves of size 0 is about 3 hops. The locality base is chosen to be 0.6 which results in an average distance between the source and destination process at the time the message is sent equal to 3.63 hops.



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.21 : Average Update Bandwidth vs. Dimension of Level 1 Neighborhood

We consider a 20 Mips processor where each instruction takes 50 nsecs. An application message is sent out by a process after every 5000 instructions, i.e. every $MsgInterval$ secs, where $MsgInterval = 5 \times 10^3 \times 50 \times 10^{-9} = 250 \mu$ secs. The average application message length is chosen to be 256 bytes and the average update message length is 4 bytes. We assume a very fast communication medium of 100 Mbits/sec which is equivalent to $\frac{100}{8} \times 10^6$ bytes/sec. Each copy instruction in the hypothetical processor can copy 1 word or 4 bytes. Hence to copy 256 bytes the processor will execute 64 instructions. We compute $T_{UserMsg}$ and $T_{UpdateMsg}$ by calculating the sum of the time to copy the message content, transmit it and an additional overhead due to queuing and

other effects. The size of processes are assumed to be small in order to be able to migrate processes rapidly. We assume that the active part of the process state is 7.5 Kbytes which has to be transferred to the new location of a process before the process can be re-started at the new location. Therefore the time to move a process across a hop $T_{MoveProc}$ is equal to $7.5 \times 10^3 \times \frac{8}{100} \times 10^{-6}$ which we approximate to 500 μ secs, i.e. a factor of 20 times $T_{UserMsg}$. The time to start and stop the process is dominated by the time to copy the process state which will take $\frac{7.5 \times 10^3}{4}$ instructions. Therefore $T_{StartProc} = T_{StopProc} = 50 \times 10^{-9} \times \frac{7.5 \times 10^3}{4}$ which is approximately 100 μ secs.

The system was simulated for 7.5 secs and the performance was monitored after 37500 μ secs.

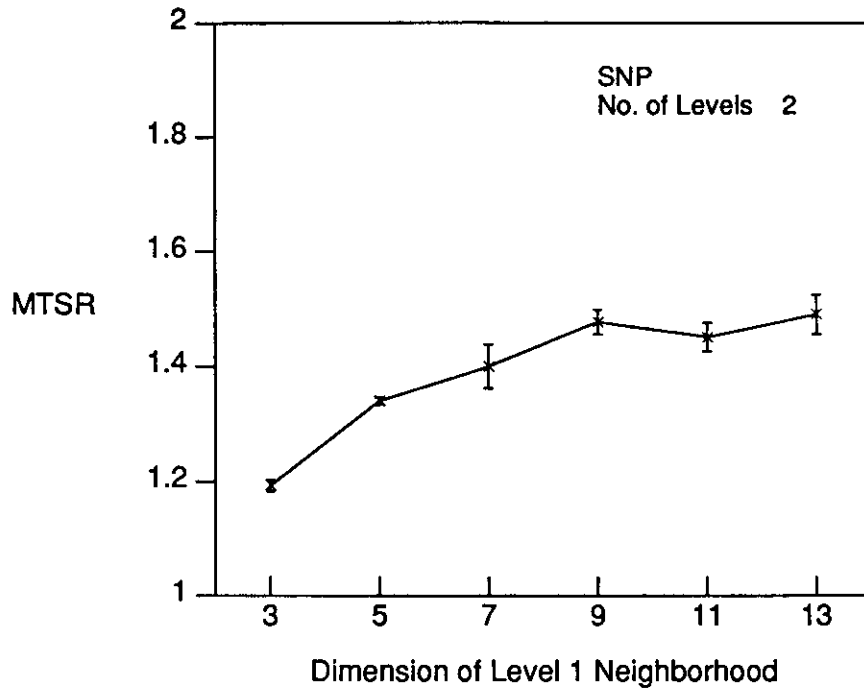
As stated before the capacities of links are assumed to be large and the simulation does not account for the additional time spent by messages in queues. The simulation also does not consider the effects of flow control and the competition amongst messages for buffers.

Figure 4.21 plots the average update bandwidth versus the dimension of the level 1 neighborhood for a 2-level SNP. The update bandwidth is due to the migration of a single process and is expressed in update_msg-hops/sec.

The convex shape of the update bandwidth curve is in accordance with our earlier analysis. When the dimension of the first level neighborhood is small (ex. $r_1 = 3$) then the process frequently moves out of its first level neighborhood resulting in frequent broadcasts to the entire system. On the other hand when the dimension of the first level neighborhood is large (ex. $r_1 = 13$) then every time the process migrates update messages have to be broadcast to a large region resulting in high update bandwidth. The update bandwidth for the 21×21 system is minimum at $r_1 = 9$.

The results obtained from the simulation and illustrated in Figure 4.21 also conform to the results obtained from the analysis of Section 4.6.1. In the analysis we assume that the dimensions of SNP are real valued. Moreover the analysis is based on the assumption that the system is very large. From Equation 4.6 of the analysis we have

$$r_i = \left(\frac{R}{l}\right)^{\frac{i}{m}} l \quad \text{for } 1 \leq i \leq m-1$$



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

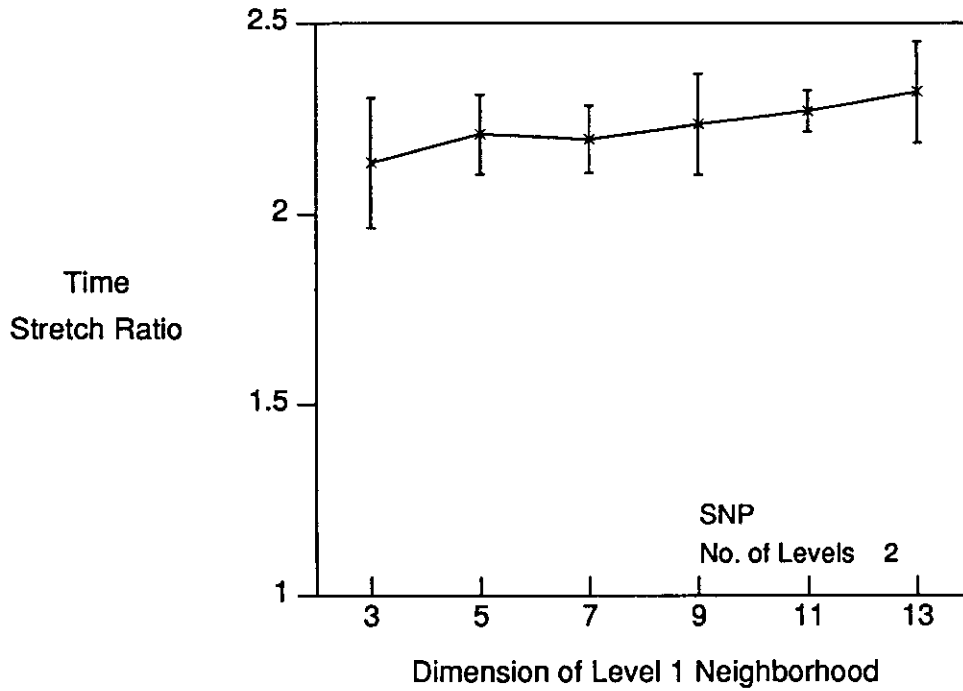
Figure 4.22 : Average Message Trajectory Stretch Ratio vs. Dimension of Level 1 Neighborhood

where R is the dimension (side) of the entire square toroidal mesh and l is a constant related to the average size of a move.

For $m = 2$, $l = 3.2$ and $R = 21$ we obtain from the analysis

$$r_1 = 3.2^{\frac{1}{2}} 21^{\frac{1}{2}} = 8.2$$

Approximating the dimension of the first level neighborhood to the closest integer value we obtain



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.23 : Average Time Stretch Ratio vs. Dimension of Level 1 Neighborhood

$$r_1 \approx 9$$

For a fixed number of levels m we obtain from Equation 4.7

$$C = vmr_1^2$$

where v is the rate of migration of the process (moves/sec).

From the simulation we observe $v = 34.13$ moves/sec. For $m = 2$ and $r_1 = 9$ we obtain from the analysis

$$C_{analysis} = 5529 \text{ update_msg-hops/sec}$$

From the simulation we obtain the minimum update bandwidth at $r_1 = 9$ as

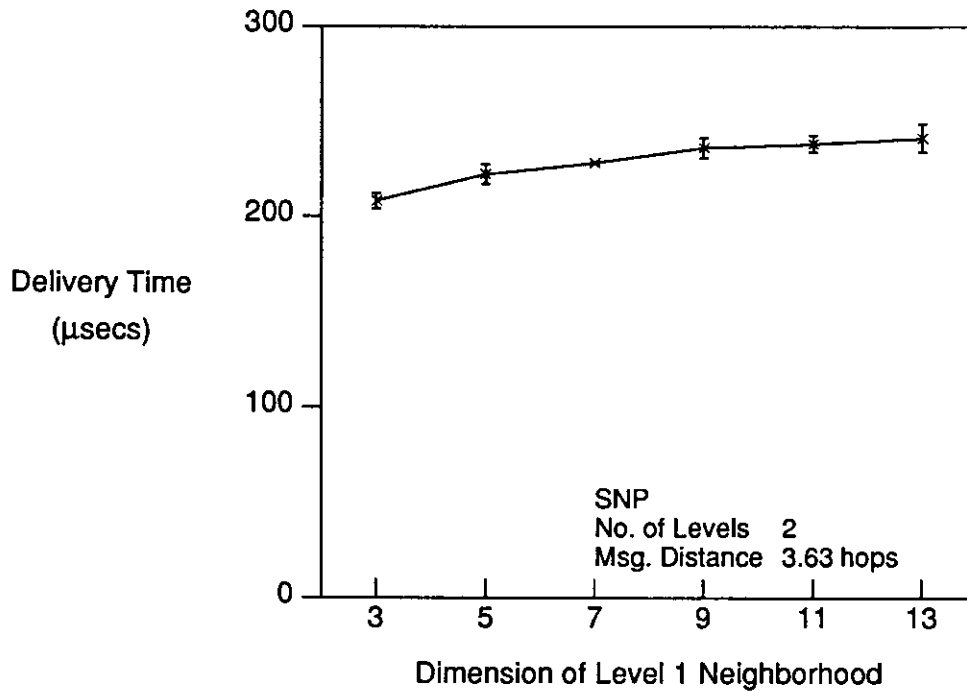
$$C_{sim} = 5540 \pm 45 \text{ update_msg-hops/sec}$$

Thus the results obtained from the analytical model of the SNP are in close conformance with the simulation results.

In the experiment of Figure 4.21 with the dimension of the first level neighborhood chosen as 9 hops we observe the average update bandwidth per link due to a single process is 6.3 update_msgs/link/sec. The percentage of broadcasts to a 9×9 neighborhood is 77.3 % while the rest (22.7 %) of the broadcasts are to the entire system, i.e. to a 21×21 mesh.

In Figure 4.22 we plot the average message trajectory stretch ratio versus the dimension of the level 1 neighborhood for a 2-level SNP. Observe that the MTSR varies from 1.2 to 1.5 as the dimension of the level 1 neighborhood is varied. For the point in the curve of Figure 4.22 where the update bandwidth is minimum, i.e. $r_1 = 9$ the average message trajectory stretch ratio is observed to be 1.5. The message trajectory stretch ratio was calculated for an average of 29846 user messages received by the process. Since the destination process can move while messages are enroute to it and the time for updates to traverse a hop is not zero hence the minimum message trajectory stretch ratio achievable by any protocol will be greater than 1.

Figure 4.23 illustrates for a 2-level SNP the change in the average time stretch ratio for messages when the dimension of the level 1 neighborhood is varied. We observe that the time stretch ratios shown in Figure 4.23 are larger than the message trajectory stretch ratios of Figure 4.22. This is because the time to forward a message and the time spent by messages in buffers are taken into account in calculating the time stretch ratio while costs associated with forwarding and buffering of messages do not figure in the calculation of the message trajectory stretch ratio. The fraction of messages that are forwarded when $r_1 = 9$ is 0.30 and hence almost a third of the messages incur the additional cost of the time taken to access the routing table while forwarding. When $r_1 = 9$ the fraction of messages that are buffered is 0.06. The parameters chosen for this experiment are such that the time to move a process across a hop is an order of magnitude larger than the time for a message to traverse a hop. Moreover since process migration is also relatively rapid, hence messages spend a large fraction of time in buffers waiting for the process to be relocated to its new location before the messages

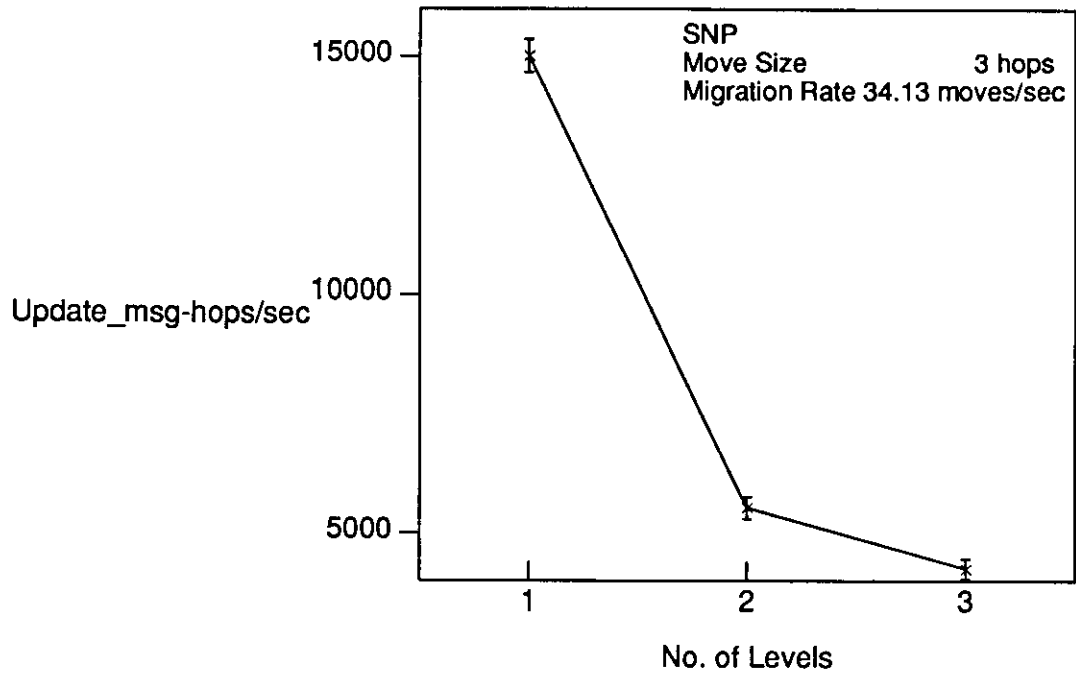


R	21	p	0.33	$T_{MoveProc}$	500 μsecs
MsgInterval	250 μsecs	LocalityBase	0.6	$T_{StopProc}$	100 μsecs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μsecs	$T_{StartProc}$	100 μsecs
LoadMgmtInterval	18750 μsecs	$T_{UpdateMsg}$	5 μsecs	T_{RT}	5 μsecs

Figure 4.24 : Average Time to Deliver a Message vs. Dimension of Level 1 Neighborhood

can be forwarded to the destination process. Therefore even though the fraction of messages that are buffered is small, the time spent by messages in buffers is large thus contributing to the increase in the time stretch ratio.

Figure 4.24 plots the average time taken for messages to be delivered to their destinations versus the dimension of the first level neighborhood. We observe that the average delivery time in Figure 4.24 like the average time stretch ratio shown Figure 4.23 is relatively constant with the change in the dimension of the lowest level neighborhood. Thus a reduction in the update bandwidth achieved by choosing the appropriate first level neighborhood dimension does not result in higher delivery times or time stretch ratios for



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.25 : Average Update Bandwidth vs. No. of Levels in Protocol

messages.

For a 21×21 system the optimal number of levels for the Shifting Neighborhood Protocol is determined from the simulation to be three levels. For a 3-level SNP with parameters as given in Table 4.8 we determine from simulations that the optimum size of the neighborhoods that results in minimum update bandwidth are

$$r_1 = 5 \text{ and } r_2 = 11$$

In Figure 4.25 we plot the minimum update bandwidth for an one, two and three level SNP. For a 1-level SNP which is equivalent to the Complete Broadcast on Migration (CBM) protocol, the update bandwidth is $15,000 \pm 70$ update_msg-hops/sec while for the 3-level SNP the average minimum update bandwidth is 4250 ± 42 update_msg-hops/sec. Thus even for a relatively small system by choosing optimal protocol parameters the update bandwidth is reduced by a factor of 3.5.

We contrast the optimal neighborhood structure values and the update bandwidth obtained in the simulation with values obtained from our analytical model. From the analytical model the optimal number of levels for the Shifting Neighborhood Protocol is given by Equation 4.9 as

$$m = 2 \ln \frac{R}{l}$$

If as before we choose $l = 3.2$ then

$$m = 2 \ln \frac{21}{3.2} = 3.7 \approx 3$$

From Equation 4.6 of the analysis we obtain the optimal neighborhood dimensions for a m level SNP as

$$r_i = \left(\frac{R}{l}\right)^{\frac{i}{m}} l \quad \text{for } 1 \leq i \leq m-1$$

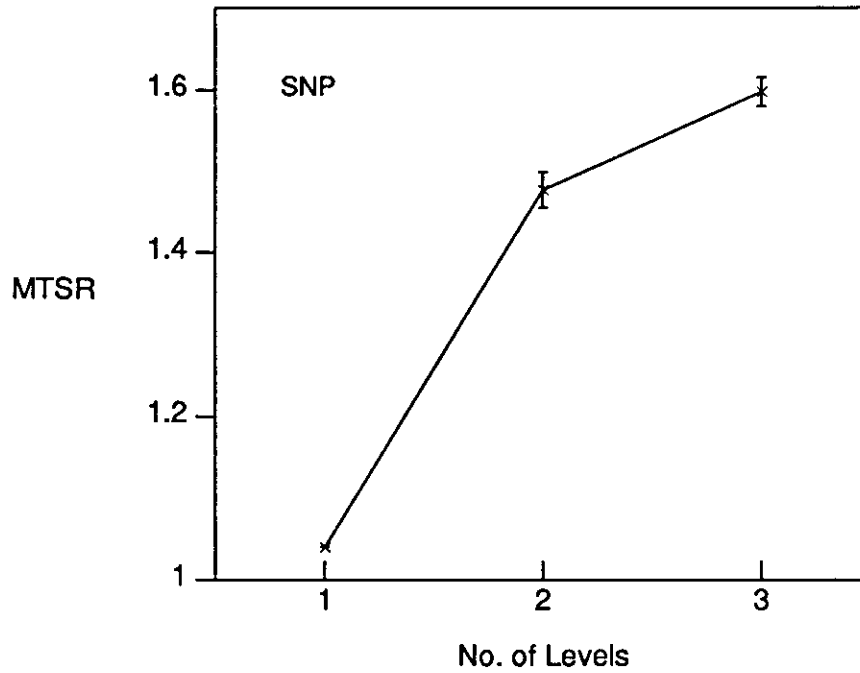
Therefore for a 3-level SNP we obtain

$$r_1 = 5.9 \approx 5$$

$$r_2 = 11.2 \approx 11$$

Since neighborhood dimensions have to be odd integers, hence r_1 is approximated from 5.9 to the closest odd integer, i.e. 5.

Therefore the optimal neighborhood structure, i.e. the optimum number of levels and dimensions of neighborhoods for a 21×21 toroidal mesh as obtained by the analysis and simulation are identical thus lending validity to the analytical model.



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.26 : Average Message Trajectory Stretch Ratio vs. No. of Levels in Protocol

Based on the analytical model the update bandwidth C for the 1-level SNP is given by

$$C = v r_1^2$$

$$= 34.13 \times 21^2 = 15051 \text{ update_msg-hops/sec}$$

This is contrast to the average update bandwidth of 15000 ± 70 update_msg-hops/sec obtained from the simulation.

For the 2-level SNP as noted earlier the average minimum update bandwidth calculated from the analysis is 5529 update_msg-hops/sec while the value obtained from the simulation is 5540 ± 45 update_msg-hops/sec.

For the 3-level SNP we obtain the average update bandwidth due to the migration of a single process from Equation 4.2 as

$$C = v \left[r_1^2 + \frac{r_2^2 l^2}{r_1^2} + \frac{r_3^2 l^2}{r_2^2} \right] \text{ update_msg-hops/sec}$$

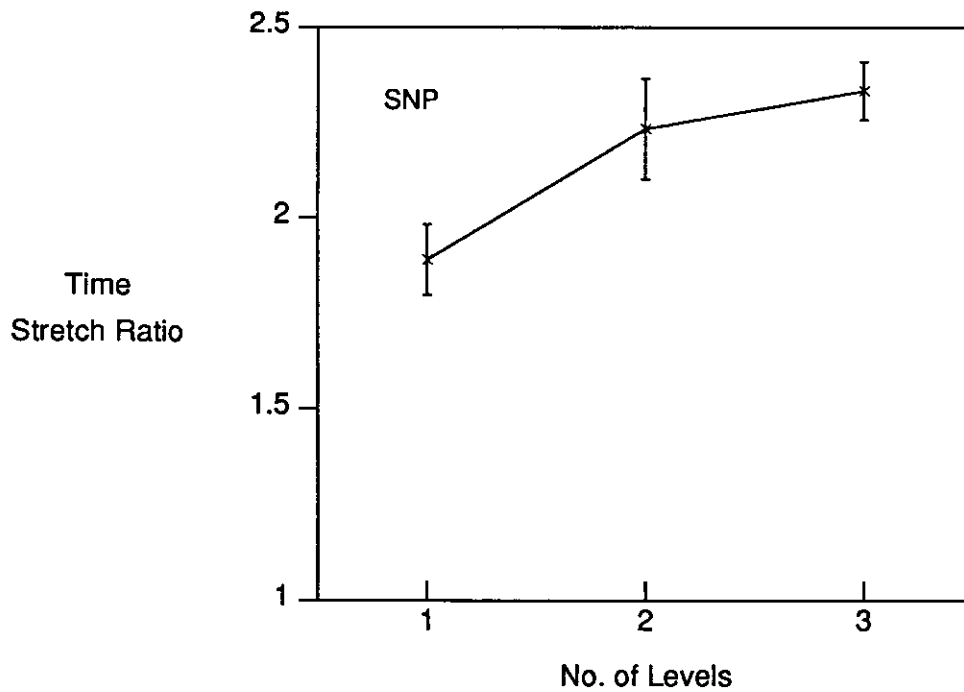
With $l = 3.2$, $v = 34.13$ moves/sec, $r_1 = 5$, $r_2 = 11$ and $R = 21$ we obtain

$$\begin{aligned} C &= 34.13 \left[25 + \frac{121}{25} \times 3.2^2 + \frac{441}{121} \times 3.2^2 \right] \\ &= 3819 \text{ update_msg-hops/sec} \end{aligned}$$

In contrast the average update bandwidth for the 3-level SNP obtained from the simulation is 4250 ± 42 update_msg-hops/sec. The reason for this discrepancy is because the results derived from the random walk model of process migration in the analytical model are accurate only when neighborhood sizes are large. In this case the size of the system is small and the number of levels in the protocol is large resulting in small neighborhood sizes.

The reduction in the average minimum update bandwidth by choosing a 3-level SNP instead of a 1-level SNP is however accompanied by an increase in the message trajectory stretch ratio. The graph of Figure 4.26 illustrates the variation of the average message trajectory stretch ratio with the number of levels of SNP in a 21×21 system. Note that when the number of levels is increased from 1 to 3 the average message trajectory stretch ratio increases by a factor of 1.53.

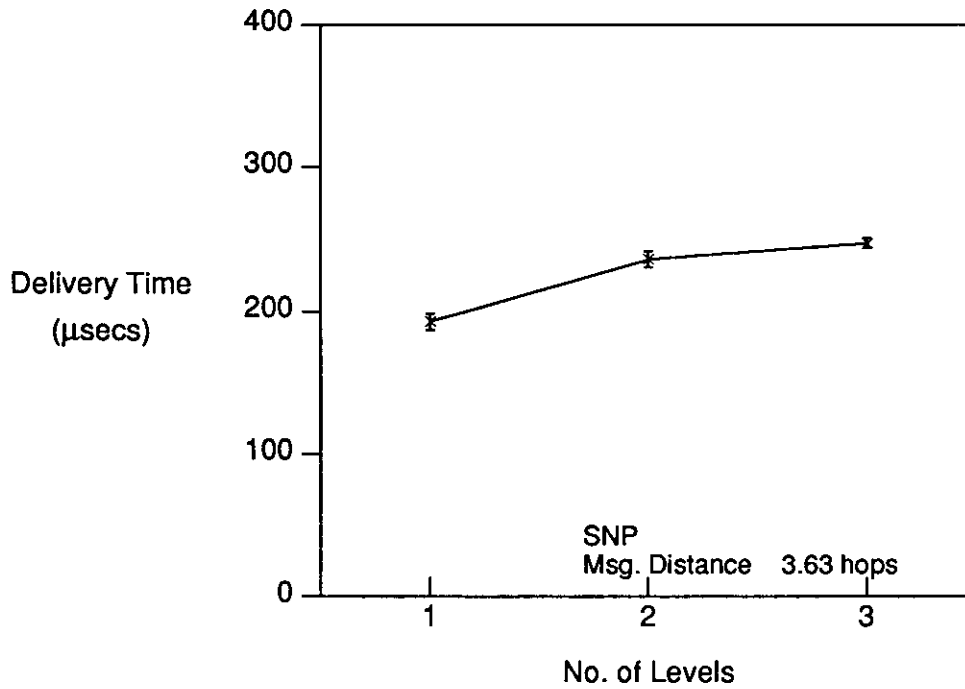
For the Shifting Neighborhood Protocol the average message trajectory stretch ratio for the 3-level scheme is 1.59 ± 0.03 . This value is somewhat high because of anomalies in the performance of SNP discussed in Section 4.3 when a process moves close to the boundary of the neighborhood of that process. Later in this section we observe that SNP(DB) and SNP(AR) have significantly lower message trajectory stretch ratios than SNP.



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.27 : Average Time Stretch Ratio vs. No. of Levels in Protocol

Observe in Figure 4.27 that the variation in the average time stretch ratio with the number of levels is much smaller than the variation of the average message trajectory stretch ratio with the number of levels. This is because the average time stretch ratio is a function of not only the number of hops that a message traverses but also the fraction of messages that are buffered and forwarded, the time taken to forward messages and the time spent by messages in buffers. From the simulation results we observe that while the fraction of messages that are buffered is constant (0.06) as the number of levels of SNP is varied, the fraction of messages that are forwarded is 0.07 for the 1-level SNP, 0.28 for the 2-level SNP and 0.40 for the 3-level SNP.

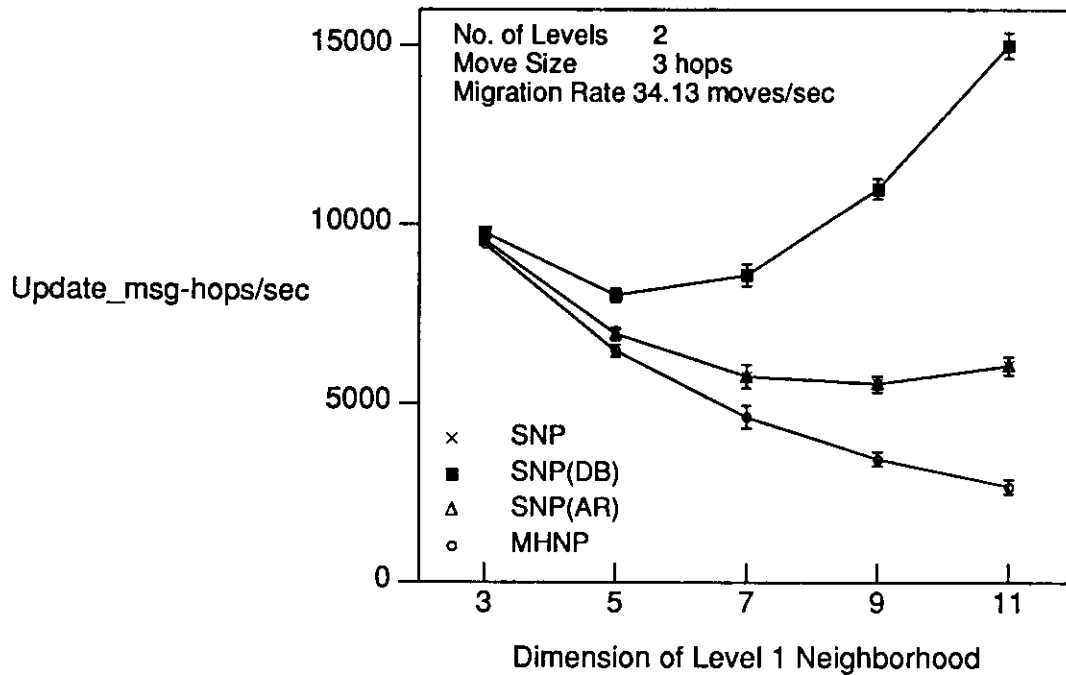


R	21	p	0.33	$T_{MoveProc}$	500 μsecs
MsgInterval	250 μsecs	LocalityBase	0.6	$T_{StopProc}$	100 μsecs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μsecs	$T_{StartProc}$	100 μsecs
LoadMgmtInterval	18750 μsecs	$T_{UpdateMsg}$	5 μsecs	T_{RT}	5 μsecs

Figure 4.28 : Average Time to Deliver A Message vs. No. of Levels in Protocol

In Figure 4.28 we observe the average time to deliver a message versus the number of levels for the Shifting Neighborhood Protocol. Note that the increase in the average delivery time with the increase in the number of levels of SNP is small in contrast to the corresponding decrease in the update bandwidth for the optimal number of levels of SNP.

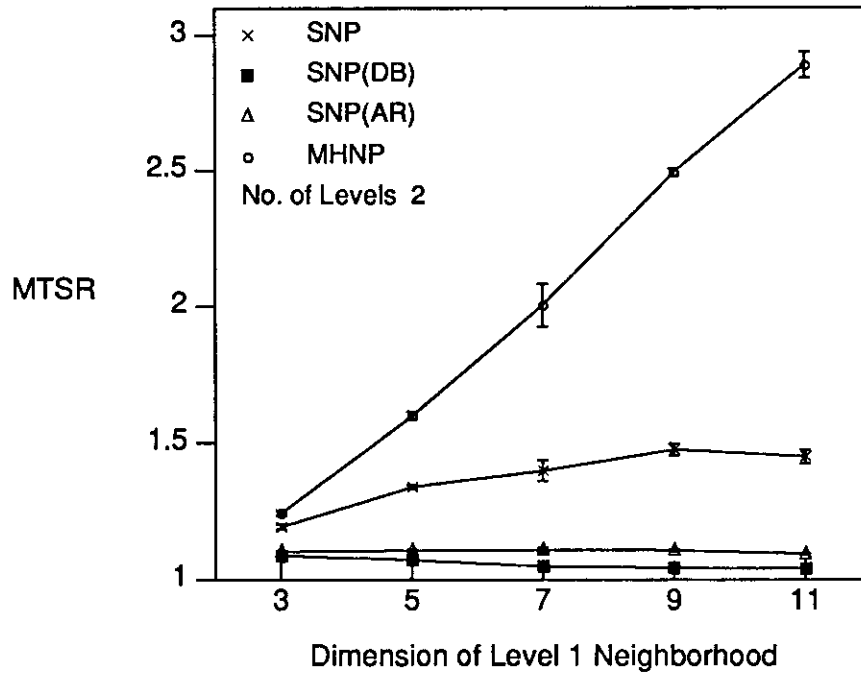
We now compare the performance of the four schemes - SNP, SNP(DB), MHNP and SNP(AR). The parameters chosen are given in Table 4.8.



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.29 : Average Update Bandwidth vs. Dimension of Level 1 Neighborhood

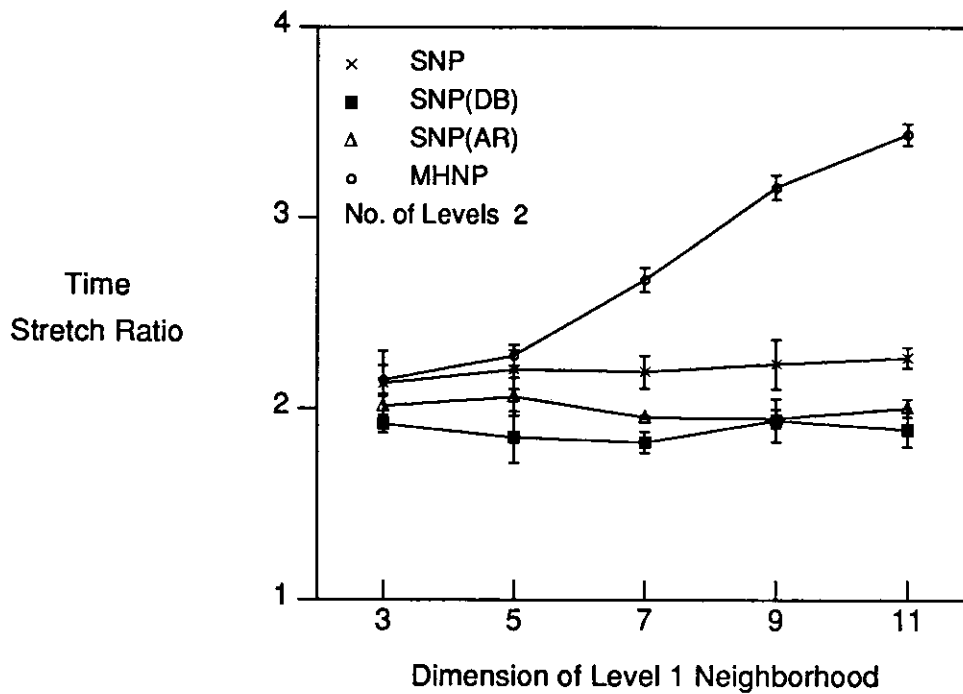
The version of SNP(AR) that we have simulated is the one where messages are routed towards the center of the lower level neighborhood of the destination process. The version of SNP(AR) where messages are routed to the closest node in the lower level neighborhood will however result in lower average message trajectory and time stretch ratios. Moreover in the simulation at each intermediate node of the message path the routing table is checked for a more recent routing entry for the destination process. In an optimized SNP(AR), routing tables in the path of a message are checked only at nodes on the boundaries of the neighborhoods of the destination process.



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.30 : Average Message Trajectory Stretch Ratio vs. Dimension of Level 1 Neighborhood

Figure 4.29 shows the average update bandwidth due to the migration of a single process versus the dimension of the first level neighborhood for a two level scheme. We first observe that the average update bandwidth for SNP and SNP(AR) is identical since the broadcast of updates takes place identically in the two schemes. As expected the average update bandwidth is the highest for SNP(DB) and the the lowest for MHNP.



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.31 : Average Time Stretch Ratio vs. Dimension of Level 1 Neighborhood

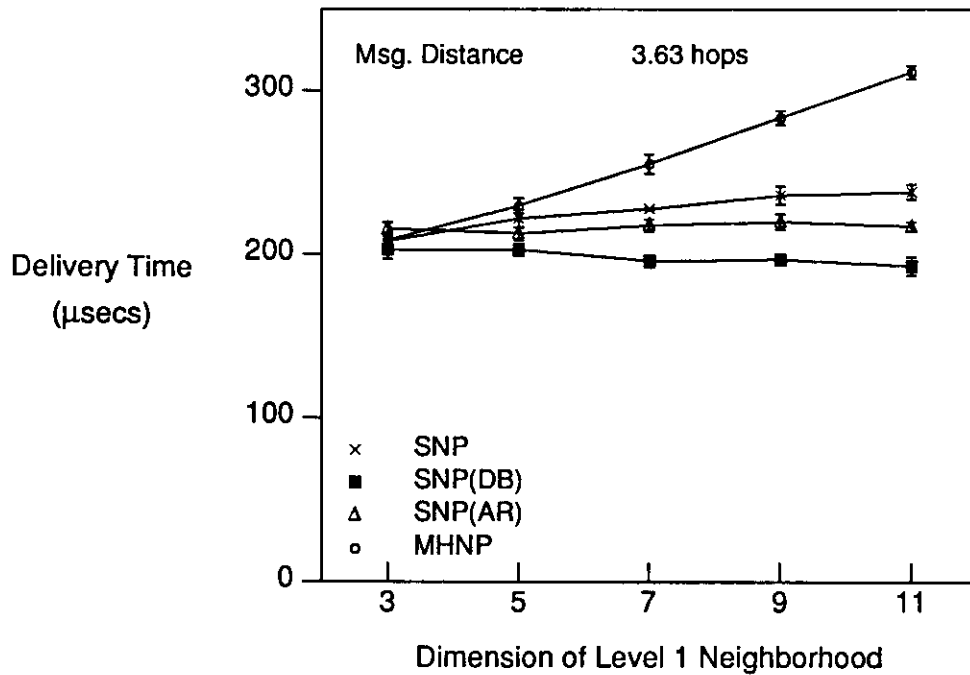
For 2-level SNP and 2-level SNP(AR) the dimensions of the lowest level neighborhood for which the update bandwidth is a minimum is

$$r_1 = 9$$

For a 2-level SNP(DB) the dimensions of the lowest level neighborhood for which the update bandwidth is a minimum is

$$r_1 = 5$$

For the MHNP the update bandwidth decreases with the increase in the size of the first



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.32 : Average Time to Deliver a Message vs. Dimension of Level 1 Neighborhood

level neighborhood. This is in accordance with the analytical expression of the update bandwidth of MHNP given in Equation 4.24.

The average update bandwidth for a 2-level SNP with optimal neighborhood size is 5540 ± 45 update_msg-hops/sec while for a 2-level SNP(DB) it is 8020 ± 397 update_msg-hops/sec which is a factor of 1.45 greater.

Figure 4.30 shows the average message trajectory stretch ratio versus the dimension of the first level neighborhood for a 2-level scheme. There are several interesting observations that we can make by observing the plots of Figure 4.30. We first observe that SNP(DB) has the lowest message trajectory stretch ratio of all the schemes. The average message trajectory stretch ratio when the update bandwidth is minimum, i.e. $r_1 = 5$, is 1.073 ± 0.008 which is extremely close to 1. Thus implementing SNP(DB) instead of SNP avoids the anomalies in the performance of SNP as discussed in Section 4.3 and results in stretch ratios almost equal to one. An indication as to why SNP(DB) has a much lower message trajectory stretch ratio than SNP can be found in simulation results that indicate that for SNP with optimal neighborhood size the fraction of messages that are forwarded are 0.28 while for SNP(DB) with optimal neighborhood size it is 0.16.

The second interesting observation to note is the significant improvement of the performance of SNP(AR) over SNP. We observe that Adaptive Routing is an extremely powerful tool that can significantly enhance the performance of SNP. From the simulation results we observe that the average message trajectory stretch ratio for SNP(AR) is almost constant at 1.104 as the dimension of the first level neighborhood is changed. In fact the average message trajectory stretch ratio for SNP(AR) is very close to that of SNP(DB) without the accompanying increase in update bandwidth.

The reduction in the update bandwidth of the 2-level MHNP with the increase in the dimension of the lowest level neighborhood as observed in Figure 4.29 is accompanied by an almost linear increase in the message trajectory stretch ratio (Figure 4.30). When the dimension of the lowest level neighborhood is increased from 3 to 13, the update bandwidth for MHNP decreases by a factor of 4.5 while the message trajectory stretch ratio increases by a factor of 2.7. It may be tempting to attempt to improve the message trajectory stretch ratio of MHNP by implementing Adaptive Routing. However this is unlikely to work since in the Moving Home Node Protocol only the Home node is kept updated and hence as a message approaches the destination process nodes on the path of the message will not have more recent information about the location of the destination process.

In Figures 4.31 and 4.32 we plot the average time stretch ratio and average delay for messages to be delivered versus the dimension of the lowest level neighborhood for a 2-level scheme. The important observation to make here is that the average delay and time stretch ratios for SNP(AR) are significantly lower than the delay and time stretch

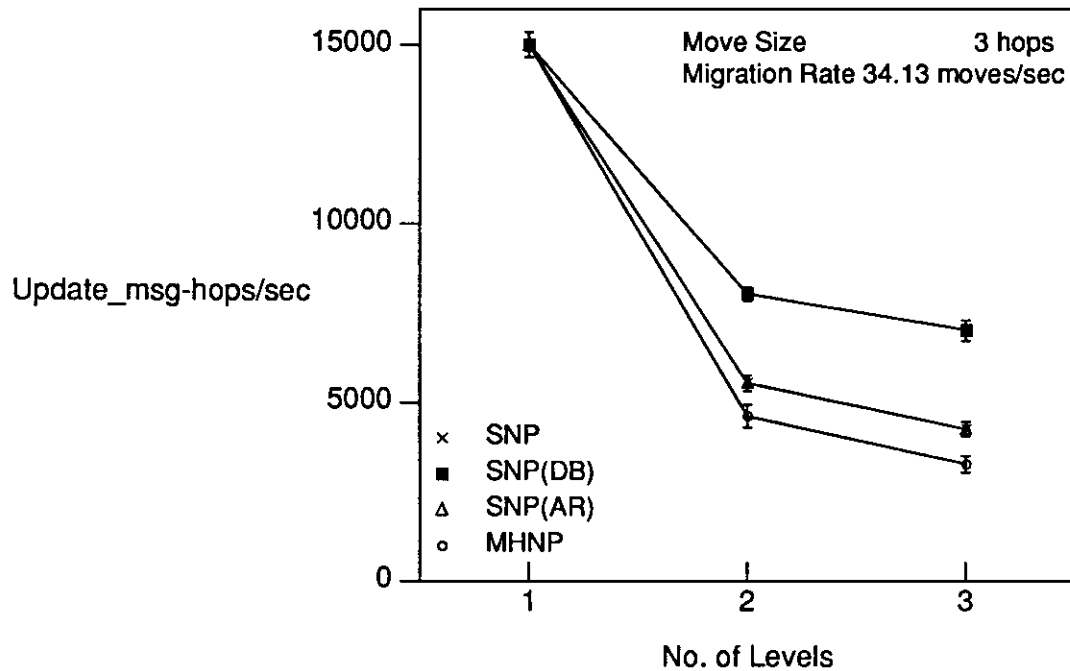
ratios for SNP.

Protocol	m = 2	m = 3
SNP & SNP(AR)	$r_1 = 9$	$r_1 = 5 \quad r_2 = 11$
SNP(DB)	$r_1 = 5$	$r_1 = 5 \quad r_2 = 7$
MHNP	$r_1 = 7$	$r_1 = 7 \quad r_2 = 13$

Table 4.9 : Optimal Neighborhood Dimensions for the Protocols

This is particularly notable since in our implementation of Adaptive Routing routing tables at each intermediate node on the path of a message are checked thus incurring a significant delay for accessing routing tables. An optimized version of SNP(AR) with routing of messages towards the closest node of the destination neighborhood and with adaptive routing only at nodes on the boundaries of the neighborhoods of the destination process will result in even better performance than that demonstrated here. Thus using Adaptive Routing for the Shifting Neighborhood Protocol results in a significant improvement of its performance.

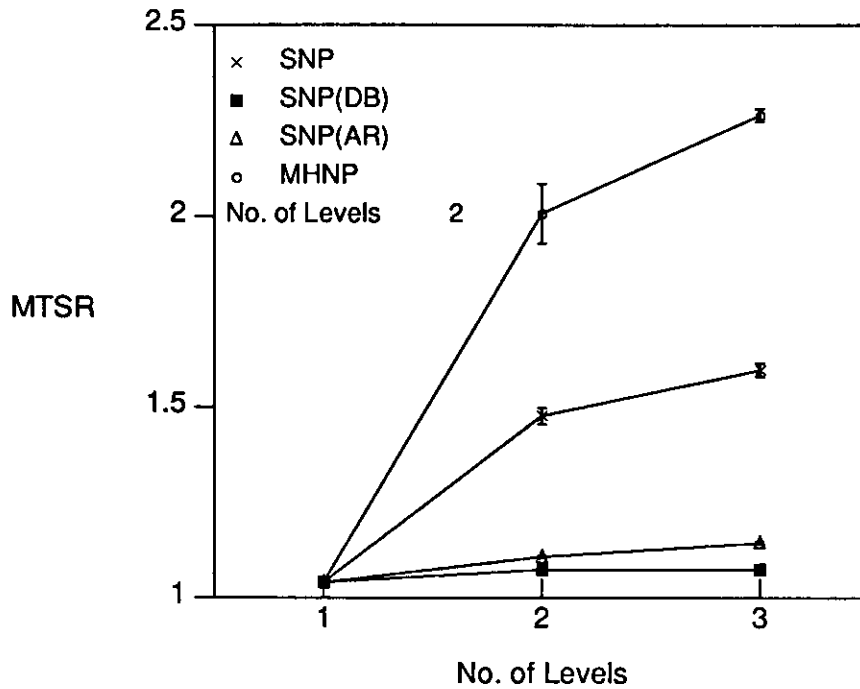
We now study the performance of the four protocols when the number of levels for the 21×21 system are increased from 1 to 3. The neighborhood dimensions for the 2-level and 3-level protocols are given in Table 4.9. The neighborhood dimensions for the protocols, except the dimension of the first level protocol for MHNP, are chosen from the simulations to be values that result in minimum update bandwidth. For the MHNP the dimension of the first level neighborhood is chosen independently based on other criteria.



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.33 : Average Update Bandwidth vs. No. of Levels in Protocol

If we observe the plot of Figures 4.33 we note that on going from two levels to three levels the relative reduction in the update bandwidth for each of the protocols is approximately the same. However the corresponding relative increase in the message trajectory stretch ratio (Figure 4.34) for MHNP and SNP are significantly higher than the increase in the message trajectory stretch ratio for SNP(DB) and SNP(AR). In fact the message trajectory stretch ratios for SNP(DB) and SNP(AR) are almost constant with the variation in the number of levels of the protocol. The minimum message trajectory stretch ratio achievable by any protocol is that achieved by the Complete Broadcast on Migration (CBM) protocol, and is equal to the message trajectory stretch ratio of the four protocols with a single level. Thus the message trajectory stretch ratio for SNP(DB) and

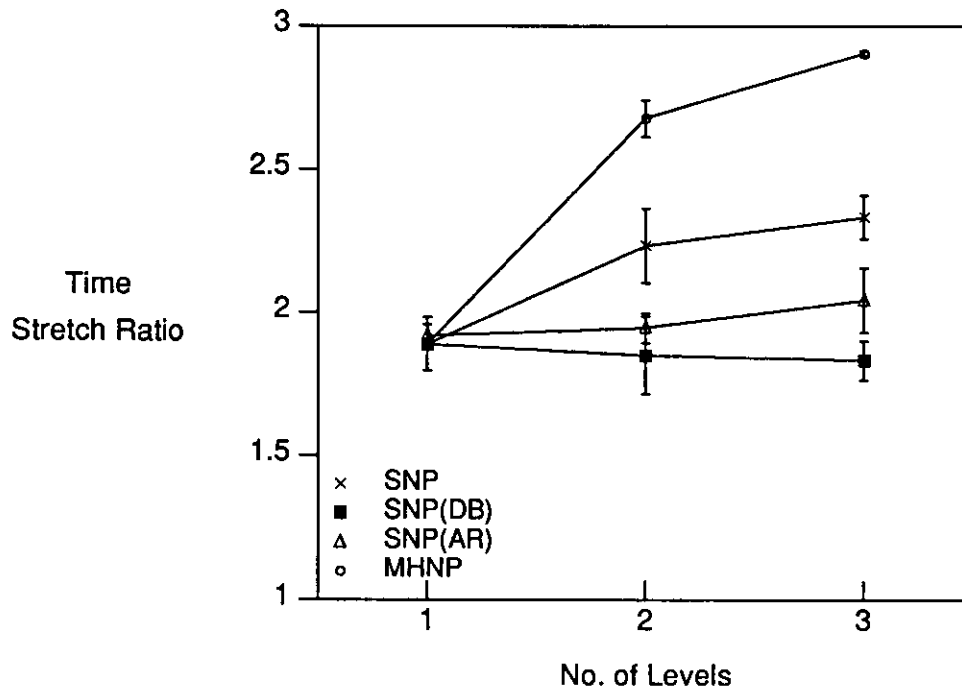


R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.34 : Average Message Trajectory Stretch Ratio vs. No. of Levels in Protocol

SNP(AR) with varying number of levels is very close to the minimum stretch ratio achievable.

Figures 4.35 and 4.36 plot the variation of the average time stretch ratio and the average time to deliver a message to a process as the number of levels for the four protocols are varied. As before note that the time stretch ratio and message delay for SNP(DB) and SNP(AR) are almost constant with the variation of the number of levels of the protocol. Note that for the 1-level SNP(AR) the time stretch ratio and delivery time are larger than for the other three protocols because of the additional delay incurred in SNP(AR) due to adaptive routing at intermediate nodes.

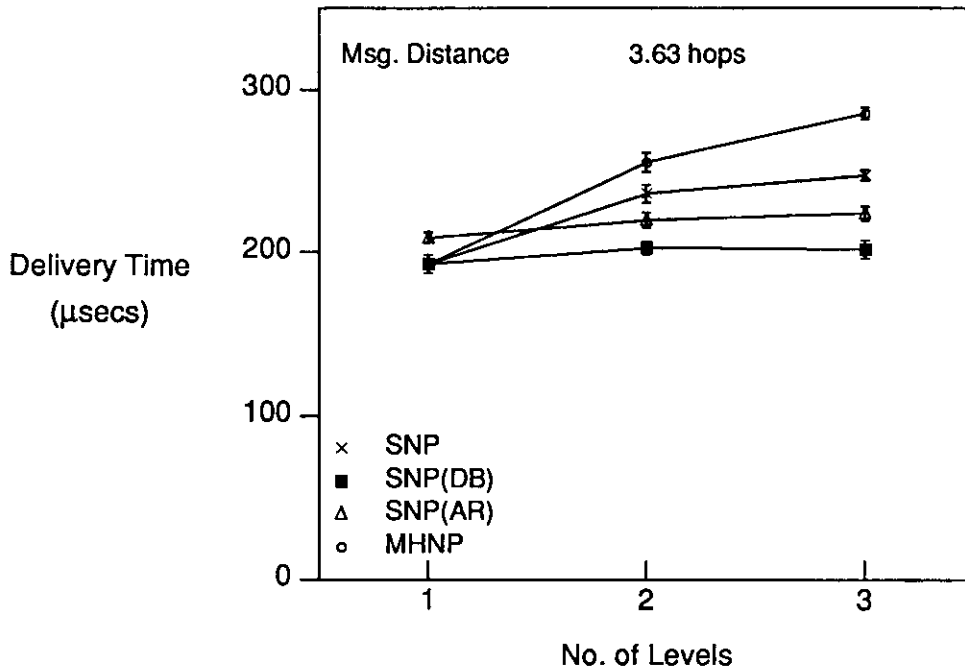


R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.35 : Average Time Stretch Ratio vs. No. of Levels in Protocol

We conclude that of the four protocols the performance of the the Shifting Neighborhood Protocol with Dynamic Broadcast SNP(DB) is the best and close to the best achievable performance by any possible protocol. However this performance is achieved at a higher update bandwidth compared to the other protocols.

The performance of the Shifting Neighborhood Protocol with Adaptive Routing SNP(AR) is close to that of SNP(DB) and is a particularly attractive protocol to implement because of the low update bandwidth in comparison to SNP(DB). An optimized version of SNP(AR), where messages are routed to the closest node of the neighborhood of the destination process instead of the center of a neighborhood and



R	21	p	0.33	$T_{MoveProc}$	500 μ secs
MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs

Figure 4.36 : Average Time to Deliver a Message vs. No. of Levels in Protocol

where messages are routed adaptively only at neighborhood boundary nodes, is likely to have performance matching that of SNP(DB).

The Moving Home Node Protocol MHNP may be chosen to be implemented if a higher time stretch ratio can be tolerated and the bandwidth available for control traffic is limited.

For the rest of this section we study the performance of SNP(DB) and SNP(AR) and how their performance is affected by the change in different system parameters.

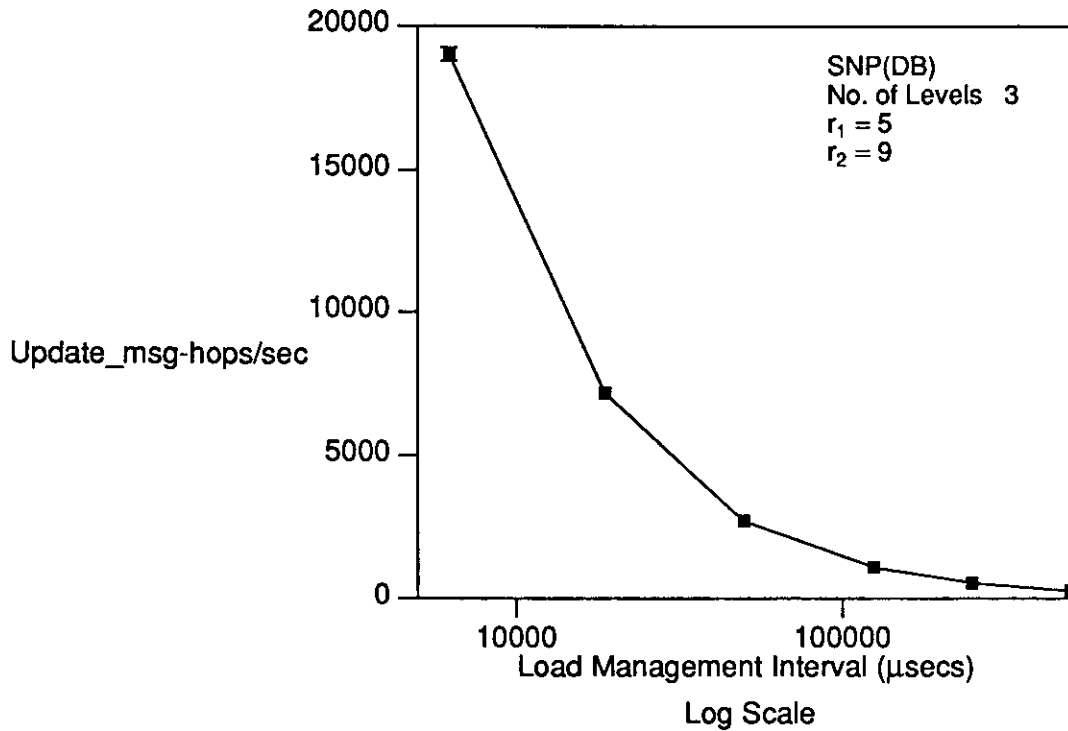
<i>R</i>	21	<i>LocalityBase</i>	0.6
<i>m</i>	3	<i>T_{UserMsg}</i>	25 μ secs
<i>r₁</i>	5	<i>T_{UpdateMsg}</i>	5 μ secs
<i>r₂</i>	9	<i>T_{MoveProc}</i>	500 μ secs
<i>MsgInterval</i>	250 μ secs	<i>T_{StopProc}</i>	100 μ secs
<i>DistMsgInterval</i>	Constant	<i>T_{StartProc}</i>	100 μ secs
<i>p</i>	0.33	<i>T_{RT}</i>	5 μ secs

Table 4.10 : Parameters Chosen for the Study of the Effect of the Variation of the Load Management Interval on the Performance of SNP(DB)

We first study the effect of varying the load management interval on the performance of SNP(DB). By varying the load management interval the rate of process migration is varied. Hence in this experiment we study the effect of the rate of process migration on the performance of SNP(DB). The parameters chosen for this set of simulations are given in Table 4.10.

The load management interval is varied from 6.25 msec to 0.5 sec, which results in an average of 25 messages to 2000 messages respectively received by a process in a load management interval. Note that the largest load management interval chosen by us is still too small for current operating systems due to the high cost of migrating a process. Our intention is to test the proposed protocols for the worst case i.e. when the rate of process migration is rapid.

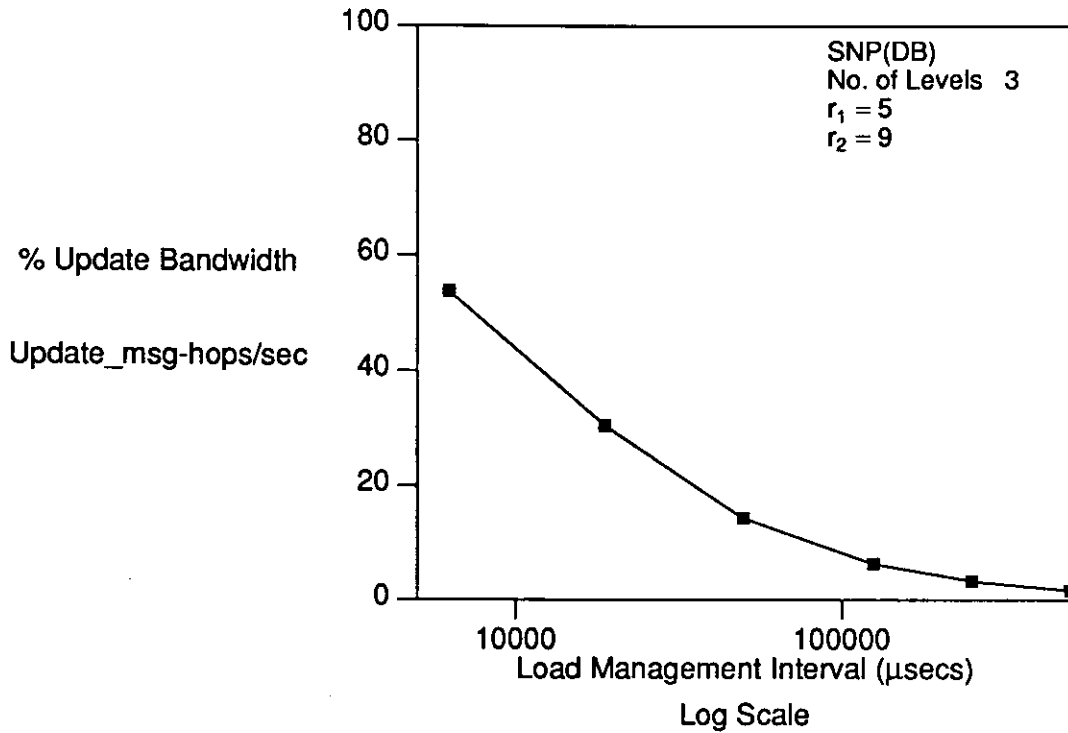
Figure 4.37 shows the variation of the average update bandwidth with the change in the load management interval. In Figure 4.38 we plot the percentage of the total message bandwidth that is due to updates versus the load management interval. The update bandwidth that we consider is the bandwidth resulting from the broadcast of routing updates due to the migration of a single process. The total message bandwidth is the sum of the bandwidths due to user messages and update messages. In calculating the traffic due to user messages we only consider application messages destined to one particular process. The load management interval is plotted on a log scale in both graphs.



R	21	LocalityBase	0.6	$T_{StopProc}$	100 μsecs
MsgInterval	250 μsecs	$T_{UserMsg}$	25 μsecs	$T_{StartProc}$	100 μsecs
DistMsgInterval	Constant	$T_{UpdateMsg}$	5 μsecs	T_{RT}	5 μsecs
ρ	0.33	$T_{MoveProc}$	500 μsecs		

Figure 4.37 : Average Update Bandwidth vs. Load Management Interval

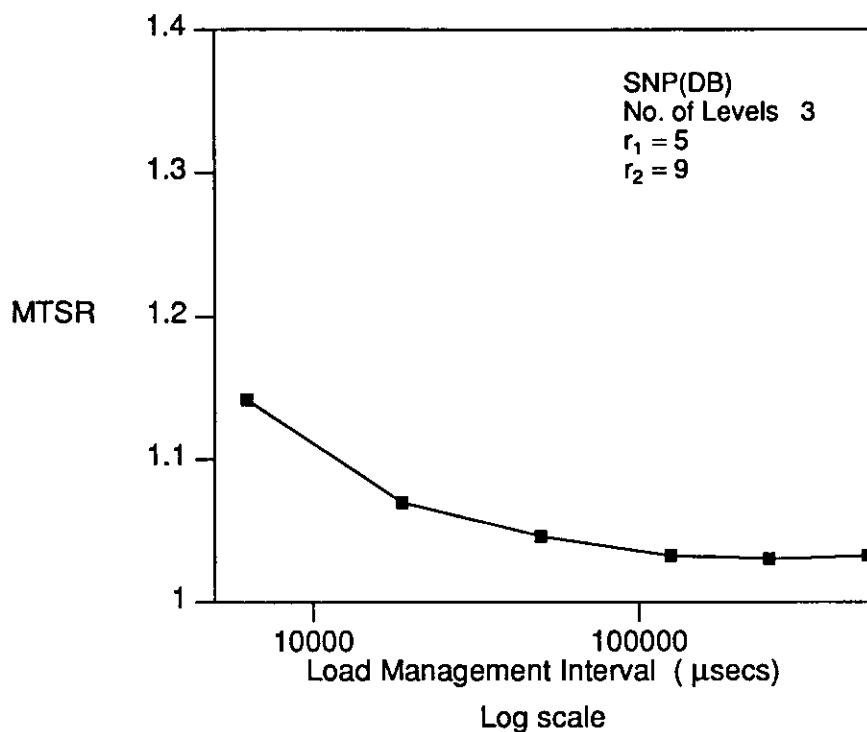
Figure 4.39 shows the corresponding variation of the average message trajectory stretch ratio with the load management interval. The load management interval is plotted on a log scale. Observe that the change in the message trajectory stretch ratio for SNP(DB) is small as the rate of process migration is varied. However as expected the update bandwidth is significantly higher at extremely rapid rates of migration.



R	21	LocalityBase	0.6	$T_{StopProc}$	100 μsecs
MsgInterval	250 μsecs	$T_{UserMsg}$	25 μsecs	$T_{StartProc}$	100 μsecs
DistMsgInterval	Constant	$T_{UpdateMsg}$	5 μsecs	T_{RT}	5 μsecs
ρ	0.33	$T_{MoveProc}$	500 μsecs		

Figure 4.38 : Percentage of the Total Bandwidth due to Update Traffic vs. Load Management Interval

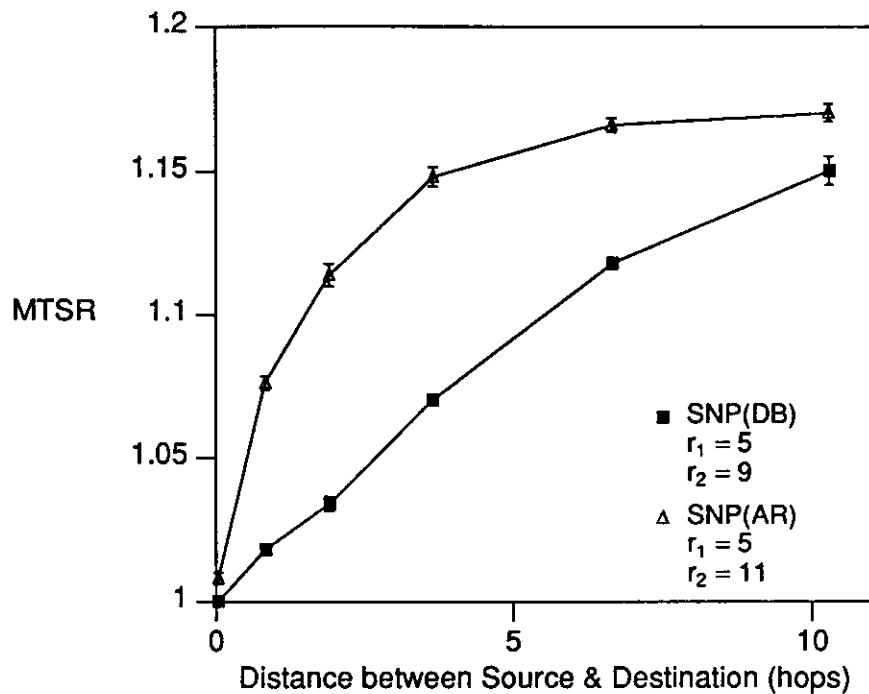
When the load management interval is 0.125 secs and the process receives an average of about 500 messages in every load management interval then the update bandwidth is 1080 ± 120 update_msg-hops/sec. This amounts to 6.19 ± 0.64 % of the total bandwidth. The corresponding average message trajectory stretch ratio is 1.032 ± 0.004 .



R	21	LocalityBase	0.6	$T_{StopProc}$	100 μsecs
MsgInterval	250 μsecs	$T_{UserMsg}$	25 μsecs	$T_{StartProc}$	100 μsecs
DistMsgInterval	Constant	$T_{UpdateMsg}$	5 μsecs	T_{RT}	5 μsecs
ρ	0.33	$T_{MoveProc}$	500 μsecs		

Figure 4.39 : Average Message Trajectory Stretch Ratio vs. Load Management Interval

We conclude that the Shifting Neighborhood Protocol with Dynamic Broadcast performs very well over a range of process migration frequencies. However at extremely high rates of process migration it is necessary to allocate larger link capacities to carry routing update traffic.



R	21	p	0.33	$T_{StopProc}$	100 μ secs
MsgInterval	250 μ secs	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UpdateMsg}$	5 μ secs	T_{RT}	5 μ secs
LoadMgmtInterval	18750 μ secs	$T_{MoveProc}$	500 μ secs		

Figure 4.40 : Average Message Trajectory Stretch Ratio vs. Mean Distance between Source & Destination at the time of the Sending of the Message

We now study the effect of changing the degree of locality of message communication between processes on the average message trajectory stretch ratios for SNP(DB) and SNP(AR). The locality base is varied from 0.01 to 0.99. With the locality base equal to 0.99 the communication model is approximately equivalent to the uniform probability model where the probability of a process sending a message to any process is equal. When the locality base is chosen to be 0.01 then the source and destination in most cases are allocated to the same node. The choice of different locality base parameters in the simulation manifests itself as the choice of different average distances between the

source and destination at the time a message is sent.

In Figure 4.40 we plot the average message trajectory ratio for SNP(DB) and SNP(AR) as the average distance between the source and destination at the time a message is sent is varied. We consider a 21×21 toroidal mesh with the average distance between the source and destination varying from 0.04 to 10.28 hops. The number of levels chosen for both the protocols is 3 and

For SNP(DB) $r_1 = 5, r_2 = 9$ and $r_3 = 21$

For SNP(AR) $r_1 = 5, r_2 = 11$ and $r_3 = 21$

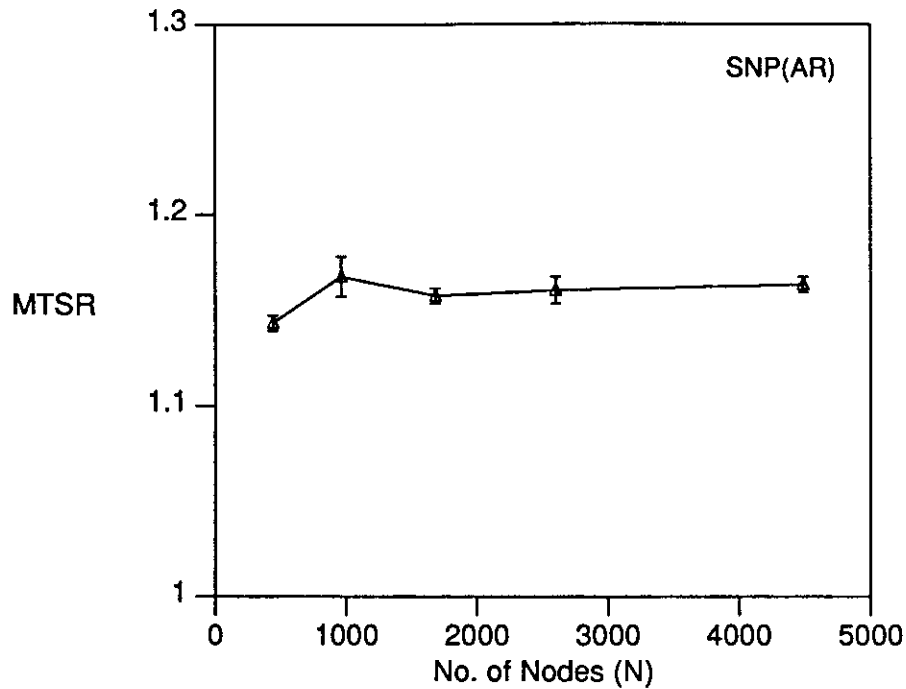
The average message trajectory stretch ratio for SNP(DB) is smaller than that for SNP(AR) over the range of degree of localities for communication. When there is no locality in communication the message trajectory stretch ratio for SNP(DB) is 1.15 ± 0.01 while for SNP(AR) it is 1.17 ± 0.006 . Hence we conclude that both protocols perform very well even when there is no locality in the communication between processes.

System Dimension	Nodes	No. of Levels	Neighborhood Dimensions
21	441	3	$r_1 = 5 \quad r_2 = 11 \quad r_3 = 21$
31	961	4	$r_1 = 5 \quad r_2 = 9 \quad r_3 = 17$ $r_4 = 31$
41	1681	5	$r_1 = 5 \quad r_2 = 9 \quad r_3 = 15$ $r_4 = 25 \quad r_5 = 41$
51	2601	5	$r_1 = 5 \quad r_2 = 9 \quad r_3 = 17$ $r_4 = 29 \quad r_5 = 51$
67	4489	6	$r_1 = 5 \quad r_2 = 9 \quad r_3 = 15$ $r_4 = 25 \quad r_5 = 41 \quad r_6 = 67$

Table 4.11 : Optimal Neighborhood Structure for SNP(AR)

Finally we study how the Shifting Neighborhood Protocol with Adaptive Routing scales up with the size of the system. The number of nodes is varied from 441 to 4489. The protocol parameters, i.e. the number of levels and the size of neighborhoods is derived from the analytical model to minimize update bandwidth. We assume the constant l to be equal to 3.2 as before. Table 4.11 lists the protocol parameters chosen for different sizes of the system.

For these set of simulations the average distance between the source and the destination at the time of sending a message is 3.73 hops. The average size of a move is about 3 hops and the average rate of migration is 34.13 moves/sec. The number of user messages received by the process is 4×10^3 msgs/sec.

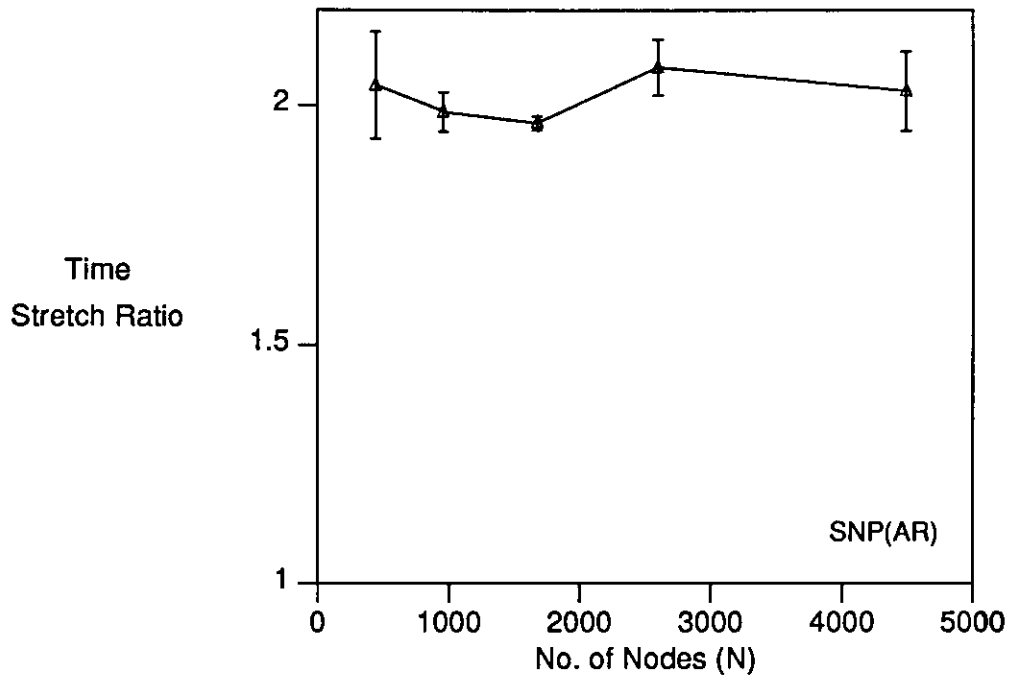


MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	$T_{MoveProc}$	500 μ secs
p	0.33	T_{RT}	5 μ secs		

Figure 4.41 : Average Message Trajectory Stretch Ratio vs. No. of Nodes for SNP(AR)

In Figures 4.41 and 4.42 we show the average message trajectory stretch ratio and the average time stretch ratio for SNP(AR) as the number of nodes is varied. Observe that both stretch ratios are relatively constant as the number of nodes N is increased.

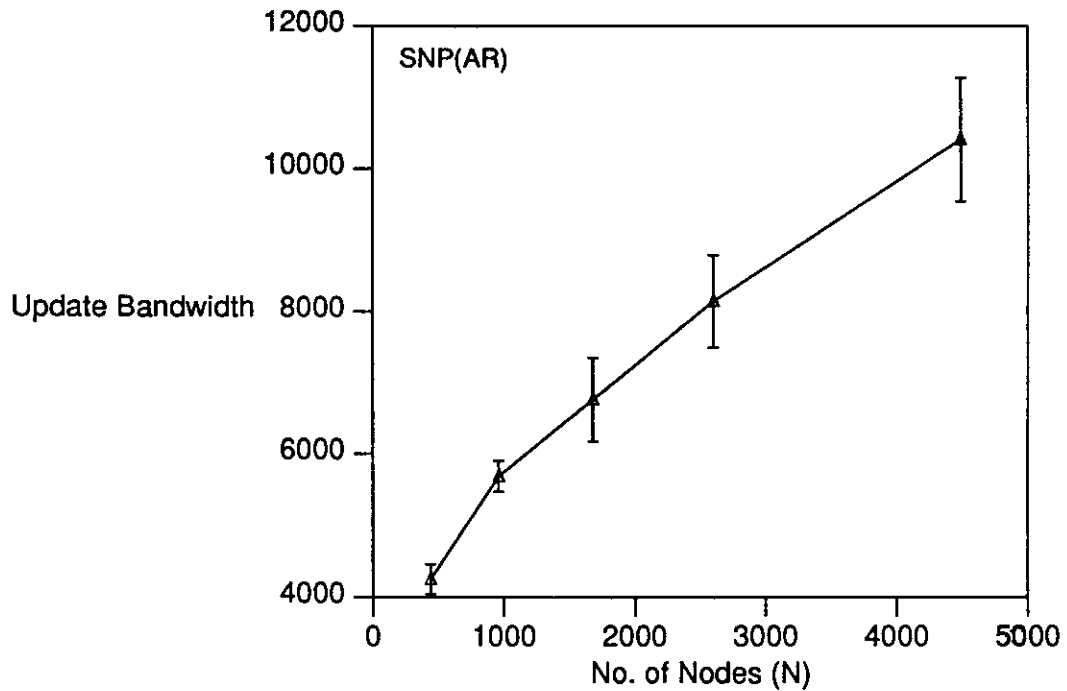
In Figure 4.43 we observe the change in the update bandwidth of SNP(AR) as the number of nodes is varied. The update bandwidth due to the migration of a single process is a slowly increasing linear function of the number of nodes. When the number of nodes is increased from 441 to 4489, i.e. a 10.2 fold increase, the average update bandwidth increases by a factor of only 2.45.



MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	$T_{MoveProc}$	500 μ secs
ρ	0.33	T_{RT}	5 μ secs		

Figure 4.42 : Average Time Stretch Ratio vs. No. of Nodes for SNP(AR)

Hence we conclude that the Shifting Neighborhood Protocol with Adaptive Routing scales up well with the size of the system. The stretch ratios remain constant with the change in the number of nodes while the update bandwidth is a slowly increasing function of the number of nodes.



MsgInterval	250 μ secs	LocalityBase	0.6	$T_{StopProc}$	100 μ secs
DistMsgInterval	Constant	$T_{UserMsg}$	25 μ secs	$T_{StartProc}$	100 μ secs
LoadMgmtInterval	18750 μ secs	$T_{UpdateMsg}$	5 μ secs	$T_{MoveProc}$	500 μ secs
ρ	0.33	T_{RT}	5 μ secs		

Figure 4.43 : Update Bandwidth vs. No. of Nodes for SNP(AR)

4.7.5 Conclusions from the Simulation

We have simulated the Shifting Neighborhood Protocol and related protocols operating in a large distributed system where processes are migrating rapidly. The effect of the variation of protocol and system parameters on the performance and overhead of the different protocols have been studied. The performance of the four protocols proposed in this chapter are compared and the conditions under which one protocol performs better than another are investigated.

For a two level SNP we study the change in the update bandwidth, the message trajectory stretch ratio, the time stretch ratio and the average message delivery time with the variation in the dimension of the lowest level neighborhood of a process. The convex shape of the update bandwidth curve obtained from the simulation is in accordance with the expression for update bandwidth derived from the analytical model of the previous section.

We obtain the optimal neighborhood dimensions for a multi-level SNP protocol that results in minimum update bandwidth. We observe that the choice of the number of levels in the protocol to be the optimal number of levels can result in a substantial saving in terms of update bandwidth. For a 21×21 system with appropriate system parameters we determine from the simulation that the optimal number of levels for SNP is three. The values of the update bandwidth obtained for SNP with different number of levels and the values for the optimal neighborhood structure for SNP are in close conformance with the values obtained from the analytical model.

In the comparison of the four protocols we make several interesting observations. SNP(DB) has the lowest message trajectory and time stretch ratios of all the schemes. The average message trajectory ratio for SNP(DB) is observed to be extremely close to 1 over a wide range of system parameter values. It is closely followed by SNP(AR) whose stretch ratios are very close to that of SNP(DB) without the accompanying increase in update bandwidth.

The Shifting Neighborhood Protocol may be implemented instead of SNP(AR) if the cost and additional implementation complexity of adaptive routing is very high. The update bandwidth for SNP is identical to that for SNP(AR). For a 21×21 system with reasonable system parameters and optimal protocol parameters we observe that the message trajectory stretch ratio for SNP even though greater than that for SNP(AR) is close to 1.5.

MHNP is the lowest cost protocol of the four protocols proposed by us. When the rate of process migration is not rapid or when the capacities of communication links are restricted then it would be preferable to implement MHNP. However accompanying the reduction of update bandwidth in MHNP is the increase in the stretch ratio for MHNP which is significantly larger than the value of the stretch ratio for the other protocols.

We study the effect of the variation of the rate of process migration on the performance of SNP(DB). We observe that the protocol performs well over a range of rates of process migration. At extremely rapid rates of process migration while the message trajectory stretch ratio is still close to 1 the update bandwidth increases considerably, thus indicating the need for the allocation of larger communication link capacities when process migration rates are very high.

We compare the performance of SNP(DB) and SNP(AR) when the degree of spatial locality in communication is varied. While the message trajectory stretch ratio for SNP(DB) is consistently lower than that of SNP(AR), both protocols perform very well even when there is no spatial locality.

Finally we study the performance of SNP(AR) with the increase in the size of the system. As predicted by the analysis we observe that the value of the message trajectory stretch ratio is independent of the size of the system. The update bandwidth is a slowly increasing function of the number of nodes. This confirms that SNP(AR) is *scalable* with the size of the system.

CHAPTER V

Other Issues

5.1 Locating Remote Routing Entries

In a distributed system if routing entries for a process are stored at each of the N nodes of the system then $N * P$ storage entries will be required to keep routing information for P processes. When the number of nodes N is very large then it may not be possible to keep routing entries for each process at every node because of the prohibitive size of memory required for routing tables at each node. In this section we investigate methods for efficient access of routing entries for a process from any node, when routing entries for a process are stored only in select nodes.

We consider two mechanisms for locating remote routing entries:

- Wave Searching
- Hashing

5.1.1 Wave Searching

When the operating system at a node requires a particular routing entry which is not in the local routing table and the operating system does not know where that routing entry may be found, then the node has to begin searching other nodes. This can be done by *broadcasting* a request for the routing entry and waiting for a response. The problem with broadcasting however is that once a routing entry is obtained it is difficult to terminate the ongoing broadcast. To address this problem we introduce a controlled broadcast scheme called *wave searching*.

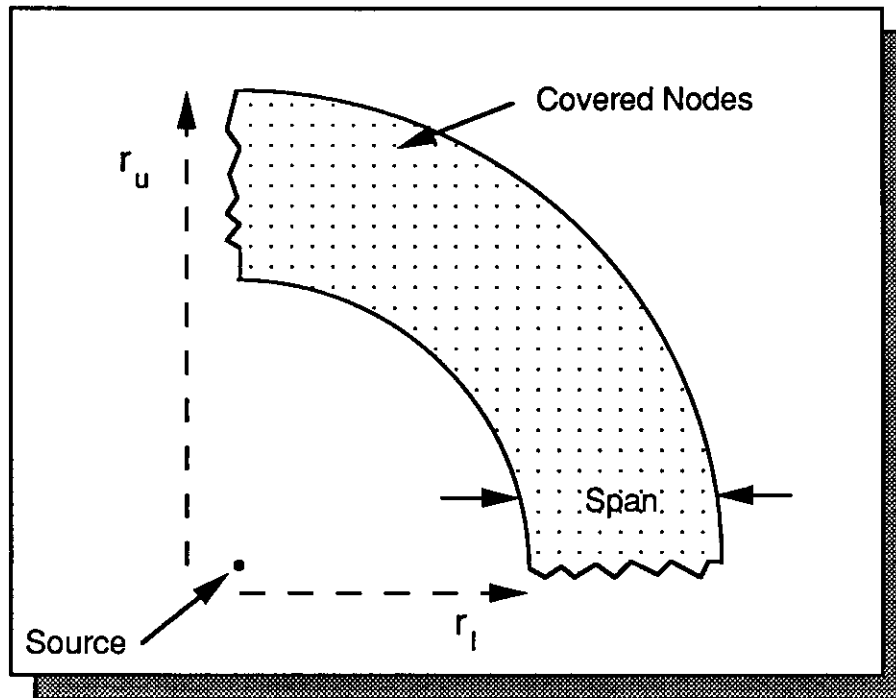


Figure 5.1 : Bounded Broadcast

Wave searching can be viewed as a wavefront of queries originating at the source node and expanding till the entry being searched for is located. The expansion of the wave takes place in discrete steps and is controlled at the source of the broadcast. A wave search originating at node U is equivalent to a controlled breadth first search of the network with U as the center.

Wave search consists of a series of *bounded broadcasts* from the source node; each bounded broadcast followed by a *wait time* when the source waits for a response indicating whether the routing entry is found or not. A bounded broadcast with *lower bound* r_l and *upper bound* r_u , is a broadcast of messages to all nodes that are at a distance between r_l and r_u from the source of the broadcast. The nodes whose distance from the source is between r_l and r_u are said to be *covered* by a bounded broadcast with bounds r_l and r_u . The difference $(r_u - r_l)$ between the bounds is the *span* of the bounded broadcast. A bounded broadcast is illustrated in Figure 5.1.

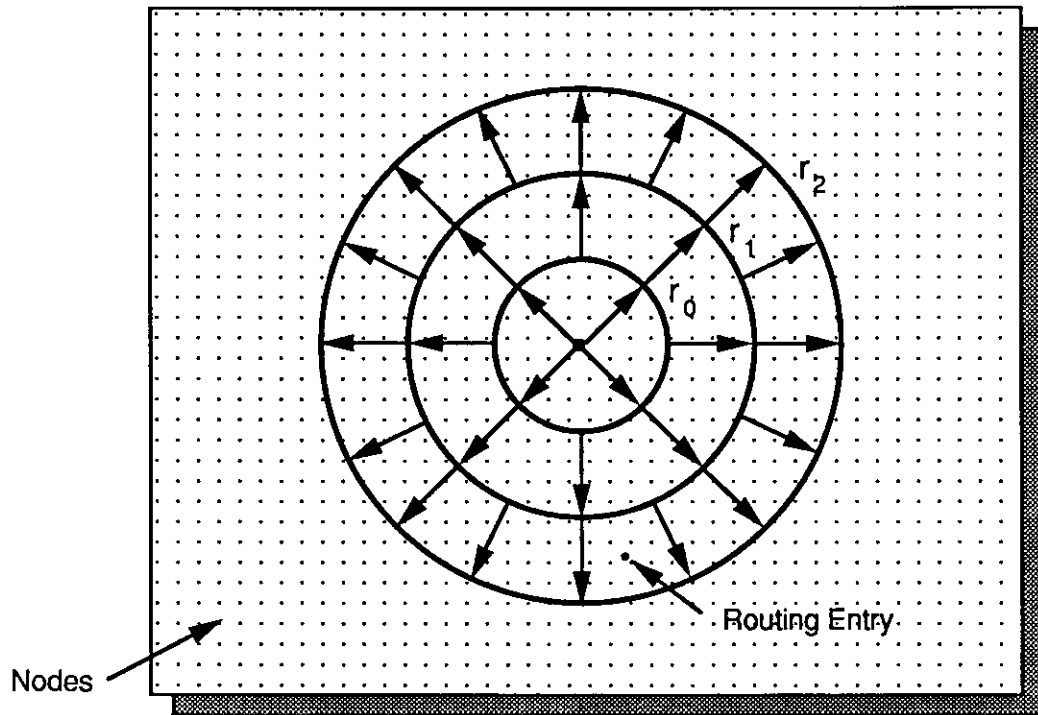


Figure 5.2 : Wave Search

Wave search is done as follows. Initially a bounded broadcast is initiated at the source node U and covers all nodes within a distance r_0 hops from U . Node U then waits for a response to the broadcast request, and if the routing entry is not found then it sends broadcast requests to nodes at distance between r_0 and r_1 hops from U . The node U continues to broadcast larger waves till the required routing entry is received, and the condition for termination of the wave search is satisfied. Wave search is shown in Figure 5.2.

The concept of wave search was suggested by Boggs [Boggs 83] for searching for a resource in an internet. The communication cost of wave search has been analyzed by Ahamad [Ahamad 87b], for various probability distributions (exponentially decreasing, linearly decreasing and uniform) of finding the location information as the distance from the source of the broadcast increases. Chlamtac [Chlamtac 87] has proposed a wave expansion algorithm in the context of collision free broadcasting in multi-hop radio networks. Hillis [Hillis 81] considered the problem of terminating broadcasts and

suggested *cancellation messages* that propagate faster than *requests* and hence catch up and cancel the propagating request.

Bounded broadcasts can be implemented by using a *shortest path spanning tree* ([Dalal 77], [Dalal 78] & [Wall 80]) that covers all the nodes. A broadcast message has a hop count field that is incremented when it is received by a node. Messages that arrive at a node are sent out on all other edges of the spanning tree connecting that node except the one it arrived on. Messages with a hop count greater than or equal to the upper bound of the bounded broadcast are not propagated. Every node receives a message only once which makes this method efficient in terms of communication bandwidth costs. This scheme is similar to one proposed by Kung and Shacham [Kung 87].

Bounded broadcast to nodes that are at a distance within distance r_l and r_u hops from the source node using a spanning tree is done as follows:

1. A message BMSG with fields r_l , r_u , $hop_count = 0$ and containing a request for the routing entry for process q taken is sent from the source node to all its neighbors on the spanning tree.
2. When a node receives a BMSG, the hop_count is incremented by one and if the hop_count is between r_l and r_u then the local routing table is checked for an entry for q . If the hop_count is less than the r_u and the routing entry is not found then the BMSG with the updated hop_count is propagated to all neighbors on the spanning tree except the predecessor node that sent the BMSG. If the hop_count is equal to the r_u or the routing entry for q is found then a BMSG_ACK message is returned to the predecessor node that sent the BMSG. If the routing entry for q is found locally then the BMSG_ACK contains the routing entry.
3. When an intermediate node receives a BMSG_ACK message the operating system updates the number of BMSG_ACK messages received by that node. If the number is equal to the number of spanning tree edges incident on the node minus one, then a BMSG_ACK message is sent to the predecessor node that sent this node the BMSG message. If any of the BMSG_ACK messages received at the intermediate node contained a routing entry then the BMSG_ACK message sent from this node also contains the routing entry.
4. When the source node receives a BMSG_ACK, the operating system updates the number of BMSG_ACK messages received by the source node. If the number is

equal to the number of spanning tree edges incident on the source node, then the operating system at the source node detects that the bounded broadcast is complete. If any of the BMSG_ACK messages received by the source node contain a routing entry then an entry for process q has been obtained from nodes that are at distance between r_l and r_u hops from the source node.

In irregular topologies, the edges of the shortest path spanning tree that are incident on a node have to be stored at that node. Unfortunately, the shortest path spanning tree for different nodes of an irregular topology need not be the same, resulting in $O(N^2)$ space for storing the links of the shortest path spanning trees, where N is the number of nodes. An alternative is to select a single spanning tree with "good" delay properties. The broadcast delay of a shortest path spanning tree rooted at the *center* of a graph has been shown by Wall ([Wall 80]) to be bounded by a small constant factor times the minimum delay. Distributed algorithms for locating the center of a graph have been discussed by Korach et. al. [Korach 84]. This strategy requires $O(N)$ storage space for the entire network.

In many regular topologies, the edges of the shortest path spanning tree can be determined algorithmically, thus avoiding the need for storing the tree.

The problem with using spanning trees for broadcast is that the failure of any node or edge of the spanning tree, partitions the tree. One solution is to switch to backup trees that do not include the failed link or node. In regular topologies it may be possible to route messages around the failure if it is detected by the neighboring node. An alternative to using spanning trees for bounded broadcast is to use the Propagation of Information with Feedback (PIF) protocol proposed in [Segall 83]. The PIF protocol consists of the flooding mechanism to propagate messages and a feedback mechanism that conveys any results of the message propagation and indicates the termination of the protocol. A *hop-count* field can be used to bound the broadcast. The disadvantage of this approach however is that a node receives a number of copies of a message resulting in large communication bandwidths and high costs of processing messages.

5.1.2 Hashing

We consider an alternate mechanism that requires much lower communication bandwidth compared to wave searching and which relies on a hash or mapping function that maps a process name to a set of addresses of nodes. These set of nodes are required to have a routing entry for that process. Hence the location of routing entries for a process are encoded in the name of a process and any node can derive the set of node addresses that have a routing entry for a particular process.

We assume the existence of a hash or mapping function h such that the application of the function to a process name q results in a set of node addresses that are uniformly distributed in the network. The hash function is known to all nodes and the operating system at any node can derive the set of nodes given by $h(q)$. We also assume that any node U can evaluate $DIST(U, V)$ the distance between U and an arbitrary node V . Thus the operating system at a node U can determine which of the k nodes given by $h(q)$ are closest to node U . These are the k -closest nodes to U that belong to $h(q)$.

Initially when a process q is created, routing entries for q are installed at all nodes belonging to $h(q)$, and contain the location of the process and the initial migration-count zero. When q migrates away from node U , the k -closest nodes in $h(q)$ are sent updates indicating the new location and migration-count of q . When a process at a node V requires a routing entry for q , a request for a routing entry is sent to the node closest to V in $h(q)$.

The value of k is chosen based on reliability requirements. The minimum value of k that can be chosen is one.

The choice of an appropriate hash function is dependent on the topology and the scheme for naming processes and addressing nodes. Ideally the location of the m nodes whose addresses are given by the application of the hash function on a process name should be such that the shortest distance from any node to the closest of the m nodes should be minimized.

Unlike the wave searching scheme that can be applied to any topology the hashing scheme can be applied only to select regular topologies for which appropriate hash functions can be found.

5.2 Reliability Issues

The protocol proposed by us for routing messages to moving processes in large distributed systems, i.e. the Shifting Neighborhood Protocol (or one of its variations) with caches, is designed to be robust to failures in the distributed system. The robustness of our protocol is derived from three features:

- Propagation of routing updates when a process migrates
- Association of a *migration-count* field with a process
- Routing function distributed uniformly over all nodes

In the Basic Routing and Migration Protocol when a process q migrates from node U the routing entry for q at U is updated and the nodes belonging to $\&surrogate(U, q)$ are also updated. After the completion of the migration of process q and the updating of the surrogate set if the node U fails then an up-to-date routing entry for q can be obtained from the surrogate set $\&surrogate(U, q)$. The size k of the surrogate set $\&surrogate(U, q)$ is chosen based on reliability requirements and the surrogate set is chosen to be the location of the k -closest routing entries for q from U . In Section 5.1 we discussed two alternate ways for choosing the surrogate sets. In the hashing scheme any node can determine the set $\&surrogate(U, q)$ while in the wave search scheme a node adjacent to U can easily obtain a routing entry belonging to $\&surrogate(U, q)$ by initiating waves of bounded broadcasts. Thus if a process q has migrated away from node U and if node U fails, then a message for q that is routed to U can be forwarded by a node adjacent to U . The ability to forward messages that are routed to a node, that is a previous location of a process, even on the failure of that node is because when a process migrates, routing entries at a surrogate set of nodes are updated in addition to the local routing entry and the surrogate set of nodes are chosen to be nodes that are nearby the node to which messages are routed. Thus in contrast to conventional forwarding address schemes a process does not become inaccessible when a node, that is a previous location of the process, fails.

The Shifting Neighborhood Protocol and related protocols are extensions of the Basic Routing and Migration Protocol with the additional feature that updates are broadcast to a neighborhood around the location of a process on the migration of a process. The Shifting Neighborhood Protocol is in contrast to proposed protocols

([Fowler 85] & [Fowler 86]) where routing entries are updated as a side effect of routing a message to a process. In our protocol, information about a process migration is also broadcast to a neighborhood of the process at the time of process migration while in Fowler's protocol the updating of routing information is done only when the process is accessed. The advantage of using a scheme that *actively* informs other nodes about the migration of a process when the process migrates is that information about process migration is disseminated in the system immediately and hence the failure of a single node does not result in the loss of this information.

The protocols we have proposed make use of a *migration-count* associated with a process that is incremented whenever the process migrates. Routing entries for a process also have a *migration-count* field associated with it. Messages that are routed based on a certain routing entry can also carry the *migration-count* field of the entry. In the BRM protocol we have shown that when a message is re-routed at a particular node, as long as the re-routed message has a larger *migration-count* than the incoming message before re-routing, the message will be delivered to the process. Since the message and all routing entries for the process carry the *migration-count*, it is relatively easy to guarantee in the presence of failures that the message is re-routed using a larger migration-count than the one it already has. In the presence of failure of nodes if the operating system needs a remote routing entry to re-route a message and if wave search is used to locate remote routing entries then a wave search can be initiated to search for routing entries for the destination process of the message. The wave search is terminated only when an entry with a higher migration-count than the one in the message is obtained. If the hashing method is used to locate remote entries a similar approach can be used.

The Shifting Neighborhood Protocol and related protocols are designed so that the all nodes in the system have an identical routing function. If the rate at which different processes receive messages is almost equal then even though different nodes have routing entries for different processes, all the nodes will have approximately equal processing load due to routing. Thus unlike several hierarchical routing schemes there are no special nodes that perform special routing functions which are vulnerable to failure.

To protect the content of routing tables and messages from alteration due to noise we can use error detecting and correcting codes.

If a node on which a process is executing abruptly fails the the process fails along with it. We can increase the availability of processes by replicating them and the operating system can perform the necessary updating to keep the state of the copies consistent if it is necessary. We can also checkpoint the process so that the process state is stored at alternate node sites or reliable storage sites periodically. If the process fails because of the failure of the node on which it is resident then it can be re-started at an alternate node.

The Basic Routing and Migration protocol can be modified to make it more reliable so that the chances of the success of process migration are improved even if the the location from which a process migrates fails during process migration.

5.3 Unbounded Migration-Count

Our protocols makes use of unbounded sequence numbers for the migration-count of a process. The migration-count field is contained in messages, routing table entries, and the process control block.

The migration-count for a process is incremented whenever a process migrates. In our protocols the migration-count field associated with routing entries for a particular process is compared to determine which routing entry for the process is most recent. When a routing entry for a process is updated in our protocols, it is always replaced with more recent information about the location of the process. For the adaptive routing of messages or for the routing of messages in the presence of node failures, the migration-count field in messages is essential to guarantee that messages are re-routed to a more recent location of the process.

Since the migration-count field associated with a process increases monotonically hence the number of bits chosen to represent the migration-count variable has to be large enough to hold the number of migrations of the process during its lifetime.

However if the migration-count field for a particular process overflows, the operating system can reinitialize the migration-count of the process by sending a broadcast message to the entire system. Additional mechanisms are required to guarantee that application messages for the process that are in transit when a migration-count re-initialization message is broadcast are reliably delivered to their destinations.

CHAPTER VI

Conclusions and Future Research

6.1 Conclusions

In this dissertation we have investigated protocols for routing messages to migrating processes in large distributed systems. We develop a decentralized routing scheme with a routing table at each node consisting of routing entries for processes. Routing tables may be *out-of-date* and *incomplete*. Our main objective has been to develop protocols that *scale* up with the size of the system.

We first present a Basic Routing and Migration (BRM) protocol which captures the fundamental synchronization and correctness issues associated with process migration and routing messages to migrating processes. In the BRM protocol it is not necessary for every node to have a routing entry for every process. If a routing entry for a particular process is required by the operating system at a node and is not present in the local routing table then it is obtained from the routing table at a nearby *surrogate* node.

The routing of messages to a particular process is based on *re-routing* or *forwarding* the messages to a more recent location of the process.

The BRM protocol is shown to be deadlock free and we prove that messages are eventually delivered to their destination processes. The correctness argument is based on the following properties of BRM: First, if a routing entry for a process is updated it is always replaced with more recent information about the location of the process. Second, when a message is *re-routed* or *forwarded* at a node because the destination process is no longer located at that node, it is sent to a more recent location of the process. Finally we assume that on the average messages travel at a faster rate than a process does. For extensions to the BRM protocol we ensure that these properties continue to hold so that the extended protocol is also correct.

In order for the operating system at a node to obtain remote routing entries we have proposed two techniques - wave searching and hashing. Wave searching can be viewed as a wavefront of queries originating at the source node and expanding till the entry being searched for is located. In the hashing scheme the location of routing entries for a particular process can be derived by applying a hash or mapping function to the process name. The hashing function is known to all the nodes. The hashing scheme is preferable to the wave search scheme because of the significant difference in the number of messages required by the two schemes to locate a routing entry for a particular process. The hashing scheme however is only applicable to regular topologies for which a suitable hash function is known. The delay for a node to obtain a remote routing entry for both schemes however is orders of magnitude larger on current hardware than the time to access a local routing entry.

To reduce the probability of accessing remote routing entries we propose a *routing table hierarchy* consisting of a local cache of routing entries, the local routing table, and remote routing tables. While entries in the local routing table are permanent, the contents of local caches adapt to the pattern of communication between processes. Routing entries that are repeatedly used are retained in the cache thus reducing the number of requests for remote routing entries.

Another purpose of the proposed caching schemes is to keep local entries up-to-date. When a message after being re-routed reaches its destination process, the location of the source process is sent an update indicating the latest location of the destination process. Moreover processes that repeatedly send messages to a destination process can be sent immediate updates when the destination process moves.

We propose three schemes to incorporate caches in the BRM protocol - the Process Caching Scheme with placement on Routing Fault (PCS(RF)), the Process Caching Scheme with placement on Cache Miss (PCS(CM)) and the Node Caching Scheme (NCS). In PCS(RF) and PCS(CM) caches are associated with a process and are moved along with the process when the process moves, thus avoiding a *cold start* of the process with empty caches after migration. In PCS(RF) an entry is added to the cache when there is a *routing fault*, i.e. the routing entry is not found locally. In PCS(CM) an entry is added to the cache when there is a *cache miss*, i.e. the routing entry is not found in the cache. Thus in PCS(CM) the cache contains entries for all processes to which messages have been sent recently. While in PCS(RF) memory space is effectively utilized by avoiding duplication of entries in the process cache and the local routing

table, the rate of routing faults increases slightly after a process migrates. This is because some entries which were frequently accessed and were located in the routing table at the previous location of the process are not in the local routing table at the new location of the process. In PCS(CM) the rate of routing faults is independent of the rate of process migration.

In NCS caches are associated with a node and shared by all the processes located at that node. The advantage of NCS is that there is no duplication of entries for a particular process at a node and the available memory space is effectively utilized. Moreover the overhead for maintaining the node cache is much smaller than for process caches. One disadvantage of NCS is that it is difficult to maintain fairness in the use of the node cache by processes on a node, particularly when processes have different rates of communication. The main disadvantage of NCS however is that when a process moves to a new location, no cache entries are moved with it, resulting in high rates of routing fault after migration.

We develop an analytical model for the performance of PCS(CM) and evaluate the average time to obtain a routing entry. The performance of the three caching schemes are compared by simulating the routing hierarchy for the three schemes. From the simulations we conclude that in a majority of the cases

$$T_{PCS(RF)} < T_{PCS(CM)} < T_{NCS}$$

where $T_{PCS(RF)}$ is the average time to find a routing entry for PCS(RF) etc. We observe from the simulations that the effective use of memory space by avoiding duplication of entries between the cache and local routing table in PCS(RF) has a greater impact on the time to obtain a routing entry than the advantage in PCS(CM) of moving *all* the recently used entries along with the process to the new location of the process thus avoiding an increase in the rate of routing faults soon after migration. We observe that on the average the time to obtain a routing entry for PCS(RF) is significantly better than PCS(CM) except when cache sizes are very large, the rate of process migration is very rapid, or the degree of locality in communication is very large. In these cases PCS(CM) performs slightly better than PCS(RF).

As expected NCS does not perform very well. The change of the rate of process migration has a pronounced effect on the average time to find a routing entry under NCS. However when processes migrate infrequently it is preferable to implement NCS since it is easier to implement and the procedures for updating the routing table and cache are simple.

The main contribution of our work is the Shifting Neighborhood Protocol (SNP) and related protocols for updating routing information in large distributed systems. SNP is an extension of the BRM protocol with the addition of the broadcast of routing updates to a limited region in the system on the migration of a process. Thus while the caching schemes make repeated communication between processes efficient, the Shifting Neighborhood Protocol is a scheme to make first time communication between two processes efficient.

The Shifting Neighborhood Protocol is a *multi-level* protocol where associated with a process are a family of nested neighborhoods covering the nodes of the system. In SNP routing entries located at nodes in different neighborhoods are updated with different frequencies depending on the location of the process. The neighborhoods of a process are not fixed but are dynamically reconfigured as the process moves. The multi-level organization serves to reduce bandwidth in our protocol.

The motivation for SNP is that distant nodes need not know about small perturbations in the location of a process because the relative increase in the path length that a message from a distant process has to take is small compared to the actual distance from the source to the destination.

We present three other variations of the Shifting Neighborhood Protocol (SNP) - the Shifting Neighborhood Protocol with Dynamic Broadcast (SNP(DB)), the Shifting Neighborhood Protocol with Adaptive Routing (SNP(AR)) and the Moving Home Node Protocol (MHNP). The four protocols differ in their performance and overhead and have different stretch ratios and update bandwidths.

We present an analytical model for the performance of SNP and related protocols and analytically determine the optimal neighborhood structure, i.e. the number of levels and the size of corresponding neighborhoods, that results in minimum update bandwidth. The migration of a process is modeled as an unrestricted random walk. The underlying topology is assumed to be a large square toroidal mesh. To obtain the optimal

neighborhood structure that minimizes update bandwidth the number of levels and the dimension of neighborhoods are assumed to be real valued. To calculate the average message trajectory stretch ratio we assume that the time for updates to be broadcast is small compared to the time between successive migrations of a process. In the analysis of SNP, to calculate the average message trajectory stretch ratio we assume the *uniform probability* model of communication between processes where a process sends a message to any process in the system, irrespective of its location, with equal probability.

If the number of levels in SNP is fixed to be 2, and the neighborhood sizes are chosen to be optimal then the average update bandwidth due to the migration of a single process is $O(R)$ update_msg-hops/sec, where R is the side of the square toroidal mesh. This is a significant improvement over the update bandwidth of $O(R^2)$ update_msg-hops/sec for the protocol where updates are broadcast to the entire system every time a process moves. The corresponding message trajectory stretch ratio for a 2-level SNP is 1 plus a term that is $O(1 / \sqrt{R})$.

The optimal number of levels for SNP that minimizes update bandwidth is $O(\log R)$. With the choice of the optimal neighborhood structure we obtain the average update bandwidth due to the migration of a single process to be $O(\log R)$ update_msg-hops/sec. For SNP with an optimal neighborhood structure we determine from the analysis that the average stretch ratio does not depend on the size of the system. Since the average update bandwidth of SNP is a slowly increasing function of the size of the system and the average stretch ratio is independent of the size of the system we conclude that SNP is *scalable* with the size of the system and is a suitable protocol for updating routing information in very large systems.

We simulate SNP and related protocols and study the effect of the variation of protocol and system parameters on the performance and overhead of the protocols. For the simulation we assume a general model of communication between processes - the *spatial locality* model. The four protocols are simulated on a square toroidal mesh.

The values of the optimal neighborhood structure and the corresponding minimum update bandwidth obtained from the simulation of SNP confirm the values obtained from the analytical model.

From the simulations we conclude that of the four protocols the performance of the the Shifting Neighborhood Protocol with Dynamic Broadcast SNP(DB) is the best and close to the best achievable performance by any possible protocol. However this performance is achieved at a higher update bandwidth compared to the other protocols. The performance of the Shifting Neighborhood Protocol with Adaptive Routing SNP(AR) is very close to that of SNP(DB) and is a particularly attractive protocol to implement because of the low update bandwidth in comparison to SNP(DB). The Shifting Neighborhood Protocol has update bandwidth equal to that for SNP(AR) but with larger stretch ratios. The Moving Home Node Protocol MHNP is a low cost protocol and may be chosen to be implemented if the average time stretch ratio that can be tolerated is higher and the bandwidth available for control traffic is limited, or if the system is small.

We study the performance of SNP(DB) and SNP(AR) and how it is affected by the change in the degree of spatial locality in the communication between processes. For both the protocols we observe that the average message trajectory stretch ratio slowly increases as the locality decreases. However both protocols have stretch ratios very close to one even when there is no spatial locality of communication between processes.

The results of the simulation of SNP(AR) indicate that while the update bandwidth increases with the size of the system, both the time and message trajectory stretch ratios are relatively constant as the number of nodes in the system is varied. Thus the simulation results indicate that SNP(AR) is scalable with the size of the system.

In conclusion the *composite protocol* that we propose for routing messages to moving processes in large distributed systems is the Shifting Neighborhood Protocol or one of its variations with caches to take advantage of temporal locality in the communication. The protocols scale well with the rate of migration of a process and the size of the system.

6.2 Future Work

The use of process migration as a tool for load management is not prevalent in current operating systems technology because of the high cost of process migration. General purpose processes currently have large states that are scattered in different parts of the node memory. The state of a process typically consists of the code, the process

control block, stacks and heap memory allocated to the process. As a result the amount of processing and the time required to collect the state of a process for migration is very large. Moreover a large number of messages are required to send the state from one node to another. In view of the potential benefits of process migration future research should investigate methods for making process migration inexpensive. The first possible approach is to reconsider the conventional representation of the state of a process. We should investigate methods to represent the state of a process in a compact form. By having a so called *freeze-dried* state it becomes easier to checkpoint the state of a process for fault tolerance and applications such as distributed simulation that allow a process to rollback to a previous state. The second approach is to provide architectural support for process migration, so that there is hardware assistance to collect and transmit the process state.

In our protocol when a process is initially created it is not necessary to create routing entries for the process at all nodes. The problem of the *distribution* of routing entries is to decide the number of routing entries for a process that should be created and where to place them. One technique for distributing routing entries in any topology is to broadcast routing entries to all the nodes and each node independently decides with some probability whether to keep the entry in its routing table or to discard it. Alternatively for a given topology, say a hypercube, we can solve the following distribution problems:

1. Find the minimum number m of routing entries and their placement at nodes of the given topology so that the maximum shortest distance from any node to the closest routing entry is at most a specified distance max_dist .
2. Place a specified number n of routing entries at nodes of the topology such that the maximum shortest from any node to the closest routing entry is minimized.

The design of hashing or mapping functions for a given topology to locate routing entries for a process is also an open problem.

Our protocol makes use of unbounded sequence numbers for the migration-count of a process. The migration-count for a process is incremented whenever a process migrates. We can investigate the modification of our protocols, under certain assumptions of the underlying system, to avoid the use of unbounded sequence numbers.

In the SNP protocol routing entries for a process are updated nonuniformly, with entries that are located at nodes close to the process being updated more frequently. While in the description and analysis of SNP we assume that routing entries for a process are uniformly distributed throughout the system, there is nothing in the design of SNP that precludes a non-uniform distribution of routing entries. An interesting problem would be to combine SNP and Kleinrock and Kamoun's mechanism so that different levels have different degrees of information about the location of a node and are updated with varying frequency. Thus a large fraction of routing entries for a process are located close to the process and are also updated more frequently than the entries located at distant nodes.

The difference between SNP with caches and Fowler's path compression protocol is that while in Fowler's protocol routing entries for a process are updated only when the process is accessed, in SNP with caches routing entries for a process are also updated at the time a process migrates. A detailed simulation of the two protocols running on a large distributed system which includes the modeling of failures will give us a better picture of the difference between the two schemes.

Related to the problem that we considered is the problem of assigning unique names to objects in large distributed systems. The problem is difficult because an unbounded number of objects may be created dynamically at different nodes of the system. Objects are also deleted which makes their name available for reuse.

Dynamic load managing distributed operating systems require large communication bandwidth capacities because of the large amount of control traffic required to communicate the changing state of the distributed system. Fortunately fiber optic networks currently being developed have bandwidths that are orders of magnitude larger than the bandwidths of conventional networks.

Finally the policy issues related to dynamic load management operating systems is an open area for research.

REFERENCES

- Ahamad 87a Ahamad, Mustaque, Mostafa H. Ammar, Jose Bernabeu, and M. Yousef Khalidi, "A Multicast Scheme for Locating Objects in a Distributed Operating System", Georgia Institute of Technology Technical Report GIT-ICS-87/01, January 1987.
- Ahamad 87b Ahamad, Mustaque, "Ripple Search: An Algorithm for Locating Resources in Large Computer Networks", *IEEE Infocom 87*, San Francisco, California, March - April, 1987.
- Amer 88 Amer, Fathy, and Yao-Nan Lien, "Hierarchical Routing Design for SURAN", *IEEE International Conference on Communications '88*, Philadelphia, June, 1988.
- Birrell 82 Birrell, Andrew D., Roy Levin, Roger M. Needham and Michael D. Schroeder, "Grapevine: An Exercise in Distributed Computing", *Communications of the ACM*, Vol. 25, No. 4, April 1982.
- Black 85 Black, Andrew P., "Supporting Distributed Applications : Experience with Eden", Technical Report TR85-03-02, Department of Computer Science, University of Washington, Seattle, Washington, March 1985.
- Boggs 83 Boggs, David R., "Internet Broadcasting", Technical Report CSL-83-3, Xerox Palo Alto Research Center, October 1983.
- Bokhari 81 Bokhari, Shahid H, "On the Mapping Problem", *IEEE Transactions on Computers*, Vol. C-30, No. 3, March 1981.
- Chlamtac 87 Chlamtac, Imrich, "The Wave Expansion Approach to Broadcasting in Multi-Hop Radio Networks", *IEEE Infocom 87*, San Francisco, California, March - April, 1987.
- Christofides 75 Christofides, Nicos, "Graph Theory - An Algorithmic Approach", Academic Press, 1975.

- Coffman 73 Coffman, Edward G., and Peter J. Denning, "Operating Systems Theory", Prentice Hall, 1973.
- Doi 86 Doi, M. and S. F. Edwards, "The Theory of Polymer Dynamics", Oxford Science Publications, Oxford University Press, 1986.
- Dalal 77 Dalal, Yogen K., "Broadcast Protocols in Packet Switched Computer Networks", PhD thesis, Stanford University, April 1977. (Digital Systems Laboratory, Technical Report No. 128)
- Dalal 78 Dalal, Yogen K., and Robert M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets", *Communications of the ACM*, Vol. 21, No. 12, December 1978.
- Douglis 87 Douglis, F., and J. K. Ousterhout, "Process Migration in the Sprite Operating System", *Seventh International Conference on Distributed Computing*, September 1987, pp. 18-25.
- Fowler 85 Fowler, Robert J., "Decentralized Object Finding Using Forwarding Addresses", PhD thesis, University of Washington, Seattle, Washington, December 1985. (Department of Computer Science Technical Report TR85-12-1)
- Fowler 86 Fowler, Robert J., "The Complexity of Using Forwarding Addresses for Decentralized Object Finding", *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing*, Calgary, Canada, August 1986.
- Gao 84 Gao, Chuanshan, Jane W. S. Liu, and Malcolm Railey, "Load Balancing Algorithms in Homogeneous Distributed Systems", *Proceedings of the 1984 International Conference on Parallel Processing*, August 1984.
- Goldberg 87 Goldberg, Arthur P., and David R. Jefferson, "Transparent Process Cloning: A Tool for Load Management of Distributed Programs", *Proceedings of the 1987 International Conference on Parallel Processing*, August 1987.

- Gopal 88 Gopal, Inder and Adrian Segall, "Directories for Networks with Casually Connected Users", *IEEE Infocom 88*, New Orleans, Louisiana, March, 1988.
- Handler 79 Handler, Gabriel Y. and Pitu B. Mirchandani, "Location on Networks - Theory and Algorithms", The MIT Press, 1979.
- Hillis 81 Hillis, W. Daniel, "The Connection Machine (Computer Architecture for the New Wave)", MIT Artificial Intelligence Laboratory, A. I. Memo No. 646, September 1981.
- Hwang 87 Hwang, Deborah Jing-Hwa, "Constructing a Highly-Available Location Service for a Distributed Environment", MS thesis, Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, Massachusetts, November 1987.
- Jul 88 Jul, Eric, H. Levy, N. Hutchinson and A. Black, "Fine-Grained Mobility in the Emerald System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988.
- Kahn 79 Kahn, Robert E., S. A. Gronemeyer, J. Burchfiel and R. C. Kunzelman, "Advances in Packet Radio Technology", *Proceedings of the IEEE* , Vol. 66, No. 11, pp. 1468-1496, October 1978.
- Kamoun 76 Kamoun, Farouk, "Design Considerations for Large Computer Communication Networks", PhD Dissertation, UCLA Computer Science Department, Los Angeles, California, March 1976.
- Kleinrock 77 Kleinrock, L. and F. Kamoun, "Hierarchical Routing for Large Networks", *Computer Networks* , vol. 1, pp. 155-174, January 1977.
- Korach 84 Korach E., D. Rotem, and N. Santoro, "Distributed Algorithms for Finding Centers and Medians in Networks", *ACM Transactions on Programming Languages and Systems*, Vol. 6, No. 3, pp. 380-401, July 1984.

- Kung 87 Kung, Roberto, and Nachum Shacham, "A Distributed Algorithm for Reliable Message Delivery over a Sub-Network", *IEEE Globecom 87*, Vol. 1, Tokyo, Japan, November 1987.
- Lampson 86 Lampson, B. W., "Designing a Global Name Service", *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing*, Calgary, Canada, August 1986.
- Laning 83 Laning, Laurence J., and Michael S. Leonard, "File Allocation in a Distributed Computer Communication Network", *IEEE Transactions on Computers*, Vol. C-32, No. 3, March 1983.
- Lauer 86 Lauer, Gregory S., "Hierarchical Routing Design for SURAN", *IEEE International Conference on Communications '86*, Toronto, Canada, Vol. 1, pp. 4.2-1 - 4.2-10, June, 1986.
- Lee 87 Lee, Soo-Young, and J. K. Aggarwal, "A Mapping Strategy for Parallel Processing", *IEEE Transactions on Computers*, Vol. C-36, No. 4, April 1987.
- Li 86 Li, Victor O. K., and Rong-Feng Chang, "Proposed Routing Algorithms for the US Army Mobile Subscriber Equipment (MSE) Network", *IEEE MILCOM 86*, Monterey, California, October, 1986.
- Lin 87 Lin, Frank C. H., and Robert M. Keller, "The Gradient Model Load Balancing Method", *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 1, January 1987.
- Liskov 86 Liskov, B. and R. Ladin, "Highly-Available Distributed Services and Fault-Tolerant Distributed Garbage Collection", *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing*, Calgary, Canada, August 1986.
- Lu 87 Lu, Chin, Arthur Chen and Jane W. S. Liu, "Protocols for Reliable Process Migration", *IEEE Infocom 87*, San Francisco, California, March - April, 1987.

- McQuillan 78 McQuillan, J., G. Falk and I. Richer, "A Review of the Development and Performance of the ARPANET Routing Algorithm", *IEEE Transactions on Communications*, Vol. COM-26, No. 12, December 1978.
- Mullen 85 Mullender, Sape J. and Paul M. B. Vitanyi, "Distributed Match-Making for Processes in Computer Networks", *Proceedings of the Fourth ACM Symposium on Principles of Distributed Computing*, Minacki, Canada, August 1985.
- Ni 85 Ni, Lionel M., Chong-Wei Xu and Thomas B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing", *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 10, October 1985.
- Oppen 83 Oppen, Derek C., and Yogen K. Dalal, "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment", *ACM Transactions on Office Information Systems*, Vol. 1, No. 3, pp. 230-253, July 1983.
- Ousterhoust 88 Ousterhoust, et al., "The Sprite Network Operating System", *Computer*, February 1988.
- Peleg 87 Peleg, David and Eli Upfal, "A Tradeoff between Space and Efficiency for Routing Tables", *IBM Research Report*, RJ5859 (58910), June 1987.
- Pierre 69 Pierre, Donald A., "Optimization Theory with Applications", John Wiley & Sons, 1969.
- Popek 85 Popek, G. J. and B. J. Walker, "The LOCUS Distributed System Architecture", Computer Systems Series, The MIT Press, 1985.
- Powell 83 Powell, M. L. and B. P. Miller, "Process Migration in DEMOS/MP", *Proc. 9th Symposium on Operating Systems Principles*, October 1983.
- Ramamoorthy 86 Ramamoorthy, C. V., Jaideep Srivastava and Wen-Tek Tsai, "A Distributed Clustering Algorithm for Large Computer Networks", *Proc. of the 6th International Conference on*

Distributed Computing Systems, Cambridge, Massachusetts, May 1986.

- Rashid 81 Rashid, R. F. and G. Robertson, "Accent: A Communication Oriented Network Operating System Kernel", *Proc. 8th Symposium on Operating Systems Principles*, December 1981.
- Reed 87 Reed, D. A., P. K. McKinley and M. F. Barr, "Performance Analysis of Switching Strategies", *Proc. 1987 Symposium on the Simulation of Computer Networks*, Colorado Springs, Colorado, August 1987.
- Santoro 85 Santoro, Nicola, and Ramez Khatib, "Labeling and Implicit Routing in Networks", *The Computer Journal*, Vol. 28, No. 1, 1985.
- Scheur 87 Scheurich, Christoph and Michel Dubois, "Dynamic Memory Allocation in a Mesh-Connected Multiprocessor", *Proc. of the 12th Annual Hawaii International Conference on System Sciences*, January 1987.
- Segall 83 Segall, A., "Distributed Network Protocols", *IEEE Transactions on Information Theory*, Vol. IT-29, No. 1, January 1983.
- Swope 86 Swope, Steven M. and Richard M. Fujimoto, "SIMON II Kernel Reference Manual", University of Utah Computer Science Department, Technical Report UUCS-86-001, March 1986.
- Tannenbaum 81 Tannenbaum, Andrew S., "Computer Networks", Prentice Hall, 1981.
- Terry 85 Terry, Douglas Brian, "Distributed Name Servers: Naming and Caching in Large Distributed Computing Environments", PhD Dissertation, Computer Science Division (EECS), University of California, Berkeley, March 1985. (Report No. UCB/CSD 85/228)
- Theimer 86 Theimer, M., "Preemptable Remote Execution Facilities for Loosely-Coupled Distributed Systems", Stanford University Technical Report STAN-CS-86-1128, June 1986.

- Wall 80 Wall, David W., "Mechanisms for Broadcast and Selective Broadcast", PhD thesis, Stanford University, April 1977. (Computer Systems Laboratory, Technical Report No. 190)
- Westcott 82 Westcott, Jil, and John Jubin, "A Distributed Routing Design for a Broadcast Environment", *IEEE MILCOM 82*, Boston, Massachusetts, Vol. 3, pp. 10.4-1 - 10.4-5, October, 1982.
- Zhou 87 Zhou, Songnian, "Performance Studies of Dynamic Load Balancing in Distributed Systems", Report No. UCB/CSD 87/376, Computer Science Division (EECS), University of California, Berkeley, California, October 1987.

Appendix 1

We show in this Appendix that for an unconstrained random walk the following holds:

If a process on every migration moves a distance that is constant then for the process to cross a boundary of radius r_i around the initial position of the process will take $\frac{r_i^2}{l^2}$ moves, where l is a constant.

Let the distance moved on each migration of the process be μ hops.

Let D be the distance from the initial location of the process to its location after n steps.

In general for an unrestricted random walk the expected value $E[D] = 0$. Instead $\sqrt{E[|D|^2]}$ is used as the *characteristic distance* that the process has traveled in n steps. This is similar to the method for calculating the characteristic length of a polymer chain as described in [Doi 86].

We will show in the following section that

$$E[|D|^2] = c_1 n \quad \text{where } c_1 \text{ is a constant}$$

Therefore in n steps the process travels an average of $\sqrt{E[|D|^2]} = \sqrt{c_1} \sqrt{n}$ hops

Because the next position of a process depends only on its current position and not its previous history, hence we can apply the scaling property to say that

To travel r hops the average number of steps taken by a process is $\frac{r^2}{l}$, where l is a constant.

Random Walk on a Two Dimensional Mesh

Consider a two dimensional mesh where each node has four neighbors.

We assume that the average size of a move is 1 hop. We will show that $E[|D|^2] = n$ for an unrestricted random walk in a two dimensional mesh. The analysis will be similar when the average size of a move is greater than 1 hop. Let D be the distance from the initial location of the process to its position after n steps. Let D_i denote the change in the position of the process on the i th step. Each step is randomly chosen to be in one of the four possible directions. Then

$$D = D_1 + D_2 + \dots + D_n$$

Hence

$$E[|D|^2] = E\left[\sum_{i=1}^n |D_i|^2\right] + 2 E\left[\sum_{i \neq j} |D_i| |D_j|\right]$$

Since D_i are independent and identically distributed for all i hence

$$E[|D|^2] = n E[|D_i|^2] + 2 n (n-1) (E[|D_i|])^2$$

We know that

$$E[|D_i|] = 1 \times \frac{1}{4} - 1 \times \frac{1}{4} + 1 \times \frac{1}{4} - 1 \times \frac{1}{4} = 0$$

and

$$E[|D_i|^2] = 4 \times \frac{1}{4} = 1$$

$$\therefore E[|D|^2] = n$$

In fact the result $E[|D|^2] = c_1 n$ holds in general for arbitrary number of dimensions and is shown for the continuous case in [Doi 86].

Appendix 2

In this Appendix we show that the point $(r_1, r_2, r_3, \dots, r_{m-1})$ at which $\frac{\partial C}{\partial r_i} = 0$ for $1 \leq i \leq m-1$, is a *minimum*, where

$$C = v \left[r_1^2 + \frac{r_2^2 l^2}{r_1^2} + \dots + \frac{r_i^2 l^2}{r_{i-1}^2} + \dots + \frac{r_m^2 l^2}{r_{m-1}^2} \right]$$

Let us define

$$C_{r_i r_j} = \frac{\partial^2 C}{\partial r_i \partial r_j}$$

and

$$A_k = \begin{bmatrix} C_{r_1 r_1} & C_{r_1 r_2} & C_{r_1 r_3} & \dots & C_{r_1 r_k} \\ C_{r_2 r_1} & C_{r_2 r_2} & C_{r_2 r_3} & \dots & C_{r_2 r_k} \\ C_{r_3 r_1} & C_{r_3 r_2} & C_{r_3 r_3} & \dots & C_{r_3 r_k} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \dots & \vdots \\ C_{r_k r_1} & C_{r_k r_2} & C_{r_k r_3} & \dots & C_{r_k r_k} \end{bmatrix}$$

Based on Sylvester's Theorem [Pierre 69], to prove that the point $(r_1, r_2, r_3, \dots, r_{m-1})$ at which $\frac{\partial C}{\partial r_i} = 0$ for $1 \leq i \leq m-1$, is a *minimum*, we have to show

The matrix A_{m-1} is *positive semidefinite*, i.e.

$\text{Det}(A_k) > 0$ for $1 \leq k \leq m-1$, where $\text{Det}(A_k)$ is the determinant of A_k .

From Equation 4.6 (Section 4.6.1) we know that

$$\begin{aligned} \frac{\partial C}{\partial r_i} &= 0 \text{ for } 1 \leq i \leq m-1, \\ \Rightarrow r_i &= \left(\frac{R}{l}\right)^{\frac{i}{m}} l \text{ for } 1 \leq i \leq m-1 \end{aligned}$$

From the expression for C we have

$$C_{r_i r_i} = 8v \left(\frac{l}{r_1}\right)^{2(i-1)} \quad \text{for } 1 \leq i \leq m-1$$

$$C_{r_i r_{i+1}} = -4v \left(\frac{l}{r_1}\right)^{2i-1} \quad \text{for } 1 \leq i \leq m-2$$

$$C_{r_i r_j} = 0 \quad \text{for } |j-i| > 2$$

$$C_{r_i r_j} = C_{r_j r_i}$$

Therefore

$$\text{Det}(A_1) = C_{r_1 r_1} = 8v > 0$$

$$\text{Det}(A_2) = C_{r_1 r_1} C_{r_2 r_2} - C_{r_1 r_2}^2 = 48v^2 \frac{l^2}{r_1^2} > 0$$

$$\begin{aligned} \text{Det}(A_3) &= C_{r_1 r_1} [C_{r_2 r_2} \times C_{r_3 r_3} - C_{r_2 r_3}^2] \\ &\quad - C_{r_1 r_2} [C_{r_2 r_1} \times C_{r_3 r_3} - C_{r_2 r_3} \times C_{r_3 r_1}] \\ &\quad + C_{r_1 r_3} [C_{r_2 r_1} \times C_{r_3 r_2} - C_{r_2 r_2} \times C_{r_3 r_1}] \\ &= 256v^3 \frac{l^6}{r_1^6} > 0 \end{aligned}$$

$$\text{Det}(A_4) = 1280v^4 \frac{l^{12}}{r_1^{12}} > 0$$

In general

$$\text{Det}(A_k) = (4v)^k (k+1) \frac{l^{k(k-1)}}{r_1^{k(k-1)}} \quad \text{for } 1 \leq k \leq m-1$$

which is always > 0 since $v > 0$.

Thus we have shown that the point $(r_1, r_2, r_3, \dots, r_{m-1})$ at which $\frac{\partial C}{\partial r_i} = 0$ for $1 \leq i \leq m-1$ is a minima.

Appendix 3

In this Appendix we determine the average distance \bar{d} between two distinct nodes of a $p \times q$ mesh. The following derivation is from [Kamoun 76]. Let X and U be two arbitrary nodes of the mesh as shown in Figure A7.1 with coordinates (x, y) and (u, v) respectively.

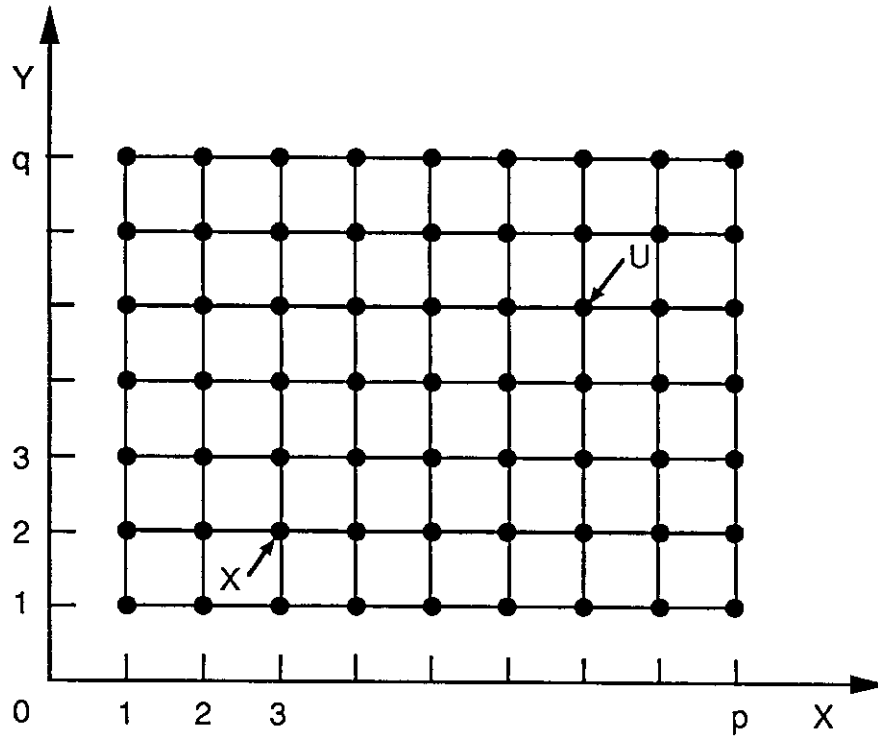


Figure A.7.1 : Average Shortest Distance between Two Nodes in a $p \times q$ mesh

The distance between the two nodes is given by

$$d_{XU} = |u - x| + |v - y|$$

Let $D(x, y)$ be the sum of the distances from any node in the mesh to the node (x, y) .
Then

$$D(x, y) = \sum_{u=1}^p \sum_{v=1}^q \left[|u - x| + |v - y| \right]$$

$$\begin{aligned}
&= q \left[\sum_{u=1}^{x-1} (x-u) + \sum_{u=x+1}^p (u-x) \right] + p \left[\sum_{v=1}^{y-1} (y-v) + \sum_{v=y+1}^q (v-y) \right] \\
&= q \left[x^2 - (p+1)x + \frac{p(p+1)}{2} \right] + p \left[y^2 - (q+1)y + \frac{q(q+1)}{2} \right]
\end{aligned}$$

The total number of pairs of distinct nodes in a $p \times q$ mesh is equal to

$$p q (p q - 1)$$

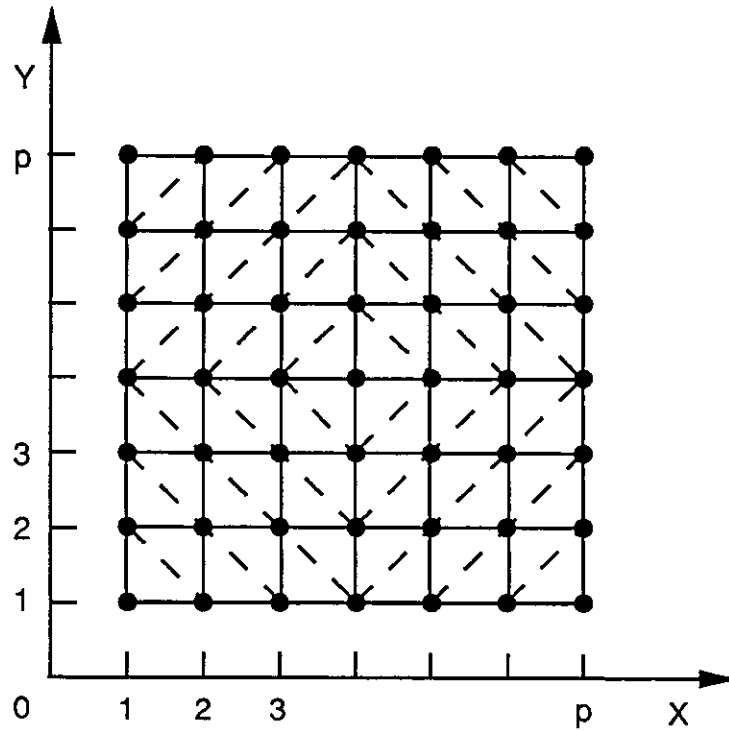
Therefore

$$\begin{aligned}
\bar{d} &= \frac{1}{p q (p q - 1)} \sum_{x=1}^p \sum_{y=1}^q D(x, y) \\
&= \frac{1}{p q (p q - 1)} \left[\frac{1}{3} q^2 p (p+1) (p-1) + \frac{1}{3} p^2 q (q+1) (q-1) \right] \\
&= \frac{p+q}{3}
\end{aligned}$$

For a $p \times p$ square mesh the average shortest distance between two distinct nodes is $\frac{2p}{3}$.

Appendix 4

Here we find the average shortest distance \bar{d} between the center of a mesh to any other random node of the mesh.



**Figure A.8.1 : Average Distance between an Arbitrary Node
and the Center of a $p \times p$ mesh**

Consider the $p \times p$ mesh shown in Figure A8.1. Without loss of generality we assume that p is odd. Let N_k be the number of nodes at distance k from the center.

$$N_k = \begin{cases} 4k & 1 \leq k \leq (p-1)/2 \\ 4(p-k) & (p+1)/2 \leq k \leq p-1 \end{cases}$$

The total number of nodes in the mesh excluding the center is

$$p^2 - 1$$

The average length of the shortest path from the center to any other node in the mesh is

$$\begin{aligned}
 \bar{d} &= \frac{1}{p^2 - 1} \left[\sum_{k=1}^{(p-1)/2} 4k^2 + \sum_{k=(p+1)/2}^{p-1} 4(p-k)k \right] \\
 &= \frac{4}{p^2 - 1} \left[\frac{(p-1)(p+1)p}{2 \times 6 \times 2} + p \left[\frac{(p-1)p}{2} - \frac{(p-1)(p+1)}{2 \times 2 \times 2} \right] \right. \\
 &\quad \left. - \frac{(p-1)p(2p-1)}{6} + \frac{(p-1)(p+1)p}{2 \times 2 \times 6} \right] \\
 &= \frac{p}{2}
 \end{aligned}$$

Therefore the average shortest distance \bar{d} from the center of a $p \times p$ mesh to any other random node of the mesh is $\frac{p}{2}$.

Appendix 5

We determine the average shortest distance \bar{d} between a random node in a $p \times p$ mesh and a random node that lies on the edge of the mesh. Let $D(x, y)$ be the average shortest distance between a particular node with coordinates (x, y) that lies on the mesh and a random node on the edge of the mesh (Figure A9.1).

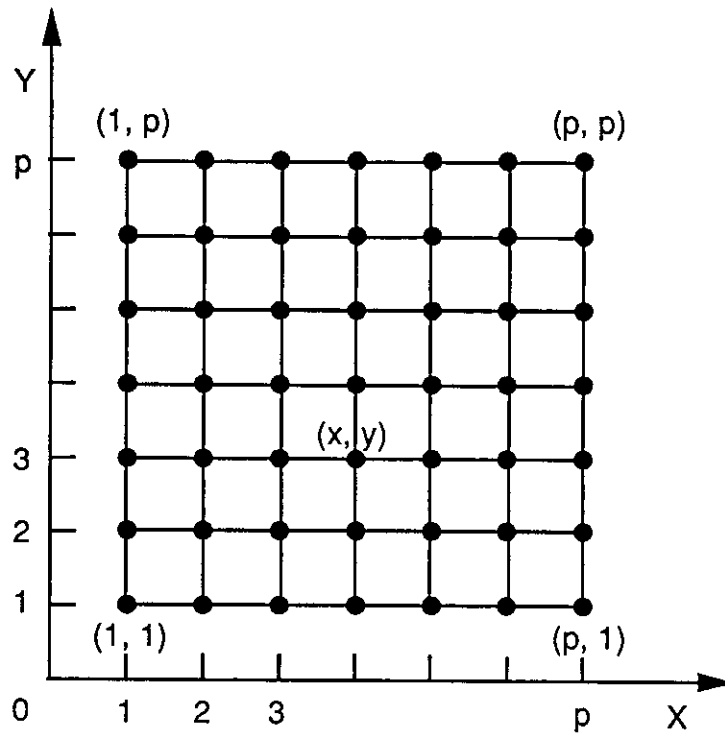


Figure A.9.1 : Average Distance Between a Node of a $p \times p$ mesh and a random Edge Node

$$\begin{aligned}
 D(x, y) = & \frac{1}{4(p-1)} \left[\sum_{v=1}^p \left[(x-1) + |v-y| \right] + \sum_{v=1}^p \left[(p-x) + |v-y| \right] \right. \\
 & \left. + \sum_{u=1}^p \left[|u-x| + (y-1) \right] - \left[(x-1) + (y-1) + (p-x) + (y-1) \right] \right]
 \end{aligned}$$

$$+ \sum_{u=1}^p \left[|u-x| + (p-y) \right] - \left[(x-1) + (p-y) + (p-x) + (p-y) \right]$$

Now from analysis similar to Appendix 3

$$\begin{aligned} \sum_{v=1}^p |v-y| &= \sum_{v=1}^{y-1} (y-v) + \sum_{v=y+1}^p (v-y) \\ &= (y-1)y - (y-1)y/2 + p(p+1)/2 - y(y+1)/2 - (p-y)y \\ &= y^2 - (p+1)y + p(p+1)/2 \end{aligned}$$

Similarly

$$\sum_{u=1}^p |u-x| = x^2 - (p+1)x + p(p+1)/2$$

$$\begin{aligned} \therefore D(x, y) &= \frac{1}{4(p-1)} \left[(x-1)p + y^2 - (p+1)y + p(p+1)/2 \right. \\ &\quad \left. + (p-x)p + y^2 - (p+1)y + p(p+1)/2 \right. \\ &\quad \left. + x^2 - (p+1)x + p(p+1)/2 + (y-1)p \right. \\ &\quad \left. - 2(y-1) - (p-1) \right. \\ &\quad \left. + x^2 - (p+1)x + p(p+1)/2 + (p-y)p \right. \\ &\quad \left. - 2(p-y) - (p-1) \right] \\ &= \frac{1}{2(p-1)} \left[2(p^2 - p + 1) + x^2 - (p+1)x + y^2 - (p+1)y \right] \end{aligned}$$

Since there are p^2 nodes in the mesh, hence

$$\begin{aligned} \bar{d} &= \frac{1}{p^2} \times \sum_{x=1}^p \sum_{y=1}^p D(x, y) \\ &= \frac{1}{2p^2(p-1)} \times \left[2p^2(p^2 - p + 1) + p \left[\sum_{x=1}^p x^2 - (p+1) \sum_{x=1}^p x \right] \right. \\ &\quad \left. + p \left[\sum_{y=1}^p y^2 - (p+1) \sum_{y=1}^p y \right] \right] \\ &= \frac{1}{6(p-1)} \times [5p^2 - 9p + 4] \\ &= \frac{5p}{6} - \frac{2}{3} \end{aligned}$$

Therefore the average distance from a random node in a $p \times p$ mesh and a random node on the edge of the mesh is $\frac{5p}{6} - \frac{2}{3}$.

Appendix 6

We determine the average distance \bar{d} from the center of a $p \times p$ mesh to nodes on the edge of the mesh.

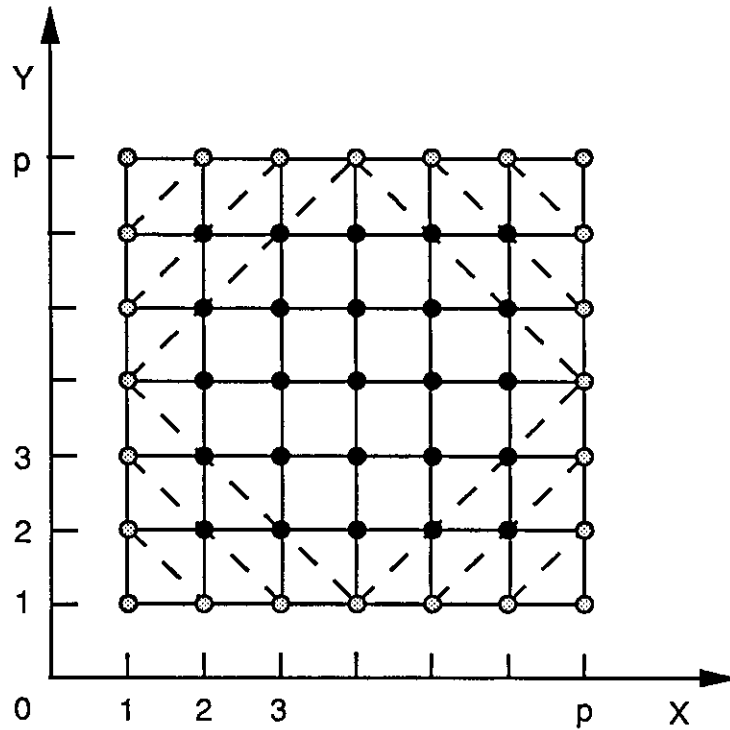


Figure A.10.1 : Average Distance between the Center of a $p \times p$ mesh and an Edge Node

Without loss of generality we assume that p is odd. From Figure A10.1 we observe that the shortest distance of the edge nodes from the center node is as follows:

No. of Edge Nodes Distance from Center

4	$(p - 1) / 2$
8	$(p + 1) / 2$
8	$(p + 3) / 2$
.	.
.	.
8	$p - 2$
4	$p - 1$

The average distance from the center of the mesh to nodes on the edge is

$$\begin{aligned}
 \therefore \bar{d} &= \frac{1}{4 \times 2 + 8(p - 1 - (p + 1) / 2)} \left[\frac{4(p - 1)}{2} + 4(p - 1) + 8 \sum_{i=(p+1)/2}^{p-2} i \right] \\
 &= \frac{3p^2 - 6p + 3}{4(p - 1)} \\
 &= \frac{3(p - 1)}{4}
 \end{aligned}$$

Appendix 7

Here we calculate the average distance \bar{d} between two nodes U and Y (Figure A11.1) such that U and Y belong to $r_2 \times r_2$ mesh, Y is the center of a $r_1 \times r_1$ mesh, where $r_2 > r_1$, and node U lies outside the $r_1 \times r_1$ mesh.

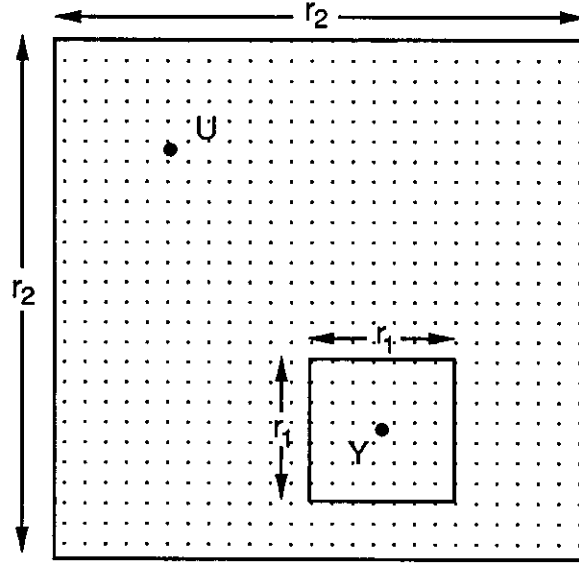


Figure A.11.1 : Average Distance between Node U and Node Y

We first find the sum of the lengths D_{sum} and the number of shortest paths L_{sum} between two arbitrary nodes in a mesh without any constraints on their relative locations. From Appendix 3 we know that the average shortest distance between two arbitrary distinct nodes of a $r_2 \times r_2$ mesh is $\frac{2r_2}{3}$ and further

$$L_{sum} = r_2^2 (r_2^2 - 1)$$

$$\begin{aligned} \therefore D_{sum} &= \frac{2r_2}{3} \times r_2^2 (r_2^2 - 1) \\ &= \frac{2r_2^3 (r_2^2 - 1)}{3} \end{aligned}$$

We have to however take into account the constraint that U lies outside the boundary of a $r_1 \times r_1$ mesh centered at Y . Therefore from the computation of all possible paths with arbitrary locations of U and Y , some paths have to be excluded. In order to do so we compute the sum of the lengths D_{diff} and number of shortest paths L_{diff} between two nodes U and Y such that U and Y lie on the $r_2 \times r_2$ mesh and U lies in a $r_1 \times r_1$ sub-mesh with Y as center. Thus we can compute \bar{d} as

$$\bar{d} = \frac{D_{sum} - D_{diff}}{L_{sum} - L_{diff}}$$

To compute D_{diff} and L_{diff} we consider three cases illustrated in Figure A.11.2

- (1) Node Y is in region 1
- (2) Node Y is in region 2
- (3) Node Y is in region 3

Case (1) Node Y is in region 1

Consider node Y the center of a $r_1 \times r_1$ mesh and U to be any node distinct from Y in the $r_1 \times r_1$ mesh. Node Y can be any arbitrary node in region 1 as shown in Figure A11.2. When Y is located in region 1 then note that the entire $r_1 \times r_1$ mesh centered at Y is located in the $r_2 \times r_2$ mesh. From Appendix 4 we know that the number of paths from the center of a $r_1 \times r_1$ mesh to any other node of the $r_1 \times r_1$ mesh is $r_1^2 - 1$ and the average shortest distance from the center to any other node of the mesh is $\frac{r_1}{2}$. For a particular node Y in region 1 the sum of the shortest path lengths from Y to an arbitrary node U that lies in a $r_1 \times r_1$ mesh centered at Y is therefore

$$\frac{r_1 (r_1^2 - 1)}{2}$$

Since there are $(r_2 - r_1 + 1)^2$ possible positions for node Y in region 1 hence the sum of the length of shortest paths $D_{diff(1)}$ between any two nodes U and Y , such that Y lies in region 1 and U lies in a $r_1 \times r_1$ mesh centered at Y is given by

$$D_{diff(1)} = (r_2 - r_1 + 1)^2 \times \frac{r_1 (r_1^2 - 1)}{2}$$

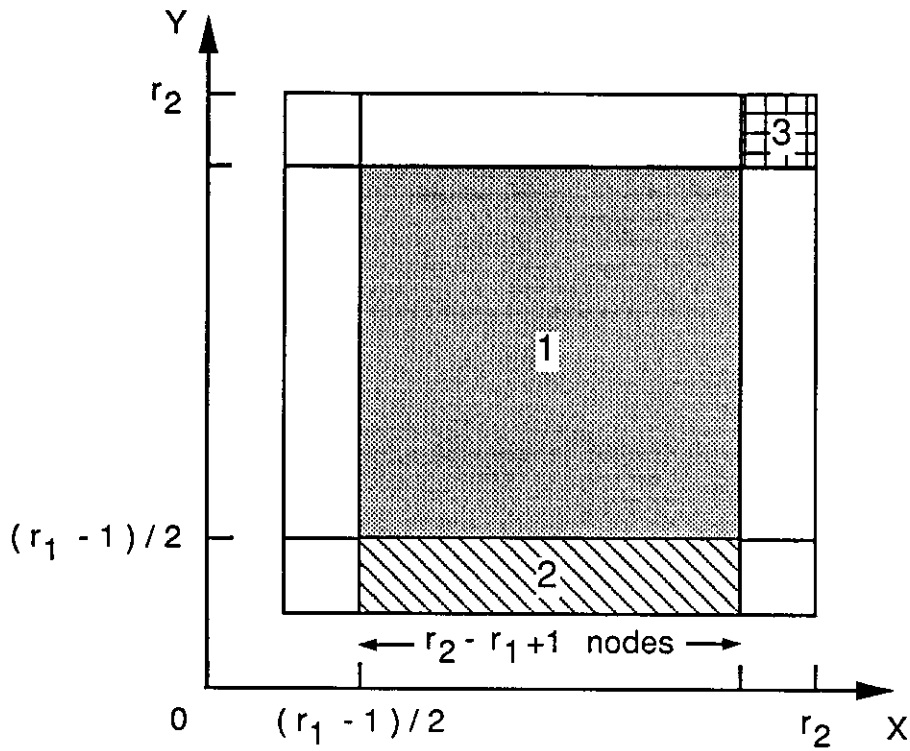


Figure A.11.2 : Different Regions where Node Y may be Located

and the corresponding number of paths $L_{diff(1)}$ is

$$L_{diff(1)} = (r_2 - r_1 + 1)^2 \times (r_1^2 - 1)$$

Case (2) Node Y is in region 2

Consider node Y to be an arbitrary node located in region 2 of the $r_2 \times r_2$ mesh. Consider another arbitrary node U that lies on a $r_1 \times r_1$ mesh centered at Y such that U is also on the $r_2 \times r_2$ mesh. We calculate the sum of the lengths of the shortest paths $D_{diff(2)}$ between U and Y and the number of paths $L_{diff(2)}$ between U and Y .

Consider region 2 shown in Figure A11.2. We consider two sub-cases here - when the y -coordinate of U is less than or equal to the y coordinate of Y and when the y -coordinate of U is greater than the y -coordinate of Y .

Case (2.1) y-coordinate of $U \leq$ y-coordinate of Y

In Figure A11.3 we determine the path lengths and number of paths from Y with coordinates $((r_1 + 1) / 2, j)$ to nodes in the $r_1 \times j$ mesh.

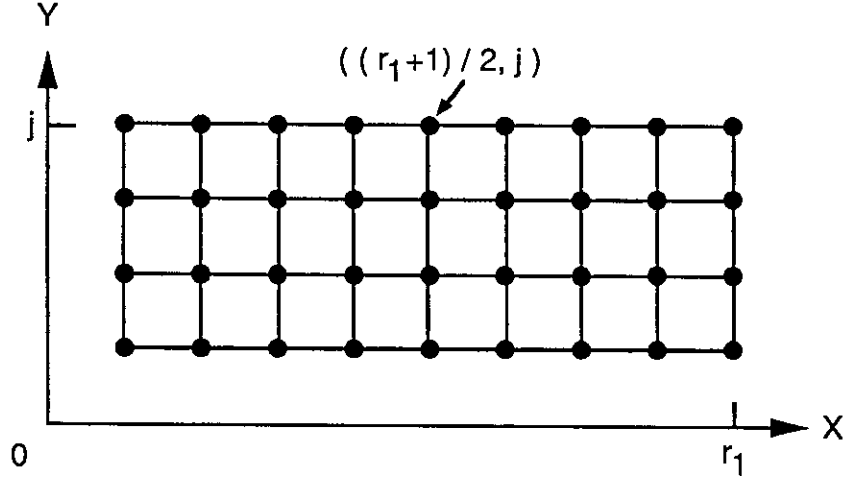


Figure A.11.3 : Sum of Path Lengths from Node Y to Nodes in the $r_1 \times j$ mesh

The value of j ranges from 1 to $(r_1 - 1) / 2$. By analysis similar to that of Appendix 3 we obtain the sum of the shortest distances from a particular node Y to nodes in the $r_1 \times j$ mesh (Figure A11.3) is

$$\begin{aligned}
 & j \left[\left(\frac{r_1 + 1}{2} \right)^2 - \frac{(r_1 + 1)(r_1 + 1)}{2} + \frac{r_1 (r_1 + 1)}{2} \right] \\
 & + r_1 \left[j^2 - (j + 1)j + \frac{j(j + 1)}{2} \right] \\
 & = \frac{j(r_1 + 1)(r_1 - 1)}{4} + \frac{j r_1 (j - 1)}{2}
 \end{aligned}$$

By keeping the x -coordinate of U fixed and varying j we obtain the sum of the shortest distance from U to nodes in the $r_1 \times j$ mesh as

$$\begin{aligned}
 & \sum_{j=1}^{(r_1-1)/2} \left[\frac{j(r_1 + 1)(r_1 - 1)}{4} + \frac{j r_1 (j - 1)}{2} \right] \\
 & = \frac{(r_1^2 - 1)(5 r_1^2 - 6 r_1 - 3)}{96}
 \end{aligned}$$

The x -coordinate of node Y can take $(r_2 - r_1 + 1)$ possible values, and hence for Case (2.1) the sum of the lengths of shortest paths from node Y to U is

$$D_{diff(2.1)} = \frac{1}{96} (r_2 - r_1 + 1) (r_1^2 - 1) (5r_1^2 - 6r_1 - 3)$$

The number of shortest paths from node Y to U for Case (2.1) is given by

$$\begin{aligned} L_{diff(2.1)} &= (r_2 - r_1 + 1) \sum_{j=1}^{(r_1-1)/2} (j r_1 - 1) \\ &= (r_2 - r_1 + 1) \frac{(r_1 - 1)}{2} \left(\frac{r_1^2}{4} + \frac{r_1}{4} - 1 \right) \end{aligned}$$

Case (2.2) y -coordinate of $U > y$ -coordinate of Y

In Figure A11.4 we determine the path lengths and number of paths from U with coordinates $((r_1 + 1) / 2, 1)$ to nodes in the $r_1 \times (r_1 - 1) / 2$ mesh.

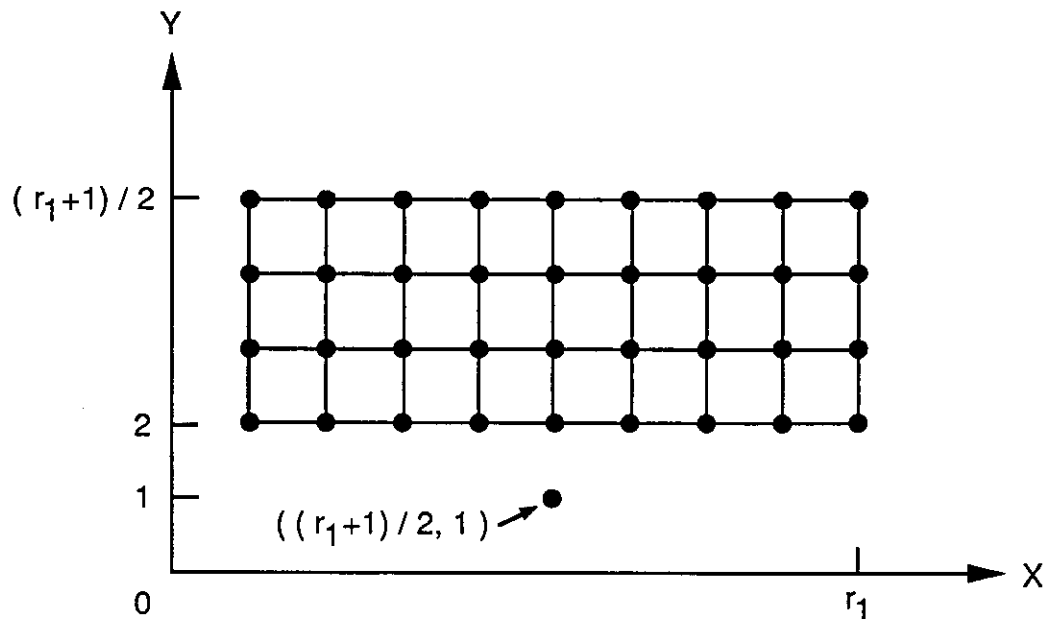


Figure A.11.4 : Case 2.2 - y -coordinate of Node $U > y$ -coordinate of Node Y

Note that when Y is located in region 2 and y -coordinate of $U > y$ -coordinate of Y then there are

$$(r_2 - r_1 + 1) \frac{(r_1 - 1)}{2}$$

possible locations of Y .

In order to compute the path lengths from Y to an arbitrary node U belonging to the $r_1 \times (r_1 - 1) / 2$ mesh shown in Figure A11.4 we consider two cases - when U belongs to a $r_1 \times (r_1 + 1) / 2$ mesh (Figure A11.5) and when U belongs to a linear array of nodes consisting of r_1 nodes with Y at the center (Figure A11.6). The number and the sum of the length of shortest paths for Case (2.2) will be the difference of these two cases.

Case (2.2.1) Figure A11.5

Consider the case illustrated in Figure A11.5. We find the sum of the shortest paths from node Y with coordinates $((r_1 + 1) / 2, 1)$ to an arbitrary node U in the $r_1 \times (r_1 + 1) / 2$ mesh.

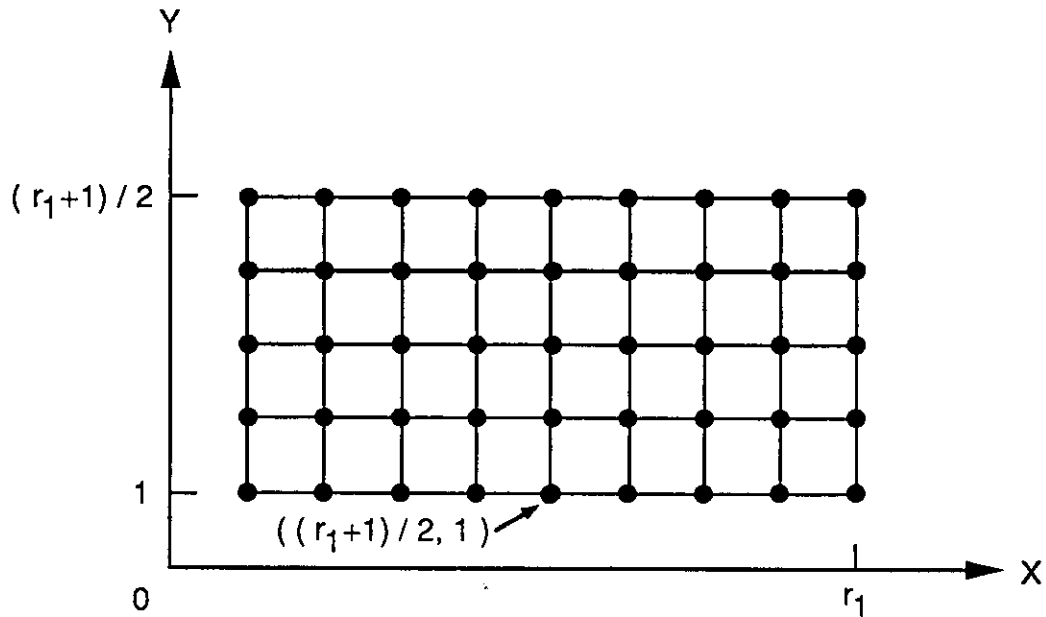


Figure A.11.5 : Case 2.2.1 - $r_1 \times (r_1 + 1) / 2$ mesh

From analysis similar to that of Appendix 3 we obtain the sum of the path lengths from Y to U as

$$D_{diff(2.2.1)} = \frac{r_1 + 1}{2} \left[\left(\frac{r_1 + 1}{2} \right)^2 - (r_1 + 1) \frac{(r_1 + 1)}{2} + r_1 \frac{(r_1 + 1)}{2} \right]$$

$$\begin{aligned}
& + r_1 \left[1 - \left(\frac{r_1 + 1}{2} + 1 \right) + \frac{r_1 + 1}{4} \left(\frac{r_1 + 1}{2} + 1 \right) \right] \\
& = \frac{r_1^3}{4} + \frac{r_1^2}{8} - \frac{r_1}{4} - \frac{1}{8}
\end{aligned}$$

Case (2.2.2) Figure A11.6

Consider Figure A11.6 where Y is the center of a linear array of r_1 nodes.

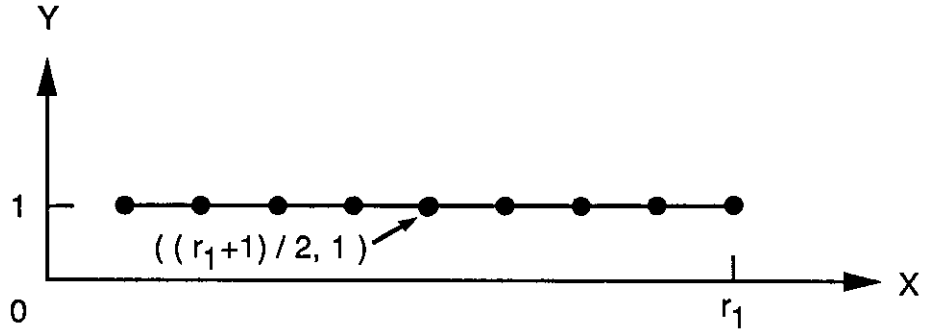


Figure A.11.6 : Case 2.2.2 - Linear Array of r_1 nodes

The sum of the path lengths from Y to an arbitrary node on the linear array is

$$\begin{aligned}
D_{diff(2.2.2)} &= 2 \sum_{i=1}^{(r_1-1)/2} i \\
&= \frac{(r_1^2 - 1)}{4}
\end{aligned}$$

$$\begin{aligned}
\therefore D_{diff(2.2.1)} - D_{diff(2.2.2)} &= \frac{r_1^3}{4} + \frac{r_1^2}{8} - \frac{r_1}{4} - \frac{1}{8} - \frac{r_1^2}{4} + \frac{1}{4} \\
&= \frac{(2r_1 - 1)(r_1^2 - 1)}{8}
\end{aligned}$$

Since there are

$$(r_2 - r_1 + 1) \frac{(r_1 - 1)}{2}$$

possible locations of Y for Case 2.2, hence we obtain the sum of the path lengths from a node Y to an arbitrary node U belonging to the $r_1 \times (r_1 - 1) / 2$ mesh as shown in Figure A11.4 as

$$D_{diff(2.2)} = \frac{1}{16} (2r_1 - 1)(r_1^2 - 1)(r_2 - r_1 + 1)(r_1 - 1)$$

The number of paths from Y to an arbitrary node U belonging to the $r_1 \times \frac{r_1 - 1}{2}$ mesh as shown in Figure A11.4 is

$$\frac{r_1 - 1}{2} r_1$$

For all possible locations of Y for Case 2.2 we have

$$L_{diff(2.2)} = \frac{1}{4} (r_2 - r_1 + 1)(r_1 - 1)^2 r_1$$

Therefore for Case 2 when Y is in region 2 we find

$$D_{diff(2)} = \frac{1}{96} (r_2 - r_1 + 1)(r_1^2 - 1)(5r_1^2 - 6r_1 - 3) + \frac{1}{16} (2r_1 - 1)(r_1^2 - 1)(r_2 - r_1 + 1)(r_1 - 1)$$

$$L_{diff(2)} = (r_2 - r_1 + 1) \frac{(r_1 - 1)}{2} \left(\frac{r_1^2}{4} + \frac{r_1}{4} - 1 \right) + \frac{1}{4} (r_2 - r_1 + 1)(r_1 - 1)^2 r_1$$

Case (3) Node Y is in region 3

Consider Figure A11.7 where node Y is located at $((r_1 + 1)/2, (r_1 + 1)/2)$ and U is an arbitrary node on the $i \times j$ mesh such that i and j range from $(r_1 + 1)/2$ to $r_1 - 1$. From analysis similar to that of Appendix 3 we observe that the distance from a node Y whose coordinates are (x, y) such that $x = y = (r_1 + 1)/2$ to an arbitrary node in the mesh of Figure A11.7 is

$$D(x, y) = j \left[x^2 - (i + 1)x + i \frac{(i + 1)}{2} \right] + i \left[y^2 - (j + 1)y + j \frac{(j + 1)}{2} \right]$$

Therefore when Y is in region 3 we have

$$D_{diff(3)} = \sum_{i=(r_1+1)/2}^{r_1-1} \sum_{j=(r_1+1)/2}^{r_1-1} D(x, y) = \frac{1}{96} (r_1^2 - 1)(r_1 + 1)(3r_1 - 1)(2r_1 - 3)$$

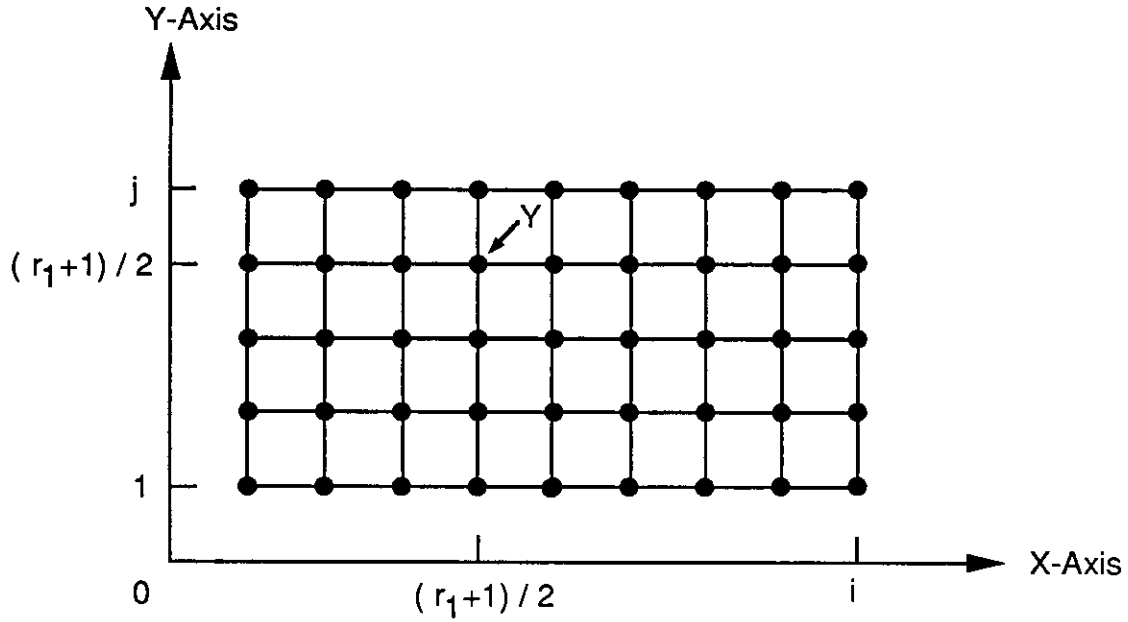


Figure A.11.7 : Node Y is in Region 3

The number of shortest paths from Y to U when Y is in region 3 is given by

$$L_{diff(3)} = \sum_{i=(r_1+1)/2}^{r_1-1} \sum_{j=(r_1+1)/2}^{r_1-1} (i j - 1)$$

$$= \frac{1}{64} (r_1 - 1)^2 \left[(3 r_1 - 1)^2 - 16 \right]$$

Since there are four regions identical to region 2 and four regions identical to region 3 hence we have

$$D_{diff} = D_{diff(1)} + 4 D_{diff(2)} + 4 D_{diff(3)}$$

and

$$L_{diff} = L_{diff(1)} + 4 L_{diff(2)} + 4 L_{diff(3)}$$

By summing up the terms and simplifying we obtain

$$D_{sum} - D_{diff} = \frac{2}{3} r_2^5 - \frac{2}{3} r_2^3 - \frac{1}{2} r_2^2 r_1 (r_1^2 - 1)$$

$$+ r_2 \left(\frac{7}{24} r_1^4 - \frac{5}{12} r_1^2 + \frac{1}{8} \right) - \frac{r_1}{24} (r_1^4 - 2 r_1^2 + 1)$$

and

$$L_{sum} - L_{diff} = r_2^4 - r_2^2 r_1^2 + \frac{1}{2} r_2 r_1 (r_1^2 - 1) - \frac{1}{16} (r_1^4 - 2 r_1^2 + 1)$$

If $r_2 \gg r_1$ and in particular when $r_1 = O(\sqrt{r_2})$ we obtain

$$\begin{aligned} \bar{d} &= \frac{D_{sum} - D_{diff}}{L_{sum} - L_{diff}} \\ &= \frac{2}{3} r_2 + \frac{2}{3} \frac{(r_1^2 - 1)}{r_2} - \frac{5 r_1 (r_1^2 - 1)}{6 r_2^2} + \frac{(6 r_1^4 - 7 r_1^2 + 1)}{6 r_2^3} \dots \end{aligned}$$

Therefore the average distance \bar{d} between two nodes U and Y such that U and Y belong to a $r_2 \times r_2$ mesh, Y is the center of a $r_1 \times r_1$ mesh, where $r_2 > r_1$, and node U lies outside the $r_1 \times r_1$ mesh is approximately $\frac{2}{3} r_2 + \frac{2}{3} \frac{r_1^2}{r_2} - \frac{2}{3} \frac{1}{r_2}$, assuming r_1 is $O(\sqrt{r_2})$.

