Complex Network Topology Learning Using ANN

T- 1247

MS Final Project

[illegible]

T01247

Developed by:

Sadia Rasheed
89-CS/MS/02
Shireen Tahira
92-CS/MS/02

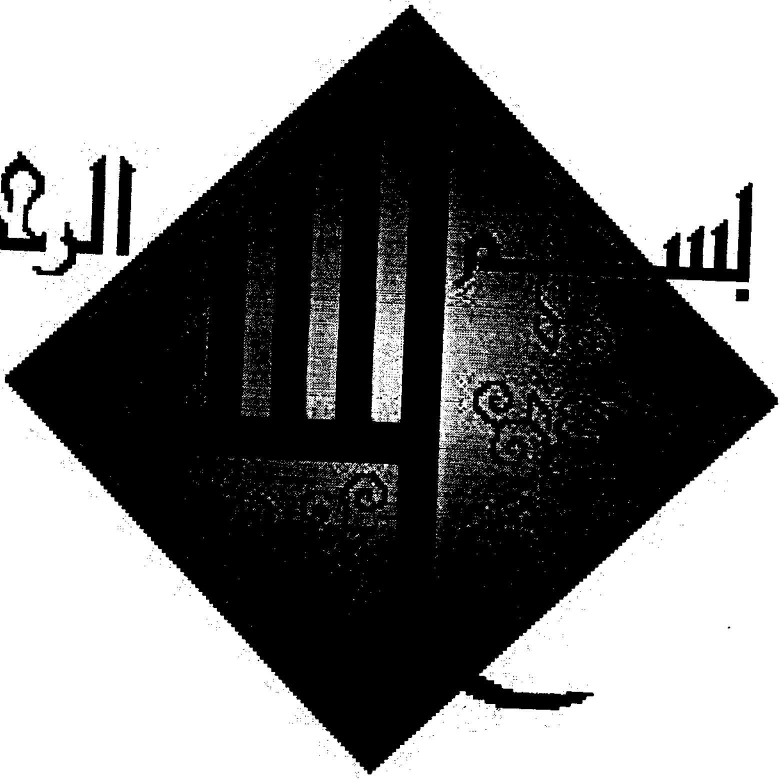
Supervised by:

Dr.Tauseef ur Rehman
Dr. Sikander Hyaat

Faculty of Applied Sciences
International Islamic University Islamabad
(2005)

[illegible] Acc. No. (FMS) T-1247

بسم الله الرحمن الرحيم



29-11-2005

International Islamic University, Islamabad Acc. No. (TM8) T-1247

Department of Computer Science

[illegible]

FINAL APPROVAL

It is certified that we have read the project report submitted by Miss **Shireen Tahira** and **Miss Sadia Rasheed**. It is our judgment that this project is of sufficient standard to warrant its acceptance by International Islamic University, Islamabad for the MS Degree in Computer Science.

COMMITTEE:

External Examiner:

Mr. Shaftab Ahmad,
Senior Principal Engineer,
PAEC.

Shaftab Ahmad

Internal Examiner:

Mr. Asim Munir,
Faculty Member,
Department of Computer Science,
IIUI.

[illegible]

Supervisors:

Dr. S. Tauseef-ur-Rehman,
EX-Head,
Department of Telecom. Engg.,
IIUI.

[illegible]

&

Dr. Sikandar Hayat Khiyal,
Head,
Department of Computer Science,
IIUI.

[illegible]

A dissertation submitted to the

Department Of Computer Science,

Faculty of Applied Sciences,

International Islamic University, Islamabad

as a partial fulfillment of the requirement

for the award of the degree of

MS in Computer Sciences.

Dedication

Dedicated to Our Parents

After Allah Almighty, We are completely indebted to our parents for this dissertation, whose affection has always been a source of encouragement for us, and whose prayers always turn out to be a key to our success.

Acknowledgment

we bestow all praises, acclamation and appreciation to *Almighty Allah*, The Most Merciful and Compassionate, The most Gracious and Beneficent, Whose bounteous blessings enabled us to pursue and perceive higher ideals of life. All praises for His *Holy Prophet Muhammad (SAW)* who enabled us to recognize our Lord and Creator and brought to us the real source of knowledge from Allah, *The Qur'an*, and who is a role model for us in every aspect of life.

We would consider it a proud privileged to express our cordial gratitude and deep sense of obligation to our reverend supervisors *Dr. Tausee-ur-Rehman, Head, department of Telecommunication Engineering* and *Dr. Sikandar Hayat Khiyal, Head, department of Computer Science, Faculty of Applied Sciences, International Islamic University Islamabad,* for their dexterous guidance, inspiring attitude, untiring help and kind behavior throughout the project efforts and presentation of this manuscript.

We will not be out of place to express our profound admiration for our parents and brothers for their prayers for our success. It was mainly due to our family's moral support and financial help during our entire academic career that enabled us to complete our work dedicatedly. We once again would like to admit that we owe all our achievements to our most loving parents, who mean most to us, for their prayers are more precious then any treasure on earth.

Sadia Raheed
89-CS/MS/02
Shireen Tahira
92-CS/MS/02

Declaration

We hereby, declare that we have developed this algorithm simulation software and the accompanied thesis entirely on the basis of our personal efforts made under the guidance of our teachers and supervisors. No portion of the work presented in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute.

Sadia Raheed

89-CS/MS/02

Shireen Tahira

92-CS/MS/02

Project in Brief

Project Title	Complex Network Topology Learning Using Artificial Neural Network.
Objective	To develop an algorithm which should determine an optimize routing path to send a packet form source to destination in a network.
Undertaken by	Sadia Rasheed, Shireen Tahira 89-CS/MS/02 92-CS/MS/02
Supervised by	Dr. S. Tauseef-Ur-Rehman Head, department of Telecommunication Engineering & Dr. Sikandar Hiyat Head, department of Computer Science, Faculty of Applied Sciences, International Islamic University Islamabad.
Starting Date	September 2004
Completion Date	
Tool Used	Microsoft Visual C++
Operating System	Windows 2000/Windows XP
System Used	Intel Pentium IV

Abstract

This Thesis presents an implementation of SSQ-Routing algorithm an algorithm based on Q-routing and artificial neural network (ANN) to solve the shortest path problem for routing in networks. The SSQ-Routing uses the Q-routing algorithm, a network routing algorithm based on Q-learning, a method from the emerging field of reinforcement learning. In this framework, the routing information at individual nodes is maintained as Q-value estimates of how long it will take to send a packet to any particular destination via each of the node's neighbors. These Q-values are updated through exploration as the packets are transmitted. Neural networks have been used with some success to perform Q-learning, and would seem to be a possible method to allow Q-routing to scale well beyond its initial table-based implementation. This thesis attempts to apply a neural network as a function approximator in an online reinforcement learning task, a field where neural networks have been used with varying degrees of success in the past. A discussion of the factors involved in neural network function approximation in reinforcement learning is provided. The main contribution of this work is the faster adaptation and improved quality of routing policies.

Table of Contents

Chapter No.	Contents	Page #
1. Introduction		
	1.1 communication Networks	1
	1.2 Routing Problem	2
	1.3 Neural Networks and Function Approximation	5
2. Literature Review		15
3. Problem Definition		
	3.1 Problem Description	18
	3.2 Proposed Solution	19
4. Design		
	4.1 Algorithm Formulation	
	4.1.1 Network Load Level	21
	4.1.2 Traffic Pattern	21
	4.1.3 Network Topology	22
	4.1.4 A Routing Policy	22
	4.1.5 Performance Measure	22
	4.1.6 Average Packet Delivery Time	23
	4.1.7 Number of Packets in the Network	23
	4.1.8 Network Topology used	23
	4.1.9 Neural Network (Kohonen's Network) Architecture used	24
	4.2 The Algorithm	25
	4.3 Data Flow Diagrams	
	4.3.1 Level 0 Data Flow Diagram	26
	4.3.2 Level 1 Data Flow Diagram	26
	4.3.3 Level 2 Data Flow Diagram	27
	4.4 Flow Charts	
	4.6.1 Main Program Flow Chart	28

5. Implementation

5.1 Division of Functional Units

5.1.1 Packet Generator 29

5.1.2 Routers (Nodes) 29

5.2 Coding Details

5.2.1 Packet Generator 30

5.2.1.1 Listening for the Request 30

5.2.1.2 Connection to Router 31

5.2.1.3 Packet Generate Request 31

5.2.2 Routers 32

5.2.2.1 Listening for Request from other Routers 33

5.2.2.2 Connection to Packet Generator 33

5.2.2.3 Connection to Other Routers 33

5.2.2.4 Algorithm Implementation 34

5.2.2.5 Call for FIND_NEXT_NODE 35

5.2.2.6 Call for UPDATE_QW 37

6. Testing

6.1 System testing 33

6.2 Development testing 33

6.3 Integration testing 34

6.3.1 Integration tests planning 34

6.3.2 Test planning activities 34

6.4 Stress testing 35

6.5 Acceptance testing 35

6.6 Performance testing 36

6.7 Reliability testing 36

6.8 Security testing 37

6.9 Testing for Simulation system 37

6.9.1 Prerequisites 37

6.9.2 TC –SSQ-Routing- PG – Packet Generation 38
6.9.3 TC – SSQ-Routing –Router – Find_next_node 39
6.9.4 TC - SSQ_Routing – Router– Update_Qweight 40
6.9.5 TC - SSQ_Routing –PG & Router Integration 41
6.9.6 TC - SSQ_Routing –PG & Router Integration 42
6.9.7 TC - SSQ_Routing –Router ID-0 & Router ID-1 Integration 43
6.9.8 TC - SSQ_Routing –Router ID-0 & Router ID-1 Integration 44

7. Results and Discussions

7.1 Experimental Setup 45
7.2 Learning at Constant Loads 45
7.3 Adaptation to Varying Traffic Conditions 51
7.4 Adaptation to Changing Network Topology 53
7.5 Load level Sustenance 58
7.6 Conclusions and Future Work 63

References

Appendix A: snapshots

A1. Packet Generator 67
A2. Router (Node-0) 68
A3. Router (Node-1) 69
A4. Router (Node-2) 70
A5. Router (Node-3) 71
A6. Router (Node-4) 72
A7. Router (Node-5) 73
A8. Router (Node-6) 74
A9. Router (Node-7) 75
A10. Router (Node-8) 76

List of Figures

Figure	Page#
Figure 4.1 Network Topology 6X6 irregular grids	24
Figure 4.2 Kohonen's Network Layers	24
Figure 4.3 Level 0 DFD	27
Figure 4.4 Level 1 DFD	27
Figure 4.5 Level 2 DFD	28
Figure 4.6 Flow Chart for the main program	29
Figure 7.1 Average packet delivery time at a low load level for the grid topology	47
Figure 7.2 Number of packets in the network for the grid topology at a low load level	48
Figure 7.3 Average packet delivery time for 7-D hypercube at low load	48
Figure 7.4 Number of packets for 7-D hypercube at low load	49
Figure 7.5 Average packet delivery time for grid topology at a medium load level	50
Figure 7.6 Number of packets in the network for grid topology at a medium load level	51
Figure 7.7 Average packet delivery time for 7-D hypercube at medium load	52
Figure 7.8 Number of packets for 7-D hypercube at medium load	52
Figure 7.9 Average packet delivery time for grid topology at high load	53
Figure 7.10 Number of packets in the network for grid topology at high load	54
Figure 7.11 Number of packets for 7-D hypercube at high load	55
Figure 7.12: Number of packets for 7-D hypercube at high load	56
Figure 7.13: Adaptation to a change in traffic pattern. Single run is shown	57
Figure 7.14: Grid Topology before link went down	57
Figure 7.15: Grid Topology after link went down	58
Figure 7.16: Adaptation to change a in the network topology	58
Figure 7.17: Average Packet Delivery time after convergence	60
Figure A1 Interface for Packet Generator	67
Figure A2 Interface for Router (Node-0)	68
Figure A3 Interface for Router (Node-1)	69
Figure A4 Interface for Router (Node-2)	70
Figure A5 Interface for Router (Node-3)	71

Figure A6 Interface for Router (Node-4) 72
Figure A7 Interface for Router (Node-5) 73
Figure A8 Interface for Router (Node-6) 74
Figure A9 Interface for Router (Node-7) 75
Figure A10 Interface for Router (Node-8) 76

Publication **77**

List of Table

Table	Page#
Table 5.1: Poisson distribution	30
Table 7.1: Parameters for the Algorithms	45
Table 7.2: Load level range	45

CHAPTER 1

Introduction

1. Introduction

Computer networks play an important and ever increasing role in the modern world. The development of the Internet, the corporate intranet, and mobile telephony have extended the reach of network connectivity to places that ten years ago would have been unthinkable. The result of these trends is that the performance of network hardware and software is being tested by the increasing load placed upon them, and new ways have to be found to solve the problems. Modern networking applications are often based upon frequently changing ad hoc network topologies and require that the network protocols running these applications are able to withstand outages of parts of the network.

In this Thesis a routing algorithm that attempts to solve some of the problems faced in dynamic, unreliable, and congested networks is suggested.

1.1 Communication Networks

In a communication network, information is transferred from one node to another as data packets. The process of sending a packet from its source node ‘s’ to its destination node ‘d’ is referred to as *packet routing*. Normally it takes multiple “hops” to transfer a packet from its source to destination node. On its way, the packet spends some time waiting in the queues of intermediate nodes while they are busy processing the packets that came earlier. Thus the delivery time of the packet, defined as the time it takes for the packet to reach its destination, depends mainly on the total time it has to spend in the queues of the intermediate nodes.

Normally, there are multiple routes that a packet could take, which means that the choice of the route is crucial to the delivery time of the packet for any source, destination(s,d) pair. If there was a global observer with current information about the queues of all nodes in the network, it would be possible to make *optimal* routing decisions: always send the packet through the route that has the shortest delivery time at the moment. In the real world, such complete, global information is not available, and the performance of the global observer has an upper bound on actual performance. Instead, the task of making routing decisions is shared by all the nodes, each using only local information. Thus, a routing policy is a collection of local decisions at the individual nodes. When a node x receives a packet P(d) destined for node d, it has to choose one of its neighboring nodes y such that the packet reaches its destination as quickly as possible.

The simplest such policy is the shortest-path algorithm, which always routes packets through the path with the minimum number of hops. This policy is not always good because some intermediate nodes, falling in a popular route, might have large queues. In such cases it would be better to send the packet through another route that may be longer in terms of hops but results in shorter delivery time. Hence as the traffic builds up at some popular routes, alternative routes must be chosen to keep the average packet delivery time low. This is the key motivation for SSQ- routing algorithm that learn alternate routes through exploration as the current routing policy begins to lead to degraded performance.

Learning effective routing policies is a challenging task. In SSQ-routing algorithm implementation, network makes routing decisions using Q-routing in which Kohonen's Neural Network is used as function approximator. Neural network has been used to some success to perform Q-learning. There has been notable success with this method using Q-learning and a neural network which approximate the value function. This dissertation presents improvements in reinforcement learning. This algorithm aims to be stable as possible as under high loads while performing in less extreme situations.

1.2 Routing Problem

Depending on the network topology, there could be multiple routes from source 's' to destination 'd' and hence the time taken by the packet to reach destination from source depends on the route it takes. So the overall goal that emerges can be stated as: What is the optimal route from a given source node to a given destination node in the current state of the network? The state of the network depends on a number of network properties like the queue lengths of all the nodes, the condition of all the links (whether they are up or down), condition of all the nodes (whether they are up or down) and so on.

If there were a central observer that had information about the current state (i.e. the packet queue length) of all the nodes in the network, it would be possible to find the best route using the Weighted Shortest Path Routing algorithm [Dijkstra's 1959].

If q_i is the waiting time in the packet queue of node x_i and δ is the link delay (same for all links) then the cost of sending a packet P(s; d) through node x_i will add $(q_i + \delta)$ to the delivery time of the packet. The weighted shortest path routing algorithm finds the route for which the total delivery time of a packet from source to destination node is minimum.

Such a central observer does not exist in any realistic communication system. The task of making routing decisions is therefore the responsibility of the individual nodes in the network. There are two possible ways of distributing this responsibility among the different nodes:

a. The first approach is that the source node computes the best route R to be traversed by the packet to reach its ultimate destination and attaches this computed route to the packet before it is sent out. Each intermediate node that receives this packet can deduce from R to which neighboring node this message should be forwarded. This approach is called Source Routing and it assumes that every (source) node has complete information about the network topology. This assumption is not useful, because knowledge about the network topology alone is not enough. To make an optimal routing decision one has to also know the queue lengths of all the nodes in then network. Also in a dynamically changing network, some links or nodes might go down and come up later, meaning that even the topology of the network is not fixed at all times. Moreover, each packet carries a lot of routing information (its complete route R) which creates a significant overhead. As a result, this approach is not very useful for adaptive routing in dynamically changing networks.

b. The second approach is that the intermediate nodes make local routing decisions.

As a node receives a packet for some destination d it decides to which neighbor this packet should be forwarded so that it reaches its destination as quickly as possible.

The destination index d is the only routing information that the packet carries. The overall route depends on the decisions of all the intermediate nodes. The following requirements have been identified for this approach [Tanenbaum 1989; Gouda 1998].

Each node in the network needs to have:

- For each of its neighbors, an estimate of how long it would take for the packet to reach its destination when sent via that neighbor.
- A heuristic to make use of this information in making routing decisions;
- A means of updating this routing information so that it changes with the change in the state of the network; and
- A mechanism of propagating this information to other nodes in the network.

This approach has lead to adaptive distance vector routing algorithms. Distributed Bellman-Ford Routing [Bellman 1958], is the state of the art and most widely used and cited distance vector routing algorithm.

In the framework of the second approach, where all the nodes share the responsibility of making local routing decisions, the routing problem can be viewed as a complex optimization problem whereby each of the local routing decisions combine to yield a global routing policy. This policy is evaluated based on the average packet delivery time under the prevailing network and traffic conditions. The quality of the policy depends, in a rather complex manner, on all the routing decisions made by all the nodes. Due to the complexity of this problem, a simplified version is usually considered. Instead of a globally optimal policy, one tries to find a collection of locally optimal ones:

When a node x receives a packet P(s; d) destined to node d, what is the best neighbor y of x to which this packet should be forwarded so that it reaches its destination as quickly as possible?

This problem is difficult for several reasons (as will be discussed in more detail later in this thesis):

a. Making such routing decisions at individual nodes requires a global view of the network which is not available; all decisions have to be made using local information available at the nodes only.

b. There is no training signal available for directly evaluating the individual routing decisions until the packets have finally reached their destination.

c. When a packet reaches its destination, such a training signal could be generated, but to make it available to the nodes that were responsible for routing the packet, the signal would have to travel to all these nodes thereby consuming a significant amount of network resources.

d. It is not known which particular decision in the entire sequence of routing decisions is to be given credit or blame and how much (the credit assignment problem).

These issues call for an approximate greedy solution to the problem where the routing policy adapts as routing takes place and overhead due to exploration is minimum.

1.3 Neural Networks and Function Approximation

Neural networks are a biologically inspired approach to machine learning that can learn to approximate complex mathematical functions. Neural networks have been used in proposed solutions to all sorts of problems, from financial forecasting to automated control systems, and have achieved some very significant successes in many of these areas. Neural networks are a very active area of research. Whilst it is obvious that no one technology can be a solution for every problem, neural networks have proved to be extraordinarily versatile and effective.

This Thesis attempts to apply a neural network as a function approximator in an online reinforcement learning task, a field where neural networks have been used with varying degrees of success in the past.

CHAPTER 2

Literature Review

2. Literature Review

Supervised and Unsupervised learning are two quite different techniques of learning. As the names suggest, supervised learning involves learning with some supervision from an external source (for example a teacher) whereas unsupervised learning does not. An example of supervised learning is a student taking an examination, having it marked and then being shown which questions they answered incorrectly. After being shown the correct answers, the student should then learn to answer those questions successfully as well. An example of unsupervised learning is someone learning to juggle by themselves. The person will start by throwing the balls and attempting to catch them again. After dropping most of the balls initially, they will gradually adjust their technique and start to keep the balls in the air.

Reinforcement learning is a type of unsupervised learning relatively new and emerging area of machine learning theory. Reinforcement learning aims to develop successful techniques for learning complex strategies from limited data in a goal-directed manner. The definition of reinforcement learning given by Sutton and Barto is "Reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning problem" [1].

A reinforcement learning problem is a problem where supervised learning cannot easily be used because there are no training sets, insufficient data or external knowledge that can be applied, a reward signal. Maximizing this reward signal is the goal of reinforcement learning. Reinforcement learning algorithms develop a *policy*, usually defined as a mapping of states and subsequent actions to expected reward.
The field is still quite young, and there are problems that are still being researched which are proving difficult to solve. For example, the problem of temporal credit assignment, when given a reward determining which set of long or short term actions were responsible for the reward, is a very difficult one. In some cases it will obviously be better to take a lower short term reward for a greater long term reward. However, the reinforcement learning approach is very promising even in these early stages, and seems

a good fit for the network routing problem as Boyan and Littman [2] showed. Temporal difference (TD) learning, is a major part of reinforcement learning theory, and covers a number of methods, such as *TD*(0), *TD*(λ), Q-learning and Sarsa [1]. What these methods have in common is the concept of accumulating the effect of differences in reward over time, hence "temporal difference".

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \qquad (2.1)$$

The equation 2.1 sometimes referred to as the "Q-function" for quite obvious reasons, is the policy update rule used in Q-learning. Confusingly, it is also the same function used by the Sarsa algorithm, although the two techniques have differences in other areas. The update rule states how the value of the Q-function is updated for each state s_t, and for every action *at* that can be taken from that state. r_{t+1} is the immediate reward from this action, and $Q(s_{t+1}, a_{t+1})$ is the discounted future reward from this state. The value γ, known as the *discounting rate* determines whether the algorithm is greedy in the short term or longer term.

A greedy policy is a policy that always takes the action that is estimated to have the highest reward. This is sometimes the best course of action, for example in a static environment. An online algorithm must keep trying other actions to ensure that the reward estimates are accurate. To do this a probability, known as epsilon (Є), is given that in any state there is a probability of epsilon that a random action will be taken. A policy like this is known as Є -greedy.

Q-learning is an off-policy reinforcement learning method, which means it learns a policy, called an estimation policy that is separate from the policy which it acts upon, the behavior policy. This allows, for example, an estimation policy that is greedy while the behavior policy may be able to sample other actions, for example epsilon-greedy. An algorithm closely related to Q-learning, but working in an on-policy manner, is Sarsa. Sarsa is close enough to Q-learning to be able to be substituted directly in some cases.

Neural networks are a technique for pattern recognition and function approximation based originally on ideas from biology and the study of neurons [3], although the theory behind neural networks is based on statistical foundations [4]. Neural networks seek to mimic the apparently simple, but extremely effective, structure of the human brain to able to approximate functions that would be far too complex to program by hand. Neural networks consist of units, corresponding to neurons, and weighted connections between them. The values of each unit are "fed forward" via the weighted connections to other units. The artificial neurons have an output value based on a specific "activation function" of which they are many, some more biologically realistic, and others better suited for other tasks. The most common are linear units and sigmoid units. Neural networks may also be made up of building blocks of smaller networks, for example perceptrons. When two layers of units are combined, such a neural network can approximate any continuous function. When three layers of units are combined any arbitrary function that may be required can be approximated [5].

Learning in neural networks is commonly achieved using the backpropagation algorithm [4]. Backpropagation takes the overall error of the output of a neural network, as measured against training data, and propagates it back through the network, calculating the error of each individual weight. The weights of the network can then be updated according to a weight update rule. Backpropagation has proved successful at solving some quite complex problems [5], including complex control tasks such as engine control, and speech and image recognition. The typical use of backpropagation is in multiple iterations over static data sets, but it may be used in online applications as a means of calculating error gradients.

Neural networks have been used before in network routing type problems, for example, using radial basis functions to optimize call set-up in a telephone network with a static topology [6]. This situation could be seen as analogous to a computer network, but the two situations are quite different. A telephone network has a relatively static topology, and quite predictable usage patterns. A computer data network often has a changeable, ad hoc topology, greatly varying usage patterns and a lack of control over bandwidth allocation. Because of this, a controller trained on prior data cannot be guaranteed to perform well under the conditions, and must be rebuilt when the topology is altered.

Neural networks have often been used in reinforcement learning. The use of neural networks in this situation is sometimes called "neuro-dynamic programming" because of the use of neural network and dynamic programming techniques together.

There have been notable successes with these methods, such as TDGammon, a reinforcement learning approach to playing backgammon. TDGammon uses temporal difference learning and a neural network which approximates the value function of the various board states in a game of backgammon. The current version of TDGammon, 3.0, is on a par with the best human players in the world today and easily the best computer backgammon player [1, 7].

Neural networks and the Sarsa algorithm were also used to good effect by Bazen et. al. to extract minutia information from fingerprint data [8]. Success with neural network function approximation and temporal difference learning was also shown with a control problem similar to network routing by Crites and Barto [9]. Crites and Barto [9] used a neural network function approximator in conjunction with temporal difference learning to control a system of elevators (lifts) and which was able to outperform all known heuristic elevator control algorithms.

Unfortunately the Q-learning algorithm is known to be unstable when used in conjunction with a linear function approximator [10, 11]. Baird [10] demonstrated an example that clearly shows this with the "star problem" a six-state Markov decision process (MDP) that has an exact solution, but when using a linear function approximator is guaranteed to fail to converge successfully. This means there is no guarantee that a linear approximator will be able to safely approximate the Q-function, and if convergence can not be guaranteed for the weaker condition of a linear approximator, then a non-linear approximator is also not guaranteed. A neural network therefore cannot guarantee to safely approximate the Q-function, although in some cases it may. Tsitsiklis and Van Roy [12] showed that all Temporal Difference algorithms are liable to become unstable when used with a non-linear function approximator, such as a neural network.

The solution Baird [10] used was a technique known as *Residual Algorithms*, which is essentially a gradient descent approach as opposed to a simple value maximization algorithm. Residual algorithms tend to be quite slow to learn and are really more suited to episodic tasks rather than infinite horizon tasks.

Eligibility traces, sometimes termed *TD*(λ) methods, are another techniques that can make function approximation more stable. Eligibility traces try to bridge the gap between temporal difference methods and Monte Carlo methods [1].

Discussion of Monte Carlo methods and dynamic programming are deemed outside the scope of this project due to the huge quantity of unnecessary detail that would require. Eligibility methods basically amount to another step of "back up" in the Q-function, so for example the last two known rewards are used plus the discounted sum of the remaining rewards, rather than the last known reward plus the discounted sum.

The Internet, with a capital "I", as opposed to the more general term internet or internetwork, is a global network of computer systems that originated in the United States as part of the military funded ARPANET which began in the early 1970s [13]. The Internet has pushed networking technology into the mainstream and it is without doubt the most important network, both in terms of technology advances and social impact, in the world. The number of hosts on the Internet is growing at an incredible rate and shows no sign of slowing down [14]. This growth has placed strain on the network infrastructure that was built on what were, at the time ARPANET was created, experimental technologies.

The Internet uses packet switching technology to transmit data, that is, data that to be transmitted over the Internet is split into small chunks, known as packets. These packets are then transmitted one at a time across the Internet where they are reassembled at the receiver. Packets on the Internet may not necessarily arrive in order, or at all, and may follow different routes to their destinations. Packet switching allows the Internet to be flexible in what physical media it can use to transmit data, as well as to be fairly resilient when faced with small amounts of data loss caused by line noise or other factors. The basic building blocks of the Internet are the TCP/IP suite of protocols [15], which may be modeled as a stack of protocols split into several layers [16]. The underlying protocol at the network layer, Internet Protocol or IP is a connection-less best effort protocol, meaning it has no connection establishment phase or authentication, and it does not provide a guarantee that the data sent will reach its destination [15]. Reliable delivery is Provided by the Transmission Control Protocol, or TCP.

However, the properties that make the Internet so effective and successful also make it vulnerable to "Internet Meltdown" or "congestion collapse" [17]. Several aspects of the underlying Internet technology are showing their age and reaching the point where other approaches may soon have to be explored if the growth rate and stability of the Internet is to be maintained. These areas include address allocation [15], routing and congestion [17].

Network congestion occurs when a part of a computer network is asked to transfer more data than its resources allow in a period of time. This results in data loss and degradation in performance. The problem of congestion occurs when the bandwidth of a specific link is exhausted by an upsurge in traffic; if packets cannot be sent down a congested link then the machine sending packets onto that link has to wait for the link to become free before sending any more. This means the sender has to buffer the packets it still has to send until the link is free, which requires that the sender have memory available to buffer the packets. If the backlog of packets gets too large for the sender to buffer, then any additional packets must be discarded or dropped. In reliable network protocols the response to a dropped packet is to resend the packet in question, and because of this, the effect of a dropped packet is multiplied. For every packet that is dropped from a reliable protocol it will be sent again at least once, adding to the congestion and therefore more packets will be dropped, and so on [15, 16].

Network congestion is increasingly becoming a problem on the Internet. In the mid-1980s the Internet was beginning to suffer from congestion problems and the possibility of congestion collapse [18]. TCP congestion control measures [17] such as Tahoe and Reno extensions have attempted to alleviate this problem, and TCP performance under load is much better than it was when the Internet cut over to TCP/IP on the 1st January 1983. The continuing growth of the Internet and the proliferation of high-bandwidth applications mean this issue is far from resolved however.

Much progress has been made in the field of congestion control in recent years, and problems such as congestion collapse [18] have largely been avoided [19] in the context of TCP, the most widely used transport protocol on the Internet. Congestion problems are by no means confined to TCP based applications however, and the area is still being very actively researched.

Congestion control can be approached at every level of the network protocol stack. At the data link layer, the hardware will do its best to avoid congestion, for example Ethernet will do exponential back off in an attempt to avoid packet collisions on an Ethernet segment [15]. Congestion control at this level is usually very simple because it is usually implemented in hardware. Ethernet is simple enough to be implemented in very small integrated packages and Ethernet hardware is accordingly very cheap.

At the network layer the queuing policy and routing algorithm come into play along with the possibility of a virtual circuit switching technology. Virtual circuit switching technology will not be discussed here for space reasons. Active queue management is an active research topic, and algorithms such as Random Early Detection, or RED, [20] and BLUE [9] have proved very successful in simulation, although practical use of these technologies is still quite limited [21]. Active Queue Management seeks to drop packets not simply based on the order they are received at a router. For example, RED drops packets randomly with a specific probability before congestion becomes a problem, in an attempt to prevent congestion occurring. These algorithms have also been extended to implement fair queuing, an attempt to make sure every traffic flow gets access to network resources fairly, and differential services, which allow different types of network packet to be given different priorities in network resource allocation. There have also been proposed algorithms that use cost metrics to price the routing of a packet through a network, so a user willing to pay more could route packets over a faster or more reliable network. Routing algorithms are discussed in the next section.

Active queue management may also be applied at the transport layer, for example the Explicit Congestion Notification, ECN [22] extension to TCP. This adds a single bit to TCP packets so congestion can be signaled explicitly rather than implicitly by dropping packets at the network layer. This approach works best with connection based protocols because rate control is more easily applied to a connection than single packets, rate control for TCP is performed by altering the transmission window size of the sender [15]. The other main Internet transport layer protocol, the User Datagram Protocol or UDP [15] must be treated separately by an active queue management algorithm, because it is best effort and connection-less much like the underlying IP. UDP tends to be bandwidth greedy and can be a major factor in network performance degradation. An ECN bit has also been proposed for IP, but it is a more recent proposal and so not many TCP/IP stacks implement it today.

The setting of timeout values is also very important because it impacts directly on how much excess traffic is put onto the network when congestion occurs. Timeout values determine how long a packet may go unacknowledged by the receiver, after a certain length of time the sender will assume the packet is lost and resend it. Getting the timeout values perfect is very difficult and depends to a large extent on the characteristics of the network involved. TCP has received a lot of attention, and there are many extensions proposed that may improve it further, including Selective Acknowledgements, or SACK, which sends less acknowledgement packets and Fast Retransmit, which attempts to guess when a packet has been lost and retransmit it before it timeouts [23].

Applications may also attempt their own congestion control. If an application uses TCP it is usually counter-productive to try and implement another layer of congestion control at the application level unless the application author has some extremely domain-specific knowledge. UDP applications, in particular streaming applications like Internet radio and video conferencing, can benefit enormously from being able to handle their own congestion control due to their greedy nature. However, it is a very difficult problem to calculate the bandwidth available across a wide variety of links and routers between two hosts. A lot of the research in this area seems to be proprietary and quite experimental.

Routing algorithms are methods for finding the best way to get from a host A to another host B. This may be via a large number of other machines or it may be in the next room. On a small, simple network the problem is almost trivial, statically allocating routes and defining them by hand, but when dealing with a huge internetwork such as the Internet this is not possible. A heavily interconnected network such as the Internet has many routes from one host to another, and these routes span many different types of link with different bandwidth and latency characteristics. Calculating the best route through such a complex system is computationally intractable and impossible to do by hand. A routing algorithm first seeks to deliver a packet successfully, and if possible deliver it by the quickest route available. If many packets are routed through the same router a bottleneck occurs, and the whole network slows down, a good routing algorithm will try to route around a bottleneck router to minimize the effect of network congestion.

There are two types of routing algorithms, *inter-domain routing algorithms* and *intra-domain routing algorithms. Inter-domain routing algorithms* route packets between domains, that is, they find routes from one large area of the Internet, for example JANET, to another large area such as LINX. These large areas are called domains or autonomous areas. Inter-domain routing algorithms include the Exterior Gateway Protocol or EGP which is now considered obsolete, and the Border Gateway Protocol or BGP [24] which the modern Internet backbone runs on. There are many problems in the field of inter-domain routing, the sheer size of the routing tables for the Internet today is close to overwhelming some of the backbone routers and some have raised questions over whether BGP is a stable enough protocol for running such a large and complex network [25].

Intra-domain routing algorithms find routes within autonomous areas. An autonomous area could be as small as 2 or 3 machines or as large as tens of thousands on a corporate network. Intra-domain routing algorithms include Routing Information Protocol or RIP, version 1 and 2 [26], and Open Shortest Path First or OSPF [27].

RIP is a distance vector routing protocol, and very simple to implement. Distance vector routing, sometimes known as *Bellman-Ford*, after its inventors, is based on the idea of each router in a network keeping a vector of distances to every other node in the network. This vector is then distributed to its nearest neighbors, which update their distance vector on the basis of the information they are sent by their neighbors. In this way routing data propagates throughout the network and eventually converges to a stable routing policy.

RIP and distance vector routing suffer from some quite serious drawbacks unfortunately. Under certain circumstances the routing tables for a network will not stabilize, a problem often called the "count to infinity" problem [15]. Various partial solutions have been proposed to this but none are particularly elegant. RIP also has scalability problems, it can only handle networks with a maximum of 16 hops, and it generates a considerable amount of network traffic in larger networks. In 1979 RIP was retired from ARPANET.

OSPF is a link-state routing algorithm. Link-state routing attempts to build up a graph representing the network at each router and uses a shortest path algorithm, usually *Dijkstra's algorithm*, to find the shortest path to a route according to the graph it has built up. The graph representation is built up by routers exchanging link-state advertisements, or LSAs. These packets contain link-state information, that is, what links a router has, and to what machines. OSPF adds several advanced features that RIP didn't have, such as message authentication, load balancing and a hierarchical structure [27]. OSPF also uses metrics to calculate the cost of taking a route, rather than the RIP approach which is merely based on hop counts, that is the number of routers on the route. This allows, for example, a 128Kbps link to be preferred over a 9.6Kbps link. The metric used is a very important factor as to the optimality of the routes chosen, and ideally should be able to take into account how congested a link is.

There are several other routing algorithms that have been proposed. Many of these are applicable only to specific routing problems such as Massively Parallel Processor, or MPP networks [28] or virtual circuit switched networks, and do not work well in a complex, dynamic environment such as a packet switched computer network.

Of more interest is Q-routing, a routing algorithm that uses a technique known as Q-learning from the field of reinforcement learning to learn the best routes and constantly update them online [2]. Boyan and Littman [2] present very impressive results in their paper. The algorithm is dynamic and suited to networks with changing topologies and traffic levels; it aims to be as stable as possible under high loads whilst still performing well in less extreme situations. The Q-routing algorithm has been developed further by others [29], but the extensions proposed are rather incidental to the underlying algorithm.

In Q-routing, the routing decision maker at each node x makes use of a table of values $Q_x(y; d)$, where each value is an estimate, for a neighbor y and destination d, of how long it takes for a packet to be delivered to node d, if sent via neighbor y, excluding time spent in node x's queue. When the node has to make a routing decision it simply chooses the neighbor y for which $Q_x(y; d)$ is minimum. Learning takes place by updating the Q values.

On sending P(d) to y, x immediately gets back y's estimate for the time remaining in the trip, namely

$$Q_y(\hat{z}, d) = \min_{z \epsilon N(y)} Q_y(z, d) \qquad (2.2)$$

Where $N(y)$ denotes the set of neighbors of node y. if the packet spent q_x units of time in x's queue, then x can revise its estimate based on this feedback:

$$\Delta Q_x(y, d) = \eta(Q_y(\hat{z}, d) + q_x + \delta - Q_x(y, d)) \qquad (2.3)$$

Where η is the "learning rate "constant for all Q-values updates and δ is a transmission delay over the link between nodes x and y (assumed same for all links).

3. PROBLEM DEFINITION

As Computer networks play an important and ever increasing role in the modern world. The development of the Internet, the corporate intranet, and mobile telephony have extended the reach of network connectivity to places that ten years ago would have been unthinkable. The result of these trends is that the performance of network hardware and software is being tested by the increasing load placed upon them, and new ways have to be found to solve the problems that the original ARPANET engineers faced nearly 50 years ago.

Modern networking applications are often based upon frequently changing ad hoc network topologies and require that the network protocols running these applications are able to withstand outages of parts of the network. The emphasis on peer-to-peer technologies of such applications as Napster c and Gnutella demonstrates the desire to move away from centralized network architectures towards more loosely defined, distributed ones. A computer data network often has a changeable, ad hoc topology, greatly varying usage patterns and a lack of control over bandwidth allocation. Because of this, a controller trained on prior data cannot be guaranteed to perform well under the conditions, and must be rebuilt when the topology is altered.

However, the properties that make the Internet so effective and successful also make it vulnerable to "Internet Meltdown" or "congestion collapse". Several aspects of the underlying Internet technology are showing their age and reaching the point where other approaches may soon have to be explored if the growth rate and stability of the Internet is to be maintained. These areas include address allocation, routing and congestion.

Network congestion occurs when a part of a computer network is asked to transfer more data than its resources allow in a period of time. This results in data loss and degradation in performance. The problem of congestion occurs when the bandwidth of a specific link is exhausted by an upsurge in traffic; if packets cannot be sent down a congested link then the machine sending packets onto that link has to wait for the link to become free before sending any more. Much progress has been made in the field of congestion control in recent years, and problems such as congestion collapse have largely been avoided in the context of TCP, the most widely used transport protocol on the Internet. Congestion problems are by no means confined to TCP based applications however, and the area is still being very actively researched.

Efficient routing of packets in computer networks is a prerequisite for wide deployment of network-enabled devices, especially mobile and distributed computing.
A computer data network often has a changeable, ad hoc topology, greatly varying usage patterns and a lack of control over bandwidth allocation. Because of this, a controller trained on prior data cannot be guaranteed to perform well under the conditions, and must be rebuilt when the topology is altered.

Current approaches to this problem have been partially successful, but are liable to breakdown when under heavy load.

A routing algorithm that attempts to solve some of the problems faced in dynamic, unreliable, and congested networks is suggested. Such an algorithm aims to have the following properties:

- It should be able to "bootstrap" itself without any need for operator intervention.
- It should be able to "route around" failed links or nodes.
- It should be able to find optimal or close to optimal routes even when the network is heavily congested.

3.1 Problem Description

Routing algorithms are methods for finding the best way to get from a host A to another host B. This may be via a large number of other machines or it may be in the next room.

- On a small, simple network the problem is almost trivial, statically allocating routes and defining them by hand.
- When dealing with a huge internetwork such as the Internet this is not possible. A heavily interconnected network such as the Internet has many routes from one host to another, and these routes span many different types of link with different bandwidth and latency characteristics.
- Calculating the best route through a complex system is computationally intractable and impossible to do by hand.
- A routing algorithm first seeks to deliver a packet successfully, and if possible deliver it by the quickest route available.
- If many packets are routed through the same router a bottleneck occurs, and the whole network slows down, a good routing algorithm will try to route around a bottleneck router to minimize the effect of network congestion.

3.2 Proposed solution

To remove these drawbacks we will develop a new intelligent routing algorithm provides *stability* in complex topology. One approach to solve complex topology is the use of neural networks as a non-linear function approximator in an online reinforcement learning task.

A Q-routing Algorithm, a network algorithm based on Q-learning, a method from the emerging field of reinforcement learning. Q-routing learns to route packets in an adaptive manner allowing slow spots in the network to be routed around, and has performed well in simulation. Neural networks have been used with some success to perform Q-learning, and would seem to be a possible method to allow Q-routing to scale well beyond its initial table-based implementation. Neural networks have often been used in reinforcement learning. The use of neural networks in this situation is sometimes called "neuro-dynamic programming" because of the use of neural network and dynamic programming techniques together. There have been notable successes with these methods using temporal difference learning and a neural network which approximates the value function. Success with neural network function approximation and temporal difference learning was also shown with a control problem similar to network routing by Crites and Barto [9]. Crites and Barto used a neural network function approximator in conjunction with temporal difference learning to control a system of elevators (lifts) and which was able to outperform all known heuristic elevator control algorithms.

The Algorithm will be developed that aims to be stable as possible as under high loads while performing well in less extreme situations. The algorithm will be analyzed and compared with the existing algorithms theoretically that whether it complete its requirements or not.

Q-routing and Kohonen's network will be combined in this algorithm. Kohonen's network will be used as function approximator. Q-value table will be updated using Kohonen's network updating rule.

Proposed Algorithm will have five modules.

- A Packet Generator module
- Routers modules (4- modules)

▪ A packet generator will generate the packets to different source routers using "Poison Distribution algorithm".
▪ When packet generated by packet generator will given to the source node, then Routing module of that source router will activated to find out the next node to hand over the packet.
▪ Routing module will be developed by using Kohonen's network algorithm as function approximator in reinforcement learning algorithm.
▪ Each router will have its own Q-weight table. When packet will be routed through different routers, Q-weight table values of each router will be updated according to weight updating rule given by new algorithm.

CHAPTER 4

Design

4. Design

In this chapter we will discuss the design, algorithm formulation and algorithm in detail.

4.1 Algorithm Formulation

In the formulation of algorithm we have few things to consider in doing so. They are described in the following.

4.1.1 Network Load Level

It is defined as the average number of packets introduced in the network per unit time. For simulation purposes, time is to be interpreted as simulation time (discrete time steps synchronized for all nodes in the network). Three ranges of network load levels are identified: low load, medium load, and high load. At low loads, exploration is very low and the amount of information per packet hop significantly affects the rate of learning. At medium loads, the exploration is directly related to the number of packets in the network. Medium load level represents the average load levels in a realistic communication network. Although the amount of exploration is high at high loads, there are a large number of packets in the network, and it is actually more difficult to learn an effective routing policy. When a node's buffer gets filled up, additional incoming packets are dropped leading to loss of information, which is called congestion. In this thesis, however, infinite packet buffers are used.

4.1.2 Traffic Pattern

It is defined as the probability distribution p(s; d) of source node s sending a packet to node d. This distribution is normalized such that:

$$\sum_{x \in V} p(s; x) = 1 \text{ for all } s \in V$$

Where V is the set of all nodes in the network. The value of p(s; s) is set to 0 for all s ϵ V. A uniform traffic pattern is one in which the probability p(s; d) is 1/n-1 where n is the number of nodes in the network.

4.1.3 Network Topology

It is made up of the nodes and links in the network. The topology changes when a link goes down or comes up. The following Q-value updates are used to model the going down of the link between nodes x and y:

$$Q_x(y, d) = \text{infinite for all } d \in V; \quad (4.1)$$

And

$$Q_y(x, d) = \text{infinite for all } d \in V; \quad (4.2)$$

4.1.4 A Routing Policy

It is characterized by the Q tables in the entire network. Changes in these Q-values by exploration leads to changes in the routing policy of the network. The algorithm is said to have converged to a routing policy when changes in Q-values are too small to affect any routing decisions. An indirect way of testing whether the routing policy has converged is to examine the average packet delivery time or the number of packets in the network as routing takes place. When average packet delivery time or number of nodes in the network stabilize or converge to a value and stay there for a long time, we can say that the routing policy has converged.

4.1.5 Performance Measure

The performances of an adaptive routing algorithm can be measured in two ways:

(a) **Speed of Adaptation** is the time it takes for the algorithm to converge to an effective routing policy starting from a random policy. It depends mainly on the amount of exploration taking place in the network.

(b) **Quality of Adaptation** is the quality of the final routing policy. This is again measured in terms of the average packet delivery time and the number of packets in the network. Quality of adaptation depends mainly on how accurate the updated QWeight-value is as compared to the old QWeight- value. Hence quality of exploration affects the quality of adaptation.

4.1.6 Average Packet Delivery Time

The main performance metric for routing algorithms is based on the delivery time of packets, which is defined as the (simulation) time interval between the introduction of a packet in the network at its source node and its removal from the network when it has reached its destination node. The average packet delivery time, computed at regular intervals, is the average over all the packets arriving at their destinations during the interval. This measure is used to monitor the network performance while learning is taking place. Average packet delivery time after learning has settled measures the quality of the final routing policy.

4.1.7 Number of Packets in the Network

A related performance metric for routing algorithms is the number of packets in the network also referred to as the amount of traffic in the network. An effective routing policy tries to keep the traffic level as low as possible. A fixed number of packets are introduced per time step at a given load level. Packets are removed from the network in two possible ways; either they reach their destination or the packets are dropped on the way due to congestion. Let $n_g(t)$, $n_r(t)$ and $n_d(t)$ denote the total number of packets generated, received and dropped at time t, respectively. Then the number of packets in the network $n_p(T)$ at the current time T is given by:

T-1247

$$n_p(T)=\sum_{t=0}^{T} (n_g(t) - n_r(t) - n_d(t)) \qquad (4.3)$$

Where time $t = 0$ denotes the beginning of the simulation. In this thesis, infinite packet buffers are used therefore no packets are dropped (i.e. $n_d(t) = 0.\ 8t$).

4.1.8 Network Topology used

The network topology used for simulation is 6X6 irregular grid shown in figure 4.1 due to Boyan and Littman (1994). In this network, there are two possible ways of routing packets between the left cluster (nodes 1 through 18) and the right cluster (nodes 19 through 36): the route including nodes 12 and 25 (R1) and the route including nodes 18 and 19 (R2). For every pair of source and destination nodes in different clusters, either of the two routes, R1 or R2 can be chosen. Convergence to effective routing policies, starting from either random or shortest path policies.

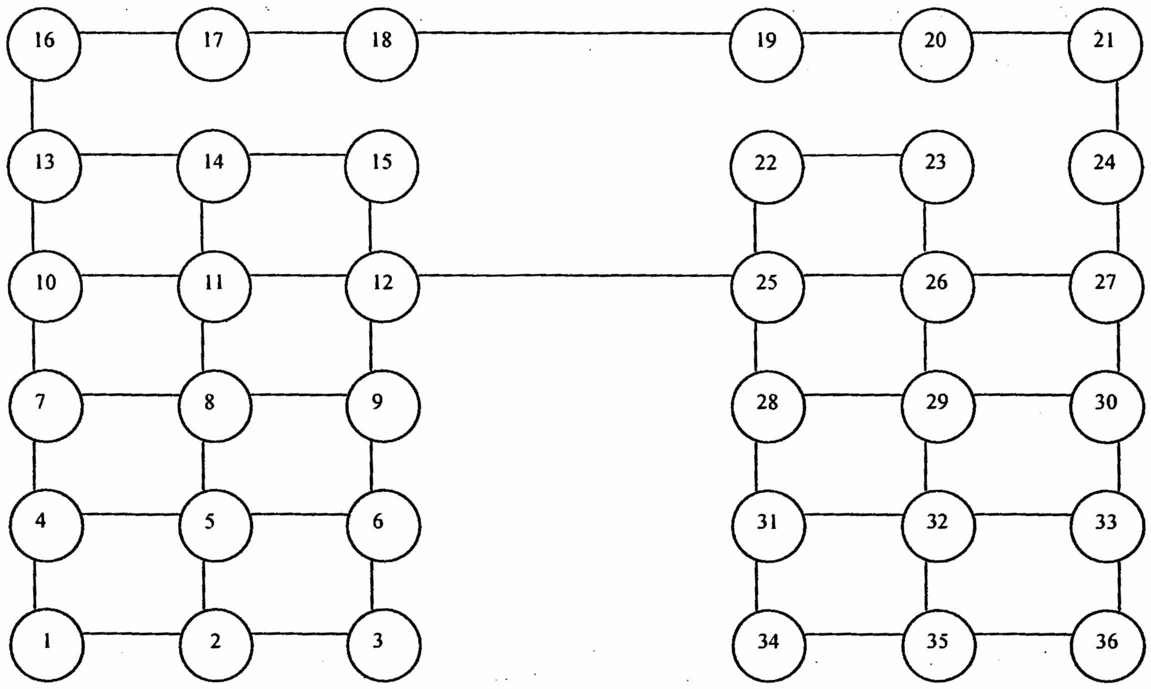


Figure 4.1: The 6X6 Irregular Grid (adapted from Boyan and Littman (1994)). The left cluster comprises of nodes 1 through 18 and the right cluster of nodes 19 through 36. The two alternative routes for traffic between clusters are the route including the link between nodes 12 and 25 (route R1) and the route involving the link between nodes 18 and 19 (route R2). R1 becomes a bottleneck with increasing loads and the adaptive routing algorithm needs to make use of R2.

4.1.9 Neural Network (Kohonen's Network) Architecture used

A Kohonen's Network consists of two layers, an Input Layer and a Kohonen's Layer, as shown in figure 4.2. The input layer receives inputs and performs no processing. The Kohonen's Layer is responsible for working out the 'winning' neuron using weights calculated during training.

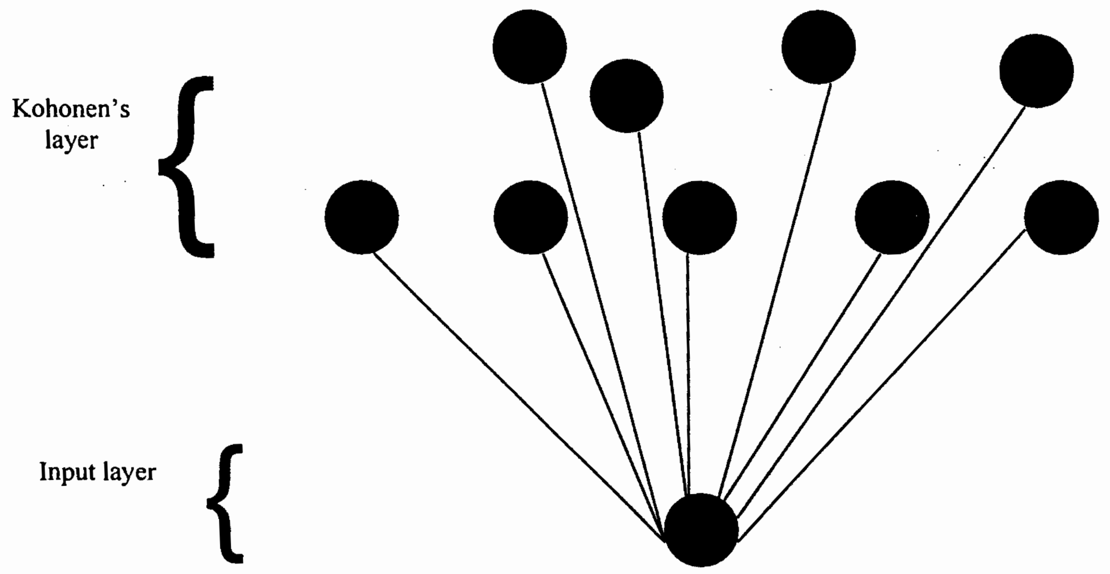


Figure 4.2: A schematic Diagram of Kohonen's Network. It shows the nodes in the input layer connected to all of the nodes in the output layer. Each connecting line represents a weight.

Learning in Kohonen's Networks

The learning process is as roughly as follows:

- initialize the weights for each output unit
- loop until weight changes are negligible
    - for each input pattern
        - present the input pattern
        - find the winning output unit
        - find all units in the neighborhood of the winner
        - update the weight vectors for all those units
    - reduce the size of neighborhoods if required

The winning output unit is simply the unit with the weight vector that has the smallest Euclidean distance to the input pattern. The neighborhood of a unit is defined as all units within some distance of that unit on the map (not in weight space). In the demonstration below all the neighborhoods are square. If the size of the neighborhood is 1 then all units no more than 1 either horizontally or vertically from any unit fall within its neighborhood. The weights of every unit in the neighborhood of the winning unit (including the winning unit itself) are updated using:

$$w_{ni} = w_{ni} + phi * r_{ij} * (N_n - w_{ni}) \qquad (4.4)$$

where

$$r_{ij} = e^{(-dist(i,j)^2}) / (2*theta)$$

phi = Learning rate parameter

theta = Learning Factor

dist(i, j) = Guassian density function.

This will move each unit in the neighborhood closer to the input pattern. As time progresses the learning rate and the neighborhood size are reduced. If the parameters are well chosen the final network should capture the natural clusters in the input data.

4.2 The Algorithm

Main steps of the algorithm developed are:

Q-routing and Kohonen's network are combined in this algorithm. Kohonen's network is used as function approximator. Q-value table is updated using Kohonen's network updating rule as in equation (4.4).

Main steps of the algorithm are:

1. Initial Q-values can be calculated as

   $$r_{ij} = e^{(-dist(i,j)^2}) / (2*theta)$$

   where

   $$dist(i,j) = \exp(-d^2/phi) / \mathrm{sqrt}(2)$$

2. Take source node as starting node.
3. Find the next neighboring node having minimum Q-value.
4. Update the Q-value for that selected node by using the equation derived from equation (4.4) as

$$\Delta Q_x(y, d) = Q_x(y, d) + phi * r_{ij} * (Q_y(\hat{z}, d) - Q_x(y, d))$$

Where

$$Q_y(\hat{z}, d) = \min_{z \epsilon N(y)} Q_y(z, d)$$

5. Decrease *theta* , *phi* and recalculate r value for the selected node only.
6. goto step 3.

4.3 Data Flow Diagrams

In this section we explain different levels of data flow diagram.

4.3.1 Level 0 Data Flow Diagram

Level-0 data flow diagram is shown in figure 4.3.

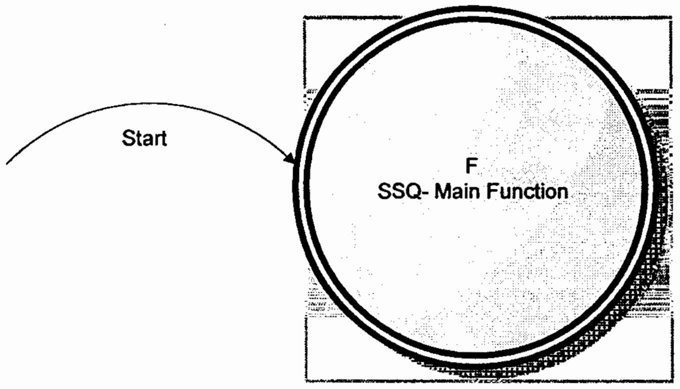


Figure- 4.3 Level 0 DFD

4.3.2 Level 1 Data Flow Diagram

Level-1 data flow diagram is In this level of diagram we have F1 input process, F2 Find path process and F3 Update weight process shown in figure 4.4.

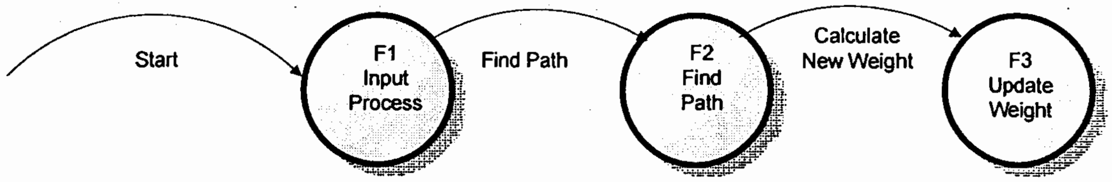


Figure- 4.4 Level 1 DFD

4.3.3 Level 2 Data Flow Diagram

In this level of data flow diagram we have explained F1 find path process shown in figure 4.5.

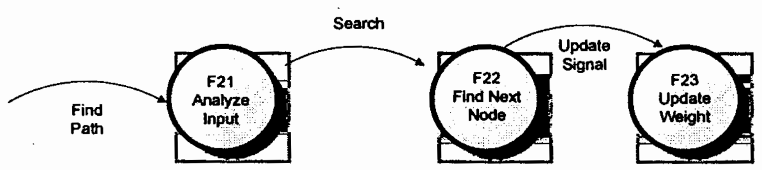


Figure- 4.5 Level 2 DFD

4.4 Flow Chart

Main algorithm flow chart is shown in figure 4.6.

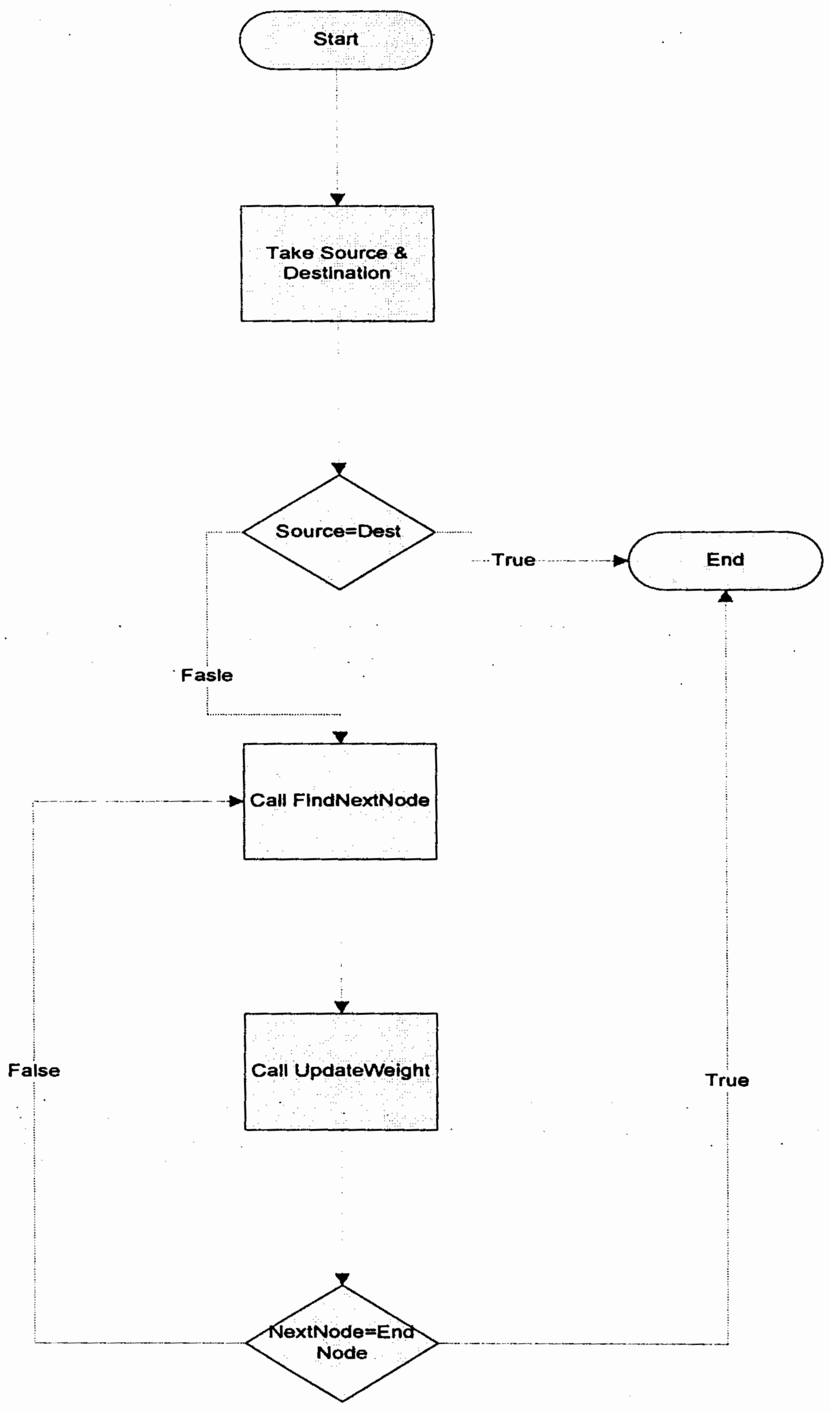


Figure- 4.6 Program Flow Chart

CHAPTER 5

Implementation

5. Implementation

In this chapter we discuss how we implement our algorithm. In section 5.1 we have explained division of functional units and coding details are discussed in section 5.2.

5.1 Division of Functional Units

Considering the nature of the whole simulation it is divided into five applications. Which are described in the following?

5.1.1 Packet Generator

This application generates packets to route between different nodes (Routers). Packets are generated randomly. Poisson distribution is used to generate packets randomly.

Table 5.1

Distribution	Functional Form	Mean	Standard Deviation
Poisson	$f_p(x) = \frac{e^{-a} a^{x}}{x!}$	a	$\sqrt{a}$

Poisson distribution generates packets according to the natural phenomena such that Poisson distribution is not an average distribution different numbers of packet are generated for each node.

In this function packets are generated after every 100 ms. Packet generator is connected to each routing server (Router). When a packet is generated a source node is analyzed and hand over to that source Router for routing to destination node.

5.1.2 Routers (Nodes)

There are 9- Routers in a network. Each router is directly connected with Packet Generator. Each Router has a unique ID such that Router 1 has an Id like ID-1 which uniquely identifies it while routing a packet. Router ID-0 is directly connected with

Router ID-1 and Router ID-2. Router ID-1 is connected with Router ID-1 and Router ID-3. While Router ID-3 is connected with Router ID-1 & 2 and so on. There is bidirectional communication between 2 nodes such that each node can send and receive data from other node.

When a packet is generated by Packet generator it is handover to a source router. Source router calls it "FIND_NEXT_NODE()" function to find a next node to route a packet to its destination. When a next node is found then packet is handover to that node and a signal is generated to update the Q- weight value from this node to new node. When a node received update signal it updates its Q-Weight value table entry for those nodes received in signal. Also the values of 'Theta', 'Phi' and 'r' are recalculated after receiving each updated signal.

5.2 Coding Details

5.2.1 Packet Generator

In this application we have two functionalities. One is to listen for request from each Router and second to generate packets using Poisson distribution on request.

5.2.1.1 Listening for the Request

Packet generator starts listening in the beginning for the requests from the Routers.

5.2.1.2 Connection to Router

When a router request for connection any available socket is assigned to it to which a bidirectional communication can be made.

5.2.1.3 Packet Generate Request

The Poisson random distribution is implemented on the packet generator to generate packet randomly. When a "Start Generation" button is pressed then Poisson distribution code is running after every 100ms a packet is generated.

5.2.2 Routers

The real implementation of Algorithm is on each Router. When a request for routing comes the algorithm runs at that time. In following it is described how Router is carrying its functionality.

5.2.2.1 Listening for Request from other Routers

All Routers start listening for the request from other Routers.

5.2.2.2 Connection to Packet Generator

Router requests to the Packet Generator for connection on clicking button "Connect to PG".

5.2.2.3 Connection to Other Routers

When Router receive request for connection any available socket is assigned to it through which bidirectional communication can be done.

5.2.2.4 Algorithm Implementation

A separate class by the name of "Router" is made for the algorithm and all functions are implemented there, which are used with object of class on calling. The details of class are given below.

5.2.2.5 Call For FIND_NEXT_NODE

When "FIND_NEXT_NODE" function is called with two parameters startnode and endnode then algorithm start to find new next node to which packet will be handed over. When the next node is found then UPDATE_QW() function is called. A new header is added to the packet and sends to the new Router found.

5.2.2.6 Call For UPDATE_QW

This Function is called by the FIND_NEXT_NODE function. When this function is called Q-Weight value from the previous node and the new found node is updated. Also the value for Theta and 'Phi' are updated. And all new values are broadcast to all routers.

CHAPTER 6

Testing

6. Testing

Testing of a program is used to check whether it produces the same results as are expected from it and how does it handles in the situation where an exception of error occurs. Testing has different types defined in sections below.

6.1 System testing

Which is to test the system as a whole to validate that it meets its specification and the objectives of its users? System testing focuses on testing the system as an entity. Generally, it is the responsibility of a group which is separate from the system development team.

It is generally good practice for system testing to be an independent activity as the testers are not themselves stakeholders in the system development. If developers are involved at this stage, they may be reluctant to design tests which reveal problems in the developed system as this is an implicit criticism of the quality of their work.

6.2 Development testing

- Hardware and software components should be tested as they are developed and as sub-systems are created. These testing activities include:
    - Unit testing.
    - Module testing
    - Sub-system testing
- However, these tests cannot cover:
    - Interactions between components or sub-systems where the interaction causes the system to behave in an unexpected way
    - The **emergent** properties of the system

As part of the development process, each component which has been developed should be tested either by its developer or by a separate testing group. The objective of this testing process is to find defects in that component. These defects should then be removed before the component is delivered for integration.

However, these tests can only be based on the component specification (if it exists) along with knowledge about the structure of the component. There may be component errors which are not discovered because these relate to the interaction of the component with other components in the system.

The emergent properties of a system are those properties which apply to the system as a whole rather than to particular components in the system. While some assessment can be made, e.g. of individual component reliability, unit and module testing be used to assess the overall reliability or performance of the whole system.

6.3 Integration testing

The major activity in the integration process is integration testing where the developer of the system carries out a series of tests as the system is put together from its components.

Integration testing should be concerned with tests which cannot be executed on individual system components or sub-systems. Interface testing is concerned with designing tests which will validate the interactions between components and property testing is concerned with testing the emergent system properties such as reliability, performance etc. As these do not emerge until the system exists as a single entity, it is clearly impossible to test them earlier in the process.

6.3.1. Integration test planning

A separate group should always be responsible for test planning for two reasons:

a. It means that test planning can be carried out at the same time as system development

b. It removes a potential conflict of interest from the development team - is their responsibility to develop software or to test (and potentially find faults with) that software. Developers may, consciously or unconsciously, design tests which they know avoid problems in the system.

For large, complex systems, integration test planning may involve hardware and software engineers and human factors specialists.

6.3.2 Test planning activities

Wherever possible (and this is really not easy) the integration test planning team should identify individual system increments which can be tested and should design tests for these increments. These decisions may be made using the delivery schedules for the different sub-systems (it makes sense to stagger delivery - getting everything on the same day is an integration nightmare) but schedule changes may mean that increments are not available when required.

Testing tools such as tools to compare test outputs, tools to automatically run tests from files of test data, simulators for hardware which is not available may have to be developed before system testing is possible. The development of these tools goes on in parallel with systems development and often represents a significant fraction of the overall system development costs

6.4 Stress testing

Stress testing is particularly important for large, multi-user systems where the load on the system varies dramatically from time to time. In essence, you estimate the maximum load that the system is likely to have to handle then test it with more than that load. What should happen is graceful failure where the level of service offered to all users is reduced. What often happens is catastrophic failure where the system moves from working reasonably for all users to a complete loss of service.

Building up the load on the system is not just a test of system performance. Because there is so much stress on the system, defects which can be corrected automatically in other situations come to light during stress testing. For example, say a screen is not properly updated but the normal use calls for this screen to be replaced quite quickly in normal use. The error may never be discovered. Stress testing slows the system down and may reveal this kind of defect.

This is not too important but stress testing can also reveal defects which are caused by built-in timing assumptions in real-time systems

6.5 Acceptance testing

Acceptance testing may take place after a system has been installed but often it takes place at the developer's premises using customer supplied data. The customer observes the system tests to check if the system meets the specified requirements.

It is important to understand that the decision on whether or not to accept a system does not necessarily depend on the system meeting every requirement and successfully executing every test supplied by the customer. The customer needs the system (presumably) so they may be willing to accept an imperfect system for installation. The problems identified are noted and the contractor may have to agree to fix these problems in the first new release after the system has been delivered.

There may also be disagreement between the customer and the contractor at this stage about what requirements actually mean. The customer may have one interpretation of the requirements and the contractor a different interpretation. Therefore, when there is a problem with an acceptance test, some negotiation is necessary to decide whether the customer or the developer has the right interpretation. Often, the result will be that some system changes have to be made and the customer has to pay for some or all of the costs of these changes

6.6 Performance testing

It may be possible to use data for stress testing for performance testing as critical performance problems are most likely to occur when the system is heavily loaded.

The major problem with performance testing is that there are rarely explicit performance requirements which are specified in a measurable way. Furthermore, there may be serious conflicts between for example security and performance requirements and the only way to fix the performance problems might be to weaken system security.

The perceived performance of a system is important (if it is an interactive system) whereby the performance is as much to do with expectations as it is with actual figures. If users use a system with a specific performance level, they will expect a new system to at least match that level, even if it offers much greater functionality. This has to be taken into account when setting performance criteria,

6.7 Reliability testing

The problem with reliability is that it is not an absolute but depends on the context of use of the system. Two different patterns of system use can result in different perceived system reliability.

For this reason, it is very important to get the operational profile right such that the predicted pattern of inputs which will be presented to the system. This is possible for some classes of system (where reliability testing is very mature) such as telephone switches where the actual usage of an existing system can be logged and used as the basis of an operational profile.

It is MUCH harder to predict an operational profile when a completely new system or process is introduced - no-one really knows how users will adapt to the change and what inputs will be generated.

Reliability testing must take into account the seriousness of system errors. For example, an error in an air traffic control system where a display was pink rather than red is much less serious than an error in the same system where the height of the aircraft was wrongly computed.

6.8 Security testing

This is an unusual form of testing because it can't really be planned in the same way. While it is possible to pre-conceive some simple security tests, effective security testing can only really be interactive and, arguably, can only be carried out once the system is in use.

Interactive testing is necessary because security problems may not have a single cause. A user may detect a potential weakness in the system and then exploit this in some other way to gain access to protected parts of the system. It is almost impossible to anticipate this in advance

The argument that security testing cannot be effective until the system is in use comes from the fact that many security problems are due to the way in which a system is used such as insecure passwords, use of over-general permission vectors, etc. These can't really be tested in a pre-production version of the system.

6.9 Testing for SSQ-Routing Simulation system

6.9.1 Prerequisites

Following are the prerequisites for SSQ-Routing Simulation:

- All applications such Routers, and Packet Generator should be running.
- Packet generator generates packets using Poisson distribution.
- SSQ-Routing is implemented on each Router so it is considered mostly.

6.9.2 TC –SSQ-Routing- PG – Packet Generation

Test Case ID	**TC- SSQ-Routing – PG – Packet Generation**
Functional Area	**PG**
Test Name	***Packet Generation***
Description (Purpose)	Packet is generated after every 5msec.
Prerequisite	PG should be connected to every Router.
Input	Mean value
Actions to perform (Procedure)	Send generated packet to the source Router.
Expected Result(s)	**Status**
Generate Packet after every 5msec using Poisson distribution and send it to the source node.	Pass

6.9.3 TC – SSQ-Routing –Router – Find_next_node

Test Case ID	**TC – SSQ-Routing –Router Find_next_node**
Functional Area	Router
Test Name	***Find_next_node***
Description (Purpose)	Find shortest distance next node to route packet.
Prerequisite	Find_next_node() function should be active.
Input	Source and destination node ID should be provided.
Actions to perform (Procedure)	Find next node and route packet to it.

Expected Result(s)	**Status**
Find next node and send packet to that node. And invoke update_Qweight() Function.	Pass

6.9.4 TC - SSQ_Routing – Router– Update_Qweight

Test Case ID	**TC - SSQ_Routing – Router Update_Qweight**	
Functional Area	Router	
Test Name	***Update_Qweight***	
Description (Purpose)	Update the QWeight value in the table for the selected node.	
Prerequisite	Update-Qweight () function should be active.	
Input	Node ID should be provided.	
Actions to perform (Procedure)	Update Qweight table Entry.	
Expected Result(s)		**Status**
Update the Qweight value for the selected Router and broadcast updated value to all Routers.		Pass

6.9.5 TC - SSQ_Routing –PG & Router Integration

Test Case ID	TC - SSQ_Routing –PG & Router Integration
Functional Area	PG & Router
Test Name	PG & Router Integration
Description (Purpose)	Router sends request to PG (Packet generator) for Connection to Router.
Prerequisite	Packet generator should be in listen state.
Input	PG's Port address.
Actions to perform (Procedure)	PG accepts the Router Request.

Expected Result(s)	**Status**
Router connected to the PG.	Pass

6.9.6 TC - SSQ_Routing –PG & Router Integration

Test Case ID	**TC - SSQ_Routing –PG & Router Integration**
Functional Area	PG & Router
Test Name	PG & Router Integration
Description (Purpose)	PG generates packet and send to the destination node (Router).
Prerequisite	Router should be connected to PG.
Input	Router's Port address.
Actions to perform (Procedure)	Packet reaches to the source node.

Expected Result(s)	**Status**
Packet reached at source node.	Pass

6.9.7 TC - SSQ_Routing –Router ID-0 & Router ID-1 Integration

Test Case ID	**TC - SSQ_Routing –Router ID-1 Requests Router ID-0 for connection.**	
Functional Area	Router ID-0 & Router ID-1	
Test Name	Router ID-1 requests Router ID-0 for connection.	
Description (Purpose)	Router ID-1 requests for connection to Router ID-0.	
Prerequisite	Router ID-0 should be in listen state.	
Input	Router ID-1's Port address.	
Actions to perform (Procedure)	Router ID-1 connects to Router ID-0.	
Expected Result(s)		**Status**
Router ID-1 connected to Router ID-0.		Pass

6.9.8 TC - SSQ_Routing –Router ID-0 & Router ID-1 Integration

Test Case ID	**TC - SSQ_Routing –Source node (Router ID-0) sends packet to destination node (Router ID-1).**
Functional Area	Router ID-0 & Router ID-1
Test Name	Router ID-0 sends packet to Router ID-1.
Description (Purpose)	Source node (Router ID-0) sends received packet from PG to destination node (Router ID-1).
Prerequisite	Router ID-0 & Router ID-1 should be connected.
Input	Router ID-1's Port address.
Actions to perform (Procedure)	Packet sends from source to destination node.
Expected Result(s)	**Status**
Packet received at destination node.	Pass

CHAPTER 7

Results and Discussion

7. Results and Discussions

In this chapter different experiments are performed to give an accurate measurement of the performance of the neural network based algorithm. All tests were carried out on the network topology given in Figure 4.1.

In this chapter, possible future directions of this thesis are also discussed.

7.1 Experimental Setup

The Q-tables were initialized with small random Q-values, except for the base cases. Learning rate for forward exploration η_f value for *'theta'*, and *'phi'* are summarized in table7.1. Performance of the algorithms was found to be the best with these parameters.

Table 7.1- Parameter values for the algorithm

Algorithms	η_f	*Theta*	*Phi*
Q-Routing	0.85	-	-
SSQ-Routing	-	0.5	0.5

In Table 7.1 Parameters for the Algorithms are shown. In Q-Routing and SSQ-routing where the learning rates vale for *'theta'* and *'Phi'* are constant.

7.2 Learning at Constant Loads

In the first set of experiments, the load level was maintained constant throughout the simulation. Results on two network topologies, namely the 36-node irregular 6*6 grid (figure 4.1) and a 128 node 7-D hypercube are presented. The speed and quality of adaptation at three load levels, low, medium and high, were compared. The typical load level values for the two topologies are given in table 7.2.

The adaptive routing algorithms were found to have relatively similar performance and learning behaviors (in average packet delivery time) within a given range.

Table7.2- Typical load values for two topologies

Topology	# Nodes	Low Load	Medium Load	High Load
6 X 6 Irregular Grid	36	0 - 1.75	1.75 - 2.50	2.50 - more
7- D hypercube	128	0 - 5	5 - 8	8 - more

In Table 7.2 Load level ranges the number stands for number of packets Introduced in the network per time step. The learning behavior of adaptive routing algorithms remains roughly similar within a given load range (low, medium or high), but if the load changes from one range to another, the behavior can change quite dramatically. In real life communication networks, the load is usually in the medium range, and occasionally changes to low or high levels.

The performance and learning behavior is significantly different from one load level to another for some routing algorithms. For example, learning behavior of Q-Routing shows different behavior at medium and high load levels (figure 7.3 and 7.5).

Three representative load levels, one in each of the three ranges, were used in the experiments. For the grid topology, they were 1.25 pkts/step, 2.5 pkts/step and 3.5 pkts/step.

For 7-D hypercube topology, they were 3 pkts/step, 7 pkts/step and 10 pkts/step. The load level ranges depend on the topology, more specifically on the average branching factor and number of nodes in the network. The 36 node 6 * 6 grid and the 128 node 7-D hyper-cube have different topologies, hence their load ranges are also different. These ranges were decided after a number of experiments.

The learning behavior was observed in terms of average packet delivery time and number of packets in the network during learning. The packet delivery times of all packets reaching their destination in a window of 25 time steps were averaged. Similarly, the number of packets in the network was averaged for every successive window of 25 time steps. Results averaged over 50 simulation runs, each starting with random initializations of Q-values for both network topologies are reported in figures 7.1 through 7.12. Statistical significance is computed at 99% confidence.

At **low load levels**, the average packet delivery time for both the grid (figure 7.1) and the hypercube (figure 7.3) and the number of packets in the network for the grid (figure 7.2)

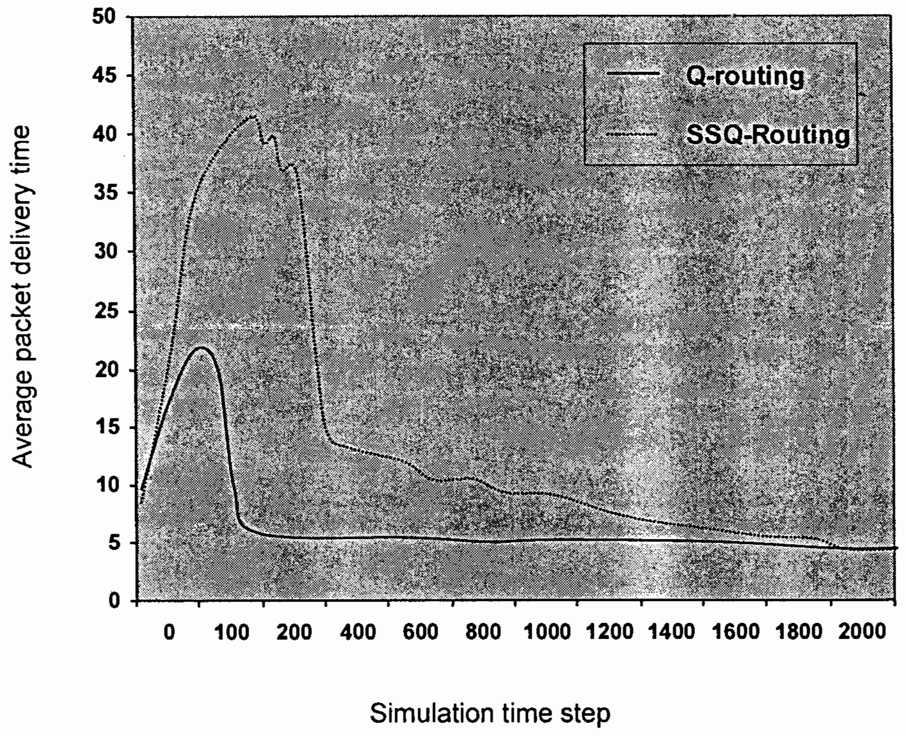


Figure 7.1: Average packet delivery time at a low load level for the grid topology: Difference between Q-Routing and SSQ-Routing is statistically significant between 300 to 800 time steps.

and the hypercube (figure 7.4) shows that there is not much gain in the speed of learning from Q-Routing to SSQ-Routing. Reason for this trend is that at low loads, what matters the most is the amount of exploration and the algorithm that allows more exploration per packet hop will learn faster. SSQ-Routing learns more than 3 times as fast as Q-Routing for both topologies.

At **medium load levels**, the average packet delivery times (figure 7.5 for the grid and figure 7.7 for the hypercube) and the number of packets in the network (figure 7.6 for the grid and figure 7.8 for the hypercube) show that both SSQ-Routing performs better than Q-Routing. This result is significant because it highlights the contribution of both the quality and quantity of exploration in learning. They contribute in two different ways to increase the performance of SSQ-Routing.

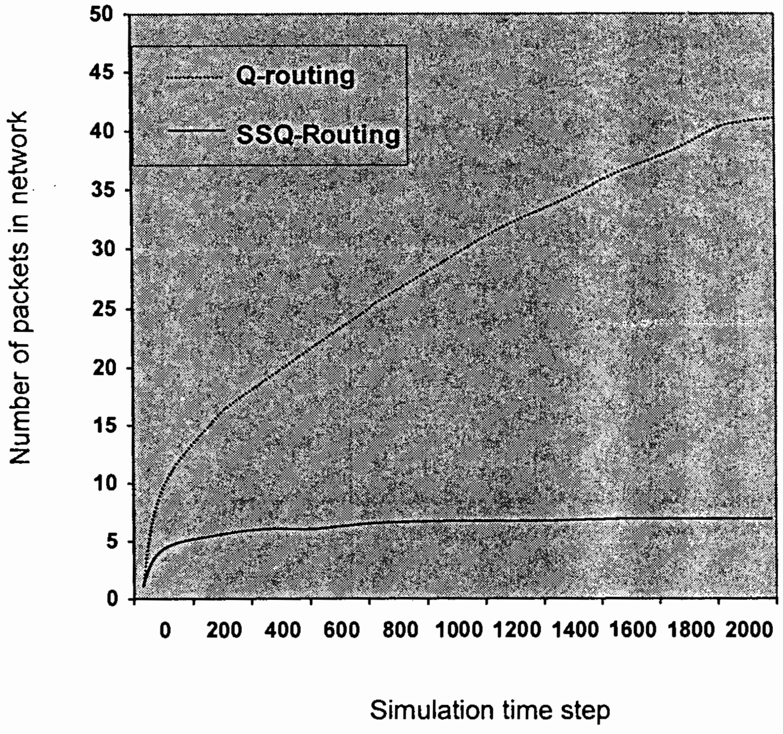


Figure 7.2: Number of packets in the network for the grid topology at a low load level: Difference between SSQ-Routing and Q-Routing is significant after 175 time step.

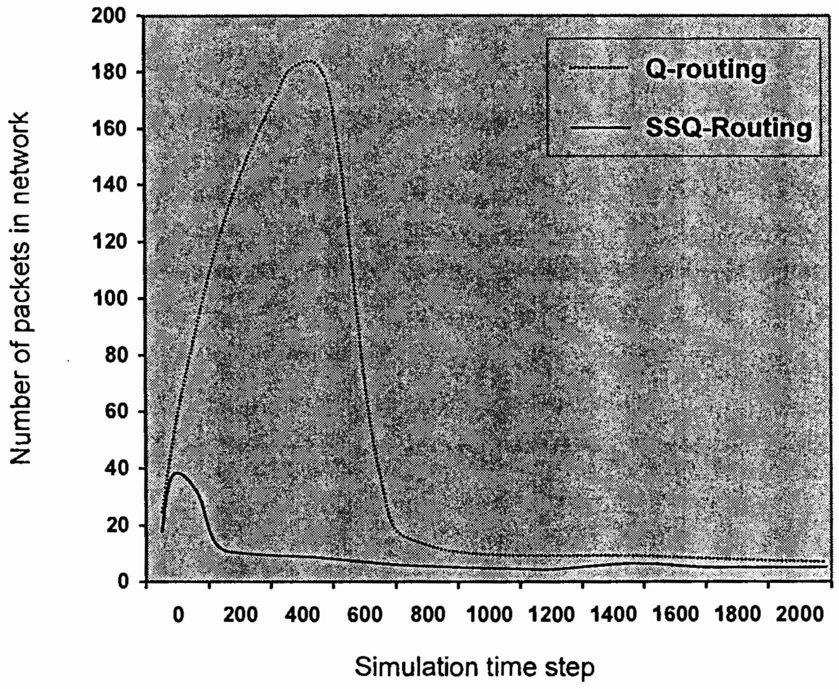


Figure 7.3: Average packet delivery time for 7-D hypercube at low load: Difference between SSQ-Routing and Q-Routing is significant between 175 to 800 time steps.

At **high load levels** the average packet delivery time (figure 7.9) and the number of packets in the network (figure 7.10) for the grid topology show that while Q-Routing converges to a qualitatively poor routing policy, and SSQ-Routing converge to qualitatively similar policies.

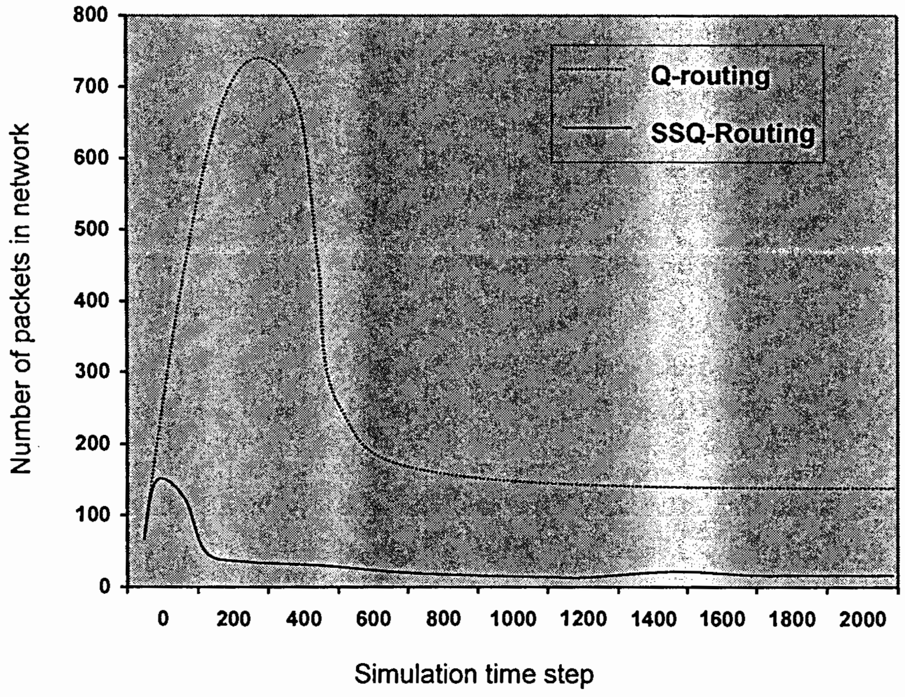


Figure 7.4: Number of packets for 7-D hypercube at low load: Difference between SSQ-Routing and Q-Routing is significant after 150 time step.

Q-Routing learns qualitatively similar routing policy as SSQ-Routing but learning is twice as fast in the later. The learning behavior for Q-Routing and SSQ-Routing is similar in both topologies at low and medium load levels. However, at high loads Q-Routing fails to converge to a policy qualitatively as good as SSQ-Routing for grid topology but this trend is not reflected in the hypercube topology. The reason being that the hypercube is a very symmetric network with not much alternatives to choose from while in grid topology, there could be multiple routing policies due to no symmetry in the topology. Hence, the performance improvements might vary with topology but in general

SSQ-Routing was found in all cases to improve significantly in speed of learning and in some topologies even in quality of the policy learned. Only grid topology is used in the next two sets of experiments, evaluating and comparing the performance of adaptive routing algorithms for adaptation to changes in traffic pattern and network topology.

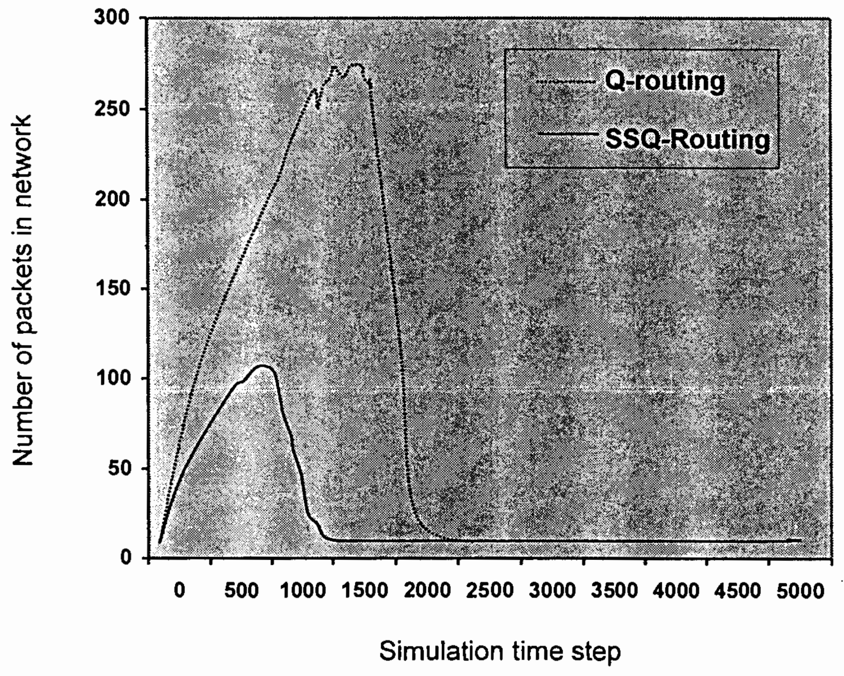


Figure 7.5: Average packet delivery time for the grid topology at a medium load level: Difference between SSQ-Routing and Q-Routing is significant between 300 to 1900 time steps.

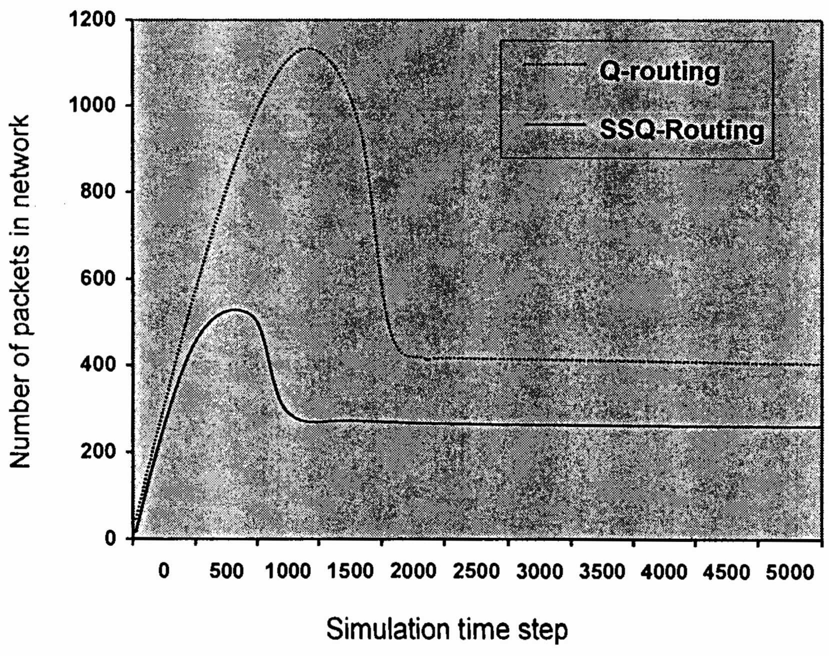


Figure 7.6: Number of packets in the network for the grid topology at a medium load level: Differences between both algorithms are significant after 500 time step and remain so for ever.

7.3 Adaptation to Varying Traffic Conditions

In the second set of experiments, Q-Routing, and SSQ-Routing, are compared with respect to their ability to adapt to variations in traffic patterns in the grid topology. Both algorithms were first allowed to learn an effective routing policy at a load level of 2 pkts/step and a uniform traffic pattern, where the probability that each node generates a packet to any other node is equal.

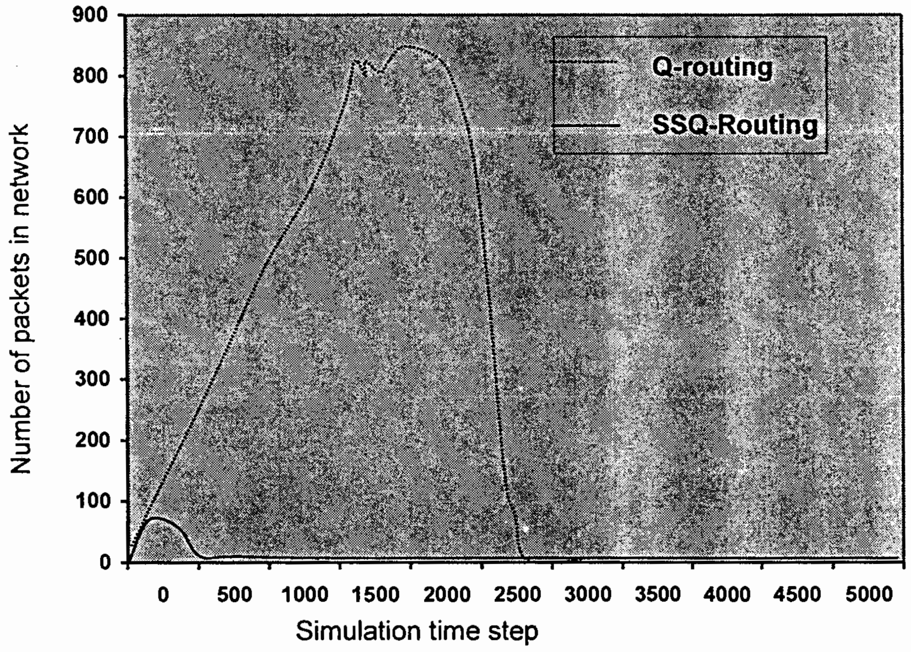


Figure 7.7: Average packet delivery time for 7-D hypercube at medium load: Difference between SSQ-Routing and Q-Routing is significant between 200 to 2750 time steps.

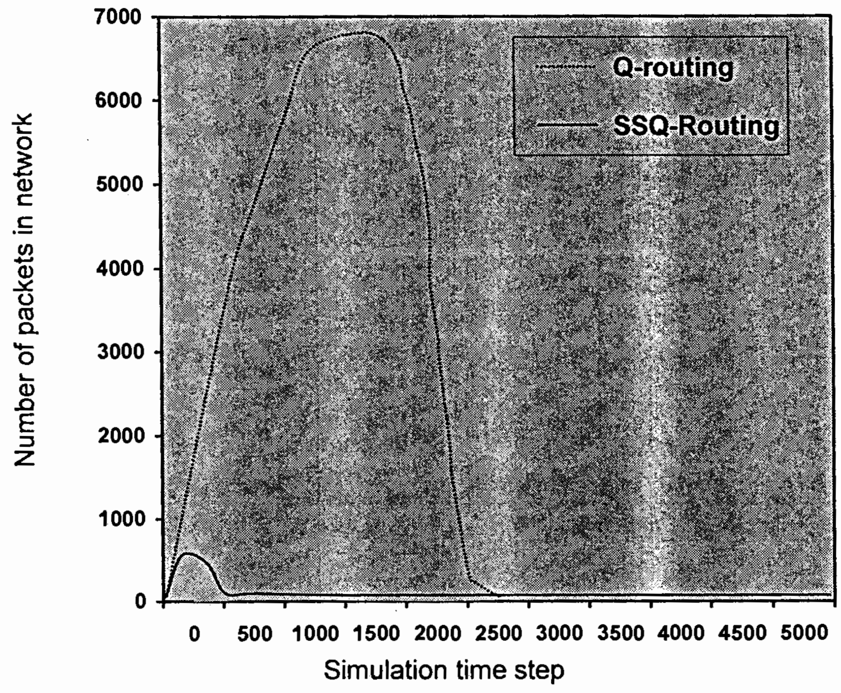


Figure 7.8: Number of packets for 7-D hypercube at medium load: Trends are very similar to the ones in figure 7.7. The significance limits are also same as in figure 7.7.

After convergence (in 2000 simulation steps), the traffic pattern was changed so that the probability of generating a packet destined to a node across the cluster becomes 10 times higher than within the cluster (figure 4.1). Only single run is shown in this case to depict the variability during adaptation process. Results from 50 test runs were used to compute statistical significance and are reported below.

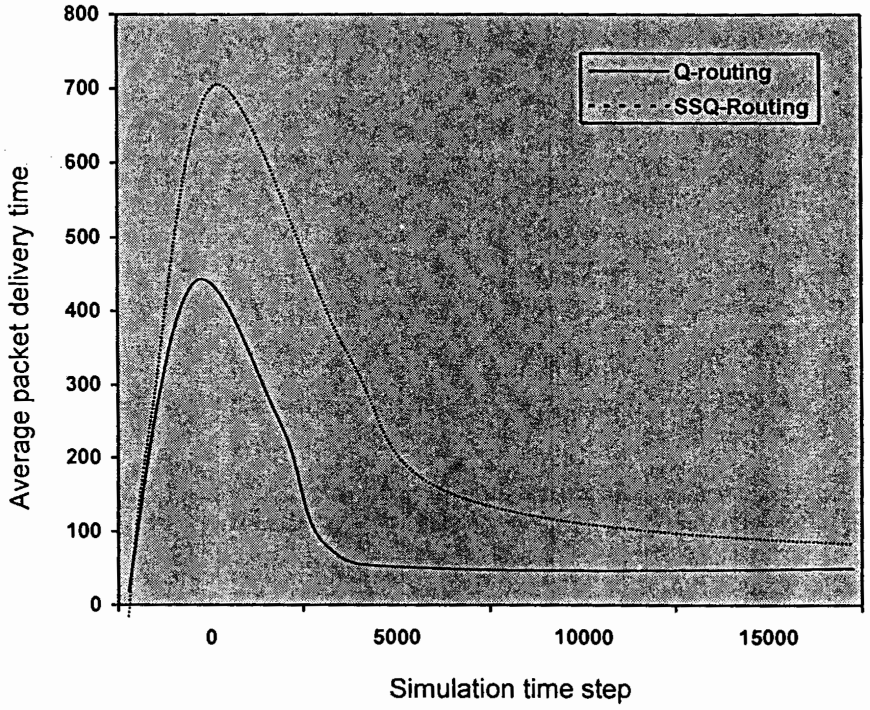


Figure 7.9: Average packet delivery time for grid topology at high load: Difference between SSQ-Routing and Q-Routing is significant between after 1000 time steps.

The average packet delivery time for each algorithm is shown in figure 7.13. Both Algorithms converge to an effective routing policy for the initial traffic condition, by time step 2000. As the traffic pattern changes at 2000, the old routing policy is no longer effective and both algorithms start updating their Q values to adapt to the change in the pattern. SSQ-Routing is faster than Q-Routing. There is a significant improvement over Q-Routing between 2600 to 2650 time steps. In fact, SSQ-Routing adapts to change in traffic pattern 50 time steps faster than Q-Routing on an average.

7.4 Adaptation to Changing Network Topology

The third set of experiments compared the routing algorithms' ability to adapt to the changes in network topology. A link was added between nodes 3 and 34 in the 6*6 grid

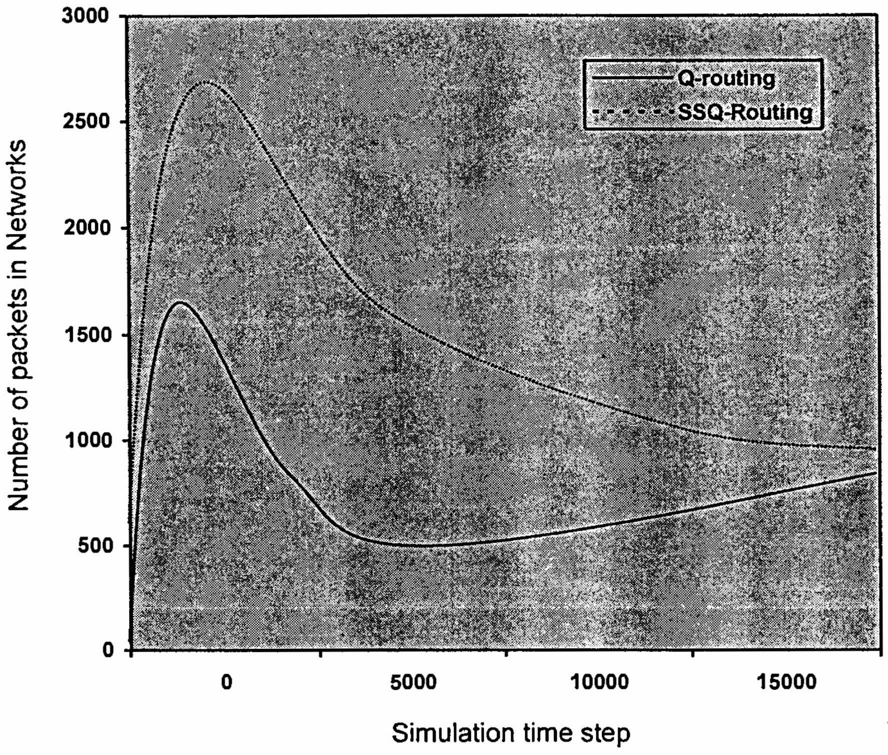


Figure 7.10: Number of packets in the network for grid topology at high load: Differences between SSQ-Routing and Q-Routing is significant during the learning phase from 1000 to 5500 time steps.

topology (figure 7.14) for these experiments. The routing algorithms were first allowed to learn an effective routing policy for the new network at a load level of 2.0 pkts/step until they converged (in 2000 time steps). At time step 2000, the link between node 12 and 25 was removed (figure 7.15). That is, the Q tables of node 12 and 25 were updated such that $Q_{12}(25,*)$ and $Q_{25}(12,*)$ were all set to Infinite Cost and the corresponding routing tables were also updated accordingly. The C values were not changed. Only single run is shown in figure 7.16 to depict the variations in adaptation process. However, statistical significance is computed over 50 runs and is given below.

Figure 7.16 shows the average packet delivery time between time steps 1800 and 3200 for both algorithms. As soon as the link between nodes 12 and 25 went down at time 2000, the average packet delivery time of both algorithms started increasing. They all tried to adapt to the change in network topology. SSQ-Routing learns better and faster Q-

Routing. The improvement from Q-Routing is significant with 99% confidence in the interval 2400 through 3200 (and beyond) time steps.

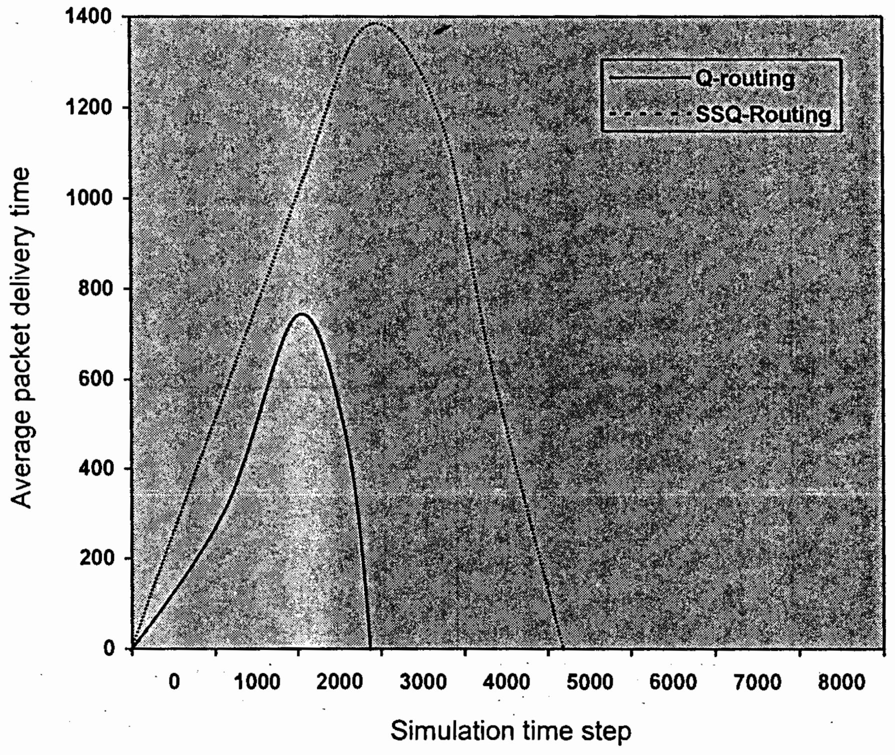


Figure 7.11: Average packet delivery time for 7-D hypercube at high load: All differences are significant between 2400 to 300 time steps. Q-Routing is significantly different from SSQ-Routing.

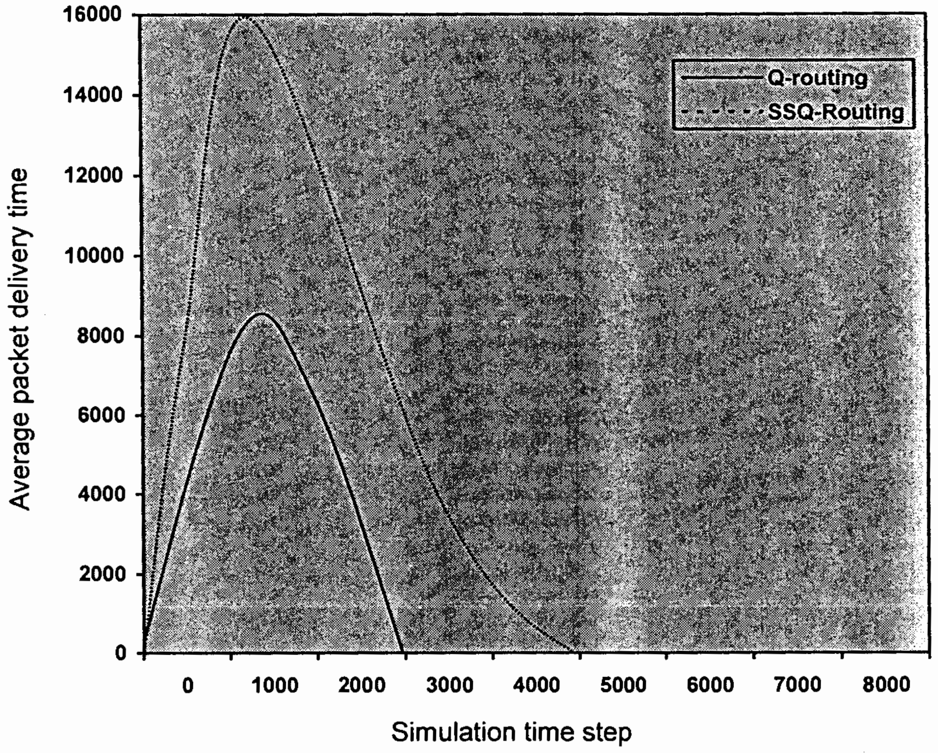


Figure 7.12: Number of packets for 7-D hypercube at high load: The trend is similar to that in figure 6.11 both for range of significant differences and quality of final policy learned.

The significant improvement in SSQ-Routing over Q-Routing is evident. Q-Routing fails to learn an effective routing policy for the changed topology even in 1000 steps (till step 3200), while SSQ-Routing has almost settled to an effective routing policy at that time.

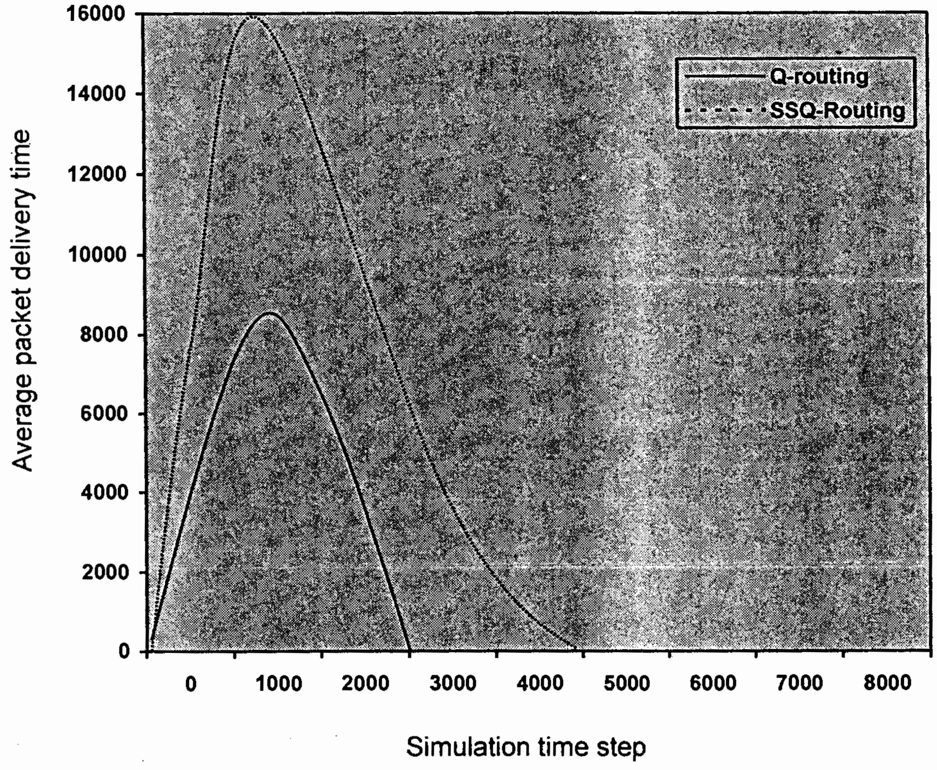


Figure 7.13: Adaptation to a change in traffic pattern. Single run is shown. Note the variations in the adaptation process. SSQ-Routing adapts around 50 time steps faster than Q-Routing. Difference between Q-Routing and SSQ-Routing is statistically significant between 2600-2650 time steps.

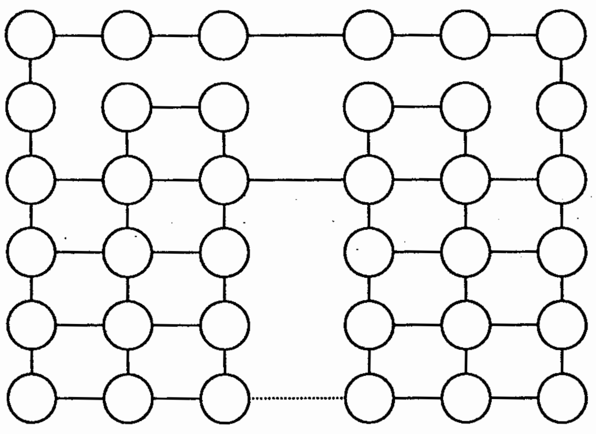


Figure 7.14: Grid topology before link 12 and 25 went down. This Topology is different from the one in figure 4.1. it has an additional link between 3 and 34 nodes.

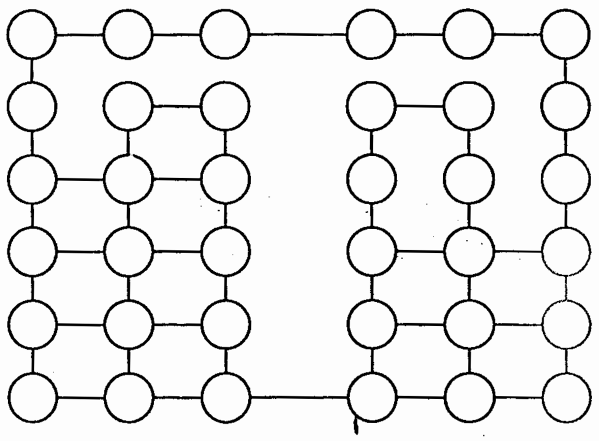


Figure 7.15: Grid topology after link between 12 and 25 has been removed, by fixing vectors $Q_{12}(25, *)$ and $Q_{25}(12,*)$ to infinite cost.

Boyan and Littman (1994) observed that Q-Routing fails to revert to the original routing policy once the network topology is restored. This problem persists in SSQ- Routing also. However, Bellman-Ford routing algorithm does not suffer from this problem and can adapt to changes in network topology very effectively.

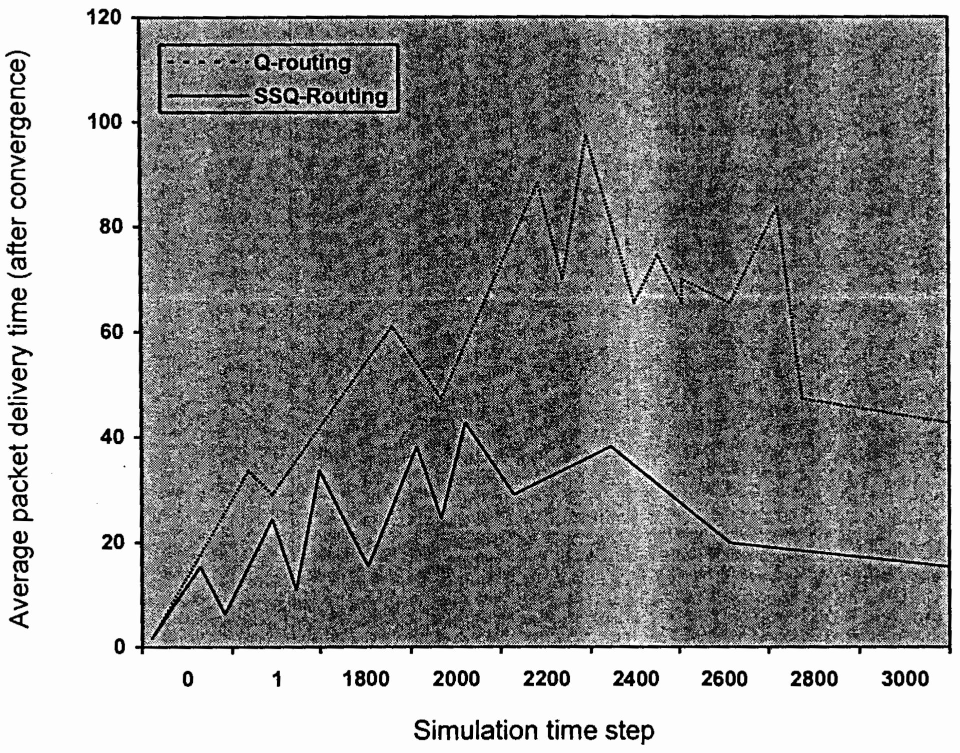


Figure 7.16: Adaptation to change a in the network topology. Q-Routing does not converge to an effective policy while SSQ-Routing converges to an effective policy very fast.

7.5 Load level Sustenance

In addition to evaluating how fast the routing algorithms learn, it is important to evaluate how good the final policy is. For this reason, the average packet delivery time, after steady state was reached, was measured for different load levels. These results indicate how much load the final routing policy can sustain.

Figure 7.17 shows the relative performance of both routing algorithms on the 6*6 gird topology (figure 4.1). The results were averaged over 20 simulations.

The non-adaptive shortest path routing is best at low load levels (0-1.5 pkts/step), but as the load increases to medium ranges (1.5-2.5 pkts/step), the nodes on popular routes start flooding and waiting time increases, degenerating the performance.

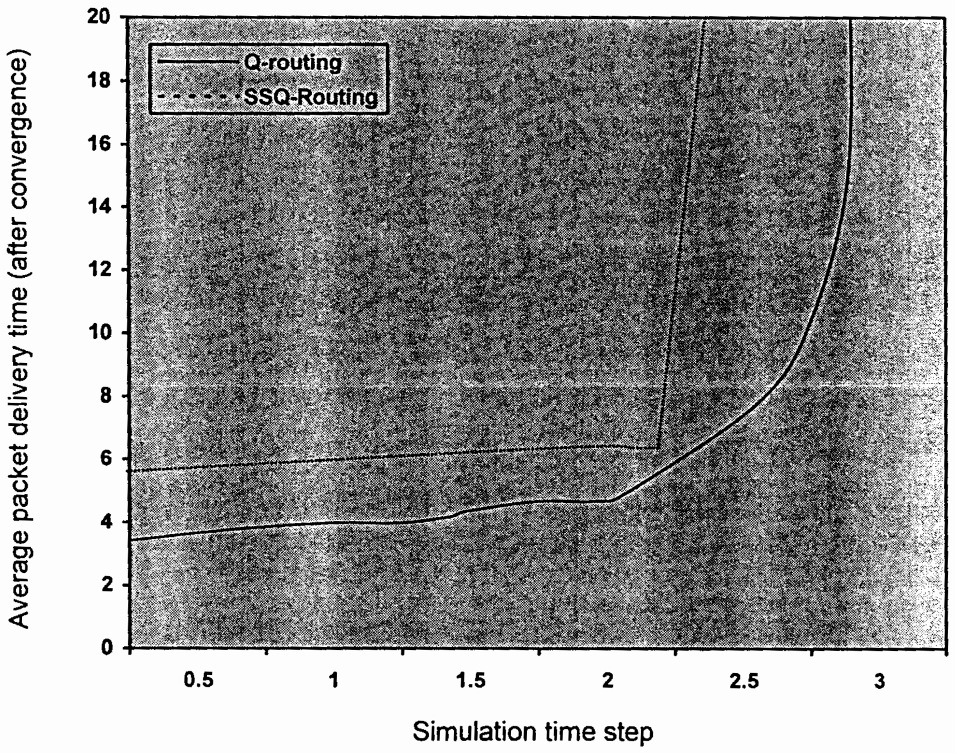


Figure 7.17: Average Packet Delivery time after convergence. The quality of the final routing policy is directly related to the amount of load level it can sustain. SSQ-Routing is much superior than Q-Routing in its ability to sustain high load levels. The difference between the two is statistically significant over all load levels with 99% confidence.

SSQ-Routing performs significantly better than Q-Routing. This is because the quantity of exploration is more important than quality at low load levels. At medium load levels, Q-Routing leads to flooding of packets in the network, and the average packet delivery time increases quickly. SS-Routing can sustain these load levels.

At high load levels (2.5 and higher pkts/step), the SSQ-Routing sustains up to 3 pkts/step before breaking down.

This result is a clear indication that both quality and quantity of exploration contribute to the final routing policy, quantity being more significant than quality. In the Q-learning framework, SSQ-Routing is the best adaptive routing algorithm currently known.

7.6 Conclusions and Future Work

SSQ-Routing with higher quantity of exploration and better quality of exploration than Q-Routing was developed in this work. It was evaluated and compared to Q-Routing.

The ability to learn an effective routing policy starting from a random policy, the ability to adapt to changes in traffic patterns, and the ability to adapt to changes in network topology. SSQ-Routing is an improvement over the conventional adaptive routing algorithms such as BF in a number of ways. First, SSQ-Routing tries to optimize more realistic criteria, the average packet delivery time, while the conventional adaptive routing algorithms try to learn the shortest path. Moreover, the amount of exploration overhead in CDRQ-Routing is significantly smaller than that in BF which exchanges complete cost tables between neighboring nodes. SSQ-Routing strikes a balance between the amounts of overhead incurred and speed of adaptation whether it is from random policy at fixed loads or to change in traffic pattern or change in network topology.

Reinforcement learning and neuro-dynamic programming are both relatively new fields in computer science. The techniques involved still have some rough edges and are not always fully understood, this is particularly the case with the application of function approximation to reinforcement learning. But the results shown above are very satisfactory yet.

A different function approximator may have shown more success. A radial basis function and tile coding, such as proposed by Sutton and Barto [31] could be a good approach to try, as Residual algorithms were considered as a solution but would require the Q-routing algorithm to be re-phrased as an episodic learning task, which may not be feasible. A more thorough and time consuming analysis of the methods proposed in this project could lead to a more successful application of neural network function approximation to the Q-routing problem.

References

[1]. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[2]. Justin Boyan and Michael Littman. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. Technical report, School of Computer Science, Carnegie Mellon University, 1993.

[3]. Kevin Gurney. *An Introduction to Neural Networks*. UCL Press Limited, 1997.

[4]. Chris M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[5]. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[6]. Si Wu and K. Y. Michael Wong. Dynamic Overload Control for Distributed Call Processors Using the Neural-Network Method. *IEEE-NN*, 9(6), 1998.

[7]. Gerald Tesauro. TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play. Technical report, IBM Corp., 1993.

[8]. Sabih H. Gerez Asker M. Bazen, Martijn van Otterlo and Mannes Poel. A Reinforcement Learning Algorithm for Minutiæ Extraction From Fingerprints. Technical report, Department of Computer Science, University of Twente, 2001.

[9]. Robert Crites and Andrew Barto. Improving Elevator Performance Using Reinforcement Learning. *Advances in Neural Information Processing Systems*, 1996.

[10]. Leemon Baird. Residual Algorithms: Reinforcement Learning with Function Approximation. Technical report, Department of Computer Science, U.S. Air Force Academy, 1995.

[11]. Geoffrey J. Gordon. Stable Function Approximation in Dynamic Programming. Technical report, Computer Science Department, Carnegie Mellon University, 1995.

[12]. John N. Tsitsiklis and Benjamin Van Roy. An Analysis of Temporal Difference Learning with Function Approximation. Technical report, Massachusetts Institute of Technology, 1996.

[13]. Tami Schuylerm Christos J. Moschovitis, Hilary Poole and Therese M. Senft. *History of the Internet*. ABC CLIO (Reference Books), 1999.

[14]. Mark K. Lottor. RFC 1296: Internet Growth (1981-1991). Technical report, Internet Engineering Task Force, 1992.

[15]. Larry L. Petersen and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, 2 edition, 2000.

[16]. Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall Inc, 1995.

[17]. Robert Braden. RFC 2039: Recommendations on Queue Management and Congestion Avoidance in the Internet. Technical report, Internet Engineering Task Force, 1998.

[18]. John Nagle. RFC 896: Congestion Control in IP/TCP Internetworks. Technical report, Internet Engineering Task Force, 1984.

[19]. W. Richard Stevens. RFC 2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. Technical report, Internet Engineering Task Force, 1997.

[20]. Random Early Detection Gateways for Congestion Avoidance. *IEEElACM Transactions on Networking*, 2(1):397–413, 1993.

[21]. Christophe Diot Martin May, Jean Bolot and Brian Lyles. Technical report, Internet Engineering Task Force, 2000.

[22]. Sally Floyd K. Ramakrishnan and D. Black. RFC 3168: The Addition of Explicit Congestion Notification (ECN) to IP. Technical report, Internet Engineering Task Force, 2001.

[23]. Sally Floyd. A Report on Some Recent Developments in TCP Congestion Control. Technical report, ACIRI, 2000.

[24]. Y. Rekhter and T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4). Technical report, Internet Engineering Task Force, 1995.

[25]. Abhijit Bhose Craig Labovitz, Abha Ahuja and Farnam Jahanian. An Analysis of BGP Convergence Properties. *Computer Communication Review*, 29 (4), 1999.

[26]. G. Malkin. RFC 1721: RIP Version 2 Protocol Analysis. Technical report, Internet Engineering Task Force, 1994.

[27]. J. Moy. RFC 2178: OSPF Version 2. Technical report, Internet Engineering Task Force, 1997.

[28]. John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1996.

[29]. Shailesh Kumar. Confidence based Dual Reinforcement Q-Routing: An Online Adaptive Network Routing Algorithm. Technical report, University of Texas, 1998.

[30]. Maria Gini Dean F. Hougen, John Fischer and James Slagle. Selforganizing Maps with Eligibility Traces: Unsupervised Control-Learning in Autonomous Robotic Systems. Technical report, Department of Computer Science, University of Minnesota, 1996.

[31]. GaryWilliam Flake. *The Computational Beauty of Nature*. MIT Press, 1998.

[32]. Richard Sutton. Reinforcement Learning Frequently Asked Questions. URL *http://www-anw.cs.umass.edu/~rich/RL-FAQ.html*.

[33]. L-J Lin. Reinforcement learning for robots using Neural Networks. PhD thesis. School of Computer Science, Carnegie Mellon University, 1993.

[34]. Littman, M., and Boyan, J. (1993a). A distributed reinforcement learning scheme for Network routing. Technical Report CMU-CS-93-165, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

[35]. Synoptic Study of Kohonen's Approach for the approximate solution of TSP by Ritesh Gandhi, 2001.

[36]. Mapping tasks to processors with aid of Kohonen's networks by HANS-ULRICH HEISS and MARCUS DORMANNS, Department of Informatics and Automation, Technical University of Ilmenau, 2002.

[37]. NEURAL NETWORKS by Christos Stergiou and Dimitrios Siganos. Technical report, McGraw-Hill, 1997.

Appendix-A
Snapshots

A1. Packet Generator

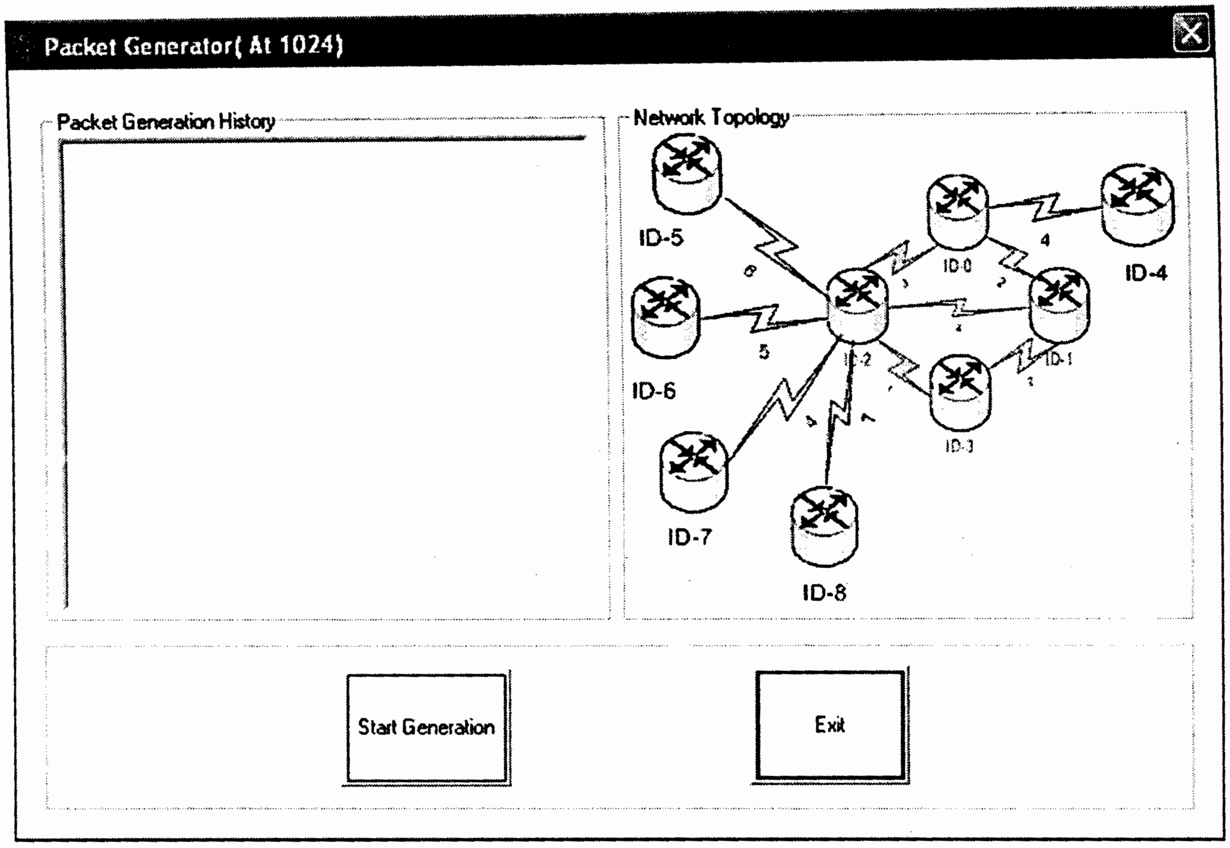


Figure A1: Interface for Packet Generator

A2. Router (Node-0)

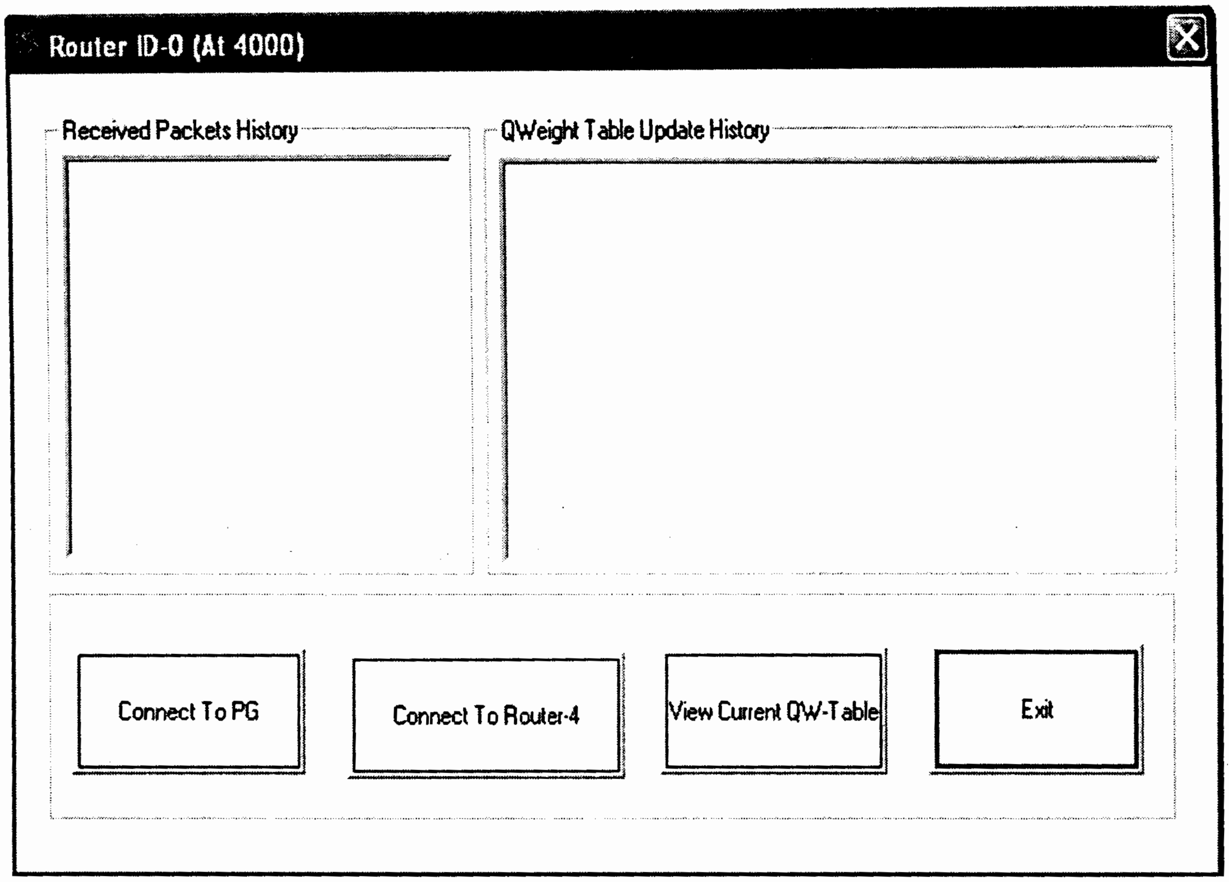


Figure A2: Interface for Router (Node-0)

A3. Router (Node-1)

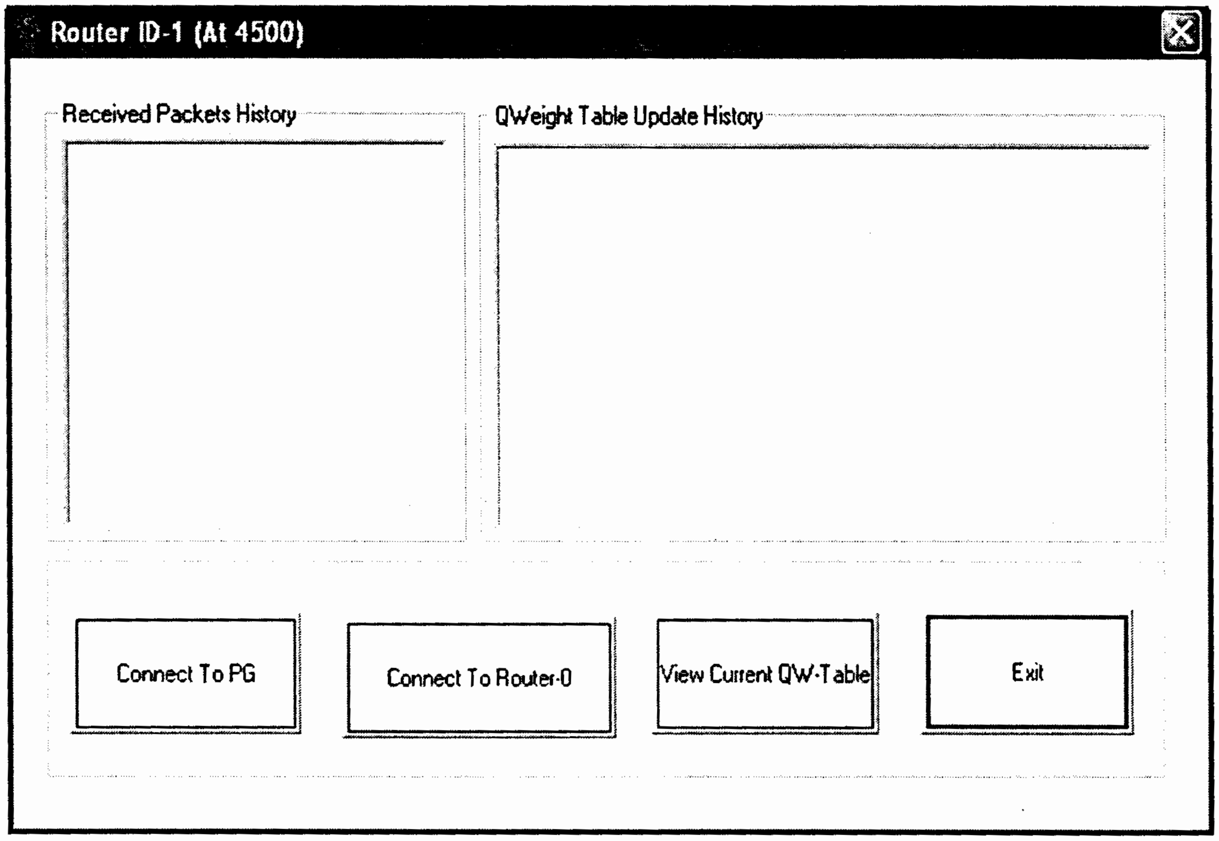


Figure A3: Interface for Router (Node-1)

A4. Router (Node-2)

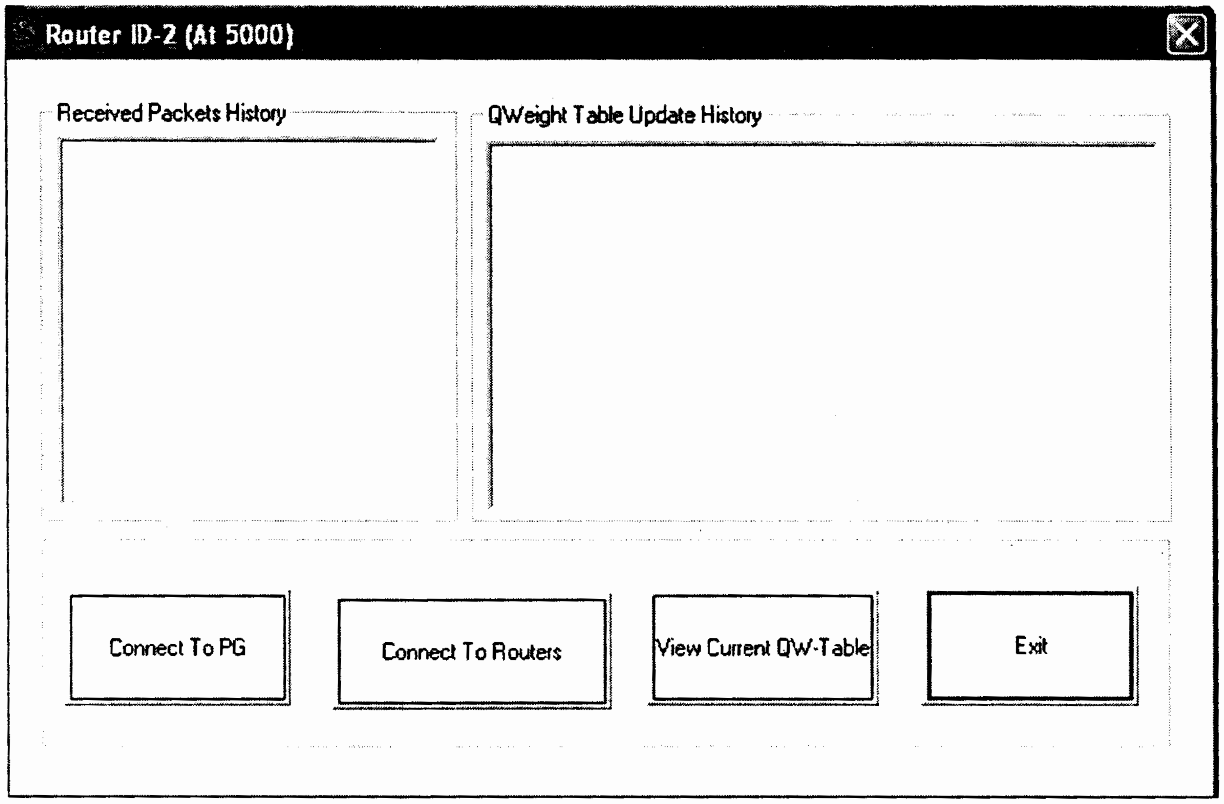


Figure A4: Interface for Router (Node-2)

A5. Router (Node-3)

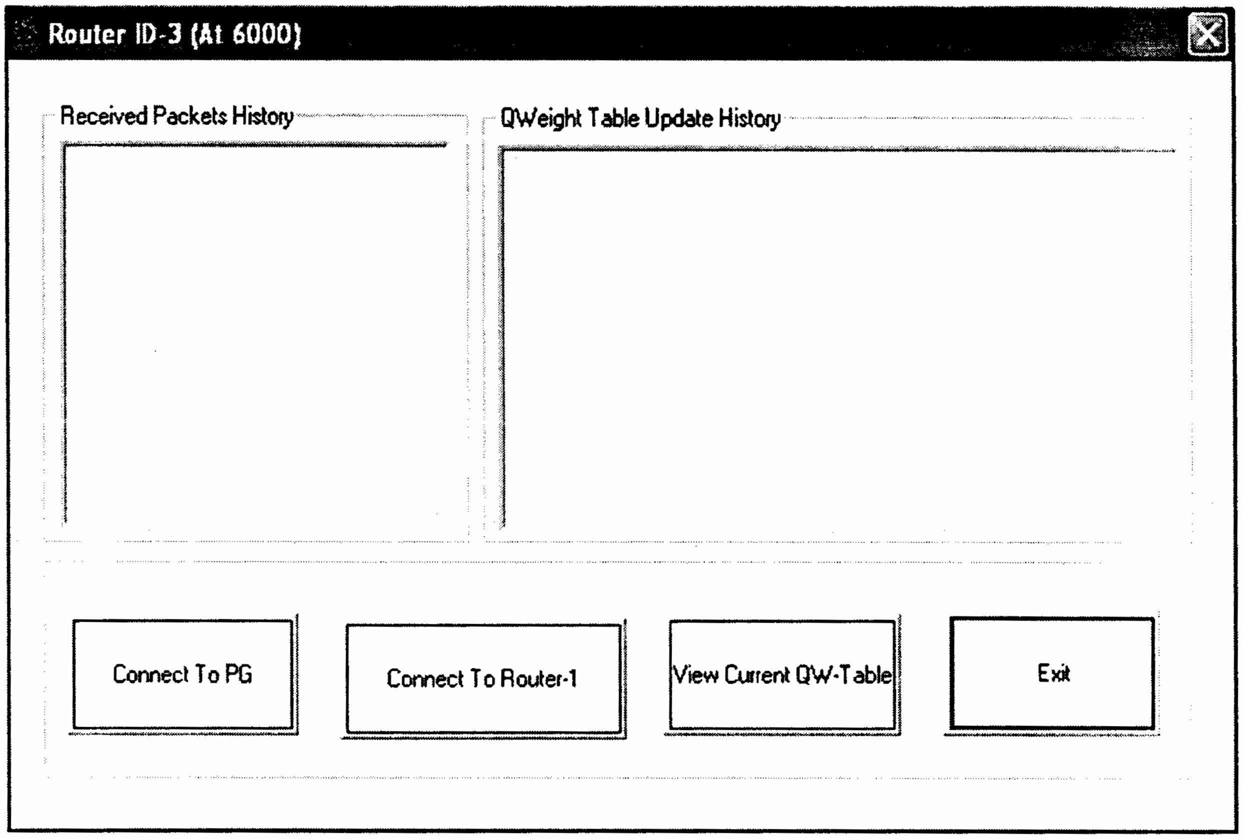


Figure A5: Interface for Router (Node-3)

A6. Router (Node-4)

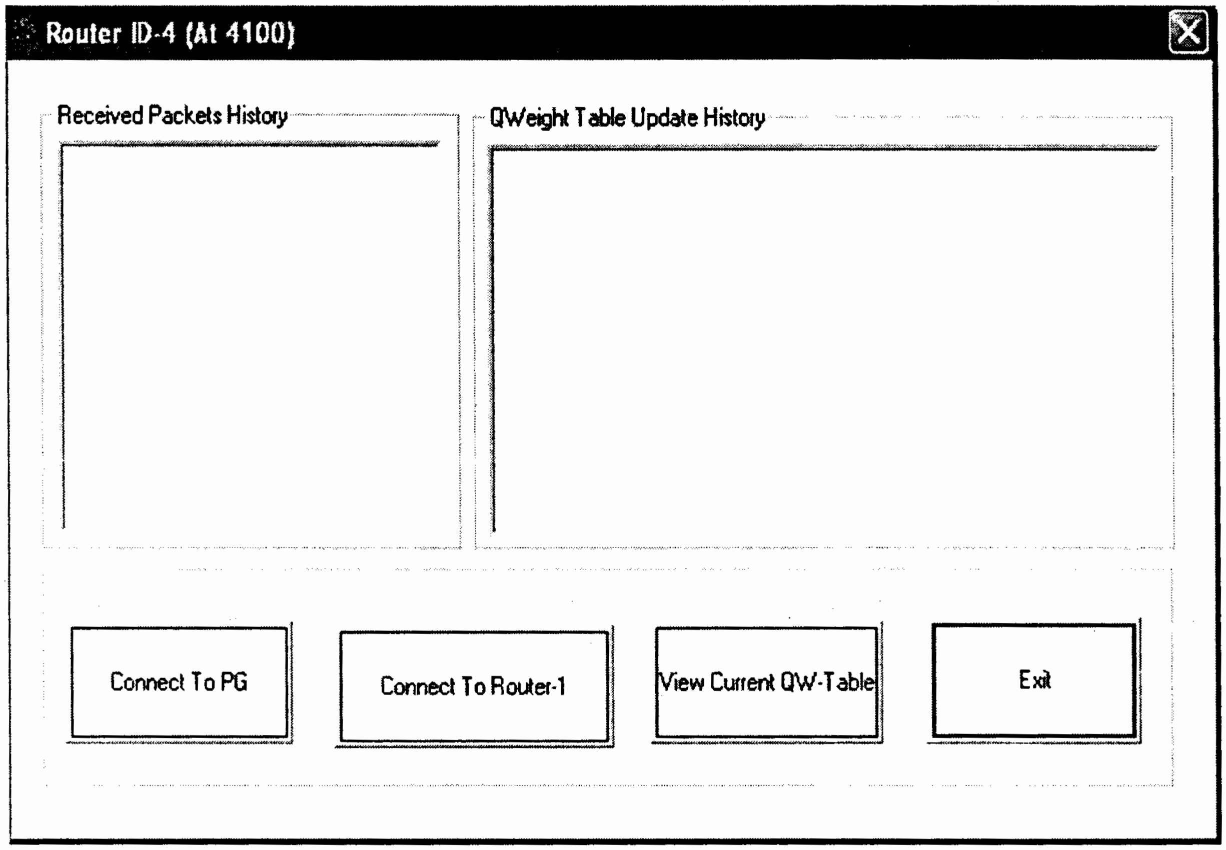


Figure A6: Interface for Router (Node-4)

A7. Router (Node-5)

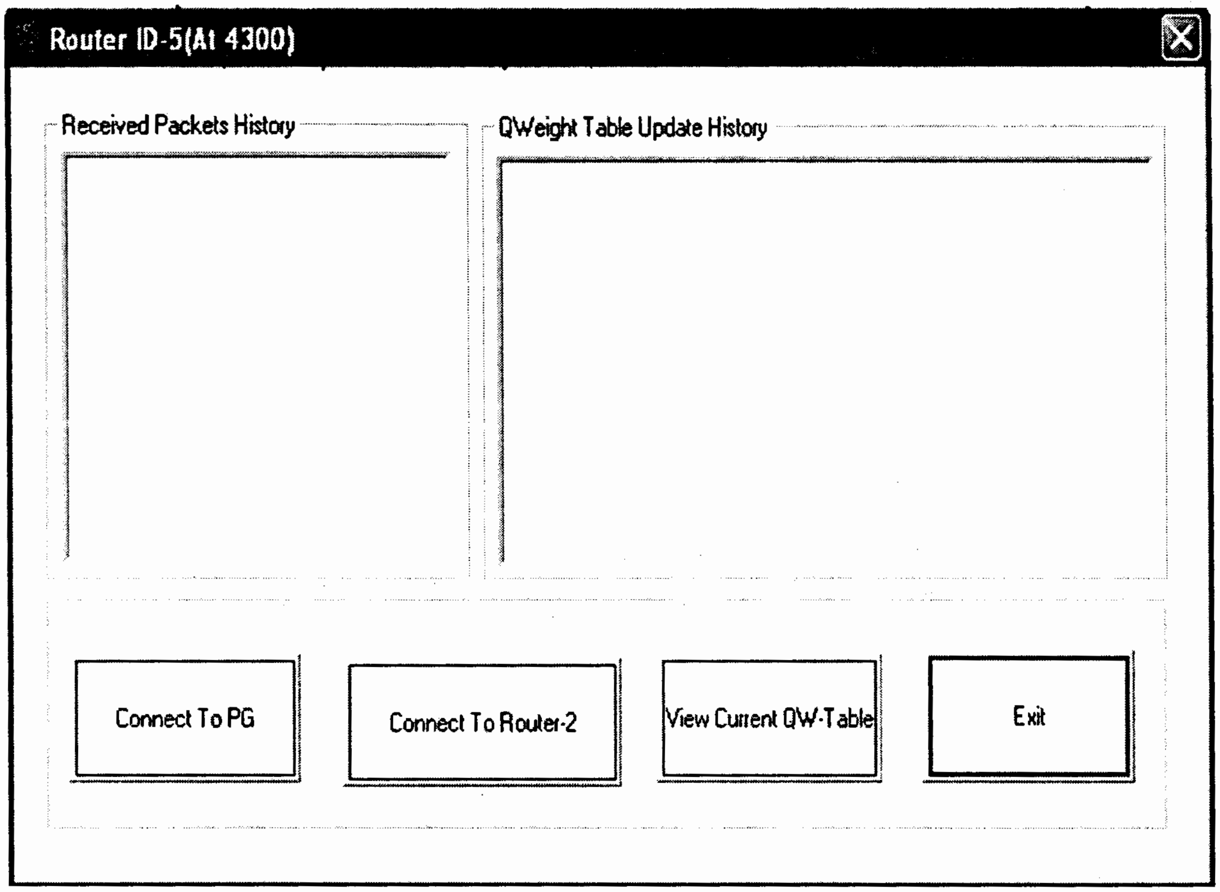


Figure A7: Interface for Router (Node-5)

A8. Router (Node-6)

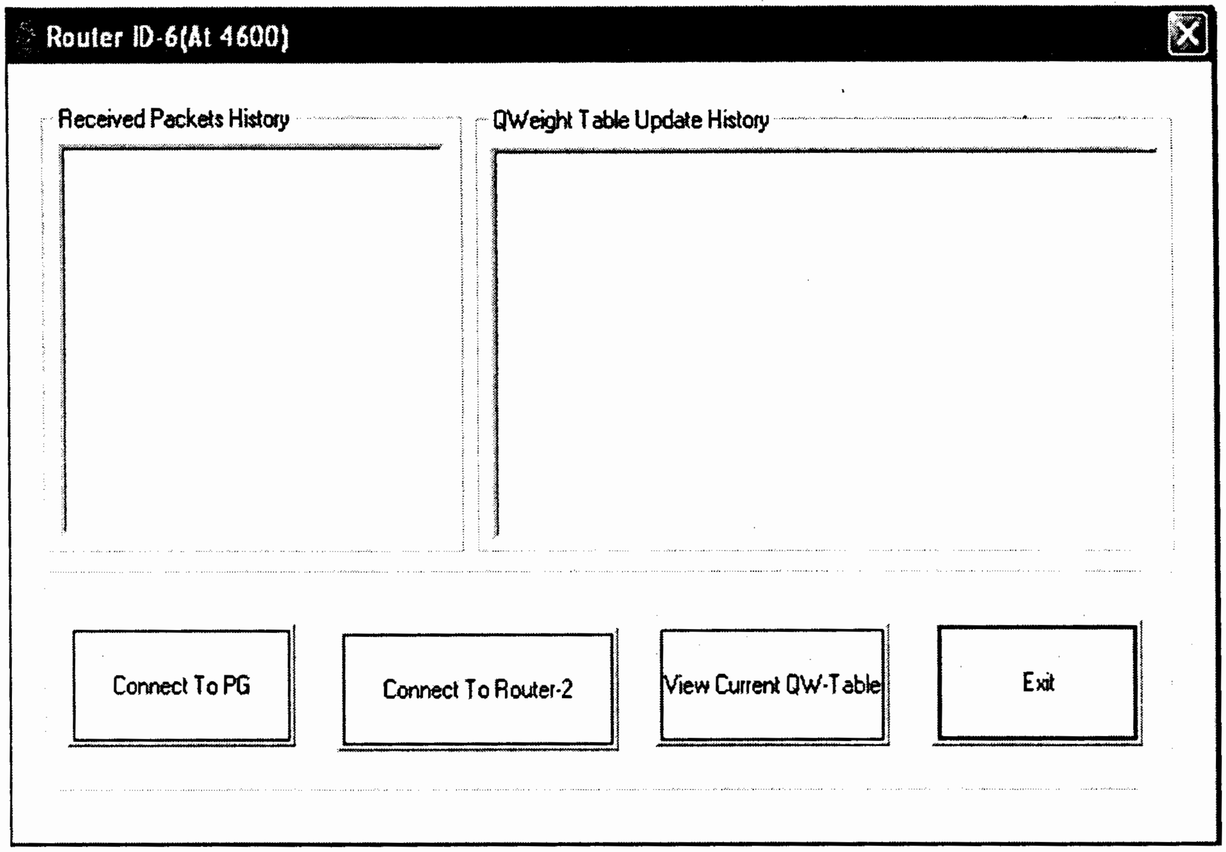


Figure A8: Interface for Router (Node-6)

A9. Router (Node-7)

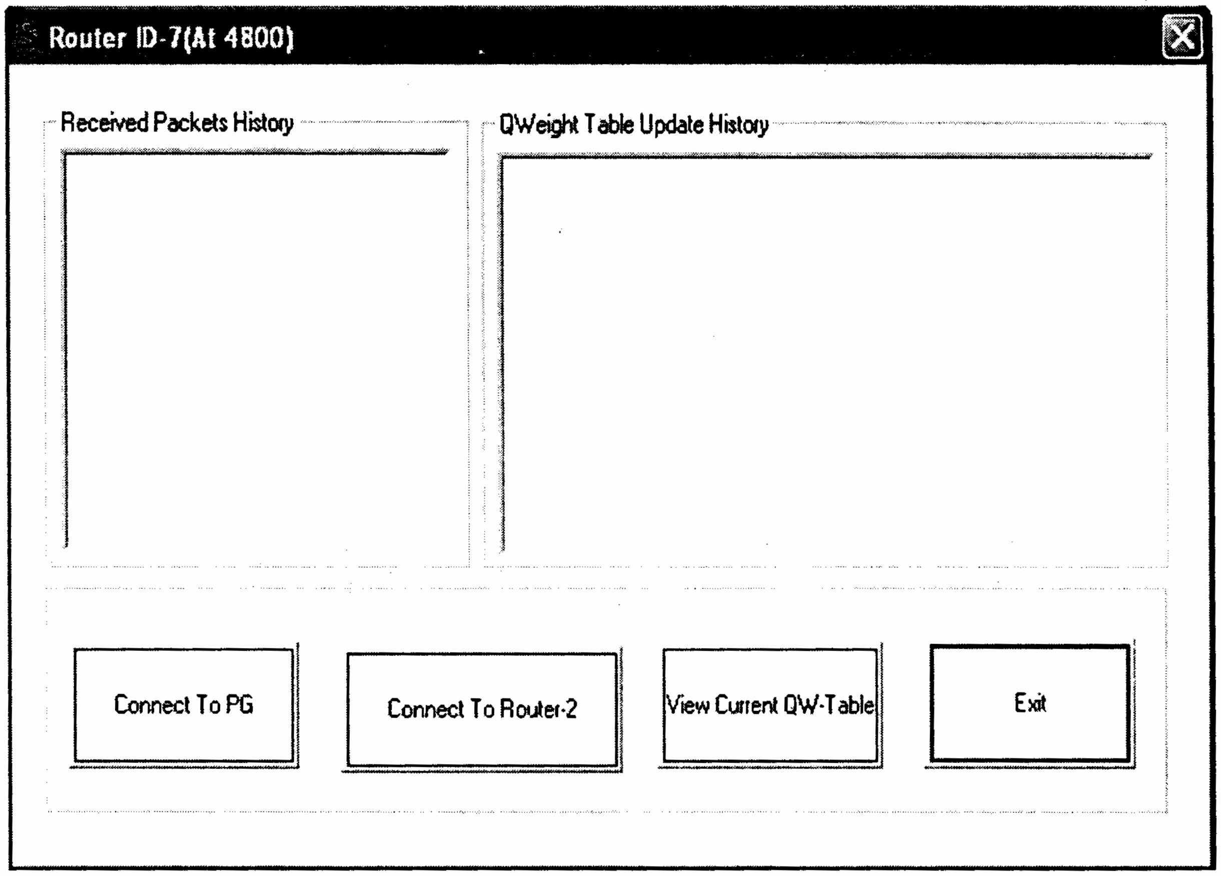


Figure A9: Interface for Router (Node-7)

A10. Router (Node-8)

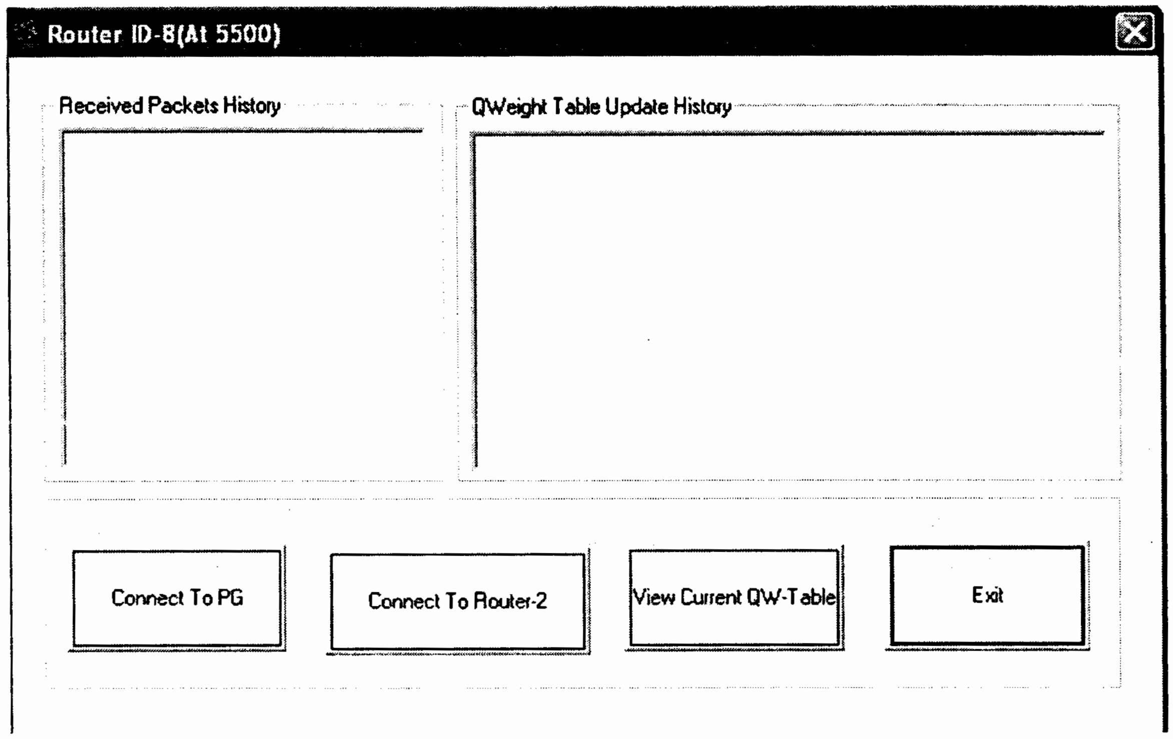


Figure A10: Interface for Router (Node-8)

Publication

© *European Journal of Scientific Research, Vol 9, No 2, 2005*

COMPLEX NETWORK TOPOLOGY LEARNING USING ANN A REINFORCEMENT LEARNING APPROACH

[1]Sadia Rasheed*, [1]Shireen Tahira, [2]Tauseef-Ur-Rehman and [1]M. Sikandar Hayat Khiyal
[1]Department of Computer Science, Faculty of Applied Sciences
International Islamic University, Islamabad Pakistan.
[2]Department of Telecom Engineering, Faculty of Applied Sciences
International Islamic University, Islamabad Pakistan

ABSTRACT

Efficient routing of packets in computer networks is a prerequisite for wide deployment of network-enabled devices, especially mobile and distributed computing. Current approaches to this problem have been partially successful. Boyan and Littman presented the Q-routing algorithm, a network routing algorithm based on Q-learning, a method from the emerging field of reinforcement learning. Q-routing learns to route packets in an adaptive manner. Neural networks have been used with some success to perform Q-learning, and would seem to be a possible method to allow Q-routing to scale well beyond its initial table-based implementation. This paper uses Q-routing algorithm. In this framework, the routing information at individual nodes is maintained as Q value estimates of how long it will take to send a packet to any particular destination via each of the node's neighbors. These Q values are updated through exploration as the packets are transmitted. This paper attempts to apply a neural network as a function approximator in an online reinforcement learning task, a field where neural networks have been used with varying degrees of success in the past. A discussion of the factors involved in neural network function approximation in reinforcement learning is provided. The main contribution of this work is the faster adaptation and improved quality of routing policies. The results achieved are satisfactory, some insight is gained into the difficulties of using neural networks as a function approximator in reinforcement learning tasks.

Key Words: Reinforcement Learning, Q-Routing, Artificial Neural Network (ANN), Function Approximation, Adhoc Networks, MANET.

1. INTRODUCTION

In a communication network information is transferred from one node to another as data packets. The process of sending a packet from its source node 's' to its destination node 'd' is referred to as *packet routing*. Normally it takes multiple "hops" to transfer a packet from its source to destination node. On its way, the packet spends some time waiting in the queues of intermediate nodes while they are busy processing the packets that **came** earlier. Thus the delivery time of the packet, defined as the time it takes for the packet to reach its destination, depends mainly on the total time it has to spend in the queues of the intermediate nodes.

Normally, there are multiple routes that a packet could take, which means that the choice of the route is crucial to the delivery time of the packet for any (s,d) pair. If there was a global observer with current information about the queues of all nodes in the network, it would be possible to make *optimal* routing decisions: always send the packet through the route that has the shortest delivery time at the moment. In the real world, such complete, global information is not available, and the performance of the global observer is an upper bound on actual performance. Instead, the task of making routing decisions is shared by all the nodes, each using only local information. Thus, a routing policy is a collection of local decisions at the individual nodes. When a node 'x' receives a packet P(d) destined for node 'd', it has to choose one of its neighboring nodes y such that the packet reaches its destination as quickly as possible.

The simplest such policy is the shortest-path algorithm, which always routes packets through the path with the minimum number of hops. This policy is not always good because some intermediate nodes, falling in a popular route, might have large queues. In such cases it would be better to send the packet through another route that may be longer in terms of hops but results in shorter delivery time. Hence as the traffic builds up at some popular routes, alternative routes must be chosen to keep the average packet delivery time low. This is the key motivation of this paper that it learns alternate routes through exploration as the current routing policy begins to lead to degraded performance.

Learning effective routing policies is a challenging task. In this paper, network makes routing decisions using Q-routing in which Kohonen's neural network is used as function approximator. This paper presents improvements in reinforcement learning. The algorithm presented here, aims to be stable as possible as under high loads while performing in less extreme situations.

2. BACKGROUND

Reinforcement learning

Reinforcement learning method is a method where supervised learning cannot easily be used because there are no sufficient data or external knowledge that can be applied, a reward signal. Maximizing this reward signal is the goal of reinforcement learning. Reinforcement learning algorithms develop a policy, usually defined as a mapping of states and subsequent actions to expected reward.

Temporal difference learning is a major part of reinforcement learning theory, and covers a number of methods such as TD(0), Q-learning and Sarsa.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha[r_{t+1}+\gamma Q(s_{t+1},a_{t+1})-Q(s_t,a_t)] \qquad (1)$$

The equation (1), referred to as the "Q-function" for quite obvious reasons, is the policy update rule used in Q-learning. The update rule states how the value of the Q-function is updated for each state s_t and for every action a_t that can be taken from that state. r_{t+1} is the immediate reward from this action, and $\gamma Q(s_{t+1},a_{t+1})$ is the discounted future reward from this state. The value γ is the discounting rate.

Q-Routing

In Q-routing, the routing decision maker at each node x makes use of a table of values $Q_x(y; d)$, where each value is an estimate, for a neighbor y and destination d, of how long it takes for a packet to be delivered to node d, if sent via neighbor y, excluding time spent in node x's queue. When the node has to make a routing decision it simply chooses the neighbor y for which $Q_x(y; d)$ is minimum. Learning takes place by updating the Q values.

On sending P(d) to y, x immediately gets back y's estimate for the time remaining in the trip, namely

$$Q_y(\hat{z}, d) = \min_{z \in N(y)} Q_y(z, d) \qquad (2)$$

where N(y) denotes the set of neighbors of node y. if the packet spent q_x units of time in x's queue, then x can revise its estimate based on this feedback:

$$\Delta Q_x(y, d)=\eta(Q_y(\hat{z}, d)+ q_x + \delta- Q_x(y, d)) \qquad (3)$$

Where η is the "learning rate" constant for all Q-values updates and δ is a transmission delay over the link between nodes x and y (assumed same for all links).

Neural Networks (Kohonen's Networks)

The objective of a Kohonen's network is to map input vectors (patterns) of arbitrary dimension N onto a discrete map with 1 or 2 dimensions. Patterns close to one another in the input space should be close to one another in the map: they should be topologically ordered. A Kohonen's network is composed of a grid of output units and N input units. The input pattern is fed to each output unit. The input lines to each output unit are weighted. These weights are initialized to small random numbers.

The learning process is as roughly as follows:

- initialize the weights for each output unit
- loop until weight changes are negligible
    - for each input pattern
        - present the input pattern
        - find the winning output unit
        - find all units in the neighborhood of the winner
        - update the weight vectors for all those units
    - Reduce the size of neighborhoods if required.

The winning output unit is simply the unit with the weight vector that has the smallest Euclidean distance to the input pattern. The neighborhood of a unit is defined as all units within some distance of that unit on the map (not in weight space). The weights of every unit in the neighborhood of the winning unit (including the winning unit itself) are updated using:

$$w_{ni} = w_{ni} + phi * r_{ij} * (N_n - w_{ni}) \qquad (4)$$

where $r_{ij} = e^{(-dist(i, j)^2)} / (2*theta)$

phi = Learning rate parameter

theta = Learning Factor

dist(i, j) = Guassian density function.

This will move each unit in the neighborhood closer to the input pattern. As time progresses the learning rate and the neighborhood size are reduced.

3. RELATED WORK

Reinforcement learning is a relatively new and emerging area of machine learning theory. Very limited work has done in this area of field. *Richard S. Sutton and Andrew G. Barto* [1] presented the basic intuitive sense of what reinforcement learning is and how it differs and relates to other fields, e.g. supervised learning and neural network, genetic algorithms and artificial life, control theory.

The field is still quite young, and there are problems that are still being researched which are proving difficult to solve. However, the reinforcement learning approach is very promising even in these early stages, and seems a good fit for the network routing problem as Boyan and Littman [2] showed. This paper describes the Q-Routing algorithm for packet routing in which a reinforcement learning module is embedded into each node of switching network.

Neural networks are a technique for pattern recognition and function approximation based originally on ideas from biology and the study of neurons [3], although the theory behind neural networks is based on statistical foundations [4]. Neural networks seek to mimic the apparently simple, but extremely effective, structure of the human brain.

Neural network has been used to some success to perform Q-learning. There has been notable success with this method using Q-learning and a neural network which approximate the value function. *Will Newton* was the one who presented the idea of using Neural Network as function approximator in online reinforcement learning task [5]. The results achieved were disappointing, but some insight is gained into the difficulties of using neural networks as a function approximator in reinforcement learning tasks.

Ritesh Gandhi [6] used Kohonen's Neural Network to solve Traveling salesman problem (TSP).

4. METHODOLOGY

SSQ-Routing Algorithm

Q-routing and Kohonen's network are combined in this algorithm. Kohonen's network is used as function approximator. Q-value table is updated using Kohonen's network updating rule as in equation (2).

Main steps of the algorithm are:

1. Initial Q-values can be calculated as

$$r_{ij} = e^{(-\text{dist}(i,j)^2}) / (2{*}\textit{theta})$$

where

$$\text{dist}(i,j)= \exp(-d^2/\textit{phi}) / \text{sqrt}(2)$$

2. Take source node as 'starting node'.
3. Find the next neighboring node having minimum Q-value.

4. Update the Q-value for that selected node by using the equation derived from equation- (4) as

$$\Delta Q_x(y, d) = Q_x(y, d) + phi * r_{ij} * (Q_y(\hat{z}, d) - Q_x(y, d))$$

Where

$$Q_y(\hat{z}, d) = \min_{z \epsilon N(y)} Q_y(z, d)$$

5. Decrease '*theta'*, '*phi'* and recalculate 'r' value for the selected node only.
6. Compare selected node with 'ending node', if not equal then, go to step 3 else exit.

5. IMPLEMENTATION

Simulation Environment

The simulation environment was built on NS-2, a discrete event simulator targeted at networking research. NS-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

Network Topology used

The network topology used for simulation is 6X6 irregular grid shown in figure-1 due to Boyan and Littman (1994). In this network, there are two possible ways of routing packets between the left cluster (nodes 1 through 18) and the right cluster (nodes 19 through 36): the route including nodes 12 and 25 (R1) and the route including nodes 18 and 19 (R2). For every pair of source and destination nodes in different clusters, either of the two routes, R1 or R2 can be chosen. Convergence to effective routing policies, starting from either random or shortest path policies.

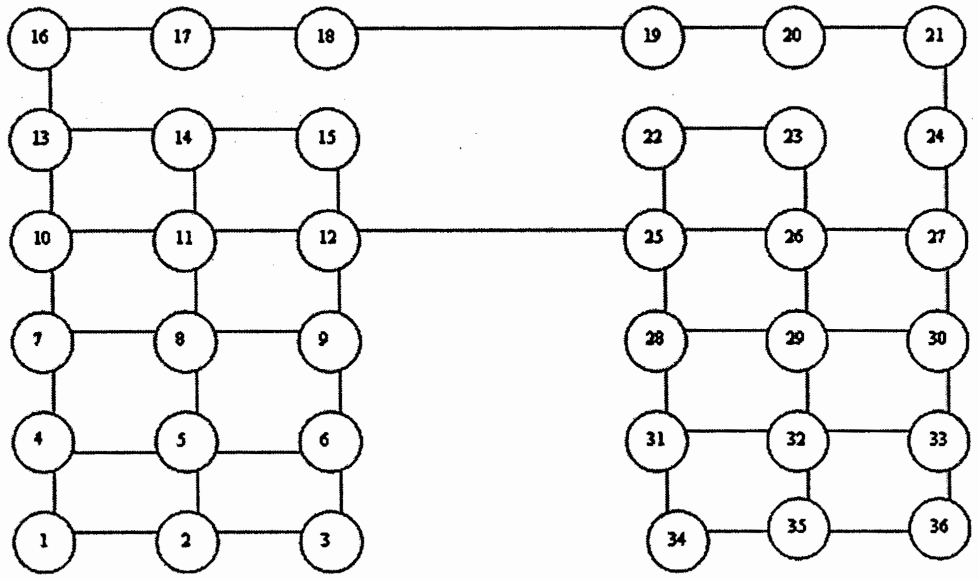


Figure- 1: The 6X6 Irregular Grid. The left cluster comprises of nodes 1 through 18 and the right cluster of nodes 19 through 39. The two alternative routes for traffic between clusters are the route including the link between nodes 12 and 25 (route R1) and the route involving the link between nodes 18 and 19 (route R2). R1 becomes a bottleneck with increasing loads and the adaptive routing algorithm needs to make use of R2.

Parameters

The Q tables were initialized with small random Q values, except for the base cases. Learning rate for forward exploration η_f , value for *'theta'*, and *'phi'* are summarized in table 1. Performance of the algorithms was found to be the best with these parameters.

Table 1- Parameters

Algorithms	η_f	*Theta*	*Phi*
Q-Routing	0.85	-	-
SSQ-Routing	-	0.5	0.5

In Table 1 Parameters for the Algorithms are shown. In Q-Routing and SSQ-routing where the learning rates vale for *'theta'* and *'Phi'* are constant.

6. RESULTS

Learning at Constant Loads

In the first set of experiments, the load level was maintained constant throughout the simulation. Results on a network topology, namely the 36-node irregular 6*6 grid (figure -1) are presented. The speed and quality of adaptation at three load levels, low, medium and high, were compared. The typical load level values for the topology are given in table- 2

The adaptive routing algorithms were found to have relatively similar performance and learning behaviors (in average packet delivery time) within a given range.

Table 2- Load Level Ranges

Topology	# Nodes	Low Load	Medium Load	High Load
6 X 6 Irregular Grid	36	0 - 1.75	1.75 - 2.50	2.50 - more
7- D hypercube	128	0 - 5	5 - 8	8 - more

In Table 2 Load level ranges the number stands for number of packets Introduced in the network per time step. The learning behavior of adaptive routing algorithms remains roughly similar within a given load range (low, medium or high), but if the load changes from one range to another the behavior can be changed quite dramatically. In real life communication networks, the load is usually in the medium range, and occasionally changes to low or high levels.

The performance and learning behavior is significantly different from one load level to another for some routing algorithms. The learning behavior was observed in terms of average packet delivery time and number of packets in the network during learning.

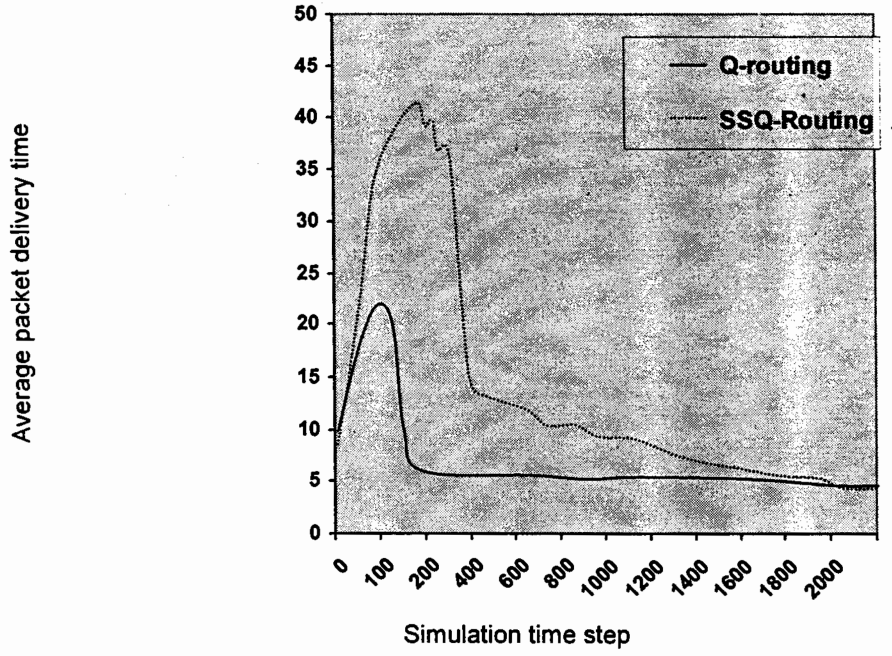


Figure 2: Average packet delivery time at a low load level for the grid topology: Difference between Q-Routing and SSQ-Routing is statistically significant between 300 to 800 time steps.

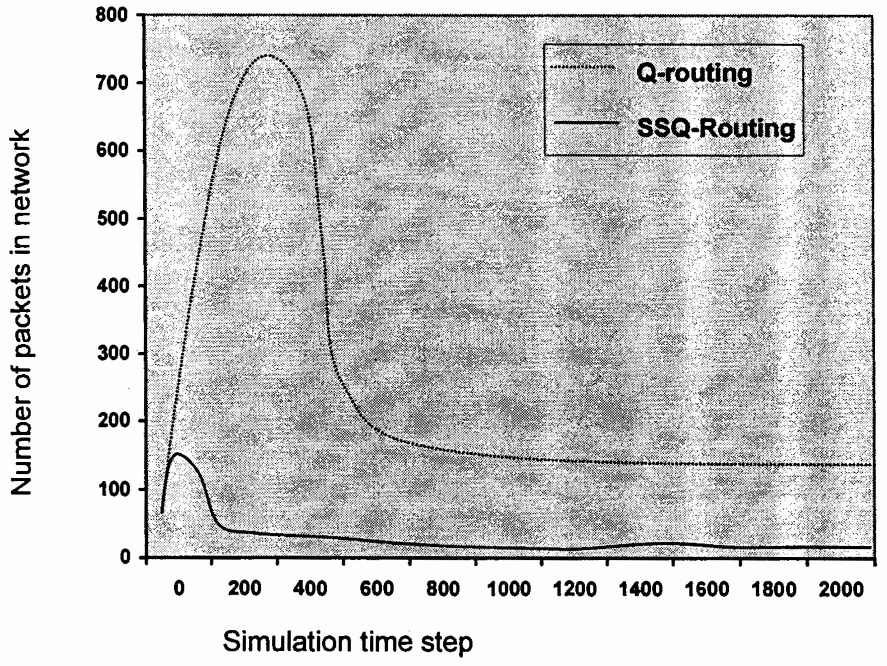


Figure 3: Number of packets for 7-D hypercube at low load: Difference between SSQ-Routing and Q-Routing is significant after 150 time step.

At low load the average packet delivery time for both the grid (figure 2) and the hypercube (figure 3) shows that there is not much gain in the speed of learning from Q-Routing to SSQ-Routing. Reason for this trend is that at low loads, what matters the most is the amount of exploration and the algorithm that allows more exploration per packet hop will learn faster. SSQ-Routing learns more than 3 times as fast as Q-Routing for both topologies.

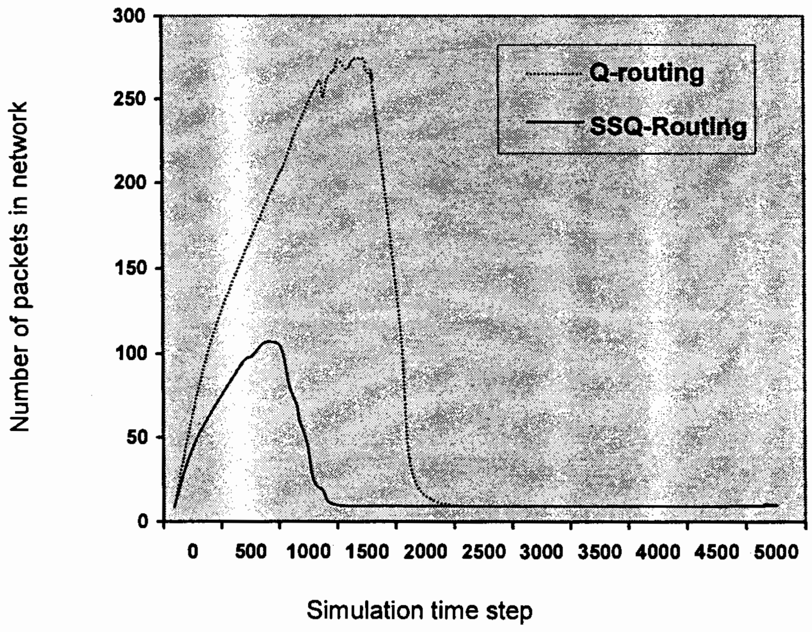


Figure 4: Average packet delivery time for the grid topology at a medium load level: Difference between SSQ-Routing and Q-Routing is significant between 300 to 1900 time steps.

At **medium load levels,** the average packet delivery times (figure 4 for the grid and 5 for the hypercube) shows that both SSQ-Routing performs better than Q-Routing. This result is significant because it highlights the contribution of both the quality and quantity of exploration in learning. They contribute in two different ways to increase the performance of SSQ-Routing.

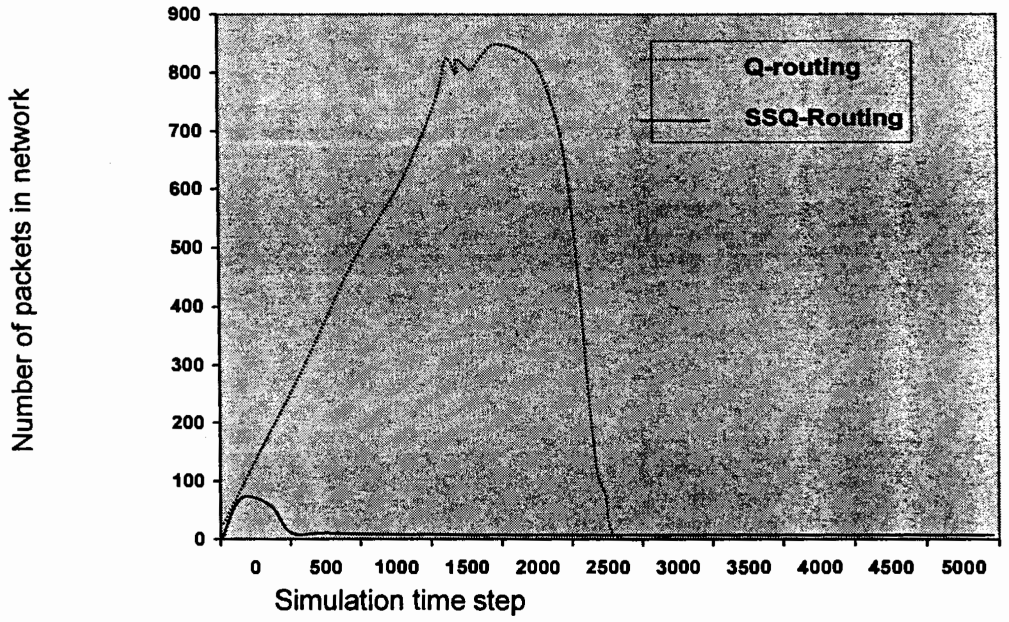


Figure 5: Average packet delivery time for 7-D hypercube at medium load: Difference between SSQ-Routing and Q-Routing is significant between 200 to 2750 time steps.

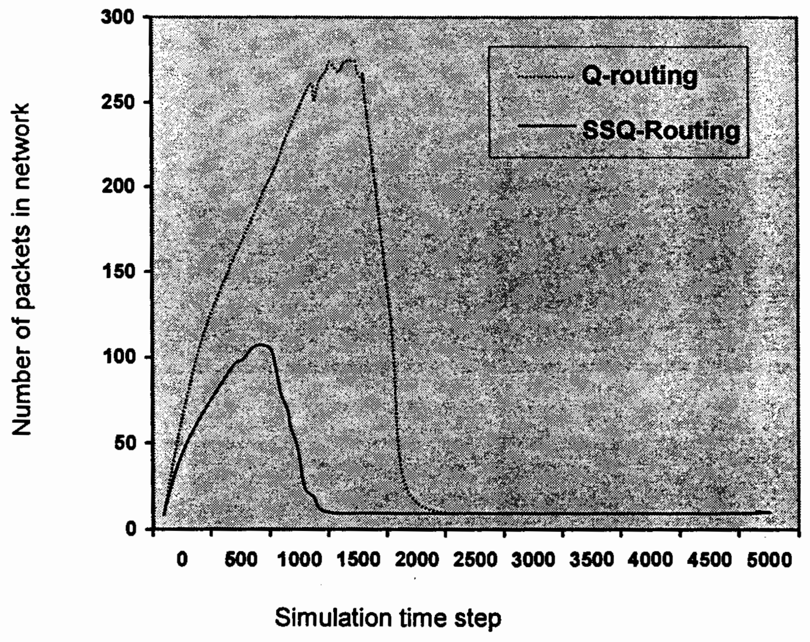


Figure 6: Average packet delivery time for grid topology at high load: Difference between SSQ-Routing and Q-Routing is significant between after 1000 time steps.

At **high load levels** the average packet delivery time (figure 6) for the grid topology show that while Q-Routing converges to a qualitatively poor routing policy, and SSQ-Routing converge to qualitatively similar policies.

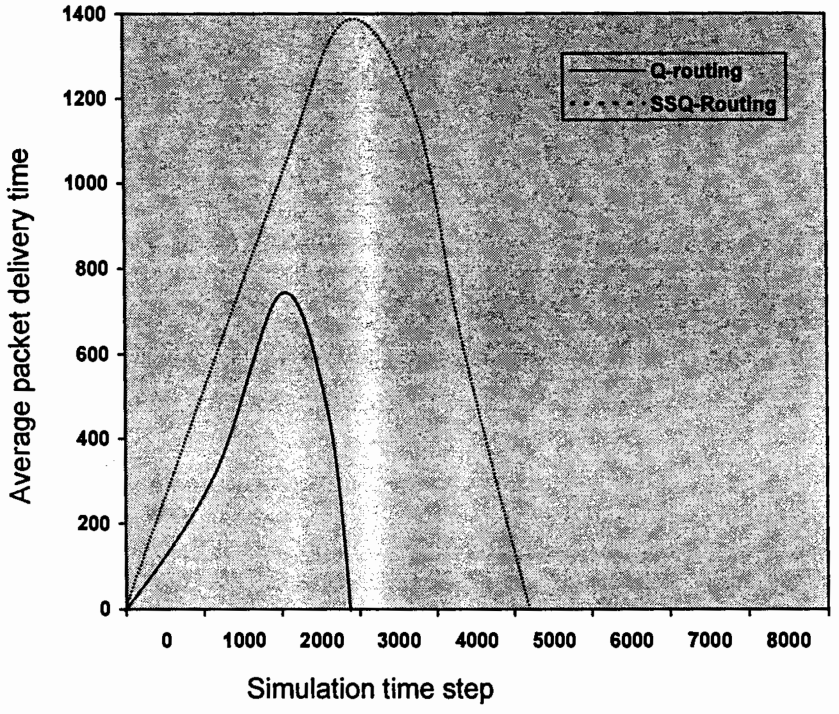


Figure 7: Average packet delivery time for 7-D hypercube at high load: All differences are significant between 2400 to 300 time steps. Q-Routing is significantly different from SSQ-Routing

Adaptation to Changing Network Topology
The third set of experiments compared the routing algorithms' ability to adapt to the changes in network topology. A link was added between nodes 3 and 34 in the 6*6 grid topology (figure-1) for these experiments. The routing algorithms were first allowed to learn an effective routing policy for the new network at a load level of 2.0 pkts/step until they converged (in 2000 time steps). At time step 2000, the link between node 12 and 25 was removed (figure 6). That is, the Q tables of node 12 and 25 were

updated such that $Q_{12}(25,*)$ and $Q_{25}(12,*)$ were all set to Infinite Cost and the corresponding routing tables were also updated accordingly. The C values were not changed. Only single run is shown in figure-7 to depict the variations in adaptation process. However, statistical significance is computed over 50 runs and is given below.

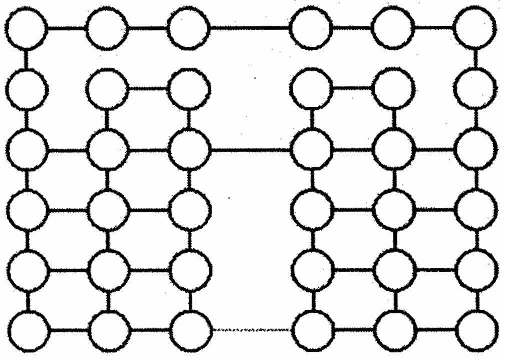


Figure 8: Grid topology before link 12 and 25 went down. This Topology is different from the one in figure 1. it has an additional link between 3 and 34 nodes.

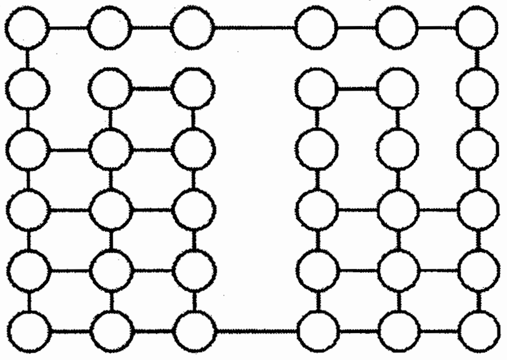


Figure 9: Grid topology after link between 12 and 25 has been removed, by fixing vectors $Q_{12}(25,*)$ and $Q_{25}(12,*)$ to infinite cost.

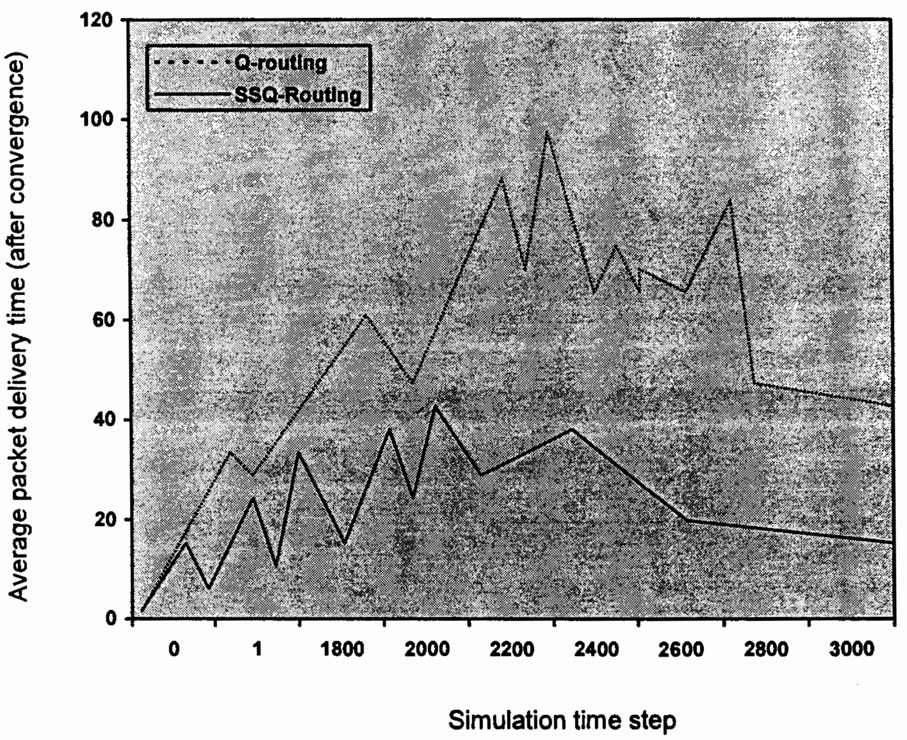


Figure 10: Adaptation to change a in the network topology. Q-Routing does not converge to an effective policy while SSQ-Routing converges to an effective policy very fast.

7. CONCLUSIONS AND FUTURE WORK

SSQ-Routing with higher quantity of exploration and better quality of exploration than

Q-Routing was developed in this work. It was evaluated and compared to Q-Routing.

The ability to learn an effective routing policy starting from a random policy, the ability to adapt to changes in traffic patterns, and the ability to adapt to changes in network topology. SSQ-Routing is an improvement over the conventional adaptive routing algorithms such as BF in a number of ways. SSQ-Routing strikes a balance between the amounts of overhead incurred and speed of adaptation whether it is from random policy at fixed loads or to change in traffic pattern or change in network topology. Reinforcement learning and neuro-dynamic programming are both relatively new fields in computer science. The techniques involved still have some rough edges and are not always fully understood, this is particularly the case with the application of function approximation to reinforcement learning. But the results shown above are very satisfactory yet.

A different function approximator may have shown more success. A radial basis function and tile coding, such as proposed by Sutton and Barto [7] could be a good approach to try, as Residual algorithms were considered as a solution but would require the Q-routing algorithm to be re-phrased as an episodic learning task, which may not be feasible. A more thorough and time consuming analysis of the methods proposed in this project could lead to a more successful application of neural network function approximation to the Q-routing problem.

REFERENCES

[1]. Robert Crites and Andrew Barto; Improving Elevator Performance Using Reinforcement Learning; *Advances in Neural Information Processing Systems*, 1996.

[2]. Justin Boyan and Michael Littman; Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach; Technical report, School of Computer Science, Carnegie Mellon University, 1993.

[3]. Kevin Gurney; An Introduction to Neural Networks; UCL Press Limited, 1997.

[4]. Chris M. Bishop; Neural Networks for Pattern Recognition; Oxford University Press, 1995.

[5]. Will Newton, A neural Network algorithm for internetwork routing; MS thesis; 2002.

[6]. Ritesh Gandhi.Synoptic Study of Kohonen's Approach for the approximate solution of TSP;

[7]. Richard S. Sutton and Andrew G. Barto; Reinforcement Learning: An introduction; MIT Press, 1998.