Distributed and Intelligent Routing Algorithm

Firat Tekiner

A thesis submitted in partial fulfilment of the requirements of
Northumbria University, Newcastle
for the degree of
Doctor of Philosophy

Research undertaken in the
School of Computing, Engineering & Information Sciences

October 2005

ACKNOWLEDGEMENTS

I would like to thank my director of studies, supervisor and above all friend Prof. Z. Ghassemlooy, for giving this opportunity to me and for his inspirational guidance and support. He has been an invariable source of help throughout the research project.

I would like to thank to my girlfriend Yontem who then became my wife at the end of this thesis for her patience and support throughout this thesis. I would also want to thank to my mum and to my dad for their support throughout my life.

I would like to acknowledge the Sheffield Hallam University and Northumbria University as this work would not be possible without the financial support provided by the Sheffield Hallam University and Northumbria University PhD scholarships throughout my PhD project.

Many thanks to Dr. Samir Al-khayatt and Dr. Mark Thompson for their support especially for tedious proofreading of the thesis. I am grateful for the encouragement and support received from many friends that I made whilst studying at Sheffield and Newcastle and especially to Dr. Bala Amavasai for his patience and support with the linux systems. I would like to thank to my colleagues and friends in the Optical Communications Research Group.

Last but not least, I would like to thank to my family and friends back home in Cyprus for their support and encouragements during this study.

DECLARATION

I hereby declare that this thesis is entirely my own work and has not be submitted in support of an application of another degree or qualification of this or any other university, institute of learning or industrial organisation.

ABSTRACT

A Network's topology and its routing algorithm are the key factors in determining the network performance. Therefore, in this thesis a generic model for implementing logical interconnection topologies in the software domain has been proposed to investigate the performance of the logical topologies and their routing algorithms for packet switched synchronous networks. A number of topologies are investigated using this model and a simple priority rule is developed to utilise the usage of the asymmetric 2 × 2 optical node. Although, logical topologies are ideal for optical (or any other) networks because of their relatively simple routing algorithms, there is a requirement for much more flexible algorithms that can be applied to arbitrary network topologies.

Antnet is a software agent based routing algorithm that is influenced by the unsophisticated and individual ant's emergent behaviour. In this work a modified antnet algorithm for packet switched networks has been proposed that offers improvement in the packet throughput and the average delay time. Link usage information known as "evaporation" has also been introduced as an additional feedback signal to the algorithm to prevent stagnation within the network for the first time in the literature for the best our knowledge. Results show that, with "evaporation" the average delay experienced by the data packets is reduced nearly 30% compared to the original antnet routing algorithm for all cases when non-uniform traffic model is employed. The multiple ant colonies concept is also introduced and applied to packet switched networks for the first time which has increased the packet throughput. However, no improvement in the average packet delay is observed in this case. Furthermore, for the first time extensive analysis on the effect of a confidence

parameter is produced here. A novel scheme which provides a more realistic implementation of the algorithms and flexibility to the programmer for simulating communication networks is proposed and used to implement these algorithms.

GLOSSARY OF ABBREVIATIONS

ACO	Ant Colony Optimisation
ADVR	Agent Distance Vector Routing
APR	Ant to Packet Ratio
AS	Autonomous Systems
ASGA	Ant System Genetic Algorithm
BGP	Border Gateway Protocol
BT	British Telecom
CAF	Cooperative Asymmetric Forward
CDRQR	Confidence Based Dual Reinforcement Q-routing
CPU	Central Processing Unit
DRQR	Dual Reinforcement Q-routing
EBGP	Enhanced Border Gateway Protocol
EGP	Exterior Gateway Protocol
EIGRP	Enhanced Interior Gateway Routing Protocol
EPSRC	Engineering and Physical Sciences Research Council
FIFO	First In First Out
GA	Genetic Algorithms
HCRNET	Hypercube Connected Rings NETwork
IDRP	Inter-Domain Routing Protocol
IGP	Internet Gateway Protocol
IGRP	Interior Gateway Routing Protocol
I-H-U	I-Heard-You
IP	Internet Protocol
ISP	Internet service Provider
LSA	Link State Advertisements
MACO	Multiple Ant Colony Optimization
MAGMA	A Multiagent Architecture for Metaheuristics
MSN	Manhattan Street Network
OSPF	Open Shortest Path First
OTDM	Optical Time Division Multiplexing
PQR	Predictive Q-routing

PVM	Parallel Virtual Machine
QoS	Quality of Service
RIP	Routing Information Protocol
SPF	Shortest Path First
TID	Task Identifier
TOAD	Terahertz Optical Asymmetric Demultiplexer
TRR	Transmission/Reflection Ratio
TSP	Travelling Salesman Problem
WDM	Wavelength Division Multiplexing

GLOSSARY OF SYMBOLS

Symbol	Definition
Δ	In and out degree of de bruijn graph
a′	Ant parameter
a	*E*vaporation constant
A	*A*gent
b′	Ant parameter
b	Evaporation constant
b_1	System parameter
b_2	System parameter
b_3	System parameter
$B_x(d,y)$	B-Table for the path from node *x* to node *d* via neighbour node *y*
c	Scale factor
c′	Contention ratio
C	The number of unordered node pairs forming a cycle
C′	Effective route capacity, Bandwidth
C_x(d,y)	C-Table for the path from node *x* to node *d* via neighbour node *y*
d	In and out degree of kautz graph
d_l	Link propagation delay
D	Diameter
D′	Route delay
D_c	Column direction
D_{ij}	Minimum cost path from source node *i* to destination node *j*
D_r	Row direction

E	Number of edges of a digraph
E(x)	Evaporation parameter
g	Noise
h	Average hop distance
k	Number of columns
k_1	Coefficient
k_2	Coefficient
L	Load
L(x)	Link usage ratio
L_k	Statistical model
l_n	Heuristic correction value
m	Columns MSN
m′	No of rows
Mk	Statistical Array
n	Rows MSN
N	Number of nodes
$N_{h\rightarrow p}$	Average number of hops per packet
N_H	Total number of hops
N_T	Total number of packets
$o_{k\rightarrow d}$	Observed trip time experienced while travelling from *k* to destination *d*
p	In and out degree of shufflenet and gemnet
P_{nd}	Probability of node n to destination *d*
q_l	Total size of the packets that are waiting in the queue
q_n	Number of packets waiting to be sent to the queue of the output port *n*
q_x	Current queue delay for the packet in node *x*

r	Reinforcement signal (goodness value)
R	Recent usage ranking parameter
R′	Routing code
R″	Reliability
$R_x(d,y)$	R-Table for the path from node *x* to node *d* via neighbour node *y*
S	Stability factor
S(k)	Stack associated with node *k*
s_a	The size of the current packet
t	Ant creation rate
t1	Timeout value
t2	Timeout value
T′	Absolute age of the ant
T	Observed trip time
T_b	System parameter
Tk	Routing table for node *k*
U	Instability factor
$U_x(d,y)$	U-Table for the path from node *x* to node *d* via neighbour node *y*
V	Set of all the edges of the digraph
v	Variance
W_d	observation time window to destination *d*
X	Distance
Y	Confidence parameter
$z\ (Q_y(\acute{z},d))$	Old estimation
α	Value that weights the importance of the heuristic function
β	Learning parameter for the recovery rate

γ	Decay in the recovery rate
$\Delta Q_x(y,d)$	Estimation value for node x to destination d via the neighbour node y
δr	Reinforcement parameter
ε	Antnet constant
η	q-function learning parameter
η_f	Learning rate parameter
μ_d	Estimated mean
σ_d^2	Variance

List of Figures

Fig. 2.1: A 24-node ($p = 2$, $k = 3$) Shufflenet topology. 14
Fig. 2.2: Pseudo code for Shufflenet routing algorithm 15
Fig. 2.3: An 8 Node ($\Delta = 2$, $D = 3$) De Bruijn graph 16
Fig. 2.4: Pseudo code for De Bruijn graph routing algorithm 18
Fig. 2.5: A 10 node ($k = 2$, $m = 5$, $p = 2$) Gemnet topology 19
Fig. 2.6: Pseudo code for Gemnet routing algorithm 20
Fig. 2.7: Kautz graph 21
Fig. 2.8: A 64 node ($m = 8$, $n = 8$) MSN 23
Fig. 2.9: 8 node 3-cube 28
Fig. 2.10: HCRNET 30
Fig. 3.1: Destination/next hop association in routing table 35
Fig. 3.2: Pseudo code for general link state routing algorithm 39
Fig. 3.3: Pseudo code for Distance Vector algorithm 41
Fig. 4.1: Agent environment interaction 53
Fig. 4.2: Main components of reinforcement learning agent 54
Fig. 4.3: The process of how ants find optimal path in real life 57
Fig. 4.4: Pseudo code for ACO meta-heuristic 59
Fig. 4.5: Pseudo code for ABC algorithm 64
Fig. 4.6: Data structures stored in every node 67
Fig. 4.7: Antnet routing algorithm's pseudo code 73
Fig. 4.8: Pseudo code for ASGA Algorithm 74
Fig. 4.9: Multiple Ant Colonies 78
Fig. 4.10: Forward exploration 82
Fig. 4.11: Forward and backward exploration 84
Fig. 4.12: Pseudo code for PQR 86
Fig. 5.1: Node model 92
Fig. 5.2: Shufflenet showing possible worst and best paths using prioritised routing 93
Fig. 5.3: Packet format 95
Fig. 5.4: Interconnection network routing algorithm framework 98
Fig. 5.5: Implementation flowchart 101
Fig. 5.6: Implementation flowchart with buffer 103
Fig. 5.7: Average number of hops versus system load for Shufflenet employing deflection routing 106
Fig. 5.8: Average number of hops versus system load for De Bruijn Graph employing deflection routing 107
Fig. 5.9: Average number of hops versus system load for Gemnet employing deflection routing 108
Fig. 5.10: Average number of hops versus system load for Shufflenet employing store-and-forward routing 108

Fig. 5.11: Average number of hops versus system load for De Bruijn Graph employing store-and-forward routing 109
Fig. 5.12: Average number of hops versus system load for Gemnet employing store-and-forward routing 109
Fig. 5.13: Contention ratio versus system load for Shufflenet employing deflection routing 111
Fig. 5.14: Contention ratio versus system load for Gemnet employing deflection routing 111
Fig. 5.15: Contention ratio versus system load for De Bruijn Graph employing deflection routing 112
Fig. 5.16: Contention ratio versus system load for Shufflenet employing store-and-forward routing . 112
Fig. 5.17: Contention ratio versus system load for Gemnet employing store-and-forward routing 113
Fig. 5.18: Contention ratio versus system load for De Bruijn Graph employing store-and-forward routing 113
Fig. 5.19: Node idle ratio versus system load for Shufflenet employing deflection routing 114
Fig. 5.20: Node idle ratio versus system load for De Bruijn Graph employing deflection routing 115
Fig. 5.21: Node idle ratio versus system load for Gemnet employing deflection routing 115
Fig. 5.22: Node idle ratio versus system load for Shufflenet employing store-and-forward routing ... 116
Fig. 5.23: Node idle ratio versus system load for De Bruijn Graph employing store-and-forward routing 116
Fig. 5.24: Node idle ratio versus system load for Gemnet employing store-and-forward routing 117
Fig. 5.25: TRR versus system load for Shufflenet employing deflection routing 118
Fig. 5.26: TRR versus system load for Gemnet employing deflection routing 118
Fig. 5.27: TRR versus system load for Shufflenet employing store-and-forward routing 119
Fig. 5.28: TRR versus system load for Gemnet employing store-and-forward routing 120
Fig. 6.1: Master worker model 127
Fig. 6.2: 29 Node random network links having the same cost 129
Fig. 6.3: 36 node irregular grid 129
Fig. 6.4: Overall system design 131
Fig. 6.5: Average packet delay versus ant creation rate 29-node for original and improved antnet algorithms for low loads 135
Fig. 6.6: Average packet delay versus ant creation rate 29-node for original and improved antnet algorithms for medium loads 135
Fig. 6.7: Average packet delay versus ant creation rate 29-node for original and improved antnet algorithms for high loads 136
Fig. 6.8: Average packet delay versus ant creation rate 36-node for original and improved antnet algorithms for low loads 137
Fig. 6.9: Average packet delay versus ant creation rate 36-node for original and improved antnet algorithms for medium loads 138
Fig. 6.10: Average packet delay versus ant creation rate 36-node for original and improved antnet algorithms for high loads 138
Fig. 6.11: Average packet delay against evaporation rate for different evaporation states, low load and 29-node 144

Fig. 6.12: Average packet delay against evaporation rate for different evaporation states, medium load and 29-node 145
Fig. 6.13: Average packet delay against evaporation rate for different evaporation states, high load and 29-node 146
Fig. 6.14: Average packet delay against evaporation rate for different evaporation states, low load and 36-node 146
Fig. 6.15: Average packet delay against evaporation rate for different evaporation states, medium load and 36-node 147
Fig. 6.16: Average packet delay against evaporation rate for different evaporation states, high load and 36-node 148
Fig. 6.17: Interaction of multiple ant colonies 149
Fig. 6.18: Throughput versus time for different antnet configurations and low load 150
Fig. 6.19: Throughput versus time for different antnet configurations and medium load 151
Fig. 6.20: Throughput versus time for different antnet configurations and high load 152
Fig. 6.21: Throughput versus time for different antnet configurations and low load 152
Fig. 6.22: Throughput versus time for different antnet configurations and medium load 153
Fig. 6.23: Throughput versus time for different antnet configurations and high load 154
Fig. 6.24: Confidence parameter Y versus the average delay for different antnet configurations with low load and 29 Node 155
Fig. 6.25: Confidence parameter Y versus the average delay for different antnet configurations with medium load and 29 Node 155
Fig. 6.26: Confidence parameter Y versus the average delay for different antnet configurations with high load and 29 Node 156
Fig. 6.27: Confidence parameter Y versus the average delay for different antnet configurations with low load and 36 Node 156
Fig. 6.28: Confidence parameter Y versus the average delay for different antnet configurations with medium load and 36 Node 157
Fig. 6.29: Confidence parameter Y versus the average delay for different antnet configurations with high load and 36 Node 157

List of Tables

Table 2.1: Comparison of network properties 31
Table 3.1: Parameters used in RIP 43
Table 3.2: Comparison of the routing protocols 49
Table 5.1: Test environment 105
Table 5.2: Average number of hops comparison for three networks and for network size of 64 110
Table 5.3: Contention ratio comparison for three networks and for network size of 64 114
Table 5.4: Node idleness ratio comparison for three networks and for network size of 64 117
Table 5.5: TRR comparison for three networks and for network size of 64 121
Table 5.6: Evaluation of the routing algorithms and topologies 122
Table 6.1: Overall average delay comparison for different system loads 158
Table 6.2: Overall average delay comparison for different system loads 159
Table 6.3: Overall throughput comparison for different system loads 159
Table 6.4: Overall throughput comparison for different system loads 159

Table of Contents

ACKNOWLEDGEMENTS i

Declaration ii

Abstract iii

Glossary of Abbreviations v

Glossary of Symbols vii

List of Figures xi

List of Tables xiv

List of Tables xiv

Table of Contents xv

CHAPTER I - Introduction 1

1.1 Organisation of Thesis 4

1.2 Aims and Objectives 6

1.3 Original Contributions 8

1.4 Publications 9

Chapter II - Review of the Logical Network Topologies 11

2.1 Introduction 11

2.2 Logical Topologies 11

2.2.1 Shufflenet 13

2.2.3 De Bruijn Graph 15

2.2.2 Gemnet 18

2.2.4 Kautz Graph 20

2.2.5 Manhattan Street Network 22

2.2.5.1 Distributed routing rules 26

2.2.6 Hypercube 27

2.2.7 Hypercube connected rings NETwork 29

2.3 Comparison of the Networks 31

2.4 Conclusions 32

Chapter III - Traditional Routing Algorithms and Protocols 33

3.1 Introduction 33

3.2 Routing Problem 33

3.3 Routing Algorithms 37

3.3.1 Dijkstra / Shortest Path Algorithm 37

3.3.2 Link State Routing Algorithm 39
3.3.3 Distance Vector Routing/ Bellman Ford Routing Algorithm 40
3.4 Routing Protocols 42
3.4.1 Routing information protocol 42
3.4.2 Interior gateway routing protocol 43
3.4.3 Open Shortest Path First 45
3.4.4 Enhanced interior gateway routing protocol 46
3.4.5 Border gateway protocol 47
3.4.6 Exterior gateway protocol 48
3.5 Comparison 48
3.6 Conclusions 50

Chapter IV - Reinforcement Learning and Ant Colony Optimisation 51

4.1 Introduction 51
4.2 Reinforcement Learning 53
4.3 Swarm Intelligence 55
4.4 Ants and Ant Colony Optimisation 56
4.5 Antnet routing algorithm and its derivatives 60
4.5.1 Mobile software agents for control in telecommunications networks 60
4.5.2 ABC routing in telecommunications networks 61
4.5.3 Regular and uniform ant routing algorithm 65
4.5.4 Cooperative Asymmetric Forward routing algorithm 65
4.5.5 Smart ants 66
4.5.6 Antnet routing algorithm 67
4.5.7 Explorer, allocator and deallocator ants 74
4.5.8 Ant System Genetic Algorithm 74
4.5.9 Improved Antnet Routing Algorithm 75
4.5.10 Limiting number of ants 76
4.5.11 Multiple ant colonies 77
4.5.12 Other Ant-Agent Based Approaches 79
4.6 Agent Distance Vector Routing 80
4.7 Q-Learning 81
4.7.1 Q-routing 82
4.7.2 Dual reinforcement Q-routing 83
4.7.3 Confidence based dual reinforcement Q-routing 84
4.7.4 Predictive Q-routing 85
4.8 Conclusions 87

Chapter V - Implementation and Evaluation of Logical Network Topologies 88

5.1 Introduction 88
5.2 Testing Criteria 89

5.3 Transmission/Reflection Ratio 91
5.4 Design 94
5.4.1 Packet format 95
5.4.2 Network routing algorithm framework 95
5.5 Implementation 99
5.6 Test Environment and Experiment Model 105
5.7 Tests and Network Comparisons 106
5.7.1 Introduction 106
5.7.2 Test 1 - Average number of hops 106
5.7.3 Test 2 - Contention 110
5.7.4 Test 3 – Node idleness percentage 114
5.7.5 Test 4 – TRR 117
5.8 Comparison 121
5.9 Conclusions 123

Chapter VI - Improved Antnet Routing Algorithm 125

6.1 Introduction 125
6.2 Parallel Virtual Machine Model 125
6.2.1 The master worker model 126
6.3 Assumptions Made 127
6.3.1 Traffic models 130
6.4 Overall System Design 130
6.5 Antnet Modifications 132
6.5.1 Deleting aged packets 133
6.5.2 Limiting the effect of r due to traffic fluctuations 133
6.5.3 Further comments on improved antnet 139
6.5.4 Limiting the number of ants 140
6.6 Stagnation Problem 140
6.6.1 Evaporation 141
6.6.2 Multiple ant colonies 148
6.7 Effect of Y on Antnet 154
6.8 Comparison 158
6.9 Conclusions 160

Chapter VII - Conclusion and Furher Work 162

7.1 Conclusions 162
7.2 Further work 166

References 169

CHAPTER I - INTRODUCTION

In today's fast growing Internet, traffic conditions changes and failures occur at some parts of the network occasionally, in an unpredictable manner. Therefore, there is a need for an algorithm to manage traffic flows and deliver packets from the source to the destination in a realistic time. The routing algorithm is the key element in networks performance and reliability, and can be seen as the "brain" of the network that manages the traffic flow through the network. An ideal routing algorithm should be node and link independent, and be able to deliver packets to their destination with the minimum amount of delay, regardless of the network size and the traffic load. The only way to achieve this would be by employing an intelligent and distributed routing algorithm. The routing algorithms currently in use lack intelligence, and need human assistance and interpretation in order to adapt themselves to failures and changes within the network. Routing is considered to be an NP-Hard Optimization problem, therefore widely used optimisation methods have been applied in the literature: to name a few, Genetic Algorithms [Liang et al, 02], Neural Networks [Dixon et al, 95], Simulated Annealing [Osman & Kelly, 96], Software Agents [Yang et al, 02] [and Reinforcement Learning [Littman & Boyan, 92].

In recent years, agent based systems and reinforcement learning have attracted researchers' interest. This is because these methods do not need any supervision and are distributed in nature. Swarm intelligence, particularly ant based systems [Dorigo et al, 02-2], Q-learning methods [Boyan & Littman, 94] and hybrid agent based Distance Vector algorithms [Amin et al, 04] have shown promising and encouraging

results. The ant-based approach applied to the routing problem was first reported in [Schoonderwoerd, 96], which itself was influenced by the work done in [Appleby & Steward, 94] on the software agents used for control in telecommunication networks. An improved version of the algorithm reported in [Schoonderwoerd, 96] was applied to connection-oriented systems [Bonabeau et al, 98]. In [Subramanian et al, 97] for the first time ant based routing was applied to the packet based connection-less systems. In [Di Caro & Dorigo, 97] the idea was applied to application of mobile agents in adaptive routing, namely antnet, which is also used as the basis in this work. In real life, ants deposit some kind of chemical substance called a pheromone to mark the path that they used. On their way back they choose the path with the most pheromones, thus the shortest path. Ants (nothing but software agents) in antnet are used to collect traffic information and to update the probabilistic distance vector routing table entries.

Although ant based routing algorithms have shown some interesting results, they are still far away from being ideal. Tests carried out in this work have shown that by detecting and dropping a few of the packets, routed through the non-optimal routes, the average delay per packet is decreased and network throughput is increased. In addition, one of the major problems with antnet approach is the stagnation, where the network freezes and consequently routing algorithm gets trapped in the local optima and therefore is unable to find new improved paths [Sim & Sun, 03]. A number of methods have been used to overcome this problem, such as: evaporation, ageing, pheromone control and hybrid algorithms [Dorigo & Stutzle, 04]. However, most of these methods, except for evaporation, are rather complex or not efficient and requires other heuristics such as genetic algorithms (GA) [White et al, 98]. Evaporation of link

is a simple method which adds negligible overhead to the node itself. Here the focus is on the antnet routing algorithm. In the original antnet algorithm the only feedback provided to the system by the software agents is the trip time observed on their way from source to destination. This feedback signal is reinforced by updating the related probability entry in the distance vector table. In this work link usage information is used as the second feedback to the system in order to prevent stagnation problems. For every link, the usage information is held at every node based on a predefined time window. Routing table entries are then reinforced with a negative signal based on the calculated evaporation information based on the link usage statistics. In addition, multiple colonies of ants used in antnet for packet switched networks also improve the performance of the algorithm compared to the original approach. However, this approach has improved node and algorithm complexity as separate data variables are kept for each colony.

The challenge is how to implement and simulate these algorithms. For a large asynchronous network size the task becomes both complicated and time consuming. Thus, researchers and network designers have used a simplified network model and a large number of assumptions to overcome the problem. Communication networks are usually simulated by discrete event simulation and analytical modelling [Banks et al, 00], [Ross, 02]. However, there is an elegant alternative simulation environment, which is based on supercomputers and network of workstations with the ability to reduce simulation time from days into hours for a complex network configuration. A supercomputer consists of thousands of high performance processing elements, with a large amount of memory, connected together with high speed interconnect resembling a small scale local area network connected to each other as in the Internet. There are

various simulation tools available for researchers. However, running big scaled network simulations is very time consuming. Therefore in this thesis a new way of simulating networking processes will also be proposed.

1.1 Organisation of Thesis

The thesis is divided into seven chapters. Following the introduction, a literature review of the widely used topologies and their routing algorithms are given in Chapter two. This chapter gives a brief description of the most commonly used logical interconnection networks and their routing algorithms. The term logical network topology describes the way packets travel within the network independently from the physical interconnection of the devices within the network. By using suitable topologies, network performance can be improved and resources can be used efficiently. Finally in chapter two, a comparison of the reviewed topologies is given and their advantages and disadvantages are discussed.

The logical network topologies and the routing algorithms reviewed in chapter two can only be used within routers or static networks. Although they are ideal for optical networks, due to simple routing algorithms used there is a growing demand for flexible algorithms that can be used in arbitrary network topologies. In Chapter 3, a review of the traditional routing algorithms together with routing protocols that are currently in use today's networks have been given. Dijkastra's shortest path algorithm is one of the well known algorithms which is widely adapted in the internet and studied in the literature. This algorithm is also the base for the link state routing algorithm and related protocols. Open shortest path first (OSPF) is deployed

throughout the internet and is the de facto routing algorithm within the networks. Exterior gateway protocol (EGP) is a gateway protocol which is mainly used between the backbones and as its name suggests, among the gateways. However, nowadays most widely used gateway protocol is border gateway protocol (BGP).

The traditional routing algorithms lack intelligence, and need human assistance and interpretation in order to adapt themselves to failures and changes. Therefore Chapter 4 analyses the ant colony optimisation and gives a review of the routing algorithms based on the ant colony optimisation. After providing a brief description of reinforcement learning, the ant colony optimisation is analysed and the antnet routing algorithm together with the algorithms influenced from it reviewed in detail. After this, Chapter 5 and 6 provides practical results and improvements proposed to the existing algorithms.

Chapter 5 gives a study of a variety of network performance issues, design, implementation and evaluation for various synchronous logical network topologies and their routing algorithms. Particularly, routing algorithms for these networks are simulated and tested as in a fully connected network. In order to test the algorithms a generic implementation framework has been developed for synchronous logical network topologies. A priority rule for the chosen algorithms has been introduced in order to reduce the amount of crosstalk experienced per packet in an optical network.

Chapter 6 outlines the weaknesses associated with the antnet routing algorithm and proposes solutions for overcoming them. The antnet routing algorithm is also

discussed and evaluated in a different platform using parallel programming paradigms. This novel approach provides a more realistic implementation of the algorithms and flexibility to the programmer. Therefore, this chapter starts with an introduction to the programming environment and model. This is followed by test criteria and the traffic model used together with all the assumptions made. Finally, novel implementations of antnet routing algorithm are presented with some results and discussion. Results show that, the performance of the original antnet algorithm has been improved up to 30%.

Finally, conclusions and suggestions for further work are discussed in Chapter 7.

1.2 Aims and Objectives

The fundamental aim of the work presented in this thesis is to design, model, introduce novel improvements and investigate the performance of routing algorithms that are suitable for use either for logical network topologies or arbitrary networks within packet switched networks. In order to achieve this aim the following research milestones were identified:

- Design an implementation model to simulate synchronous packet switched networks and simulate the well known logical network topologies and their routing algorithms.
- Investigate the suitability of these topologies and algorithms to Optical time domain multiplexing (OTDM) systems and behaviour of the routing algorithm against several performance parameters such as packet delay, contention ratio and node idleness level.

- Model logical interconnection networks with the requirements of the 2x2 terahertz optical asymmetric demultiplexer (TOAD) Router designed in our research group. Improve the performance of the TOAD based router.
- Evaluate routing algorithm performance of the logical networks and their routing algorithms.
- Design and develop a new simulation model to implement and simulate the distributed routing algorithms for asynchronous packet switched networks suitable for arbitrary networks.
- Investigate the traditional routing algorithms that are in use in today's networks.
- Investigate the adaptive and intelligent routing algorithms, particularly reinforcement learning based algorithms suitable to packet switched networks.
- Extend the work on the antnet routing algorithm to improve its performance in terms of network throughput and packet delay and improve its adaptability to different traffic models and changing environments.
- Investigate antnet's performance with respect to number of novel improvements proposed.

1.3 Original Contributions

During this work the following original contributions have been made:

- A new and novel way of implementing synchronous routing algorithms for logical network topologies has been proposed (Section 5.5).
- Simple priority rule for reducing the amount of noise experienced per packet to use with 2x2 TOAD based router has been introduced (Section 5.3).
- A novel way of implementing distributed routing algorithms on clusters and workstations and supercomputers is proposed (Section 6.4). This work has also been submitted to the engineering and physical sciences research council (EPSRC) to investigate this further on using the supercomputers available via national supercomputing services.
- Improvements to the antnet routing algorithm have been proposed and tested to overcome the weaknesses spotted during the simulations run on the original version of the algorithm (Section 6.5).
- A novel evaporation technique is used in order to overcome the stagnation problem that is experienced by the antnet routing algorithm (Section 6.6.1).
- For the first time in the literature, multiple ant colonies are applied to packet switching networks in order to the overcome the stagnation problem (Section 6.6.2).

- The effect of the Confidence parameter Y on Antnet has been investigated for various traffic models and network configurations for the first time (Section 6.7).

1.4 Publications

The work in this thesis has resulted in the following publications, which are listed in reverse chronological order.

1. F. Tekiner and Z. Ghassemlooy "Improved Antnet Routing Algorithm for Packet Switching", Mediterranean Journal of Computer Networks, Oct 05, Vol 1, No.2, pp. 69-76.

2. F. Tekiner, Z. Ghassemlooy and S. Al-khayatt, "Improved Antnet Routing Algorithm with Link Probability Evaporation Over the Given Time Window", in Proceedings of SOFTCOM04, Split/Venice, 11-13 Oct 2004, pp. 502-505.

3. F. Tekiner, Z. Ghassemlooy and S. Al-khayatt, "Investigation of Antnet Routing Algorithm by Employing Multiple Ant Colonies for Packet Wwitched Networks to Overcome the Stagnation Problem", Proceedings of LCS04, London 13-15 Sep 2004, pp. 185-188.

4. F. Tekiner, Z. Ghassemlooy and S. Al-khayatt, "Antnet Routing Algorithm-Improved Version", in: Proceedings of CSNDSP04, Newcastle, UK, 22-22 July 2004, pp. 416-419.

5. F. Tekiner, Z. Ghassemlooy and T. R Srikanth, "Comparison of the Q-routing and Shortest Path Routing Algorithms" PGNET04, 28 - 29 June 2004, Liverpool, UK, pp. 428 - 432.

6. F. Tekiner, Z. Ghassemlooy and S. Al-khayatt, "Implementation and Evaluation of Shufflenet, Gemnet and De Bruijn Graph Logical Network Topologies", IASTED 2004, Innsbruck Austria. 16-19 Feb 2004, pp. 582 – 587.

7. F. Tekiner, Z. Ghassemlooy and S. Al-khayatt, "Packet Loss Analysis of Output Buffered Shufflenet", 16-17 June 2003, PGNET 2003, ISBN: 1902560094.

8. F. Tekiner, Z. Ghassemlooy and S. Al-khayatt, M. Thompson, "Prioritized Routing in Logical Network Topologies", Set for Europe Young Engineers 2002, House of Commons Poster Presentation, 9 December 2002.

9. F. Tekiner, Z. Ghassemlooy, S. Al-khayatt and M. Thompson, "Prioritized Shufflenet Routing in TOAD based 2X2 OTDM Router", Senacitel 2002, Valdivia, Chile, 13-16 November 2002.

CHAPTER II - REVIEW OF THE LOGICAL NETWORK TOPOLOGIES

2.1 Introduction

This chapter gives a brief description of the most commonly used logical interconnection networks and their routing algorithms. Logical network topology describes the way packets travel within the network independently from the physical interconnection of the devices within the network [Acampora et al, 88]. By using suitable topologies, network performance can be improved and resources can be used efficiently. The chapter commences with an introduction to the logical topologies and explains why they are needed. This is followed by the review of the widely used topologies and their routing algorithms. Finally, a comparison of the reviewed topologies is given and their advantages and disadvantages are discussed.

2.2 Logical Topologies

There are four major issues when selecting and designing a network: switching method, network operation mode, control strategy and network topology with its routing algorithm [Feng, 81]. As for switching, there are two main techniques, circuit switching and packet switching [Qiao & Yoo, 99], [Tanenbaum, 03]. The former switches in the order of mili-seconds, whereas the latter switches in the order of microseconds. Packet switching is widely used in today's networks, which provide high bandwidth utilisation by means of dynamically allocating bandwidth on demand [Qiao & Yoo, 99]. Operation mode can be either asynchronous or synchronous. In the

synchronous mode, packets are sent when a time slot is available, whereas in asynchronous networks, packets are sent when a link is available. A typical network consists of several nodes (switching elements) and interconnection links which could be controlled either by distributed nodes or by a centralised node(s). In distributed control on receiving a packet, routing decisions are made at each node independently, whereas in the centralised control, a routing decision is made by a central processor(s) and is based on the current situation of the network.

The network topology outlines how the nodes are connected to each other. Although the single-hop network is the most basic, it requires a pre-transmission phase and fast tuning transceivers (as in wavelength division multiplexing (WDM) [Mukherjee, 92-1]). A multihop network [Mukherjee, 92-2], regular or arbitrary [Yao et al, 01], overcomes these drawbacks by finding the path at every intermediate node rather than defining the path at the start. A network topology is regular if node connections are based on a specific connectivity pattern which is also called an interconnection network. However, in the arbitrary network topologies, there is no fixed predefined pattern as its name suggests and they can be dynamic and scalable. The latter characteristics facilitate new nodes being added or removed without making any changes in the network structure as in the case of the Internet. On the other hand, regular network topologies, with well defined connectivity patterns, are the alternative option for more static environments (such as router and multiprocessor architectures, etc.) as they provide simpler node structures and routing schemes. Therefore, interconnection networks can either be used as an underlying physical topology itself as in multiprocessor parallel networks [Grammatikakis et al, 98], or as a logical topology based on the underlying physical topology. The idea of imposing a logical

topology for the broadcast networks (such as star, tree and bus) was first introduced in [Acampora et al, 88] to maintain and improve the performance of the network. There are a number of logical network topologies that have been applied for multihop networks [Baransel et al, 95]. In multihop packet-routing networks, packets traverse a number of intermediate nodes until they arrive at their destination [Banerjee et al, 99]. At each node, routing decisions are carried out by simply processing only a small fraction of the information contained within the packet header. The routing algorithm used at each node is identical and a decision is made at every node to find the best possible port allocation to the incoming packets. This decision is made independently, with no regard for the routing decisions made at the rest of the nodes within the network. However, each network topology has its own specific routing algorithms that are much simpler than the routing algorithms used in the arbitrary networks, where in most cases routing tables are used [Tanenbaum, 03]. In this chapter, the most commonly used logical network topologies will be reviewed together with their routing algorithms. Implementation and results for the selected networks will be given in the next chapter.

2.2.1 Shufflenet

The shufflenet is classified as a regular multihop network topology that is symmetric and homogeneous, i.e. the network looks the same from any node [Banerjee et al, 99]. Thus, the number of hops from a source to a destination is minimised. Due to its cylindrical connectivity pattern (see Fig. 2.1) routing is reliable and fault tolerant. Moreover, deflection routing can be applied straightforwardly with promising results.

A (p,k) Shufflenet consists of $N = kp^k$ (k = 1, 2, 3,... and p = 1, 2, 3,...) nodes arranged in k columns of p^k nodes each and with the last (k-th) and first columns connected together cylindrically where p is in and out degree of the nodes. Fig. 2.1 shows a 24-node Shufflenet configuration. In general, node (row, column) is connected to nodes (p.row mod p^k, (column + 1) mod k), (p.row mod p^k + 1, (column + 1) mod k),... (p.row mod p^k + p, (column + 1) mod k.

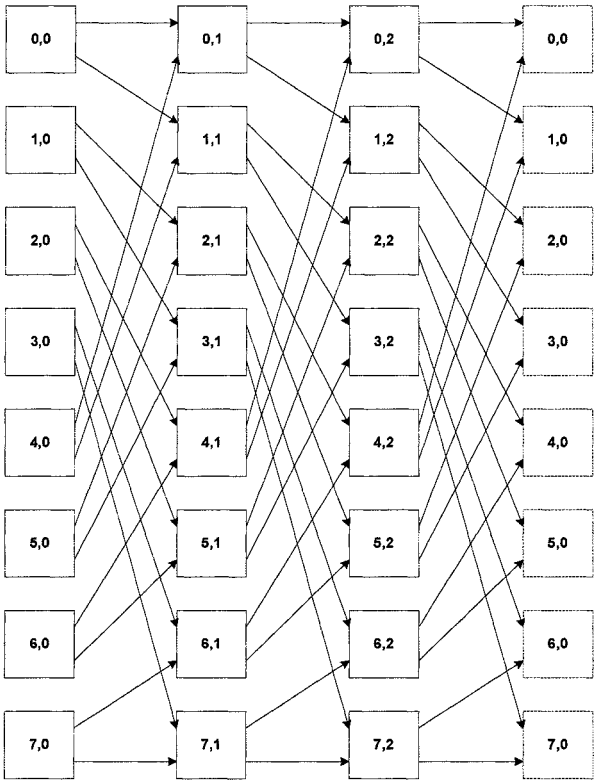

Fig. 2.1: A 24-node (p = 2, k = 3) Shufflenet topology.

In simple static Shufflenet routing, there is only one path from source to destination, which is fixed [Acampora & Karol, 89] [Karol & Shaikh, 88]. However, if a packet cannot reach its destination in one pass (i.e. it is k-hops away) then it will take one of the many minimum hop routing paths available. A simple adaptive algorithm exploits this multiplicity of potential routing paths, by routing packets via less-congested paths whenever possible [Karol & Shaikh, 88]. Generally there are two kinds of packets

defined in adaptive routing: **TYPE-S** packets that have a single minimum hop path from the current location to the destination and **TYPE-M** packets that have multiple minimum hop paths from the current location to the destination [Karol & Shaikh, 88]. **TYPE-M** packets have no preference on which output port they should go, since there are more paths available. The pseudo code for the routing algorithm is given in Fig. 2.2 [Karol & Shaikh, 88].

Find *distance(X) between source(s) node* (r^s, c^s) *and destination(d) node* (r^d, c^d)

$$X = \begin{cases} (k + c^d + c^s) \bmod k & c^d \neq c^s \\ k & c^d = c^s \end{cases}$$

where r and c is the column and row coordinates ($r = 0,1,2,...p^{k-1}$ *and* $c = 0,1,2,...k-1$)

Define *Packet Type (S or M)*

If $r^s_{k-1-X} = r^s_{k-1}, r^s_{k-2-X} = r^s_{k-2}, ..., and\ r^s_0 = r^s_X$

Then *Packet Type is S*

Else *Packet Type is M*

If *Packet Type is M*

Then *forward packet to any of the available port*

Else *Routing code* = $r^s_{k-2}, r^s_{k-3}, ..., r^s_2, r^s_1, r^s_0, r^d_{X-1}, (c^s + 1) \bmod k$

Fig. 2.2: Pseudo code for Shufflenet routing algorithm

2.2.3 De Bruijn Graph

De Bruijn graphs can support much larger numbers of nodes than the same degree Shufflenets by having the same average number of hops [Sivarajan & Ramaswami, 91]. Moreover, it retains the simple addressing and routing properties of the Shufflenets.

A $G(\Delta,D)$ is a directed De Bruijn graph which consist of $N = \Delta^D$ nodes with the set of $\{0,1,2,\ldots\ \Delta\text{-}1\}^D$ nodes where there is an edge from node $(a_1, a_2, \ldots. a_D)$ to node $(b_1, b_{2,\ldots.}\ b_D)$ if and only if equation 2.1 is satisfied.

$$b_i = a_i + 1,\ a_i, b_i \in \{0, 1, 2,\ldots\ \Delta\text{-}1\} \qquad , 1 < i\text{-}D\text{-}1 \qquad (2.1)$$

Where D is the diameter of the De Bruijn graph with a deflection penalty D as the packets has to make a loop back (since there is only single path that exists in the De Bruijn Graphs for a pair of node). Each node has in and out degree of Δ, and Δ nodes (i.e. 000,111) have self-loops. Self-loops only exists in the logical graph but not in the physical network configuration. The De Bruijn graph structure is inherently asymmetric due to the nodes with self-loops (see Fig. 2.3) [Sivarajan & Ramaswami, 94].

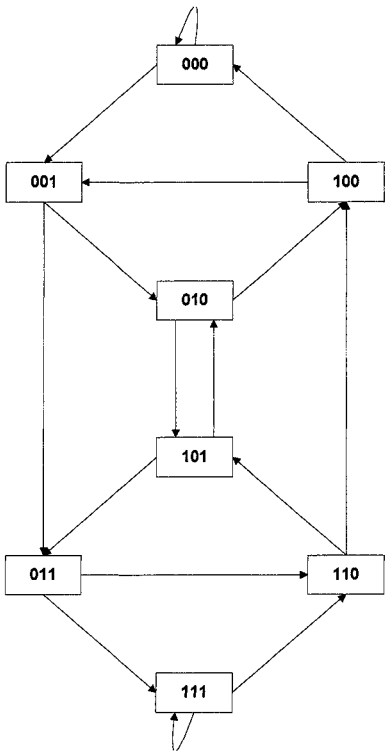

Fig. 2.3: An 8 Node ($\Delta = 2$, $D = 3$) De Bruijn graph

There is one-to-one correspondence between the connectivity of the nodes in the De Bruijn graph $G(\Delta, D)$ with all the possible states of a Δ shift register of length D. If state b can be reached from state a in one shift operation in the shift register then there is an edge from node a to b. Therefore, the De Bruijn graph can be seen as the state transition diagram of a shift register [Golomb, 82]. A node in the De Bruijn graph can be represented by a sequence of D digits as defined in the shift register analogy [Golomb, 82]. An edge from node A to node B can be represented by a string of $D + 1$ digit(s). Consequently, any path in the graph of length k from source to destination nodes can be represented by a string $D + k$ digits. The first D digits represent the source node and the last D digits represent the destination node.

To find routing code from source to destination, the algorithm determines the longest suffix[1] of the source which appears as a prefix of the destination. In order to find this, two operations, **shift-match (i, A, B)** and **merge (i, A, B)**, based on the shift register analogy are defined for De Bruijn graphs as follows:

Given two strings $A = (a_1, a_2, \ldots a_D)$ and $B = (b_1, b_2, \ldots b_D)$, **shift-match (i, A, B)** operation is **true** *if and only if:*

$$A = (a_{i+1}, a_{i+2}, \ldots a_{i+D}) \text{ to node } B = (b_1, b_2, \ldots b_{D-i}) \qquad (2.2)$$

[1] Suffix or prefix of a string is what is obtained by removing number of symbols from the end or beginning of the word.

A string (or a sequence) of length $\boldsymbol{D + i}$ given by $(a_1, ..., a_D, b_{D-i+1}, ..., b_D)$ is defined as **merge (i, A, B)**, where $0 \leq i \leq D$. Note that i also gives the hop distance (shortest number of hops) between nodes A and B. Pseudo code for the routing algorithm is given in Fig. 2.4 [Sivarajan & Ramaswami, 94].

```
Find i between nodes A and B by using shift-match (i, A, B) operation
i = 0;
while (shift-match (i, A, B) is False)
        i = i++;
end while
Routing code (shortest path) between nodes A and B is given by merge operation (i, A, B)
Routing code = merge (i, A, B)
```

Fig. 2.4: Pseudo code for De Bruijn graph routing algorithm

2.2.2 Gemnet

A (k,m',p) Gemnet consists of $N = k$ x m' (k = 1, 2, 3..., and m' = 1, 2, 3) nodes arranged in k columns of m' nodes each and with the last (k-th) and first columns connected together cylindrically where p is the in-and-out degree of the nodes [Iness et al, 95] [Jaekel et al, 98]. Fig. 2.5 shows a 10 nodes Gemnet configuration. In general, node (row, column) is connected to nodes (c', [$r' + i$] mod m') where i = 0, 1, 2,...p-1. c' and r' are defined in equations 2.3 and 2.4:

$$c' = (c+1) \bmod k ; \tag{2.3}$$

$$r' = r \text{ x } p. \tag{2.4}$$

The diameter (maximum hop distance between any two nodes) is given as [Jaekel et al, 98]:

$$\lceil \log_p m' \rceil + k\text{-}1. \qquad (2.5)$$

The major advantage of the Gemnet compared to the Shufflenet and the De Bruijn graph (covered in the next section) is it is they are not bound by the network size. Hence, it is scalable, but not perfectly symmetric. It is only symmetric if the k and m' values match the Shufflenets p and kp^k values [Iness et al, 95] [Banerjee et al, 99]. Thus, Gemnet becomes a Shufflenet.

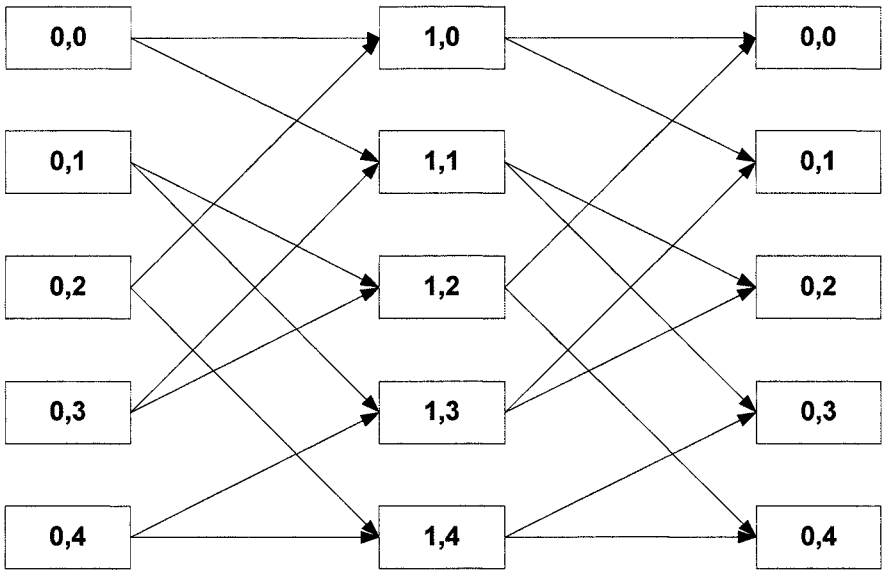

Fig. 2.5: A 10 node ($k = 2$, $m' = 5$, $p = 2$) Gemnet topology

In Gemnet, routing is simple and adaptive since there exist multiple paths. The pseudo codes for the routing algorithm is given in Fig. 2.6 [Jaekel et al, 98].. The route code R' which specifies the shortest path from source (c^s, r^s) to destination (c^d, r^d) is expressed as a sequence of h base-p digits,

$R' = [a_1 a_2 ... a_h]_p$, i^{th} node will route the packet on its a_i outgoing link[2].

[2] Each outgoing link is numbered individually from 0 to p for each node on Gemnet.

Find *distance (D) between nodes* (c^s,r^s) *and* (c^d,r^d)

$D = [(k+c^d)-c^s] \bmod k$

Define *Packet Type (S or M)*

Let Number of alternative shortest paths $S = \left\lceil \frac{(p^h - r)}{m} \right\rceil$

If $S > 1$

Then *Packet Type is M*

Else *Packet Type is S*

If *Packet Type is M*

Then *forward packet to any of the available port*

Else (if packet type is S) *Routing code*$(R') = [m + r^d - ((r^s x p^h) \bmod m')] \bmod m'$

Where, h is the smallest integer = $(D + ik)$, and $i = 0, 1, 2, \ldots$,.

Fig. 2.6: Pseudo code for Gemnet routing algorithm

2.2.4 Kautz Graph

A Kautz graph is a digraph that uses exactly the same routing algorithm as in the De Bruijn graph however it accommodates more nodes than the De Bruijn graph with same degree and diameter.

A (d,k) Kautz graph is a directed digraph, where d is the in and out degree and k is the diameter of the graph with deflection penalty k, Fig. 2.7 [Panchapakesan & Sengupta, 95].

In Kautz graph each node is labelled with strings of length $k(a_1, a_2, \ldots a_k)$ from an alphabet $d + 1$ letters that do not contain two consecutive same letters. Formally speaking, $a_i \neq a_{i+1}$, where $1 \leq i \leq k-1$. A (d,k) Kautz graph consists of $N = d^k + d^{k-1}$ nodes and $E = Nd$ edges where there is an edge from node $(a_1, a_2, \ldots a_k)$ to node $(b_1, b_2, \ldots b_k)$ if and only if

$$b_i = a_i + 1,\ a_i,\ b_i \in \{0, 1, 2, \ldots d\text{-}1\},\ 1 < i\text{-}k\text{-}1 \qquad (2.6)$$

The Kautz graph structure is perfectly symmetric unlike De Bruijn graph as it does not have nodes with self loops.

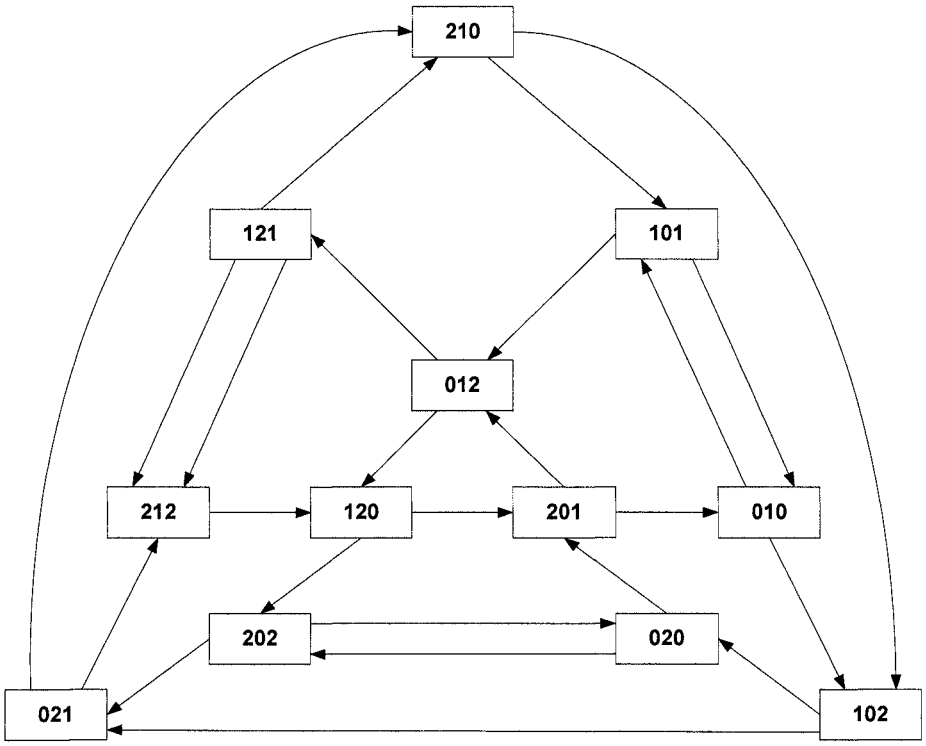

Fig. 2.7: Kautz graph

A lower bound to the average hop distance (h) for a packet to travel between two arbitrary nodes is given by [Panchapakesan & Sengupta, 95] in:

$$h = p + \frac{k-p}{N(N-1)} - \frac{(p-1)E}{N(N-1)} - \frac{Ed-2C}{N(N-1)}\left[\frac{d^{p-1}-d}{(d-1)^2} - \frac{p-2}{d-1}\right] \qquad (2.7)$$

where C is the number of unordered node pairs forming a cycle length of two and in the case of Kautz graph it is defined as in equation 2.8:

$$C = \frac{d(d+1)}{2} \tag{2.8}$$

and p is the largest integer that satisfies the following equation:

$$\frac{N(N-1)-E}{Ed-2C} \geq 1 + d + d^2 + \ldots + d^{p-3} \tag{2.9}$$

If $k \leq 2$, then p does not exist and therfore it is assumed that $p = k$.

The routing algorithm used in Kautz graph is exactly the same algorithm that is used in the De Bruijn graph as both of the networks are based on the shift register analogy.

2.2.5 Manhattan Street Network

The Manhattan street network (MSN) is a two connected mesh structured network. The bidirectional links in MSN are arranged in a structure that it resembles the streets of Manhattan [Maxemchuck, 87]. [Bassil & Cruz, 96] studied non-uniform traffic models for MSN.

A (m,n) MSN consists of $N = mn$ (m = 1,2,3..., and m = 1,2,3) nodes arranged in m columns of n nodes each connected together with adjacent row links and column links alternating in direction where in and out degree of nodes is always 2. Fig. 2.8 shows a

64 nodes MSN configuration. MSN is similar to the slotted loop system and packet based network [Maxemchuck, 89].

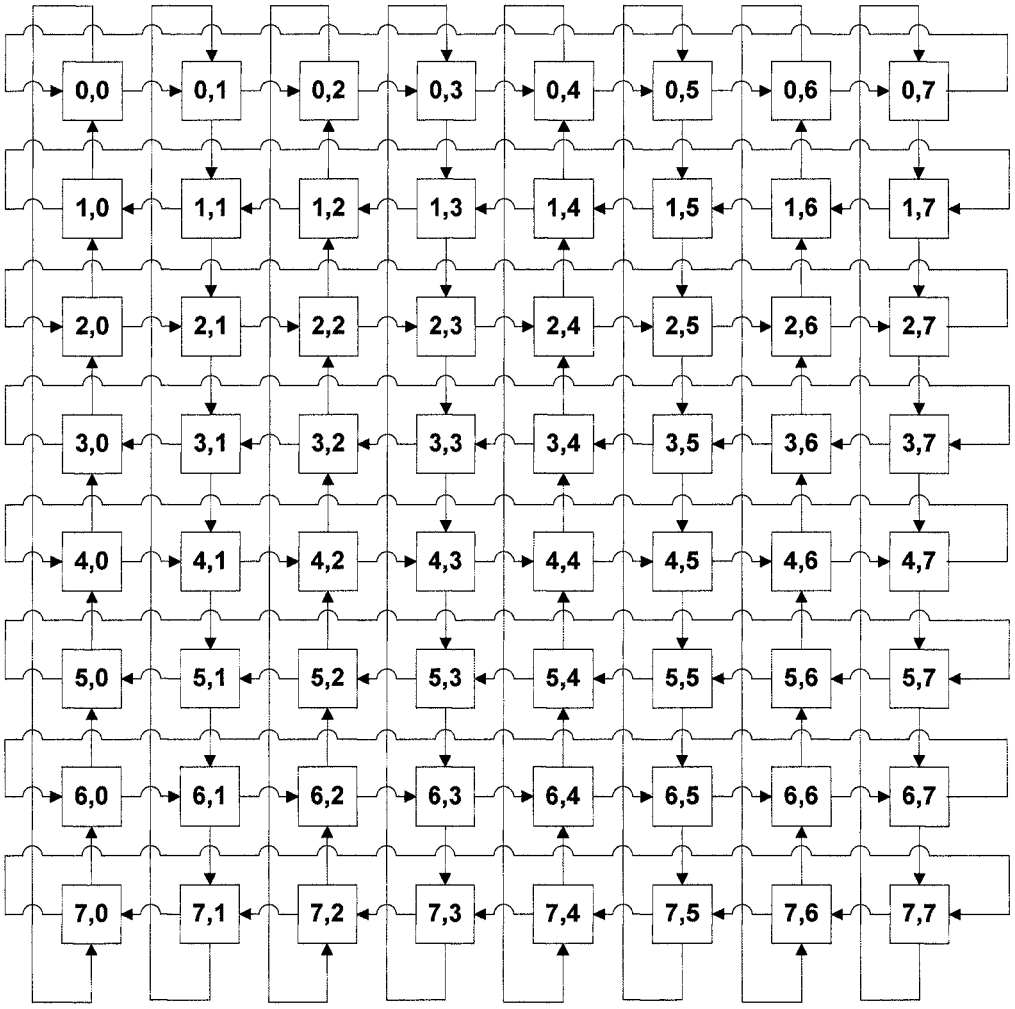

Fig. 2.8: A 64 node (m = 8, n = 8) MSN

The diameter (maximum hop distance between any two nodes) is n when $\frac{n}{2}$ is odd and $n+1$ when $\frac{n}{2}$ is even. Deflection penalty is always 4 whatever the network size is. A packet makes a loop of 4 hops when a it is deflected and returns back to the node that it has been deflected. Thus, MSN is considered as one of the most suitable networks for deflection routing as it has a low and constant deflection penalty.

The average number of hops h required for a packet to travel between the arbitrary nodes in a (m, n) MSN when $m = n$ is $\sqrt{N}$, whereas when $m \neq n$, h is defined as:

$$h = \frac{\left\lfloor \frac{N}{4}(m+n+4) - n - 4 \right\rfloor}{N-1}, \text{ where } \frac{m}{2} \text{ is even and } \frac{n}{2} \text{ is odd,} \quad (2.10)$$

$$h = \frac{\left\lfloor \frac{N}{4}(m+n+4) - n \right\rfloor}{N-1}, \text{ where } \frac{m}{2} \text{ is even and } \frac{n}{2} \text{ is even,} \quad (2.11)$$

$$h = \frac{\left\lfloor \frac{N}{4}(m+n+4) - m - n - 4 \right\rfloor}{N-1}, \text{ where } \frac{m}{2} \text{ is odd and } \frac{n}{2} \text{ is odd} \quad (2.12)$$

In MSN, routing decisions must be made at every node independently from the other nodes. Moreover, in MSN if network size is known, shortest path from source to destination is known. Labelling of the nodes (addressing) plays important role in MSN's routing. Each node has a unique "absolute address" which is defined as (row, column) (see the labelling in the Fig. 2.8). However, routing only depends on the relative address not on the absolute address. Relative address is the relative position from source to destination, which is enabled using the same routing algorithm in every node. In original MSN's configuration [Maxemchuck, 87] there are two types of relative addressing, integer and fractional addressing.

The relative address (r,c) in an m x n integer addressed network given as:

$$r = \frac{m}{2} - \left\lfloor \left(\frac{m}{2} - D_c(\mathrm{r}_{source} - \mathrm{r}_{destination})\right) \bmod m \right\rfloor \quad (2.13)$$

$$c = \frac{n}{2} - \left[(\frac{n}{2} - D_r(c_{source} - c_{destination})) \bmod n \right] \quad (2.14)$$

The relative address (r,c) in an m x n fractionally addressed network given as:

$$r = 1 - [(1 - D_c(r_{source} - r_{destination})) \bmod 2] \quad (2.15)$$

$$c = 1 - [(1 - D_r(c_{source} - c_{destination})) \bmod 2] \quad (2.16)$$

Column direction[3] D_c and row direction D_r are dependent on the direction of the links at the destination node, which are defined as follows,

D_c *is* +1 *when* $c_{destination}$ *is even*

D_c *is* -1 *when* $c_{destination}$ *is odd*

D_r *is* +1 *when* $r_{destination}$ *is even*

D_r *is* +1 *when* $r_{destination}$ *is odd*

Fractional addressing has two main advantages over integer addressing. In the former, it is easy to expand the network as new rows and columns can be added to the system without changing the addresses of the existing nodes. Also, the routing rules used are independent from the size of the MSN (independent from the number of rows and columns). Therefore, in an integer based network, the arithmetic logic unit (or the routing algorithm) which calculates the relative address, must be changed whenever the size of network changes. On the other hand, there is no need to do any changes in the fractionally addressed network.

[3] Direction represents the way the packet travels within the network. i.e. when D_c = -1 packet travels dowanwards and when it is +1 packet travels upwards direction. when D_r = -1 packet travels from right to left and when it is +1 packet travels from left to right.

2.2.5.1 Distributed routing rules

Rule1 finds the shortest path for the integer addressed MSN, whereas **Rule2** and **Rule3** reduce the number of calculations performed at each node. However, this results in longer paths in some cases. **Rule1** works as follows:

Select the preferred path if there is one, otherwise select either path (if there is zero or more than one path exists). Preferred packet directions for row and column can be found by the following equations respectively,

$$rd = \frac{n+1}{2} + r_{source} - r_{destination} \tag{2.17}$$

- If row direction (*rd*) value is positive then packet direction is on positive[4] direction
- If row direction (*rd*) value is negative then packet direction is on negative direction
- If row direction (*rd*) value is 0 then packet direction is either direction (packet can be directed to any direction on row wise)

$$cd = \frac{n+1}{2} + c_{source} - c_{destination} \tag{2.18}$$

[4] Negative direction is defined as packets travelling from right to left (or up to down) and positive direction is the opposite.

- If column direction (*rd*) value is positive then packet direction is on positive direction
- If column direction (*rd*) value is negative then packet direction is on negative direction
- If column direction (*rd*) value is 0 then packet direction is either direction (packet can be directed to any direction on column wise).

Rule1 finds the preferred output port for the packet regardless of the directions of the physical output links.

2.2.6 Hypercube

The Hypercube structure was proposed for computer networks in [Bhuyan & Agrawal, 84] and its properties have been studied briefly in [Saad & Schultz, 88] and [Katseff, 88].

An n-dimensional p-ary hypercube consists of $N = p^n$ Nodes (p = 1, 2, 3..., n = 1, 2, 3...) each of which have p neighbours. Fig. 2.9 shows an 8 nodes 3-cube configuration. In hypercube, nodes are labelled in base-p numbers. The in and out degree of a node depend on the hypercube configuration and it is always p. There is a bi-directional link between any two nodes if and only if the binary representations of their labels differ by one and only one bit, i.e., node $(x_{n-1}, x_{n-2}, \ldots, x_i, x_0)_{\text{base } p}$ is connected to node $(y_{n-1}, y_{n-2}, \ldots, y_i, y_0)_{\text{base p}}$ if $(x_i \neq y_i)$ where $0 \leq i \leq n$.

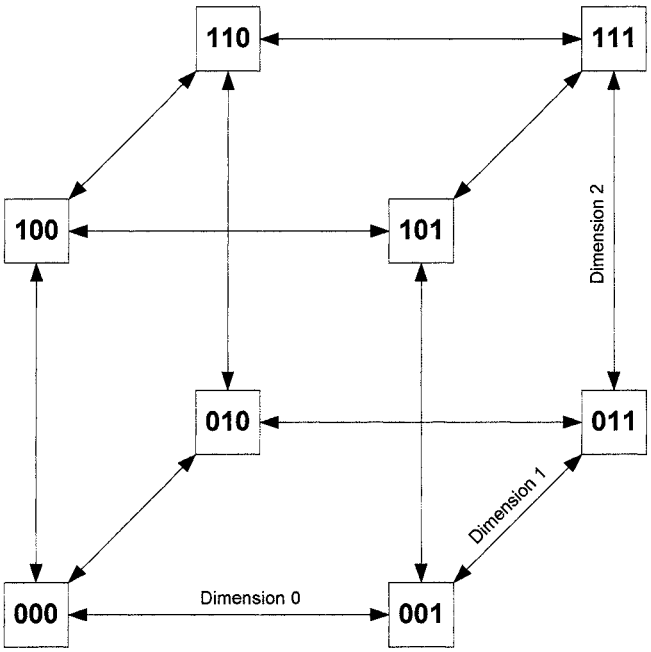

Fig. 2.9: 8 node 3-cube

In general, p-cube can be constructed recursively from lower dimensional cubes. By joining every node of two identical (p-1) cubes whose nodes are numbered from 0 to 2^{n-1} a p-cube can be obtained. The 3-cube in Fig. 2.9 can be obtained by combining two 2-cubes (2-cube is a square). The diameter (maximum hop distance between any two nodes) is p with a deflection penalty of 1 (as bidirectional links are used in hypercube). The average number of hops h required for a packet to travel between two arbitrary nodes in an n-dimensional p-cube is defined as:

$$h = n(1-\frac{1}{p}), \text{ where } h = \frac{n}{2} \text{ in binary hypercube} \qquad (2.19)$$

The main disadvantage of a hypercube is that, its in and out degree increases with the network size. Thus, one can say that hypercube is not a practical solution as every node has to be changed every time network size increases which is a costly exercise.

On the other hand the routing algorithm of hypercube is really simple. A route between two nodes $X = (x_{n-1}, x_{n-2}, \ldots, x_i, x_0)_{base\ p}$ and $Y = (y_{n-1}, y_{n-2}, \ldots, y_i, y_0)_{base\ p}$ is defined by an exclusive or XOR operator given as:

$$\text{Route code}(R) = X \otimes Y \tag{2.20}$$

After finding the route code, the dimensions corresponding to the non-zero positions of the R are traversed in arbitrary order from X to reach Y. Let s be the shortest distance between two arbitrary nodes in a hypercube, then the number of alternate paths between them is defined as k.

2.2.7 Hypercube connected rings NETwork

As name suggests, a hypercube connected ring network (HCRNET) is obtained by replacing each of the nodes of an n-dimensional hypercube by n node rings as shown in the Fig. 2.10 [Banerjee & Sarkar, 94].

A HCRNET consists of $N = n2^n$ (n = 1,2,3...) nodes arranged in 2^n rings of n nodes each. Fig. 2.10 shows a 24 nodes HCRNET configuration. In general, hypercube connections are unidirectional and ring connections are bi-directional. The nodes are labelled as (x_h, x_r), where x_h is the label of the ring which is obtained from the corresponding node's label in the original hypercube, defined as $0 \le x_h \le n$. x_r is the x_r^{th} node in the ring x_h consisting of n nodes, defined as $0 \le x_r \le n$. Node (x_h, x_r) is connected to node $(x_h \otimes 2^{x_r}, x_r)$ via the x_r-dimension of the hypercube. Therefore, the in and out degree of each node is 3(2+1) and it remains unchanged for different

configurations. The diameter (maximum hop distance between any two nodes) is $n+\frac{1}{2}n$.

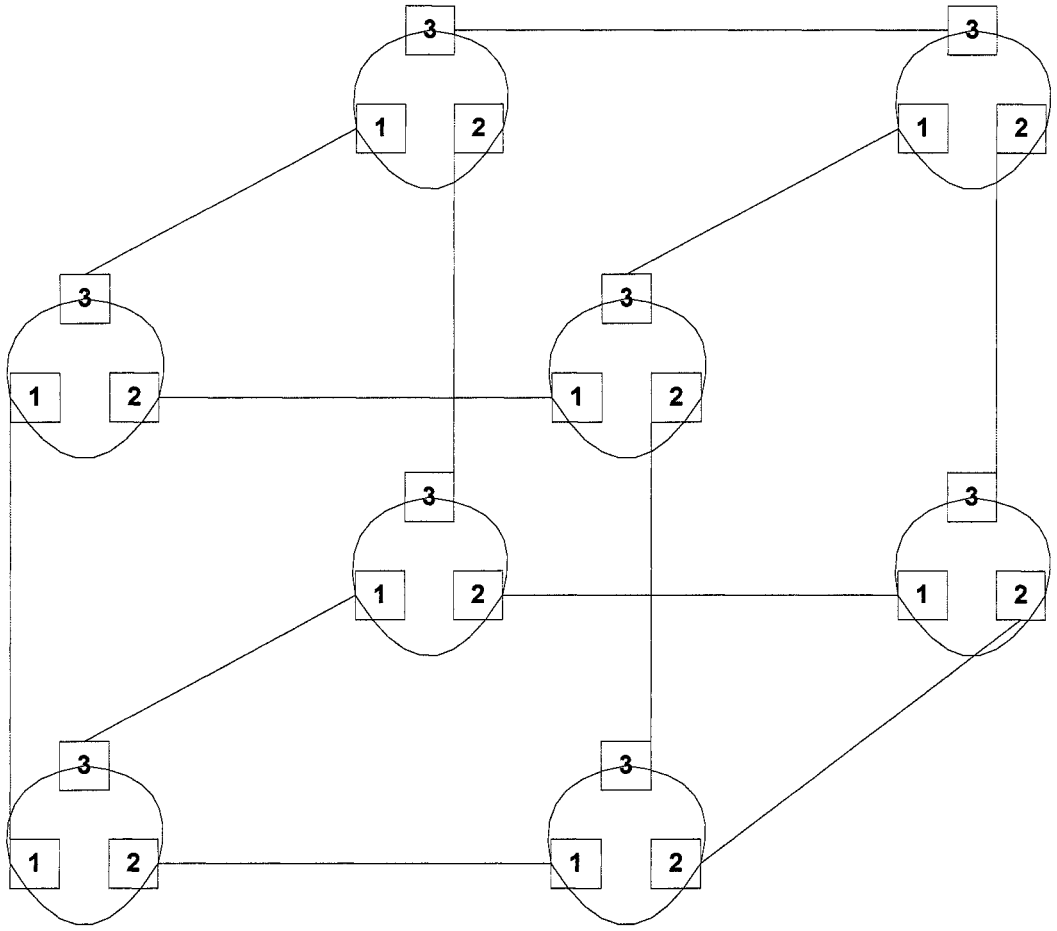

Fig. 2.10: HCRNET

In HCRNET, route from source (c^s,r^s) to destination (c^d,r^d) can be broken into three parts [Banerjee & Sarkar, 94]:

- Routing within the ring of the source node,
- Routing from node x to a node in the destination node's ring (say y) via nodes in intermediate rings,
- Routing within the ring of the destination node,

Based on this, two routing methods are described in [Banerjee & Sarkar, 94], a shortest path routing scheme and a longer path scheme which is a faster and near optimal. Details can be found in [Banerjee & Sarkar, 94].

2.3 Comparison of the Networks

A comparison of the network topologies that have been reviewed is shown in Table 2.1.

Table 2.1: Comparison of network properties

Properties	MSN	Shufflenet	Gemnet	De bruijn	Kautz Graph	Hypercube	HCRNET
Network Config.	(m,n)	(p,k)	(k,m,p)	(Δ,D)	(d,k)	n dim. p-ary	n dim.
Number of nodes (N)	mn	kp^k	km	Δ^D	$d^k + d^{k-1}$	p^n	nr_n
In&out degree	2	k	p	Δ	d	p	3
Diameter	$\sqrt{N}$	$2k$ -1	$[\log_p m]$ + k-1	D	k	p	$n + n/2$
Multiple Path Exists	Yes	Yes	Yes	No	No	Yes	Yes
Deflection Penalty	4	k	m	D	k	1	Unknown
Avg. no. of hops	$\sqrt{N}$	$\log_d N$	$\log_d N$	$\log_d N$	$\log_d N$	$\log_p N$	$\log_d N$
Symmetry	Perfectly Symmetric	Perfectly Symmetric	Depends on the Config.	Good	Perfectly Symmetric	Perfectly Symmetric	Perfectly Symmetric
Scalability	Fair	Poor	Very Good	Poor	Poor	Poor	Good
Routing Simplicity	Low	Very Low	Low	Very Low	Very Low	Very Low	Poor
Link Dir.	Uni	Uni	Uni	Uni	Uni	Bi	Uni and Bi

For all topologies the average number of hops required to travel from source to the destination per packet is in order of $\log_d N$ except for MSN. In MSN, Shufflenet, Gemnet, Hypercube, HCRNET there are multiple paths exists. In Hypercube all the links are bidirectional whereas in HCRNET, within the rings, links are unidirectional and connection of rings is bidirectional. The rest of the networks have unidirectional links. The deflection penalty is constant for MSN and Hypercube. However, in others the average number of hops depends on the size of the network. The De Bruijn graph is the only network that is not perfectly symmetric whereas the Gemnet is the only network that is very scalable. Thus, it can accommodate any number of nodes.

2.4 Conclusions

In this chapter, a review of the most common non-blocking type logical network topologies was presented with the focus on routing algorithms. Comparison of the reviewed algorithms was given and it was identified that for all topologies except MSN the number of hops from source to destination is in the order of $\log_d N$. After this review three topologies, Shufflenet, De Bruijn graph and Gemnet has been selected for implementation as they are the most suitable architectures for OTDM networks. An implementation model and performance investigations of the selected topologies together with some adaptation to the OTDM TOAD will be given in chapter 5. The next chapter will offer a review of the routing algorithms for the arbitrary networks which will be followed by a review of intelligent routing algorithms based on the ant colony optimisation in chapter 4.

CHAPTER III - TRADITIONAL ROUTING ALGORITHMS AND PROTOCOLS

3.1 Introduction

This chapter gives a review of the traditional routing algorithms together with routing protocols that are currently used in today's networks. The chapter commences with an introduction to the routing problem and explains its requirements and characteristics. This is followed by a review of the widely used routing algorithms and protocols. Finally, a comparison of the routing protocols is given and advantages and disadvantages are discussed.

3.2 Routing Problem

Routing is the act of moving information (packets) across a network or Internet from a source to a destination. Routing involves two major acts, one to determine the optimal path to a particular destination (usually in software) and the other to switch the packet to the assigned port which is done by hardware. The switching carried out in the hardware domain is a much simpler process whereas determining the optimal path, which is a stochastic process, is a challenging task [Steenstrup, 95]. Moreover, the optimal path defined may change depending on the network's needs [Steenstrup, 95].

The problem of routing arises when a node in the network tries to send some data (packet) to a destination node [Tanenbaum, 03]. A routing algorithm is required to

find the optimal path (i.e. the shortest path) from source to destination. Solutions to the routing problem must satisfy the following properties where the first two are safety properties that must always be satisfied and the rest must be satisfied eventually [Tanenbaum, 03]:

1. ***Correctness.*** Algorithm should find the optimal or best route, whatever the network condition is.
2. ***Fault Tolerance (Robustness).*** Algorithm not only should be able to deliver the packets to its destination even in the most problematic conditions, but it should also be able to cope with node failures, link failures and path congestion.
3. ***Optimality***. Is the ability to select the best path (route) from the source to the destination. The shortest path normally defines the optimal path but this depends on the metrics and requirements of the network (or system). For example, a path with less buffering delay may be the optimal path, although it might take a longer path.
4. ***Simplicity (low overhead).*** Algorithm should be efficient and simple.
5. ***Stability.*** Algorithm should reach equilibrium after a certain run time.
6. ***Scalability.*** The performance of the algorithm should improve as the resources increase (in the same ratio) and it should also be able to cope with increases in network traffic.
7. ***Fairness.*** A packet should reach its destination within a reasonable time.

Unlike the routing algorithms reviewed in the previous chapter, the algorithms reviewed in this chapter are not based on a specific topology. Hence, they are

sometimes named as routing algorithms for arbitrary topologies. Another common characteristic of these algorithms is that they are all table based, a data structure stored at every node (or router). The routing table entries are defined as routing metrics [Steenstrup, 95] and are used to evaluate the best possible path for a packet to travel to its associated destination. A typical example of a routing table is given in Fig. 3.1 where every identified destination node is associated with an output port and a routing metric.

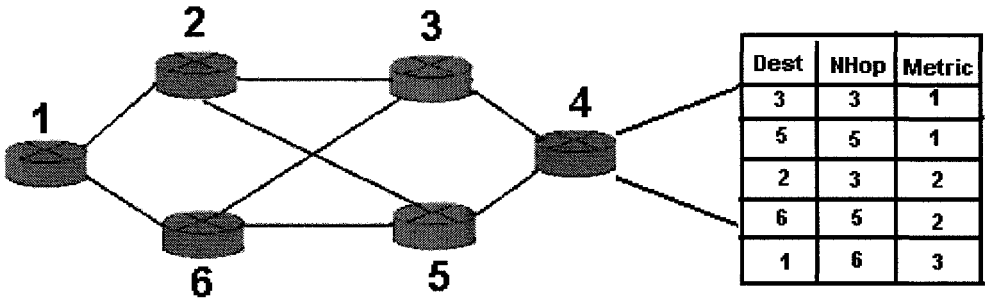

Fig. 3.1: Destination/next hop association in routing table

A routing metric is a standard of measurement, such as path bandwidth, that is used by routing algorithms to determine the optimal path to a destination. Routing algorithms use many different metrics to determine the best route. More advanced routing algorithms base their route selection process on multiple metrics [Cormen et al, 02] by combining them in a single (hybrid) metric. The metrics are:

- **Path length**
- **Reliability**
- **Delay**
- **Bandwidth**
- **Load**
- **Communication cost**

Path length: is the most common routing metric. Some routing algorithms allow the assigning arbitrary costs manually to each network link. In this case, path length is the sum of the costs associated with each link traversed. Other routing algorithms define hop count, a metric that specifies the number of passes through internetworking products.

Reliability: in the context of routing algorithms, refers to the dependability (usually described in terms of the bit-error rate) of each network link. Some network links might go down more often than others. On occurrence of a network failure, certain links might be repaired more easily and quickly than other links. Any reliability factors can be taken into account in the assignment of the reliability ratings, which are arbitrary numeric values usually assigned to network links.

Routing (packet) delay: refers to the length of time required to move a packet from source to destination through the network. Delay depends on many factors, including the bandwidth of intermediate network links, the queues at each node along the way, network congestion on all intermediate network links, and the physical distance travelled by the packet. Since delay is a conglomeration of several important variables, it is a common and useful metric.

Bandwidth: refers to the available traffic capacity of a link. All other characteristics being equal, a 10-Mbps Ethernet link would be preferable to a 64-kbps leased line. Although bandwidth is a rating of the maximum attainable throughput on a link, routes through links with greater bandwidth do not necessarily provide better routes than those through slower links. For example, if a faster link is busier, then the actual time required to send a packet to the destination could be greater.

Load: refers to the degree to which a network resource, such as a node, is busy. Load can be calculated in a number of ways, including central processing unit (CPU) utilization and packets processed per second.

Cost: is another important metric, which is expressed by the different metrics, by different network designers as a function of bandwidth, delay, data rate, financial cost etc. It is usually defined as 1/(bandwidth) in some routers [Bellman, 59].

3.3 Routing Algorithms

The Bellman Ford distance vector routing algorithm and Dijkastra's shortest path routing algorithm are the bases for almost all the routing protocols that are currently in use today [Streenstrup, 95]. A variation of the Distance Vector algorithm is used in the Routing Information Protocol (RIP) and Interior Gateway Routing Protocol (IGRP) whereas the shortest path routing algorithm is the basis for the OSPF routing protocol [Cisco-02]. On the other hand, the Enhanced Interior Gateway Routing Protocol (EIGRP) uses both of the algorithms [Sportack, 99].

3.3.1 Dijkstra / Shortest Path Algorithm

The Dijkstra [Dijkstra, 59] algorithm is one of the most widely deployed routing algorithms in today's Internet. This algorithm computes the routes based on the cost metric, which is expressed by different metrics by network designers as a function of delay, data rate, financial cost, etc. Using the Dijkstra algorithm, every node in the network computes its routes to every other node in the network [Bertsekas, 91].

The Dijkstra algorithm is also termed as a *greedy algorithm,* since it optimises the routes to all the nodes to a single least cost path, and always uses this path without

any regards to future consequences (such as congestion on the most diverted path if the metric is based on the hop count). To find the least cost path or the shortest path, the algorithm chooses a local optimum cost[5] at each step, under the assumption that this will lead to a global optimum cost.

The Dijkstra algorithm can be best explained by presenting the network as a directed graph or digraph, where every node is treated as an edge of the digraph and every link as the vertex connecting any two edges [Bertsekas, 91]. The costs of these vertices are always non-negative. Let V be the set of all the edges of the digraph, $c[s,w]$ is the array of the cost metrics of the vertices between any edge s and w, and $D[w]$ is the array of the least costs (of the shortest path) to any node to be computed. Initially, the cost of the shortest path to any other edge is considered as *infinity i.e.* $D[w] = \infty$. If there are n edges in the digraph, every edge s starts computing the costs $D[w]$ of the shortest paths to other nodes by choosing its neighbour, $w \in (V\text{-}s)$ with minimal value of $c[s,w]$. Then it computes the routes for every other edge $v \in (V\text{-}s\text{-}w)$ as in equation 3.1., where $c[w,v]$ is the costs of the vertex, connecting v to w. This process is also called **relaxing the nodes** and it continues until all the digraph edges are visited. When the process is completed, all nodes in the network have the shortest path to every other node in the network.

$$D[v] = \text{minimum}(D[v], D[w]+c[w,v]) \qquad (3.1)$$

In order not to recompute the previously visited paths, every visited node is marked. If there are e number of edges and n number of vertices in the digraph, the complexity

[5] Optimum cost refers to the least cost of the path to a particular node at that point in transition of the execution of the algorithm in a recursive manner

of the Dijkstra algorithm or the number of iterations carried is O(e log N) where N is number of nodes [Shankar et al, 95].

3.3.2 Link State Routing Algorithm

In the link state routing algorithm, the node (router) running (the link-state routing algorithm (LSA)) advertises the states of its local network links, rather than using the incremental calculation used in the distance vector algorithms. In the link state routing algorithms, topological databases (routing tables) are used to a construct graph of interconnected nodes and links that represents the network. Each node maintains a map of the network in its routing tables with the entries for every known node with their attached links, and neighbouring nodes. The pseudo codes for the link state routing algorithms is given in Fig. 3.2 [Bertsekas, 91].

```
Set Dij to infinity for all j not equal to i
Loop N
        Find the node k not in N for which dik is smallest
                Add k to N
                For each j not in N
                        If Dik + Dkj < Dij then
                                Set Dij to Dik + Dkj
                                Set next_hop(i,j) to next_hop(i,k) concatenated with
                                link from k to j
```

Fig. 3.2: Pseudo code for general link state routing algorithm

Therefore, an efficient flooding[6] algorithm is a crucial requirement. Dijkastra's shortest path algorithm is used to maintain the routing tables.

[6] Flooding is a type of routing algorithm that all of the received packets are sent to (flood) to all of the possible destinations (output ports).

3.3.3 Distance Vector Routing/ Bellman Ford Routing Algorithm

The distance vector routing algorithm was first implemented in the ARPANET based on the Bellman-Ford [Bellman, 59] algorithm. The distance vector algorithm views networks in terms of adjacent routers and estimated distance metrics, rather than predicting the entire network topology. The distance metric can be the geographical distance, hop count or a hybrid function that uses delays, throughput, reliability, etc. The distance vector algorithm estimates the minimum length (cost) paths by using a simple distributed algorithm in the form of:

$$D_{ij} = 0 \ , \ i = j \tag{3.2}$$

$$D_{ij} = \min_{k}\left[D_{ik} + D_{kj}\right] \ , \ i \neq j \tag{3.3}$$

Where D_{ij} is the minimum cost path from source node i to destination node j. D_{ij} is initialized to infinity when $i \neq j$. Every node i executes (equations 3.2, 3.3) concurrently and iteratively to find the minimum length path to node j.

The main advantage of the distance vector algorithm approach is that node i does not have to know all the D_{ij} values, but rather need know only D_{ik} values for those nodes k which are immediate neighbours of i. Thus the distance vector algorithm can be executed in a distributed manner since there is no need for global knowledge. Hence, it is sometimes named as distributed and asynchronous Bellman Ford algorithm [Bertsekas, 91].

```
For all i
        For all j not equal to i
                If j is an immediate neighbour of i then
                        Set Dij to Costij
                        Set next hop (i, j) to j
                Else
                        Set Dij to infinity
                        Set next hop (i, j) to "unknown"
Loop infinity
        For all i
                For all neighbours k of i
                        Send Dij and next hop (i, j) to k for all j
                        Receive Dkj and next hop (k, j) from k for all j
        For all j
                For all neighbours k of i
                        If Costik + Dkj < Dij then
                                Set next hop (i, j) to k
                                Set Dij to Costik + Dkj
```

Fig. 3.3: Pseudo code for Distance Vector algorithm

The pseudo code for the basic distance vector routing table is given in Fig. 3.3 [Bertsekas, 91]. Initially, every node *i* know the cost of the link to its neighbour node *k* only and sets the rest of the possible nodes to infinity. In other words, if the cost of the links and the associated output ports (neighbours) is represented as a table, node *i* would only know the row associated with itself on its neighbour's tables. After the initialisation process, every node updates its neighbour's routing table with the information that it has. Every neighbour node checks the updated message it receives and chooses the minimum path. This process continues in an iterative manner and every node sends updated messages at regular intervals or whenever a change occurs.

Unlike distance vector algorithm in the link state routing algorithms, link state advertisements are broadcasted to all nodes within the same hierarchical area.

3.4 Routing Protocols

3.4.1 Routing information protocol

Routing information protocol (RIP) is a basic distance vector protocol and it is one of the most commonly used routing protocols [Cisco-02]. In RIP, each node in the network keeps a routing table that contains entries for all the reachable nodes in the network, a distance metric associated with each entry and the via-node (next-hop node) if it is not the neighbour of that node. This table is first initialised with the neighbours, before every node sends its current routing update at regular intervals as in typical distance vector routing algorithms[7] to all their neighbours and then each node compares tables and chooses the minimum cost route. RIP uses a very simple metric, hop count (here it is used for age indication), to measure the goodness of the path. Moreover, it only maintains information about the best path. However, unlike the original Bellman-Ford algorithm, RIP retains information about the best cost path to the destination path. RIP detects node or link failures from the missing update messages. For example, some RIP implementations assign infinity for the neighbours if they do not receive an update message for 180 seconds (6 successive update messages). The main drawback of RIP is that it limits the maximum hop value to 16. Hence, the diameter of the network is limited to 16. RIP is a very simple and primitive protocol compared to OSPF and Interior gateway routing protocol (IGRP). However, for a small sized network, RIP has very little overhead in terms of bandwidth and is relatively simple to configure and implement.

In RIP version 1 [Hedrick, 88] two types of RIP messages are used: Request Messages and Response Messages. A router requests an update when it first joins a

[7] In the original RIP configuration it sends the update messages in every 30 seconds.

network and all routers respond with Response Messages containing the routes. In addition to this, Response Messages are sent every 30 seconds by default. The default values used in RIP 1 are given in Table 3.1. Triggered updates occur when there is a change in the parameters and it contains the changed information to speed-up network convergence.

Table 3.1: Parameters used in RIP

Parameter	RIP Default Value
Infinity	16 (fixed)
Update time	30 sec
Invalid time	180 sec
Flush time	120 sec
Holddown	Not used

The RIP 2 specification described in [Malkin, 98] allows more information to be included in RIP packets and enables the use of a simple authentication mechanism to secure table updates. More importantly, RIP 2 supports subnet masks, a critical feature that is not available in RIP 1.

3.4.2 Interior gateway routing protocol

IGRP was developed by CISCO in the mid 80's for Internet protocol (IP) networks to overcome the weaknesses of the RIP protocol [Hedrick, 91]. It is a Distance Vector routing protocol which uses a composite metric to calculate the best routes. The composite metric function is defined as follows:

$$\frac{\left(\frac{k_1}{C'}+\frac{k_2}{D'}\right)}{R''} \tag{3.4}$$

Where, D' is the routing delay (1 - 2^{24}), Route delay in tens of microseconds. C' is the effective route capacity (1200 kbps - 100 GBps). R'' is the reliability (1 - 255), The value 255 means 100 percent reliability, 0 means no reliability. k_1 and k_2 are the coefficients that determines the impact of C' and D', and L is the load (1 - 255), 255 is 100 percent loading.

The weights of these values are defined by the network administrator according to the needs of the system. The route delay may include the following: transmission delays, queuing delays, propagation delays and processing delays. C' is computed as the product of sum of minimum bandwidth and the percentage of the available capacity based on the load (equation 3.5). Coefficients k_1 and k_2 are assigned to determine the relative importance of the available bandwidth and the delay in the function. IGRP also supports multi path routing by assigning multiple paths for the given range of values. This is done by a variable called "variance" which selects the paths within the specified range defined by the variance. In particular, the acceptable routes must have value less than or equal to variance v times the minimum cost.

$$C' = \sum_{i=0}^{N} C_{\min i} L \qquad (3.5)$$

IGRP uses hold-downs, split horizons and poison-reverse updates to improve reliability and stability of the protocol. Hold-downs inform the node to keep any changes that might affect routes for some period of time. Nodes detect failures from lack of regularly scheduled update messages from its neighbours. The node then calculates new routes and sends this information to its neighbours. However, sometimes nodes do not receive update messages in the correct order, due to delays

(which cause correct information to be overwritten). Thus, hold-downs are used to prevent this happening. The split horizon rule is used to prevent the updating of the node from which the information is gathered from. Poison reverse acts in the same way but is not only limited to the neighbouring nodes.

3.4.3 Open Shortest Path First

The main characteristic of OSPF is that it is based on the Dijkastra's shortest path first algorithm (SPF) [Moy, 98]. It uses SPF algorithm to calculate the changes in the routing table. The algorithm first sends a 'hello' message to every output port to identify its neighbours and learn their network addresses. On receiving hello messages nodes reply back. The hello packets are also transmitted every 10 seconds by default, to check connection between the nodes. If a node in the network does not receive a hello packet from one of its neighbours, then it starts a dead interval counter for that node. If no hello packet is received within the dead interval time (usually 4 times the hello interval time), then the node will send a link state update indicating link failure [Steenstrup , 95].

In order to compute the link costs, every node sends out echo packets on every point-to-point link, and receiving nodes are required to reply back immediately. By measuring the round-trip time and dividing it by two, the sending node can get a reasonable estimate of the delay. The delay of the link can also be manually assigned by the network administrator. In addition, each node sends periodic link state advertisements (LSAs) to every available destination in the network to inform other nodes of the changes that it has discovered. LSAs are also flooded, once the algorithm initialises itself and learns about its neighbours.

One other characteristic of OSPF is that it supports hierarchal routing, where the network is divided into many areas with area "zero" as the backbone area, which interconnects with all other areas in the network. Every area is identified with a unique code. If hierarchical routing is not required, all the nodes in the network are configured to be in area "zero" or the backbone area by default. All the packets between areas should route across the backbone area.

3.4.4 Enhanced interior gateway routing protocol

Enhanced interior gateway routing protocol (EIGRP) is similar to IGRP in many ways, but is different in that it has properties of both a distance vector routing protocol and a link state routing protocol [Steenstrup , 95]. In EIGRP, instead of making periodic updates, nodes send partial updates when the metric for a route changes. EIGRP uses four mechanisms to improve the performance of the routing algorithm. The neighbour recovery/discovery mechanism sends hello packets to discover nodes' neighbours and failures within them. A reliable transport protocol ensures that packets are delivered to all neighbours, if necessary using acknowledgement messages. Two staged process is used to track validity and correctness of the received update messages. Protocol dependent modules are responsible for network layer specific requirements [Steenstrup, 95]. In addition to the four mechanisms explained above, four additional data structures are used in the EIGRP: neighbour tables, topology tables, route states and route tagging.

For every discovered neighbour, there is an associated table containing its address and round trip time. The topology table keeps all of the destination addresses, together with the metrics that have been advertised so far by the neighbours. A topology table

entry is in passive state while the node is recalculating it and active state otherwise. This information is kept in the route states. In addition, route tags are used to maintain and control routing policies in a more efficient way.

EIGRP uses the following packet types: hello and acknowledgement update and query and reply [Sportack, 99]. Hello packets are broadcast to all neighbour nodes for discovery and recovery requiring no acknowledgement. The hello message with no data is an acknowledgement message. Update information is used to send the new link and node information to other nodes. Query packets are broadcast when the destination has no feasible successors and reply packets are sent as a response to the query message.

3.4.5 Border gateway protocol

The BGP is an interautonomous system routing protocol [Malkin, 98]. An autonomous system is a network or group of networks under a common administration and with common routing policies, such as a university network. BGP version 4 is the current de facto exterior routing protocol in the Internet and is the protocol used between Internet service provider (ISP)'s. Customer networks, such as universities and corporations, usually employ an Interior gateway protocol (IGP) such as RIP or OSPF for the exchange of routing information within their networks. Customers connected to ISPs, and ISPs use BGP to exchange customer and ISP routes. When BGP is used between autonomous systems (AS), the protocol is referred to as external BGP (EBGP). BGP learns multiple paths via internal and external BGP speakers and it finds the best path and updates associated properties in its routing

table. These properties are referred to as BGP attributes which are weight, local preference, multi-exit discriminator, origin, AS_path[8], next hop and community

3.4.6 Exterior gateway protocol

EGP is for exchanging routing information between two neighbour gateway hosts in a network of autonomous systems [Rosen, 82]. It uses periodic message exchanges Hello/I-Heard-You (I-H-U) to monitor the status of the links with its neighbours. The routing table contains a list of known routers, the addresses they can reach, and a cost metric associated with the path to each router so that the best available route is chosen. However, unlike BGP, EGP do not attempt to choose the best route by itself. EGP updates the associated distance-vector entries in the routing table, but does not evaluate this information. This is because the routing metrics from different autonomous systems are not directly comparable. Each AS may use different criteria for developing these values. Therefore, EGP leaves the choice of a 'best' route to someone else. When EGP was designed, the network relied upon a group of trusted core gateways to process and distributes the routes received from all autonomous systems. These core gateways were expected to have the necessary information in order to choose the best external routes.

3.5 Comparison

Although the distance vector routing algorithms are simpler they have slow convergence and have split horizon problems. Link state algorithms are more complex and require more resources compared to the distance vector routing algorithms but they converge faster than the distance vector routing algorithms. Distance Vector

[8] Autonomous Systems (AS)

algorithms need computational complexity of O (N^3). In the worst case scenario, the algorithm must be iterated N (number of nodes) times with each iteration having to be done N-1 times and for each iteration minimisation having to be taken N-1 times. On the other hand, the Dijkastra's link state algorithm has the computational complexity of O (N^2). Some approaches, as in EIGRP, use a hybrid approach to benefit from the advantages of the both algorithms. A comparison of the traditional routing algorithms is given in Table 3.2.

Table 3.2: Comparison of the routing protocols

Algorithm	Type	Characteristics
RIP	Distance Vector	A very basic protocol, with some problems and limitations as it uses limited hop counter as the routing metric
IGRP	Distance Vector	Designed for IP networks, uses a composite metric, hold-downs, split horizons and poison-reverse updates to improve reliability and stability
EIGRP	Hybrid	Uses four mechanisms, neighbour recovery/ discovery, reliable transport protocol, DUAL finite state machine, protocol dependent modules to improve the performance of the routing algorithm
EGP	Distance Vector	Inter domain protocol, restrict itself to two levels of domains
BGP	Path-vector	No restrictions unlike to EGP, exchanges vector of destinations and domain level path information with its neighbours
Internet Domain Routing Protocol (IDRP)	Path-vector	Uses all of the ideas in the BGP-4
OSPF	Link State	Uses flooding to advertise update messages and can be used with inter-domain routing protocols

3.6 Conclusions

In this chapter a review of the traditional routing algorithms together with routing protocols that are currently in use today's networks were given. Dijkastra's shortest path algorithm is one of the well known algorithms widely adapted in the Internet and studied in the literature. It also forms the basis for the link state routing algorithm and protocols. OSPF is deployed throughout the internet and is the de facto routing algorithm with in the networks. EGP is a gateway protocol which is mainly used between the backbones and as it name suggests, among the gateways. However, BGP is the most widely used gateway protocol.

The review highlighted that all of the algorithms that has been reviewed here (which are also widely used in the Internet) needs human interpretation in order to adapt to the changes (i.e. node/link failures, adding/removing links/nodes, congestion). Therefore, there needs to be an administrator to monitor these algorithms and than modify the variables used in these algorithms based on status of the network. Because of this, there is a growing body of research in intelligent algorithms that can adapt themselves to the changes in the network. Therefore, next chapter will be review of the intelligent algorithms.

CHAPTER IV - REINFORCEMENT LEARNING AND ANT COLONY OPTIMISATION

"An agent is something that perceives and acts in an environment" [Russell & Norvig, 95]

4.1 Introduction

This chapter analyses ant colony optimisation (ACO) and reviews the routing algorithms based on ant colony optimisation. It begins with an introduction to reinforcement learning and describes how this works. After providing a brief description of reinforcement learning, ant colony optimisation is analysed and the antnet routing algorithm together with the algorithms derived from it are reviewed in detail.

The routing algorithms discussed in the previous chapter lacked intelligence, thus requiring human assistance and interpretation in order to adapt themselves to failures and changes. Routing is considered to be an NP-Hard Optimization problem, therefore widely used optimization problems have been applied in the literature. To name a few, Genetic Algorithms [Liang et al, 02], Neural Networks [Dixon et al, 95], Simulated Annealing [Osman & Kelly, 96], Software Agents [Yang et al, 02] [Amin & Mikler, 02-1] and Reinforcement Learning [Littman & Boyan, 92].

The routing problem has many similarities with the travelling salesman problem (TSP) [Larra Naga et al, 99] and can be considered to be a scheduling problem. Therefore most of the research that has been applied to TSP and scheduling problems has been adapted to the routing problem. However, most of the meta-heuristic optimization tools are not capable of producing satisfactory results for real time systems, as they need some period of time for the learning process and demand high processing power.

In recent years, agent based systems and reinforcement learning have attracted researchers' interest. This is because these methods do not need any supervision and are distributed in nature [Bonabeau et al, 00] [Kunkle, 01]. Swarm intelligence (particularly ant based systems) [Dorigo et al, 02-2], Q-learning [Boyan & Littman, 94] methods and hybrid agent based distance vector algorithms [Amin et al, 01] have shown promising and encouraging results.

Unlike other optimisation problems the routing problem which is a dynamic optimisation problem, has to be solved in real time. [Schoonderwoerd, 96] was the first to apply ant based systems to the routing problem. This work was influenced by the work reported in [Appleby & Steward, 94] and was applied to circuit-switched and connection oriented systems. [White & Pagurek, 98] and [Bonabeau et al, 98] later on improved [Schoonderwoerd, 96] the algorithm and applied it to the connection oriented systems. [Subramanian et al, 97] was the first to apply the ant colony concept to packet based connectionless systems. Others such as, [Heusse et al, 98] and [Di Caro & Dorigo, 97], [Di Caro & Dorigo, 98-1], [Di Caro & Dorigo, 98-2], [Di Caro & Dorigo, 98-3] and [Di Caro & Dorigo, 98-4] followed in the footsteps of

[Subramanian 97]. In the following section we will first review reinforcement learning and its applications to the routing problem, particularly to the applications of agent and ant based routing algorithms. The chapter then introduces the work reported in [Appleby & Steward, 94] which has influenced the work reported in [Schoonderwoerd, 96] and gives a review of ant-based routing algorithms.

4.2 Reinforcement Learning

Reinforcement learning is an agent based learning method where agents are rewarded for the actions that they take. Therefore agents are reinforced with the rewards and directed towards better solutions. In reinforcement learning, an agent should be able to sense the environment and be able to take actions that could change the environment or the state of the environment. Also, an agent should have a goal or goals in order to take actions [Weyns et al, 04]. Based on the action taken, agent receives a feedback which is called "reward" or "reinforcement" (see Fig. 4.1). The, agent then uses this reward to learn and use this experience in the actions that it takes.

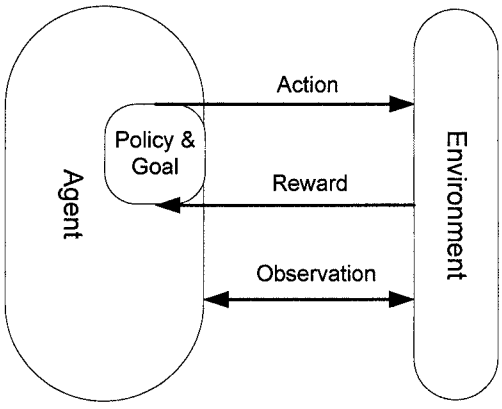

Fig. 4.1: Agent environment interaction

Agents are never told what the right actions are and are left to their own accord to explore and make appropriate decisions in order to increase their knowledge. Beyond the agent and environment [Sutton & Barto, 98] identified the four main aspects of reinforcement learning: a policy, a reward function, a value function and a model of the environment (optional), see Fig. 4.2 [Kaelbling et al, 96].

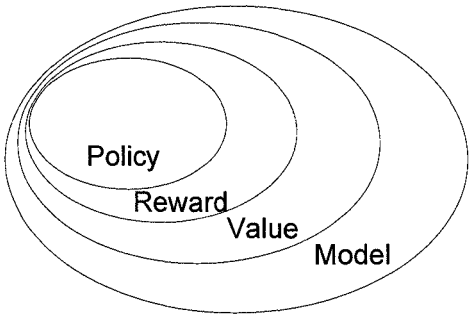

Fig. 4.2: Main components of reinforcement learning agent

Fig 4.2 shows the main components of reinforcement learning agent. Policy is the way that agent behaves based on knowledge gained from previous actions. It is composed of a set of rules that the agent has to obey. The reward function informs the agent how well it has performed its tasks based on its goal. Based on the reward function an agent comes up with a solution to the problem which is called the "value function". This function is the long term representation of the reward function. The environment model tells the agent what might happen or what the expected reward is based on the action taken. Models are built on statistical knowledge and experience gained by the agent. More detailed information about the reinforcement models can be found in [Sutton & Barto, 98], [Kaelbling et al, 96] and [Andrei, 02].

There is a trade off between exploiting and exploring in reinforcement learning. When there are less agents in the system that are exploring for a new solution they get stuck

in the local optima and cannot find the optimum solution. On the other hand, when they explore more, they create more overhead in the network thus resulting in low network performance. Therefore there has to be a balance between exploring and exploiting in order to find the optimal solution for a given problem.

4.3 Swarm Intelligence

"Artificial life is the study of man-made systems that exhibit behaviour characteristics of natural living systems" [Schoonderwoerd, 96]

Swarm intelligence is a form of reinforcement learning based on observations from nature [Carnahan & Simha, 01]. Researchers have been trying to analyse and understand the dynamics of nature, as it is the perfect working model that can be used to develop the meta-heuristics. Genetic algorithms, neural networks, simulated annealing and swarm intelligence are the most studied examples. All of these approaches have one goal; *to apply existing well observed dynamics of nature on man made systems to solve hard optimisation problems*. Goldberg and his students [Goldberg, 89] applied the Darwinian approach to optimise the given problem [Chambers, 95] using genetic algorithms. Neural networks, where many processing elements are interconnected like neurons process a given task in a highly parallel fashion just like human brain [Dixon et al, 95]. Annealing is a chemical process that is used to harden metals and glass starting at high temperatures and then cooling it slowly. Simulated annealing is a process that attempts to find the minimum (optimum) cost (solution) in the given cost function (problem). In recent years however, swarm intelligence, especially animals that have emergent collective behaviour, has attracted researcher's interest where solutions for large tasks emerge as

a result of the behaviours of individual entries. Bird-flocking, fish-schooling, nest building and ant colonies have been simulated to solve NP-hard optimisation problems [Camazine et al, 01]. These systems are distributed in nature with no chosen leader, hence they are sometimes named as "unsupervised learning".

4.4 Ants and Ant Colony Optimisation

Individual ants are very unsophisticated and simple insects that are capable of performing a variety of complicated tasks when they act in collective manner. Real ant behaviour has been observed by many researchers and they have agreed on the following collective behaviour of several ant species reported in, [Beckers et al, 92], [Franks, 89], [Schoonderwoerd, 96] and [Dorigo et al, 02-1]. Ants in real life:

- explore particular areas for food
- build and protect their nests
- sort brood and food items
- cooperate in carrying large items
- immigrate as colonies
- leave pheromones on their way back
- preferentially exploit the richest available food source
- store information on the nature(uses world as a memory)
- make their decision on a stochastic way
- always finds the shortest paths to their nests or food source
- are blind, can not foresee future
- have very limited memory

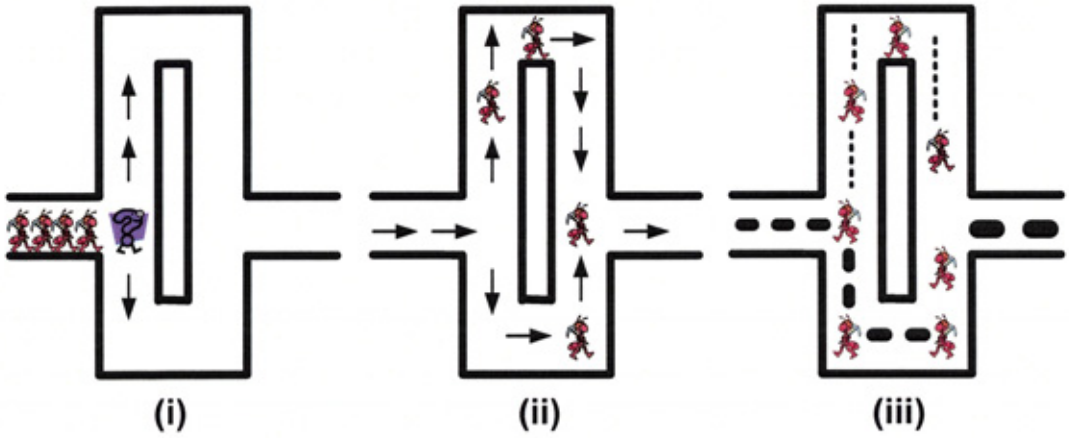

Fig. 4.3: The process of how ants find optimal path in real life

It is believed that these behaviours emerge from the interactions between the large numbers of ants and the environment. Also, it has been observed by many researchers that in nature ants always find the shortest path from their nests to the food source. They do this by using the notion of "stigmergy", which is a form of indirect communication through the environment [Beckers et al, 92]. Ants lay pheromone, a chemical substance, to mark the path that they use to the food source. For example, in Fig. 4.3(i) ants initially have no idea which direction to take and choose one of the paths randomly. It is expected on average that half of the ants to choose the shorter path and the other half to choose the longer path (they have no information or knowledge of the paths at this stage). It can be observed from Fig. 4.3(ii) that the ants that have chosen the shorter path should reach to their destinations more quickly than those that selected the longest path. These ants will also take the shorter path on their way back to the source before the others. In Fig. 4.3(iii), the number of ants choosing the shorter path will be increased after this step as there will be more pheromone laid on to the shorter path. Recently, there has been increased research activities inspired from these "simple" insects reported in [Beckers et al, 92]. They applied ant behaviour to solve NP-hard optimisation problems and used ants (agents) to sense the

environment and to find solutions. However, there are differences between real ants and artificial ants[9] where an artificial ant is nothing but a software agent. Artificial ant based systems have the following characteristics:

- ants (agents) do not communicate directly, rather they change the environment (update some common data structures)
- ants deposit pheromone as a function of quality of the solution found,
- ants make their decision based on a probabilistic way, based on a defined probability function
- ant systems are resilient and dynamic
- there are no supervisors
- ant systems are highly distributed
- ant systems are robust
- ant systems are highly scalable
- ants are self organizing, no interaction is required from outside
- ants perform backwards learning
- ants have some but limited memory

[9] Ants and agents will be used interchangeably throughout the text.

```
// ACO Meta-heuristic
 Begin
While (not terminate)
        Begin schedule activities
        // The following Three functions are executed concurrently in parallel
                Generate_ants ();
                Evaporate_ pheromone ();
                Daemon_actions ();
                // executed based on central knowledge or another op. method
                // routing tables can be updated here independent from the ants
        End schedule activities
End While

Generate_ants ();
// ants are created here and policy is given in this function
Begin
        Initialise_ant ();
        // on initialisation random dest. is given to the ant and M data stack is updated
        While (solution not found)
                A = Local_ant_routing_table;
                P = Transition_Probabilities (A, M, problem_constraints);
                Next_state = apply_ant_decision_policy (P, problem_constraints);
                If (forward_mode)
                        Deposit_pheronome (); // on to the visited arc
                        Update_ant_routing_table;
                End If
                Update M; //as routing table is changed
        End While
        If (Backward_mode)
                Evaluate_solution ();
                Deposit_pheronome_on_all_visited_nodes ();
                Update M; //as routing table is changed
        End If
        // forward mode and backward mode functions are mutually exclusive
        // when both is not executed then daemon has updated the tables.
        Die (); // Once ant reaches back to source node it dies
End.
```

Fig. 4.4: Pseudo code for ACO meta-heuristic

Ant-colony based algorithms were pioneered by [Coiorni & Dorigo, 92] and [Dorigo et al, 96] as a multi-agent approach inspired from real ant colonies in nature and based on the Monte Carlo model [Sutton & Barto, 98] and [Fishman 96]. Ant based systems were first applied to the travelling salesman problem (TSP) and in [Dorigo & Gambardella, 97] [Dorigo & Gambardella, 99] this heuristic is named as ant colony optimisation [Dorigo & Di Caro, 99] [Dorigo et al, 02-1]. The pseudo code for the general ACO meta-heuristic [Dorigo et al, 99] is given in Fig. 4.4.

In Fig. 4.4, first three functions can be executed concurrently in a distributed system, but some synchronisation may be needed. *Daemon_actions ()* requires additional knowledge to the locally available information, where this knowledge could be centralised knowledge or another optimisation method such as GA. Forward_mode and backward_mode are executed in a mutually exclusive manner; ants either update pheromone tables before or after they built the solution, on their way back to the source node. Initially, three ant based meta-heuristics have been implemented in [Dorigo et al, 99] namely: ant-density, ant-quantity and ant-cycle. In the first two, ants deposit pheromone after building the complete solution. However this has been found to be inefficient. Ant-cycle heuristic has been proposed, where backward ants are used to lay the pheromones.

4.5 Antnet routing algorithm and its derivatives

4.5.1 Mobile software agents for control in telecommunications networks

[Appleby & Steward, 94] used software mobile agents to establish (to route) telephone calls in British telecom (BT)'s telephone network and to optimise the load balancing in the networks. Two kinds of software agents used in their system namely:

1. Load management agents: designed to distribute load evenly
2. Parent agents: responsible for managing the population level and task allocation of the load management agents.

A load management agent provides lower level of control, each agent is launched from source to destination and routed by a variation of the Dijkastra's well-known routing algorithm [Bertsekas et al, 92] which updates routing tables on their way to destination. The best route is defined as the "*largest minimum spare capacity*" of all possible routes which is to distribute load evenly avoiding high local node loadings.

Parent agents provide second level of control. They travel along the network and discover the most congested parts of the network, the nodes that creates most traffic. [Appleby & Steward, 94] applied the following important properties to achieve robustness:

- There should be no direct inter agent communication
- There should be a large number of agents
- The agents should be able to dynamically alter their population and task allocation.

4.5.2 ABC routing in telecommunications networks

The work reported in [Schoonderwoerd, 96] applied the ants concept to the routing problem for the first time based on the work reported in[Appleby & Steward, 94]. In this work, ant colony optimisation was applied to circuit switched telephone networks. Routing tables are replaced with a table of probabilities and ant based agents, rather than ordinary agents are used for updating the routing tables in order to optimise the performance of the algorithm on changing data traffic loads. Ants have the following properties [Schoonderwoerd, 96] and [Schoonderwoerd et al, 96]:

- Two kinds of ants are used in the routing algorithm, forward and backward ants
- Each ant (forward ant) is initially sent to a random destination
- On reaching their destination, ants die and create backward ants to travel back to source following the marked path
- Ants are delayed at congested nodes, preventing them from laying more pheromone on the path that encourages following ants (this increases the probability of usage of other paths)
- Laying pheromone is proportional to the ants age, older ants lay less pheromone.

In ABC each row and column in the routing table corresponds to a neighbour, and to a destination node respectively. Backward ants travelling from the destination to the source node normally update the routing table entries. This update takes place in two ways: (i) a positive reinforcement on path taken by the agent (equation 4.1) or (ii) a negative reinforcement on the other paths (equation 4.2). Basically, an ant increments the pheromone level, i.e. increasing probability of path used so that the ants coming behind could use that particular path.

$$r(t+1) = \frac{r(t) + \delta r}{1 + \delta r} \tag{4.1}$$

$$r(t+1) = \frac{r(t)}{1 + \delta r}, n \neq i - 1 \tag{4.2}$$

Where δr is the reinforcement parameter which is based on ants characteristics showing how well the ant is performing. [Schoonderwoerd, 96] used ants absolute age

T', which is the measured time spent in the network, to calculate the reinforcement parameter as:

$$\delta r = \frac{a'}{T'} + b' \tag{4.3}$$

Where a' and b' are ants parameters. Stagnation is considered to be one of the biggest issues in reinforcement learning. In ABC routing, the table freezes after some time since probability entries are used in the routing table. Thus, the algorithm is enabling to exploit the newly found optimal routes. In other words, the algorithm sticks in local optima and is not therefore able to reach the global optimum solution. This will be explained in detail and further solutions will be provided with novel approaches in Chapter 6. In ABC routing, a scheme called the noise function, which can be considered as an exploration parameter has been used to ensure that the algorithm does not get trapped in the local optima. Based on the noise function[10] an ant chooses a random destination at each hop, thus being able to fully search the entire network. Later on it will be shown that using the noise function, the network can be managed more appropriately to find best paths performance [Bonabeau et al, 99]. However, the main drawback of this algorithm is that it can only find single paths from source to destination. The pseudo code for the ABC algorithm is given on Fig. 4.5 [Bonabeau et al, 99]. In the initialisation steps, ants explore the network in the absence of network traffic which enables ants to initialise the routing table entries.

[10] Noise is set to 0.05 in the ABC routing algorithm where 5% of the ants are forwarded to the random destinations.

```
Main
Initialization // no data traffic ants discover paths and updates routing table entries
For t = 0 to t_initialise
Begin
        For i = 1 to N
        Begin
                Launch_ant(random(destination))
        End Loop
        Route ants based on probability entires and Update routing table entries
End Loop
If (t_interval) is reached // time interval for ant generation
Then
        Destination_node = random (possible destinations)
        Sent forward ant (destination_node, source_node)
End If
If (Ant_received)
        Function_ ant_received ( )
End If
End Main
Function_ ant_received ( )
If (backward ant)
Then
        For i = 1 to N
                Begin
                        If i = node came from
                                r(t+1) = (r(t) + δr) / (1 + δr)
                        Else
                                r(t+1) = r(t) / (1 + δr)
                End Loop
        If (current node = destination node)
        Then
                Kill(backward Ant)
        End If
        Push_stack (next_hop_node, Ant);
        Sent_ant (next_hop node);
End If
If (forward ant)
Then
        If NOT (current node = destination node)
        Then
                If rand(1) == g  // noise(g) set to 0.05
                Then
                        Sent_ant (random(output port))
                        // Set next hop randomly from the possible output ports.
                End if
                Else
                        Choose next hop from routing table among the output ports.
        End if
        Else
                Change type to Forward Ant
End if
```

Fig. 4.5: Pseudo code for ABC algorithm

4.5.3 Regular and uniform ant routing algorithm

[Subramanian et al, 97] also applied ant colonies to packet switched networks to solve routing problem in real time. ABC routing algorithm rules are used to update the routing tables. Two routing algorithms have been proposed namely, regular ants and uniform ants. Regular ants use routing tables while searching for the optimal solution, whereas uniform ants, being unbiased, use randomised rules to find their path. Therefore, they explore paths rather than going through the paths, which are directed by the routing tables. Moreover, uniform ants, can find multiple paths as they explore the network whereas regular ants are only capable of finding a single path. This makes regular ants more complex compared to the uniform ants as they find their way around. The most remarkable characteristic of the approach in [Subramanian et al, 97] is that, ants are piggybacked on the data packets, thus utilising the network much more efficiently. However, this algorithm is based on the assumption that the traffic on the path followed by the forward ant is same as the traffic on the path followed by the backward ant. However, this is not a valid assumption to make any longer, since today's packet switched networks are no longer symmetrical (i.e. upload and download links have different capacities).

4.5.4 Cooperative Asymmetric Forward routing algorithm

[Heusse et al, 98] also proposed a packet-switching version of the [Schoonderwoerd, 96] routing algorithm, which is named as the cooperative asymmetric forward (CAF) routing algorithm. The idea was also inspired from the link state and distance vector routing algorithms where agents update the routing tables rather than broadcast the messages used in OSPF routing. Two types of agents have been used, forward agents and back propagating agents. Forward agents share the same queues as the data

packets enabling agents to collect realistic information about the network's performance. On the other hand, propagating agents retrace their route to the source by using high priority queues and update the routing tables in a much quicker way (rather than sharing low priority queues with the ordinary data packets). Once a loop is detected they update their data structures rather than deleting the entries (unlike the approach in [Di Caro & Dorigo, 98-4] which will be explained in detail later in this chapter).

4.5.5 Smart ants

[Bonabeau et al, 98] extended the [Schoonderwoerd, 96] algorithm by embedding dynamic programming that made agents "smarter" and improved the performance of the algorithm. The algorithm is based on the assumption that a sub- path within the optimal path is also optimal. Although ants update the links associated with the source-destination pair, paths can also be updated on their way to their destination. Routing table entries are updated using the same rules as in the ABC algorithm, except for the absolute age which is replaced with ant's relative age (T_i - T_f). The reinforcement parameter is given as:

$$\delta r = \frac{a}{T_i - T_f} + b \qquad (4.4)$$

Where, T_f is the ant's absolute age on visiting the node f and T_i is the ant's age on visiting the intermediate node i.

4.5.6 Antnet routing algorithm

[Di Caro & Dorigo, 97] [Di Caro & Dorigo, 98-4] proposed the antnet routing algorithm where ants explore the network and collect information about network status by using routing policies based on the local information at each node. Moreover, while exploring the network, ants updates the routing tables to build a statistical model of the network status. Ants also communicate with each other via these tables. In this algorithm, two types of agents namely, **forward ants (agents)** and **backward ants (agents)** are used to update the routing table entries. In each node there are two kinds of queues, **low priority** and **high priority**. The data packets and the forward agents use low priority queues whereas the backward agents only use the high priority queues.

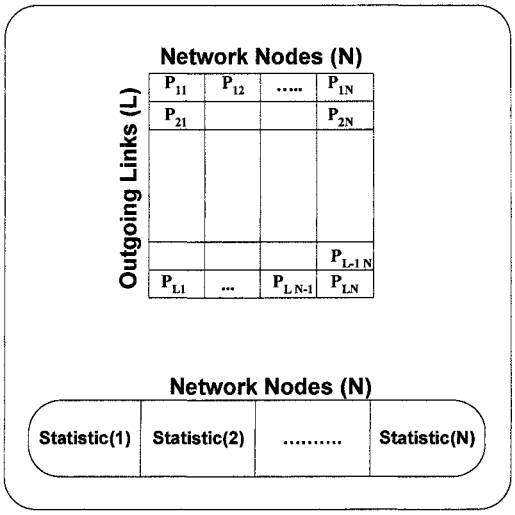

Fig. 4.6: Data structures stored in every node

Agents communicate through the two data structures stored in every node (see Fig. 4.6) as outlined below:

i. A distance vector routing table T_k with distance metric defined by the probabilistic entries. For each possible destination d and neighbour node n a probability value P_{nd} is used reflects the goodness of the link (path) given as:

$$\sum_{n \in N_k} P_{nd} = 1, d \in [1, N],\ N_k = \{Neighbours(k)\} \tag{4.5}$$

ii. An array M_k (μ_d, σ_d^2, W_d) defines a simple statistical traffic model experienced by the node k over the network. Where, W_d is the observation time window used to compute the estimated mean μ_d and variance σ_d^2 that gives as:

$$\mu_d = \mu_d + \eta(o_{k \to d} - \mu_d)^{11} \tag{4.6}$$

$$\sigma_d^2 = \sigma_d^2 + \eta((o_{k \to d} - \mu_d)^2 - \sigma_d^2) \tag{4.7}$$

Where, $o_{k \to d}$ is the new observed trip time experienced by the agent while travelling from node k to destination d. Behaviour of the Antnet can be summarised in the following six steps:

1. At regular intervals[12] (defined by the user) from every node an agent A is sent to a randomly selected destination node d.

2. Each agent first defines the possible neighbour nodes (**unvisited neighbour nodes**) at the current node by using its routing table. Then, the agent chooses the

[11] η is the factor that weights the number of most recent samples that will effect the average and is set to 0.005 [Di Caro & Dorigo, 98-4].
[12] This value is set as 0.3seconds [Di Caro & Dorigo, 98-4].

next node n within the identified possible (unvisited) nodes by using the probabilistic values and by taking into account the state of the associated queue by using the equation 4.8[13].

$$P'_{nd} = \frac{P_{nd} + \alpha\, l_n}{1 + \alpha(|N_k| - 1)} \tag{4.8}$$

Where, l_n is the heuristic correction value with respect to the probability values stored in the routing table that gives a quantitative measure associated with the queue waiting time and defined as [Di Caro & Dorigo, 98-4]:

$$l_n = 1 - \frac{q_n}{\sum_{n'=1}^{|N_k|} q_{n'}} \tag{4.9}$$

And q_n is the length of the bits (or number of packets if packet size is fixed) waiting to be sent to the queue of the output port n of the node k.

3. If the selected port is full then the agent is stored in the first in first out (FIFO) output buffer of port and waits for its turn. It is assumed that the buffer size is infinitely large.

4. The identifier of every visited node and time elapsed since the agent was despatched from the source is pushed into the stack $S(k)$ carried by the agent.

[13] α is the value that weights the importance of the heuristic function with respect to the probability values stored in the routing table and it is set to 0.3.

5. If an agent is forced to visit previously visited node, hence if a cycle exists, then it deletes all the entries for the nodes associated with the cycle. Moreover, if the cycle length is greater than half of the agent's age, then the agent is destroyed.

6. When a forward agent reaches the destination it changes its type to a backward agent and returns back to the source node via the same path. On the return journey, agent pops all visited node from its stack and updates the associated routing table entries (probabilities) for all nodes by using the following rules:

   - M_k (μ_d, σ_d^2, W_d) is updated with the values stored in the stack memory *S(k)*. The time elapsed to arrive (for the forward ant) at the destination ($o_{k \to d}$) is used to update the estimated mean μ'_d, variance σ'^2_d and the best value over the observation window W'_d.

   - The routing table T_k is updated by incrementing the probability of choosing neighbour f when destination is d' $P_{fd'}$ and decrementing the other probabilities $P_{nd'}$ as in equations 4.10 and 4.11. In other words, the solution is updated with a positive reinforcement signal as in equation 4.10 on the selected next node and the rest of the solutions (nodes) are updated with a negative reinforcement signal as in equation 4.11.

$$P_{fd'} = P_{fd'} + r(1 - P_{fd'}) \tag{4.10}$$

$$P_{nd'} = P_{nd'} - rP_{nd'}, n \varepsilon N_k, n \neq f \tag{4.11}$$

Observed trip time T[14] is used to calculate the reinforcement signal r (goodness measure) as the reinforcement signal as in equation 4.12. T is the only feedback that the agents receive from the network that is used to find the optimal solution.

$$r = \begin{cases} \frac{T}{c\mu}, c \geq 1 & \text{if } \frac{T}{c\mu} < 1 \\ 1 & \text{Otherwise} \end{cases} \qquad (4.12)$$

Where, c is the scale factor used to saturate "out-of-scale" values below 1. In the original antnet algorithm the following values were used for the constant variables [Di Caro & Dorigo, 98-4]:

$$\eta = 0.005, \alpha = 0.3, c = 2, \varepsilon = 0.25, \ t = 0.5.$$

Pseudo code for the original Antnet algorithm is given in Fig. 4.7 [Di Caro & Dorigo, 98-4]. Later the following improvements have been proposed by [Di Caro & Dorigo, 98-4]:

- Forward ants use high priority queues as backward ants rather than using the low priority queues as in the previous approach
- No information is kept in the ants stack regarding the delays experienced in each node
- Backward ants update the routing table entries based on the estimated ant trip times which is computed using statistical model L_k

[14] Variables used in antnet are not related with the variables used in the other algorithms.

$$L_k^i = \frac{d_l + (q_l + s_a)}{B_l} \tag{4.13}$$

Where, d_l is the link propagation delay, q_l is the total size of the packets that are waiting in the queue, s_a is the size of the current packet.

In antnet algorithm time delay from the current node is the only feedback to the agent. The performance of the antnet algorithm depends on the several static parameters that are used. These parameters need tuning using the daemon function explained in section 4.3 to tune the performance of the routing algorithm. However, it is not possible to implement the daemon function in a real network as it requires global knowledge about the network which is not available in distributed systems. Pseudo code for antnet routing algorithm is given in Fig. 4.7 [Di Caro & Dorigo, 98-4].

```
While (NOT KILL)

        If (t_interval) is reached // time interval for ant generation
        Then
                Destination_node = random (possible destinations)
                Sent forward agent (destination_node, source_node)
        End If

        // it is assumed that received packets are handled by the distributed programming
        environment and they are not being lost while processing other packets (agents).


        If (forward ant)
        Then
                If NOT (current node = destination node)
                Then

                        // Set next hop node from unvisited nodes in the routing table
                        based on the probabilistic entries.
                        Next_hop_node = Select_Probability (unvisited nodes, routing
                        table);
                        Push_stack(next_hop_node, Agent);
                        Sent_agent (Low_priority_queue, next_hop node);
                        // agent is placed to the low priority queue, if it is empty it will be
sent
                End If
                Else
                        Change angent's type to backward agent when it reaches to the
destination
                        Agent type = backward;
        Endif

        If (backward ant)
        Then
                If NOT (current node = destination node) // destination = source node in
        this case
                Then
                        Next_hop_node = pop_stack();
                Endif

                Update(M array);
                Calculate R;
                Update (Routing table);
                Sent_agent (High_priority_queue, next_hop node);
                // agent is placed to the high priority queue, if it is empty it will be sent
        Endif

End while;
// Kill is a special msg type that will be used in the simulation which can only be sent by the
Master. When kill is sent it will be placed in front of the high priority queue rather than end of
```

Fig. 4.7: Antnet routing algorithm's pseudo code

4.5.7 Explorer, allocator and deallocator ants

[White et al, 98-1] applied ant systems to connection oriented circuit switch networks. In [White et al, 98-1] ants choose their destination based on the probabilistic table and keep a data structure, named tabu list of the nodes visited. This approach prevents cycles and avoids visiting previously visited nodes. In this work, a GA has been used to optimise the constant parameters that have been used in the algorithm. They used three types of ants:

- **Explorer ants**: search for a path from source to destination
- **Allocator ants**: allocate resources from source to destination
- **Deallocator ants:** free allocated resources.

4.5.8 Ant System Genetic Algorithm

Ant colony algorithms are sensitive to the number of parameters used, such as, the number of agents and the sensitivity of the link cost. Moreover, sometimes shorter path becomes unavailable or it takes some time to discover. Therefore, in [White et al, 98-2] a GA was used to optimise the path usage and to avoid solution stacking in local optima. In Fig. 4.8, the pseudo code for ant system genetic algorithm (ASGA) is given [White et al, 98-2].

```
Initialize Population
For i = 1 to Iterations
Begin
        For j = 1 to population size
        Begin
                Use_agent_to_solve_problem (j);
        End
        For j = 1 to population size
                Reinforce_path_followed_by_agent (j);
        End
        Generate_agent_parameters ();
End
```

Fig. 4.8: Pseudo code for ASGA Algorithm

4.5.9 Improved Antnet Routing Algorithm

[Baran & Sosa, 00] used intelligent initialisation of probabilistic routing tables with deterministic values. In the initial states of the original approach, software agents are forwarded based on the uniformly distributed probabilities without taking into account the initial state of the network. However in [Baran, 01] routing table entries for the neighbouring nodes are initialised with higher probability values. Although this improves the performance of the algorithm in the initial states, in the long term its effect on performance is negligible. It has been further shown that, when a link or node failure occurs, the probability corresponding to the failing node is distributed equally amongst the others. [Baran, 01]'s approach seems to improve the algorithms response to the failures in a short time period, but over longer period the gain in performance improvement is negligible. [Baran & Sosa, 00] further suggested limiting the number of ants in the system by four times the number of links. The first problem with this scheme is that this criterion can only be available globally and as the network evolves it is obvious that number of links would not stay the same. Moreover, although restricting the number of ants improves the performance of the algorithm and reduces the overhead that may be created by the ants in the network, it reduces the convergence of the algorithm. A simple scheme used in [Tekiner et al, 04-3] to limit the number of ants where number of ants are depend on the number of packets in the system at any given time. In general a large number of ants collecting information about the status of the network mean faster convergence while consuming more system resources. This is the main dilemma in reinforcement learning as has been discussed earlier in this chapter. Lastly, [Baran & Sosa, 00] have used a deterministic approach in choosing the next hop. However, it has been pointed out that this might lead to a possible infinite looping and congestion in the optimal route

[Sim & Sun, 03]. This congestion has also been observed in a similar approach adopted in this thesis.

4.5.10 Limiting number of ants

Although, [Baran & Sosa, 00]'s approach on limiting the number of ants is proved to have a significant impact on the performance of the algorithm. The main drawback is that it uses some constant values for limiting the number of ants that can be different on every network. [Akon et al, 04] proposed a timeout approach to control ant populations in a dynamic manner. Whenever an ant is generated at the source node *s* to destination *d*, a timeout value t_1 is also been set. If the created ant fails to return back to the source node for whatever reason, t_1 is used to regenerate a new ant to the destination with the same value. Otherwise, on successful return t_1 is updated as t_2 based on the reinforcement value *r* scored and ant is sent back to destination (equation 4.14) [Akon et al, 04]

$$t_2 = T_b x\left(b_1 - \frac{b_2}{1+e^{-b_3 r}}\right) \tag{4.14}$$

Where, b_1, b_2, b_3 and T_b are system parameters[15].

By using the above formula, the number of ants is kept low when it is not needed. However, this reduces the probability of discovering changes in the network, since there will be a reduced number of ants to collect up-to-date information. In addition, in [Akon et al, 04] backward ants updates the timeout values in the intermediate nodes based on the reinforcement value on its way back to the source node.

[15] $b_1 = 2.5$, $b_2 = 2$, $b_3 = 100$ and T_b is set to ant creation rate.

4.5.11 Multiple ant colonies

The idea of using multiple ant colonies was first applied to wavelength routing in [Varela & Sinclair, 99]. In this work, more than one ant colony is used to distribute the different wavelengths over the entire network to achieve increased availability for the required wavelength at any given time. In the original antnet, ants are only attracted by the pheromone trails laid by the ants from their own species. In addition to this, in [Varela & Sinclair, 99] ants are repelled by the pheromone laid by other species. Therefore, routing decision implementation is based on more than one routing table. Three variants of the algorithm have been proposed namely: local update, global update/distance and global update/occupancy. In the local update table entries are updated in every hop whereas, in the global updates, table entries are updated at the end of each cycle[16]. Also, two variants of the global update have been used where two different repulsion techniques are employed to distribute the wavelengths based on the distance or the link occupancy.

In [Sim & Sun, 02] the idea of multiple ant colonies has been applied to the antnet routing algorithm in circuit switched networks. In [Sim & Sun, 02] it is assumed that there exists more than one ant colony. Let us say that, there are two ant colonies existing in the network, named as red and blue, where each use different pheromone tables. For example, red ants only update the pheromones laid by the red colonies and so on. By using two colonies, more than one optimal path becomes available for every source destination pair. Therefore, higher load balancing is achieved. Unlike the approach in [Varela & Sinclair, 99], where ant colonies repel each other, in [Sim & Sun, 02] no information on whether ant colonies repel or attract each other is given.

[16] An ant is said to complete its cycle when it completes it journey from source node to the destination node.

To date the application of multiple ant colony algorithms to the packet-switched networks has not been reported, and consequently there is no performance evaluation.

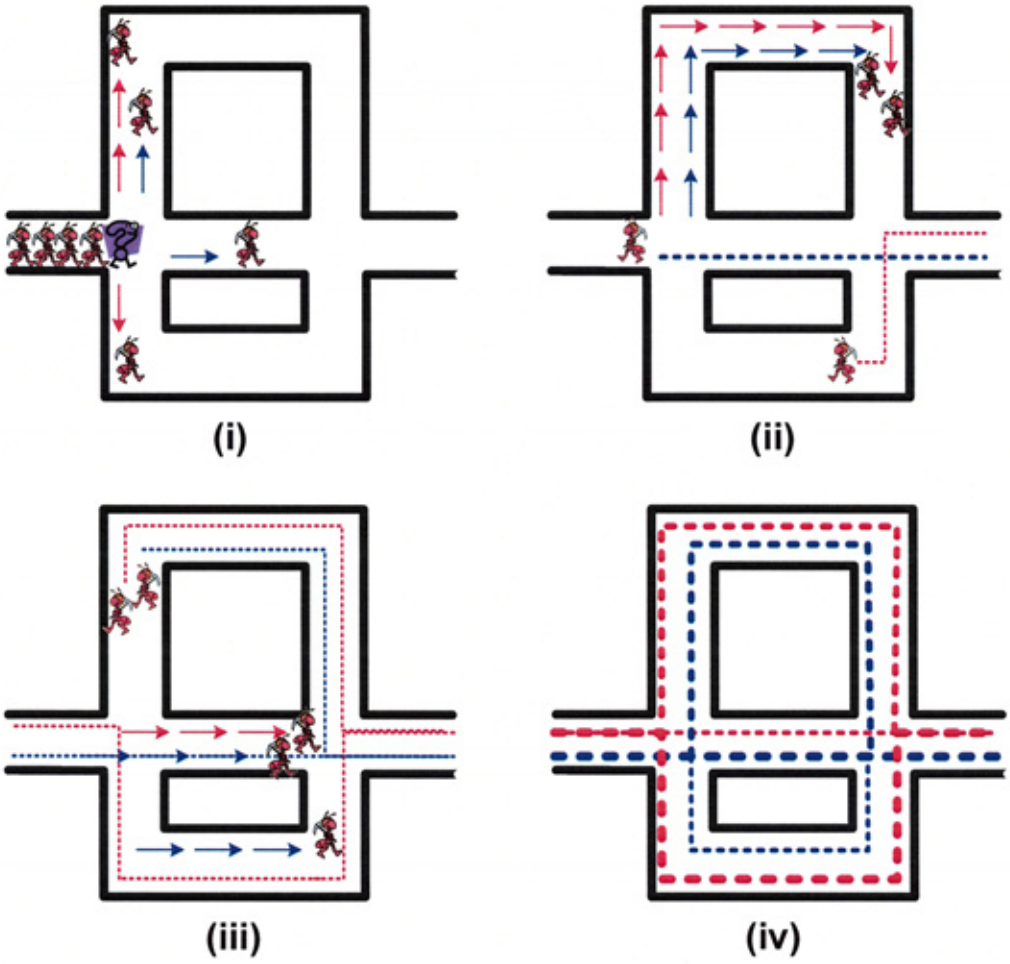

Fig. 4.9: Multiple Ant Colonies

In Fig. 4.9 there are two ant colonies red and blue. In Fig. 4.9(i) ants from both colonies do not know where to go initially as there is no pheromone laid from both colonies. They both make unbiased decisions and randomly choose one of available three paths. In Fig. 4.9(ii) the first pair reaches to their destinations and follow the path that they came from to return back to the nest. It can be seen from Fig. 4.9(iii) that some pheromone is laid on most of the paths by both of the colonies but not

equally. Finally, Fig. 4.9(iv) shows the pheromone laid by both of the colonies. It can be clearly seen that there is a good balance in the distribution of pheromones. Therefore, the load balancing of the network is very high. The middle path which is also the shortest path is dominated by the blue colonies hence with very little amount of pheromone laid by the red colonies. The second shortest path, which is the path below the graph, is dominated by the red colonies. Therefore, red colonies avoid the congested middle path and the upper path is equally shared by both colonies in this scenario. One can say that by using multiple colonies, improved load balancing can be achieved and hence improved performance. To date, no application of multiple ant colony optimisation (MACO) is reported in packet switched networks. In Chapter 6, we will apply MACO to packet switched networks and compare the results with existing approaches.

4.5.12 Other Ant-Agent Based Approaches

A number of researches proposed hybrid versions of ACO meta heuristics. Multi AGent Metaheruistics Architectur (MAGMA) is one of these approaches where agents acts in a multi-level architecture, each level corresponding to a different level of abstraction [Roli & Milano, 02]. There are four levels in MAGMA and there are four kinds of specialised agents each of them running a different algorithm. Agents communicate with each other in two ways, horizontally, by communicating with the agents on their level only and vertically by communicating with the other agents on different levels. This communication is very flexible and any kind of algorithm and protocol can be used within MAGMA. Different meta-heuristics as well as ant colonies have been embedded and tested in MAGMA [Roli & Milano, 02]. [Di Fatta et al, 00] applied ant systems to packet based active networks. In this approach they

used a dynamic congestion metric by taking into account delays experienced in the routers in addition to the transmission delays.

[Li et al, 00] used ant systems on circuit switched telecommunication networks. In this approach on detection of a cycle an ant dies rather than removing the cycle from its stack. [Guoying et al, 00] applied AS to real time multicast routing networks. [Michalareas & Sacks, 01] used deterministic rules for the data packets and probabilistic rules for the ants travelling in the network as in the antnet routing. [Matsou & Mori, 01] used accelerated ants to improve the convergence speed to improve network performance. Ants are accelerated by allowing every ant to update more than one entry at a time in the routing table, based on the routing history unlike antnet, where an ant could update only one entry. [Chu et al, 02] applied ant systems to the multicast routing problem to meet quality of service (QoS) requirements. In [Gunes et al, 02] ant systems are applied to multihop wireless networks to reduce the overhead for routing. [Kassabalidis et al, 01] reviewed and implemented the antnet routing algorithm in their limited work. [Botee & Bonabeau, 99] used traditional GA's to tune the constant parameters used in the antcolony optimisation applied to the TSP problem. [Liang et al, 02] applied ant systems together with GA to packet routing networks. [Liang and Zincir-Heywood, 02] investigated the performance of the antnet routing algorithm and simulated it on an event driven C++ simulator.

4.6 Agent Distance Vector Routing

Agent distance vector routing (ADVR) [Amin et al, 01] is another algorithm that has been inspired by the ants. ADVR is a hybrid routing algorithm that uses distance vector routing together with agents. In ADVR, the aim is to use network resources

efficiently. Therefore a hybrid approach has been employed to agent migration; depth-first-search method is used together with the ant system. In depth-first-search [Amin et al, 04] agents exchange their migration history with the other agents that are visiting the same node. This is to ensure efficient utilisation of the network resources by avoiding the transfer of the same information among the same nodes. However, the main drawback of this method is that, after some period an agent starts making the same decision that results in clusters of agents doing the same thing. Also, in [Amin et al, 01] it was argued that increasing agent populations increases utilisation of network resources, although more agents mean higher parallelism. In order to find the optimal number of agents (best agent population) [Amin & Mikler, 02-2,04] proposed a dynamic agent population solution. In this approach, an agent that has not updated a table entry for a long time is allowed to die, whereas a busy agent is allowed to clone itself on the auxiliary agents.

4.7 Q-Learning

Q-learning is a model free algorithm that learns from the delayed reinforcements and it is one of the easiest approaches to implement in reinforcement learning, thus one of the most popular [Sutton & Barto, 98]. Q-learning is applied to the routing problem in the Q-routing algorithm, where routing table in the distance vector algorithm [Sutton & Barto, 98] is replaced by the table of estimations (Q values) based on the link delay. In Q-routing the same routing policies are used as in the distance vector routing algorithm. The Q-function is used to update routing table entries in Q-routing [Boyan & Littman, 94]. However, it has been suggested that neural networks can be used for approximating the Q-function by incorporating diverse parameters of the network such as time delays and queue lengths [Boyan & Littman, 94].

4.7.1 Q-routing

In Q-routing, when x sends a packet to the node d via its neighbour y, it receives y's estimated remaining trip times to the destination d. Then it selects the neighbour with the smallest time (equation 4.15). Q-routing uses forward exploration to update the Q values in the routing table (named as Q-table) that represents the delivery time estimation to the given destination (equation 4.16) [Boyan & Littman, 94].

$$Q_y(z',d) = \min_{z \in N(y)} Q_y(z,d) \tag{4.15}$$

$$\Delta Q_x(y,d) = \eta_f (Q_y(z',d) + q_y - Q_x(y,d)) \tag{4.16}$$

Where, $\Delta Q_x(y,d)$ is the new estimation value for node x to destination d via the neighbour node y (Fig. 4.10). This new estimation is calculated by subtracting the old estimation value $Q_x(y,d)$ from the sum of the estimation time for packets travelling from node y to destination d via neighbour z ($Q_y(\acute{z},d)$) and current queue delay for the packet in node x (q_x). η_f[17] is the learning rate parameter defined by the programmer [Tekiner et al, 04-4]. In Fig. 4.10, x updates its $Q_x(y,d)$ value pertaining to the remaining path of packet p via node y.

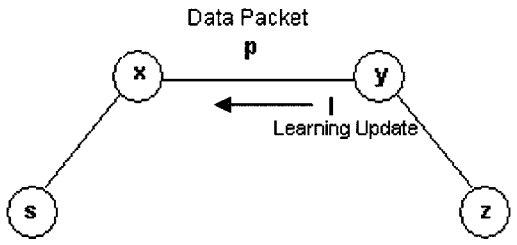

Fig. 4.10: Forward exploration

[17] η_f is set to 0.7 in original Q-routing algorithm [Boyan & Littman, 94].

4.7.2 Dual reinforcement Q-routing

Backward exploration together with forward exploration is applied in dual reinforcement Q-routing (DRQR) algorithm in order to improve the learning rate of the Q-routing algorithm [Kumar & Miikkulainen, 97]. In DRQR, exact delay values learnt from the backward learning have also been used in the routing tables in addition to the estimation values learnt from forward learning in Q-routing. This has doubled the learning information available in the algorithm thus improved the learning rate of the algorithm [Kumar & Miikkulainen, 97].

In DRQR algorithm, when x sends a packet to node y to get its estimated remaining trip times, y also gets x's estimated trip times for its link with s. In Fig. 4.11, a packet at node x arriving from source node s is sent to node y, also carries the estimated time that it takes from node x to s, $Q_x(\acute{z},s)$ (equation 4.17). With this information node y updates its own estimate $Q_y(x,s)$ for the entry node x associated with the destination s (equation 4.18). Therefore, in DRQR both backward and forward exploration can be used to update the Q entries. However, this adds an overhead to the packet and to the algorithm.

$$Q_x(z',s) = \min_{z \in N(y)} Q_y(z,s) \tag{4.17}$$

$$\Delta Q_y(y,s) = \eta_b(Q_x(z',s) + q_y - Q_y(x,s)) \tag{4.18}$$

In Fig. 4.11, during the forward exploration, the node sending the data packet (node x) updates its $Q_x(y, d)$ value pertaining to the remaining path of packet p via node y. However, during backward exploration, the receiving node y updates its $Q_y(x,s)$ value

pertaining to the traversed path of packet p via node x. The main drawback of both the Q-routing and the DRQR approaches is that, although they are capable of detecting node and link failures, they are incapable of restoring them.

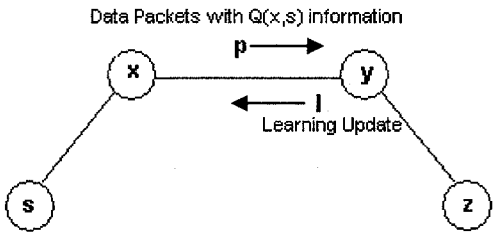

Fig. 4.11: Forward and backward exploration.

4.7.3 Confidence based dual reinforcement Q-routing

In confidence based dual reinforcement learning (CDRQR) a "confidence" parameter is used to control the reliability of the enforced signal [Kumar & Miikkulaine, 02]. When a Q-routing entry is not updated for a long time it does not give a true estimate of the associated link. Thus, one can say that it is not reliable. In order to avoid this, a dynamic learning parameter is used in CDRQR. In addition to the Q-table used in the original Q-routing algorithm a C-table ($C_x(d,y)$) is also used. $C_x(d,y)$ defines the confidence parameter, a number between 0 and 1, for the path from node x to node d via neighbour node y. A value of 1 means the Q-table entry represents the current state of the network that has just been updated. On the other hand, value 0 means that the corresponding link has never been used before and updated.

In CDRQR, a node is provided with the estimate for the remaining trip time and the associated confidence value, and uses them to define the learning parameter. In other words, the learning parameter used in CDRQR is dynamic and changes based on the

confidence parameters. The Q-estimate is calculated in CDRQR as [Kumar & Miikkulaine, 02]:

$$\Delta Q_x(y,d) = \eta_f(C_{old}, C_{est})\left(Q_y(z',d) + q_y - Q_x(y,d)\right) \tag{4.19}$$

The learning rate function $\eta_f(C_{old}, C_{est})$ is defined as:

$$\eta_f(C_{old}, C_{est}) = max(C_{est}, 1 - C_{old}) \tag{4.20}$$

According to [Kumar & Miikkulaine, 02] learning rate should be **high** if the confidence in the old Q value is **low** and/or the confidence in the estimated Q value is **high**.

In order to provide more reliable estimates, the confidence values are given by:

$$C_{upd} = \begin{cases} \textbf{a:} & \lambda C_{old} \\ \textbf{b:} & C_{old} + \eta_f(C_{old}, C_{est})(C_{est} - C_{old}) \end{cases} \tag{4.21}$$

If the Q values associated with the links have not been updated within the last time step, every C value decays with some λ as in equation 4.21(a). Otherwise, if the Q value is updated in the last time step, the associated C values are updated as in equation 4.21(b).

4.7.4 Predictive Q-routing

Predictive Q-routing (PQR) is a memory-based Q-learning algorithm to improve the performance of the Q-routing under low loads [Choi & Yeung, 96]. PQR uses the memory to keep the best experiences learned and reuses to predict the traffic model of

the network. PQR, therefore, distributes the network load evenly to achieve better performance in terms of delivery times. In PQR in addition to Q-tables (equation 4.15) three more tables are kept in the nodes. The B-table, $B_x(d,y)$, represents the best estimated delivery time from node x to node d via neighbour node y. When a path is congested the algorithm diverts the traffic through other nodes. Then, eventually this path becomes available again. The R-table, $R_x(d,y)$, is used to keep the recovery rate of this and represents the path from node x to node d via neighbour node y. The U-table, $U_x(d,y)$, keeps track of the last update time from node x to node d via neighbour node y. In Fig. 4.12 the pseudo code for PQR is given [Choi & Yeung, 96].

Table Updates (when a packet arrives to a node)

$$\Delta Q_x(y,d) = \eta\,(Q_y(z',d) + q_y - Q_x(y,d))$$
$$B_x(y,d) = min\left(B_x(y,d), Q_x(y,d)\right)$$
if($\Delta Q_x(y,d) < 0$)
$$\Delta R_x(y,d) = \Delta Q_x(y,d) / \left(current\_time - U_x(y,d)\right)$$
$$R_x(y,d) = R_x(y,d) + \beta \Delta R_x(y,d)$$
else if($\Delta Q_x(y,d) > 0$)
$$R_x(y,d) = \gamma R_x(y,d)$$
$$U_x(y,d) = current\_time$$

Routing Policy (to send a packet from node x to y)

$$\Delta t = current\_time - U_x(y,d)$$
$$Q_x(y,d) = \max\left(\left(Q_x(y,d) + \Delta t R_x(y,d)\right) B_x(y,d)\right)$$
$$y = \arg\min_y \left\{Q'_x(y,d)\right\}$$

Fig. 4.12: Pseudo code for PQR

In PQR there are three learning parameters used.

1. η is the original Q-function learning parameter that is used in Q-routing and it is set to 1.
2. β is the learning parameter for the recovery rate and is set to 0.7.

3. γ is used to control the decay in the recovery rate where this value is usually greater than β and set to 0.9.

4.8 Conclusions

This chapter presented an introduction to reinforcement learning and ant colony optimisation, followed by a detailed review of antnet routing algorithm and its evolution. Finally, a review of the well known Q-routing algorithm and its derivatives has been given.

Ants uses probabilistic routing tables whereas in link state and distance vector routing table entries are deterministic. Also, ants use less resources on the nodes compared to these algorithms. Moreover, ants are dynamic and self-organising systems whereas distance vector and link state algorithms require human supervision. Q-routing does not guarantee finding the shortest path always. Moreover, it can only find a single path, but cannot explore multiple paths. In Q-routing, routing is proportional to the data traffic. Therefore network emerges lower in low data traffic hence routing algorithm works slower. This is not a desired property as a routing algorithm should perform the same under all traffic conditions. Dual reinforcement routing uses forward learning together with backward learning with a considerably small overhead. CDRQR algorithm uses confidence parameter to make the Q-routing algorithm more reliable. On the other hand, PQR algorithm uses memory to improve the load balance of the network, thus the performance of the network. Implementation and the modifications proposed for the antnet routing algorithm will be given in Chapter 6.

CHAPTER V - IMPLEMENTATION AND EVALUATION OF LOGICAL NETWORK TOPOLOGIES

5.1 Introduction

This chapter is a study of a variety of network performance issues, design, implementation and evaluation for various synchronous logical network topologies and their routing algorithms. Particularly, routing algorithms for these networks will be simulated and tested as in a fully connected network. In order to test the algorithms, a generic implementation framework has been developed for synchronous logical network topologies. For simulations, all nodes create a packet with a random destination based on the defined system load, and place it on one of its output ports. All nodes execute the same algorithm and make routing decision in order to forward packets to the related output port. The algorithm runs for specified number of loops (iterations) and stops execution after reaching specified numbers of iterations.

The remainder of this chapter is organised as follows: In section 5.2 the testing criteria and the transmission/reflection ratio (TRR) are defined. This is followed by the node model incorporated in a particular network topology. A design and routing algorithm framework based on TOAD are covered in section 5.4. Implementation of the network and routing algorithms are presented in section 5.5. The test environment is described in section 5.6. Finally, an evaluation of Shufflenet, Gemnet and De Bruijn graphs is given in section 5.7.

5.2 Testing Criteria

The following test criteria have been selected to evaluate the performance of the routing algorithms for different network sizes under different system loads:

1. **Average number of hops**
2. **Contention percentage**
3. **Node idleness percentage**
4. **Transmission-reflection-ratio**

TRR is used to evaluate the proposed priority rule (see section 5.3) based on the routing algorithms that can make use of the available multiple paths. Also, each network will be compared with deflection routing (no-buffer) and store-and-forward routing (with one-level buffer) configurations. It has been shown that there is no performance gain in the system when employing multi-level buffering [Maxemchuck, 93]. Tests carried out with more than one level of buffer have also proven this point. This is because, when the buffer size is increased there are more packets at the system at any given time, thus more congestion. Based on the deflection and store-and-forward routing strategies, two types of nodes are used in the tests, with/without buffering (one level). Furthermore, a multilevel buffer requires buffer management algorithms [Hunter et al, 98], which are beyond the scope of this thesis. In this work the following are assumed: (i) one level of buffer is either full or empty, (ii) in the event of congestion a packet is stored in the buffer if it is not full, and (iii) a packet stored in the buffer is released within the next iteration as it has priority over the packet in the node's inputs.

Algorithms will be evaluated for their routing capabilities and adaptability for optical networks in particular all OTDM. Simulations will be carried out for a varying number of nodes and different loads (theoretically this system should work for up to infinite number of nodes based on the network configuration). As the number of nodes increases, the number of packets wanting to reach their destinations will also increase, then resulting in packet congestion. In order to verify this, the number of packets, the number of congestions, the percentage nodes idleness for all iteration and the TRR (see section 5.3) will be compared for a varying number of nodes (i.e. 24, 32, 64, 128, 256, 384, 512 and 1024 nodes). In addition, the impact of the request load on the performance of the algorithms will also be investigated, as different algorithms have different behaviours on different load levels. Parameters that will changed during simulation are the number of nodes, the number of messages created per test, the total number of hops, the total number of times a node was idle, the total number of times an output port is used (transmission and reflection) and the total number of congestion.

(i) The performance analysis and best/average/worst case comparison of the algorithms is determined according to the following:

Average number of hops per packet $N_{h\text{-}p}$ from source to destination is defined as

$$N_{h-p} = \frac{N_T}{N_H} \tag{5.1}$$

Where N_T and N_H are the total number of packets and hops, respectively.

(ii) Percentage of the idle of nodes in the network q is given as:

$$q = \frac{N_q}{N \times I} \times 100 \quad (\%) \tag{5.2}$$

Where N and I are the Shufflenet size and the total number of iterations, respectively. q is an important factor with regards to utilization of the network and N_q is the total number of idle nodes.

(iii) Contention occurs when two or more packets are destined for the same output port. Congested packets are either buffered or deflected to the next available port. The contention ratio c′ which measures this is defined as:

$$c' = \frac{N_c}{N_T} \tag{5.3}$$

Where N_C is total number of contentions that occurred. This is an important parameter as the number of contentions occurring within the system has direct effect on the number of hops taken by a packet, which is the main performance measure.

5.3 Transmission/Reflection Ratio

Fig. 5.1 shows a logical node structure of a TOAD based 2 × 2 routers that will be used as the node model in the simulations [Gao, 03]. As shown in Fig. 5.1 packets can be inserted (added) or dropped at the node on request and of course on availability of output port. Previous studies have shown that in TOAD based routers the crosstalks observed at the output ports are not the same. Crosstalk observed at the reflection port is slightly higher compared to the transmission port [Gao, 03].

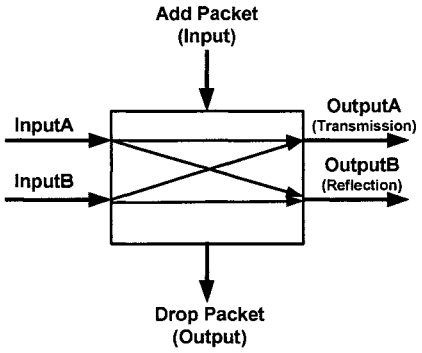

Fig. 5.1: Node model

Therefore, the implication of asymmetrical crosstalk at the output port in a network environment would be to select routes, which avoid the reflection output ports as much as possible in order to minimise the overall crosstalk observed by a packet when reaching its destination. TRR is defined as:

$$TRR = \frac{N_{TR}}{N_R} \tag{5.4}$$

Where N_{TR} and N_R are the total number of transmissions and reflections, respectively. TRR for each packet should be maximised in order to ensure that packet reaches the destination with the least amount of crosstalk. In the best-case scenario, only the path utilising transmission ports is used. In the worst-case scenario TRR is equal to zero since a path that goes through all the reflection ports has been selected.

As discussed previously, each node is a 2 × 2 TOAD router with two output ports, with the reflection port (Output-B) having higher crosstalk than the transmission port (Output-A). In deflection routing if two packets are destined to the same output port, then one of them is deflected. Whereas, in store-and-forward routing if one-level of buffering is full, then the packet is deflected to the available port. As shown in Fig.

5.2, packet taking the worst path (all reflection ports) will experience increased crosstalk than those taking the best path. The best-case scenario is to select a path that links all the transmission ports until the packet reaches its destination. To achieve this, a priority scheme has been introduced, where packets going through the transmission port have higher priority over the reflection port. A packet received by a node can be routed arbitrarily to one of the two output ports without affecting the path length to the packet destination. However, when the packet is TYPE-M (see Chapter 2), nodes attempt to avoid the reflection port and instead use the transmission port. In doing so, packet TRR is increased. In Shufflenet, TRR also reflects the output port utilisation. However, introducing the priority scheme decreases the output port utilisation dramatically, thus resulting in increased network congestion [Hluchyj & Karol, 88].

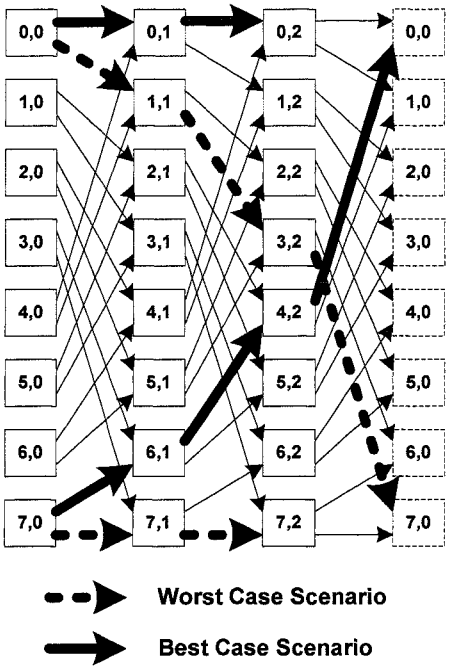

Fig. 5.2: Shufflenet showing possible worst and best paths using prioritised routing

5.4 Design

In developing all the algorithms that have been reviewed in the literature and the proposed algorithms in this thesis a number of assumptions and conditions for the interconnection networks have been made, which are as follow:

- All nodes in the system have a unique address
- All nodes within the system are fully connected
- Packets are processed synchronously by delaying them at the input ports so that they arrive at the node at the same time
- Packets are delayed while the node is processing the header containing the destination information
- Each node is identically the same (i.e. processing power, architecture, routing algorithm...)
- Packet size is fixed through out the simulations
- There is no packet loss due to the congestion as packets are deflected when there is no buffer available
- There is no packet loss due to the loss in the transmission (it is assumed that packet restoration algorithm runs in the node)
- The packet in the buffer has priority over the packets in the input ports
- The existing packet in the system has priority over the new packet that is about to inserted to the system. It is assumed that there is a buffering and control mechanism for the new packets.

5.4.1 Packet format

The packet format is assumed to be fixed, composed of a header and payload as shown in Fig. 5.3. The header carries the information required for the packet routing algorithm. It is composed of the packet age (number of hops experienced by packet), a unique packet identifier and the packet destination address. It is assumed that the packet is delayed while the header is being processed at each node. Although it is assumed that there is sufficient time for header processing, packet header content should be kept to a reasonable size in order to ensure faster processing, thus increased throughput. In practical systems, if a packet cannot be processed within the given time (in the dedicated time slot) then it is regarded as a lost packet, since there is no mechanism to store partial routing information. The packet payload carring the data has no effect on the routing decision.

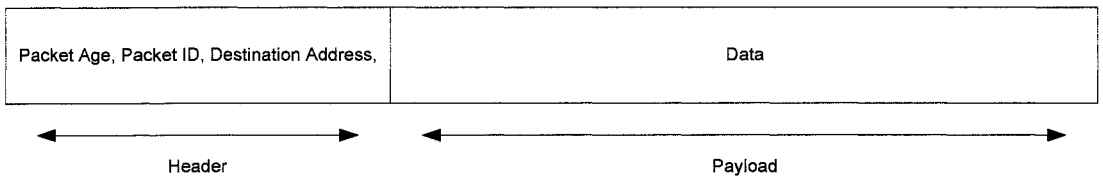

Fig. 5.3: Packet format

5.4.2 Network routing algorithm framework

A framework for the routing algorithm implementation and the simulation is given in Fig. 5.4. This Meta model is based on the information gathered from the review of the networks and the routing algorithm properties and will be used as the base model for implementing the routing algorithm of the selected networks. The design and implementation process are broken down into 8 main steps with each step representing and identifying certain aspects of the simulation as outline below:

STEP 1- Initialization: This step is the start-up process of the algorithms. Each node clears its output ports, buffers and all the variables.

STEP 2- Check if packet has reached its destination: Node checks the destination address of the packet to see whether the packet reached to its destination or not. A packet is dropped (deleted and counter incremented) if the current node is its destination.

STEP 3- Find the node connections: Every network has a different connectivity pattern. Based on the connectivity pattern, a node defines its actual connections, which are mapped to the node's output ports.

STEP 4- Routing algorithm: The node that makes the routing decision (i.e. assigning the output port) uses all the information contained within the header. This is the ***"heart"*** of the simulation, where different strategies and rules have been used to handle the packets that directly affect the performance of the algorithm.

STEP 5- Copy packets from input to output port: Based on the information and decision taken in STEP 4, the packet is copied from the input port to the selected output port. In the case of contention (if there are more than one packets that are forwarded to the same port), only one packet is granted access to its preferred output port whereas the rest of the packets are deflected or buffered.

STEP 6- Create new packet: Node creates a new packet and places it on one of its output ports (if empty). Packets are created based on the probability of packet creation rate. This probability is also used to define the system load.

STEP 7- Copy packets from output to the next node's input port: Node copies the packet at its output port to the next node's input port based on the node connections defined in STEP 2.

STEP 8- Display current network status: Each node is displayed in the logical network structure with packets in their input ports. This is used to track packets, hence, it can be used to verify the correctness of the algorithm.

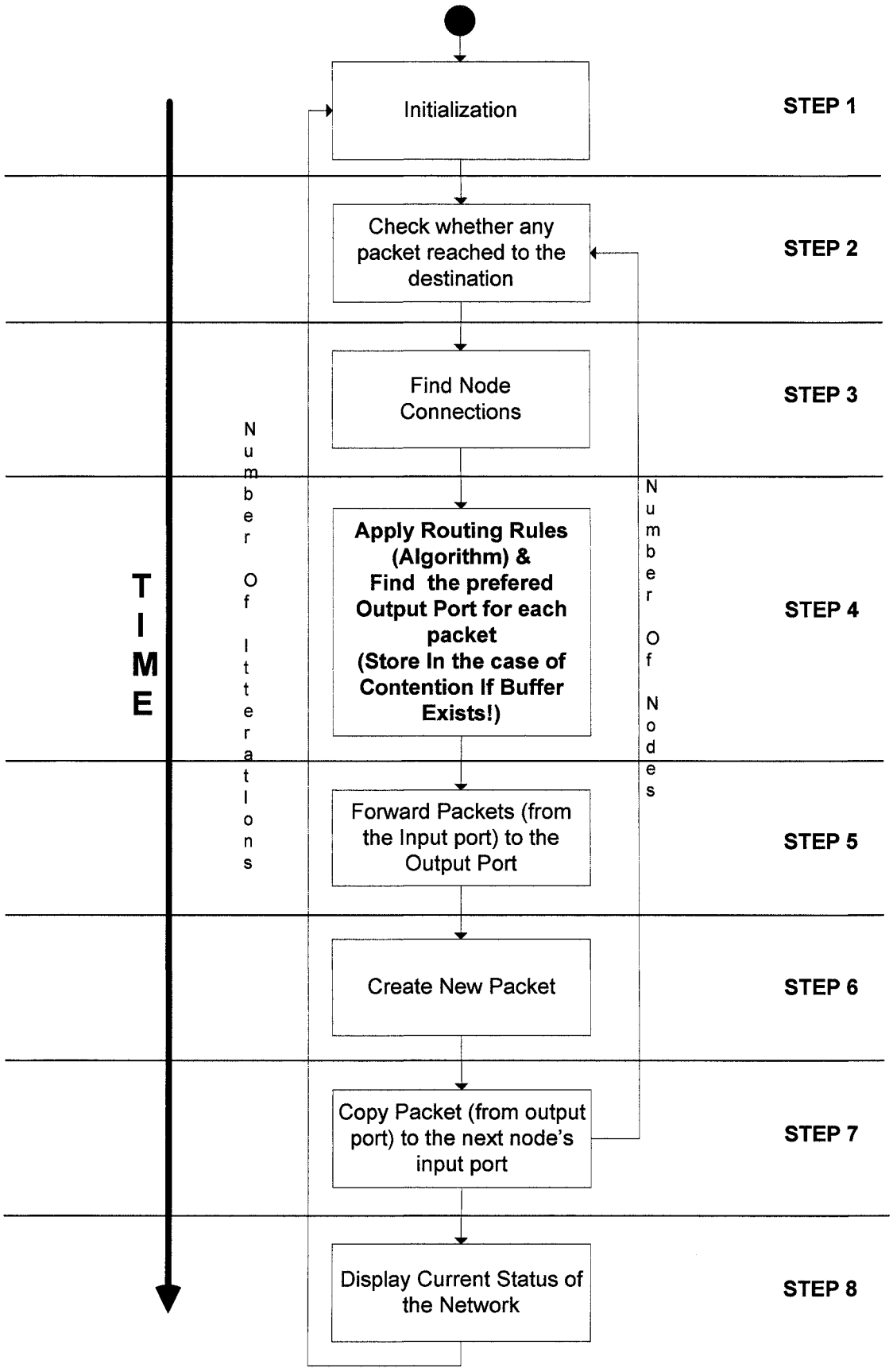

Fig. 5.4: Interconnection network routing algorithm framework

STEP5 and STEP7 could be combined into one step by copying packets from the input port of one node to the input port of the next node. However, in order to guarantee mutual exclusion packets must first be copied to the node output port. After all nodes process the packets in their input ports, packets can then be copied to the next nodes. Otherwise, the packet that is copied to the input port overwrites the existing packet that has not been processed.

5.5 Implementation

This section will describe the implementation details of the simulation according to the system design highlighted in the previous sections. The program structure will be presented. Implementation of the networks and algorithms are based on the eight steps defined by the design model as in the previous section except for the first four steps given as follow:

STEP 1- Initialisation: Each node initialises (clears) the variables and converts the source addresses (source address is node itself) into a binary format.

STEP 2– To check if a packet has reached its destination: If packet has reached its destination then node deletes the packet in its input port and increments its sent_out counter.

STEP 3– To finds the node connections: Once the initialization step is finished the packets at input ports are checked to see if they have reached their destination. The node then checks to see if there is a packet in its input port. On detecting a packet, the

node then converts the packet's destination address into binary. Finally, node finds the node connection based on the interconnection pattern of the network.

STEP4 - Routing algorithm: Following STEP 3, node applies the routing algorithm in order to find packet type and packets output port preference. Since routing decision is specified in a distributed manner, then every node re-routes the packet at every hop.

The implementation of proposed topologies is best illustrated in Figs. 5.5 and 5.6. Networks are implemented with/without buffers based on two different contention resolution schemes. For the store-and-forward routing implementation only Fig. 5.5 applies. Thus, making the deflection routing configuration implementation much simpler and less complex. There are four different scenarios that might occur if there is no packet in the buffer or if buffer does not exist; not valid inputs A and B, valid input A valid and not valid input B, not valid input A valid input B, and valid input A and B. Note, valid input means there is a packet in the input port.

In the first scenario where there are no packets in the input ports, the node checks *randomvar* which defines the packet creation rate and if it is true then it creates a new packet and places it randomly to one of the output ports. Otherwise, it continues with the next step. In the second and third steps, if there is only one packet then node places it to the preferred output port (after executing the routing algorithm) and checks *randomvar* to create a new packet and places it to the empty output port.

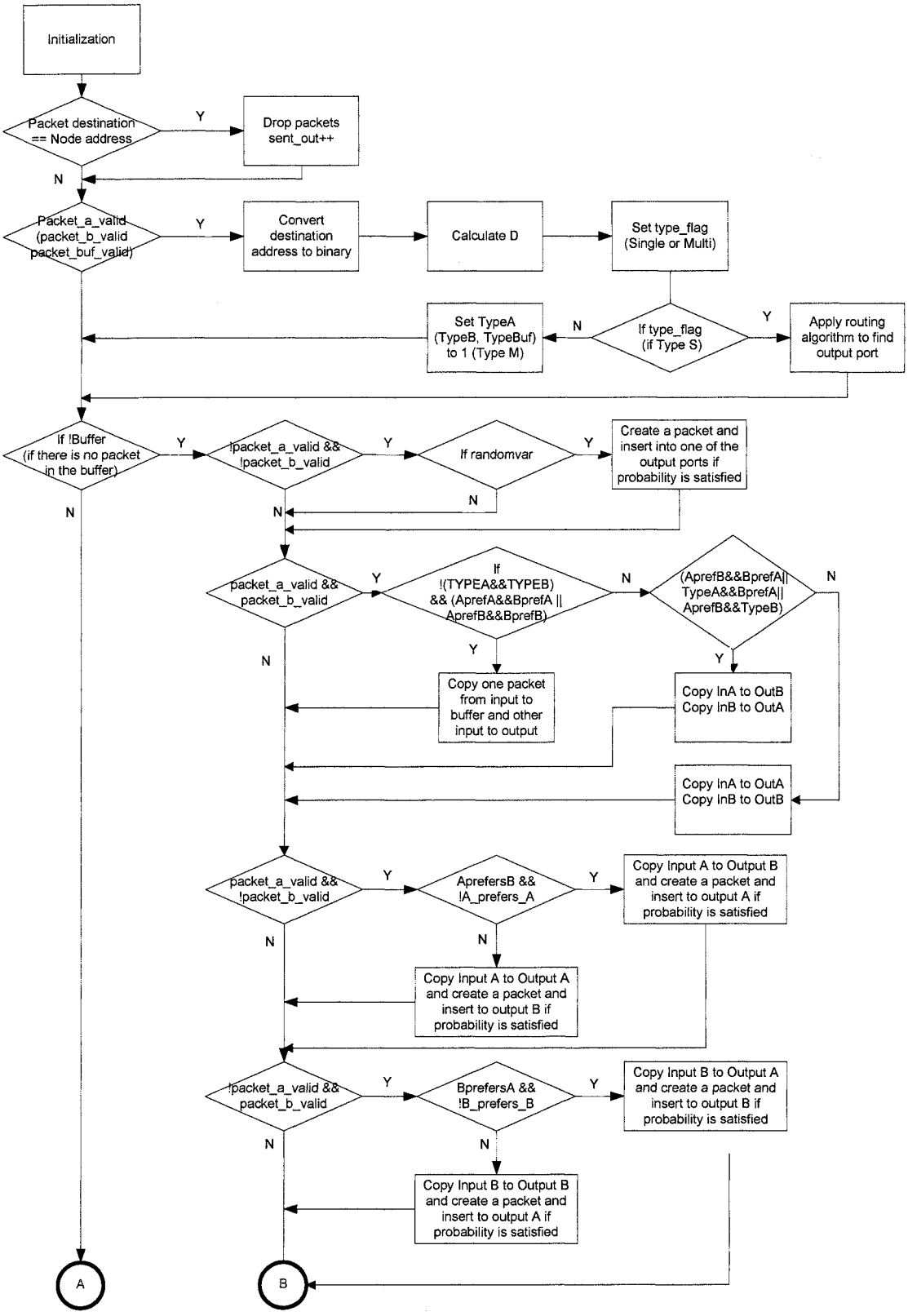

Fig. 5.5: Implementation flowchart

In the last step, there are two possibilities, either contention occurs or packets are forwarded without contention. For the case of contention one of the packets is placed in the buffer and other is passed to the preferred output port. However, if there is no buffer in the system, one of the packets is forwarded to the preferred output and the other is deflected to the other output port. In routing algorithm fairness is satisfied as follows;

- If both packets prefer to go to the output A then packet in the input B is deflected or buffered and,
- If both packets prefer output B then packet in the input A is deflected or buffered.

With the store-and-forward routing, in some cases, there might be three packets at a node that prefer to go to the same output port (packet in buffer, inputs A and B). In this case it is assumed that a packet in the buffer has priority over packets at the input ports. Therefore there are four different cases, as in the deflection routing configuration, which are: Case 1; node checks the existence of a packet in the buffer, and on detecting one it forwards it to the preferred output port and places a new packet on the other port. Otherwise, node creates and places a new packet in any of the output ports depending on the *randomvar*'s state. Cases 2 and 3; if contention occurs, packet in the buffer is forwarded to the output port and the packet in the input port is forwarded to the buffer. Otherwise, both packets are forwarded to the preferred output ports.

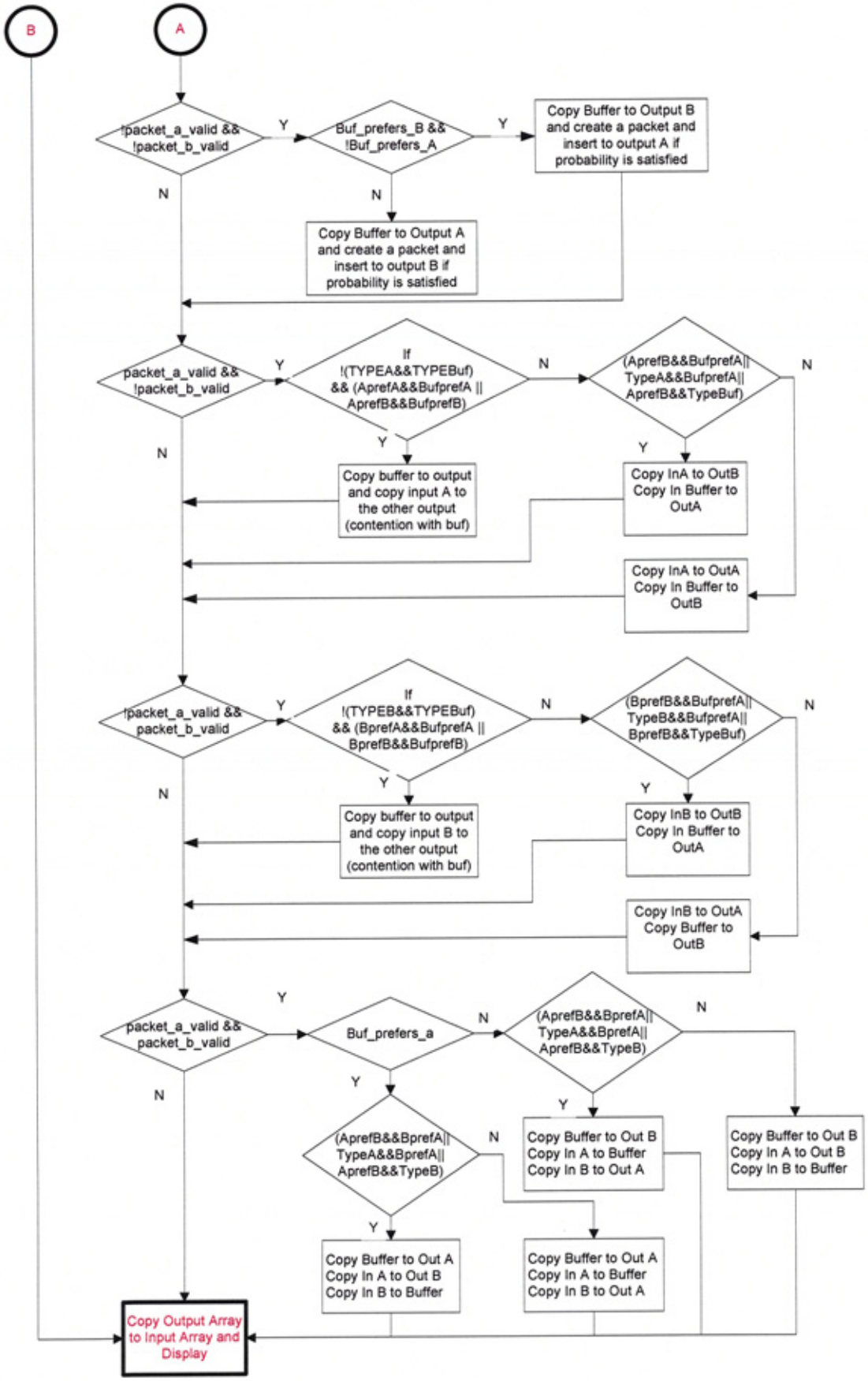

Fig. 5.6: Implementation flowchart with buffer

Case 4; since there are three packets in the system, regardless of the preferences of packets, one of them, in the inputs, is forwarded to the buffer. In the worst-case scenario, three of the packets are forwarded to the same output port and three of them are single path packets. In this case, only packet within the buffer is forwarded to the preferred output port and other two are deflected or buffered. Fairness is assured on forwarding the inputs packet randomly as in the deflection routing case.

STEP 5 – Copy packets from the input ports to the output ports: This step takes place within the STEP 4 whilst allocating the output ports to the packets. STEPs 5 and 7 are done in two different steps in order to use the resources in a mutually exclusive manner.

STEP 6 – Create new packet: If there is an unallocated output port, a new packet is created and placed at it based on the traffic model (see STEP 4).

STEP 7 – Copy packets from the output port to next node's input port: After processing all packets in the output ports, they are forwarded to the next nodes input ports according to the interconnectivity pattern.

STEP 8 – Display current network status: After each iteration, current network status is displayed in order to trace and find whether a packet loss has occurred or whether packets are routed correctly.

5.6 Test Environment and Experiment Model

The experiments are implemented and compiled with Ansi C compiler. Table 5.1 shows the test environment.

Table 5.1: Test environment

Processor Type	**Intel Pentium 4**
Processor speed	1.7 GHz
Ram type	133 MHz SDRAM
Ram size	512 MB
Operating system	Windows XP
Vendor and version	Microsoft, Professional
Compiler	Borland C++ 5.02

In the experiments, each active node generates random packets at its output port destined (if there is one available) to random destinations as in the model defined in section 5.2. All nodes execute the same code and stop execution after a specified number of iterations is reached. Experiments have been conducted for varying number of loads (packet creation probabilities, 0.033, 0.066, 0.100, 0.125, 0.166, 0.200, 0.250, 0.333, 0.5, 1), from 8 to 1024 nodes (the number of nodes for each network varies depending on the network properties, see chapter 3). On completion, the results for the number of nodes, number of messages created per test, total number of hops, total number of times that node stayed idle, total number of times that each output port is used (transmission and reflection) and the total number of congestion occurred) are calculated and displayed.

5.7 Tests and Network Comparisons

5.7.1 Introduction

In this section test results together with a discussion will be given. Selected networks have been tested with varying number of nodes and different load levels for store-and-forward routing and deflection routing.

5.7.2 Test 1 - Average number of hops

The average number of hops against the system load for different network sizes and different schemes are shown in Fig.s 5.7, 5.8 and 5.9.

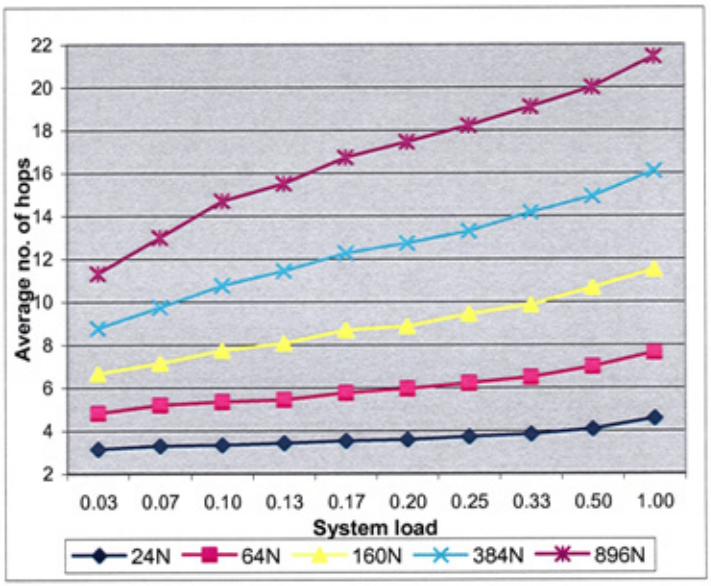

Fig. 5.7: Average number of hops versus system load for Shufflenet employing deflection routing

As can be seen from the figs the number of hops increases very little with the system load for small size networks (24N and 64N), whereas for large size network the increase is linear for the Shufflenet, and Gemnet, and exponential for the De Brujin Graph. This shows that for a large size networks, both Shufflenet and Gemnet are scalable and can cope with increasing traffic when there is no buffer. However, De Bruijn graph could not cope with increased traffic when no buffer is used. Moreover,

a De Bruijn graph with more than 256 nodes could not cope with congestion and therefore is unable to deliver packets to their destinations (see Fig. 5.14 for percentage of contentions occurred per packet). This is because the De Bruijn graph is not designed for deflection routing. Hence, more number of hops is needed for the De Bruijn graph.

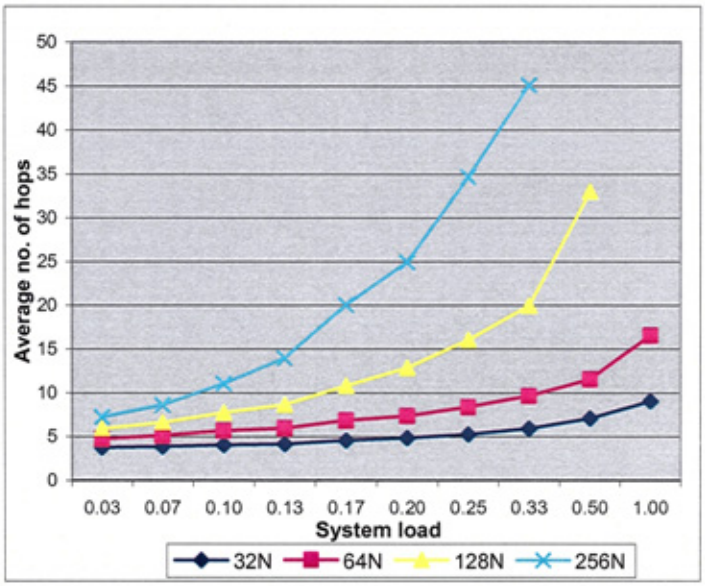

Fig. 5.8: Average number of hops versus system load for De Bruijn Graph employing deflection routing

We compare all three networks for network size of 64, since this is a common number of nodes for all of the networks. The same interconnection pattern is used for all three cases with the only difference in the routing algorithms used. Gemnet and Shufflenet show the same performance although they use different strategies, whereas the De Bruijn graph performs worse than the other two networks especially for larger network sizes. It can be concluded that the routing algorithm's performance depends on the network properties and configuration.

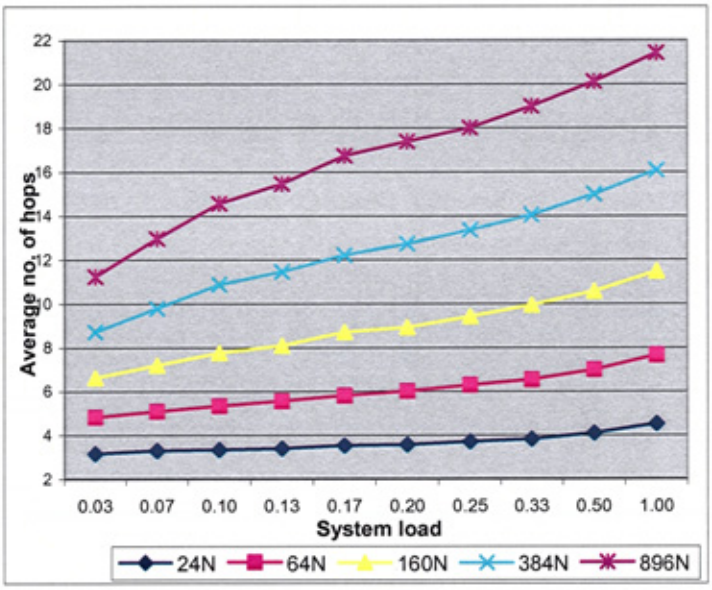

Fig. 5.9: Average number of hops versus system load for Gemnet employing deflection routing

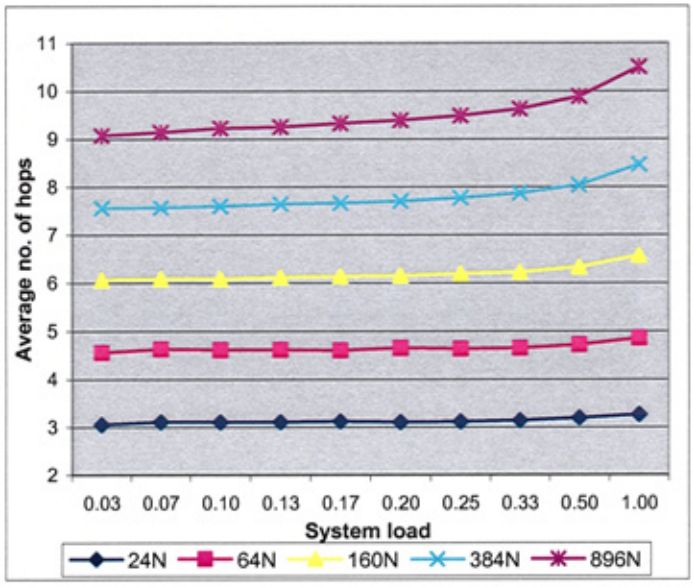

Fig. 5.10: Average number of hops versus system load for Shufflenet employing store-and-forward routing

When a buffer is employed in the nodes, the performance of the networks and their routing algorithms increase dramatically, see Figs. 5.7 - 5.12. Moreover, this increase is much higher in the De Bruijn graph compared to the other two. Therefore, it can be concluded that the De Bruijn graph utilises the buffer more efficiently compared to

the other two. On the other hand with small network sizes the De Bruijn graph performs better than the others.

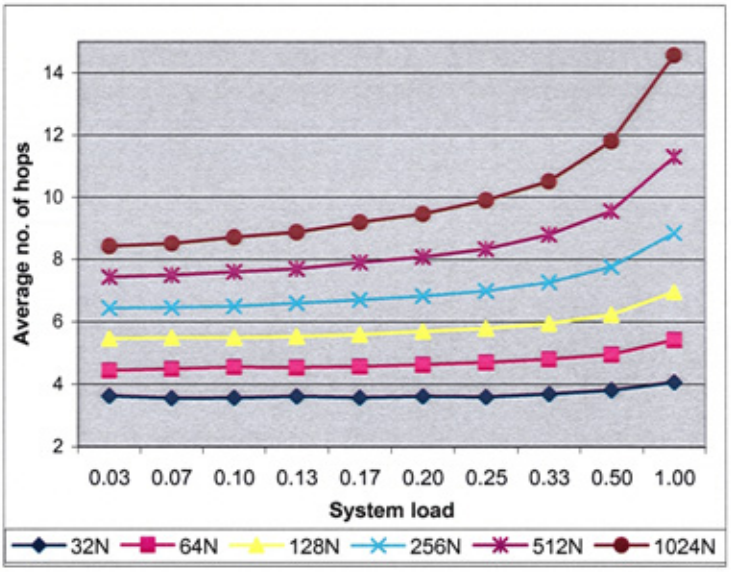

Fig. 5.11: Average number of hops versus system load for De Bruijn Graph employing store-and-forward routing

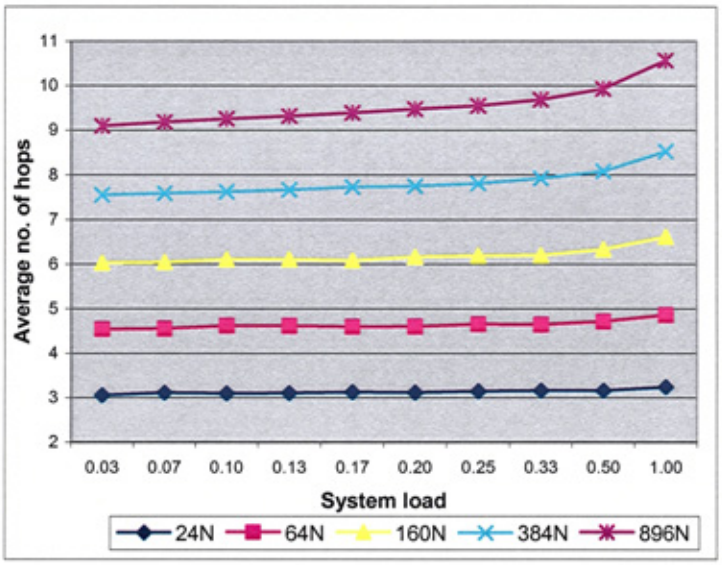

Fig. 5.12: Average number of hops versus system load for Gemnet employing store-and-forward routing

When Fig.s 5.7-5.12 are compared, the De Bruijn graph gives the same performance (slightly better) as the other two networks under low system loads (load < 0.200). However, at high loads (>0.25) the number of hops required for the De Bruijn graph

is higher (e.g. around 30-40% at system load 1 with the biggest network size) compared with the other two. This is expected at high loads, since the possibility of having three packets in a node at any given time is high where one of the packets needs to be deflected.

Comparing deflection routing tests with the store-and-forward routing tests show that there is an improvement in performance in the latter case, see Table 5.2. However, this improvement is gained at the cost of system complexity.

Table 5.2: Average number of hops comparison for three networks and for network size of 64

	Average number of hops	
Network type	**With buffer**	**Without buffer**
Shufflenet	**4.64**	**5.97**
De Bruijn	**4.68**	**8.13**
Gemnet	**4.62**	**5.99**

5.7.3 Test 2 - Contention

Figs 5.13, 5.14 and 5.15 show the contention ratio versus the system load for deflection routing for different network size. The poor performance reported by the Debruijn Graph in the previous test can be explained by looking at the Fig. 5.14. Shufflenet and Gemnet have similar characteristics with lower contention ration compared with the De Bruijn graph. This is because these networks can act as a 'buffer' when there is a need for buffering. In the De Bruijn graph contention ratio increases exponentially especially for larger network sizes.

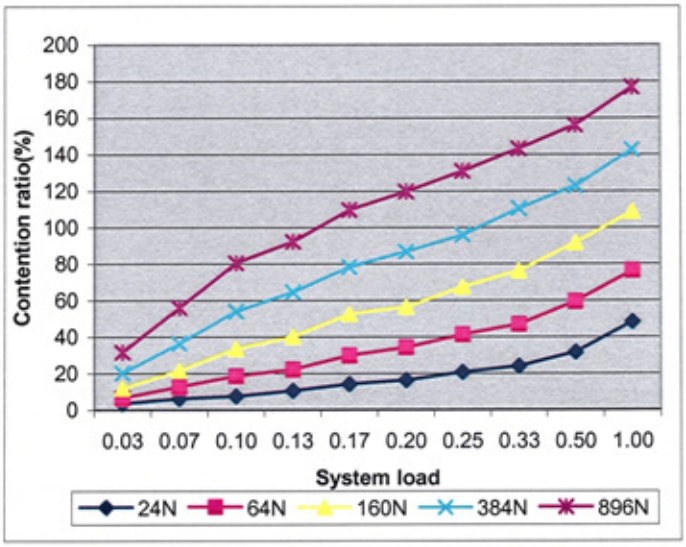

Fig. 5.13: Contention ratio versus system load for Shufflenet employing deflection routing

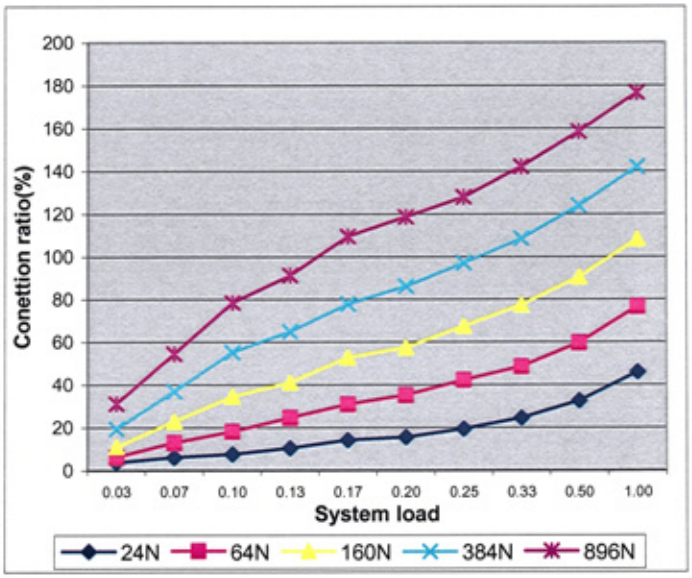

Fig. 5.14: Contention ratio versus system load for Gemnet employing deflection routing

Fig.s 5.16 and 5.17 show that for the Shufflenet and Gemnet contention percentage increases when store-and-forward routing is employed. In addition, Fig.s 5.16, 5.17 and 5.18 show that the number of packets that contention occurred has increased for all networks when the number of nodes and the system load is increased.

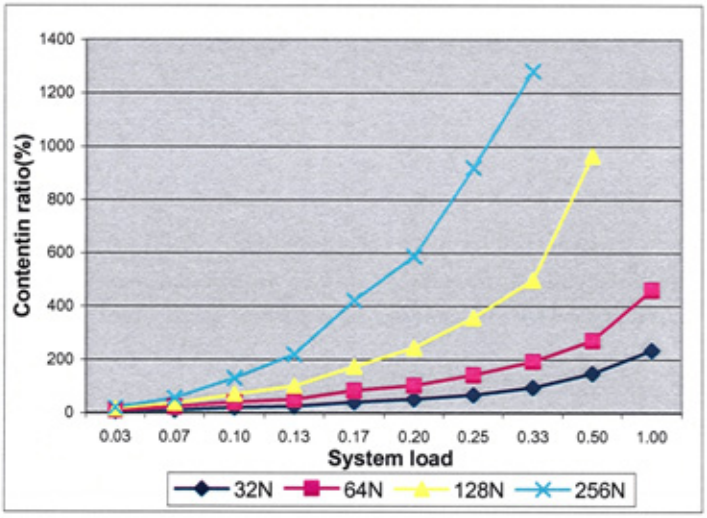

Fig. 5.15: Contention ratio versus system load for De Bruijn Graph employing deflection routing

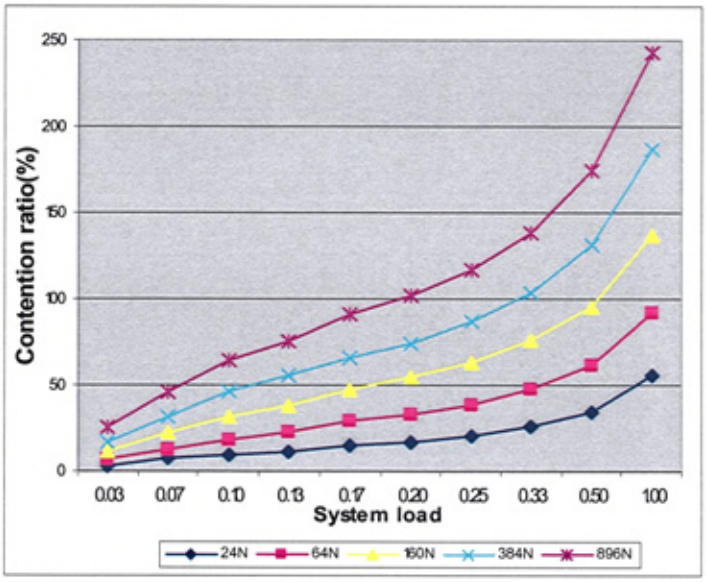

Fig. 5.16: Contention ratio versus system load for Shufflenet employing store-and-forward routing

In the De Bruijn Graph, more packets experienced contention compared to the other two. Comparing results for the store-and–forward routing with the deflection routing shows that only for the De Bruijn graph the contention ratio has been reduced. This is because, in the Shufflenet and Gemnet when two packets are at the input ports destined to the same output port and when there is a packet already in the buffer that is also wanting to go to the same port regardless of the type of the packet (Multiple or Single), the packet in the buffer is forwarded to the output port.

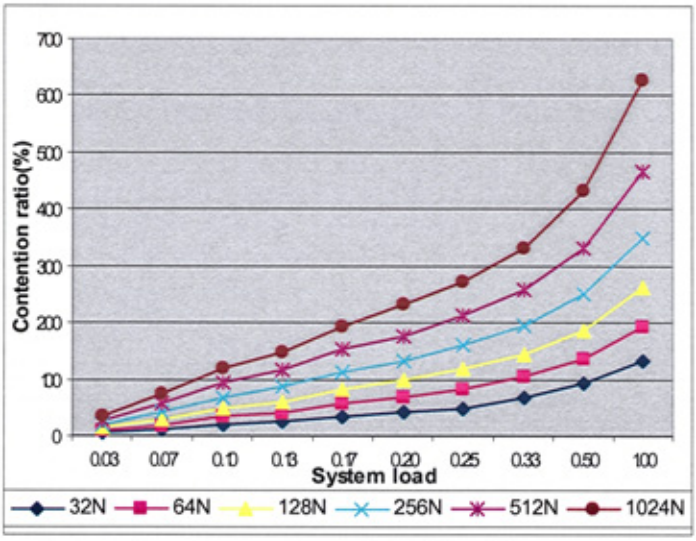

Fig. 5.17: Contention ratio versus system load for Gemnet employing store-and-forward routing

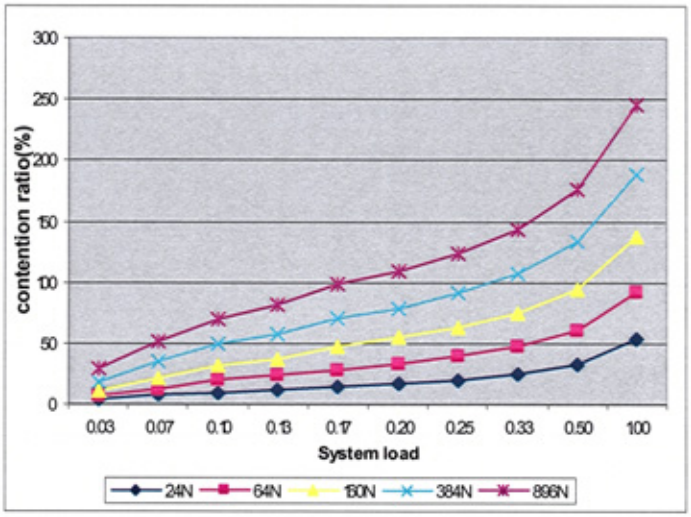

Fig. 5.18: Contention ratio versus system load for De Bruijn Graph employing store-and-forward routing

Table 5.3 shows that packets on average experienced 60% less contention in the De Bruijn graph when store-and-forward routing is employed compared to other two networks. Therefore, one can say that, the average number of hops has a considerable impact on the contention percentage. As shown in Table 5.2 the average number of hops is almost 80% lower when store-and forward routing employed in the De Bruijn graph compared with the deflection routing.

Table 5.3: Contention ratio comparison for three networks and for network size of 64

	Contention Ratio (%)	
Network type	**With buffer**	**Without buffer**
Shufflenet	35.88	34.56
De Bruijn	73.62	136.51
Gemnet	35.82	35.26

5.7.4 Test 3 – Node idleness percentage

As described earlier, node idleness ratio represents the proportion of the network nodes that are idle in each iteration. Fig.s 5.19, 5.20 and 5.21 show system load versus node idle ratio for deflection routing. As the network load increases, the number of nodes that have not done any processing decreases, as there are more packets generated by nodes. However, it can be seen from the Fig.s that the number of nodes that are idle decreases. This is due to having more nodes, thus, more paths. Consequently, a smaller number of nodes is occupied at any given time.

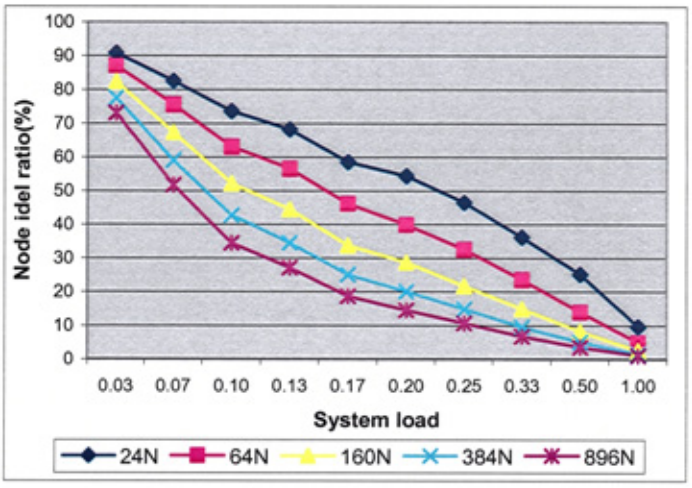

Fig. 5.19: Node idle ratio versus system load for Shufflenet employing deflection routing

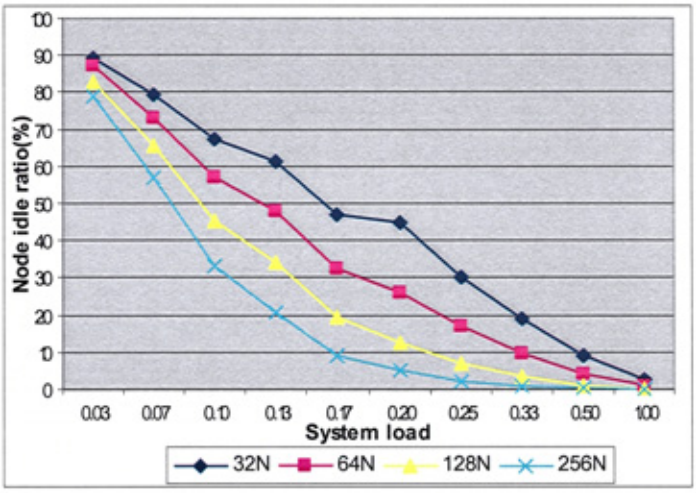

Fig. 5.20: Node idle ratio versus system load for De Bruijn Graph employing deflection routing

In the De Bruijn graph, fewer nodes were idle during the simulations compared to the other two as its ability to discover new and alternative paths is limited. Hence, more hops are required for a packet to reach to its destination. Therefore, more system resources are used.

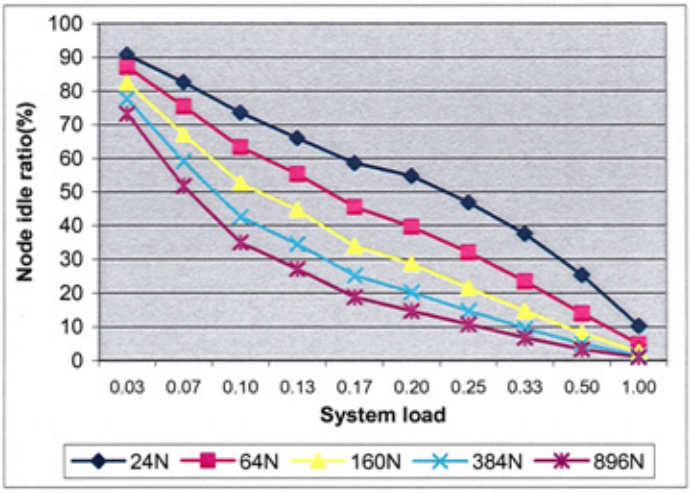

Fig. 5.21: Node idle ratio versus system load for Gemnet employing deflection routing

In the store-and-forward routing the difference between the De Bruijn graph and other two networks are much smaller. One can say that, the De Bruijn graph uses network resources more efficiently than other two networks by looking at the test results. However, having a smaller number of nodes remaining idle does not necessarily mean

that the De Bruijn graphs resource utilisation is better than the others. In fact, it is the opposite as in the De Bruijn graph, packets experience more hops to reach their destinations, thus resulting in increased crosstalk and power penalty.

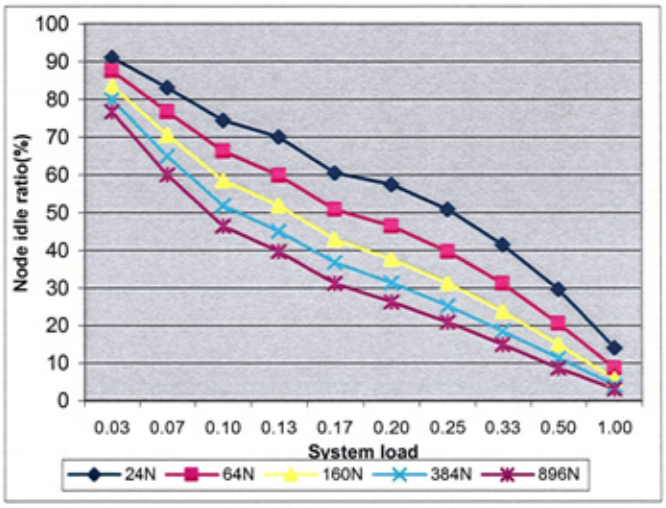

Fig. 5.22: Node idle ratio versus system load for Shufflenet employing store-and-forward routing

Fig.s 5.22, 5.23 and 5.24 show the results for the store-and-forward routing. As in the deflection routing tests node idleness ratio increases with the number of nodes. However, in this case the performance of all the networks for varying network sizes is closer. This is due to the fact that algorithms can handle more packets when buffer is available.

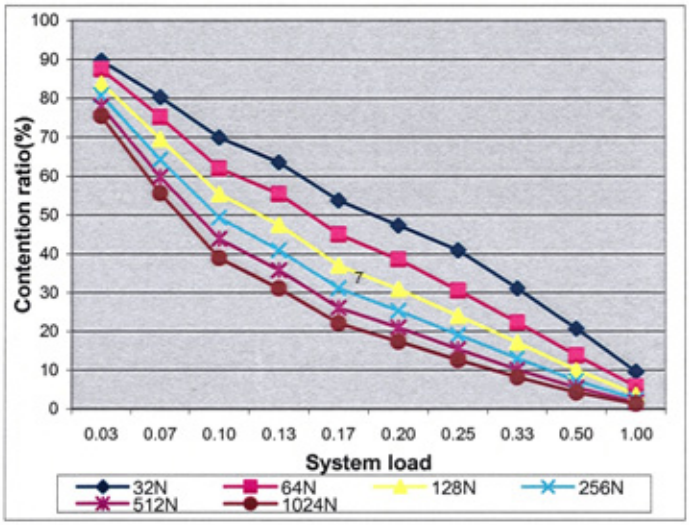

Fig. 5.23: Node idle ratio versus system load for De Bruijn Graph employing store-and-forward routing

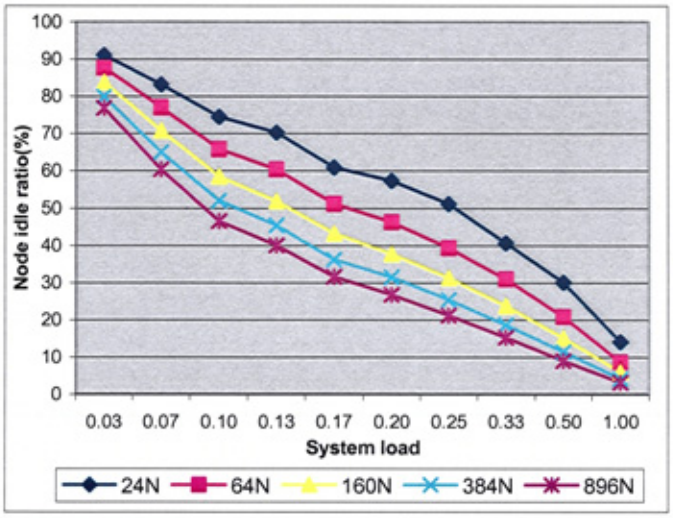

Fig. 5.24: Node idle ratio versus system load for Gemnet employing store-and-forward routing

Table 5.4 shows that, in all networks when deflection routing is employed there are less number of idle nodes available when compared to the store-and-forward routing. This was an expected result, as a node with buffer can accommodate three packets at any given time, whereas a node without buffer can accommodate only two.

Table 5.4: Node idleness ratio comparison for three networks and for network size of 64

	Node Idleness Ratio (%)	
Network type	**With buffer**	**Without buffer**
Shufflenet	48.75	44.22
De Bruijn	43.61	35.39
Gemnet	48.77	43.89

5.7.5 Test 4 – TRR

Fig.s 5.25 and 5.26 show the TRR versus the system load for different Shufflenet and Gemnet configurations employing deflection routing algorithm. During the tests for the De Bruijn graph, no gain in TRR was observed. This was an expected result as there are no multiple paths exist. Therefore, no TRR results are presented for the De Bruijn graph.

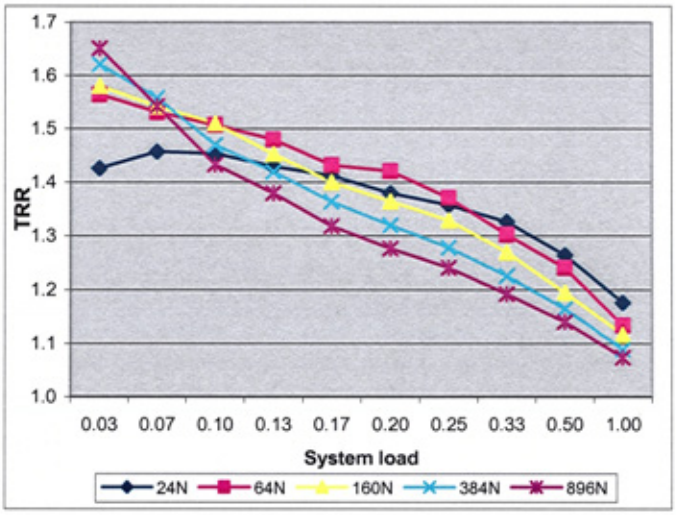

Fig. 5.25: TRR versus system load for Shufflenet employing deflection routing

For all cases, TRR drops as the system load increases. This is because of reduced number of multiple paths at high system load. Moreover, as the system load increases the possibility of two packets arriving simultaneously at a node increases. Thus, both output ports of the router are used. As expected at low load (< ~0.17), larger configurations show higher TRR. Whereas, at high load (> ~0.17) the TRR is higher for smaller Shufflenet configurations. This is because, for larger Shufflenet configurations there are more packets in the system, and therefore the possibility of finding non-congested paths decreases.

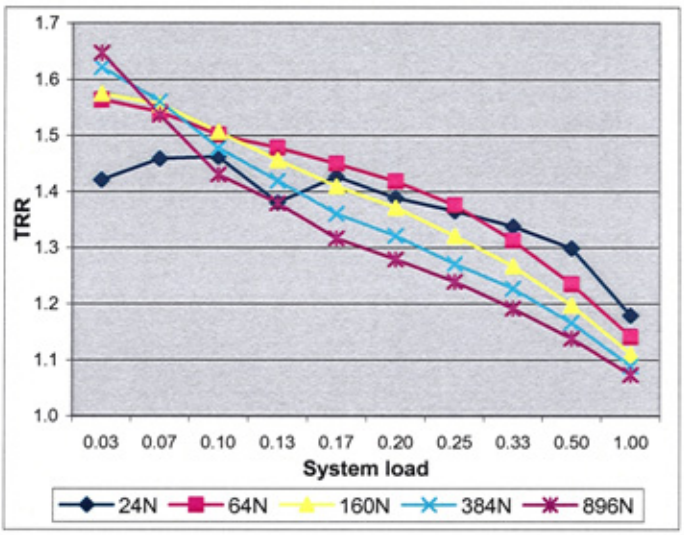

Fig. 5.26: TRR versus system load for Gemnet employing deflection routing

Fig. 5.27 and 5.28 show TRR versus the system load when employing store-and-forward routing. As expected, the TRR profile is almost same as in Fig. 5.25 and 5.26, since TRR does not dependent on buffering. Note that the deflection routing gives slightly improved performance. This is because the average number of hops taken per packet is 25% higher than store-and-forward routing [Maxemchuck, 89]. Indeed, TRR is directly proportional to the number of multiple paths that exist. Therefore, one can state that having a buffer has no effect on the number of multiple paths.

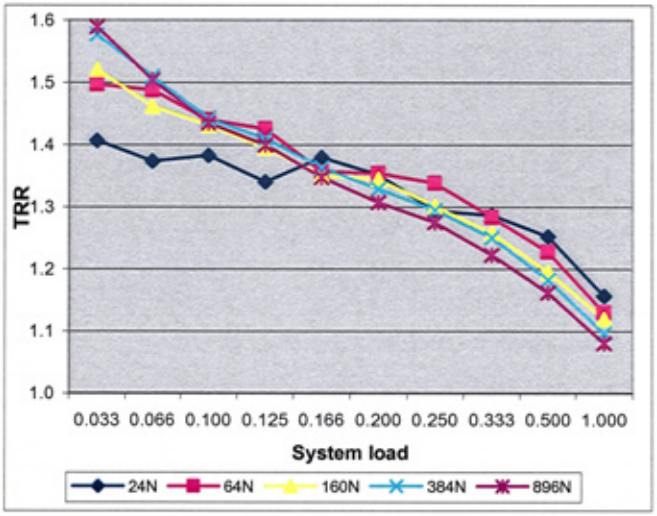

Fig. 5.27: TRR versus system load for Shufflenet employing store-and-forward routing

As in Fig. 5.27 and 5.28, the decrease in the TRR as the load increases is due to an increased number of packets within the network, thus increasing the occurrence of deflection. Packets not being deflected also decrease the possibility of multiple hops. When a packet is deflected, it is at least k hops away from its destination, which makes it TYPE-M packet.

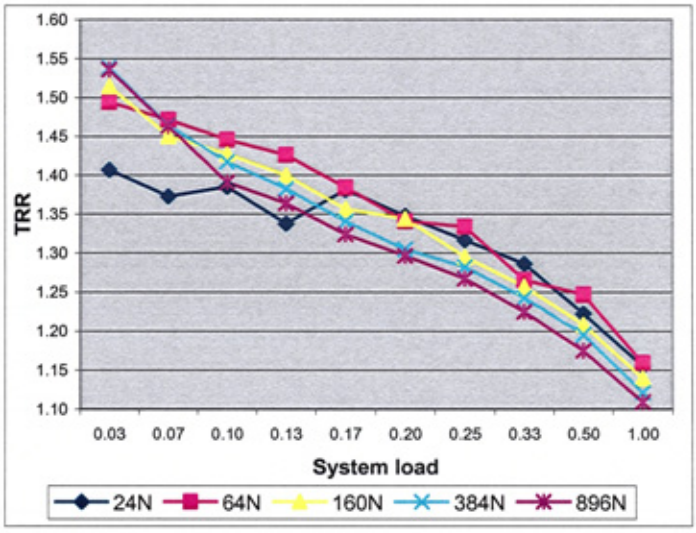

Fig. 5.28: TRR versus system load for Gemnet employing store-and-forward routing

Table 5.5 shows that Shufflenet performs slightly better than Gemnet in deflection routing. However, in the store-and-forward routing Gemnet performed slightly better in high loads although the behaviour of both networks was not as stable as deflection routing tests. Moreover, it can be concluded from the Fig.s that both of the networks performed reasonably well in deflection routing as TRR results are slightly higher than the store-and-forward routing. However, the difference is very small and it can be concluded that buffering does not have much effect on TRR. The De Bruijn graph has not been included in these tests as TRR is always 1 due to the lack of multi paths. Therefore, one can say that, the De Bruijn graph performed worst in TRR tests.

Table 5.5 also shows that, when prioritised routing is employed in De Bruijn graph it has no effect on the TRR performance. In fact, there are no TYPE-M packets exist in De Bruijn graph which the prioritised algorithm is based on. Also, deflection routing performed slightly better than the store-and-forward routing in the Shufflenet and Gemnet, Table 5.5.

Table 5.5: TRR comparison for three networks and for network size of 64

	TRR	
Network type	**With buffer**	**Without buffer**
Shufflenet	1.35	1.39
De Bruijn	1.00	1.00
Gemnet	1.35	1.40

5.8 Comparison

Different performance metrics and formulas were defined to help to analyse the networks and their routing algorithms that were outlined in Chapter 2. The test criteria listed in the requirements part, will enable the simulation results to be analysed and evaluated in the most effective way. The system design should enable the required information to be collected and analysed. The simulation based model was outlined after realisation of the requirements was highlighted. The design model is based on the routing algorithm properties and on the information presented in Chapter 2.

The requirement design model also proposed to test the correctness of the routing algorithms. This model covers the requirements defined in Chapter 2 as closely as possible and it enables to retrieve quantitative data from the implementation of the model to prove the correctness of the algorithm.

The designed model is the basis of the implementation of the algorithms. It provides great flexibility (even specific network and algorithm requirements can be easily implemented and satisfied) to the implementation process of the algorithms that has been covered successfully.

A simple priority scheme was developed where packets are routed via the transmission port instead of the reflection as much as possible. Simulation results showed that TRR > 1 is achievable over a wide range of system load for both deflection and store-and-forward routings. Higher TRR means that packet will experience lower crosstalk when propagating through the network. One drawback of prioritised scheme is that the output port (channel) utilisation is imbalanced as one of the ports has priority over the other. The algorithms were ranked as in the Table 5.6; 1 represents the best and 3 represent the worst algorithms. Stability is defined based on the fluctuations on the simulation results.

Table 5.6: Evaluation of the routing algorithms and topologies

	Shufflenet	**De Bruijn Graph**	**Gemnet**
Average number of hops	1	3	1
Contention	1	3	2
Node idleness	1	3	1
TRR	1	3	1
Routing Algorithm Simplicity	2	1	3
Scalability	3	2	1
Stability	1	2	3
OTDM adaptability	1	3	1
Efficient use of buffer	2	1	2

The shufflenet and Gemnet performed very closely and better than the De Bruijn on overall comparison. However, in an application that requires very simple routing scheme and less processing, De Bruijn would be the best choice among the others although it performed worst in overall evaluation.

The test results also showed that store-and-forward routing performed better than deflection routing with regards to the average number of hops. However, with store-and-forward routing it may take longer (time wise) to reach destination. However, TRR results showed that Shufflenet and Gemnet are more suitable to OTDM networks compared with the De Bruijn graph.

As a summary, Gemnet and Shufflenet have the most efficient routing algorithms. However, De Bruijn graph's routing may be the choice for use with systems that requires low processing power. On the other hand, Gemnet is the only scalable network and could be selected in a specific system that requires high flexibility level.

5.9 Conclusions

The aim of this chapter was to simulate and evaluate the logical network topologies suitable for the OTDM networks which operate in synchronous mode as no such evaluation exists in the literature to the best knowledge of authors. Therefore, a generic model for implementing logical interconnection topologies in software was proposed to investigate the performance of the logical topologies and their routing algorithms for packet based synchronous networks. The model is generic for synchronous transfer modes and therefore can be used to implement any logical topology using any programming language. Three topologies were investigated and implemented namely: Shufflenet, De Bruijn graph and Gemnet. Results for the average packet delay showed that the De Bruijn graph performed the worst. Also, it was observed that the De Bruijn graph made more efficient use of buffering compared to other algorithms.

In addition, a simple priority rule named as TRR was defined in order to utilise the usage of the asymmetric 2x2 node. Results showed that TRR has no effect on the performance of the topology and the algorithm. However, it increased the utilisation of the transmission port significantly. Also, no gains were observed in the De Bruijm graph since there are no multiple paths existing.

This chapter also evaluated a few of the well known logical network topologies and their routing algorithms. The next chapter will evaluate intelligent antnet routing algorithm for arbitrary networks. Novel improvements and modifications will also be introduced for the antnet routing algorithm.

CHAPTER VI - IMPROVED ANTNET ROUTING ALGORITHM

6.1 Introduction

The aim of this chapter is to outline the weaknesses associated with the antnet routing algorithm and propose solutions for overcoming them. Unlike previous tests that were carried out in Chpater 5, the antnet routing algorithm is being evaluated on a different platform and its performance being investigated based on the criteria defined. The chapter starts with an introduction to the programming environment and model. This is followed by test criteria and the traffic model used together with all the assumptions made. Finally, novel implementations of antnet routing algorithm are presented with some results and discussion. The performance test criteria selected are those widely reported in the literature [Tannebaum, 03]. The results will be analysed for average packet delay and network throughput.

OSPF, RIP and Q-routing algorithms have been implemented in the previous work reported in [Di Caro, 04] showing that antnet performed better than all. This has also been verified in the work presented here.

6.2 Parallel Virtual Machine Model

This section will look at the parallel virtual machine (PVM) programming model before proceeding with the design, as the system design is closely related to the underlying model [Geist et al, 94]. PVM is a very flexible message passing system

that enables (mostly Unix) networks of computers to be used as a single distributed memory parallel computer (note, PVM version 3 can be used in different Operating System environments). This network is referred to as the virtual machine. PVM allows the user to use standard tools to develop a program using the C/C++, programming language.

To begin the application, the user manually starts one copy of the application known as the master or initiating task. This task subsequently starts other PVM tasks within the virtual machine. Eventually there are a number of active tasks computing and communicating to solve the problem. Once the tasks are started they can interact through the explicit message-passing functions provided by PVM. It is possible to use different parallel programming models with PVM. The following section will look at the master-worker programming model before proceeding with the design of the system.

6.2.1 The master worker model

The master worker is a classic parallel programming model. One processor (master) is devoted to assigning computations to all the other processors (workers) involved. The master-slave (or host-node) model in which a separate "control" program termed the master is responsible for process spawning, initialisation, collection and display of results, and perhaps timing of functions (Fig. 6.1). The design part will be based on this classic model.

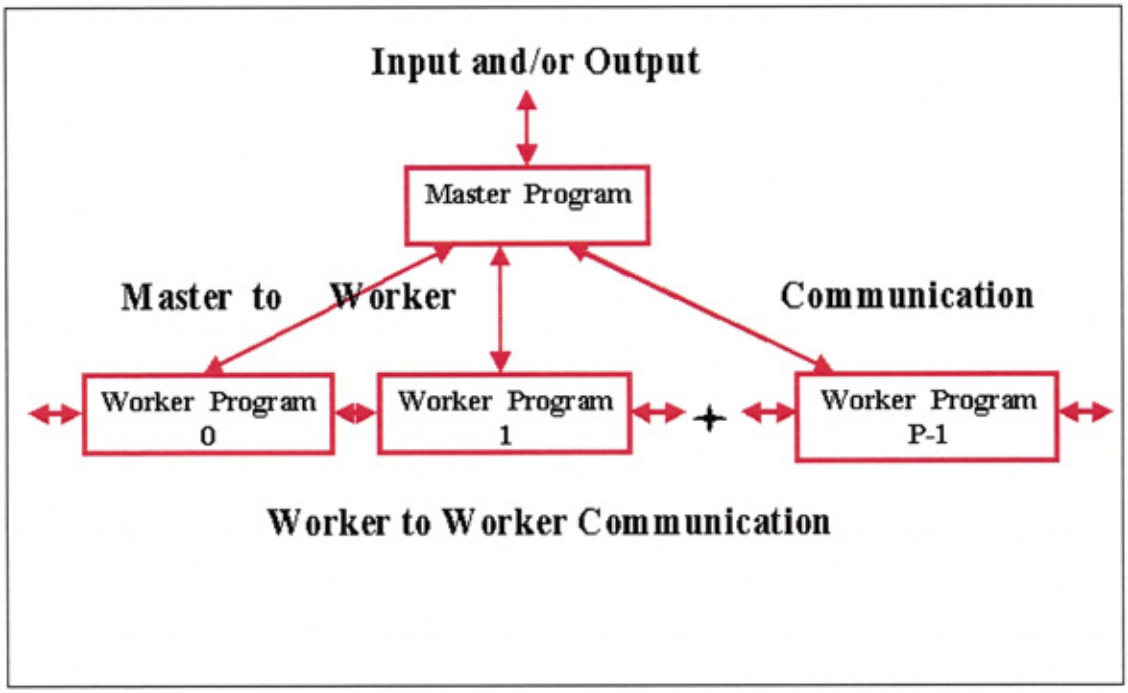

Fig. 6.1: Master worker model

6.3 Assumptions Made

The following section firstly looks at the assumptions made and then describes the design model based on these assumptions and the master model described in the previous section. All algorithms that were considered in the literature review and those to be implemented are based on a number of assumptions and conditions for the distributed environment:

- Algorithms are implemented in the C language in a parallel environment by using parallel virtual machine
- Assigning every process to a different node both on the same machine and different machines simulates parallel behaviour
- Non-uniform data traffic with 50% of the traffic being forwarded to the hotspot nodes

- No packet creation destined to the immediate neighbours
- 5 seconds are given to discover all the paths and to initialise the probabilistic routing table entries
- Ant creation rate is set to varying rates
- Network topology with each link having equal cost is adopted, see Figs. 6.2 and 6.3[18]. Traffic is generated to nodes 1 and 21 on 29 node network and from the bottom 6 nodes (0-5) on the left and on the right (31-36) of the network on 36 node configuration.
- 36 node network is widely used topology [Boyan & Littman, 94] in artificial intelligence to force the algorithm to learn and use the longer path when shorter path is congested and 29 node random topology has been designed to simulate balanced network scenarios.
- For each simulation, packet generation is stopped after creation of 2500 packets per node and simulation is stopped after all packets are arrived to their destinations or detected and deleted from the network
- Every simulation is run 8 times and the average of the results is used for accuracy It is assumed that the packet size is fixed and there is no packet loss
- All experiments are implemented for varying link evaporation rates, since it has a significant effect on the performance of the algorithm

All these assumptions and conditions should be satisfied by the distributed system together with the design and implementation of the algorithm.

While designing and implementing the system, additional assumptions and issues with regard to the communication network and model are also taken into consideration.

[18] In addition to these 14 node nfsnet has also been implemented and simulated [Tekiner et al, 04-2]

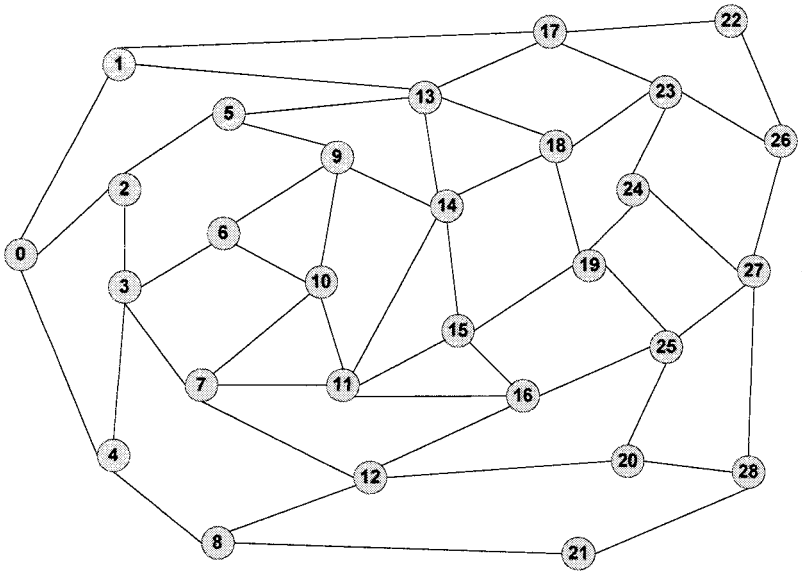

Fig. 6.2: 29 Node random network links having the same cost

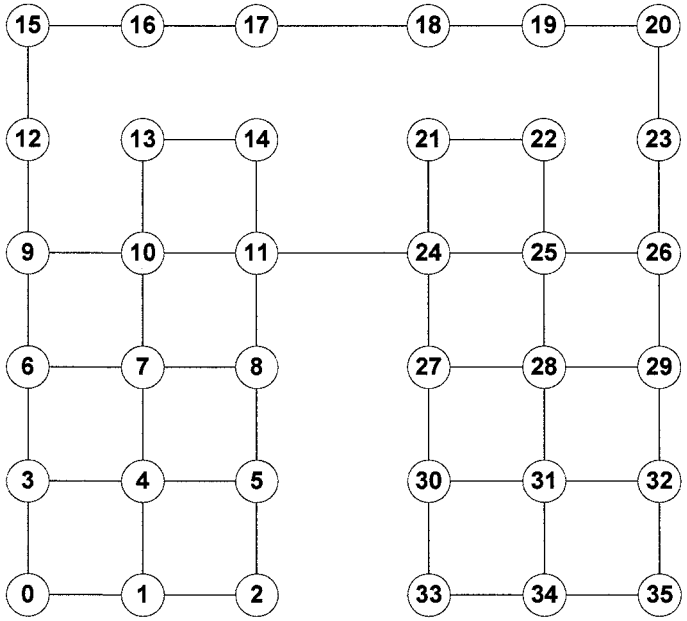

Fig. 6.3: 36 node irregular grid

Message delivery guaranteed: Messages should be delivered without being changed or lost.

Message order preservation: The system should ensure that no message overtaking occurs. This means that messages should be delivered in the order that they have been sent.

Message transfer delays: Messages should be delivered to their destinations in a finite amount of time. However, transfer time can be different according to the communication system load.

Network topology is known: Nodes know the path to other nodes.

6.3.1 Traffic models

Algorithms will be run on three different topologies and with 3 different traffic loads. Uniform and non-uniform (hot spot) traffic models will be used with the following load levels and distributions:

1. **Low Loads**: A packet is created in every 1 second.
2. **Poisson Distribution**: Lambda is set to 0.5 seconds
3. **High Loads**: A packet is created in every 0.001 second

It has been observed during testing that the modifications and improvements introduced in this thesis are only effective under hotspot traffic conditions therefore no results for uniform traffic models will be presented here although tests were carried out.

6.4 Overall System Design

Fig. 6.4 shows a general view of the test (simulation) model that will be implemented. A master process is executed from one of the nodes and is responsible for the creation (spawn) of the worker nodes that will do the simulation. The master process is also responsible for the initialisation of the system and display of information (test results). Workers exchange messages between each other to carry out tests and return results to the master process.

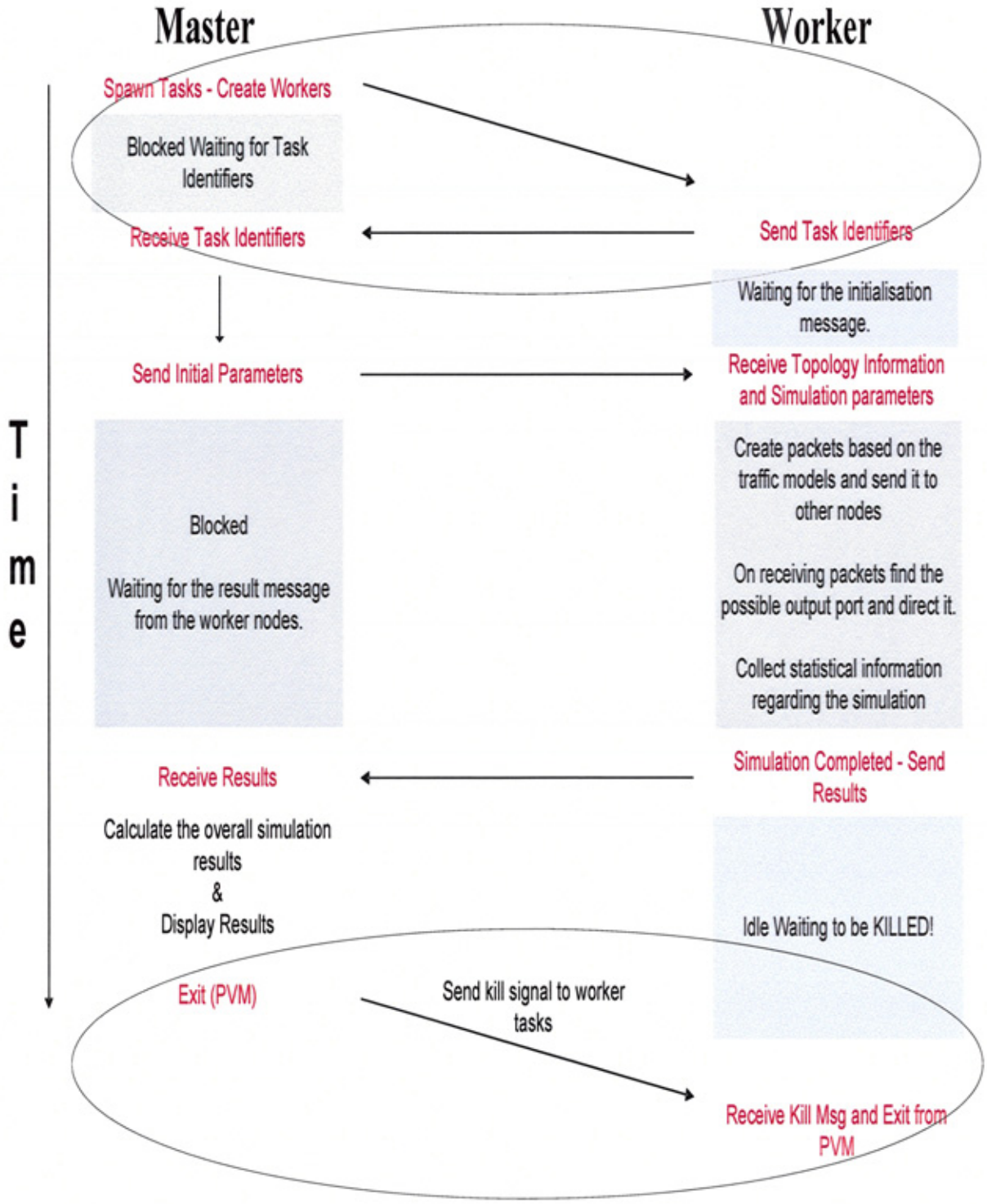

Fig. 6.4: Overall system design

First of all, the master process creates a number of nodes that are required to execute the simulation and receives their task identifiers (tids) in one operation. These task identifiers are later used to enable the communication between the nodes (they are unique addresses for the nodes within the PVM system). After receiving the task identifiers, the master process prepares the initialisation message and forwards it to all of the nodes (by using the stored task identifiers). In the initialisation message, the workers just know their neighbour processes or the token holder, according to the algorithm's implementation, depending on the structure model that they use (dynamic or static). Workers carry out the simulation according to the initialisation data they receive. The master process stays idle throughout the simulation and waits for the results of the simulation (such as the number of messages exchanged per node and time taken, etc). After successful completion of the simulation, the worker processes send the message containing results to the master process. The master process waits for all the worker processes to send the result message. After receiving all the result messages, the master process calculates the results and displays them in an comprehensible manner. The master process exits from the PVM system just after displaying the results. This result allows the worker node to exit from PVM and kill signals are automatically sent to worker nodes by the PVM system.

6.5 Antnet Modifications

Based on the original antnet routing algorithm three modifications have been proposed and tested as outlined below.

6.5.1 Deleting aged packets

During the simulations it was discovered that some packets travel within the network for a number of hops until they reach their destination when the original antnet is employed. This problem occurs because the routing table used in the antnet routing algorithm is probabilistic and therefore a few packets have possibility to be directed to non-optimal routes and cycle. This is similar to the problem experienced by ants that they are also deleted when they are forced to visit pre-visited nodes (when a cycle occurs). A simple rule is defined to detect and drop these packets as follows:

$$\textit{if } PACKET\,AGE > 2\,x\,NO\_OF\_NODES \qquad (6.1)$$

$$\textbf{\textit{then}}\ DROP\ PACKET$$

This rule is based on the information obtained from experimental results. It was observed that only 0.5% of the packets experience this problem. Therefore, only 0.5% of the packets are dropped from the network. However, when the packet age condition is set to ***1XNO_OF_NODES,*** the packet loss increased to almost 7%. On the other hand, when the condition is set to ***3XNO_OF_NODES,*** the loss rate decreased to 0.3% with no further improvement in the performance. Therefore, packets that travel continuously in the given networks conditions are dropped based on the equation 6.1. However, a different threshold level may be needed for different configurations.

6.5.2 Limiting the effect of *r* due to traffic fluctuations

The reinforcement r applied to the routing table entries is limited by the lower and upper bounds defined as follows:

$$\textit{if } (NO\_OF\_NODES <= 5) \qquad (6.2)$$

$$0.1 < r < (1 - 0.1 * NO\_OF\_NODES)$$

$$\textbf{\textit{else}}\ /*\ \textit{if}\ (NO\_OF\_NODES > 5)\ */$$

$$0.05 < r < (1 - 0.05 * NO\_OF\_NODES)$$

The values used are based on experimental results and it is intended to limit the effect of the traffic fluctuations in the network at a given time. A similar method to control the effect of r has also been reported in [Tekiner et al, 04-1]. However, if r does not satisfy these values for three consecutive times and is less than 0.95, then reinforcement is applied to the routing table entry. It is believed that by limiting the impact of r on the routing table entries the algorithm would not freeze as easily as was the case in the originally proposed algorithm.

Fig.s 6.5-6.10 shows the performance comparison of the original and modified antnets in terms of the average packet delay experienced per packet for different ant creation rates for 29-node and 36-node networks and for three different loads. With the modified algorithm, packet delay is reduced compared with the original routing algorithm. This is because in the original routing algorithm some packets travel with very high number of hops within the network, thus using a considerable amount of network resources and bandwidth. Although only around 0.7 % of the packets experience this problem, they occupy large amount of system resources as it takes more than double hops than average to reach the destination.

In Fig. 6.5 it can be seen that as the number of ants increases the average packet delay increases. The biggest improvement (5%) has been achieved when ant creation rate is set to 2.5 sec. Moreover, the best performance is achieved when ant creation rate is set to 1 sec.

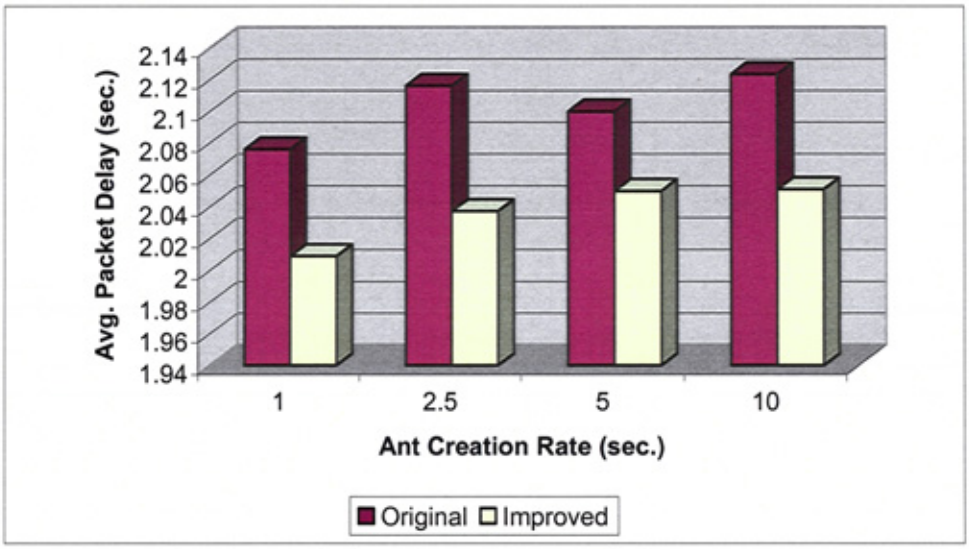

Fig. 6.5: Average packet delay versus ant creation rate 29-node for original and improved antnet algorithms for low loads

In Fig. 6.6 it can be seen that as the number of ants increases in the network the average packet delay decreases when ant creation rate is less than 5. The biggest improvement has been achieved when ant creation rate is set to 2.5 sec is around 5%, similar to the previous result. Improved antnet performed the best when ant rate is set to 2.5. This is because fewer numbers of packets are available at the network at any given time when compared to the previous test. Therefore, increasing the number of packets in the network decreased the contention.

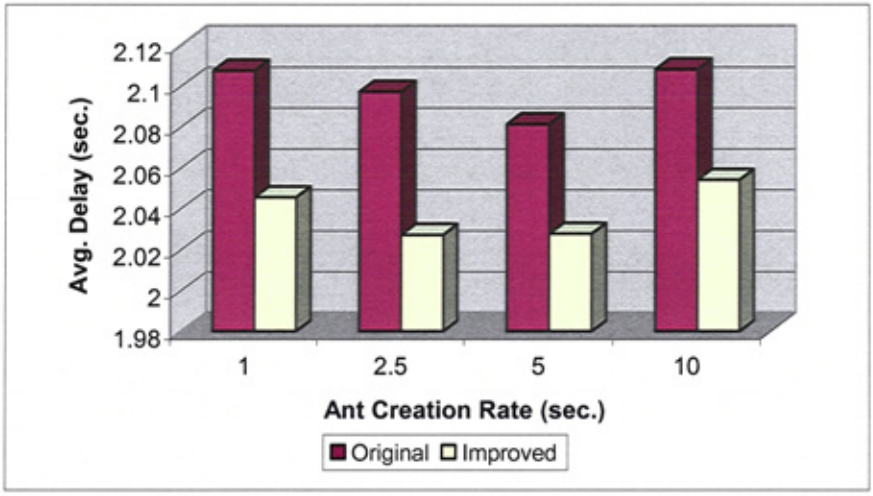

Fig. 6.6: Average packet delay versus ant creation rate 29-node for original and improved antnet algorithms for medium loads

Fig. 6.7 shows that unlike previous tests the biggest improvement is achieved with the high load when ant rate is set to 10 seconds. Moreover, this improvement is bigger than the previous tests and it is around 6.5%. This is due to the fact that fewer numbers of packets are created in any given time. Thus, creating more ants decreases the packet ant ratio, which affects the performance of the network. Therefore, this proves that increasing the number of packets within the system does not always lead to improved performance, since utilisation also play an important role in the performance of the network.

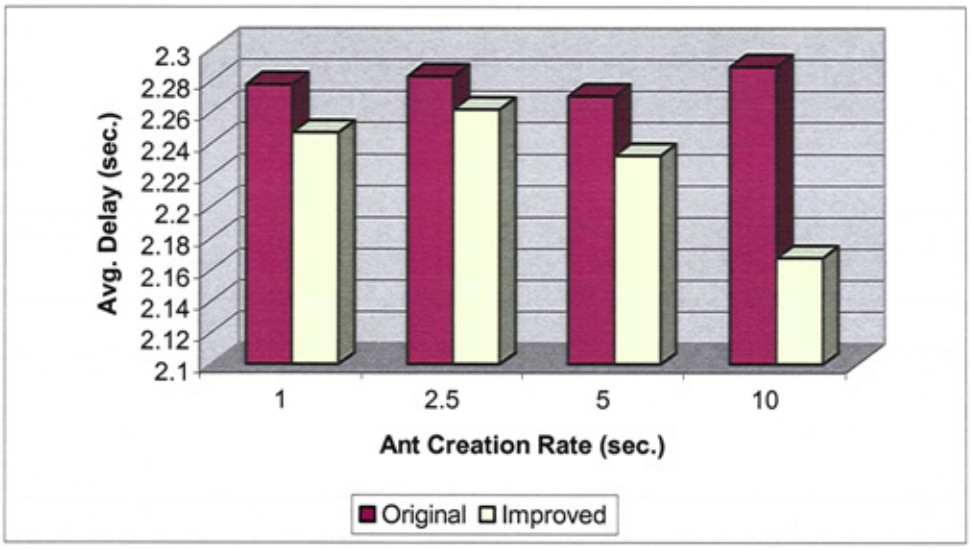

Fig. 6.7: Average packet delay versus ant creation rate 29-node for original and improved antnet algorithms for high loads

Fig.s 6.8-6.10 shows a significant improvement in the average packet delay performance compared with the original version of the antnet for 36 node irregular grid. This is partly due to the architectural characteristics of the irregular grid itself. In irregular grid connectivity is higher compared to the 29-node random network. Therefore, there are more paths available for the ants to explore. Moreover, since there are only two paths connecting each side of the grid, it is more likely to get unreliable reinforcement signal r from the system. Thus, limiting the effect of r hides

the effect of instantaneous changes over the algorithm. From Fig. 6.8, it can be seen that as the number of ants decreases the average packet delay decreases. However, with the modified algorithm, packet delay is reduced compared with the original routing algorithm. This is because in the original routing algorithm some packets travel with a very high number of hops within the network thus utilising considerable amount of network resources and bandwidth.

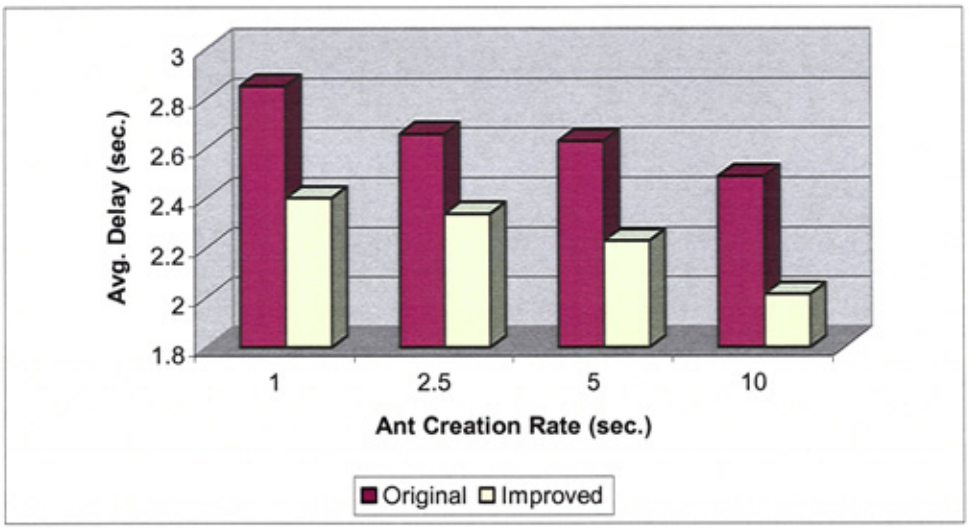

Fig. 6.8: Average packet delay versus ant creation rate 36-node for original and improved antnet algorithms for low loads

Fig. 6.9 shows the ideal case for the improved algorithm, illustrating that ant creation rate do not have any effect on the average packet delay for the case with the poisson traffic distribution (0.1, where packet is created in every 0.1 secs). Therefore, delays experienced per packets stay almost the same for different ant creation rates.

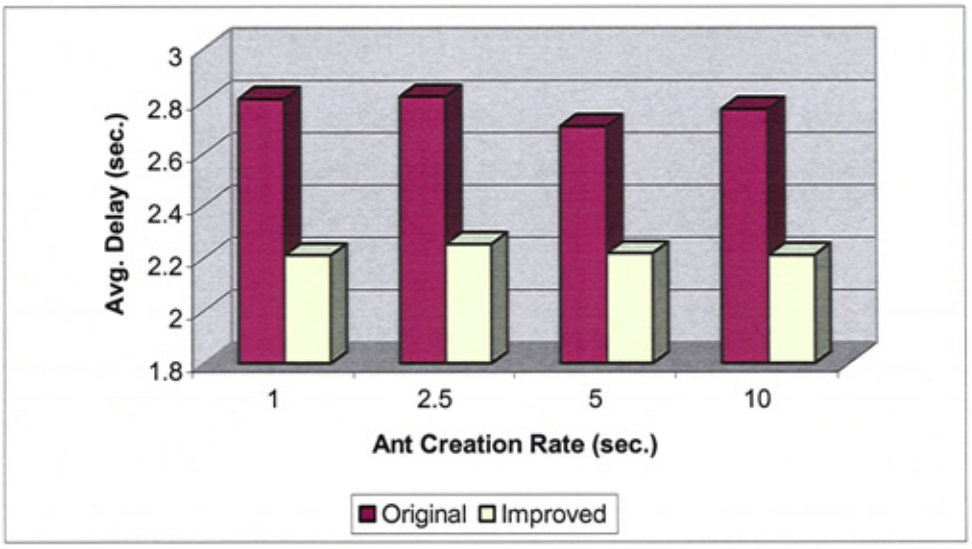

Fig. 6.9: Average packet delay versus ant creation rate 36-node for original and improved antnet algorithms for medium loads

Fig. 6.10 shows that the average packet delay for high load is similar to the medium load case (see Fig. 6.9) at lower ant creation rates. However the improved algorithm performs better with high ant creation rates. The performance gain achieved for the improved antnet with high loads when ant rate is set to 10 is around 33% compared to the original approach.

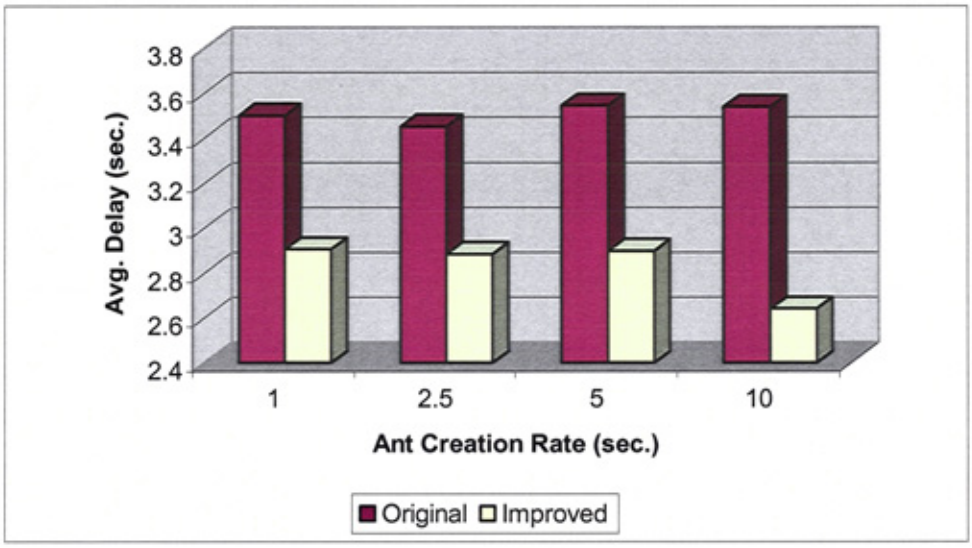

Fig. 6.10: Average packet delay versus ant creation rate 36-node for original and improved antnet algorithms for high loads

6.5.3 Further comments on improved antnet

The modified antnet showed improved average packet delay compared with the original antnet routing algorithm for all cases. For example, when the ant creation rate is set to 10 (1 ant per 10 seconds is created), ~4% and ~33% improvement for 29-node and 36-node irregular grid are observed respectively. With different ant creation rates improvements in the range between 3.5-6.5% for 29-node and 21.5-33% for 36-node configurations are observed. The significant gain achieved with the irregular grid is due to its architectural characteristics. Only two paths are available for packets to travel from left-hand side of the grid to the right-hand side and vice versa. Therefore, packets reroute themselves in order to fully utilise less congested nodes as often as possible.

It was also observed that there is a dramatic decrease in the performance of the antnet algorithm with high loads. This is because the network cannot saturate the number of packets that are created and routed within the network. Furthermore, improvements have the biggest impact on the high loads, as there is more congestion at any given time in the network. Thus, packets are continually being forwarded to different routes in order to avoid congestion which results in more packets being recycled. Moreover, the effect of the reinforcement parameter r is limited within a certain range. Therefore, fluctuations in the traffic do not affect the proposed algorithm's performance as much as in the original version. Fluctuations in the network are expected to happen more often due to the high traffic load.

Moreover, further simulations for ant creation rates greater than 10 showed that the average delay increased and throughput of the algorithm decreased slightly in both algorithms. Therefore, an ant creation rate between 1 and 10 is optimal for the proposed system. However, since node and link failures have not been implemented and investigated in this work, it is not possible to comment on the effect of ant creation rate on the performance of the network in problematic conditions.

6.5.4 Limiting the number of ants

As it can be seen from the previous results, although the ant rate has an effect on the performance, it is not possible to comment on which ant creation rate is the most suitable for different network configurations. Hence, the need for the third improvement is outlined in this section. In the modified algorithm, the number of ants is limited by the ant-to-packet ratio (APR) utilization as in equation 6.3. This value is defined after a number of simulations run by different topologies. Although, this has no major impact on the performance of the algorithm (when compared with results on $1.0 > ant\ creation\ rate$[19] > 0.1), it would increase the adaptability and suitability of the algorithm. In other words it makes the algorithm more generic.

$$Utilization = \frac{NO\ OF\ ANTS\ CREATED}{NO\ OF\ PACKETS\ SEND} = 0.001 \qquad (6.3)$$

6.6 Stagnation Problem

Although the proposed modifications improve the performance of the algorithm, they do not avert stagnation problems. Stagnation occurs when a network reaches its

[19] Ant creation rate defines the frequency (in time) of the forward ants to be created by the node. Therefore, a low ant rate, means a high number of ants in the system.

equilibrium state. This is an undesirable property of the routing algorithm where ants recursively choose the same path to reach their destination. Therefore, routing optimisation may get stranded in the local optima and as a result it may not be able to discover new paths that become optimal due to the changes in network traffic and topology (i.e. due to link/node failures, deleting/adding new nodes or uniform traffic). Moreover, finding the shortest path, by the ants, is a statistical process [Baran & Sosa, 00]. It is highly probable that other ants will use the non-optimal paths chosen by leading ants at the start of the exploration. Several methods such as noise function, evaporation, multiple ant colonies and other heuristics beside the ant colony optimisation have been used and studied in the literature to overcome the stagnation problem [Sim & Sun, 03], [Dorigo & Stutzle, 04]. However, the most commonly used method is the noise function where predefined percentages of the ants are forwarded to random destinations. However, this is only a primitive method and does not provide a solution.

6.6.1 Evaporation

In this work link usage information is used as the second feedback in the system to prevent the stagnation problem. For every link the usage information is held at the node that they are connected to and for a predefined time window (period). Then, routing table entries are reinforced periodically (in the rate of the time window) based on the evaporation information and link usage statistics. The real life scenario influences this, where chemical substances deposited by the ants evaporate over time. Link usage ratio $L(x)$ defined as in (6.4) is used for calculating the evaporation rate $E(x)$ defined in (6.5)[20]:

[20] In [Tekiner et al, 04-03] $a = 0$, $b = 1$.

$$L(x) = \frac{ant\_send(x)}{\sum_{i=0}^{N} ant\_send(i)} \tag{6.4}$$

$$E(x) = \left[1 - \left(ax\frac{(N-R)}{N} + bxL(x)\right)\right]xP(x), \quad \text{where } a + b = 1 \tag{6.5}$$

Equation 6.5 is divided into two parts; the recent usage ranking parameter R and the link usage ratio[21] $L(x)$. The effect of each part is controlled by constants a and b, respectively. If $a = 0$ and $b = 1$, then evaporation is only based on the recent usage ranking, whereas and if $a = 1$ and $b = 0$[22], then evaporation is based on the link usage ratio. For example, in real life scenario, lets consider that there exists N paths from current node y to the destination node d. Then, if the last ant is forwarded through the neighbour node x, then one would expect chemical substance to be laid on the path to x to evaporate the least. The major difference between the real life scenario and antnet scenario is in the simulation environment. Where in the latter we deal with discrete values, therefore probabilities associated with the links cannot be evaporated continuously, and are evaporated at the end of each time window. Therefore, (6.5) measures the proportion of ants send via node x within that time window. On the other hand, the so called ranking parameter represents the node where an ant has laid its pheromone the last among N number of neighbours. For example, $R = 0$ suggests that particular link was the last to be used, thus indicating a pheromone has just been laid. Therefore, least amount of pheromone expected to be evaporated from that node (x).

[21] In our previous work evaporation is defined as link usage ratio only, where $a = 0$ and $b = 1$ [Tekiner et al, 04-03]

[22] When $a = 1.0$ and $b = 0.0$ it is represented as evap(1.0,0.0).

Evaporation rate $E(x)$ for the neighbour x calculated is subtracted from the probability associated with node x, see equation 6.6. The amount of probability evaporated from node x is then distributed equally to the other neighbouring nodes (since, sum of all probabilities for the neighbours is 1), see equation 6.7. Variables that keep track of the number of ants forwarded to the neighbours are reset after a specific time window and new information is gathered at the next time window.

$$P(i) = P(i) - E(x), \quad i = x. \tag{6.6}$$

Since, for all N, the sum of all probabilities is equal to 1, then probability of evaporation from node x distributed equally to the other neighbours is given by:

$$P(i) = P(i) + \frac{E(x)}{N-1}, i \neq x \tag{6.7}$$

Results for the average delay against the ant evaporation rate are presented in Figs. 6.11 to 6.16 for different values of a and b are compared with the case with no evaporation for two different network configurations and three different network loads. It can be seen from the Fig.s for that all cases the algorithm with the evaporation has performed better than the non-evaporation version (evap (0, 0), a and b are both 0, thus $E(x)$ is 0). Results also show that, with the evaporation algorithm the lowest average delay is achieved when the evaporation window is set to 1 sec. Thus showing ~7% and ~16% improvement for 29 random network and 36-node irregular grid, respectively when compared with the non-evaporating algorithm. However, while conducting tests it was observed that the evaporation algorithm was

not as stable as the non-evaporation algorithm. Not much difference was observed on 10 tests conducted by non-evaporation configuration, whereas there was around 0.2 to 0.5 seconds difference between the best and worst cases observed with the evaporation case for 29 and 36 node configurations, respectively. It is believed that this is due to the hot spot traffic model used, and secondly for the algorithm capturing the wrong time window to evaporate the links.

In Fig. 6.11 there is no linear relationship between the ant evaporation rate and the average packet delay. Evaporation rate greater than 4, evap (1,0) shows the best average delay compared to all other cases. However, for lower evaporation rates, evap (0,1) out performs the rest, particularly when evaporation rate is 2.5. Since the network load set was very low the algorithm failed to find the best evaporation window for evaporating and updating the path. Moreover, links should have been evaporated more often in order to give an improved understanding of the current state of the network to the algorithm.

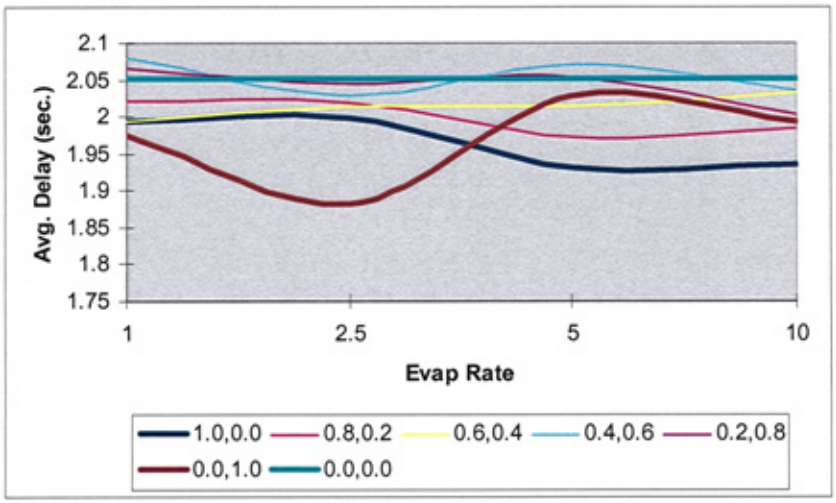

Fig. 6.11: Average packet delay against evaporation rate for different evaporation states, low load and 29-node

Fig. 6.12 shows the results for 29-node random network with the poisson traffic distribution for varying evaporation rates. It can be seen from the Fig. that the maximum delay is observed for evap (0,0) case setting the upper bound and evap (0, 1) setting the lower bound for all values of evaporation rates. This also shows an 8% improvement in terms of overall performance compared with non-evaporation version of the algorithm.

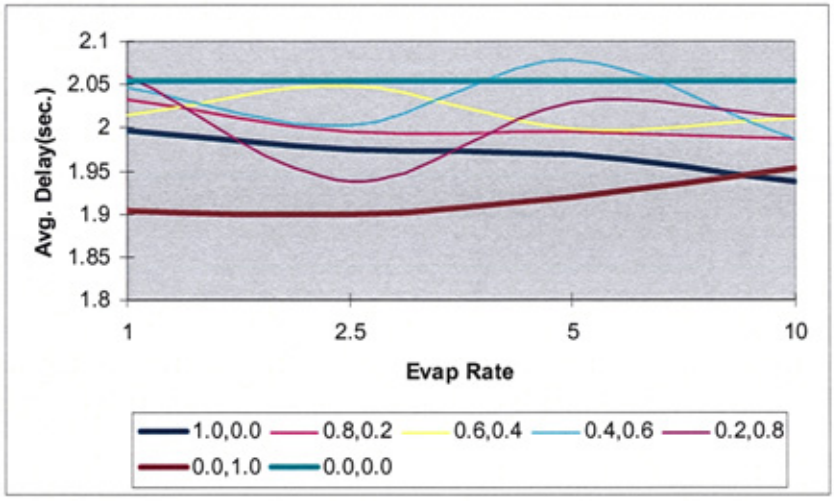

Fig. 6.12: Average packet delay against evaporation rate for different evaporation states, medium load and 29-node

For high load (see Fig. 6.13) the average packet delay is slightly higher for all cases, with evap (0,1) giving the best delay performance, similar in profile of that low load case as in Fig. 6.11. In all cases in Fig. 6.11 and 6.12 the algorithm performed better with low average packet delay for low evaporation rate. This is because of the high rate of packet creation per path at any given time.

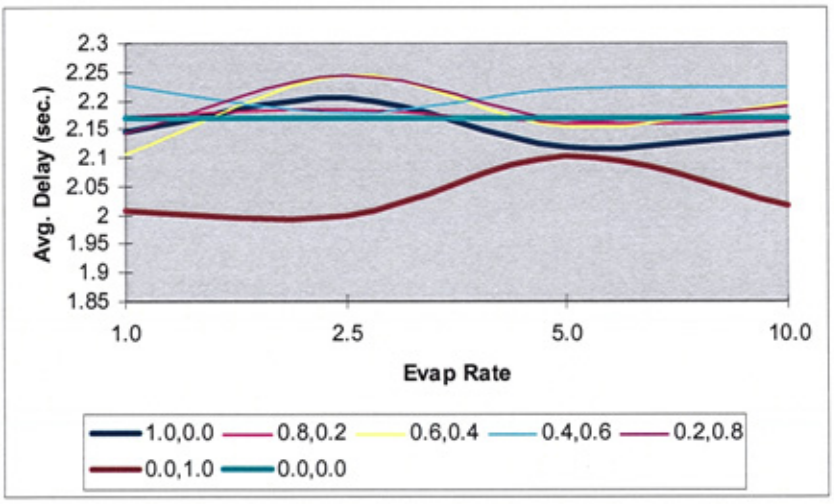

Fig. 6.13: Average packet delay against evaporation rate for different evaporation states, high load and 29-node

For 29-nodes, evap (0, 1) performed the best compared to all other configurations, with ~8% improvement when evap (0, 1) is employed with the improved version of the antnet routing algorithm. The ideal evaporation rate for the evap (0, 1) is 2.5 seconds. Therefore, further tests were run with this evaporation rate for 29-node random network. Fig.s 6.14 to 6.16 compare the effect of *a* and *b* (see equation 6.5) for 36-node irregular grid for three different system loads.

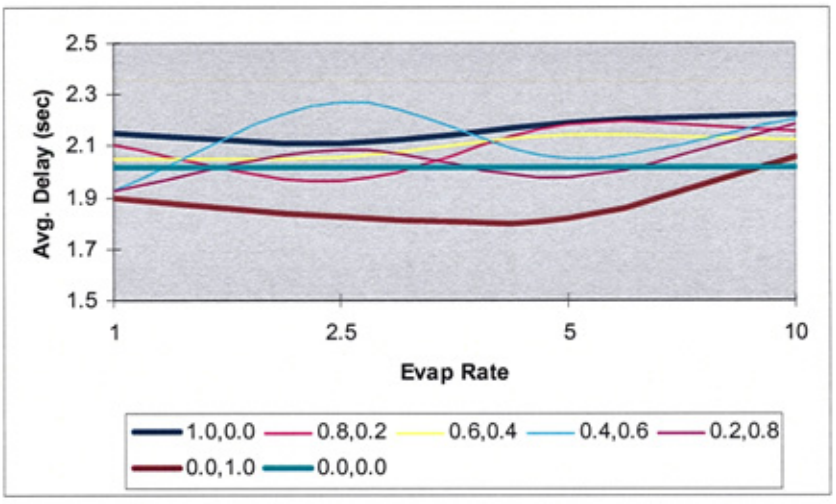

Fig. 6.14: Average packet delay against evaporation rate for different evaporation states, low load and 36-node

On the other hand, this is not the case for the evap (0, 1) as it offers improved performance than all other evaporation rates. This is because there are only two paths that exist from left of the grid to the right of the grid and vice versa. Therefore, in reality link usage count of two links has effect on the performance of the algorithm. However, here link count is applied to all the links thus wrong link evaporation is favoured that performed worse. In addition, packet creation rate worsen the situation where very few packets are created compared to evaporation rate.

In Fig. 6.15 it can be seen that evap (0, 1) performed better than the improved antnet when evap rate is less than 5 for medium loads. Medium load is the ideal network load for this simulation environment where algorithm has more chance compared to other loads to get the best evaporation window. Thus, it has the most impact on the performance of the algorithm. Comparing evap (0, 1) and improved antnet it can be observed that there is 18% improvement for the best case (when evap rate is 1 sec) and 12% for the worst case (when evap rate is 5 seconds).

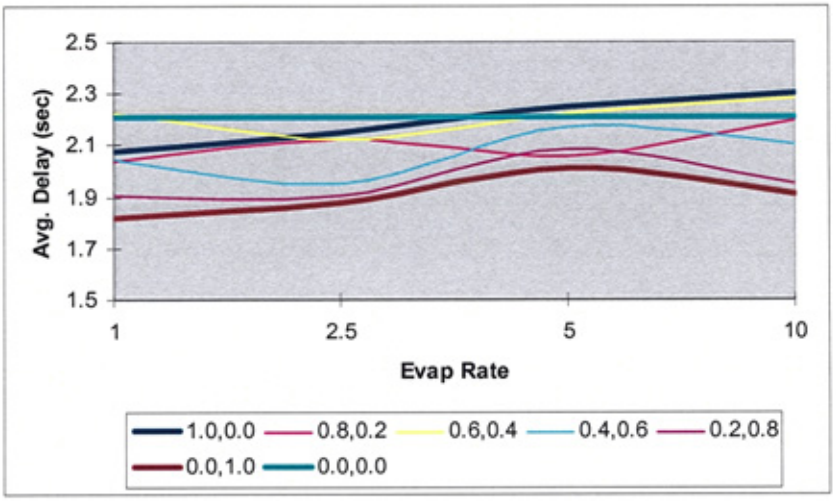

Fig. 6.15: Average packet delay against evaporation rate for different evaporation states, medium load and 36-node

When comparing Figs. 6.16 and 6.15, it can be seen that the trend of the average packet delay is completely opposite. With the high load it is expected that evaporation rate would have the opposite effect. Irregular grid is a special test case where algorithm needs to learn the usage of longer path from one side of the grid to another side. In this particular case, network gets congested heavily with high packet creation, therefore the time taken by the routing algorithm to process each packet increases and as a result the algorithm performance improves when operating at high evaporation rate.

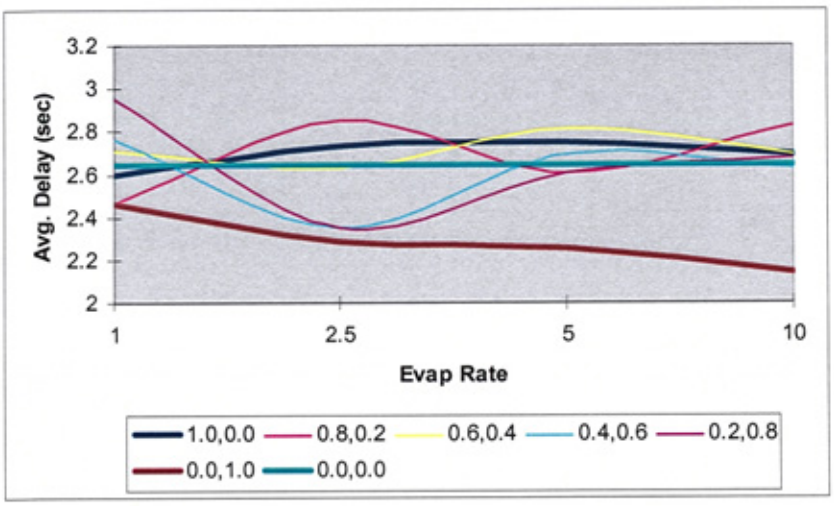

Fig. 6.16: Average packet delay against evaporation rate for different evaporation states, high load and 36-node

6.6.2 Multiple ant colonies

The idea of using multiple ant colonies was first applied to wavelength routing in optical wavelength division multiplexing [Varela & Sinclair, 99]. In this work, more than one ant colony was used to distribute different wavelengths over the network in order to accomplish increased availability of wavelengths routing. In the original antnet, ants are only attracted by the pheromone trails laid by the ants from their own species (as there is only one colony considered at any given time). Also introduced in

this work was the concept of ants being repelled by the pheromone laid by other species, in addition ants being attracted to the pheromone trails laid by their own species. Three variants of the algorithms have been proposed namely: local update, global update/distance and global update/occupancy [Varela & Sinclair, 99]. In the local update and the global update table entries are updated every hop and at the end of each cycle[23], respectively. In the global update two variants (i.e. different repulsion techniques) are used to distribute wavelengths within the network based on the distance or the link occupancy.

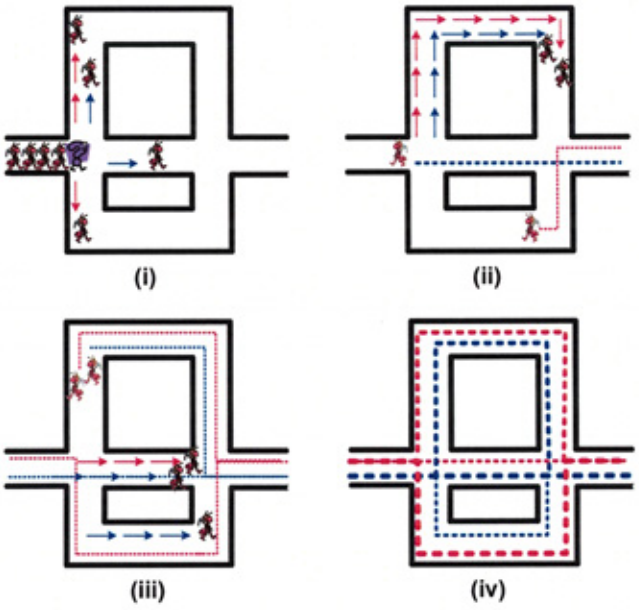

Fig. 6.17: Interaction of multiple ant colonies

Multiple ant colonies have also been applied in circuit switched networks, assuming that there exists more than one ant colony and no information is given on ant colonies repelling or attracting each other as reported in [Sim & Sun, 02]. Here, the application of multiple ant colony algorithms in packet-switched networks has been reported.

[23] An ant is said to complete its cycle when it completes it journey from source node to the destination node.

Here we investigate two ant colonies as illustrated in Fig. 6.17, where red and blue ants each use different pheromone tables. The red ants only update the pheromones laid by the red colonies and so on. We have shown that using two colonies, more than one optimal path becomes available for every source destination pair and a higher load balancing is achieved. As a result, more than one routing table is used (two in this case). The costs of adopting multiple ant colonies are the need for increased resources and system complexity (for further analysis see [Tekiner et al, 04-2]).

Fig.s 6.18-6.20 shows the throughput versus time for five different antnet implementations for 29-node random-net and three different system loads. In Fig.s 6.18 and 6.19 antnet employing multiple colonies performed the best where throughput is 800 kb/sec higher than the original antnet. The sharp decrease around 135 second is due to packet creation being stopped at this stage and packets left in the buffers are being processed. Since throughput was highest with multiple antnet it finished the simulation first whereas original antnet finished the simulation last.

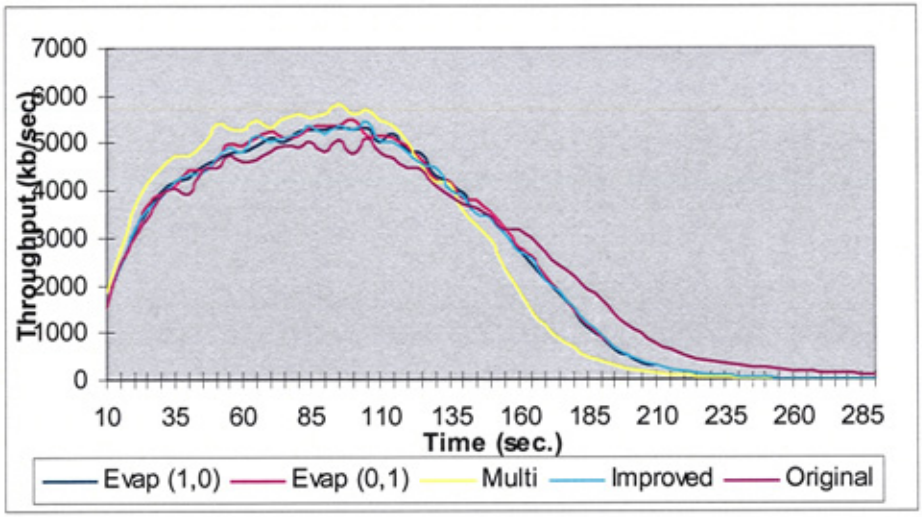

Fig. 6.18: Throughput versus time for different antnet configurations and low load

As shown in Figs. 6.18 and 6.19 there is very little difference between the results, thus choosing which to adopt becomes rather a daunting task. Thus one can say that, employing evaporation have little effect on the throughput performance of the network. Antnet generally has no effect on the throughput, which is also confirmed by the results presented in [Di Caro, 04], [Dorigo & Stutzle, 04].

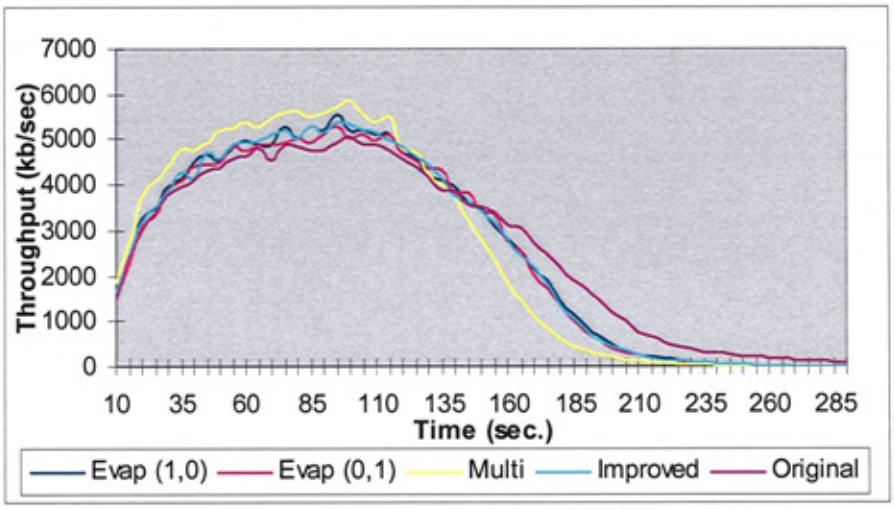

Fig. 6.19: Throughput versus time for different antnet configurations and medium load

In the high loads case however, it can be seen from Fig. 6.20 that multiple ant colonies has no effect on the throughput. On the other hand, the original antnet algorithm still performs the worst. This is because, packets that are routed within the network without reaching their destination decrease the overall throughput.

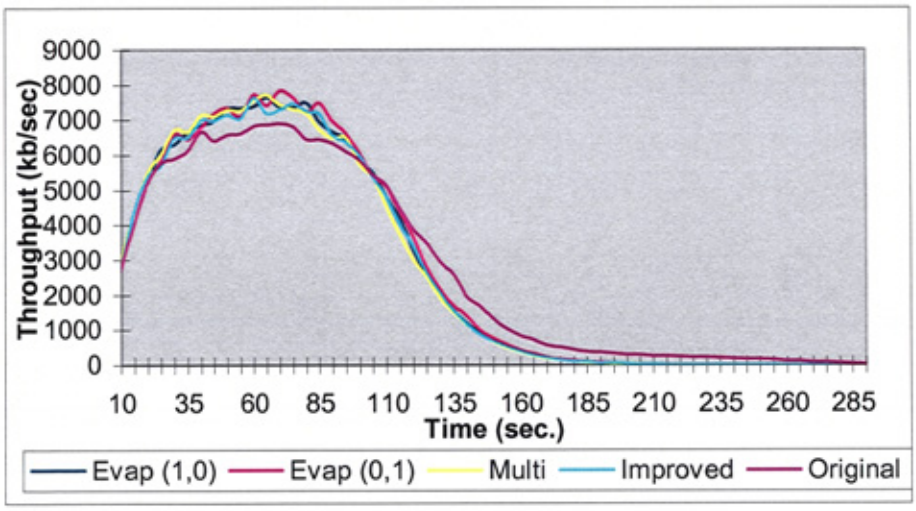

Fig. 6.20: Throughput versus time for different antnet configurations and high load

Fig.s 6.21-6.23 shows the throughput for 36-node irregular grid versus time for various antnet implementations for three loads. It can be seen from the Fig.s that multiple antnet is no longer the best performing configuration. This is due to the fact that in 36-node irregular grid there are only two paths available from left to the right of the grid with one path being the shortest route (in terms of distance but not fastest). Thus, in the case of two colonies being employed, they do not have a chance to balance the available paths but switch between them.

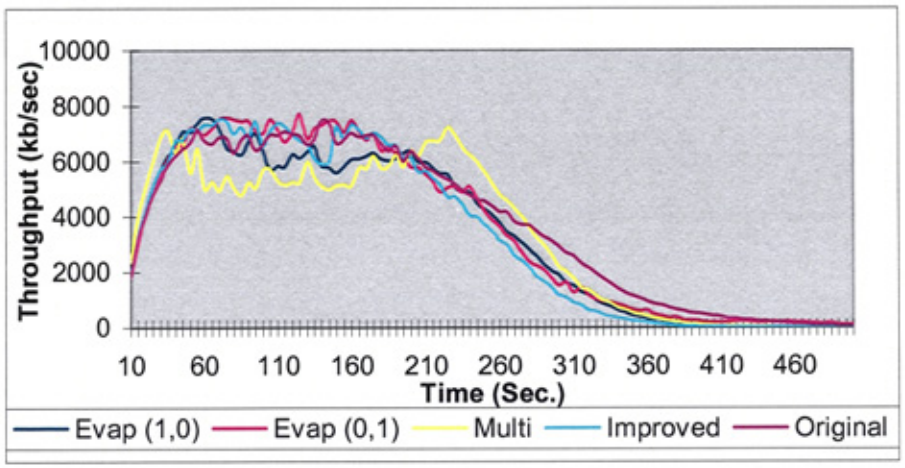

Fig. 6.21: Throughput versus time for different antnet configurations and low load

From Fig.s. 6.21 and 6.22 it can be seen that on average evap (0, 1) performed slightly better than the others. This shows that with low and medium loads algorithm managed to capture the sub paths leading to the two main paths that are connecting the left of the grid to the right of the grid. On the other hand, it can also be seen that evap (1, 0) failed to rank correctly the consecutive sub paths as some packets and ants had to go through loops.

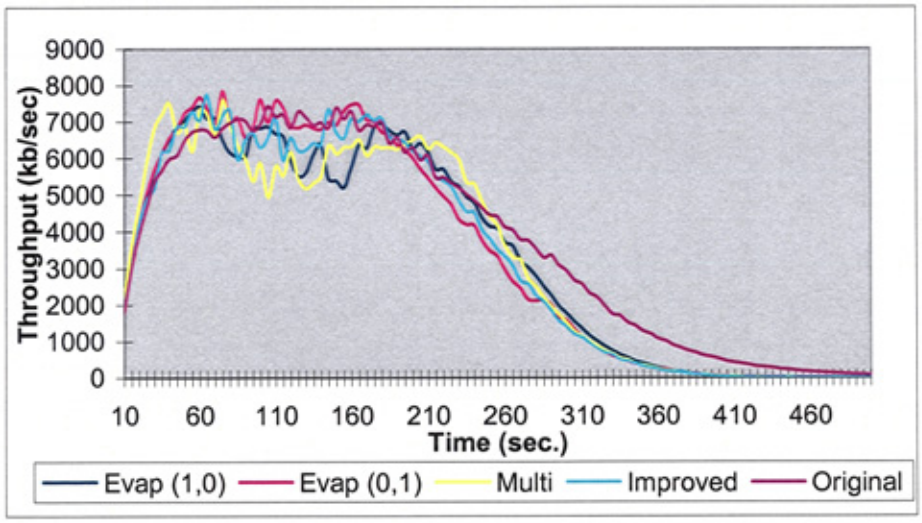

Fig. 6.22: Throughput versus time for different antnet configurations and medium load

Fig. 6.23 shows the only results where the original antnet algorithm performed better than the others. This is because both evaporation configurations are misguided by the continuous change that happens in the ideal paths. In other words, ants and packets have been forwarded to the non-ideal paths due to congestion occurring at almost every node. However, this does not mean that they reach their location late; it only means that they reach their destination via a number of additional hops.

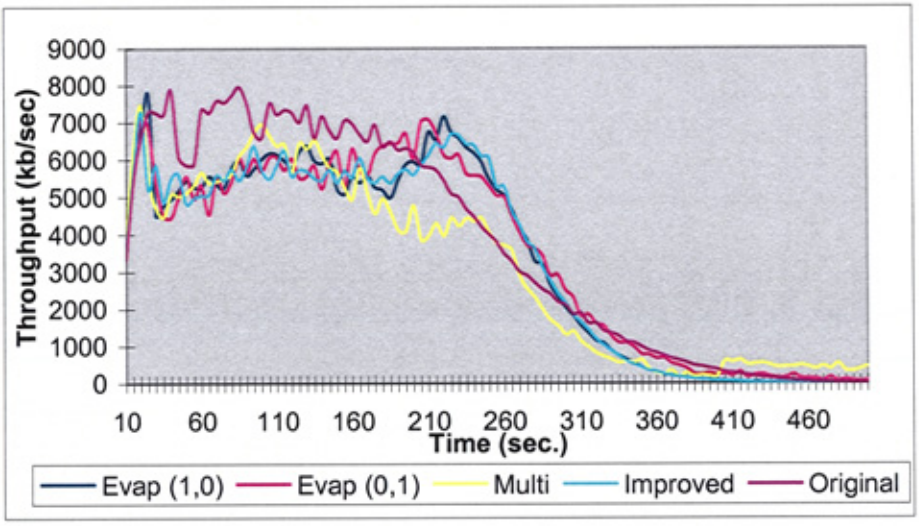

Fig. 6.23: Throughput versus time for different antnet configurations and high load

6.7 Effect of *Y* on Antnet

In [Dorigo & Stutzle, 04] it has been highlighted that the confidence parameter *Y* has a major effect on the performance of the antnet routing algorithm. No evaluation of the network performance has been made that takes *Y* into account. Therefore, here for the first time the effect of *Y* is compared by using different antnet configurations. *Y* defines the effect of reinforcement signal *r* on the optimisation. In other words, if *Y* is 1 then reinforced value changes the current value completely. However if *Y* is 0 then reinforced values has no effect on the performance. Therefore, effect of *Y* on the performance of the proposed versions of the algorithm needs to be investigated. It has been reported in [Di Caro & Dorigo, 98-4] that the ideal value of *Y* for the original antnet routing algorithm is 0.75- 0.8. Hence, in the previous tests *Y* was set to 0.8.

Fig.s 6.24-6.26 shows the average delay versus confidence parameter *Y* for various antnet configurations. For Fig. 6.24 the average packet delay increases as *Y* decreases. Evap (0,1) gives better performance than the others at high values of *Y*.

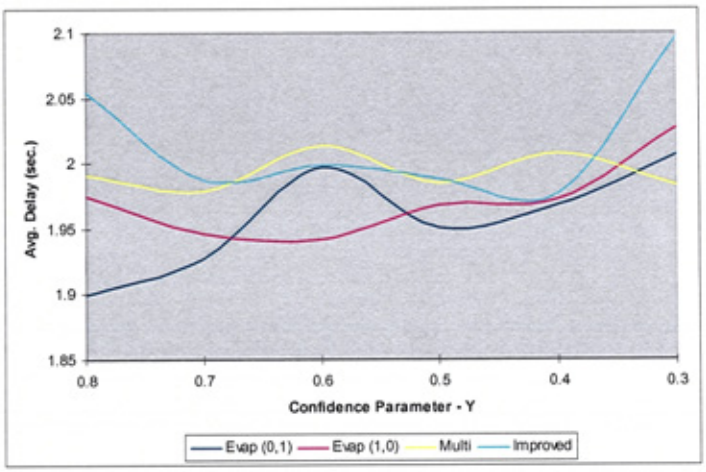

Fig. 6.24: Confidence parameter Y versus the average delay for different antnet configurations with low load and 29 Node

On the other hand in Fig. 6.25 as *Y* increases the average delay remained almost unchanged for all configurations except for evap (0, 1) where there is a cyclic delay characteristic. When *Y* is equal to 0.8 and 0.4, evap (0, 1) gives the lowest delay.

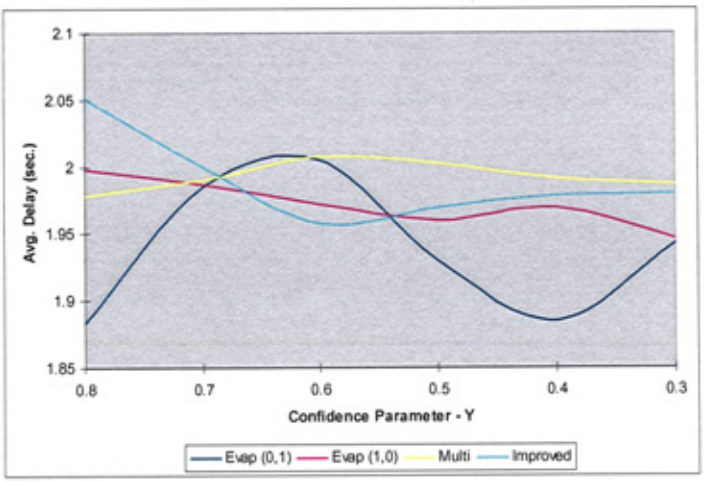

Fig. 6.25: Confidence parameter Y versus the average delay for different antnet configurations with medium load and 29 Node

Fig. 6.26 shows the average delay for different values of *Y* for high system loads. It can be seen that unlike previous results evap (0, 1) performs better only when Y is set to 0.8 and 0.4 similar to Fig. 6.25. This shows that the default confidence level used is best suited to this system load.

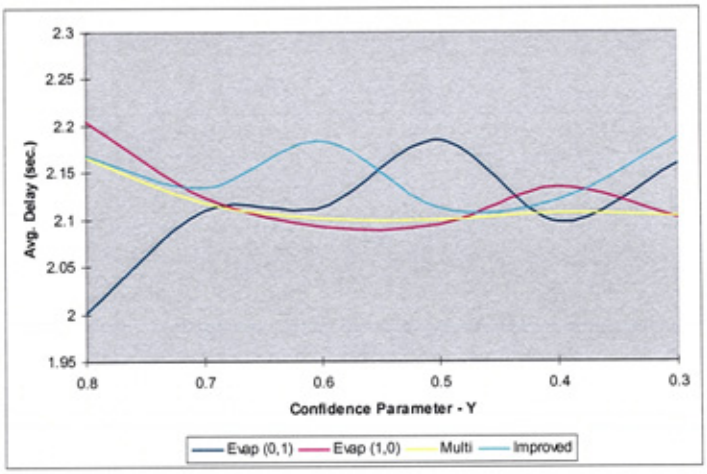

Fig. 6.26: Confidence parameter Y versus the average delay for different antnet configurations with high load and 29 Node

Fig.s 6.27 to 6.29 shows the results for the irregular grid for a range of *Y* values and for different system loads. Unlike to randomnet, evap (0, 1) performs better in almost every occasion for all values of *Y*. However, evap (0, 1)'s performance was not as consistent as the others. Fig. 6.27 show that evap (0, 1) gives the best performance when *Y* is set to 0.3 for low system loads. This is probably because algorithm is only limited to two main routes for finding the best path from each sides of the grid. Therefore a low confidence parameter is more suited to the irregular grid.

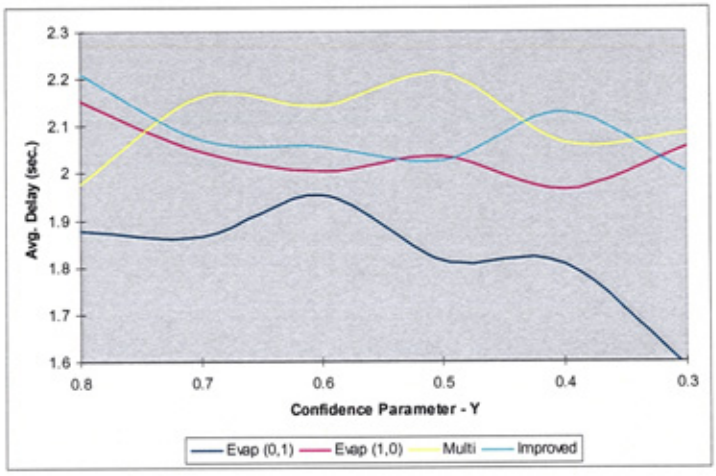

Fig. 6.27: Confidence parameter Y versus the average delay for different antnet configurations with low load and 36 Node

Fig. 6.28 show that evap (0, 1) offers improved performance for $Y = 0.5$ and low system loads, whereas for other values of Y the performance is similar to others as in the previous results.

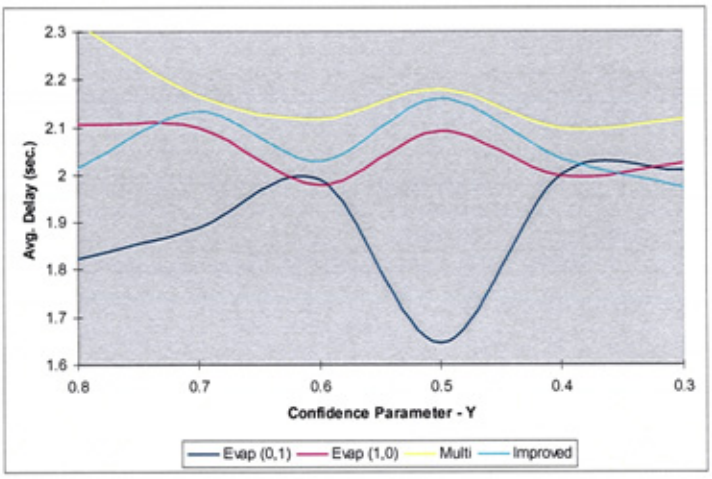

Fig. 6.28: Confidence parameter Y versus the average delay for different antnet configurations with medium load and 36 Node

In Fig. 6.29, there is almost 0.8 sec. difference on the average delay per packet when Y is set to 0.6 and 0.4 for evap (0, 1). This is because at a high network load the algorithm adaptation to changes becomes rather challenging.

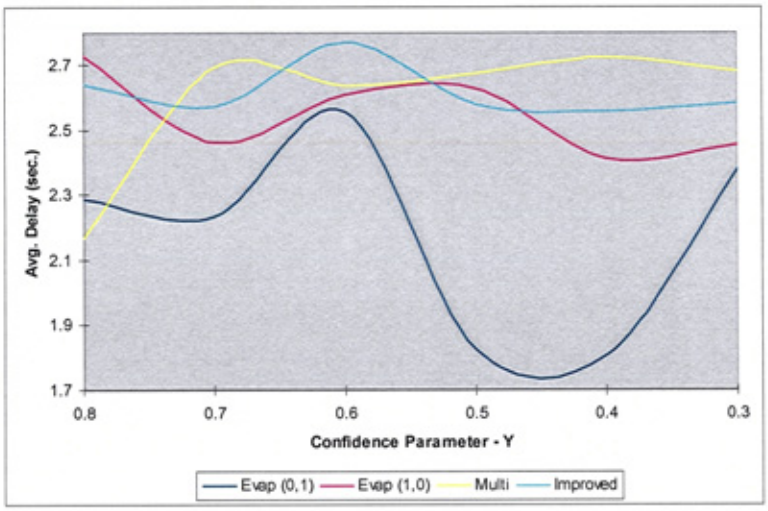

Fig. 6.29: Confidence parameter Y versus the average delay for different antnet configurations with high load and 36 Node

6.8 Comparison

Results for the average packet delay for the original, improved, evaporation and multiple ant colonies are given in Tables 6.1 and 6.2 for three different system loads. It can be seen from the tables that for all cases the algorithm with evap (0, 1) offers improved performance compared with the others. Also in some cases improved antnet performed better than the multiple antnet and vice versa. The original antnet performed the worst in all cases. Results also show that, the evaporation algorithm has performed at least 10% better than the improved antnet in every case for 29-node network and around at least 22.5% for 36-node irregular grid. Also, evap (1, 0) performed worse than evap (0, 1). However, while conducting the tests it was observed that evaporation algorithm was not as stable as the non-evaporation algorithm. There was not much difference on the 10 tests conducted by non-evaporation configuration. On the other hand, there is around 0.2 to 0.5s difference among the best and worst cases observed with the evaporation case for 29 and 36 node configurations, respectively. It is believed that, this is due to two main reasons. Firstly, due to the non-uniform traffic model used in the tests, and secondly the algorithm capturing the wrong time window as that is the only factor on choosing the time to evaporate.

Table 6.1: Overall average delay comparison for different system loads

Avg Delay(s)	L-Low	L-Mid	L-High
Evap (1,0)	1.99	1.97	2.20
Evap (0,1)	1.88	1.89	2.00
Multi	1.97	1.90	2.16
Improved	2.05	2.05	2.16
Original	2.12	2.10	2.28

Table 6.2: Overall average delay comparison for different system loads

Avg Delay(s)	L-Low	L-Mid	L-High
Evap (1,0)	2.11	2.15	2.73
Evap (0,1)	1.82	1.88	2.29
Multi	2.32	1.98	2.17
Improved	2.01	2.21	2.64
Original	2.495	2.77	3.54

Tables 6.3 and 6.4 show the average throughput of the network for the original, improved, evaporation and multiple antnet for three different system loads. As can be seen from the Fig.s the evap (0, 1) and multiple antnet outperformed others in all cases. On the other hand, in some cases packet creation has been stopped around 100 seconds and 200 seconds (every node in the network stops creating packets after 2500 packet creation.) for 29 and 36 node configurations, respectively. When a network is congested multiple ant colonies performed better than the others with 29-node network and with the irregular grid, multiple ant colonies performed the worst. This is because there are only two paths available from left of the grid to the right and from the right of the grid to the left. Therefore, each colony can dominate only one path and cannot explore more paths.

Table 6.3: Overall throughput comparison for different system loads

Throughput(kb/s)	L-Low	L-Mid	L-High
Evap (1,0)	1838.80	1864.59	2320.91
Evap (0,1)	1887.62	1813.60	2217.04
Multi	1864.91	1986.88	2118.02
Improved	1832.94	1869.88	2100.60
Original	1620.58	1590.89	1593.30

Table 6.4: Overall throughput comparison for different system loads

Throughput(kb/s)	L-Low	L-Mid	L-High
Evap (1,0)	2380.39	2485.92	2721.89
Evap (0,1)	2421.63	2470.46	2574.70
Multi	2589.00	2692.17	2711.07
Improved	2471.21	2412.01	2612.92
Original	2031.48	1946.90	2130.65

In addition, tests were carried out for different values of confidence parameter Y in order to observe its effect on the performance of the improved and new versions of the algorithm. For all the tests with 29-node grid, the ideal Y value is 0.8 as the original authors of the antnet have suggested it. On the other hand with 36-node irregular grid, the ideal Y value is around 0.4 and 0.5. Therefore, one can say that for extreme conditions as in irregular grid the confidence in applying the reinforcement signal is 50%.

6.9 Conclusions

In this chapter, it has been shown that by detecting and dropping packets that travel continuously within the network one can improve the antnet's performance in terms of network throughput and the average packet delay. The effect of traffic fluctuations on the network performance has been limited by the introduction of boundaries, by limiting the number of ants within the network at any given time.

One of the major problems with the antnet is stagnation and adaptability. This occurs, when the network freezes and consequently the routing algorithm gets trapped in local optima and is therefore unable to find new improved paths.

In the original antnet routing algorithm, the only feedback signal used was the reinforcement applied by using the agents' trip times. Here, in addition the link usage statistics were used as a feedback signal and reinforced to the solution by the evaporation parameter to improve the performance of the algorithm.

In addition multiple ant colonies were applied to packet switched networks. It was shown that, by employing multiple ant colonies in packet switched networks,

throughput can be increased. On the other hand, no gain in average packet delay was observed. One major disadvantage of using multiple ant colonies is the resources used in every node. More colonies will be required (thus routing tables), when applying multiple ant colonies to larger sized networks in order to exploit more paths. No direct interaction among the different ant colonies was considered. Thus, methods used in [White et al, 98-2] could be applied to improve the performance of multiple ant colonies in packet switched networks. On the other hand, further tests showed that evaporating path probabilities and applying multiple ant colonies had no effect on uniform and non-problematic traffic conditions. This was an expected outcome as there was no need to change the solution found if traffic stays static at all the time. When network conditions do not change, there are no new improved paths available. Thus, the solution found was the optimal solution.

In [Di Caro, 04] and [Dorigo & Stutzle, 04] results for OSPF, RIP and Q-routing algorithms are compared with antnet and it is reported that antnet performed better in all of the cases. In this thesis implementation and initial tests with these algorithms have also verified these results. Therefore, extensive tests with these algorithms have not been carried out in this thesis and the focus has been only on the antnet and its improvements. However, hybrid version of IGRP and Q-routing algorithms using ants has also been implemented as part of this work with no performance gain.

CHAPTER VII - CONCLUSION AND FURHER WORK

7.1 Conclusions

The primary objective of this work was to model and analyse distributed routing algorithms that are suitable for packet switched networks. This thesis divided into two main threads first of which proposed a model to implement logical network topologies and provided an investigation of these with some improvements for OTDM systems. The second thread investigated routing algorithms for arbitrary networks. It started with a major review of traditional routing algorithms that are in use in today's networks and this was followed by a review of reinforcement learning approaches to the routing problem. The antnet routing algorithm has been implemented and various modifications have been proposed. A second reinforcement signal has been used for the first time in the literature to overcome the stagnation problem together with the first implementation of multiple ant colonies on packet switched networks. A novel way of simulating communication networks has been proposed and algorithms have been implemented using this model.

In Chapter 2, a review of the most common non-blocking type logical network topologies has been given. This provided an insight to the logical network topologies with a focus in routing algorithms. Comparison of the reviewed algorithms was also provided.

Since logical network topologies are limited, Chapter 3 gave a review of the traditional routing algorithms together with routing protocols that are currently in use today's networks. Dijkastra's shortest path algorithm is one of the well known algorithms which is widely adapted in the Internet and studied in the literature. Dijkastra's algorithm is also the base for the Link State Routing algorithm and protocols. OSPF is deployed throughout the internet and is the de facto routing algorithm within the networks. EGP is a gateway protocol which is mainly used between backbones and as it name suggests, among the gateways. However, BGP is the most widely used gateway protocol.

Traditional routing algorithms and protocols lack intelligence, they need human assistance and interpretation in order to adapt themselves to failures and changes. Moreover, they are mainly table based which makes search process highly undesirable. Due to these problems, there is a great amount of research concentrated on finding an intelligent distributed routing algorithm.

In recent years, agent based systems and reinforcement learning have been widely applied to routing. Swarm intelligence particularly ant based systems, Q-learning methods and hybrid agent based distance vector algorithms have shown promising and encouraging results. Therefore, Chapter 4 presented an introduction to reinforcement learning and ant colony optimisation, followed by a detailed review of the antnet routing algorithm and its evolution. Finally, a review of the well known Q-routing algorithm and its derivatives were given.

The aim of Chapter 5 was to simulate and evaluate the logical network topologies suitable for OTDM networks which operate in synchronous mode since there is no information on such evaluation in the literature to the best knowledge of author. Therefore, a generic model for implementing logical interconnection topologies in software was proposed in order to investigate the performance of the logical topologies and their routing algorithms for packet based synchronous networks. This model proposed is generic for any synchronous transfer modes and therefore can be used for implementing any logical topology using any programming language. Three topologies have been investigated and implemented namely: Shufflenet, De Bruijn graph and Gemnet. Results for the average packet delay showed that the De Bruijn graph performed the worst. Also, it is observed that the De Bruijn graph made use of buffering more efficiently compared to the other algorithms.

In addition, a simple priority rule named as TRR was introduced in order to utilise the usage of the asymmetric 2x2 node. Results showed that TRR has no effect on the performance of the topology and the algorithm. However, it increased the utilisation of the transmission port significantly. Also, it was observed that since there are no multiple paths existing in the De Bruijn graph therefore no gain was observed.

In Chapter 6, it was shown that by detecting and dropping packets that travel continuously within the network one can improve the antnet's performance in terms of network throughput and the average packet delay. The effect of traffic fluctuations on the network performance can be limited by the introduction of boundaries, (i.e. by limiting the number of ants within the network at any given time). Although limiting

the number of ants increased the network utilisation, it reduced the chance of finding the best and new routes and detecting failures and problematic conditions.

One of the major problems with the antnet is stagnation and adaptability. This occurs, when the network freezes and consequently the routing algorithm gets trapped in the local optima and is therefore unable to find new improved paths.

In the original antnet routing algorithm the only feedback signal used was the reinforcement applied by using the agents trip times. Here, in addition the link usage statistics were used as a second feedback signal and reinforced to the solution by evaporation parameter to improve the performance of the algorithm. The evaporation parameter was calculated based on the link usage statistics held by every node. Simulation results showed that under non-uniform traffic conditions the algorithm with evaporation displayed reduced average delay per packet compared with algorithm with no evaporation. Moreover, the simple rules and unsophisticated variables used in the improved version did not add a significant overhead to the network and to the node. However, inconsistency observed in the average delay was the main drawback observed during the simulations.

In addition, multiple ant colonies were applied to packet switched networks. It was shown, that by employing multiple ant colonies in packet switched networks, throughput can be increased. On the other hand, no gain in the average packet delay was observed. One major disadvantage of using multiple ant colonies is the resources used in every node.

Results for OSPF, RIP and Q-routing algorithms were compared with the antnet showing that the later offered improved performance in all cases [Di Caro, 04] and [Dorigo & Stutzle, 04]. Our implementation and initial tests with these algorithms has also verified these results. Therefore, extensive tests with these algorithms have not been carried out and our work focused on the antnet routing algorithm and to its improvements. However, hybrid version of IGRP and Q-routing algorithms using ants has also been implemented as part of this work with no performance gain.

In summary, this thesis has presented the design, analysis, and simulation results of routing algorithms suitable for both logical and arbitrary networks. The primary objective of this study was to investigate the potential and limitations of such networks together with their routing algorithms. Noise and crosstalk characteristics of OTDM switch and stagnation in antnet routing algorithm have been addressed. A novel way of simulating communication networks has been proposed and used.

7.2 Further work

In this section, the remaining outstanding issues and potential improvements that are believed to be interesting and deserve future investigation are presented.

[Abbattista et al, 95] and [Liang et al, 02] used genetic algorithm to optimise ant colony for the TSP and routing problem, respectively in offline system [Chen, 97]. In both methods information gathered from the algorithm during offline simulations were used to tune parameters of the algorithms [Botee & Bonabeau, 99]. However, in real systems, routing problems must be addressed and resolved in real time to ensure improved performance. Therefore one can employ distributed genetic algorithm to

optimise the constant variables used in antnet routing algorithm. For ant based system GA mutation operation can be applied in two different ways. (i) using the probability of randomly selected path(s) to increase the exploration probability of the network, and (ii) a more traditional technique where mutation operator is applied to the path (sequence) that backward ant follows by changing some of the entries based on the mutation operator used.

In the electrical domain the routing table could be extremely large (> 500,000 entries) requiring a large amount of computation time. In many applications this could act as the bottleneck, thus preventing further network expansion. In optical domain, the problem becomes even more problematic, since there are no optical memories. Thus, one has no option but to use greatly reduced routing tables in optical domain. To overcome the problem of searching a large size routing table an alternative routing algorithm with dynamic and intelligent characteristics might be considered. The ant based routing algorithm, with the ability to adapt and learn, is one such alternative. In addition, unlike the original unsupervised learning approach, the biased and unbiased ants with prior knowledge could be used to make decisions in some cases (i.e. traditional routing protocols and algorithms such as BGP, OSPF and IGRP could run at the background on different processor with an associated GA running on different processor). In addition to the antnet routing algorithm other reinforcement learning approaches such as Q-routing algorithm can also be used together with GA.

The pioneer of agent based routing algorithm [Appleby & Steward, 94] used different agents to solve routing problem while keeping other ants in the network in order to eliminate the need for network administrators. As a future work similar approaches

can also be used, where worker ants representing data packets within the network can only change the pheromone levels of path that they travel and also use low priority queues. However, beside the worker ants there can be other ants with higher priorities queues carrying high priority packets that can travel faster (using high priority queues) within the network and be able to update more than one entry in the routing table. Importance of the packet could also be defined by its length, age, and closeness to its destination. In addition, there could also be controller ants which will measure network load and number of ants passing through the network and feed this information to the algorithm.

In addition IGRP, OSPF and BGP will also be optimised in real time by the GA model developed for the antnet. Since these currently need human interpretation in real life in order to adapt to changes occurring in the network.

The author ultimately believes that further study in intelligent routing algorithms should exploit these revelations to produce algorithms for the next generation communication networks.

References

[Abbattista et al, 95]: F. Abbattista, N. Abbattista, and L. Caponetti, "An Evolutionary and Cooperative Agents Model for Optimization", Proc. of Int. Conf. on Evolutionary Computation, Perth, Nov 1995, pp. 668 - 671

[Acampora & Karol, 89]: A. S. Acampora and Mark J. Karol, "An Overview of Lightwave Packet Networks", IEEE Network, 1989, pp.29 - 41

[Acampora et al, 88]: A. S. Acampora, M. J. Karol and M. G. Hluchyj, "Multihop Lightwave Networks: A New Approach to Achieve Terabit Capabilities", in Proc. ICC'88, vol. 1, 1988, pp. 1478 - 1484

[Akon et al, 04]: M. M. Akon, D. Goswami and S. A. Jyoti, "A New Routing Table Update and Ant Migration Scheme for Ant Based Control in Telecommunication Networks", Parallel Architectures, Algorithms and Networks, Proceedings. 7th International Symposium, 10-12 May 2004, pp. 508 - 513

[Amin & Mikler, 02-1]: K. A. Amin, A. R. Mikler "Towards Resource Efficient and Scalable Routing: An Agent-based Approach", 2002

[Amin & Mikler, 02-2]: K. A. Amin and A. R. Mikler, "Dynamic Agent Population in Agent-based Distance Vector Routing", ISDA 2002: Second International Workshop on Intelligent Systems Design and Applications, Atlanta, USA, Augt 2002, pp. 195 - 200

[Amin et al, 01]: K. Amin, J. Mayes and A. Mikler, "Agent-based Distance Vector Routing", Lecture Notes In Computer Science, Vol. 2164, Springer-Verlag, London, 2001, ISBN:3540424601, pp. 41 - 50

[Amin et al, 04]: K. Amin, A. Mikler, "Agent-Based Distance Vector Routing: A Resource Efficient and Scalable Approach to Routing in Large Communication

Networks", Journal of Systems and Software, Vol. 71, Issue 3 May 2004, pp. 215 - 227

[Andrei, 02]: S. Andrei, "A survey on network routing and reinforcement learning", 2002, retrieved on Jun 2004 from:

www.math.tau.ac.il/~mansour/rlcourse/student_proj/asuciu/network1.htm

[Appleby & Steward, 94]: S. Appleby and S. Steward, "Mobile Software Agents for Control in Telecommunication Networks", British Telecom Technical Journal, Vol. 12, 1994, pp. 104-113.

[Banerjee & Sarkar, 94]: S. Banerjee and D. Sarkar, "Hypercube Connected Rings: A Fault-Tolerant and Scalable Architecture for Virtual Lightwave Network Topology," Proc. IEEE INFOCOM '94, Boston, June 1994, pp. 1236 - 1243

[Banerjee et al, 99]: S. Banerjee, V. Jain and S. Shah, "Regular Multihop Logical Topologies for Lightwave Networks", IEEE Communications Surveys & Tutorials, Vol. 2, No. 1, First Quarter 1999, pp. 2 - 18

[Banks et al, 00]: J. Banks, J. S. Carson, B. L. Nelson and D. M. Nicol, "Discrete-Event System Simulation", Prentice Hall, 3rd Edition, ISBN: 0130887021, August 15, 2000.

[Baran & Sosa, 00]: B. Baran, R. Sosa, "A New Approach for AntNet Routing", ICCCN-2000 Computer Communications and Networks, 2000, Las Vegas, 16-18 Oct. 2000, pp. 303 - 308

[Baran, 01]: B. Baran, "Improved AntNet Routing", ACM SIGCOMM, Workshop on Data com. in Latin America and the Caribbean, Vol. 31, No. 2, April 2001, pp. 42 – 48, ISBN:0146-4833.

[Baransel et al, 95]: C. baransel , W. Dobosiewicz and P. Gburzynski, "Routing in Multi-hop Packet Switching Networks: Gbps Challenge", IEEE Network, Vol. 9, No. 3, 1995, pp. 38 - 61

[Bassil & Cruz, 96]: J. Brassil and R. Cruz, "Nonuniform Traffic in the Manhattan Street Network", Journal of Performance Evaluation, Vol. 25, 1996, pp. 233 - 242

[Beckers et al, 92]: R. Beckers, J.L. Deneubourg, and S. Goss, "Trails and U-turns in the Selection of a Path by the Ant Lasius Niger," Journal of Theoretical Biology, Vol. 159, 1992, pp. 397 - 415

[Bellman, 59]: R.E. Bellman, "On a Routing Problem", Quarterly of Applied Mathematics, Vol. 16, 1959, pp. 87 - 89

[Bertsekas, 91]: D. Bertsekas, "Data Networks", 2nd Edition, Prentice Hall, 1991, ISBN: 0132009161

[Bhuyan & Agrawal, 84]: L. N. Bhuyan and D.P. Agrawal, "Generalized Hypercube and Hypercube Structures for a Computer Network," IEEE Trans. Computers, Vol. C-33, No. 4, April 1984, pp. 323 - 333

[Bonabeau et al, 00]: E. Bonabeau, M. Dorigo and G. Theraulaz, "Inspiration for Optimization from Social Insect Behaviour", Nature, Vol. 406, July 2000, pp. 39 - 42

[Bonabeau et al, 98]: E. Bonabeau, F. H´enaux, S. Gu´erin, D. Snyers, P. Kuntz, and G. Theraulaz, "Routing in Telecommunications Networks with "smart" Ant-like Agents", Proceedings of IATA '98, Second International Workshop on Intelligent Agents for Telecommunication Applications, Lecture Notes in Artificial Intelligence, Vol. 1437, Springer-Verlag, 1998, pp. 60 - 71

[Bonabeau et al, 99]: E. Bonabeau, M. Dorigo, and G. Theraulaz, "Swarm Intelligence from Natural to Artificial Systems", Oxford University Press, 1st Edition, 1999, ISBN: 0195131592

[Botee & Bonabeau, 99]: H.M. Botee and E. Bonabeau, "Evolving Ant Colony Optimization", Advances in Complex Systems, Vol. 1, No. 2/3, 1999, pp. 149 - 159

[Boyan & Littman, 94]: J. A. Boyan and M. L. Littman., "Packet Switching in Dynamically Changing network: A Reinforcement Learning Approach", Proceedings of the Advances in Neural Information Processing Systems, 1994, 671-678.

[Camazine et al, 01]: S. Camazine, J. L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz and E. Bonabeau, "Self-Organization in Biological Systems", Princeton University Press, April 1, 2001, ISBN: 0691116245

[Cantu-Paz, 97]: E. Cantu-Paz, A survey of parallel genetic algorithms, Tech. Rep., The University of Illinois, 1997, retrieved on May 2004 from: ftp://ftpilligal.ge.uiuc.edu/pub/papers/IlliGALs/97003.ps.Z.

[Carnahan & Simha, 01]: J. Carnahan and R. Simha "Nature's algorithms", IEEE Potentials, April/May 2001

[Chambers, 95]: L. Chambers, "Practical Handbook of Genetic Algorithms , Applications Volume 1", CRC Press, 1995 , Chp 11, Scheduling and routing problems, pp. 367 - 430

[Chen, 97]: K. Chen, "A simple Learning Algorithm for the Travelling Salesman Problem", Physical Review E, Vol. 55, 1997, pp. 7809 - 7812

[Choi & Yeung, 96]: S. P. M. Choi and S. Y. Yeung, "Predictive Q-routing: A Memory-based Reinforcement Learning Approach to Adaptive Traffic Control", Advances in Neural Information Processing Systems, Vol. 8, MIT Press, 1996 pp. 945 - 951.

[Chu et al, 02]: C. H. Chu, J. Gu, X. D. Hou, and Q. Gu, "A Heuristic Ant Algorithm for Solving QoS Multicast Routing Problem", 2002 IEEE Congress on Evolutionary Computation, Honolulu, Hawaii, May 12-17, 2002, pp. 1630 - 1635

[Cisco-01]: Cisco, "Internetworking Technology Handbook", 2002, Online Edition, Retrieved from: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ , Cisco, Internetworking technologies handbook, Chapter 46, OSPF

[Cisco-02]: Cisco, "Internetworking Technology Handbook", 2002, Online Edition, Retrieved from: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ , Cisco, Internetworking technologies handbook, Chapter 47, RIP

[Coiorni & Dorigo, 92] A. Coiorni and M. Dorigo "Distributed Optimization by Ant Colonies", Proceedings of First European Conference on Artificial Life, 1992, pp. 134 - 142

[Cormen et al, 02]: T. Cormen, C.Leiserson and R Rivest., 2001, "Introduction to Algorithms", Second Edition, MIT Press, ISBN: 0262032937

[Di Caro & Dorigo, 97]: G. Di Caro and M. Dorigo, "AntNet: A Mobile Agents Approach to Adaptive Routing", Universit´e Libre de Bruxelles, IRIDIA, Technical Report 97-12, 1997

[Di Caro & Dorigo, 98-1]: G. Di Caro and M. Dorigo, "AntNet: Distributed Stigmergetic Control for Communications Networks", Journal of Artificial Intelligence Research (JAIR), Vol. 9, 1998, pp. 317 - 365

[Di Caro & Dorigo, 98-2]: G. Di Caro, and M. Dorigo, "An Adaptive Multi-agent Routing Algorithm Inspired by Ants Behaviour", PART98—Fifth Annual Australasian Conference on Parallel and Real-Time Systems, Springer-Verlag, 1998, pp. 261 - 272

[Di Caro & Dorigo, 98-3]: G. Di Caro, and M. Dorigo, "Mobile Agents for Adaptive Routing", Proceedings of the 31st International Conference on System Sciences (HICSS-31), Los Alamitos, CA, Vol. 7, 1998, pp. 74 - 83

[Di Caro & Dorigo, 98-4]: G. Di Caro and M. Dorigo, "Two Ant Colony Algorithms for Best-effort Routing in Datagram Networks", Proc. 10th International Conference on Parallel and Distributed Computing and Systems, Las Vegas, Nevada, October 28-31, 1998

[Di Caro, 04]: G. Di Caro "Ant Colony Optimization and its application to adaptive routing in telecommunication networks" PhD thesis in Applied Sciences, Polytechnic School, Université Libre de Bruxelles, Brussels, Belgium, 2004

[Di Fatta et al, 00]: G. Di Fatta, S. Gaglio, G. Lo Re and M. Ortolani, "Adaptive Routing in Active Networks", OpenArch 2000 IEEE Third Conference on Open Architecture and Network Programming, Tel Aviv, March 2000

[Dijkstra, 59]: E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", Numerische mathematik, 1959, Vol. 1, pp. 267 - 271

[Dixon et al, 95]: M. W. Dixon, G. R. Cole and M. I. Bellgard. "A Neural Network Shortest Path Algorithm for Routing in Packet-switched Communication Networks", ICC'95, Seattle, USA, June 18-22, 1995, pp. 1602 - 1606

[Dorigo & Di Caro, 99]: M. Dorigo and G. Di Caro, "The Ant Colony Optimization Meta-heuristic", New ideas in optimization, McGraw-Hill, London, 1999, pp. 11 - 32

[Dorigo & Gambardella, 97]: M. Dorigo and L. M. Gambardella, "Ant Colonies for the Traveling Salesman Problem", BioSystems, Vol. 43, 1997, pp. 73-81

[Dorigo & Gambardella, 99]: M. Dorigo and M. L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem", IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, 1999, pp. 53 - 66

[Dorigo & Stutzle, 04]: Marco Dorigo and Thomas Stutzle, "Ant Colony Optimization", Bradford Book, July 2004, ISBN: 0262042193

[Dorigo et al, 02-1]: M. Dorigo, L. M. Gambardella, M. Middendorf and T. Stutzle, "Guest Editorial: Special Section on Ant Colony Optimization", Evolutionary Computation, IEEE Transactions on , Vol. 6, No. 4 , Aug 2002, pp. 317 - 319

[Dorigo et al, 02-2] M. Dorigo, G. Di Caro, and M. Sampels, "Ant Algorithms", Proceedings of ANTS 2002, Third International Workshop on Ant Algorithms, Brussels, Belgium, September 12--14, 2002", Lecture Notes in Computer Science Vol. 2463, Springer-Verlag, 2002.

[Dorigo et al, 96]: Marco Dorigo and Vittorio Maniezzo. "Ant system: Optimization by a Colony of Cooperating Agents" IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol. 26, No. 1, 1996, pp. 29 - 41

[Dorigo et al, 99]: M. Dorigo, G. Di Caro and L. M. Gambardella, "Ant Algorithms for Discrete Optimization", Artificial Life, Vol. 5, No. 2, 1999, pp. 137 - 172

[Feng, 81]: T. Y. Feng, "A survey of interconnection networks", IEEE Computation Magazine, Vol. C-30, No. 12, 1981, pp. 12 - 27

[Fishman 96]: G. S. Fishman, "Monte Carlo", Springer Verlag, 2nd edition, April 25, 1996, ISBN: 038794527X

[Franks, 89]: N. Franks, "Army ants: A Collective Intelligence", American Scientist, Vol 77, Mar-Apr, 1989, pp. 139 - 145

[Gao et al, 01]: R. Gao, Z. Ghassemlooy, G. Swift and P. Ball, "Simulation of All-Optical Time Division Multiplexed Router", SPIE Photonics West, USA, 2001

[Gao, 03]: R. Gao, "All Optical Packet Network Routing in High Speed Communication Systems", PhD Thesis, Sheffield Hallam University, 2003

[Geist et al, 94]: A. Geist, A Bequelin, J Dongarra, W Jiang, and V Sundaram., "PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing", MIT Press, 1994, ISBN: 0262571080

[Goldberg, 89]: D. E. Goldberg, "Genetic algorithms in search, optimisation and machine learning", Addison-Wesley, 1989, ISBN: 0201157675

[Golomb, 82]: S. W. Golomb, "Shift Register Sequences", Agean Park Press, 1982, ISBN: 0894120484

[Grammatikakis et al, 98]: M. D. Grammatikakis, D. F. Hsu and M. Kraetzl, "Packet Routing in Fixed-Connection Networks: A Survey", Journal of Parallel and Distributed Computing Vol. 54, 1998, pp. 77 - 132

[Gunes et al, 02]: M. Gunes, U. Sorges, and I. Bouazizi, "ARA - The Ant-Colony Based Routing Algorithm for MANETs", 2002 International Conference on Parallel Processing Workshops (ICPPW'02), Vancouver, B.C., Canada, Aug. 18 - 21, 2002, pp. 79 - 85

[Guoying et al, 00]: L. Guoying, L. Zemin, and Z. Zheng, "Multicast Routing Based on ant Algorithm for Delay-bounded and Load-balancing Traffic",25th Annual IEEE Conference on Local Computer Networks (LCN'00), November 08 - 10, 2000, pp. 362 - 368

[Hedrick, 88]: C. L. Hedrick, 1988, "Routing Internet Protocol (RIP)", RFC 1058, IETF

[Hedrick, 91]: C. L. Hedrick Rutgers, "An Introduction to IGRP", CISCO White Papers, Document ID: 26825, 1991

[Heusse et al, 98]: M. Heusse, S. Gu´erin, D. Snyers, and P. Kuntz, "Adaptive Agent-Driven Routing and Load Balancing in Communication Networks", Advances in Complex Systems, Vol. 1, 1998, pp. 237 - 254

[Hluchyj & Karol, 88]: M. G. Hluchyj and M.K. Karol, "ShuffleNet: An Application of Generalized Perfect Shuffles to Multihop Lightwave Network'' IEEE INFOCOM '88, 1988, pp. 379 - 390

[Hunter et al, 98]: D. K. Hunter, M. C. Chia and I. Andonovic, "Buffering in Optical Packet Switches", Journal of Lightwave Technology, Vol. 16, No. 12, 1998, pp. 2081 - 2094

[Iness et al, 95]: J. Iness, S. Banerjee, and B. Mukherjee, "GEMNet: A Genaralized, Shuffle Exchange-Based, Regular, Scalable, Modular, Multihop, WDM Lightwave Network," IEEE/ACM Trans. on Networking, Vol. 3, No. 4, Aug. 1995, pp. 470 - 476

[Jaekel et al, 98]: A. Jaekel, S. Bandyopadhyay and A. Sengupta, "A Flexible Architecture for Multihop Optical Networks" IEEE International Conference on Computer Communication and Networks, ICCCN98, 1998, pp. 472 - 478

[Kaelbling et al, 96]: L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey", Journal of Artificial Intelligence Research, Vol. 4, 1996, pp. 237 - 285

[Karol & Shaikh, 88]: M. J. Karol and S. Shaikh, "A Simple Adaptive Routing Scheme for ShuffleNet Multihop Lightwave Networks," GLOBECOM '88, Conf. Rec., Nov. 1988, pp.1640 - 1647

[Kassabalidis et al, 01]: I. Kassabalidis, M.A.El-Sharkawi, R.J.Marks II, P. Arabshahi and A.A.Gray, "Swarm Intelligence for Routing in Communication Networks", IEEE Globecom 2001, San Antonio, Texas, Nov. 25-29, 2001, pp. 3613 - 3617

[Katseff , 88]: H. P. Katseff, "Incomplete Hypercubes," IEEE Trans. on Computer, Vol. 37, No. 5, May 1988, pp. 604–608

[Kumar & Miikkulaine, 02]: S. Kumar and R. Miikkulainen,"Confidence-Based Q-routing: An On-Line Adaptive Network Routing Algorithm", Smart Engineering Systems: Neural Networks, Fuzzy Logic, Data Mining, and Evolutionary Programming: Vol. 8, 2002

[Kumar & Miikkulainen, 97]: S. Kumar and R. Miikkulainen., "Dual Reinforcement Q-routing: An Online Adaptive Routing Algorithm", Artificial Neural Networks in Engineering, Vol. 7, 1997, pp. 231 - 238

[Kunkle, 01]: D. R. Kunkle, "Self Organizing Computation and Information Systems: Ant Systems and Algorithms", Introduction to Artificial Intelligence, November 2001

[Larra Naga et al, 99]: P. Larra Naga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators", Articial Intelligence Review, Vol. 13, 1999, pp. 129 - 170

[Li et al, 00]: L. Li, Z. Liu, and Z. Zhou, "A New Dynamic Distributed Routing Algorithm on Telecommunication Networks", International Conference on Communication Technologies (ICCT2000) in conjunction with 16th IFIP World Computer Congress (WCC2000), in Beijing, Aug. 21-24, 2000

[Liang and Zincir-Heywood, 02]: S. Liang, A. N. Zincir-Heywood, "The Effect of Routing under Local Information using a Social Insect Metaphor", IEEE Congress on Evolutionary Computation, May 2002, pp. 1438 - 1443

[Liang et al, 02]: S. Liang, A. N. Zincir-Heywood and M. I. Heywood, "Intelligent packets for dynamic network routing using distributed genetic algorithm", Genetic and evolutionary computation conference (GECCO 2002), July 9-13, 2002, pp. 88 - 96

[Littman & Boyan, 92]: M. Littman and J. Boyan "A Distributed Reinforcement Learning Scheme for Network Routing", Technical Report Carnegie Mellon University, 1992

[Malkin, 98]: G. Malkin., 1998, "RIP Version 2", RFC 2453, IETF

[Matsou & Mori, 01]: H. Matsuo and K. Mori, "Accelerated Ants Routing in Dynamic Networks", Proc. Intl. Conf. On Software Engineering, Artificial Intelligence, Networking and Parallel /Distributed Computing, Aug. 2001, pp.333 - 339

[Maxemchuck, 87]: N. F. Maxemchuck, "Routing in the Manhattan Street Network," IEEE Trans. Commun., Vol. COM-35, May 1987, pp. 503 - 512

[Maxemchuck, 89]: N. F. Maxemchuk, "Comparison of Deflection and store-and-forward techniques in the MSN and Shuffle Exchange networks," Proc. IEEE INFOCOM '89, April 1989

[Maxemchuck, 93]: N. F. Maxemchuck, "Regular mesh topologies in local and metropolitan area networks," AT&T Tech. J., vol. 64, pp. 1659–1686, Sept. 1985. 796–798, Dec. 1993

[Michalareas & Sacks, 01]: T.Michalareas and L.Sacks, "Link-State & Ant-like Algorithm Behaviour for Single-Constrained Routing", IEEE Workshop on High Performance Switching and Routing, HPSR 2001, May 2001

[Mitchell, 97]: M. Mitchell, "An Introduction to Genetic Algorithms", MIT Press, Third Printing , 1997, Chp. 1, ISBN: 0262631857

[Moy, 98]: J. T. Moy., 1998, "OSPF Version 2", RFC 2328, IETF

[Mukherjee, 92-1]: B. Mukherjee "WDM-Based Local Lightwave Networks Part I: Single-Hop Systems", IEEE Network, May 1992, pp. 12 - 27

[Mukherjee, 92-2]: B. Mukherjee, "WDM-based local lightwave networks - Part II Multihop systems," IEEE Network, vol. 6, July 1992, pp. 20 - 32

[Osman & Kelly, 96]: I. H. Osman, J. P. Kelly, "Meta-Heuristics: Theory & Applications", Kluwer Academic Publishers, 1996, ISBN: 0792397002

[Panchapakesan & Sengupta, 95]: G. Panchapakesan and A. Sengupta, "On Multihop Optical Network Topology Using Kautz Digraphs," Proc. IEEE INFOCOM '95, Boston, April 1995, pp. 675 - 682

[Qiao & Yoo, 99]: C. Qiao and M. Yoo, "Optical Burst Switching (OBS) - A New Paradigm for an Optical Internet", Journal of High Speed Networks, Vol. 8, No. 1, 1999, pp. 69-84.

[Ramaswami & Sivarajan, 98]: R. Ramaswami and K. N. Sivarajan, "Optical Networks: A Practical Perspective", Chp.s 1,9,14, 1998, ISBN: 1558604456

[Rekhter, 95]: Y. Rekhter., "Border Gateway Protocol 4 (BGP-4)", RFC 1771, IETF, 1995

[Roli & Milano, 02]: A. Roli and M. Milano, "MAGMA: A Multiagent Architecture for Metaheuristics", technical report in University degli Studi di Bologna,August 28, 2002

[Rosen, 82]: E. C. Rosen, "Exterior Gateway Protocol (EGP)", RFC 827, 1982

[Ross, 02]: S. M. Ross, "Simulation", Academic Press Elsevier Science, Third Eddition, 2002, ISBN: 0125980531

[Russell & Norvig, 95]: S. J. Russell and P. Norvig, "Artificial Intelligence: Modern Approach", Prentice Hall, 1st edition, January 15, 1995, ISBN: 0131038052, Chp20, pp. 598 - 624

[Saad & Schultz, 88]: Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," IEEE Trans. Comput., Bol. C-37, July 1988, pp. 867–872

[Schoonderwoerd et al, 96]: R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, "Ant-based Load Balancing in Telecommunications Networks", Adaptive Behavior, Vol. 5 No. 2, 1996, pp. 169 - 207

[Schoonderwoerd et al, 97]: R. Schoonderwoerd, O.E. Holland, J.L. Bruten, "Ant-like Agents for Load Balancing in Telecommunications Networks", Proc. 1 st ACM International Conference on Autonomous Agents, February 5-8, 1997, Marina del Rey, CA, USA, pp. 209 - 216

[Schoonderwoerd, 96]: R. Schoonderwoerd, "Collective Intelligence for Network Control", M.S. Thesis, Delft University of Technology, Faculty of Technical Informatics, May 1996

[Shankar et al, 95]: A. Udaya Shankar, Cengiz Alaettinoglu, Klaudia Dussa-Zieger and Ibrahim Matta, "Transient and Steady-State Performance of Routing Protocols: Distance-Vector versus Link-State", Journal of Internetworking: Research and Experience, Vol. 6, 1995, pp. 59 - 87

[Sim & Sun, 02]: K. M. Sim and W. H. Sun, "Multiple Ant-colony Optimization for Network Routing", 1st Int. Symp. Cyberworld, Tokyo, Japan, November 2002, pp. 277–281

[Sim & Sun, 03]: K. M. Sim and W. H. Sun, "Ant Colony Optimization for Routing and Load-balancing: Survey and New Directions", Systems, Part A, IEEE Transnactions on, Man and Cybernetics, Vol. 33, No.5, Sep 2003, pp. 560 - 572

[Sivarajan & Ramaswami, 91]: K. Sivarajan and R. Ramaswami, "Multihop Lightwave Networks based on De Bruijn Graphs," IEEE INFOCOM '91, Bal Harbor, FL, April 1991, pp. 1001 - 1011

[Sivarajan & Ramaswami, 94]: K. Sivarajan and R. Ramaswami, "Multihop Lightwave Network Based on de Bruin Graph" IEEE/ACM Trans. on Networking, vol. 2, no. 1, Feb. 1994

[Sportack, 99]: M. A. Sportack, "IP Routing Fundamentals", Indianapolis, Cisco Press, 1999, 157870071X

[Steenstrup , 95]: M. E. Steenstrup, "Routing in Communications Networks", Prentice-Hall, 1st edition, 2995, ISBN: 0130107522

[Subramanian et al, 97]: D. Subramanian, P. Druschel and J. Chen, "Ants and Reinforcement Learning: A Case Study in Routing in Dynamic Networks", IJCAI-97, Palo Alto, Morgan Kauffman, 1997, pp. 832 - 838

[Sutton & Barto, 98]: R. S. Sutton amd A. G. Barto, "Reinforcement Learning: An Introduction" MIT Press, 1998, Chp.s 1-5, ISBN: 0262193981

[Tanenbaum, 03]: A. S. Tanenbaum, "Computer Networks", Fourth Edition, Prentice-hall, 2003, chp. 1,5, ISBN: 0130661023

[Tekiner et al, 02]: F. Tekiner, Z. Ghaseemlooy, M. Thompson, S. Alkhayatt, "Prioritized Shufflenet Routing in TOAD based 2X2 OTDM Router", Senacitel, 2002

[Tekiner et al, 04-1]: F. Tekiner, Z. Ghassemlooy, S. Al-khayatt, Antnet Routing Algorithm-Improved Version, in: Proceedings of CSNDSP04, Newcastle, UK, pp. 416-419, 22-22 July 2004.

[Tekiner et al, 04-2]:F. Tekiner, Z. Ghassemlooy, S. Al-khayatt, Investigation of antnet routing algorithm by employing multiple ant colonies for packet switched networks to overcome the stagnation problem", in: Proceedings of LCS04, London13-15 Sep 2004, pp. 185 - 188.

[Tekiner et al, 04-3]:F. Tekiner, Z. Ghassemlooy, S. Al-khayatt, Improved antnet routing algorithm with link probability evaporation over the given time window, in Proceedings of SOFTCOM04, Split/Venice, 11-13 Oct 2004, pp. 502 - 505

[Tekiner et al, 04-4]: F. Tekiner, Z. Ghassemlooy and T. R Srikanth, "Comparison of the Q-routing and Shortest Path Routing Algorithms" PGNET04, 28 - 29 June 2004, Liverpool, UK, pp. 428 - 432

[Varela & Sinclair, 99]: G. N. Varela, M. C. Sinclair, "Ant Colony Optimisation for Virtual-wavelength-path Routing and Wavelength Allocation", Proc. Congress on Evolutionary Computation (CEC'99), Washington DC, USA 1999, pp. 1809 - 1816

[Weyns et al, 04]: D Weyns, K Schelfthout, T Holvoet and O Glorieux., 2004, "Role Based Model for Adaptive Agents", Proceedings of the AISB 2004, Fourth Symposium on Adaptive Agents and Multi-agent Systems, 2004, pp. 75 - 86

[White & Pagurek, 98]: T. White and B. Pagurek, "Towards Multi-Swarm Problem Solving in Networks", 3rd International Conference on Multi-Agent Systems (ICMAS '98), July 1998, pp. 333 - 340

[White et al, 98-1]: T. White , B. Pagurek and F. Oppacher, "Connection Management using Adaptive Mobile Agents", International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98), July, 1998, pp. 802 - 809

[White et al, 98-2]: T. White, B. Pagurek and F. Oppacher, "ASGA: Improving the Ant System by Integration with Genetic Algorithms" 3rd Conference on Genetic Programming (GP/SGA'98), July 1998, pp.610 - 617

[Yang et al, 02]: Y. Yang, N. Zincir-Heywood, M. I. Heywood and S. Srinivas "Agent Based Routing Algorithms on LAN", Canadian Conference IEEE, 2002

[Yao et al, 01]: S. Yao, S. J. B. Yoo, B. Mukherjee, "A Comparison Study Between Slotted and Unslotted All-Optical Packet-switched Network with Priority-based Routing", Optical Fiber Communication Conference and Exhibit, OFC 2001, 2001, pp. TuK2-T1-3