# Reinforcement Learning for Routing in Communication Networks

Walter H. Andrag

Thesis presented in partial fulfilment
of the requirements for the degree of
Master of Science
at the University of Stellenbosch

Supervisor: Prof Christian W. Omlin

April 2003

Stellenbosch University http://scholar.sun.ac.za

# Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Signature: Date:

Stellenbosch University http://scholar.sun.ac.za

# Abstract

Routing policies for packet-switched communication networks must be able to adapt to changing traffic patterns and topologies. We study the feasibility of implementing an adaptive routing policy using the Q-Learning algorithm which learns sequences of actions from delayed rewards. The Q-Routing algorithm adapts a network's routing policy based on local information alone and converges toward an optimal solution. We demonstrate that Q-Routing is a viable alternative to other adaptive routing methods such as Bellman-Ford. We also study variations of Q-Routing designed to better explore possible routes and to take into consideration limited buffer size and optimize multiple objectives.

Stellenbosch University http://scholar.sun.ac.za

# Opsomming

Die roetering in kommunikasienetwerke moet kan aanpas by veranderings in netwerktopologie en verkeersverspreidings. Ons bestudeer die bruikbaarheid van 'n aanpasbare roeteringsalgoritme gebaseer op die "Q-Learning"-algoritme wat dit moontlik maak om 'n reeks besluite te kan neem gebaseer op vertraagde vergoedings. Die roeteringsalgoritme gebruik slegs nabygelëe inligting om roeteringsbesluite te maak en konvergeer na 'n optimale oplossing. Ons demonstreer dat die roeteringsalgoritme 'n goeie alternatief vir aanpasbare roetering is, aangesien dit in baie opsigte beter vaar as die Bellman-Ford algoritme. Ons bestudeer ook variasies van die roeteringsalgoritme wat beter paaie kan ontdek, minder geheue gebruik by netwerkelemente, en wat meer as een doelfunksie kan optimeer.

Stellenbosch University http://scholar.sun.ac.za

# Acknowledgements

I would like to sincerely thank my supervisor, Prof. C. W. Omlin, for all the inspiration, assistance and funding he provided.

This work was also made possible by funding from the South African National Research Foundation, Telkom-Siemens Centre of Excellence for ATM and Broadband Networks and their Applications and the Harry Crossley Scholarship Fund.

Stellenbosch University http://scholar.sun.ac.za

# Contents

1 **Introduction** **1**

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.3 Premises . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.4 Hypotheses . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.5 Technical Objectives . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6 Methodology . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.7 Achievements . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.8 Thesis Organization . . . . . . . . . . . . . . . . . . . . . . . . 5

2 **Routing in Communication Networks** **6**

2.1 The Routing Problem . . . . . . . . . . . . . . . . . . . . . . . . 6

2.1.1 Performance Criterion . . . . . . . . . . . . . . . . . . . . 8

2.1.2 Decision Time . . . . . . . . . . . . . . . . . . . . . . . . 8

2.1.3 Decision Place . . . . . . . . . . . . . . . . . . . . . . . . 9

2.1.4 Network Information Source . . . . . . . . . . . . . . . . . 9

2.1.5 Routing Information Update Timing . . . . . . . . . . . . 9

2.2 Conventional Routing Strategies . . . . . . . . . . . . . . . . . . 10

Stellenbosch University http://scholar.sun.ac.za

2.2.1 Flooding . . . . . 10
2.2.2 Random Routing . . . . . 10
2.2.3 Fixed Routing . . . . . 11
2.2.4 Adaptive Routing . . . . . 11
2.2.5 Link-State Routing . . . . . 12
2.2.6 Distance-Vector Routing . . . . . 12
2.3 Mobile Agents . . . . . 13
2.3.1 Active Networks . . . . . 13
2.3.2 Social Insect Metaphors . . . . . 14
2.4 Summary . . . . . 14

**3 Reinforcement Learning** **16**
3.1 Value Functions . . . . . 17
3.2 Temporal-Difference Learning . . . . . 19
3.3 Q-Learning . . . . . 20
3.4 TD($\lambda$) Learning . . . . . 22
3.5 Q($\lambda$) Learning . . . . . 24
3.6 Convergence Properties of Q-Learning . . . . . 25
3.7 Exploration vs Exploitation . . . . . 27
3.8 Summary . . . . . 29

**4 Q-Learning for Traffic Routing** **30**
4.1 Optimization of Packet Delivery Time . . . . . 30
4.1.1 Q-Routing . . . . . 30
4.1.2 DRQ-Routing . . . . . 35

Stellenbosch University http://scholar.sun.ac.za

4.1.3 CQ-Routing . . . . . 39
4.1.4 CDRQ-Routing . . . . . 42
4.1.5 Probabilistic CDRQ-Routing . . . . . 44
4.2 Finite Buffer Size . . . . . 48
4.3 Optimization of Multiple Objectives . . . . . 50
4.4 Summary . . . . . 59

**5 Conclusion** **61**

5.1 Conclusion . . . . . 61
5.2 Future Work . . . . . 62
5.2.1 Realistic Simulations . . . . . 62
5.2.2 Improved Routing . . . . . 63

Stellenbosch University http://scholar.sun.ac.za

# List of Tables

2.1 Design elements of a routing strategy . . . . . . . . . . . . . . . . . . . . 8

4.1 The parameters used in the simulations . . . . . . . . . . . . . . . . . . . 49

Stellenbosch University http://scholar.sun.ac.za

# List of Figures

3.1 The agent-environment interaction. . . . . 16

3.2 Estimating $V^{\pi}$ with TD(0). . . . . 20

3.3 Estimating $Q^{*}$ with Q-Learning. . . . . 22

3.4 Estimating $V^{\pi}$ with TD($\lambda$). . . . . 23

3.5 Watkins's Q($\lambda$) algorithm. . . . . 24

4.1 The British Synchronous Digital Hierarchy (SDH) network topology. . . 32

4.2 Average packet delivery times for network load 1.2 for the SDH network topology. . . . . 34

4.3 Average packet delivery times for network load 2.2 for the SDH network topology. . . . . 34

4.4 Average packet delivery times for network load 3.2 for the SDH network topology. . . . . 35

4.5 Average packet delivery times of Bellman-Ford for high network load for the SDH network topology. Error bars show standard deviations. . . . 36

4.6 Average packet delivery times of Q-Routing for high network load for the SDH network topology. Error bars show standard deviations. . . . 36

4.7 Comparing the average packet delivery times of Q-Routing and DRQ-Routing for network load 2.0 for the SDH network topology. . . . . . 37

4.8 Comparing the average packet delivery times of Q-Routing and DRQ-Routing for network load 3.0 for the SDH network topology. . . . . . 38

Stellenbosch University http://scholar.sun.ac.za

4.9 Comparing the average packet delivery times of Q-Routing and DRQ-Routing for network load 4.0 for the SDH network topology. . . . . . . 38

4.10 Comparing the average packet delivery times of Q-Routing and CQ-Routing for network load 2.0 for the SDH network topology. . . . . . . 40

4.11 Comparing the average packet delivery times of Q-Routing and CQ-Routing for network load 3.0 for the SDH network topology. . . . . . . 41

4.12 Comparing the average packet delivery times of Q-Routing and CQ-Routing for network load 4.0 for the SDH network topology. . . . . . . 41

4.13 Comparing the average packet delivery times of Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing for network load 2.0 for the SDH network topology. . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

4.14 Comparing the average packet delivery times of Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing for network load 3.0 for the SDH network topology. . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

4.15 Comparing the average packet delivery times of Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing for network load 4.0 for the SDH network topology. . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

4.16 The variance function of Equation 36 for $\beta$ of 0.2, 0.4, 0.6 and 0.8. . . . 45

4.17 The Average Packet Delivery Time for the SDH network for network load 1.5; $\beta$ of 0.2, 0.4, 0.6 and 0.8. . . . . . . . . . . . . . . . . . . . . . . 46

4.18 The Average Packet Delivery Time for the SDH network for network load 3.0; $\beta$ of 0.2, 0.4, 0.6 and 0.8. . . . . . . . . . . . . . . . . . . . . . . 47

4.19 The Average Packet Delivery Time for the SDH network for network load 4.5; $\beta$ of 0.2, 0.4, 0.6 and 0.8. . . . . . . . . . . . . . . . . . . . . . . 47

4.20 The Congestion Risk of Equation 38 for $\theta$ of 3, 6 and 15. . . . . . . . . 49

4.21 The 13 node network topology used for the finite buffer simulation. . . 50

4.22 Average packet delivery time for low load. . . . . . . . . . . . . . . . . . 51

4.23 Number of packets dropped for low load. . . . . . . . . . . . . . . . . . . 51

Stellenbosch University http://scholar.sun.ac.za

4.24 Average packet delivery time for medium load. . . . . . . . . . . . . . . . 52
4.25 Number of packets dropped for medium load. . . . . . . . . . . . . . . . 52
4.26 Average packet delivery time for high load. . . . . . . . . . . . . . . . . . 53
4.27 Number of packets dropped for high load. . . . . . . . . . . . . . . . . . . 53
4.28 The network topology for the 36 node grid. . . . . . . . . . . . . . . . . . 54
4.29 The average packet delivery time for single versus multiple objective optimization for the 36 node grid for differing $\alpha$. . . . . . . . . . . . . 55
4.30 Details of the steady state behaviour of Figure 4.29. . . . . . . . . . . 56
4.31 The average cost for single versus multiple objective optimization for the 36 node grid for differing $\alpha$. . . . . . . . . . . . . . . . . . . . . . . . . 56
4.32 The average packet delivery time for single versus multiple objective optimization for the BT SDH network for differing $\alpha$. . . . . . . . . . 57
4.33 Details of the steady state behaviour of Figure 4.32. . . . . . . . . . . 57
4.34 The average cost for single versus multiple objective optimization for the BT SDH network for differing $\alpha$. . . . . . . . . . . . . . . . . . . . . . 58
4.35 The average saving of multiple objective optimization of cost and delivery time for the BT SDH network versus $\alpha$. . . . . . . . . . . . . . . . . . 58

Stellenbosch University http://scholar.sun.ac.za

# Chapter 1

# Introduction

## 1.1 Motivation

Modern communication networks must cope with ever increasing demands on network resources. The range of services offered leads to both regular and less predictable traffic patterns. Adaptive routing is able to respond to changing traffic patterns and topology, thus providing efficient use of network resources. In networks characterized by a constantly changing topology, adaptive routing is essential. Adaptation may be necessary in traditional networks due to failures of links or nodes; in mobile ad-hoc networks, mobile routers are able to move randomly, thus constantly and unpredictably changing the network topology.

In order to adapt routing to changing network conditions, a centralized routing strategy needs information about the status of all nodes and links in the network. However, this information transmission overhead consumes valuable network resources. This highlights the need to make distributed routing decisions based on locally available information only.

## 1.2 Problem Statement

A packet-switched communication network can be modeled as a set of nodes and interconnecting links. Data is exchanged over these communication links as a sequence of packets. In general, nodes are not fully connected; thus, the packets must pass through

Stellenbosch University http://scholar.sun.ac.za



intermediate nodes. The *route* is the sequence of nodes along which a packet travels to its final destination. In most networks, there may be more than one route between pairs of nodes. The routing problem consists of finding the *optimal* route between source and destination nodes, where the optimal route is the one that delivers packets to their final destination in the shortest time possible.

## 1.3 Premises

The premises of the packet routing domain which we believe make adaptive routing indispensable are as follows:

1. A network is a highly dynamic environment in which traffic patterns may be unpredictable and links or nodes may fail.
2. A central routing mechanism which has global information about the state of the network is generally not feasible because of the overhead involved.
3. Thus, we need a good routing policy which
   (a) uses only local information and
   (b) minimizes average packet delivery time.

## 1.4 Hypotheses

Machine learning covers a broad field of methods concerned with the ability of programs to learn from experience, thereby improving their performance. We wish to test the following hypotheses in this thesis:

1. Machine learning is a viable alternative to static routing because
   (a) it can adapt to changing environments, i.e. changes in traffic patterns or network topology;
   (b) we can learn from experience, i.e. past traffic and routing patterns.

Stellenbosch University http://scholar.sun.ac.za



2. Reinforcement learning is a field in machine learning concerned with programs taking optimal action sequences so as to achieve a goal. Reinforcement learning is well-suited for adaptive routing because

   (a) it is goal-oriented, i.e. actions are to be learned with a desired outcome. The goal of routing is to deliver packets with minimum delay.

   (b) Reinforcement learning allows to acquire a policy, i.e. a sequence of actions that lead to a desired outcome. Actions in routing are propagation of packets to a neighbouring node, and the desired outcome is the delivery of packets to their intended final destination.

   (c) Reinforcement learning algorithms are able to learn policies from delayed rewards. We only know whether a good route was chosen for a packet once it has reached its destination.

   (d) Reinforcement learning algorithms can adapt to changes in the environment. As traffic patterns change, or nodes or links fail, a routing policy will have to adapt.

## 1.5 Technical Objectives

The objectives we set out to achieve in our investigation are as follows:

1. Implement a distributed adaptive packet routing algorithm which minimizes the average packet delivery time, while using only locally available information.

2. Compare the performance of the adaptive routing policy to standard routing algorithms.

3. Improve the adaptive routing mechanism to discover new routes.

4. Extend and test the algorithms under more realistic scenarios including nodes with finite buffer size and optimization of multiple objectives.

## 1.6 Methodology

We will use the following methodology for achieving our technical objectives:

Stellenbosch University http://scholar.sun.ac.za



1. Q-Learning is a reinforcement learning algorithm that is able to learn an optimal sequence of actions in an environment which maximizes rewards received from the environment. Q-Routing is an implementation of Q-Learning, which is able to distributively route packets in a network. Each node is able to make a routing decision using only locally available information. The reward received is the packet delivery time; thus, the goal is to minimize the average delivery time of all packets.

2. We will empirically evaluate the routing algorithms by simulation and compare their performance under different traffic loads and network topologies. We will use the average packet delivery time and the average number of dropped packets as the performance measure.

3. We will investigate possible performance improvements to Q-Routing designed to increase the exploration ability of the algorithm, which enables the discovery of new routes in the network. This improvement is made possible by adding a probabilistic component to routing decisions.

4. We will consider more realistic network scenarios: networks with finite buffers and networks where there are multiple objectives to be optimized:

   (a) Previous work explored the performance of Q-Routing in networks with infinite packet buffers. We will examine the more realistic case of finite buffers. Congestion control is achieved by avoiding nodes with a high level of congestion.

   (b) We will also examine the performance of a new routing algorithm, able to optimize multiple possibly conflicting objectives, e.g. packet delivery time versus cost. We examine the implications of this trade-off.

## 1.7 Achievements

We have learned the following from our investigation:

1. Q-Routing is able to route packets minimizing the average delay while only using local information.

2. Q-Routing compares well with standard routing algorithms. In particular, it converges to a more stable routing policy than our version of the distributed

Stellenbosch University http://scholar.sun.ac.za



Bellman-Ford algorithm.

3. We demonstrated the efficiency of new path discovery of a proposed algorithm based on probabilistic exploration.

4. Two extended algorithms were also shown to perform well in more realistic network scenarios:

   (a) In networks with limited buffers, the algorithm is able to perform congestion control, dropping fewer packets at overloaded nodes.

   (b) An improved routing algorithm is also able to optimize multiple objectives. However, competing objectives may establish a trade-off.

## 1.8 Thesis Organization

In Chapter 2, we discuss the routing problem in more detail and look at some of the approaches used to solve it. Chapter 3 discusses the field of reinforcement learning, presenting techniques of solving reinforcement learning problems. In Chapter 4, we present the simulation results of the comparison between different routing algorithms by evaluating performance under various scenarios. Conclusions and directions of future research are presented in Chapter 5.

Stellenbosch University http://scholar.sun.ac.za

# Chapter 2

# Routing in Communication Networks

In this chapter, we examine the routing problem and investigate different approaches that have been proposed for solving it. We define the routing problem, and discuss the general requirements of routing algorithms. Network routing is very complex; thus, we discuss some of the characteristics that differentiate between different routing algorithms.

## 2.1 The Routing Problem

We consider a communication network [27; 15] as a undirected weighted graph $G = (N, L)$ with a set of nodes $N$, and a set of bidirectional links $L$, connecting the nodes. Each link has a capacity and a user-defined associated cost. We define a path as a sequence of nodes connecting a source to a destination node. There may be multiple paths between sources and destinations. The general routing problem consists of finding the optimal path between source and destination nodes satisfying some performance criterion.

We will discuss the routing problem in the context of packet switching. In a packet-switched network, data is broken up into a sequence of packets which are sent from node to node until the destination is reached. The routing decision at each node consists of deciding to which neighbouring node to send a packet.

Stellenbosch University http://scholar.sun.ac.za



A routing algorithm has the following requirements [27]:

- Correctness
- Simplicity
- Efficiency
- Robustness
- Stability
- Fairness
- Optimality

The *correctness* of a routing algorithm refers to the fact that it must route all packets to the correct destinations. *Simple* routing algorithms are also preferred, as they have less routing overhead, which in turn increase the *efficiency* of the network. All packet routing schemes have a certain amount of processing and transmission overhead, which may negatively impact the efficiency of the network. The benefits of overheads must be balanced with the decrease in efficiency caused.

Some of these requirements are in competition with each other, e.g. robustness and stability. A routing algorithm is said to be *robust* when it is able to adapt to node or link failures and changes in network load conditions. When an overload is detected in a section of the network, traffic is rerouted to less congested regions. If the routing algorithm responds too quickly, these less congested regions will in turn become congested. The routing algorithm is called *unstable* if it continually shifts the load between different sections of the network. On the other hand, if the network adapts too slowly, packets may be dropped at congested nodes.

There also exists a trade-off between *optimality* and *fairness*: if a certain performance criterion favours the exchange of packets between nearby nodes, the throughput may be increased. This may appear unfair to nodes with a high proportion of long-distance traffic.

We briefly discuss the various design elements that contribute to a routing strategy as presented in [27] (see Table 2.1).

Stellenbosch University http://scholar.sun.ac.za



| Performance criterion | Network information source |
|---|---|
| Number of hops | None |
| Cost | Local |
| Delay | Adjacent nodes |
| Throughput | Nodes along route |
| | All Nodes |
| **Decision time** | |
| Packet | **Network information update timing** |
| Session | Continuous |
| | Periodic |
| **Decision place** | Major load change |
| Each node (distributed) | Topology change |
| Central node (centralized) | |
| Originating node (source) | |

Table 2.1: Design elements of a routing strategy

### 2.1.1 Performance Criterion

A routing policy has to decide to which neighbouring node to forward a packet to based on some performance criterion. The simplest choice is to select the neighbour which is on the minimum hop path to the packet's destination. A more general approach is to assign a link cost to each link and to select the minimum cost path. The specific cost metric used determines the optimal path. If the link cost is inversely proportional to the link capacity, the least-cost path maximizes the throughput whereas it minimizes the average packet delay when the link cost is the measured link delay.

Other possible cost metrics are reliability, load and communications cost. The metric can also be a combination of several performance criteria; i.e. the optimal route over multiple objectives.

### 2.1.2 Decision Time

The decision time of routing decisions refer to two types of packet-switched networks. In a *datagram* packet switching network, each node makes a routing decision for each incoming packet. However, there is another approach, called *virtual-circuit* packet switching, where the routing decision is made only once per *session.* If a source node wants to communicate with a destination node, a virtual-circuit between source and

Stellenbosch University http://scholar.sun.ac.za



destination is established. After the connection has been set up, each node selects the neighbour based on the virtual-circuit identifier. Thus, all subsequent packets of a session will follow the same route through the network.

### 2.1.3 Decision Place

The decision place refers to where routing decisions are made. In *centralized* routing, there is a central control node which collects information from the network and computes routing tables which are distributed to all nodes. The problem with this approach is that the controlling node is a single point of failure. *Distributed* routing algorithms make routing decisions at each node; thus, they are more robust. In *source* routing algorithms, the originating node selects the route through the network.

### 2.1.4 Network Information Source

Most routing algorithms utilize some information about the network topology, traffic load or link cost. Distributed routing may utilize information available locally to the node such as the cost of each link. Nodes may also make routing decisions based on information from neighbouring nodes, or all nodes on a path. Centralized routing makes use of information from all nodes. Some algorithms do not use any network state information, e.g. flooding and random routing.

### 2.1.5 Routing Information Update Timing

If the routing strategy uses locally available information, routing updates are continuous. For all other strategies that make use of network information, routing information updates are made periodically in order to adapt to changing network conditions. The accuracy of information depends on how frequently the information is updated. Thus, with more accurate information, better routing decisions are made. However, information updates consume valuable network resources.

Stellenbosch University http://scholar.sun.ac.za



## 2.2 Conventional Routing Strategies

Network routing is a very complex problem and many different approaches to solving it have been proposed. We briefly discuss some of the routing strategies used, ranging from the simple to the more complex adaptive routing strategies.

### 2.2.1 Flooding

Flooding [27] is simple routing strategy whereby each node forwards a packet to each of its neighbours, except the node where the packet came from. Nodes do not need any information about the network topology beyond their immediate neighbours. Packets need a sequence number and the destination node embedded in their headers so that a destination node can discard duplicate packets. Forwarded packets which return to a previously visited node must also be discarded; otherwise, the number of packets in circulation will increase without bound. Another way to accomplish this is for each packet to have a hop count which is incremented at each node, and discarded when a predetermined limit is reached.

Since all possible routes between source and destination are tried, a packet is guaranteed to reach the destination if it is reachable; thus, flooding is very robust. It has been used in military networks where link or node failures may frequently occur [15]. Another property of flooding is that at least one packet will travel along the shortest route. This may be used in some networks to set up virtual-circuits. Because all nodes directly or indirectly connected to the source node are visited, flooding can be used to distribute important information (e.g. routing information) to all nodes.

The biggest disadvantage of flooding is of course the high level of network bandwidth that is wasted on duplicate packets.

### 2.2.2 Random Routing

Another simple, robust routing strategy is that of random routing [27], where each node randomly selects the node to forward a packet to, excluding the node where the packet came from. Although this strategy will in general not select the shortest path, it generates less traffic than flooding. A refinement of this technique is to select an

Stellenbosch University http://scholar.sun.ac.za



outgoing link with a probability proportional to the data rate of the link. This strategy attempts to ensure a good traffic distribution.

### 2.2.3 Fixed Routing

Fixed routing – also called static shortest path routing – computes least-cost paths for all origin-destination nodes in the network. From these fixed paths, routing tables are computed and sent to each node. As the least-cost paths are computed once, the link costs cannot be based on dynamic variables such as traffic. Instead, the network is designed based on an anticipated traffic distribution.

Fixed routing is simple and it is very effective in reliable networks with stable load. The disadvantage is that it does not react to congestion or node failures, or unforeseen traffic patterns.

### 2.2.4 Adaptive Routing

In order to increase efficiency, adaptive routing methods dynamically alter routes when node or link failures are detected or when congestion develops. For a network to adapt to these changes, it needs to collect and exchange network state information between nodes, such as delay or throughput [26]. The optimality of the new routes depends on the quality of the network information, which necessitates an increased information exchange. However, there exists a trade-off between the quality of information and the overhead: overhead consumes network resources, which may degrade the overall network performance.

A serious problem with adaptive routing is that it may become unstable if a routing policy reacts too quickly to congestion [15; 27; 14]. If the adaptive routing redirects most traffic away from the congested part of the network, congestion may develop elsewhere; thus, traffic will again shift to a different part of the network. This oscillation will continue indefinitely if not properly managed by the routing algorithm.

As it takes time for the network information to reach relevant nodes, there is never a true picture of the network state. Temporary routing loops [11; 7] can develop, where packets circulate through the network until all nodes have consistent routing tables. This looping wastes bandwidth and increases delay.

Stellenbosch University http://scholar.sun.ac.za



Although adaptive routing is complex, it is widely used as it improves the network performance, and helps in congestion control.

### 2.2.5 Link-State Routing

Link-state routing [26] is a distributed, adaptive routing algorithm where each node maintains a view of the whole network topology with a cost for each link. To update their view of the current network state, nodes regularly broadcast the link costs of outgoing links to all other nodes using flooding. Each node uses its view to calculate the shortest paths to all destinations with Dijkstra's algorithm. Each node needs storage space proportional to $O(N^2)$, where $N$ is the number of nodes in the network.

Open Shortest Path First (OSPF) is the link-state routing protocol used in the Internet [11]. Instabilities are avoided by disseminating the link cost information quickly, and by representing the link-costs by a slowly changing measure of average link utilization [26; 27]. Rapid link cost dissemination can be achieved if routing packets have higher priority than data packets. Routing loops are still possible, but since they disappear in time proportional to the diameter $D$ of the network, they are short-lived.

### 2.2.6 Distance-Vector Routing

Distance-vector routing is another distributed, adaptive routing approach based on the Bellman-Ford algorithm [10; 26]. Each node maintains a set of distances to all destinations via each of its neighbours. Thus, the storage needed at each node is proportional to $O(N \times e)$, where $e$ is the average number of neighbours of each node in the network. Each node routes an incoming packet to the neighbour with the minimum distance to the destination.

Nodes update their distance tables by exchanging *distance-vectors* with their neighbours. The distance-vector a node transmits consists of the current shortest distance from a node to each destination. Upon receiving a distance-vector, a node computes a new distance table by selecting the minimum between the current and received shortest distances. If the distance table changes, the node will again broadcast its newly computed distance-vector to all neighbours. This asynchronous update mechanism converges to the shortest distances for all connected pairs of nodes [7].

Stellenbosch University http://scholar.sun.ac.za



The original ARPANET used the distributed Bellman-Ford algorithm; however, it was replaced in 1979 by a brute-force link-state algorithm because of several drawbacks [27; 7]. It was found to react slowly to failures and link cost changes. The problem is that the distances exchanged between nodes may contain paths with loops. The looping of packets wastes bandwidth and is called the *bouncing effect*. If the network is disconnected, the algorithm does not even terminate; this is also referred to as the *counting-to-infinity* problem.

Mechanisms to overcome these problems have been proposed which use various node coordination techniques, diffusing computations and maintaining only loop-free paths [7; 11; 1; 26]. These techniques all eliminate long-lived loops, and some also eliminate short-lived loops. However, these techniques all have increased communication overhead to differing degrees.

## 2.3 Mobile Agents

As the network and its traffic are a highly dynamical system, it has been argued that mobile software agents are a good approach for adaptive routing in such a complex, inherently distributed environment [16; 6]. The use of multiple cooperating agents may facilitate a high level of availability, adaptability and fault-tolerance in modern communication networks. Mobile agents may also serve useful in design, abstracting the interactions between entities in a complex system.

### 2.3.1 Active Networks

The new approach of *active networks* enable nodes to execute custom code embedded in packets. This allows packets to route themselves and perform computations at network nodes on the route [31; 16]. In addition to routing, this approach also allows flexible incorporation of new services into a network without the need to redesign the network infrastructure [31].

The chief problems facing active networks are ensuring the *security* and *scalability* of the networks. Before executing mobile code, the node must trust the code. One way of doing this is with Proof-Carrying Code (PCC) [22]. The mobile code includes a formal proof of its properties, which the processing node can verify. The question is whether

Stellenbosch University http://scholar.sun.ac.za



the increased flexibility justifies the extra overhead of per packet execution, and how well this paradigm scales to very large networks.

### 2.3.2 Social Insect Metaphors

Ant-colony optimization is a method of solving combinatorial optimization problems inspired from the foraging behaviour of ants [6]. In nature, ants are able to find the shortest distance to a food source by laying trails of pheromones. A large collection of ants cooperate on a task by this indirect form of communication through the environment, called *stigmergy*.

Adaptive distributed route discovery is performed by artificial software ants that explore the network [25; 6]. Throughout the network, ants are launched to randomly selected destination nodes. These ants share the queues at nodes with data packets, and record the experienced delay which is used for updating the routing tables. Each ant can be thought of as performing a single Monte Carlo experiment on the actual network, and the result is the experienced delay. The system as a whole performs parallel Monte Carlo experiments with exploration biased towards more useful regions of the state space [6].

The resulting routing is very robust as it does not depend on individual ants, but rather on the collective behaviour of the entire ant colony.

## 2.4 Summary

The aim of packet-switched networks is to make more efficient use of network resources by forwarding packets between nodes on a hop-by-hop fashion. The routing decision at each node consists of deciding which neighbour to send a packet to. We discussed the simple routing strategies of flooding, random routing and fixed routing.

Adaptive routing increases the efficiency of a network by redirecting traffic away from congested areas or dynamically changing routes in networks characterized by a constantly changing topology. Adaptive routing strategies have to avoid oscillations in the network which arise if they adapt too quickly to congestion. We discussed the two classes of adaptive routing algorithms: link-state, and distance-vector algorithms.

Stellenbosch University http://scholar.sun.ac.za



Mobile software agents may prove helpful in managing the complexity of distributed, dynamic networks. We discussed the potential of active networks, where packets route themselves by executing code on a router. The emergent behaviour exhibited by ant colonies also offer valuable insight into optimization of a complex dynamical system. Promising results have already been obtained by routing based on a collection of simple ant-like software agents.

Stellenbosch University http://scholar.sun.ac.za

# Chapter 3

# Reinforcement Learning

A broad range of learning problems can be cast into the reinforcement learning framework [13; 20]. Broadly stated, reinforcement learning is the problem of learning to achieve a goal through interaction in a dynamic environment. The learning entity which is responsible for taking actions is called an *agent*. The agent continually interacts with the environment by taking actions, and receiving rewards and state information, as shown in Figure 3.1. The goal of the agent is to experiment with different action sequences in order to maximize the reward received over time.

An important aspect of reinforcement learning algorithms is that they are able to learn from *delayed rewards*. In some problems, an agent has to execute a specific sequence of actions before it receives a reward. To learn such a sequence, an agent has to overcome the problem of *temporal credit assignment*, i.e. an agent has to decide which states in the action sequence were responsible for the received reward. Reinforcement learning algorithms therefore are concerned with finding the optimal sequence of actions through

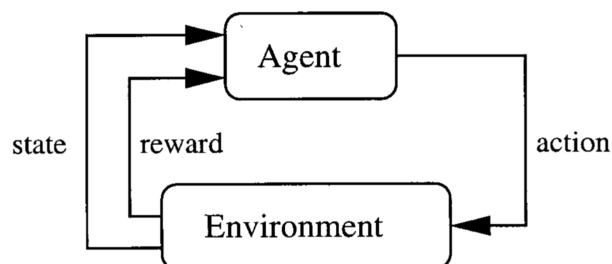


Figure 3.1: The agent-environment interaction.

Stellenbosch University http://scholar.sun.ac.za



trial-and-error interactions in an environment that maximizes the received reward over time.

Reinforcement learning algorithms differ from supervised learning algorithms in that they are not trained on input/output pairs specifying which action is the best at each state. Instead, they are guided to the goal by the rewards received. In other words, the reward received after each action fully specifies the problem to be solved. Another difference to supervised learning is that a task often has no separate training and testing phases. Instead, some tasks require continual learning throughout an agent's life.

## 3.1 Value Functions

We can formulate the reinforcement learning task an agent faces as a Markov decision process (MDP) [13]. A *finite* Markov decision process is characterized by:

- a finite set of states $S$,
- a finite set of actions $A$,
- a reward function $R : S \times A \rightarrow \Re$, and
- a state transition function $T : S \times A \times S \rightarrow \Re$, where $T(s, a, s')$ is the probability of advancing from state $s$ to $s'$ when taking action $a$.

The model is called *Markov* if the transition probabilities $T$ are independent of previous states and actions. Thus, the next state is specified probabilistically by the transition function $T$ and the current state and action alone. Note that the model is a *nondeterministic* MDP because the actions are chosen probabilistically.

At each time step $t$, an agent observes the state $s_t$ and takes action $a_t$. The environment responds by returning a reward $r_{t+1} = R(s_t, a_t)$ and the next state $s_{t+1}$ with probability $T(s_t, a_t, s_{t+1})$. This process is repeated continually until the agent achieves its goal, or indefinitely for non-episodic tasks.

The policy $\pi(s, a)$ of an agent is a mapping of each state $s$ and action $a$ to the probability of taking action $a$ in state $s$. The goal of an agent is to improve its policy by maximizing the cumulative reward, also called the *expected return*, the agent receives over time.

Stellenbosch University http://scholar.sun.ac.za



There are different ways of calculating the expected return $R_t$, based on the specific task the agent has to solve. Some tasks can be broken up into a series of episodes or trials, where each episode ends in a *terminal* state. At the end of each episode, the agent is reset to a starting state. In such *episodic tasks*, we obtain the expected return by summing the total received rewards over a finite horizon $h$:

$$R_t = \sum_{k=0}^{h} r_{t+k+1} \tag{1}$$

Some tasks never end; thus, the above sum may be infinite. This problem may be solved by discounting future rewards:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2}$$

where $\gamma$ is the *discount rate* and $0 \leq \gamma < 1$. In our discussions, we will focus exclusively on this case, which is called the *discounted infinite horizon* case. Episodic tasks can also be handled by this definition of expected return by introducing an *absorbing state* which is entered just after the terminal state. The only transition from the absorbing state is to itself, with an associated reward of zero.

Most reinforcement learning algorithms are based on estimating *value functions* that estimate the utility of states. The value or utility of a state is the future reward, or return, that an agent can expect. As the future rewards depend on which actions an agent takes, the value function depends on the particular policy the agent follows. The value $V^\pi(s)$ of a state $s$ under policy $\pi$, is the expected return by following policy $\pi$ from state $s$:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\}, \tag{3}$$

where $E_\pi\{\}$ denotes the expected return when policy $\pi$ is followed. For the discounted infinite horizon case, we have:

$$V^\pi(s) = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}. \tag{4}$$

The *optimal value function* $V^*$ is attained by maximizing $V^\pi$ for all states:

$$V^*(s) = \max_\pi V^\pi(\forall s). \tag{5}$$

The *optimal policy* is defined as the policy corresponding to the optimal value function in the maximization above:

$$\pi^* = \arg\max_\pi V^\pi(\forall s). \tag{6}$$

Stellenbosch University http://scholar.sun.ac.za



In a MDP, we have a model of the environment dynamics in the form of state transition probabilities $T$ and the reward function $R$; thus, we can use the dynamic programming technique called *value iteration* to find the optimal value function. Once we have the optimal value function, we can obtain the *optimal policy* $\pi^*$ by choosing, in each state, the action that results in the maximum value function of all the immediate successor states:

$$\pi^*(s) = \arg\max_a V^*(s'), \tag{7}$$

where $s'$ is the successor of state $s$.

In reinforcement learning problems, an agent generally does not have access to the environment dynamics in the form of the transition probabilities $T$; thus, we cannot use dynamic programming techniques. In the next sections, we examine reinforcement learning methods based on dynamic programming [1], where we do not have access to the environment dynamics. Instead, an agent has to learn from the environment through the rewards experienced by taking different actions.

## 3.2 Temporal-Difference Learning

We now turn our attention to the problem of learning the optimal policy without perfect knowledge of the environment. The only way we can learn about the environment is to explore it by taking actions, observing the reward and use the experience to update the value function. One way of solving the problem is to incrementally estimate the value function $V^\pi$ as we encounter each new state. We denote this approximate value function by $V$.

The class of temporal-difference learning [28] algorithms update the current estimate $V(s_t)$ by using the value function estimates of *temporally successive* states. Temporal-difference methods are called *bootstrapping* methods, because they update estimates based on other estimates. By looking one step ahead at the value function of the next state, we can update the current value function estimate as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \tag{8}$$

where $\alpha$ is the step size parameter.

[1] Barto and Sutton [30] present a unified view relating dynamic programming, Monte Carlo, and temporal-difference methods for solving reinforcement learning problems.

Stellenbosch University http://scholar.sun.ac.za



```
Initialize V(s) arbitrarily, π to the policy to be evaluated
repeat for each episode:
    Initialize s
    repeat for each step in episode:
        choose action a in state s from policy π
        take action a; observe reward r, and next state s'
        V(s) ← V(s) + α[r + γV(s') − V(s)]
        s ← s'
    until s is terminal
```

Figure 3.2: Estimating $V^{\pi}$ with TD(0).

The algorithm, called TD(0) for reasons we will see shortly, is shown in Figure 3.2. Recall that the value function $V(s)$ is the expected return of following policy $\pi$ from state $s$. Thus, the TD(0) algorithm *predicts* the reward an agent will receive by following policy $\pi$ from state $s$. It has been shown that TD(0) converges with probability 1 to $V^{\pi}$ for any fixed $\pi$ with an appropriate choice of $\alpha$. If we denote $\alpha_k(a)$ as the step size parameter after the $k$th selection of action $a$, a suitable choice is $\alpha_k(a) = \frac{1}{k}$. This follows from the well-known result in stochastic approximation theory giving the conditions for convergence with probability 1 as:

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \qquad \text{and} \qquad \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty. \tag{9}$$

Although this is a useful theoretical result, the step size decrease above is seldom used in practice [30]. Instead, a constant step size $\alpha_k(a) = \alpha$ is used. This may be so for two reasons: first, the convergence is often slow or needs considerable tuning for a satisfactory convergence rate; second, in non-stationary environments, convergence is undesirable as the reward function $R$ may change over time, thus, we want our learned policy to continually change in response to the latest received rewards.

## 3.3 Q-Learning

In the previous section, we saw how TD(0) can be used for predicting the expected reward of a particular policy $\pi$ by estimating the value function. In this section, we look at the *control* problem; we want to find the optimal policy $\pi^*$.

Stellenbosch University http://scholar.sun.ac.za



If the agent knows the transition probabilities $T$ of the environment, it can choose the action that leads to the successor state with the combined maximum value function (Equation 7) and immediate reward. The problem is that we generally do not have a model of the environment; thus, we do not know which actions take us to which states. The solution is to define a new value function $Q^{\pi}(s,a)$, defined as the value of taking action $a$ in state $s$ while following policy $\pi$. This new value function is called the *action*-value function, and $V^{\pi}(s)$ the *state*-value function.

We define $Q^*(s,a)$ as the expected return of taking action $a$ in state $s$, and following the *optimal policy* from then on. Thus, we can write $Q^*(s,a)$ in terms of $V^*(s)$:

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \tag{10}$$

Recall that $V^*(s)$ is the value of taking the best step initially, so we also have:

$$V^*(s) = \max_a Q^*(s,a), \tag{11}$$

which enables us to write Equation 10 recursively:

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1},a') \mid s_t = s, a_t = a\}. \tag{12}$$

Whereas TD(0) is used to predict the expected return of states while following policy $\pi$, Q-Learning [34] incrementally estimates the optimal action-value function $Q^*(s,a)$. The update rule is given by:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]. \tag{13}$$

The Q-Learning algorithm shown in Figure 3.3 converges to the optimal action-value function $Q^*$ with probability 1 under the same conditions for $\alpha$ as in TD(0), provided each state-action pair is tried infinitely often. We will prove the conversion results in a later section.

In the Q-Learning algorithm, we must select actions based on a suitable exploration strategy derived from $Q$. Any strategy that guarantees that each state-action pair will be tried infinitely often will suffice. One of the simplest strategies is $\epsilon$-greedy, where an agent chooses the action with maximal Q-value in that state with probability $1-\epsilon$ and a random action with a small probability $\epsilon$. When an agent chooses an action with maximum Q-value, it is *exploiting* previously stored information, whereas random actions result in *exploration*. We will discuss the tradeoff between exploration and exploitation in Section 3.7.

Stellenbosch University http://scholar.sun.ac.za



```
Initialize Q(s,a) arbitrarily
repeat for each episode:
    Initialize s
    repeat for each step in episode:
        choose action a in state s using exploration policy derived from Q
        take action a; observe reward r, and next state s'
        Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)]
        s ← s'
    until s is terminal
```

Figure 3.3: Estimating $Q^*$ with Q-Learning.

Q-Learning is called an *off-policy* learning algorithm because it converges to the optimal value function *independent* of the exploration policy being followed. In other words, the details of the particular exploration strategy do not influence the value function, but only the rate of convergence. There is also an on-policy Q-Learning algorithm called SARSA [30], in which the exploration strategy is taken into account. However, both algorithms converge to the same value function when $\epsilon$, the probability of exploration, decreases towards zero.

## 3.4 TD($\lambda$) Learning

The TD(0) learning method we studied previously is a special case of a class of temporal-difference learning methods called TD($\lambda$), with $\lambda = 0$. In the update rule of TD(0) (Equation 8), we look ahead one step to the value function of the next state. The update moves the estimate closer to the target value of estimated return:

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}). \tag{14}$$

We can generalize the target to the case of $n$ steps, also called the *corrected n-step truncated return*:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}). \tag{15}$$

It can be shown [30] that the expected value of the corrected n-step truncated return is an improvement over the current value function as an approximation to the true value function. This is called the *error reduction property*.

Stellenbosch University http://scholar.sun.ac.za



```
Initialize V(s) arbitrarily, and e(s) = 0 for all s ∈ S
repeat for each episode:
    Initialize s
    repeat for each step in episode:
        choose action a in state s from policy π
        take action a; observe reward r, and next state s'
        δ ← r + γV(s') − V(s)
        e(s) ← e(s) + 1
        For all s:
            V(s) ← V(s) + αδe(s)
            e(s) ← γλe(s)
        s ← s'
    until s is terminal
```

Figure 3.4: Estimating $V^{\pi}$ with TD($\lambda$).

The idea of the TD($\lambda$) algorithm due to Sutton [28] is to use a weighted average of n-step returns as the target for the value function update. The update is towards a return called the $\lambda$ *return*:

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}, \tag{16}$$

where each n-step return is weighted by $\lambda^{n-1}$ and $0 \leq \lambda \leq 1$. This weighted average also has the error reduction property. The value function update becomes:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t^{\lambda} - V(s_t)]. \tag{17}$$

This update rule is problematic as it uses at each step information of states many steps in the future. Using the concept of *eligibility traces*, we can change the update rule into an incremental version. The eligibility trace $e_t(s)$ is a variable which keeps track of how recently a state has been visited. At each time step, the eligibility trace of the state just visited is incremented by 1 and all the other states decay with a factor $\gamma\lambda$:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t, \end{cases} \tag{18}$$

for all $s \in S$.

The eligibility trace records how eligible a state is to undergo change. Thus, it is a method of solving the temporal credit assignment problem by crediting states which occurred recently more than those which occurred a long time ago.

Stellenbosch University http://scholar.sun.ac.za



```
Initialize Q(s, a) arbitrarily, and e(s, a) = 0 for all s, a
repeat for each episode:
    Initialize s, a
    repeat for each step in episode:
        take action a; observe reward r, and next state s'
        choose action a' from s' using policy derived from Q
        a* ← arg max_b Q(s', b) (if a' ties for the max, then a* ← a')
        δ ← r + γQ(s', a*) − Q(s, a)
        e(s, a) ← e(s, a) + 1
        For all s, a:
            Q(s, a) ← Q(s, a) + αδe(s, a)
            if a' = a*
              then e(s, a) ← γλe(s, a)
              else e(s, a) ← 0
        s ← s'
        a ← a'
    until s is terminal
```

Figure 3.5: Watkins's Q($\lambda$) algorithm.

The one-step TD error between the current and the next state is:

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t). \tag{19}$$

Eligibility traces specify how much to update other states based on the one-step error. The eligibility traces effectively distribute the error to the states in proportion to how recently they were visited. At each step of the TD($\lambda$) algorithm, each state is updated towards the product of the one-step error and its eligibility:

$$V_{t+1}(s) = V_t(s) + \alpha\delta_t e_t(s), \qquad \text{for all } s \in S. \tag{20}$$

Figure 3.4 shows the online tabular version of TD($\lambda$) which has been proven [9; 28] to converge to $V^*$ with probability 1.

## 3.5 Q($\lambda$) Learning

By modifying TD($\lambda$) to learn the action-value function $Q^*$, we can change Q-Learning into a multi-step version. Watkins [33; 30] and Peng [23] proposed two slightly different

Stellenbosch University http://scholar.sun.ac.za



versions, differing in the way they handle exploratory actions. We follow Watkins's approach.

We replace the state-value function estimate $V(s)$ with the action-value function estimate $Q(s,a)$ in the update rule of TD($\lambda$). The eligibility trace $e_t(s,a)$ is also changed to be a function of state-action pairs. We now have for the update:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a), \tag{21}$$

where

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t). \tag{22}$$

The algorithm for Q($\lambda$) can be found in Figure 3.5. When selecting an exploratory action, the eligibility trace for that state and action is set to zero. This is done because this is an off-policy algorithm, where we approximate the greedy policy while following an exploratory policy. Thus, we cannot give credit to states where we take a non-greedy action. Setting the eligibility trace to zero precludes these states from being updated.

## 3.6 Convergence Properties of Q-Learning

As we have stated in Section 3.3, the Q-Learning algorithm converges to the optimal action-value function $Q^*$. By relating the Q-Learning algorithm to a theorem in stochastic approximation theory, Jaakkola et al [12] prove the convergence of Q-Learning in the case of an non-deterministic MDP. We will prove [20] the simpler case of a deterministic MDP.

In a deterministic MDP, the step size parameter $\alpha$ in Equation 13 may be set to 1. The proof consists of showing that the error between the estimated and actual Q-values is reduced by at least a factor $\gamma$ after an update by the new update equation:

$$Q(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a). \tag{23}$$

**Theorem 3.1 (Convergence of Q-Learning for deterministic Markov decision processes).** *Consider an agent in a deterministic MDP with bounded rewards* $| R(s,a) | \leq c \quad \forall s,a$ *for some constant c. Assume that* $\hat{Q}(s,a)$ *is initialized with arbitrary finite values for all s and a and that the agent updates* $\hat{Q}(s,a)$ *according to Equation 23 with* $0 \leq \gamma < 1$. *Let* $\hat{Q}_n(s,a)$ *denote the value function estimate after the*

Stellenbosch University http://scholar.sun.ac.za



*$n$-th update. If each state-action pair is visited infinitely often, $\hat{Q}_n(s,a)$ will converge to the optimal value function $Q^*(s,a)$ as $n \to \infty$, for all $s$, $a$.*

*Proof.* As each state-action transition occurs infinitely often, we can consider consecutive intervals during which each state-action transition has occurred at least once. The proof shows that the maximum error is reduced by at least a factor $\gamma$ after each such interval. Let $\hat{Q}_n$ represent the value function estimates after $n$ updates, and let $\Delta_n$ be the maximum error of $\hat{Q}_n$:

$$\Delta_n \equiv \max_{s,a} \mid \hat{Q}_n(s,a) - Q^*(s,a) \mid$$

Let $s'$ denote the next state if the agent follows action $a$ in state $s$. For any table entry $\hat{Q}_n(s,a)$ that is updated in iteration $n+1$, the magnitude of the error of the new estimate $\hat{Q}_{n+1}(s,a)$ is

$$\begin{aligned} \mid \hat{Q}_{n+1}(s,a) - Q^*(s,a) \mid &= \mid (r + \gamma \max_{a'} \hat{Q}_n(s',a')) - (r + \gamma \max_{a'} Q^*(s',a')) \mid \\ &= \gamma \mid \max_{a'} \hat{Q}_n(s',a') - \max_{a'} Q^*(s',a') \mid \\ &\leq \gamma \max_{a'} \mid \hat{Q}_n(s',a') - Q^*(s',a') \mid \\ &\leq \gamma \max_{s'',a'} \mid \hat{Q}_n(s'',a') - Q^*(s'',a') \mid \end{aligned}$$

The first inequality follows because for any two functions $f_1$ and $f_2$ we have:

$$\mid \max_a f_1(a) - \max_a f_2(a) \mid \quad \leq \quad \max_a \mid f_1(a) - f_2(a) \mid$$

The second inequality introduces a new variable $s''$ over which we perform the maximization. This inequality is legitimate because the maximum value will be at least as great when we allow this additional variable to vary. Note that the last inequality is exactly the definition of our maximum error $\Delta_n$, thus we have:

$$\mid \hat{Q}_{n+1}(s,a) - Q^*(s,a) \mid \quad \leq \quad \gamma \Delta_n$$

This means that the error after updating $\hat{Q}_{n+1}(s,a)$ for any $s,a$ is at most $\gamma$ times the maximum error in the $\hat{Q}_n$ table, $\Delta_n$ . The largest error in the initial table $\Delta_0$ is bounded, because $\hat{Q}_0(s,a)$ and $Q^*(s,a)$ are bounded. After the first interval during which each $s,a$ pair is visited, the largest error in the table will be at most $\gamma\Delta_0$. After $k$ such intervals, the error will be at most $\gamma^k\Delta_0$. Since each state-action pair is visited infinitely often, the number of such intervals is infinite, so $k \to \infty$. Because $\gamma < 1$, we have $\Delta_n \to 0$ as $n \to \infty$, which proves the theorem. □

Stellenbosch University http://scholar.sun.ac.za



## 3.7 Exploration vs Exploitation

When an agent has to act optimally in an unknown environment, it has to balance the two opposing objectives of exploration and exploitation. In order to find an optimal control policy, the agent has to sufficiently explore the environment. However, exploration is typically an expensive operation; thus, it is sensible for the agent to exploit the information it has already learned. If the agent only exploits, it may not find the optimal policy; if it just explores, it wastes too much time exploring parts of the state space which are irrelevant to the task at hand. In other words, we have to both explore and exploit. The question arises of when to explore and when to exploit, which defines the trade-off which most reinforcement learning algorithms have to solve. Thrun [32] examines this trade-off and presents techniques for balancing each objective.

There are two major types of exploration methods: *directed* and *undirected* exploration. Undirected exploration is based on randomness, e.g. an agent may take a random action with a uniform probability distribution. On the other hand, directed exploration techniques estimate the exploration utility of actions, and choose actions which maximize the expected information gained by exploration.

The simplest method for undirected exploration selects actions from a uniform probability distribution. This results in a random walk over the state space and makes no use of previously learned information. With $\epsilon$-greedy action selection, an agent selects the action with maximum value function most of the time, but with a small probability $\epsilon$ it selects a random action with uniform probability distribution.

One problem with the $\epsilon$-greedy method is that it selects exploratory actions with equal probability; thus, bad and good actions are equally likely. In some tasks where the agent receives large negative reinforcement for bad actions, it may be better to weight the probability distribution – e.g. Gibbs or Boltzmann distribution – as a function of the action-values already learned. This is called *softmax* action selection. The probability of selecting action $a$ is given by:

$$P(a \mid s) = \frac{e^{Q(s,a)/\tau}}{\sum_{b \in A(s)} e^{Q(s,b)/\tau}}, \tag{24}$$

where $A(s)$ is the set of actions available in state $s$, and $\tau$ is the "temperature" parameter. With a high temperature, all actions are almost equally probable, whereas lower temperatures give more weight to actions with higher Q-values. Typically, learning is

Stellenbosch University http://scholar.sun.ac.za



started with a high temperature and is then gradually reduced over time. This annealing schedule encourages exploration in the beginning of learning and gradually shifts to an increasingly greedy policy.

In all of the reinforcement learning algorithms we have discussed so far, convergence to an optimal policy was guaranteed as long as each state-action pair was visited infinitely often. The simple undirected exploration techniques satisfy the convergence criterion, but they may not be as efficient as directed exploration where specific knowledge guides the exploration. These are heuristic techniques, as it is not known in advance if an action will improve an agent's exploration knowledge.

One of the proposed heuristic techniques selects actions which have been selected less frequently [5], so called *counter-based exploration*. Each state-action value has a counter which is incremented each time the particular state-action sequence has been encountered. Using these counters, the neighbouring state which has been selected the least number of times will be selected.

Another technique, *recency-based exploration*, selects actions which were taken less recently [30; 29] by recording the time since the last action was taken.

*Error-based exploration* favours actions which have shown to have a high prediction error [21; 24]. The assumption is that states with high prediction error have not been sufficiently explored. The efficiency of this method depends on the accuracy of the prediction error.

Q-Learning can perform directed exploration based on a purely exploitative policy by setting all initial values to an overestimate of the value of each state-action pair. Exploitation will then select unexplored actions because of the high value of taking such an action. Over time, this ensures that all state-action sequences are followed. The exploration is gradually decreased as the estimated value function converges to the true value function. The only problem with this technique is that it works well only in stationary environments. As the reward function changes over time in a dynamic environment, convergence to one policy is not desirable; thus, this method is often used in combination with other exploration techniques.

Stellenbosch University http://scholar.sun.ac.za



## 3.8 Summary

The reinforcement learning problem is a general framework in which an agent interacts with an environment by executing actions and receiving rewards. The agent's goal is to maximize the rewards received over time. Most reinforcement learning algorithms can learn from delayed rewards by estimating future rewards, while continuously interacting with the environment. The value function is this estimated future reward for each state which the agent is attempting to learn.

The learning task can be formalized by using the theory of Markov decision processes, which can be used to solve sequential decision optimization problems. Reinforcement learning algorithms are related to dynamic programming approaches to solving Markov decision processes. The difference is that, in general, reinforcement learning algorithms do not have a model of the environment; instead, they learn from the environment from experience.

Q-Learning is a reinforcement learning algorithm which is able to learn the optimal control policy without having prior knowledge of the environment. It uses temporal difference methods for estimating the value function of the current state-action pair by using the value function of the next state. By updating more than one state through the concept of eligibility traces, the reinforcement learning algorithm may learn more efficiently. This is one of the basic mechanisms for temporal credit assignment. Eligibility traces are especially useful when rewards are delayed by many steps, or when a task does not completely satisfy the Markov property.

All of the reinforcement learning algorithms discussed have been proved to converge to the optimal value function with probability 1, provided that all states are updated infinitely often.

We also discussed the importance of exploration in learning the optimal control policy. There is a trade-off between exploration and exploitation which has to be addressed in all reinforcement learning algorithms.

Stellenbosch University http://scholar.sun.ac.za

# Chapter 4

# Q-Learning for Traffic Routing

In this chapter, we examine Q-Learning for routing packets in a network, and compare its performance with the Shortest Path and Bellman-Ford routing algorithms. We proceed to study variations of Q-Learning designed to improve the routing performance.

## 4.1 Optimization of Packet Delivery Time

We consider the problem of routing individual data packets to their destination nodes on a path, minimizing some objective function, e.g. the average packet delivery time, or the total cost.

### 4.1.1 Q-Routing

Q-Routing [19; 2] is a distributed, adaptive, on-line routing algorithm that uses the Q-Learning framework.

In Q-Routing, each node decides which neighbouring node to forward packets to in order to minimize the average packet delivery time based solely on local information. This is important because, as networks grow in size, the amount of routing information to be transmitted becomes significant, and may impact on the overall network performance. Each node has to store routing information in order to make the routing decision. This routing information needs to be updated continuously to reflect the current network state.

Stellenbosch University http://scholar.sun.ac.za



The time a packet takes to reach its destination node consists of the total transmission delay on intermediate links, and the total waiting time in queues at the intermediate nodes. The routing information at each node is stored in a Q-table as the estimated travel time from each neighbouring node to each destination node. As a packet is sent, the receiving node sends the estimate for the remaining travel time for this packet back to the sending node. This estimate is then used by the sending node as a reinforcement signal to update the corresponding Q-value. After an initial learning period, the Q-values converge to the actual estimates; the stored network state will more closely represent the actual network state, resulting in better performance.

Each node $x$ maintains a table of Q-values $Q_x(y,d)$ where $y \in N(x)$, the set of neighbours of node $x$, and $d \in V$, the set of nodes of the network. Each Q-value represents the estimated time for a packet to reach its destination $d$, from node $y$, *excluding* the waiting time in $y$'s queue. If the Q-values represent the current state of the network, the best choice is to send the packet to the neighbouring node with the least estimated delivery time for destination $d$. This is a local greedy policy.

Thus, if node $x$ receives a packet $P(s,d)$ with source $s$ and destination $d$, it selects the neighbouring node $y$ from the vector of Q-values, $Q_x(*,d)$, for which the $Q_x(y,d)$ value is minimum. The packet is then sent to node $y$ where the best estimate $Q_y(\hat{z},d)$ is computed and sent back to node $x$. Node $x$ then updates its Q-value $Q_x(y,d)$. The estimate $Q_y(\hat{z},d)$ sent back from node $y$ represents the estimated time for a packet to travel from $y$ to $d$. This means that the estimated travel time from $x$ to $d$ equals the estimate from $y$ plus the transmission time $\delta$, between $x$ and $y$, and the waiting time, $q_y$ in queue $y$ as shown in the next equation:

$$Est(y,d) = Q_y(\hat{z},d) + q_y + \delta, \tag{25}$$

where $Q_y(\hat{z},d)$ is the minimum estimated delivery time from node $y$ to $d$:

$$Q_y(\hat{z},d) = \min_{z \epsilon N(y)} Q_y(z,d). \tag{26}$$

The Q-value of the sending node $x$ is updated as follows:

$$Q_x(y,d) = Q_x(y,d) + \eta_f(Est(y,d) - Q_x(y,d)), \tag{27}$$

where $\eta_f$ is the learning rate.

If enough packets are sent to each node and the traffic pattern remains stationary, the Q-values will converge to their optimum values, because the Q-Learning algorithm

Stellenbosch University http://scholar.sun.ac.za



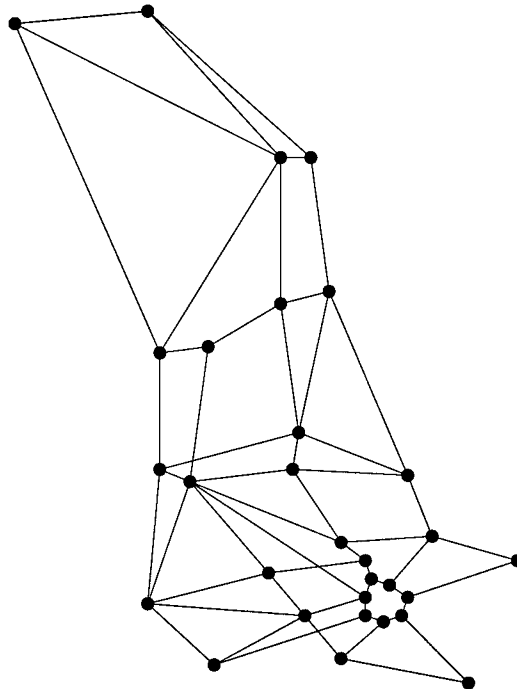

Figure 4.1: The British Synchronous Digital Hierarchy (SDH) network topology.

converges to the optimal policy, as proved in the previous chapter. The overhead of Q-Routing is minimal compared to other distributed routing algorithms, such as Bellman-Ford. Each node just has to send an estimate to the node from which it received the packet. If this estimate is small compared to the packet size, the estimate transmission becomes negligible.

We ran simulations on the network topology of the British Synchronous Digital Hierarchy (SDH) [25] of Figure 4.1 with low, medium and high network loads. We compared the performance of Q-Routing to that of Bellman-Ford and static Shortest Path routing. In all cases, the results show the average over 25 runs. All node buffers were of length 200 to ensure that no packets were dropped.

We generated packets with a uniform random distribution. At each simulation time step, each node generates a packet with a probability $p$ with a random destination node. If there are $n$ nodes in the network the overall load in the network is $L = n \times p$ packets generated per simulation time step.

For Q-Routing we found empirically that a learning rate of $\eta_f = 0.95$ resulted in good routing performance. The Shortest Path routing algorithm routes packets on the route

Stellenbosch University http://scholar.sun.ac.za



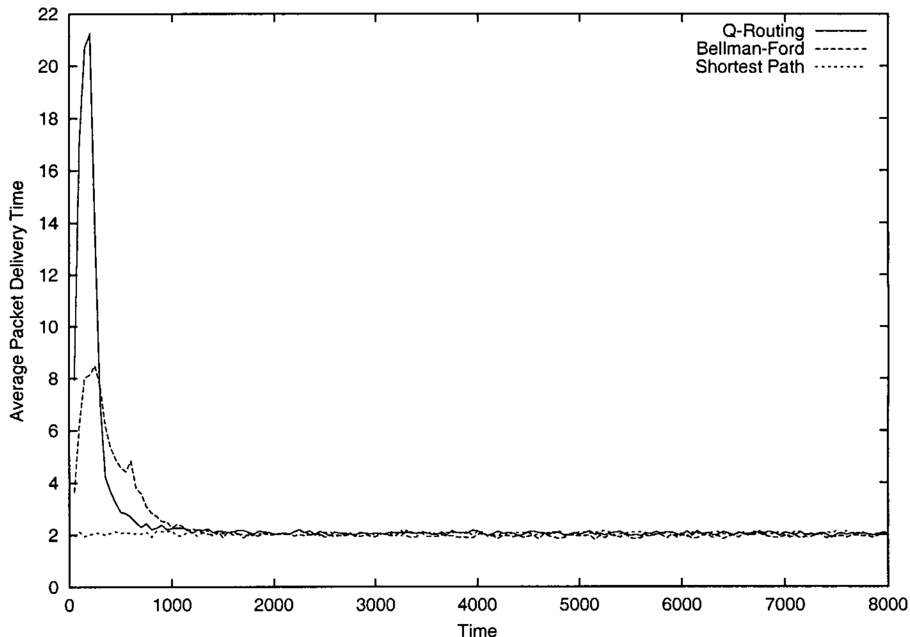


Figure 4.2: Average packet delivery times for network load 1.2 for the SDH network topology.

with the minimum number of hops to the destination.

We selected the average queue length as a metric for comparing link costs in the Bellman-Ford algorithm. Because the queue lengths fluctuate, distance vectors at all nodes are always out of date and thus only *estimated* distances. For this reason, we need to update distances towards an estimated value by using a step size parameter, or learning rate. Empirical results showed that for Bellman-Ford a learning rate of 0.9 gives good results.

Normally, Bellman-Ford sends a distance vector $V_n$ of its current shortest distances for all possible destinations to each of its $k$ neighbours. In order to fairly compare Bellman-Ford and Q-Routing, both algorithms must send the same amount of routing information over the network. We accomplished this by having Bellman-Ford send the distance vector to a neighbour only after $n$ packets were sent to that neighbour.

Figure 4.2 shows the performance of the three routing algorithms for low load $L = 1.2$ packets per time step. From the figure, it is clear that the static Shortest Path routing performs the best. Bellman-Ford has a lower initial peak than Q-Routing, but after time step 1000 there is not much difference between the three algorithms.

Stellenbosch University http://scholar.sun.ac.za



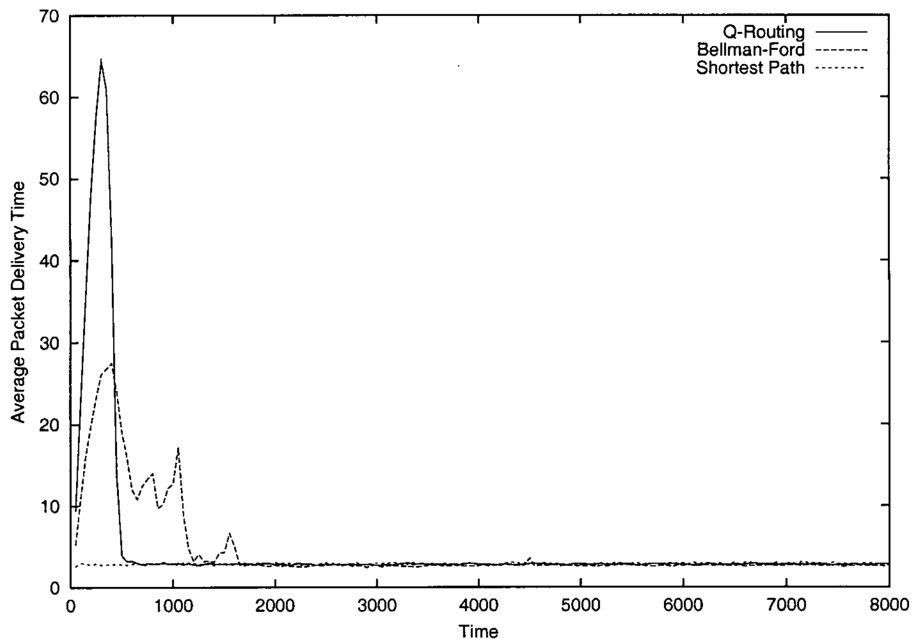


Figure 4.3: Average packet delivery times for network load 2.2 for the SDH network topology.

With medium load of 2.2 packets per time step (Figure 4.3) performance is similar to that of low load, except that the peaks of Q-Routing and Bellman-Ford are higher during the training phase. Shortest Path routing is still superior.

Figure 4.4 shows the performance with a load of 3.2 packets per time step. Here, we see that static Shortest Path routing begins to break down, as the average packet delivery time increases linearly up to time step 2000, after which it settles. Even though the buffers could store 200 packets, buffers still overflowed, dropping an average of 4 packets every 50 time steps. Q-Routing has the lowest average packet delivery time, but it has a higher initial peak during the training phase. It also converges about three times faster than Bellman-Ford.

Under low load conditions, the minimal hop path is the optimal route and the static Shortest Path routing algorithm performs best. Under high loads, node buffers fill up, and the shortest routes become congested, increasing the delay of packets on those routes. The adaptive routing algorithms perform better by diverting traffic away from the congested nodes.

Stellenbosch University http://scholar.sun.ac.za



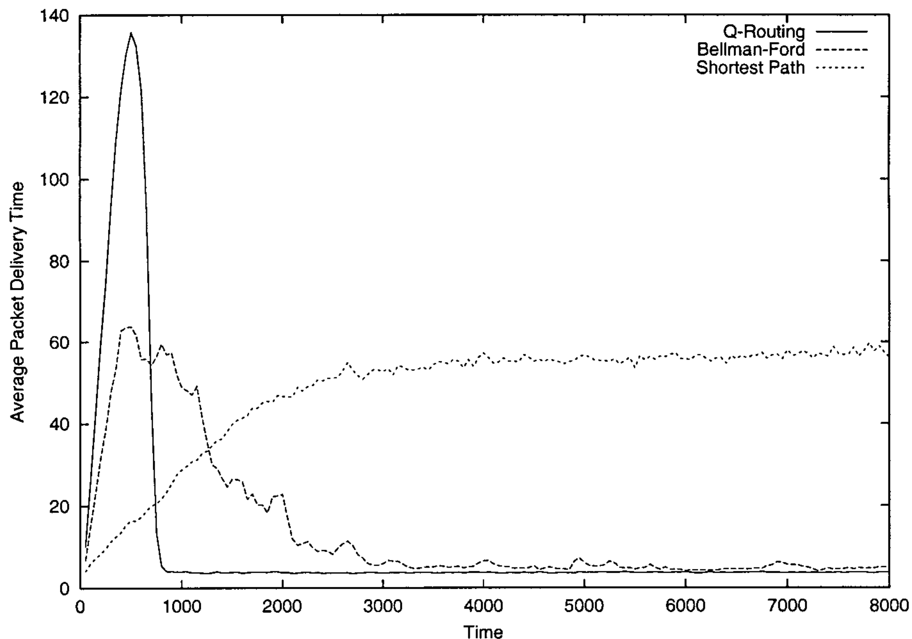


Figure 4.4: Average packet delivery times for network load 3.2 for the SDH network topology.

Figure 4.5 shows the average packet delivery time of Bellman-Ford over a longer duration of 40000 time steps, with error bars corresponding to 1 standard deviation. This shows long term instability caused by routing loops. Temporary routing loops occur because updates to routing tables take time to propagate through the network, causing routing tables to be adjusted using information that may be out of date. There are methods that maintain loop-free paths, or use predecessor information to eliminate the performance problems associated with Bellman-Ford [11; 7]. The ARPANET initially used the Bellman-Ford algorithm, but it was replaced in 1979 by a brute force link state algorithm as a result of long-lived loops. [1]

Figure 4.6 shows that Q-Routing does not have the long term instability problems associated with Bellman-Ford under high load conditions.

### 4.1.2 DRQ-Routing

In Q-Routing, there is one Q-value update for each packet hop through the network. In Dual Reinforcement Q-Routing [18] (DRQ-Routing), there is an additional Q-value update for each packet hop. As node $x$ sends a packet $P(s, d)$ to node $y$, it appends

Stellenbosch University http://scholar.sun.ac.za



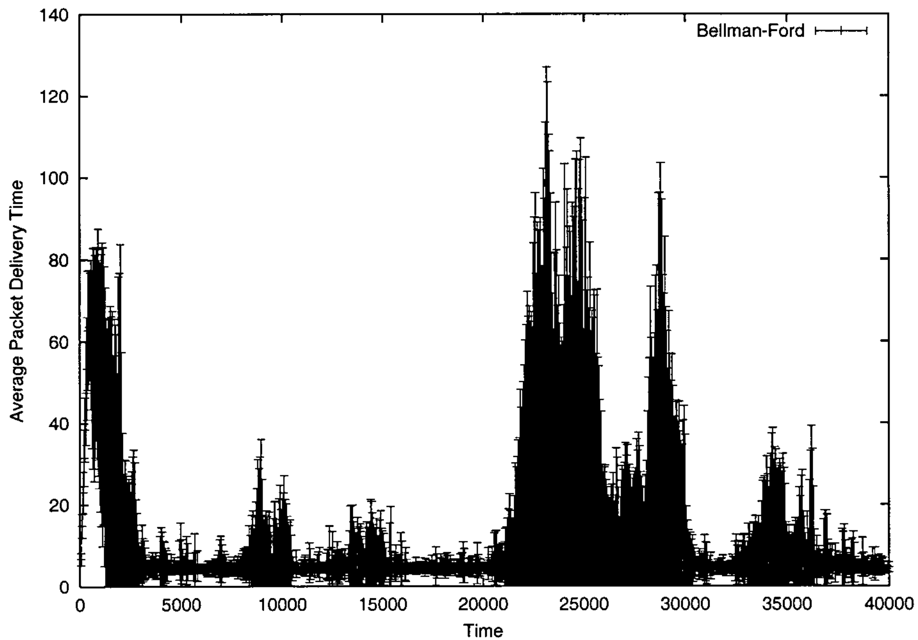


Figure 4.5: Average packet delivery times of Bellman-Ford for high network load for the SDH network topology. Error bars show standard deviations.

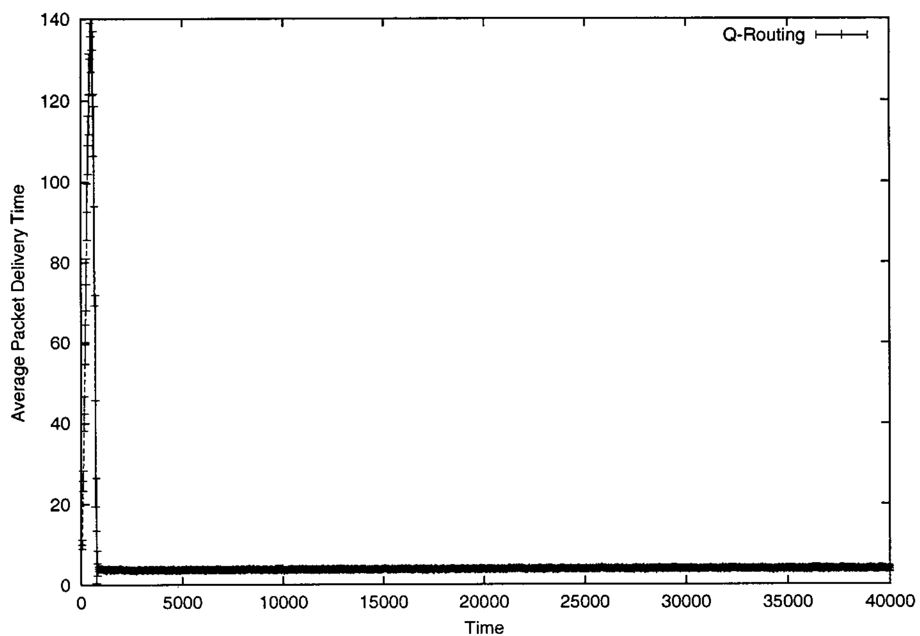


Figure 4.6: Average packet delivery times of Q-Routing for high network load for the SDH network topology. Error bars show standard deviations.

Stellenbosch University http://scholar.sun.ac.za



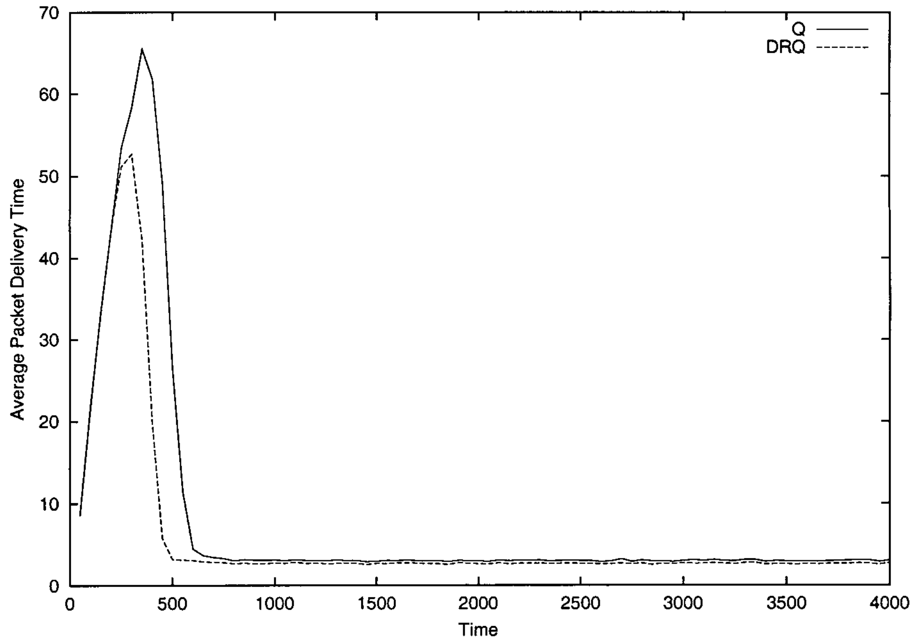


Figure 4.7: Comparing the average packet delivery times of Q-Routing and DRQ-Routing for network load 2.0 for the SDH network topology.

$Q_x(\hat{z}, s)$, which is the estimated travel time to reach the source from node $x$, to the packet. The receiving node, $y$, then updates the $Q_y(x, s)$ value. This is called backward exploration as the updates are performed in the reverse direction of travel. The update rule is:

$$Q_y(x, s) = Q_y(x, s) + \eta_b(Est(x, s) - Q_y(x, s)), \tag{28}$$

where $\eta_b$ is the backward learning rate.

This heuristic aims at improving the efficiency because, with one packet delivery, it estimates the best packet delivery path in both directions simultaneously. The overhead of DRQ-Routing is twice that of Q-Routing; however, it is negligible if the packet size is large in comparison with the extra estimates which have to be sent.

Figures 4.7 to 4.9 shows the results for network loads of 2.0, 3.0 and 4.0 packets per time step respectively for the 30 node SDH network. The forward learning rate, $\eta_f$ and backward learning rate, $\eta_b$ were both set to 0.9 in order to compare the relative performance of the two algorithms. Results show the average over 25 simulation runs.

In all cases, the results show that DRQ-Routing learns a good routing policy much

Stellenbosch University http://scholar.sun.ac.za



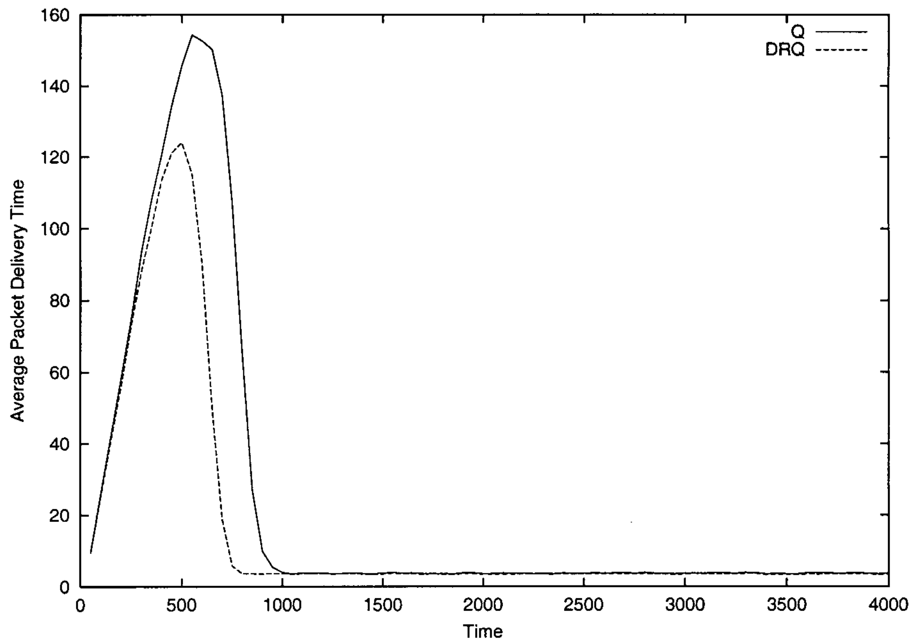


Figure 4.8: Comparing the average packet delivery times of Q-Routing and DRQ-Routing for network load 3.0 for the SDH network topology.

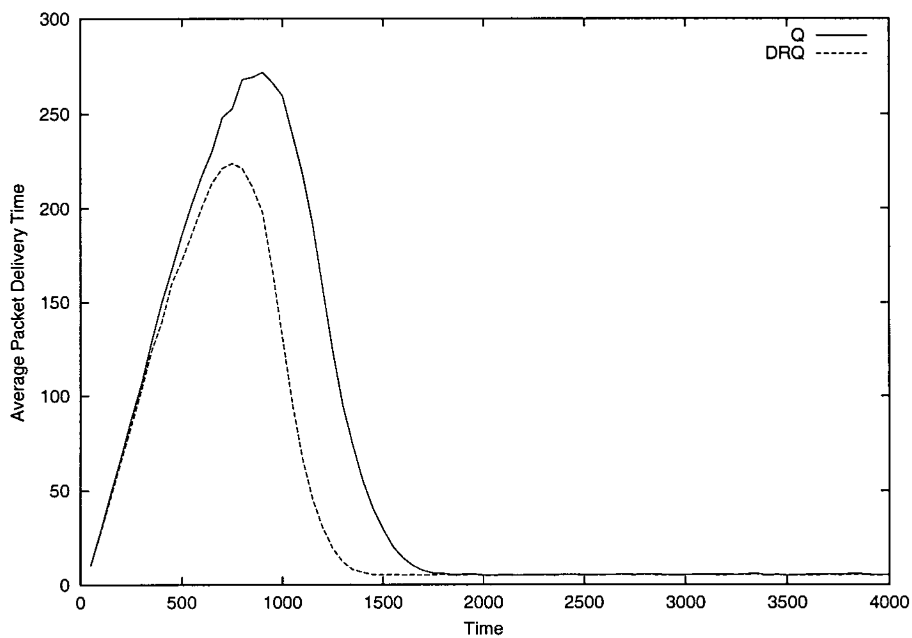


Figure 4.9: Comparing the average packet delivery times of Q-Routing and DRQ-Routing for network load 4.0 for the SDH network topology.

Stellenbosch University http://scholar.sun.ac.za



faster than Q-Routing.

### 4.1.3 CQ-Routing

A Q-value estimates the packet delivery time from a node, via its neighbour, to a destination. In Q-Routing, there is no measure of how close a Q-value is to the actual delivery time. The idea of Confidence-based Q-Routing [17] (CQ-Routing) is to attach a confidence measure to each Q-value to determine how much to update a particular Q-value. Confidence values range between 0 and 1 to reflect the accuracy of the corresponding Q-value. A value of 0 means the Q-value does not reflect the current network state, whereas a value of 1 denotes a completely reliable Q-value.

As a node sends a packet to a neighbour, the neighbour sends a confidence value $C_{est}$ together with the Q-value, back to the sending node. This confidence value depends on how recently an update to this Q-value occurred. The learning rate $\eta_f$ , of Equation 27 now becomes a function of the value $C_{est}$, and the old confidence value $C_{old}$ at the sending node.

The learning rate function proposed by Kumar [17] is:

$$\eta(C_{old}, C_{est}) = max(C_{est}, 1 - C_{old}) \tag{29}$$

This results in a *high* learning rate when confidence in the old Q-value $C_{old}$ is *low* or when confidence in the transmitted Q-value estimate $C_{est}$ is *high*.

The Q-value update rule of Equation 27 is replaced by:

$$Q_x(y,d) = Q_x(y,d) + \eta(C_x(y,d), C_y(\hat{z},d)) \times (Est(y,d) - Q_x(y,d)) \tag{30}$$

The confidence values are updated continuously to reflect the accuracy of the corresponding Q-values. Base case Q-values of the form $Q_x(y,y)$ represent the constant transmission delay $\delta$ between node $x$ and its neighbour *and* destination $y$. Thus, the corresponding $C_x(y,y)$, are not updated as they have a constant value of 1, i.e. there is full confidence in the corresponding Q-value.

Whenever a Q-value is updated according to Equation 30, the corresponding confidence estimate is updated as follows:

$$C_x(y,d) = C_x(y,d) + \eta(C_x(y,d), C_y(\hat{z},d)) \times (C_y(\hat{z},d) - C_x(y,d)) \tag{31}$$

Stellenbosch University http://scholar.sun.ac.za



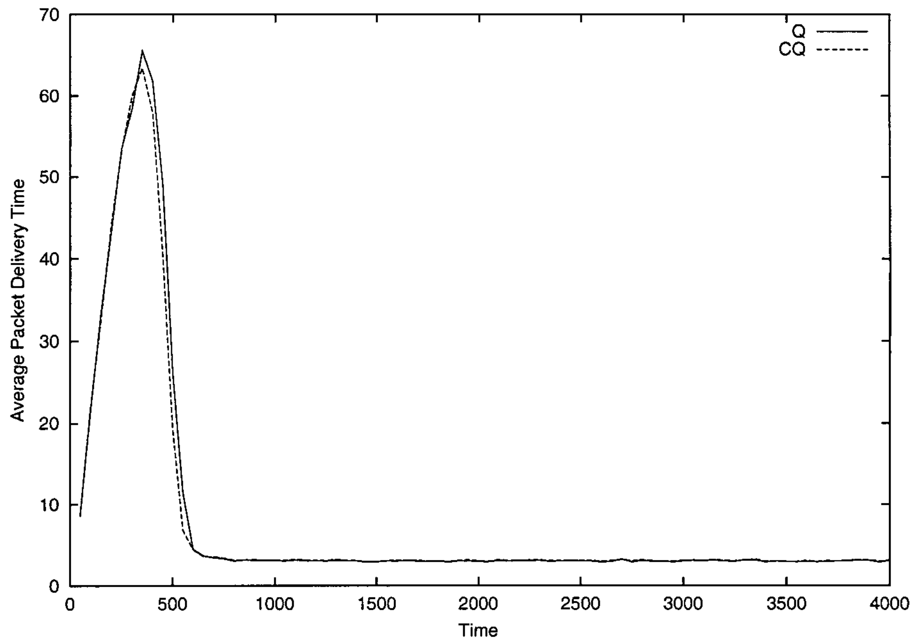


Figure 4.10: Comparing the average packet delivery times of Q-Routing and CQ-Routing for network load 2.0 for the SDH network topology.

All other confidence values which were not updated in the last time step, except the base cases, must decay with time, reflecting that the information of the Q-value is getting out of date:

$$C_x(y, d) = \lambda C_x(y, d), \tag{32}$$

where $\lambda$ is a constant decay factor between 0 and 1 and $C_x$ decays exponentially.

Figures 4.10 to 4.12 shows the results for network loads of 2.0, 3.0 and 4.0 packets per time step, respectively, for the 30 node SDH network. For CQ-Routing the confidence decay factor $\lambda$ was set to 0.99 and the learning rate for Q-Routing was 0.9. Results show the average over 25 simulation runs.

CQ-Routing has a faster convergence time than Q-Routing, especially for medium and high network loads where there are more of a difference.

Stellenbosch University http://scholar.sun.ac.za



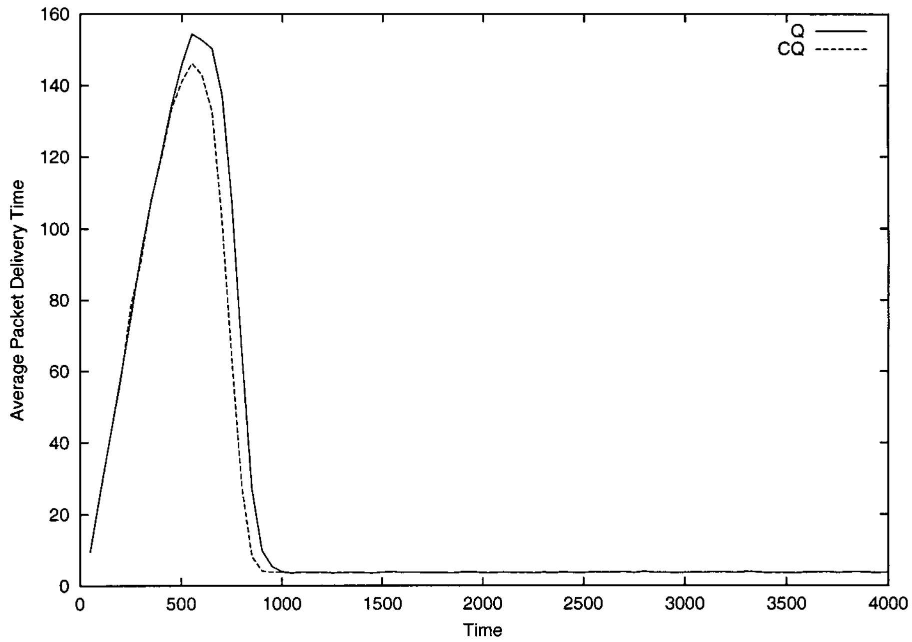


Figure 4.11: Comparing the average packet delivery times of Q-Routing and CQ-Routing for network load 3.0 for the SDH network topology.

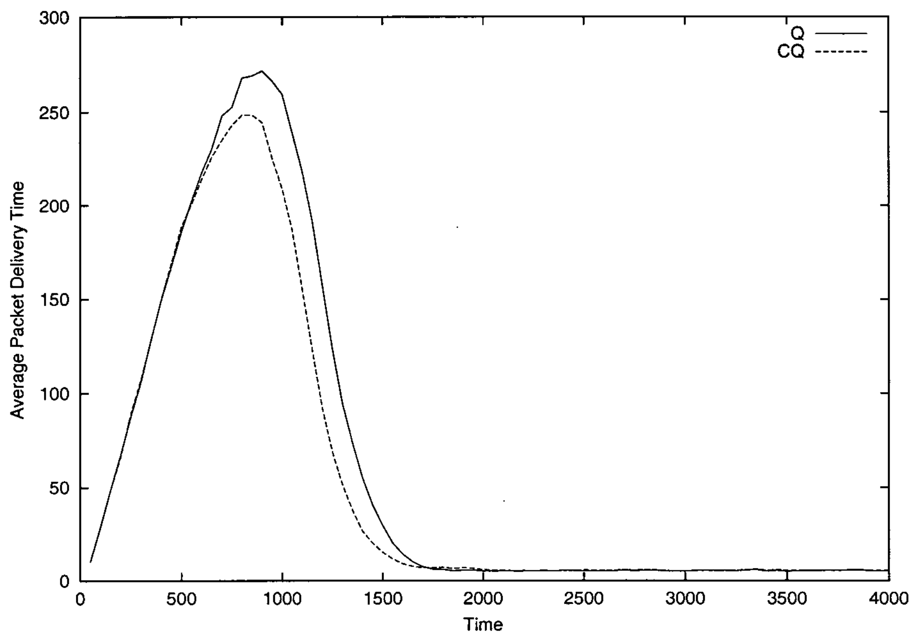


Figure 4.12: Comparing the average packet delivery times of Q-Routing and CQ-Routing for network load 4.0 for the SDH network topology.

Stellenbosch University http://scholar.sun.ac.za



### 4.1.4 CDRQ-Routing

When we add confidence values to DRQ-Routing, we obtain Confidence-based Dual Reinforcement Q-Routing [17] (CDRQ-Routing). This combines the benefits of adaptive learning rates of CQ-Routing, and dual direction exploration of DRQ-Routing into one algorithm.

In the forward direction of exploration, the Q-value update of the sending node $x$ is the same as the update rule of CQ-Routing (Equation 30). In the backward exploration direction, the Q-value update of the receiving node $y$ is:

$$Q_y(x,s) = Q_y(x,s) + \eta(C_y(x,s), C_x(\hat{z},s)) \times (Est(x,s) - Q_y(x,s)), \tag{33}$$

where $\eta(C_y(x,s), C_x(\hat{z},s))$ is the learning rate function of Equation 29, and $Est(x,s)$ represents the current estimated travel time from node $x$ to the source node $s$ (Equation 25), *including* the queuing delay at node $x$.

The corresponding confidence value update at the receiving node is:

$$C_y(x,s) = C_y(x,s) + \eta(C_y(x,s), C_x(\hat{z},s)) \times (C_x(\hat{z},s) - C_y(x,s)) \tag{34}$$

The confidence values in the forward direction is updated by Equation 31 and all remaining confidence values which were not updated are adjusted according to Equation 32.

Figures 4.13 to 4.15 show the results for network loads of 2.0, 3.0 and 4.0 packets per time step respectively for the 30 node SDH network. For CDRQ-Routing, the confidence decay factor $\lambda$ was set to 0.99. The results show the performance of CDRQ-Routing together with CQ-Routing, DRQ-Routing and Q-Routing of the previous three sections. Results show the average over 25 simulation runs.

The results show that by combining CQ-Routing and DRQ-Routing into CDRQ-Routing improves the convergence time, especially for higher loads. It also shows that the improvement of DRQ-Routing is more drastic than that of CQ-Routing, relative to normal Q-Routing.

Stellenbosch University http://scholar.sun.ac.za



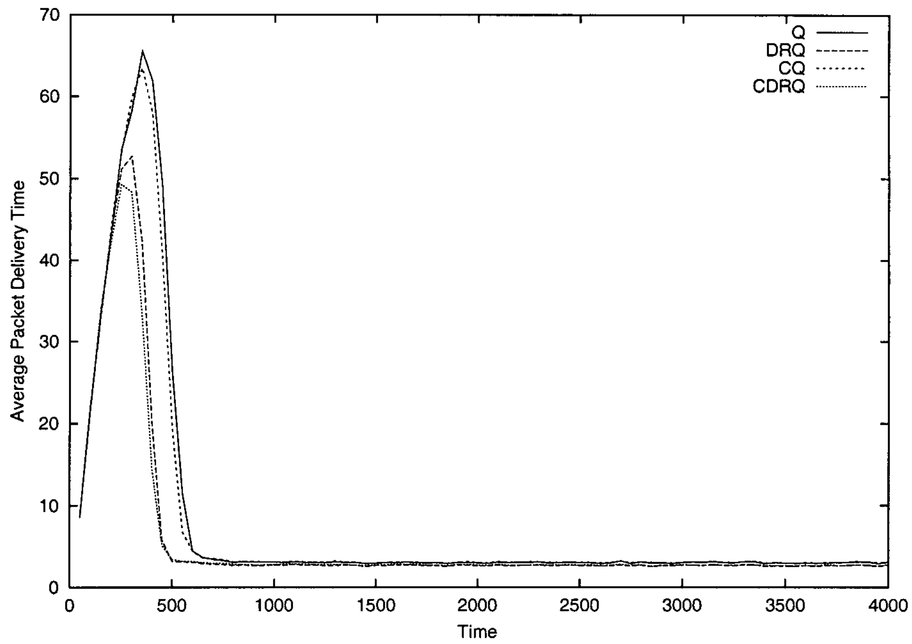


Figure 4.13: Comparing the average packet delivery times of Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing for network load 2.0 for the SDH network topology.

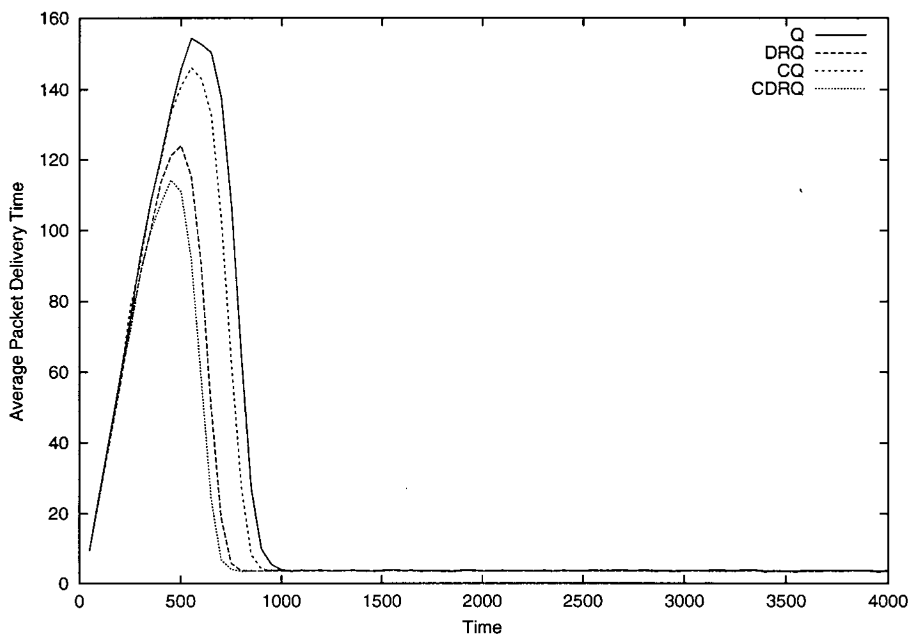


Figure 4.14: Comparing the average packet delivery times of Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing for network load 3.0 for the SDH network topology.

Stellenbosch University http://scholar.sun.ac.za



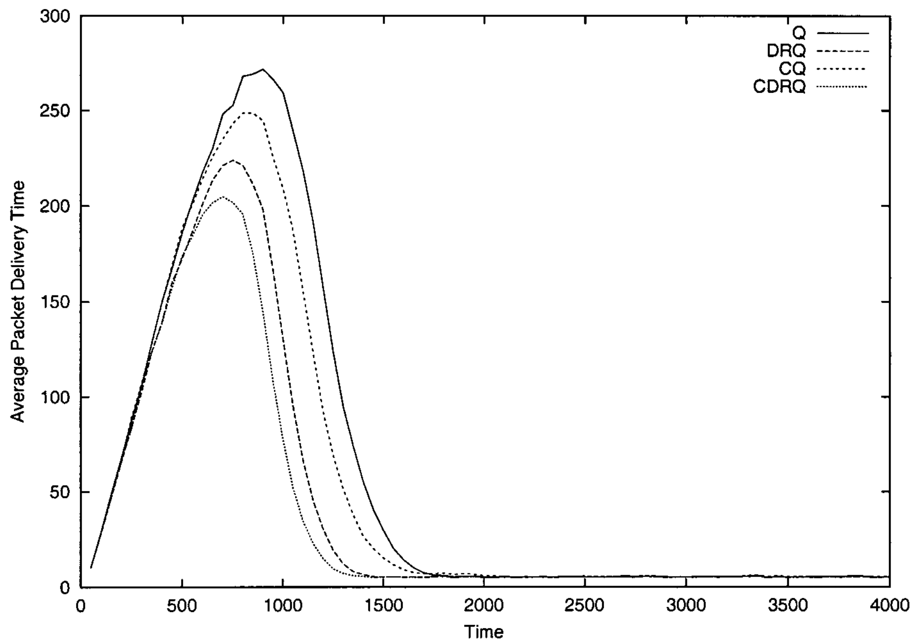


Figure 4.15: Comparing the average packet delivery times of Q-Routing, CQ-Routing, DRQ-Routing and CDRQ-Routing for network load 4.0 for the SDH network topology.

### 4.1.5 Probabilistic CDRQ-Routing

The confidence values used in CQ-Routing are used to decide by how much to update a Q-value; they do not influence the routing decisions directly. This section introduces an extension to CQ-Routing which uses the confidence values to make probabilistic routing decisions. The resulting algorithm is called Probabilistic Confidence-based Dual Reinforcement Q-Routing (Probabilistic CDRQ-Routing) proposed by Kumar [17].

When a node $x$ makes a routing decision with CDRQ-Routing, it selects the neighbour with the minimum Q-value from the vector $Q_x(*, d)$. Probabilistic CDRQ-Routing instead selects the neighbour $y$ with minimum randomly generated Q-value, $Q'_x(y, d)$. This randomly generated Q-value is taken from the Gaussian probability distribution with the mean equal to $Q_x(y, d)$, and the variance depending on some function of the confidence value $C_x(y, d)$.

The Gaussian probability distribution is chosen because the probability of generating a value near the mean Q-value is high for small variance, and is more spread out for higher variance. The variance function $v(C)$ must be chosen such that the variance is

Stellenbosch University http://scholar.sun.ac.za



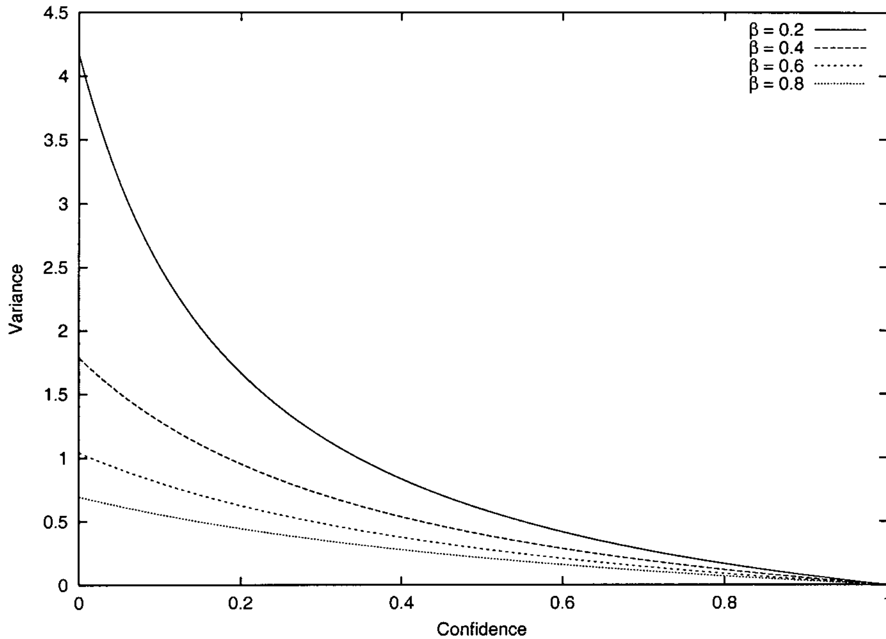


Figure 4.16: The variance function of Equation 36 for $\beta$ of 0.2, 0.4, 0.6 and 0.8.

0 when the confidence value is 1, resulting in a randomly generated value equal to the original Q-value. When the confidence value is low, the Q-value may be inaccurate; thus, we need to increase the variance, which leads to a large deviation of the randomly generated Q$'$-value.

Kumar suggested the function $v(C) = \frac{1}{C} - 1$ which gives infinite variance values for confidence values of 0. This problem can be remedied by adding an offset $\beta$ to the numerator, and a constant $k$:

$$v(C) = \frac{1}{C+\beta} + k \tag{35}$$

We can solve for the constant $k$ by noting that, for a confidence value equal to 1, we need the variance to be 0 i.e. $v(1) = 0$. This yields $k = \frac{-1}{1+\beta}$. Substitution of $k$ results in:

$$v(C) = \frac{1}{C+\beta} - \frac{1}{1+\beta} \tag{36}$$

Figure 4.16 shows the variance function of Equation 36 for different values of $\beta$.

The aim of Probabilistic CDRQ-Routing is to overcome the hysteresis effect [8; 19] of Q-Routing. This effect occurs when the network load is lowered after a period

Stellenbosch University http://scholar.sun.ac.za



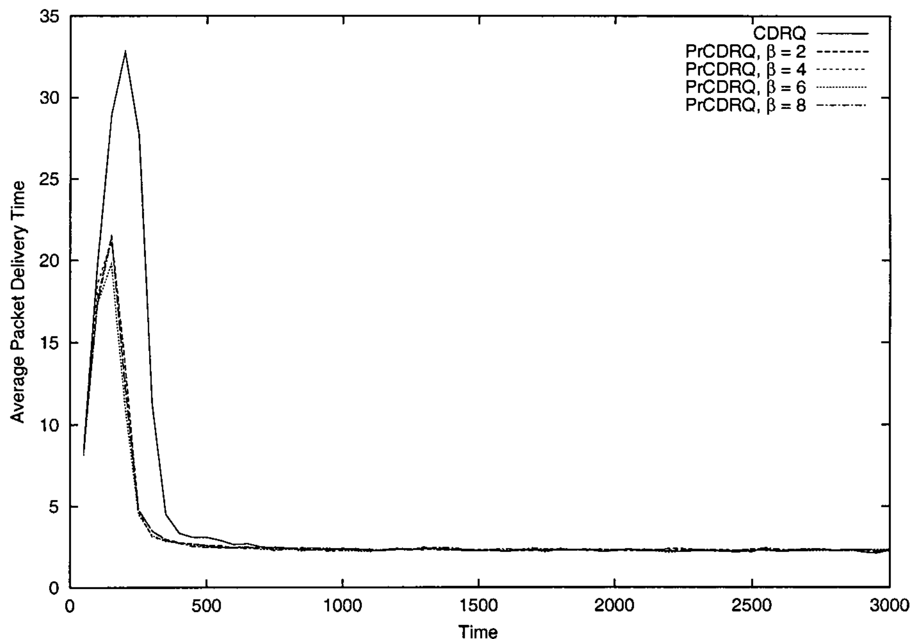


Figure 4.17: The Average Packet Delivery Time for the SDH network for network load 1.5; $\beta$ of 0.2, 0.4, 0.6 and 0.8.

of high load, but the routing policy fails to adapt to the new optimal path. This occurs because routes are selected based on *minimum* Q-values; thus, routes which previously have shown to result in high delivery times are not selected. Consequently, the Q-values of these routes are not updated to reflect the new network state. To overcome this problem, we need to periodically send packets over what appear to be suboptimal routes; i.e. we need to explore.

Exploration is encouraged by introducing randomness in Q-values which have not been updated in a while. As the confidence in a Q-value decreases over time, there is a non-zero probability that another route will be explored. There is an important trade-off between exploration and exploitation to be considered, as discussed in Section 3.7.

Simulations were run on the SDH network (Figure 4.1) for low (1.5), medium (3.0) and high (4.5) network load with $\beta$ equal to 0.2, 0.4, 0.6 and 0.8. All node buffers were of sufficient length that no packets were dropped during any simulation. The results show the average packet delivery time of 25 runs.

As can be seen from Figures 4.17 to 4.19, Probabilistic CDRQ-Routing learns a suitable routing policy faster than CDRQ-Routing, especially for low and medium network

Stellenbosch University http://scholar.sun.ac.za



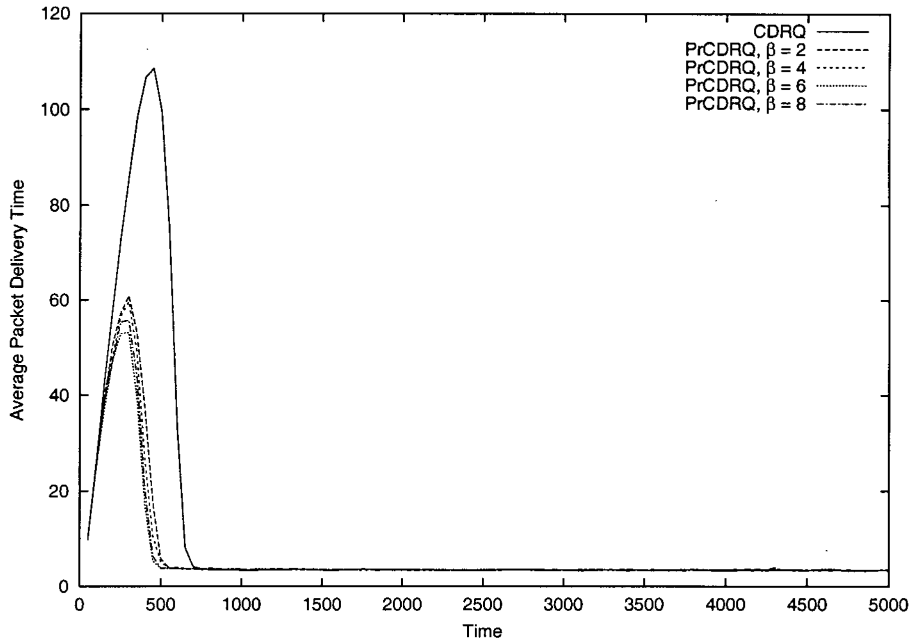


Figure 4.18: The Average Packet Delivery Time for the SDH network for network load 3.0; $\beta$ of 0.2, 0.4, 0.6 and 0.8.

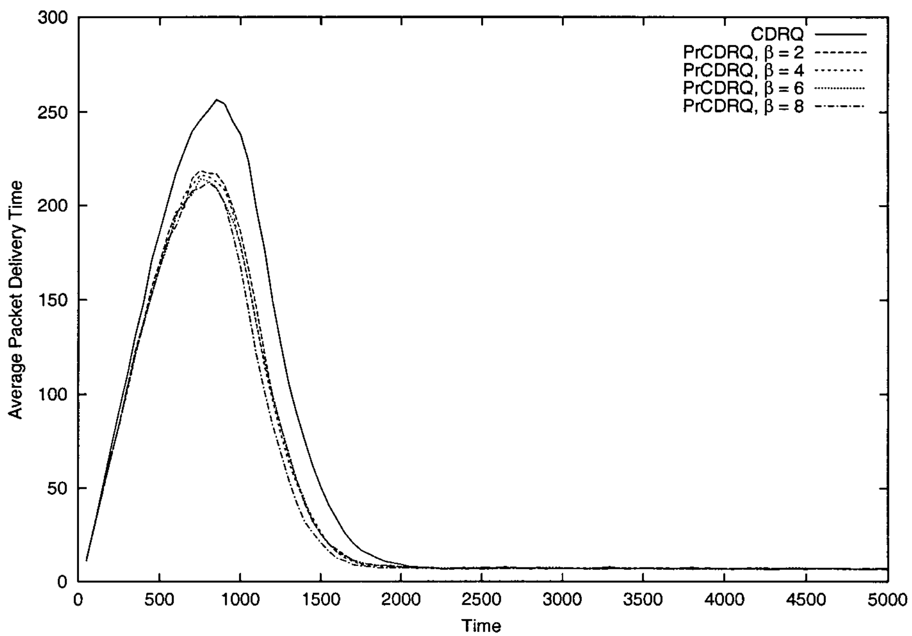


Figure 4.19: The Average Packet Delivery Time for the SDH network for network load 4.5; $\beta$ of 0.2, 0.4, 0.6 and 0.8.

Stellenbosch University http://scholar.sun.ac.za



loads. This shows that the extra exploration pays off in the training phase, without incurring a performance penalty after convergence. It can also be seen that the choice of the parameter $\beta$, which controls the maximum variance is not critical.

## 4.2 Finite Buffer Size

Realistic networks have buffers of finite size; thus, queue utilization at nodes needs to be considered in addition to packet delivery time. If buffers become full, congestion develops and packets may be dropped. This section explores a congestion control scheme which addresses this problem [4]. The routing algorithm now also needs to consider the number of packets waiting in queues; thus, we need to modify the equations for updating Q-values.

Let $B_x$ be the buffer size of node $x$ and let $q_x$ be its current queue length. We can define a congestion risk function $g(\frac{q_x}{B_x})$ that depends on the how full the queue is. The congestion risk is the risk of dropping a packet that is sent through node $x$. The following function for calculating the congestion risk has been proposed [18]:

$$g_\theta(\alpha) = \frac{\alpha}{(1-\alpha)^\theta}, \tag{37}$$

where $\theta$ controls the rate of growth of the congestion risk $g$.

One practical problem with Equation 37 is that it grows to infinity as $\alpha$ tends to 1. By utilizing a maximum congestion risk value, it becomes impossible to discriminate between different queue utilizations above the cutoff congestion risk value. The following smooth function has a maximum value of 1 at $\alpha = 1$:

$$g_\theta(\alpha) = \alpha^\theta, \tag{38}$$

where $\theta$ controls the rate of growth of the congestion risk $g$. Figure 4.20 shows the congestion risk for $\theta = \{3, 6, 15\}$.

We add a term to the routing update equation for calculating the new packet delivery time estimate which reflects the congestion risk of using a node:

$$Est(a,b) = Q_a(\hat{z}, b) + q_a + \delta + w \cdot g_\theta(\frac{A_\Lambda(q_a)}{B_a}), \tag{39}$$

where the congestion control scaling factor $w$ determines the extent to which the routing algorithm penalizes full queues. $A_\Lambda(q_a)$ is the average queue length of node $a$ over a

Stellenbosch University http://scholar.sun.ac.za



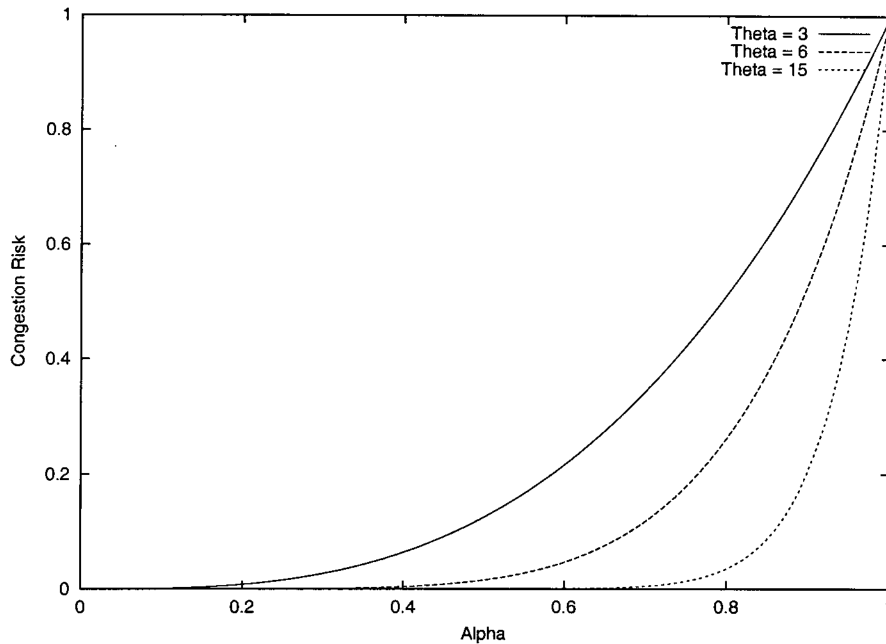


Figure 4.20: The Congestion Risk of Equation 38 for $\theta$ of 3, 6 and 15.

moving window of length $\Lambda$. Using the average queue length has the effect of delaying the congestion control by the window size parameter $\Lambda$, so that the algorithm does not react too quickly to bursty traffic.

For our experiments, we used the network topology shown in Figure 4.21. This network has node 8 on the shortest route between nodes 7,10,12 and 9,11,13, respectively; thus, we can expect congestion to develop at node 8. The queues at all nodes were of length 15. We generated network traffic uniformly across the network during the initial training phase. From time step 1500, we generated additional traffic between nodes 7 to 9, 10 to 11, and 12 to 13, respectively. Results are shown averaged over 25 simulation runs.

| Load | $w$ | $\theta$ | Window size ($\Lambda$) |
|---|---|---|---|
| 1.4 | 4 | 3 | 1 |
| 1.8 | 8 | 5 | 10 |
| 2.0 | 12 | 7 | 50 |
| | 16 | 9 | 100 |
| | 20 | 12 | |

Table 4.1: The parameters used in the simulations

Stellenbosch University http://scholar.sun.ac.za



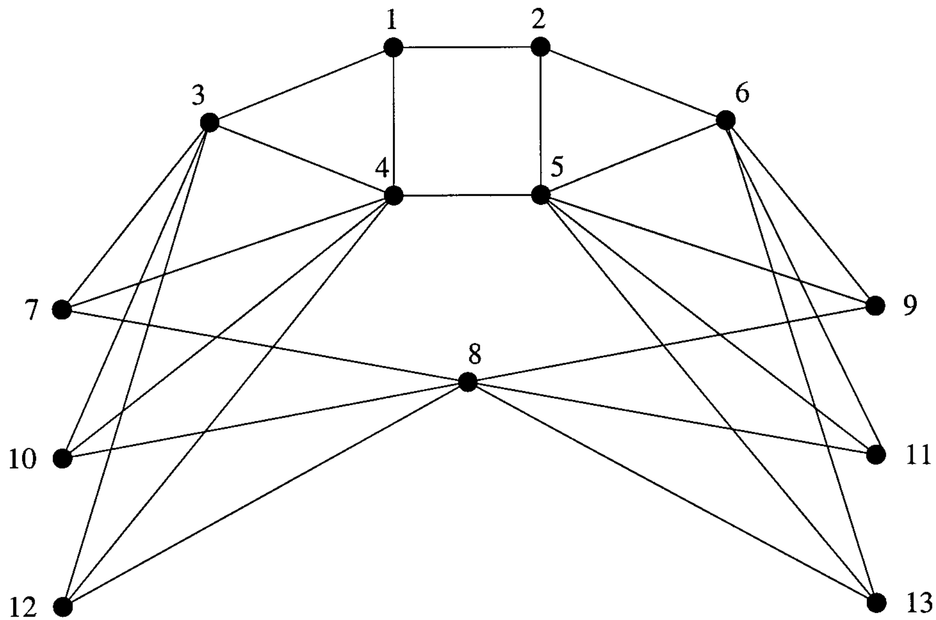


Figure 4.21: The 13 node network topology used for the finite buffer simulation.

An exhaustive search over all combinations of parameters shown in Table 4.1 was performed. The simulations with the lowest number of dropped packets for low (1.4), medium (1.8) and high (2.0) loads are shown in Figure 4.22 to Figure 4.27.

In all cases, the average packet delivery time of CDRQ-Routing with Congestion Control is initially higher, but not too different from normal CDRQ-Routing. In Figures 4.23, 4.25 and 4.27 we can see that considerably less packets are dropped with our congestion control scheme.

## 4.3 Optimization of Multiple Objectives

In realistic networks, costs such as use of a link and processing at nodes also need to be considered in addition to packet delivery time. Since our routing algorithm also needs to consider the cost of using a link, we need to change the meaning of Q-values [3]. Q-values are now the combination of the estimated delivery time and the estimated delivery cost.

The Q-value update rules are the same as for CDRQ-Routing of Section 4.1.4, but the calculation of $Est(a, b)$ of Equation 25 now includes an extra cost term. We propose

Stellenbosch University http://scholar.sun.ac.za



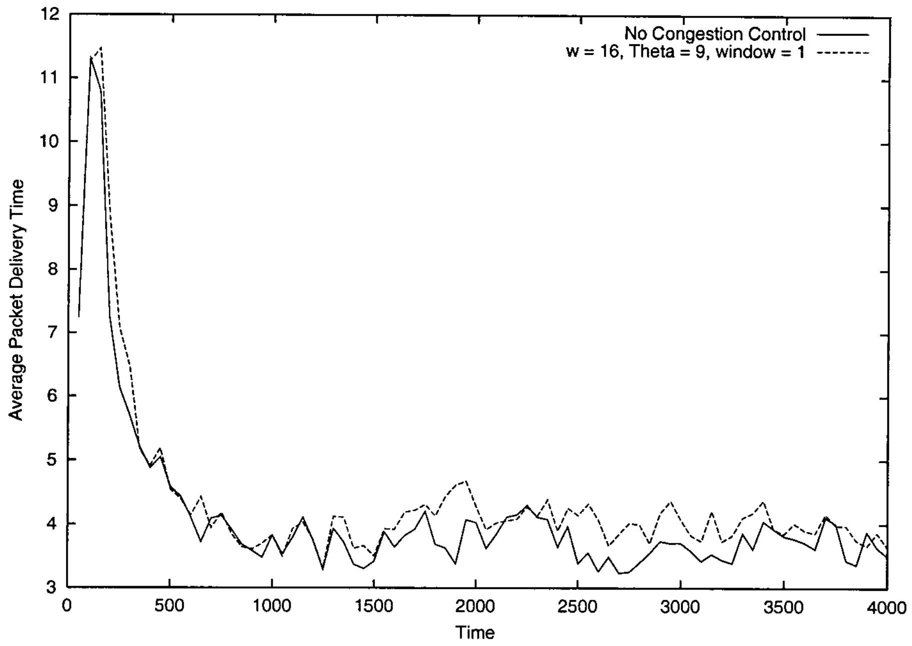


Figure 4.22: Average packet delivery time for low load.

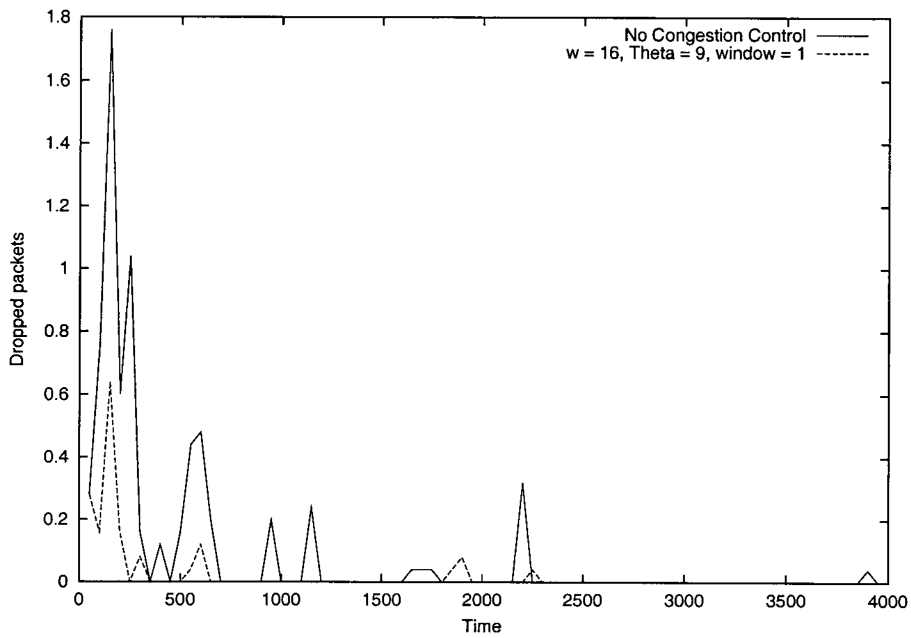


Figure 4.23: Number of packets dropped for low load.

Stellenbosch University http://scholar.sun.ac.za



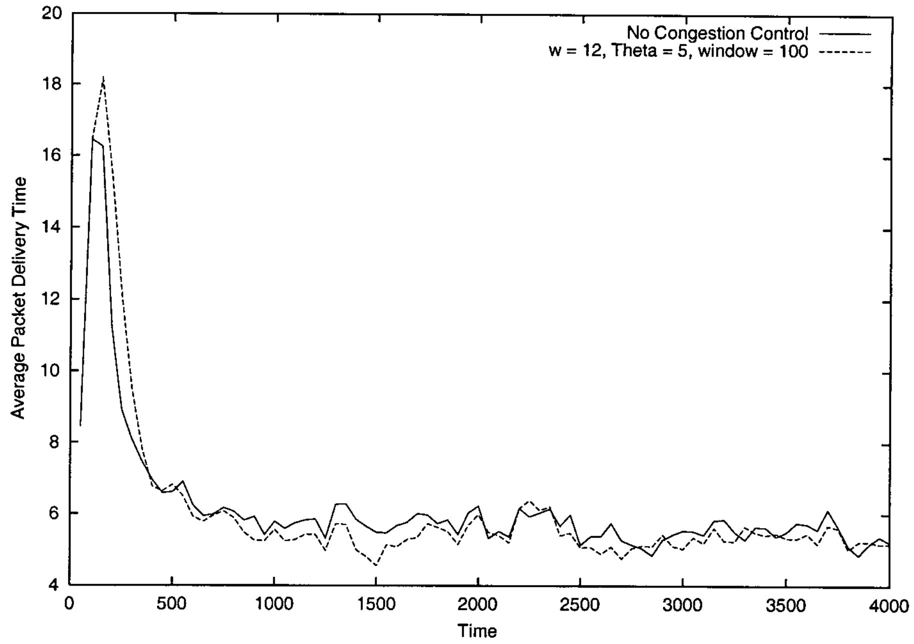


Figure 4.24: Average packet delivery time for medium load.

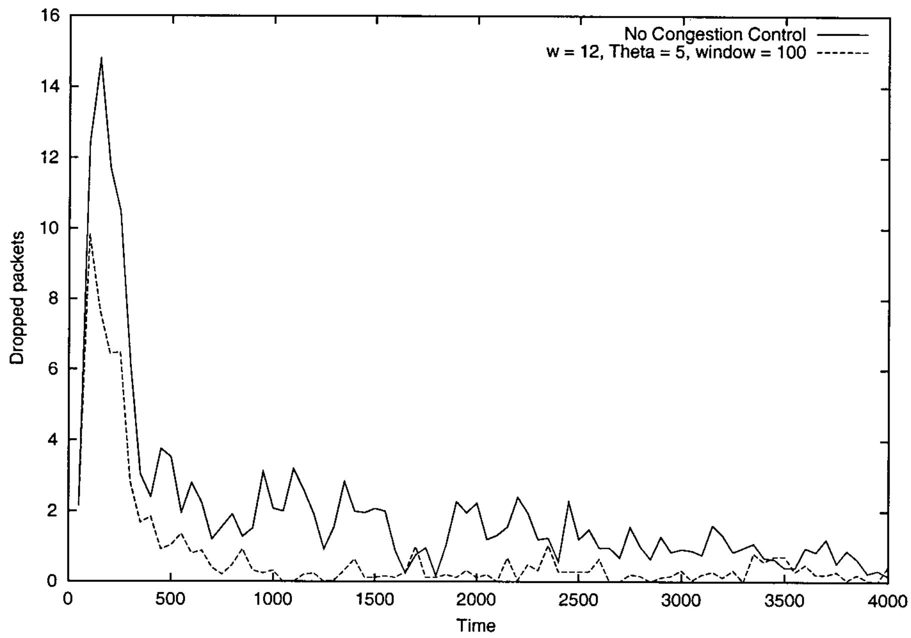


Figure 4.25: Number of packets dropped for medium load.

Stellenbosch University http://scholar.sun.ac.za



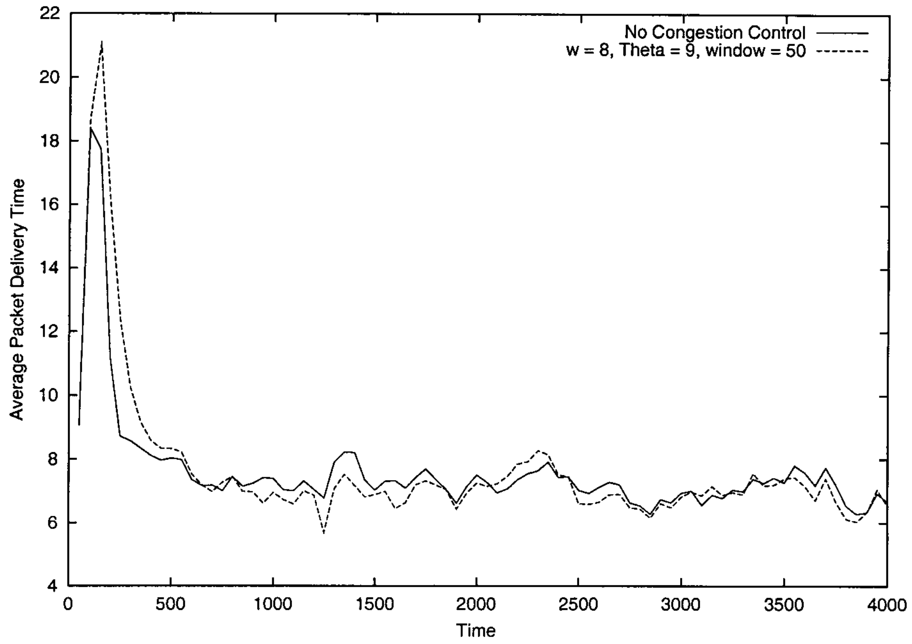


Figure 4.26: Average packet delivery time for high load.

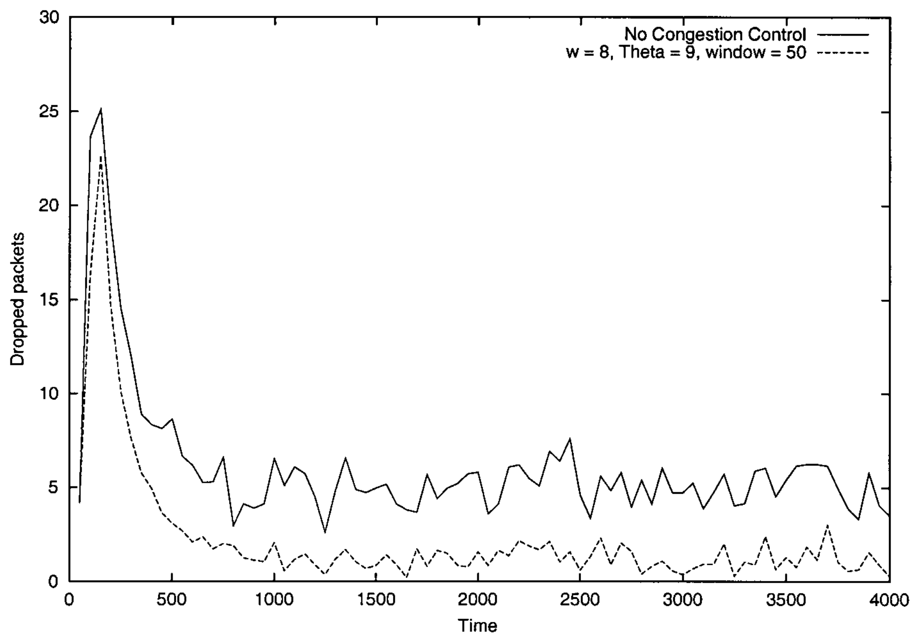


Figure 4.27: Number of packets dropped for high load.

Stellenbosch University http://scholar.sun.ac.za



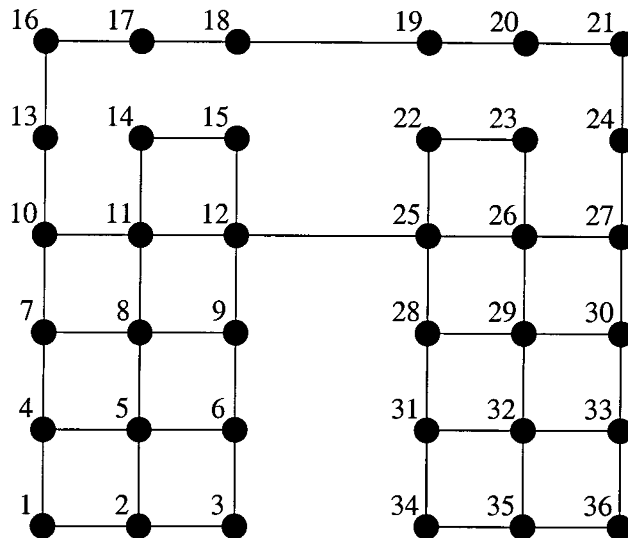


Figure 4.28: The network topology for the 36 node grid.

to update these estimates as follows:

$$Est(a, b) = Q_a(\hat{z}, b) + q_a + \delta + \alpha \cdot Cost(a, b), \tag{40}$$

where $Cost(a, b)$ is the cost of using the link between $a$ and $b$, and $\alpha$ is the factor determining how much to take the cost into consideration.

We conducted experiments on two networks: a 36 node grid due to Boyan and Littman [19] (Figure 4.28), and the 30 node British Telecom SDH network depicted in Figure 4.1. In all the experiments, the average packet delivery time of every 50 time steps was measured and averaged over 50 simulation runs. The queues at all nodes were of sufficient length so that no packets were lost in any simulation.

Link costs of one unit were assigned to all the links in the network in Figure 4.28, except between node 18 and 19, where the cost was 2, and between node 12 and 25 where the cost was 50 units. The load was fixed at 1.8 packets generated per time step, and the confidence decay factor was arbitrarily set to 0.9. The average delivery time is shown in Figure 4.29. When $\alpha = 0$ the cost is ignored, i.e. routing decisions are made based only on the single objective of minimizing packet delivery time.

We observe a linear increase in both the maximum delivery time, and the time to convergence with respect to $\alpha$. Details of the steady state behaviour is shown in Figure 4.30.

Stellenbosch University http://scholar.sun.ac.za



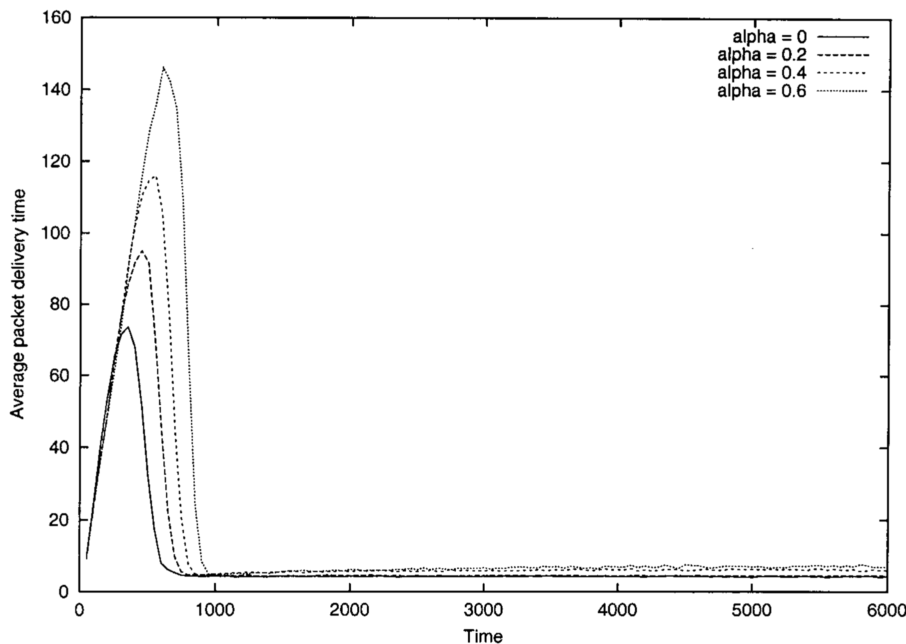


Figure 4.29: The average packet delivery time for single versus multiple objective optimization for the 36 node grid for differing $\alpha$.

Figure 4.31 shows the average cost for $\alpha$ between 0 and 0.6. Comparing Figures 4.30 and 4.31 reveals the trade-off between delay and cost. Between time steps 1000 and 4000, the average packet delivery times increase slightly for $\alpha = 0.4$ and $\alpha = 0.6$, but the delivery costs decrease.

This illustrates the trade-off that exists when attempting to simultaneously optimize competing objective functions. As can be seen, this network is very unbalanced and the shortest paths between most of the nodes between the two clusters are along the expensive link between node 12 and 25. The packets can either follow the short path and have a low delivery time, but high cost, or travel via the long route and increase the delivery time and decrease the cost. This explains the observed trade-off.

In the SDH network depicted in Figure 4.1, link costs were randomly assigned so that ten percent of the links were on average ten times as expensive as the remaining links. The network load was set at 3.0 packets generated per time step. Figures 4.32 through 4.34 show that the multiple objective optimization again has a longer learning period, but this time the average cost *and* the delivery times are less. Intuitively, this can be explained by observing that this network is more balanced, i.e. there are more alternative routes to choose from, while still avoiding costly links.

Stellenbosch University http://scholar.sun.ac.za



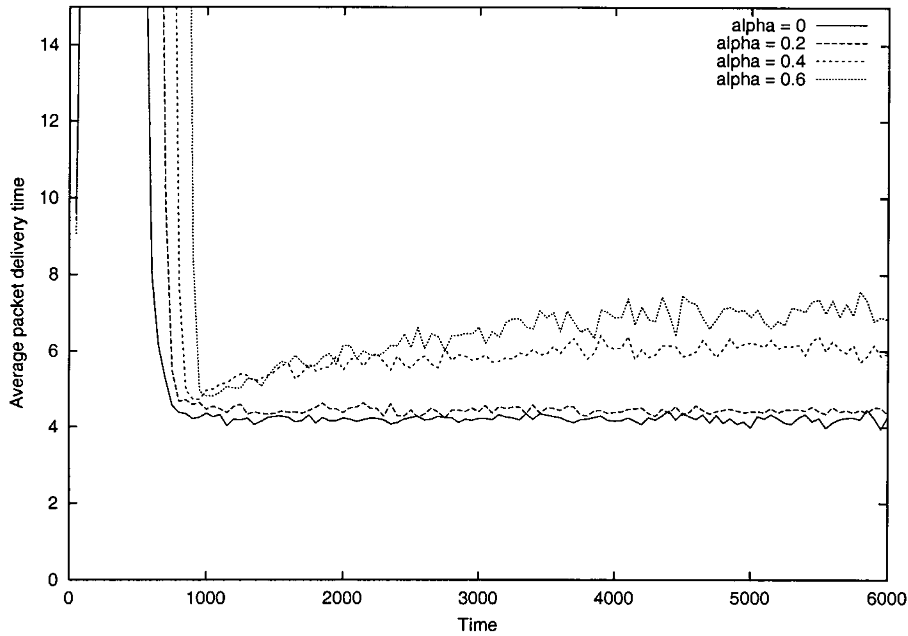


Figure 4.30: Details of the steady state behaviour of Figure 4.29.

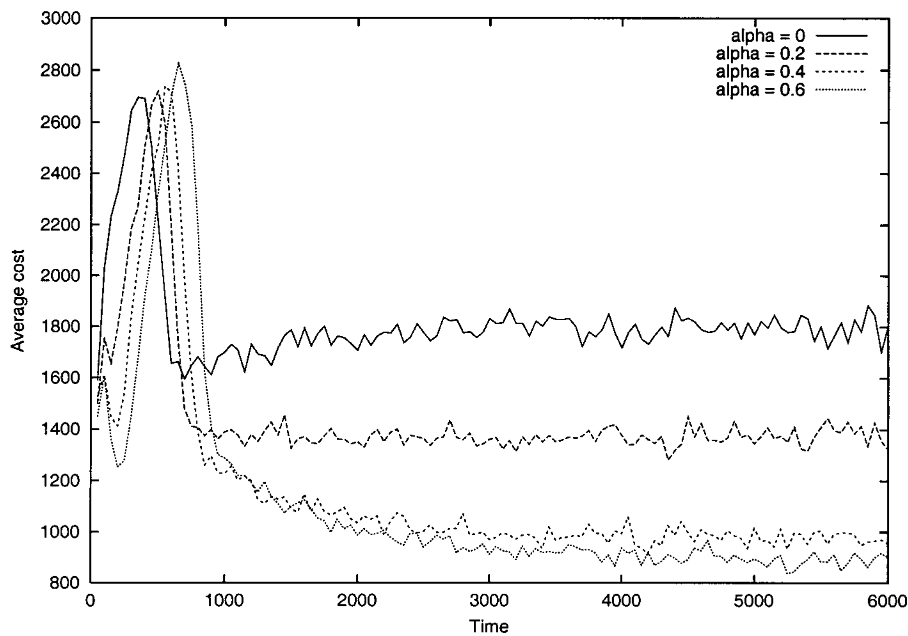


Figure 4.31: The average cost for single versus multiple objective optimization for the 36 node grid for differing $\alpha$.

Stellenbosch University http://scholar.sun.ac.za



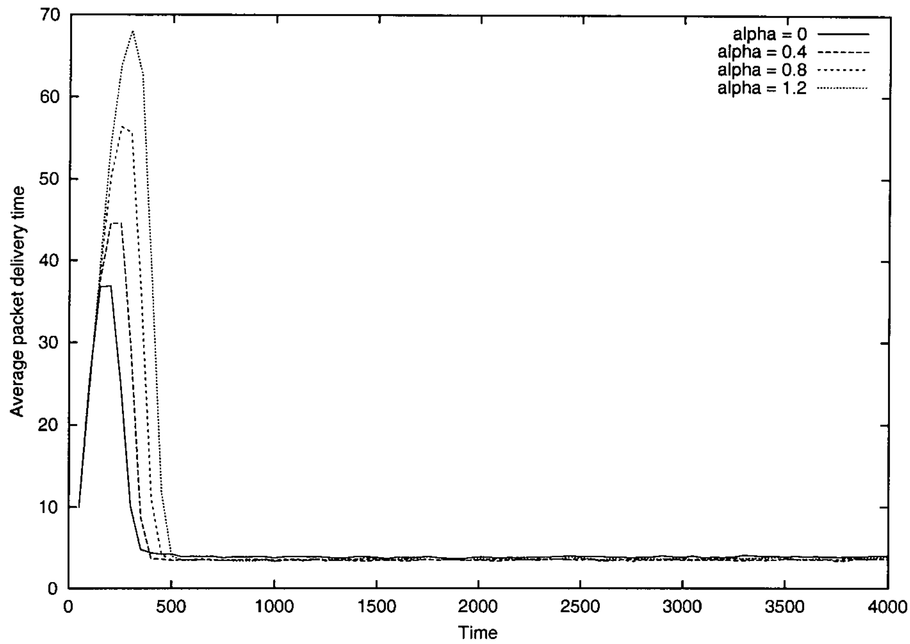


Figure 4.32: The average packet delivery time for single versus multiple objective optimization for the BT SDH network for differing $\alpha$.

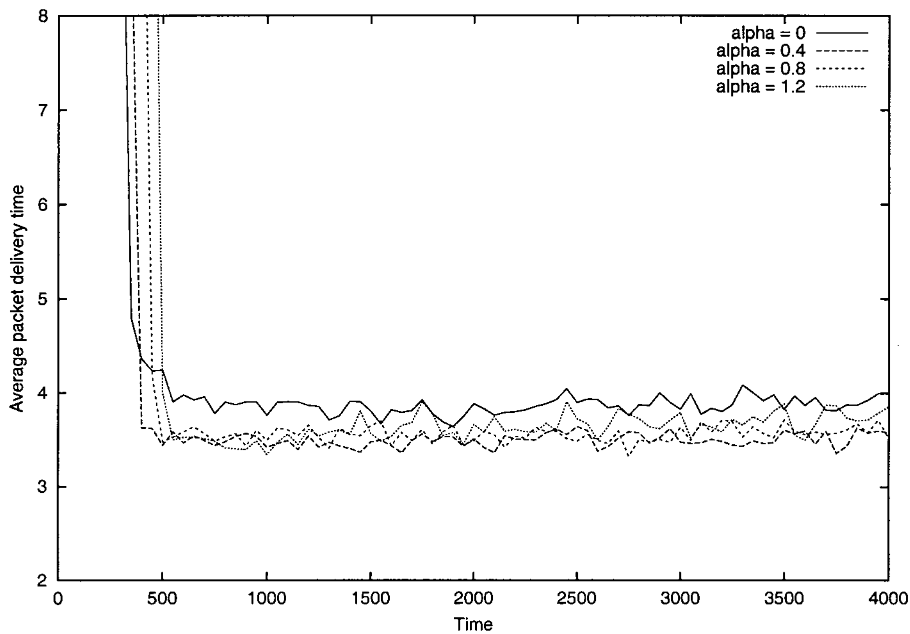


Figure 4.33: Details of the steady state behaviour of Figure 4.32.

Stellenbosch University http://scholar.sun.ac.za



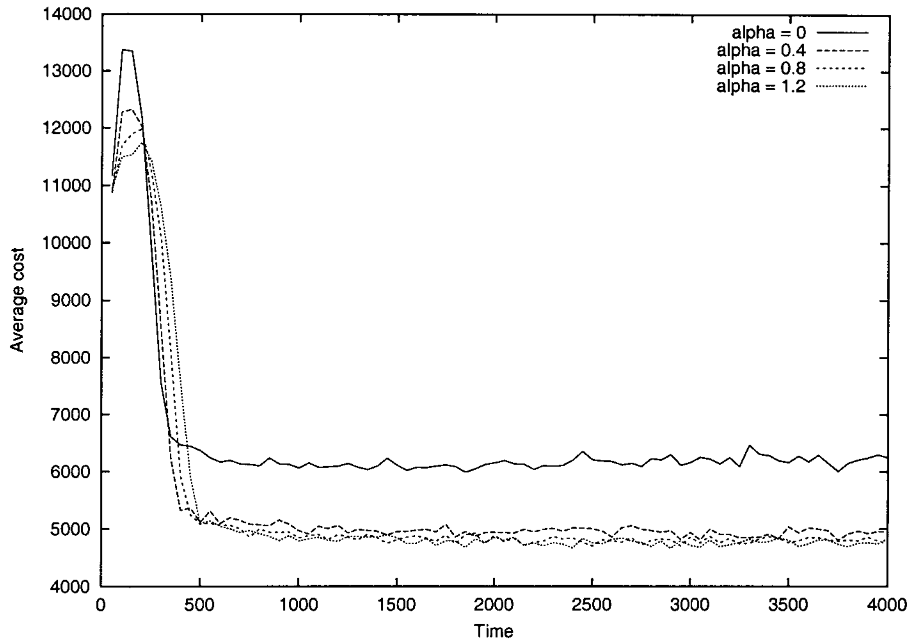


Figure 4.34: The average cost for single versus multiple objective optimization for the BT SDH network for differing $\alpha$.

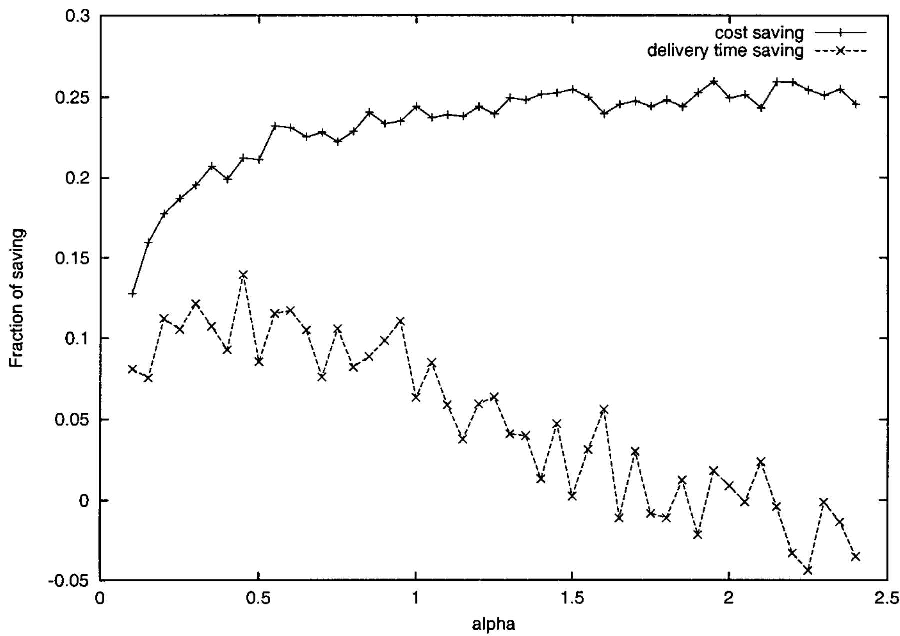


Figure 4.35: The average saving of multiple objective optimization of cost and delivery time for the BT SDH network versus $\alpha$.

Stellenbosch University http://scholar.sun.ac.za



Figure 4.35 shows the savings in terms of cost, and in terms of delivery time as a function of $\alpha$'s.

We have demonstrated that there is a trade-off between packet delivery time and average cost if links are priced differently. If there are alternative routes that are equally expensive in terms of number of hops, but unequal in cost, the new routing algorithm can save about 25 percent of the cost and 10 percent on the average packet delivery time.

## 4.4 Summary

In this chapter, we examined the distributed, adaptive traffic routing algorithm called Q-Routing. It is a distributed algorithm, as each node maintains a Q-table for estimating the average delivery time via its neighbours to all destinations. These delivery time estimates are incrementally updated based on local information of neighbouring nodes. Each node routes the packet to the neighbour with the minimum estimated delay.

Results showed that Q-Routing was able to route packets more efficiently at higher network loads than the static Shortest Path algorithm. We also found Q-Routing to be more stable than a straightforward implementation of the distributed Bellman-Ford algorithm, using average queue length as metric.

CDRQ-Routing is an enhancement over Q-Routing which uses dual direction exploration and confidence values in updating Q-value estimates. These enhancements resulted in faster learning speeds and improved routing performance.

Probabilistic CDRQ-Routing is a further enhancement to CDRQ-Routing which attempts to find new faster paths by increasing the amount of exploration, based on the confidence of a Q-value. This method proved to be promising as it converged faster than normal CDRQ-Routing during the training phase.

All the routing algorithms we discussed earlier assumed infinite packet buffers at all nodes. In realistic networks, this assumption no longer holds; thus, we considered the problem where nodes have finite buffers. We extended CDRQ-Routing to also consider queue utilization by using a congestion risk function. Our congestion control method dropped considerably less packets without increasing the average packet delivery time appreciably.

Stellenbosch University http://scholar.sun.ac.za



We also considered the case of minimizing the multiple objectives of packet delivery time and cost. Results showed that in general there is a trade-off between these two objectives.

Stellenbosch University http://scholar.sun.ac.za

# Chapter 5

# Conclusion

## 5.1 Conclusion

In this thesis, we investigated the use of reinforcement learning algorithms for routing packets in communication networks. Adaptive routing policies are needed in networks with unpredictable changes in traffic patterns and network topologies. We showed that Q-Routing, based on the Q-Learning algorithm, is a viable approach for distributed adaptive routing in dynamic networks. Moreover, this adaptive routing algorithm uses only local information in making routing decisions. This means less network resources are wasted in propagating global network state information across the network.

Simulation results suggest that the performance of Q-Routing is superior to that of the distributed Bellman-Ford routing algorithm. We found that Q-Routing converged to a more stable routing policy than Bellman-Ford at high traffic loads.

Confidence-based dual reinforcement Q-Routing (CDRQ-Routing) is an extension of Q-Routing which uses confidence values in updating Q-values in both the forward and backward direction. CDRQ-Routing converges much faster than Q-Routing with little extra overhead.

Probabilistic CDRQ-Routing is a further enhancement to CDRQ-Routing where routing decisions are made probabilistically. Taking exploratory actions improves the chances of finding new routes. The level of exploration is increased if the confidence in Q-values at a node are low. Making exploratory moves only when the confidence is low is important, as unnecessary exploration may degrade performance. Empirical results

Stellenbosch University http://scholar.sun.ac.za



show that probabilistic CDRQ-Routing finds good routing policies faster than normal CDRQ-Routing without wasting valuable network resources.

We also investigated two routing algorithms designed to improve performance in more realistic networks. Most network nodes have limited buffer capacity and as buffers on popular routes fill up, congestion develops. Our new routing algorithm is able to perform congestion control by rerouting traffic to less loaded regions of the network. The result is that our routing algorithm drops much fewer packets. The other enhanced routing algorithm is able to optimize the multiple objectives of minimizing packet delivery time and cost.

## 5.2 Future Work

We showed that the various Q-Routing algorithms are viable alternatives to standard distance-vector routing algorithms. To make an even stronger argument, work is needed in obtaining more realistic simulations and improving the routing algorithm performance. We will shortly discuss some of the ideas that might contribute to this work.

### 5.2.1 Realistic Simulations

More accurate simulation results may be obtained by experimenting with more realistic network and traffic models. In our investigation of the routing algorithms, all network nodes and links were identical. In a more realistic heterogeneous network, the processing speeds at nodes and link capacities can be different, and need to be incorporated into the model. This is one area where multiple objective optimization may be useful, since it is reasonable that fast nodes and links are more expensive.

The traffic model can also be improved to achieve more realistic simulations. Traffic must be generated with more realistic spatial and temporal distributions to evaluate routing performance under various dynamic load conditions. To illustrate the robustness, the routing performance also needs to be evaluated under different node and link failure scenarios. Another improvement will be to consider the use of different traffic classes with different requirements and priorities. This would allow the routing algorithm to drop low priority packets if congestion develops.

Stellenbosch University http://scholar.sun.ac.za



### 5.2.2 Improved Routing

The routing performance may be improved by a few additions to the routing algorithms. Encoding prior knowledge of a few good routes into the initial Q-values is one way of improving the convergence time of Q-Routing. This should result in less unnecessary exploration.

The recovery of Q-Routing after node or link failures can be improved by sending special routing packets from nodes near failed network components. These routing packets should have higher priority than data packets thus ensuring the fast propagation of the information of the failed nodes. This is likely to shorten the duration of the bouncing behaviour of packets between two nodes with inconsistent routing tables, by updating their routing tables to avoid the loop between them.

The routing policy of Q-Routing depends on the traffic distribution, i.e. it is data driven. The result is that areas of the network with very little traffic may not be explored sufficiently. Another idea is to generate independent routing packets which explore routes in the network and update routing tables, similar to the ants in AntNet [6]. Data packets are then routed according to the routing tables which the exploring routing packets continually update.

AntNet can be seen as a parallel replicated Monte Carlo system [6]. Barto and Sutton [30] argue that a first-visit Monte Carlo simulation system is equivalent to TD($\lambda$) with $\lambda = 1$. On the other hand, Q-Learning is a control method based on TD(0), i.e. $\lambda = 0$. It would be interesting to see how good the performance would be of an ant-like Q($\lambda$)-Routing algorithm with $0 < \lambda < 1$. This new algorithm could retain the best properties of both algorithms.

Q($\lambda$)-Routing could for example converge faster than Q-Routing, at the expense of extra routing overhead. Upon receipt of a packet at a destination, a routing packet containing the whole route taken, can be sent back to the origin node. The relevant routing table entry at each node along the way can then be updated. This record of the route taken can be seen as an eligibility trace. To limit the routing overhead, these special packets may only be sent periodically, or if the local network conditions changed.

Stellenbosch University http://scholar.sun.ac.za

# Bibliography


[1] C. Alaettinoglu and A. U. Shankar, "Stepwise assertional design of distance-vector routing algorithms", in *IFIP Protocol Specification Testing and Verification*, pp. 399–413, 1992.

[2] W. H. Andrag and C. W. Omlin, "Distributed intelligent multi-agents for telecommunication network management", in *Proceedings of the 2nd Annual South African Telecommunications, Networks and Applications Conference (Sept 1999), Durban, South Africa*, 1999.

[3] W. H. Andrag and C. W. Omlin, "Optimization of multiple objectives in telecommunication networks using intelligent agents", in *Proceedings of the 3rd Annual South African Telecommunications, Networks and Applications Conference (Sept 2000), Cape Town, South Africa*, 2000.

[4] W. H. Andrag and C. W. Omlin, "Q-Learning for adaptive congestion control in networks with finite buffer size", in *Proceedings of the 4th Annual South African Telecommunications, Networks and Applications Conference (Sept 2001), Wild Coast, South Africa*, 2001.

[5] A. G. Barto and S. P. Singh, "On the computational economics of reinforcement learning", in *Connectionist Models, Proceedings of the 1990 Summer School* (D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, eds.), pp. 35–44, Morgan Kaufmann, 1990.

[6] G. D. Caro and M. Dorigo, "Antnet: Distributed stigmergetic control for communications networks", *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.

Stellenbosch University http://scholar.sun.ac.za



[7] C. Cheng, R. Riley, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves, "A loop-free Bellman-Ford routing protocol without bouncing effect", *ACM SIGCOMM*, pp. 224–237, 1989.

[8] S. Choi and D. Yeung, "Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control", *Advances in Neural Information Processing Systems*, vol. 8, pp. 945–951, 1996.

[9] P. Dayan and T. Sejnowski, "TD($\lambda$) converges with probability 1", *Machine Learning*, vol. 14, pp. 295–301, 1994.

[10] L. Ford and D. Fulkerson, *Flows in Networks.* Prentice-Hall Inc., 1962.

[11] J. J. Garcia-Luna-Aceves and S. Murthy, "A path-finding algorithm for loop-free routing", *ACM Transactions on Networking*, vol. 5, pp. 148–160, 1997.

[12] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms", in *Advances in Neural Information Processing Systems* (J. D. Cowan, G. Tesauro, and J. Alspector, eds.), vol. 6, pp. 703–710, Morgan Kaufmann Publishers, Inc., 1994.

[13] L. P. Kaelbling and M. L. Littman., "Reinforcement learning: A survey", *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285., 1996.

[14] F. Kelly, "Modelling communication networks, present and future", in *Philosophical Transactions of the Royal Society*, pp. 437–463, 1996.

[15] A. Kerschenbaum, *Telecommunications Network Design Algorithms.* McGraw-Hill, 1993.

[16] K. H. Kramer, N. Minar, and P. Maes, "Tutorial: Mobile software agents for dynamic routing", *Mobile Computing and Communications Review*, vol. 3, no. 2, pp. 12–16, 1999.

[17] S. Kumar, "Confidence based dual reinforcement Q-routing: an on-line adaptive network routing algorithm", Master's thesis, University of Texas at Austin, 1998.

[18] S. Kumar and R. Miikkulainen, "Dual reinforcement Q-routing: An on-line adaptive routing algorithm", in *Proceedings of the Artificial Neural Networks in Engineering Conference*, (St. Louis, USA.), pp. 231–238, 1997.

Stellenbosch University http://scholar.sun.ac.za



[19] M. L. Littman and J. A. Boyan, "A distributed reinforcement learning scheme for network routing", in *Proceedings of the 1993 International Workshop on Applications of Neural Networks to Telecommunications.*, (Hillsdale NJ), pp. 45–51, Lawrence Erlbaum Associates, 1993.

[20] T. M. Mitchell, *Machine Learning.* McGraw-Hill, 1997.

[21] A. W. Moore, *Efficient Memory-based Learning for Robot Control.* PhD thesis, Trinity Hall, University of Cambridge, England, 1990.

[22] G. Necula, "Proof-carrying code", in *Proceedings of the 24th Annual ACM Symposium on Principles of Programming Languages*, pp. 106–119, ACM Press, 1997.

[23] J. Peng and R. J. Williams, "Incremental multi-step Q-learning", in *International Conference on Machine Learning*, pp. 226–232, 1994.

[24] J. Schmidhuber, "Adaptive confidence and adaptive curiosity", Technical Report FKI-149-91, Technische Universitat Munchen, Germany, 1991.

[25] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, "Ant-based load balancing in telecommunications networks", *Adaptive Behavior*, vol. 5, pp. 169–207, 1997.

[26] A. U. Shankar, C. Alaettinoglu, I. Matta, and K. Dussa-Zieger, "Performance comparison of routing protocols using MaRS: Distance-vector versus link-state", in *Proc. 1992 ACM SIGMETRICS and PERFORMANCE '92 Int'l. Conf. on Measurement and Modeling of Computer Systems*, (Newport, Rhode Island, USA), p. 181, 1-5 1992.

[27] W. Stallings, *Data and Computer Communications.* Prentice-Hall International, 5 ed., 1997.

[28] R. Sutton, "Learning to predict by the methods of temporal differences", *Machine Learning*, vol. 3, pp. 9–44, 1988.

[29] R. S. Sutton, "Integrated modeling and control based on reinforcement learning and dynamic programming", in *Advances in Neural Information Processing Systems* (R. P. Lippmann, J. E. Moody, and D. S. Touretzky, eds.), vol. 3, pp. 471–478, Morgan Kaufmann Publishers, Inc., 1991.

Stellenbosch University http://scholar.sun.ac.za



[30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[31] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A Survey of Active Network Research", *IEEE Communications Magazine*, vol. 35, pp. 80–86, 1997.

[32] S. B. Thrun, "The role of exploration in learning control with neural networks", in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches* (D. A. White and D. A. Sofge, eds.), (Florence, Kentucky), Van Nostrand Reinhold, 1992.

[33] C. Watkins, *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.

[34] C. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, pp. 279–292, 1992.