Application of ant based routing and intelligent control to telecommunications network management

David Nathanial Legge

Application of

Ant Based Routing

and

Intelligent Control

to

Telecommunications Network Management

David Nathanial Legge

School of Engineering

University of Durham

June 2006

A thesis submitted for the degree of

Doctor of Philosophy (PhD)

of the

University of Durham

David N. Legge

Application of Ant Based Routing and Intelligent Control to
Telecommunications Network Management

PhD 2004

Abstract

This thesis investigates the use of novel Artificial Intelligence techniques to improve the control of telecommunications networks. The approaches include the use of Ant-Based Routing and software Agents to encapsulate learning mechanisms to improve the performance of the Ant-System and a highly modular approach to network-node configuration and management into which this routing system can be incorporated.

The management system uses intelligent Agents distributed across the nodes of the network to automate the process of network configuration. This is important in the context of increasingly complex network management, which will be accentuated with the introduction of IPv6 and QoS-aware hardware.

The proposed novel solution allows an Agent, with a Neural Network based Q-Learning capability, to adapt the response speed of the Ant-System - increasing it to counteract congestion, but reducing it to improve stability otherwise. It has the ability to adapt its strategy and learn new ones for different network topologies. The solution has been shown to improve the performance of the Ant-System, as well as outperform a simple non-learning strategy which was not able to adapt to different networks.

This approach has a wide region of applicability to such areas as road-traffic management, and more generally, positioning of learning techniques into complex domains. Both Agent architectures are Subsumption style, blending short-term responses with longer term goal-driven behaviour. It is predicted that this will be an important approach for the application of AI, as it allows modular design of systems in a similar fashion to the frameworks developed for interoperability of telecommunications systems.

Declaration

I hereby declare that this thesis is a record of work undertaken by myself, that it has not been the subject of any previous application for a degree, and that all sources of information have been duly acknowledged.

Acknowledgements

I would firstly like to thank Peter Baxendale for his help and support supervising me over the last few years, and giving me a nudge when I most needed it and Phil Mars for setting me on the road in the first place.

I would also like to thank the many friends in Durham I have made, all the Andys and Daves, Richard and *the* Paul, amongst many others, for their time at local hostelries and Tony McFarlane for his company in the lab. Further afield my gratitude goes to Spiros, Malcolm, Daniel and Eduardo, (mostly of York) for their occasional shots of inspiration.

My immediate family have been behind me all the way, along with my Aunt and Uncle for looking after me while I was waiting for my house - and countless Sunday lunches since.

Last but not least, my thanks go to Angela for supporting me, and especially for taking me to Thailand, where *anywhere* you stop to sit, you can watch some industrious insects finding their way around *en masse*.....

Contents

Table of Contents

Table of Equations and Figures

Nomenclature

α	Learning Rate
γ	Future Reward Discount Rate
δ	Error Signal
δ_h	Hidden Layer Error Signal
δ_o	Output Layer Error Signal
δ_{ok}	Output Node Error Signal
Δw	Change in Neural Network Synapse Weight
η	Network Efficiency
λ	Eligibility Trace Discount Rate
τ	Volume of Control Traffic per Node
μ	Momentum Term
a	Action
d	Desired Output
k	Index of a node layer
K	Number of nodes in layer
o	Output Signal from Neural Network
o_k	Output Signal from Particular Node
p	Probability
Q(s , a)	State-Action Value Function
r	Reward Value
s	State
s_1	Initial State
s_t	State in time t
t	Time
V(s)	State Value Function
w_{ab}	Neural Network Synapse Weight between Nodes a and b

AAL	ATM Adaption Layer
AAMAS	Adaptive Agents and Multi-Agent Systems
ABC	Ant-Based Control
ABR	Available Bit Rate
ACL	Agent Communication Language
ADSL	Asynchronous Digital Subscriber Line
AI	Artificial Intelligence
AISB	Artificial Intelligence and Simulation of Behaviour
ANN	Artificial Neural Network
ANS	Agent Name Server
AOP	Agent Oriented Programming
ARPA	Advanced Research Project Agency
AS	Autonomous System
AT&T	AT&T Corporation
ATM	Asynchronous Transfer Mode
BDI	Belief Desire Intention
BT	British Telecommunications
CAC	Connection Admission Control
CBR	Constant Bit Rate
CBQ	Class Based Queuing
CLP	Cell Loss Priority
CNFM	Co-operative Network-Fault Management
CORBA	Common Object Request Broker Architecture
CoS	Class of Service
DiffServ	Differentiated Services
DHCP	Dynamic Host Configuration Protocol
DR	Deposition Rate
DRR	Deficit Round Robin
DP	Dynamic Programming
DS	Differentiated Services
DVA	Distance Vector Algorithm
EBCI	Explicit Backward Congestion Indicator

EFCI	Explicit Forward Congestion Indicator
EGP	Exterior Gateway Protocol
EX-OR	Exclusive-OR
FDDI	Fibre Distributed Data Interface
FIFO	First In First Out
FIPA	Foundation for Intelligent Physical Agents
FTP	File Transfer Protocol
GFR	Guaranteed Frame Rate
GGP	Gateway-to-Gateway Protocol
GUI	Graphical User Interface
HTTP	Hyper-Text Transfer Protocol
IAB	Internet Advisory Board
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IGRP	Interior Gateway Routing Protocol
IntServ	Integrated Services
IP	Internet Protocol
IPL	Initial Pheromone Level
IPng	Internet Protocol New Generation
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISCA	Inter-Society Communication Agent
ISO	International Standards Organisation
ISP	Internet Service Provider
IS-IS	Intermediate System to Intermediate System
JOONE	Java Object Oriented Network Engine
JVM	Java Virtual Machine
LAN	Local Area Network
LSA	Link State Algorithm
LUT	Look-Up Table
Mbps	Mega-bits per second
MC	Monte Carlo

MIB	Management Information Base
MPLS	Multi-Protocol Label Switching
MSE	Mean Squared Error
nam	Network Animator
nRT	non-Real Time
OMG	Object Management Group
OOP	Object Oriented Programming
OSI	Open System Integration
OSPF	Open Shortest Path First
NN	Neural Network
ns	Network Simulator
PC	Personal Computer
PCR	Peak Cell Rate
PSTN	Public Switched Telephone Network
PVC	Permanent Virtual Circuit or Connection
QL	Q-Learning
QoS	Quality of Service
RFC	Request For Comment (IETF Document)
RIP	Routing Information Protocol
RL	Reinforcement Learning
RMON	Remote-MONitoring
RMSE	Root Mean Squared Error
RSVP	Resource reSerVation Protocol
SD	Standard Deviation
SNMP	Simple Network Management Protocol
SoR	Sphere of Responsibility
SPF	Shortest Path First
SVC	Switched Virtual Circuit or Connection
TCL	Tool Control Language
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol Suite
TD	Temporal Difference

TDL	Temporal Difference Learning
ToS	Type of Service
TS	Traffic Shaping
TTL	Time To Live
UBR	Unspecified Bit Rate
UDP	User Datagram Protocol
VBR-nrt	Variable Bit Rate - Non-Real-Time
VBR-rt	Variable Bit Rate - Real Time
VP	Virtual Path
WFQ	Weighted Fair Queuing
X-OR	Exclusive-OR

Chapter 1: Introduction

This thesis investigates novel ways to automate and improve the process of controlling a telecommunications network. This is important because an automatic response to a problem could provide a quicker solution than getting a service engineer out of bed, driving 50 miles to the customer's site and then start investigating the problem. This also involves the diagnosis process, which by its nature often involves trial and error before the actual problem is found and is thus a lengthy process.

There are other reasons, some anecdotal perhaps, why the process of computer systems maintenance as a whole must become more automated. The new version of the Internet Protocol, IPv6, uses 128-bits for its address space, as opposed to 32-bits. This turns IP addresses from being about as humanly-memorable as a telephone number, to not-at-all memorable. Routing tables could be typed in by the network administrator when relatively few hosts existed. To do so now would be tedious work, prone to errors; a single one of which can prevent a network working properly and be virtually untraceable.

Another departure from convention is attributed to the move to distributed systems; this is a natural progression, since computer systems were originally expensive and monolithic, but are now a lot cheaper, portable, and the processing power is more easily spread geographically. Distributed systems are more resilient – which was one of the motivating factors behind the origins of the Internet. If a single point is affected by congestion, or power-failure, and that point is the centralised control point, then the whole network suffers; if however the control systems are spread, and the same happens, there will be comparatively few problems.

There are also *speed of response* advantages to distributed control – information does not have to be assembled at a central point before a decision can be made. For large geographically spread networks this can add significant propagation delays to the control scheme, and greatly reduce its effectiveness.

Automation of configuration provides great advantages now that the PC (Personal Computer) has become a fixture in every household. Just ten years ago, relatively few households were connected to the Internet, whereas now the exception is the house that isn't. Whereas before, networking would have been exclusively the domain of people that had a good working knowledge of their machine, now it has become the domain of the mass-market, where the owner doesn't want to know *how* it works, just that it *does*. This has meant that every function must be automated or at least the subject of a Graphical User Interface (GUI), to ease the user through the process.

The area of Artificial Intelligence, AI, is a natural place to start in searching for new techniques to apply to the area of network control. The aim of traditional AI (referred to as strict-AI within this thesis for clarity) is to mimic human behaviour - the formal test described for strict-AI is the Turing Test[1] [Turing 1954] - and this is often seen as a end in itself. However, in the quest for this goal there are many techniques that can be, and have been, applied to other areas. This situation is also echoed in the more broadly-defined AI; such as Nature Inspired solutions as in Swarm Intelligence and the technique explored later in this thesis - Ant-Based Routing.

Many ideas have been borrowed from nature - both human-based (e.g. Neural Networks) and more widely based (e.g. aeroplanes) - and the most successful of these strike a balance between mimicking the initially inspiring behaviour and adapting where the inspiration becomes impracticable; any model or analogy is only accurate to a certain degree - notice that no modern planes have flapping wings. This Engineer's pragmatic view can cause friction with parties who wish to stay pure to the inspiration.

1The Turing Test is passed if a human is unable to distinguish between a human and a computer's responses in a two-way conversation (conducted in a similar way to a "chatroom" on the web).

A strength of techniques drawn from AI is that a significant effort is spent working on improving adaptability - an area in which humans are very strong. From this work has come the search for systems that can learn from experience, and find the best action for different situations.

Conversely, it is also important to realise that often the cost of genericism is complexity in implementation – there is a reason that the analysis of many systems, Newton's Laws for example, start with many assumptions (ignoring friction for example) and once a satisfactory model has been built these assumptions can then be incorporated into a more accurate model. This is true everywhere except when a one-size-fits-all approach is suitable anyway.

1.1 Aim of this Thesis

The aim of this thesis is to describe the current state of the art in conventional Network Management and Control, compare that with alternative and experimental approaches - especially those attempting to control networks in an intelligent fashion - and then develop new and novel techniques to extend and improve on those described.

Areas that are of significant interest are:

- simplifying configuration of networks, since traditionally the process has been prone to human error;
- making the network more resilient by automatic handling of faults and failures;
- automating network management and control tasks, to allow autonomous operation and aid a speedy recovery, whilst also optimising the use of network resources.

1.2 Overview of this Thesis

This section provides a brief description of the thesis:

Chapter Two lays out the conventional methods of Network Management, having defined what that entails. The *de facto* standards are described, mainly the TCP/IP suite of protocols developed by the IETF, which include transport, network, routing, management and configuration protocols. Also illustrated are rival efforts, such as the OSI structure and ATM.

This is contrasted with the state-of-the-art review in Chapter Three, which aims to outline the research efforts to apply elements of Artificial Intelligence, and related techniques, to the area of Network Management and Control. To begin with the concept of a software Agent is introduced, as this is a common theme running through much of the research described. Different themes are then developed, using the fact that many AI research areas map quite naturally to applications in Telecommunications.

Chapter Four describes a novel research effort to build a highly modular, distributed, system to implement Network Management using Agents. The system is implemented on PCs and the Agents control the network switches *by proxy* across the network. The Agents form societies both within each switch and across the network, taking on some interesting aspects. The chapter ends with a discussion on the potential for this system to be extended.

A full investigation into the implementation of Ant-Based Routing is undertaken in Chapter Five. This includes analysing the underlying mechanisms behind the algorithms, as well as research into the correct parameter settings required for optimum performance. The system is tested on two simple scenarios, each representing two generic situations.

Chapter Six describes the novel approach of using an Agent to manipulate the

parameters controlling the Ant-Based Routing System strategically, to improve its response to changing network conditions. This includes; describing the theory behind the technique used - using a Neural Network to implement Q-Learning, describing the testing regime and quantifying the resulting improvement. The scheme is then tested on the situations described in the previous chapter, as well as a larger, more complex network, which again represents a generic network architecture.

The penultimate chapter, Chapter Seven, describes various avenues for further work, including direct extensions to the research described, other applications for the techniques described as well as alternative approaches to the work presented.

Finally, the thesis is concluded with Chapter Eight, which draws together the work with some final observations and comments.

Chapter 2: Network Management and Control

This chapter is an introduction and discussion of traditional network control and management methods and techniques. First of all, however a brief introduction to the structures used to build networks is given, as this is essential for the basis of the rest of the chapter.

2.1 Network Structures

There are numerous different media, over which data can be transported; these include simple telephone line (copper-wire pairs), co-axial cable, optical-fibre (multi-mode and single-mode), not to mention wireless technology - microwave links and radio waves. Each medium will require its own management, but which medium is used should be, and is, irrelevant to the particular specific application accessed by the user.

To achieve this - so that the entire communication system does not need rewriting for every small technology change, or indeed so that the user can switch applications while retaining connectivity - at least one layer of abstraction is necessary from the physical interface to the user-application; and three or four abstractions would break the task down to significantly more manageable tasks.

Protocols are usually designed to fill a single specific level of the hierarchy - although sometimes they combine two designated layers, or split a single designated layer in two. In theory the interfaces between the levels of the structure are well-defined, so that any protocol can be used with any other.

As already alluded to, the Transport Control Protocol (TCP) is one such protocol that fits into a series of abstractions. TCP is a major part of the TCP/IP suite, the *de facto* standard in the Internet, and is shown in Figure 2.1. The TCP/IP suite as a whole does not conform to the ISO-OSI model and is simpler (i.e. it has less layers) but they are related, having been conceived at around the same time.

Whilst the OSI model was being produced by the inter-governmental body the ITU-T (International Telecommunications Union - Telecommunications Standardisation Sector), the TCP/IP suite was developed and deployed by the Internet Engineering Task Force, IETF, and its management body the Internet Advisory Board, IAB.

The TCP/IP suite consists of the Link, Internet, Transport and Application Layers. The Link Layer deals with the physical interface, so is medium-dependant, but the other layers are abstracted from this, so can ignore the specific details.

<table>
<tr><th>OSI Model</th><th>IP/TCP "Model"</th><th colspan="2">Example Protocols</th></tr>
<tr><td>Application Layer</td><td rowspan="3">Application Layer</td><td colspan="2" rowspan="3">Telnet, FTP, HTTP</td></tr>
<tr><td>Presentation Layer</td></tr>
<tr><td>Session Layer</td></tr>
<tr><td>Transport Layer</td><td>Transport Layer</td><td>TCP</td><td>UDP</td></tr>
<tr><td>Network Layer</td><td>Internet Layer</td><td colspan="2">IP</td></tr>
<tr><td rowspan="2">Link Layer</td><td rowspan="3">Physical Layer</td><td rowspan="2">Ethernet</td><td>AAL</td></tr>
<tr><td>ATM</td></tr>
<tr><td>Physical Layer</td><td colspan="2">MAC</td></tr>
</table>

Figure 2.1 Diagram to Compare OSI model and TCP/IP models

The Network Layer is occupied by the Internet Protocol (IP), which is currently version 4 (IPv4), although the transition to version 6 (IPv6) has begun. The Transport layer sits, diagrammatically, on top of the Network Layer - and is occupied by TCP, or its lighter-weight brother UDP (User Datagram Protocol). The Application Layer sits above the Transport Layer and is where the user's programs reside - typical applications being

Telnet, FTP (File Transfer Protocol) and HTTP (Hyper-Text Transfer Protocol) for web-traffic - which use TCP and UDP.

The OSI framework, also shown in Figure 2.1 breaks down into more layers and is the traditional "7-layer model"; with three layers instead of the single Application layer of the TCP/IP model, and two layers covering the layer occupied by IP/TCP Link Layer. The higher layers (Those equivalent to the TCP application layer) are not considered further in this thesis. A full discussion of the OSI model is beyond the scope of this thesis, but details can be found in any standard telecommunications text, e.g. [Walrand 1999].

Both "models" provide defined interfaces and levels of abstraction to enable communication protocols to be developed which are interoperable with generic higher and lower layers; effectively introducing platform independence. This is done by tightly defining what interactions can occur between consecutive layers - neighbouring layers being the only ones that can inter-communicate.

The OSI model was overtaken by TCP/IP in terms of volume of deployment, which is the incumbent technology fuelling the rise of the Internet. An installed base of that magnitude is unlikely to be shifted to anything else (which is partly the current problem with moving to Ipv6); however, it does outline a framework that is helpful to consider, and provide a common language for discussion.

The layers most relevant to this thesis are the Transport and the Internet layer. These two layers are both shielded from the fine details of the actual physical media used, but also from the details of the operating system and application as well.

The main functions of the protocols in these two layers are:

- TCP - provides end-to-end connectivity, by checking that all packets are received in good time, and by taking action if not; then reassembling the

packets in the correct order, which can then be passed to the relevant application being run by the user.

- IP - to provide an end-point for packets destined for the local machine, and an intermediate relay for other packets, which are to be forwarded to the most suitable next-hop.

From these descriptions it can be seen that the responsibility of forwarding - or Routing - is that of IP, at the Network Layer. This topic is discussed further later in the chapter. It is important to note that IP is a connectionless protocol; it does not explicitly set up a route before sending traffic, or force the traffic from a particular flow to follow the same path - routing is done on a per-hop-per-packet basis. It is the task of TCP to handle the implications of this.

2.2 Network Management and Control Defined

Although Network Management and Network Control may sound synonymous, they have specific technical meanings. Indeed, Network Management has two elements - Network Control and Network Monitoring [Stallings 1993]. The first of these, Network Control, attempts to ensure that the network can be kept working; whilst the second, Network Monitoring, means that the administrator is aware that either it is working, it is not working, or that it is about to stop working.

This section discusses the function of Network Management as a whole, as a precursor to the descriptions of various protocols described in the rest of the chapter which implement Control and Monitoring.

Two major initiatives tackling Network Management have taken place, and the story mimics very closely the one described in the previous section. The OSI effort produced a system to work alongside its protocol stack, but was again outpaced by the IETF,

creators of the TCP/IP suite. The OSI system will not be discussed here, other than for its groundwork specifying the basic requirements.

2.2.1 Functions of Network Management

The OSI model used the following areas to define the function of Network Management, and they are broadly regarded as the standard definitions [Stallings 1993]:

- Fault Management - facilities which detect, isolate and correct faults.
- Accounting Management - facilities for charging users and costing.
- Configuration and Name Management - facilities that control, identify and handle data for smooth operation of the network.
- Performance Management - facilities to evaluate effectiveness.
- Security Management - facilities to protect the network and resources.

2.2.2 Functions of Network Monitoring

Network Monitoring is the observation part of Network Management. It is used to measure and analyse the performance and status of the network and its configuration. Although it is used for detection of problems on the network, it does this in a way that is more suited to off-line correction than to on-line, real-time problem resolution, so that the administrator has some insight into what might be wrong on the network. The next section describes the *de facto* standard protocols to perform such functions.

In the context of this thesis, Network Monitoring is given less focus than Network Control, described next, but is included here for completeness, and to place next chapter into context.

2.2.3 Functions of Network Control

As opposed to Network Monitoring, Network Control is the active component to

Network Management; it manipulates parameters to ensure keeping the network running smoothly, or recover it when it is not.

The complexity of even small networks is immense. Just trying to configure a desktop PC to work over a network can be very frustrating - a common enough experience. The operation of a network is inherently orders of magnitude more complex than that of a single PC.

Many aspects of Network Control have optimal settings incompatible with the optimal settings of other aspects, so the optimal balance of the two is extremely fine. Finding out everything about a network can be done by flooding it with traffic designed to discover details of it, but the very act of attempting to measure a network in this way can adversely affect the very performance being measured. In an extreme case all the systems resources are used in the measurement or control so that no actual user-traffic is being transported - the *raison d'etre* of the network.

Another example is the use of the TCP. In attempting to find the maximum rate at which hosts using TCP can send traffic they invariably exceed it, causing congestion. The host increases the transmission rate until it receives no acknowledgement and then starts again - increasing its rate slightly less aggressively. This makes it congestion *causing* rather than congestion *avoiding*. But if the host settles at a lower transmission rate than is possible, it wouldn't be doing its job. Therein lies the quandary.

2.3 Network Monitoring Protocols

This section gives a brief overview of the IETF-led TCP/IP Network Monitoring System. It describes the development of SNMP and then its augmentation by RMON to improve its functionality.

2.3.1 SNMP

The Simple Network Management Protocol, SNMP, described in RFC 1441 [Case et al. 1993] was supposed to be a stop-gap (hence "Simple", which is a misnomer) while a more sophisticated protocol was produced; the more advanced protocol has yet to happen.

The protocol works by using a monitoring process, or *agent*, on every hardware device on the network (or by the use of a proxy agent for those that do not support SNMP) that generates a UDP packet when a threshold is exceeded, or when some other significant event occurs, as an early warning system. The events range from automated counting of regular events to *traps* which are fault-conditions that must be reported.

UDP is used to transport the data since this bypasses time-delaying procedures associated with TCP, such as connection-setup. This also avoids a significant amount of extra traffic associated with TCP setting up a connection etc. There is no encryption or authentication to protect SNMP messages other than the fact that each device belongs to a *community* which is branded onto every message to prevent a device reporting to the wrong system.

SNMP is the protocol by which the data is collected, but it is then stored in a Management Information Base, or MIB, defined in RFC 1213. The MIB is an address-space which contains many counters of events (e.g. the number of IP packets forwarded) and configuration data. MIB has become a generic term, hence the existence of many other MIBs (e.g. ATM-MIB, ADSL-MIB etc.) but the SNMP-MIB is the original.

Although SNMP is a useful tool by which information can be harvested about the network, and warnings can be received when something goes wrong it has significant shortcomings. These include the amount of data produced for a relatively small amount of information, the amount of traffic generated to transport this data; and as SNMP runs

at the TCP/IP-model's Application Layer (but uses UDP not TCP) it requires the lower layers to be working and cannot therefore monitor those facilities in the first place. The information gained only refers to particular devices rather than whole network segments, so an additional system, RMON, was designed to tackle this shortcoming.

2.3.2 RMON

RMON, Remote MONitoring, RFC 2819 [Waldbusser 2000] concentrates on Ethernet and Token-Ring networks. RMON also localises many of the processes, thus reducing the traffic generated, but at the expense of requiring more processing power across the network hardware. RMON is actually an extension to the MIB - it defines a further address space - rather than being an alternative to SNMP.

2.4 Network Control: Routing

The routing of traffic on a network is crucial to the effective operation of the network, and as such is an important part of Network Control. Ideally, every packet can be routed by the "shortest" route - but this could be taken to mean physical distance (unlikely), smallest number of hops, lowest level of delays, highest bandwidth route, or some other measure. The easiest measure, or metric, is the number of hops, since all that is required is a counter-field in a message to be incremented as it passes through each node. Physical distance rarely has much relevance, due to the speed of propagation making it an almost negligible effect, but the level of delays caused by congestion because of the available bandwidth can have a major effect on the route chosen.

The routing used on a network can be *static* or *dynamic*. *Static* routing is obviously the simpler to implement, but could also fail to cope with any change in the network. Usually this type of routing is programmed by the network administrator.

Dynamic Routing is that which is updated automatically by the network control system. Ideally Dynamic Routing would be responsive to problems caused by congestion and other constraints, but without causing oscillations - known as "Route Flapping" [Labovitz et al. 2000 & 2001] - which was a significant problem in early implementations.

In describing routing protocols, one invariably is drawn towards a summary of the development of the Internet, and consideration of how ARPAnet developed etc. In preference, the reader is directed to other texts, such as [Huitema 2000], for in-depth discussions of these developments.

By far the largest application of routing algorithms is in the Internet itself. The routing algorithms used within the Internet have had to be effective, and are also continuously proven on a very large scale. A major problem in attempting to introduce a new algorithm is the sheer volume of uptake that is required to gain a foothold, no matter how much better the new algorithm might be. Here, we consider the two most significant protocols used in the Internet - RIP and OSPF.

2.4.1 RIP

The Routing Information Protocol, RIP, was developed for the Internet in its early days. Its effectiveness comes from its simplicity. As an example of a *Distance Vector Algorithm*[1] Protocol it is inherently less complex than the other class of algorithms, *Link State,* which are described later.

Because of its simplicity, RIP is well suited to "small" networks, of 10 routers say, but is not suitable for large networks, as it has insufficient protection against the formation of loops (traffic passing through a single node more than once), which can cause congestion as well as producing "black-holes" into which traffic can disappear.

1 Also referred to as Bellman-Ford Algorithms, since they are based on the Bellman Equation.

The original Algorithm is described in RFC 1058 [Hedrick 1988]. A second version was later produced to address the loop-issue already mentioned; the latest revision is contained in RFC 2453 [Malkin 1998].

The second version of RIP was developed despite opinions that the development of OSPF, described next, would render it obsolete. This was done due to the ease of implementation in smaller networks, when the complexity of OSPF isn't necessarily desirable. The second version is described here.

As already mentioned, the main feature of the protocol, is that it is based upon one of a class of algorithms - Distance Vector Algorithms (DVA). This means that each node attempts to discover a metric for every other node in existence on the network, but exchanges minimal information with its neighbours in order to do this.

The minimal information referred to, is (i) the existence of *any* route, and (ii) some measure of the length of that route (its distance). This leaves each node, having exchanged the information, knowing the distance to each node, and the next-hop for that node. In the event of more than one route for any node, the distances are compared to find the shortest.

Although, the term *distance* has been used, this is not always an accurate description; a more general term for quantifying a route is *metric*. A metric can be based upon almost anything to distinguish between two routes - number of nodes passed through (hops), time taken, cost of using the route, a quotient of congestion encountered, bandwidth available etc.

The major strength of RIP is its simplicity but such simplicity causes its own problems. These problems include its inability to prove a negative - the only way to disprove the

existence of a route is for the metric to increase to infinity - but this will, by definition, require infinite time. To counter this, an arbitrary limit is imposed as the maximum metric, as a pseudo-infinity. The value used was sixteen - the maximum "diameter" of an RIP-network is 16. Thus it will still take 16 updates at most to discredit a route, and the administrator must ensure the network is small enough for this hop-limit not to be broken.

Another problem with using a low value as the pseudo-infinity, is the limitation on the range of values that the metric can take; it effectively rules out quotients of bandwidth due to the sheer range of data-rates a link can have - from a handful of kilobytes, to many gigabytes, per second.

Mechanisms have been inserted to address the problems associated with "counting to infinity", these include the simple precaution of not advertising a route to the node that is the previous-hop in that route; this precaution is known as *split horizon*. A further modification is known as *split horizon with poisoned reverse*; the poisoned reverse element means that in the event of a route timing-out, a node actively attempts to remove it from other nodes by advertising its distance as infinity.

Another manifestation of the counting to infinity problem is when loops are formed by two nodes that bounce packets between each other, as both have decided the other is the best next-hop; this is most likely to happen with an intermediate node, thus causing the problem over a bigger area of the network. It could be solved by recording within each packet a list of every node already visited, but this can be a significant overhead and is extra complexity in a deliberately simple protocol.

Other problems exist because a routing update is only valid at the time at which it is sent. By time the information is received it is potentially out of date, so the rate of table-updates is critical. Conversely, too-frequent updating of the routing information may cause problems because of the volume of traffic involved.

Further to periodic updates, *Triggered* updates are generated when a node detects a change in the network. These updates can cause large waves of routing traffic across the network if not carefully managed. A snowballing effect is created across the network by a natural synchronisation of nodes producing updates. However this can be solved by the use of stochastically inserted delays. Triggered updates are minimised in size and volume by only sending information about routes that have actually changed.

Various timers are used to attempt to keep information as accurate as possible; these include removing routes that are not reinforced periodically - otherwise it is difficult to detect a node that has failed. On the other hand, since the protocol uses UDP, each individual update-packet is not guaranteed to be delivered. Except for the most extreme conditions, a sufficiently large proportion of the packets should get through and keep the system operational.

Having described all these short-comings, if RIP really is restricted to small networks, including small networks with an isolating gateway to the Internet, then these problems will not cause significant symptoms. For all but these smallest networks, RIP is unsuitable, and OSPF must be used.

2.4.2 Open Shortest Path First, OSPF

OSPF is a *Link-State* routing algorithm. It is based upon the work of [Dijkstra 1959] whose algorithm is also known as *Shortest Path First*, SPF. The name OSPF derives from Open-SPF, meaning that it is an open implementation of SPF, and was developed by the IETF.

The latest version of the algorithm is described in RFC 2328 [Moy 1998] and requires 245 pages, compared to RIP's 39 pages, which does say something about the difference in complexity.

OSPF attempts to create a full map of the network on every node within the network giving each node its own bird's-eye view of the network. It does this by using a *replicated distributed database,* a tool of distributed computing. This compares with the RIP method, which is based more on the *need-to-know* ethos. OSPF is a comprehensive routing protocol which solves many of the problems associated with RIP - it eliminates loops, does not require counting-to-infinity to identify problems, and can use more comprehensive metrics and, importantly, multiple ones.

A *hello* packet is used to establish connectivity between neighbours. This connectivity must be refreshed periodically by retransmission of the hello packets. A typical refresh rate is 10s although this figure is configurable by changing the *HelloInterval* parameter. If the link is not refreshed it may be assumed to be defunct. There is an expectancy that not every packet sent will arrive, so this assumption will not be made immediately. OSPF waits for three consecutive missed packets, so 40s can elapse before a failure is detected (this is also a configurable parameter, *RouterDeadInterval*). It is also likely that the router itself will have some interrupt routine to detect the loss of a link. OSPFs Hello Protocol also ensures links are bidirectional, by requiring a confirmatory *hello* packet in return, and this process ensures that the *HelloInterval and RouterDeadInterval* parameters just described are compatible on both routers.

Each node on the network then declares what links it has direct access to, along with one or more metrics describing that link. This information is then broadcast to all the other surrounding nodes. Each node also keeps its own database of the topological information and from this, builds a network map. Although this is done separately on every node, the same algorithm is used, and so it is guaranteed to converge over time. Since each node has a complete map of the network loops are prevented from occurring as routes are defined using what should be, in the static case full information.

The process of ensuring every router has the identical lists of links is described as

Database Synchronisation. At the beginning of the process of database exchange, two routers that have exchanged *hello* packets, are described as being *adjacent.* Once the process has been completed they are described as *fully adjacent.* This process is described as reliable-flooding. It means that when one router starts the flood (by sending out its database on each of its links), every link in the network either transports a database update or an explicit acknowledgement packet; otherwise retransmission is triggered. This flooding tests both directions on each link.

In the event of a change in the state of the link, e.g. the link is broken, further messages are broadcast to this affect. Additionally, periodical update messages are broadcast every thirty minutes, and are expected at this time. Any links not confirmed within an hour are removed, 3600 seconds being the maximum permissible lifetime of an un-refreshed record. A link can be explicitly removed by advertising the age of its record as one hour, although OSPF only allows this to be done by the originating router.

A safeguard to prevent incorrect information is that a link will only be considered valid if its existence is confirmed by the routers at both end of that link This prevents unidirectional links causing "black-holes" where packets are routed to use such a link in the wrong direction and end up being discarded.

As described, ageing mechanisms for the information in the databases are built in to the protocol; this means that in the event of part of the network being isolated, then the information will eventually be removed from the database. There is also provision for authentication and the use of passwords to prevent misinformation caused by faulty routers and malicious attacks. A checksum is calculated by the originating router, and this stays with the record to check for corruption. A corrupted record is not acknowledged so that a retransmission occurs.

Some routing protocols use a *spanning-tree* to forward updates across a network. A spanning-tree includes each node once and once only (which can provide some benefits

in terms of efficiency and synchronisation). OSPF does not. Resilience is provided by ensuring the failure of a single node does not prevent synchronisation of the rest of the network. Conversely an equivalent single node failure could cause problems within a spanning tree.

The link-state records are sent, up to five at a time, in specified formats using OSPF's own transport mechanism (it does not use UDP or TCP, but uses IP directly). In periodic updates the whole database will be sent - usually requiring more than one packet, but triggered updates only contain the records that are affected by the triggering event.

OSPF does not specify a metric that must be used to quantify the cost associated with the links - it leaves this up to the network administrator/designer. The only restrictions are:

- The "sense" of the metric is that a lower number will signify a preferred link.
- The summing of the metrics along a route must make some logical sense to allow direct comparison of different routes.
- The sum of the metrics for every route (and therefore for an individual link also) must be between 1 and 2^{16}-1 (35,535).

As can be seen, OSPF is a comprehensive protocol and has been designed to fit into a hierarchical routing/network structure. This means that OSPF is capable of running on an Autonomous System (AS). An AS is a network on the Internet which is under the administrative control of one body, as opposed to the core network. The Internet is made up of many ASs. The term AS is used rather than using terms networks and subnetworks for clarity. OSPF is able to route traffic to the edge of the AS to be passed up the hierarchy. OSPF is able to distinguish between routing information passed to it from external sources and its own information; it inherently trusts its own information over other sources.

2.4.3 OSPF versus RIP

OSPF has a similar relationship to RIP as TCP has to UDP. OSPF and TCP are comprehensive protocols designed to address all the issues; however, in doing so, they have become complex and impose relatively large overheads. Just as there are occasions when End-to-End connection guarantees are unnecessary (say for a real-time video transmission) then there are occasions when all the features of OSPF aren't required; as already stated, this can be true in setting up a small LAN, (Local Area Network) on an ad-hoc basis.

Although its shortcomings are well documented RIP does provide a lightweight solution for small networks where the complexity of OSPF is unnecessary.

2.4.4 Other Routing Protocols

OSPF and RIP are both examples of an Internal Gateway Protocol, IGP; they are designed for routing within an "Autonomous System", or AS. An AS is of relatively limited size, and could be the network controlled by an ISP or a company; they are the bottom-end of the Internet, which users directly connect to. The top end of the Internet is only really made up of servers and routers (few hosts and users), and routing in this section is handled by a separate class of protocols, External Gateway Protocols (EGP), or Gateway to Gateway Protocols (GGP).

Other IGPs do exist, such as IS-IS (an OSI development) and IGRP, but these will not be considered here, as OSPF and RIP are the *de facto* standards within the Internet. The reader is referred to [Huitema 2000] for details of other protocols.

2.5 Admission and Congestion Control

The issue of implementing Connection and Admission Control, CAC, does not affect Internet traffic at the time of writing, since there is no Quality of Service, QoS,

provision at the IP-layer that would make it necessary. Currently the Internet, through the Internet Protocol version 4, IPv4, only offers a Best-Effort service with no guarantees of delivery.

The discussion here, cannot merely just discuss the current *de facto* standard technology as in the previous section on Routing, since IPv6 has not been deployed in sufficient quantity to be considered *de facto*. The current protocol designed for this task is the Resource reSerVation Protocol, RSVP; this is described after a discussion of why the task is non-trivial.

2.5.1 Statistical Multiplexing

The implementation of the CAC procedure is far from simple; it adds considerable overhead to the communication process. The network must find an unbroken route that has sufficient resource to carry the traffic. This is complicated by the many different parameters that must be considered - these include overall delay, variation in this delay (the jitter), burst size, average rate, etc. In addition to these difficulties, all the traffic is multiplexed onto the links; to improve the utilisation and efficiency the technique of statistical multiplexing is employed.

Statistical Multiplexing relies on the probability of a number of different independent sources not all transmitting at their peak rate at the same time. As a simple example, if each source has a peak rate of 25% of the bandwidth but an average rate of 10% of one particular link; to guarantee no problems, only four sources would be admitted to the network. However, this would mean that only 40% of the link was being used regularly. A deceptively obvious solution here is to allow ten sources onto the link, thus giving 100% on average. Whether allowing all ten sources onto the link is a sensible policy depends on the details of each source; the distribution of the traffic production around the mean and peak will decide this, and in this example there is no margin for error.

Statistical Multiplexing introduces economies of scale and works best when combining (statistically) similar traffic sources.

Implicit in the admission of traffic procedure is some calculation as to the characteristics of the traffic, and whether there are sufficient available resources given the traffic already incumbent in the network, as well as, arguably, whether there are sufficient resources for future traffic demands. The traffic source itself is required to predict its own traffic characteristics, so that it can agree a contract with the network. There also needs to be some actual cost (money being the most immediately obvious candidate) associated with using up resources, or the first traffic source could block-book all the available resources.

2.5.2 RSVP

RSVP, IP's Resource reSerVation Protocol, defined in RFC 2205 [Braden et al 1997] is an attempt to implement end-to-end QoS provision on the incumbent technology, IPv4, which was not designed with QoS in mind. RSVP can only work with nodes that are RSVP-enabled, so there may be parts of network where the service is still only best-effort - in which case it could be argued that there is little point in attempting to implement it in the rest of the network.

RSVP contains an Admission Control function. RSVP-aware nodes interpret the control packets distributed about an offered flow, and decide whether the resources are available to support it on that node. Other checks possible include whether the user offering the load is permitted access to those resources. Failure means an error message is returned and offending packets are dropped or marked, whilst success means the requests are passed on to the next node.

RSVP also contains a *Traffic Shaping* method, defined in RFC 2212 [Shenker et al. 1997] to help ensure that the traffic patterns stay within the resources that are available. Traffic Shaping is done by the user, or ingress node on their behalf, and is simply a

buffer to ensure the traffic is compliant when it enters the network; it is to the users own benefit to reduce the likelihood of dropped packets.

2.5.3 IPv6

IPv6 was designed from an early stage to support QoS requirements. A field has been set aside for the class of traffic which is then split in two for Real-Time and non-Real-Time traffic; this is shown in Figure 2.2. Note that there is *no* specified relationship between the relative priorities for RT and nRT. Although these considerations have been built into IPv6, it remains to be seen if they are implemented and deployed successfully[2]

No	*Non-Real-Time (Congestion Controlled)*	*No*	*Real-Time (Non-Congestion-Controlled)*
	Low Priority		***Low Priority***
0	Uncharacterised traffic	8	Most willing to discard (e.g. high-rate video)
1	Filler traffic	9	.
2	Unattended data transfer (e.g. e-mail)	10	..
3	(Reserved)	11	...
4	Attended bulk transfer (e.g. FTP, HTTP)	12	...
5	(Reserved)	13	..
6	Interactive Traffic (e.g. Telnet)	14	.
7	Internet Control Traffic (e.g. OSPF)	15	Least willing to discard (e.g. low-rate audio)
	High Priority		***High Priority***

Figure 2.2: Table of IPv6 Traffic Classes and Priorities

There is a responsibility on the user of such priority schemes to be truthful about the contents of the traffic. It would be very easy to mark everything at the highest priority to the detriment of other users on the network.

2.5.4 ATM Traffic Management

Asynchronous Transfer Mode, ATM, is a connection-oriented packet-switched

2 IPv4 has a TOS field but it wasn't initially used. It is now used to identify DiffServe traffic classes

technology which was designed from the start to offer QoS guarantees. Before any offered traffic can be carried across the network, the connection must be admitted - the network will check that it has the resources to handle the level of traffic anticipated - and a contract agreed. If the traffic source breaches the contract then punitive action can be taken by the network, such as early discarding of its traffic.

The simplest Connection Admission Control, CAC, procedure would be to sum the maximum instantaneous traffic offering of every source, called the Peak Cell Rate, PCR, in ATM terminology. However this would result in low utilisation of the network resources (as described in Section 2.5.1). Other strategies could use the PCR along with the Sustainable Cell Rate and Maximum Burst Size to tune the level of multiplexing to allow greater utilisation.

As well as policing the traffic flows to ensure that each flow abides by its traffic contract, it is also desirable, if not essential, that the sources, or the ingress node (on the sources behalf), shape the traffic to ensure it does comply with the contract.

2.6 Configuration Complexity

This section discusses attempts to simplify the process of configuring networks. Human configuration of networks is very susceptible to error since it involves many steps and many text entries; a single incorrect entry can prevent an entire network from working.

2.6.1 DHCP

DHCP, Dynamic Host Configuration Protocol, RFC 1534 [Droms 1993] and RFC 1542 [Wimer 1993] for automatically configuring hosts using IP/TCP was partly promoted with the dual aims of preserving IP addresses which were running in short supply at the time, and enabling a network administrator to configure and manage an entire network

from a central workstation. Obviously this makes the network dependent on the central server, but the default lease time (the maximum time before a host stops using an IP address it has been assigned under DHCP) is often 3 days, so some latency exists.

DHCP works by assigning IP addresses on-demand and with a specified time-out, after which the hosts must apply for an extension or new address. Special "hosts" such as servers can have their IP address reserved to keep them within the scheme, but able to use a "permanent" address. DHCP can also cover more than one "segment" of a network, with the use of a *relay agent.*

2.6.2 OSPF

OSPF, discussed in the section on Routing, being a dynamic routing protocol, offers simplification in the process of network configuration; the network administrator can rely on it to sort out the routing, rather than having to manually configure each router. This protocol is discussed in Section 2.4.2.

2.7 Chapter Summary

This chapter has outlined the traditional mechanisms used to communicate information around computer networks. Two related frameworks have been outlined, OSI and TCP/IP, which aid the design, implementation and integration of constantly improving technology into existing communications systems.

The Internet standard Network Management protocol is then described. Although it concentrates on the monitoring side, the Simple Network Management Protocol, SNMP, and a vital extension RMON, is important because it allows information to be gathered regularly by the network to highlight problems as soon as they happen.

The various routing protocols that are used within the Internet have been outlined, so that they can be compared with the work described later in this work. The routing decisions can be heavily affected by the implications of Quality of Services constraints - the mechanisms to handle these are also described, as these also affect whether traffic offered to the network is taken up.

Finally, the effect of overly-complex configuration requirements are discussed in the context of setting-up these networks. The Internet standard, DHCP (Dynamic Host Configuration Protocol) automates the tedious work of setting the IP addresses on many machines by hand, making the network administrator's life easier. Interestingly, in contrast to many of the techniques described in this thesis, DHCP actually centralises control rather than distributing it.

The following chapter contrasts the implementations presented here, with more experimental approaches that attempt to improve the effectiveness by controlling the network intelligently.

Chapter 3: Intelligent Network Management and Control

Having described in the previous chapter the purpose and means of Network Management and Control within the area of Telecommunications, this Chapter now outlines and describes work that has been done to automate these functions, utilising and integrating theories and practice from the area of Artificial Intelligence, AI.

There are numerous areas in which AI can be utilised within telecommunications, and indeed there have been implementations; some of them on real systems, but significantly more on simulated systems.

AI has been applied to various research areas. Broadly, these efforts fall into the following areas:

- Automation of Negotiation, e.g. in resource allocation
- Intelligent Threshold Setting, e.g. in congestion notification
- Swarm Intelligence, e.g. Ant-Based Routing
- Reasoning and Planning, e.g. Disaster Recovery
- Auto-Configuration

This Chapter describes these efforts and existing work in applications as well as critical analysis of their potential. Given that a common thread through a number of these applications is the presence of "agents", a brief introduction to them is given first.

3.1 A Brief Introduction to Agents

The basic concept behind an agent is a simple one; that is, in the author's opinion, it is a vehicle for Artificial Intelligence, in the *very* broadest sense of the term - it could just be a particular algorithm. The boundaries are blurred between Agents and ordinary software Objects - in essence, while a print server (daemon) *could* be described as an agent, it is a trivial case and therefore uninteresting to researchers.

This section aims to give an brief outline of Agents, and a more detailed breakdown of those aspects directly relevant in the context of this thesis. The reader is directed to the many standard texts on the subject, such as [Russell and Norvig 1995] in the case of greater interest.

Having stated the author's opinion, it must also be stated that the term "Agent" is taken to mean many (different) things to many (different) people. This is mainly because of the wide variety of contexts, applications and indeed academic subjects Agents are used in. The Agent community is made up of, amongst others, Computer Scientists, Mathematicians, Psychologists, Biologists, Physicists, Philosophers, Linguists, Logicians, and of course Engineers. Even when there is agreement between the various parties, it is not necessarily obvious due to different and conflicting terminology used.

Agents are essentially a means to encapsulate software. This encapsulation then means that everything external to the agent is considered the *environment*, with which the agent interacts. This environment may indeed contain other agents. Interactions with the environment can then be formalised, meaning anything crossing the boundary is either an input or an output. Processing is performed by the agent which considers its *sensors*, and decides what actions it should take, using *effectors*. Ideally in between the inputs and outputs is some reasoning or planning or learning etc.

This encapsulation, shown in Figure 3.1 allows researchers to concentrate on the

processes in the middle - the more interesting parts - rather than start from scratch each time. This is a similar concept to that of the OSI and TCP/IP models (layers are used to abstract physical details away from the higher-level control strategies) and also the basics of Object Oriented Programming (OOP). "Agent Oriented Programming" is seen by some as a logical next step from OOP [Jennings 1999].

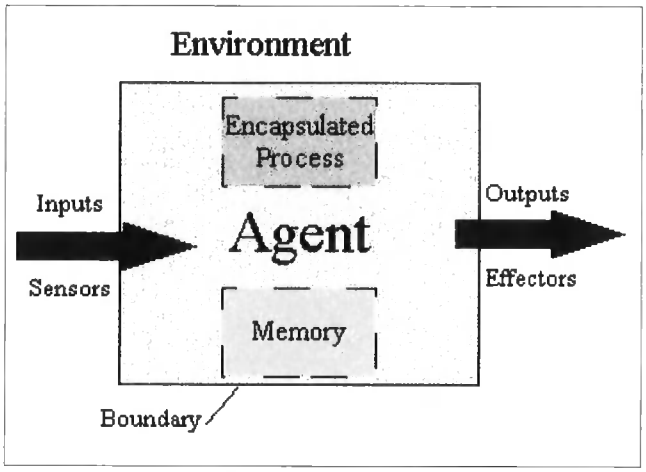


Figure 3.1: Basic Diagram of an Agent

There are many different things that an Agent could be armed with, including reasoning, learning, negotiation, trading, and many other skills. These skills, which ultimately will be combined, could be implemented by any number of different techniques - Neural Networks, Genetic Algorithms, State Machines and Reinforcement Learning techniques are all examples.

It is reasonably obvious that an ultimate aim of, and motivation for, an agent is to take over the role of a human in an automated system. This also fits in with the Turing Test [Turing 1950], which states that a human should not be able to tell whether the thing it is interacting with is a computer or a human. This is not, however, to preclude inspiration from the animal world.

A popular application to be quoted is agents that represent a person on the Internet and book travel and flights for them. However, in the context of this thesis, such

applications concentrate on different areas of agent research, so are not discussed further.

As mentioned already, Agents are not necessarily isolated. Indeed much of the more relevant research is in the area of Agents interacting, either competing or team-working, and this is generally referred to as a Society of Agents. This leads to the idea, in the application described, of an agent representing the person booking, and the travel agent being represented by another agent, thus necessitating negotiation between the two.

Entire societies of agents can also be abstracted to be seen as a single agent. It is often easier to program a number of simple agents that interact together to produce the required complex behaviour. Although these simple agents do not then display "intelligent" behaviour, the society as a whole does - hence the abstraction. There are no better examples of this than the original proposing work, [Brooks 1986], and the very readable popular science book by [Braitenberg 1986].

The following sub-sections go on to formalise the definitions of agents, and then outline some of the properties required.

3.1.1 Definitions of Agents

Definitions of Agents are as varied as the field is wide; as a start the more generic definitions are more helpful. [Jennings 1999], a leader in the Agent-community, defines them as "situated problem-solvers".

Jennings' definition gives a feeling of what they are for, but further definitions include the following:

"An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meets its design objectives" [Wooldridge 1997]

"An agent is a considered as a physical or abstract entity capable of acting on itself and

in its environment, to manipulate a partial representation of its environment, and to communicate with other agents. The agent behaviour is a consequence of its perception, knowledge and interactions with other agents." [Gaïti 1996].

Although these definitions are helpful, they are descriptive. The next step is to establish the properties required to define an Agent.

3.1.2 Properties of Agents

The exact properties, and combination thereof, is a source of great debate. The following represent a selection and are relevant to the discussions within this thesis. They are listed here and then detailed below:

- Autonomous
- Intelligent
- Interactive
- Reactive
- Proactive
- Goal-Driven

There are other properties which it can be argued should be included, which are not discussed in this thesis. For example, the property of Rationality is interesting because it is measured against a theoretical "ideal rational agent". This acknowledges that the agent must operate in Real-Time and with imperfect sensors, so this ideal must be curtailed to the concept of Bounded Rationality [Russell & Norvig 1995][Simon 1957]. This, and other properties, are left to more weighty tomes.

Autonomy - This effectively means, in a software sense, that the Agent retains its own process thread and associated thread control. The agent should persist in the environment[1] longer than a single execution of its algorithm. Unlike an object, where any other object can call a public method, all calls on an agent are in the form of a request message. It is then up to the agent whether or not the request is fulfilled. The agent may decide it doesn't have the resources, that it can't achieve the task, or that the

1 Persistence can also be argued to be a separate property of Agents

request is in contradiction to its own goals.

The requirement for an Agent to be Intelligent has already been mentioned. It means that an Agent should have something a little extra than an ordinary software object - it must encapsulate some intelligent algorithm, some learning, some reasoning etc.

As discussed before, it is not always necessary that intelligence be displayed at the level of the individual agent, giving rise to *Simple Reactive Agents,* but intelligence can also be displayed at the level of the society. An example of this is the Ant-Based Systems discussed later, where the individual follows simple rules which in turn give rise to complex behaviour at the society (or colony) level.

There is also an inference that the agent's responses should not merely be pre-programmed, but that the agent should gain experience over time, which affects the agent's response.

Although there is a stereotype for an AI system in the future being capable of completely independent and creative thought, autonomy could merely be the adaptive setting of the length of a time-out to its optimum, or some relatively simple task.

Reactivity and Proactivity are two extremes on the same scale. A purely reactive agent would only ever *react* to events in its environment, whereas a purely proactive agent would basically only ever reason with itself not achieving anything.

For example, if an Agent had the goal of crossing a road, a purely reactive agent would be merely attempting to avoid the traffic. A purely proactive agent would sit on one side of the road considering plans on how it might cross. Neither of these approaches will be successful.

To be successful, an agent should combine the two approaches. This is equivalent to a human having some subconscious reflex reactions, but also the ability to consciously decide on a course of action. This could be seen as "artificial instinct" mixed with "artificial intelligence".

The requirement for an Agent to be Interactive means that it must be able to act in a social context as well as react (and proact) with its environment[2]. Much research in the field of Agents has focused on the process of communication, and then on the underlying sociological processes that this enables. The ability to work as a team may greatly enhance the value of agents, just as a team of humans can achieve more than when working as a group of isolated individuals.

Finally an Agent should really be Goal-Driven. This means that it should have long term and short term goals and, ideally, be able to change and adopt different ones. Thus the Agent can prioritise between goals and decide which it will concentrate on at any given time.

3.1.3 Subsumption Architecture

There are a number of different standard architectures used to build agents; these include the Belief-Desire-Intention (BDI) Architecture, Blackboard Architecture and Subsumption Architecture. However, only Subsumption is discussed here, as it is pertinent to this thesis

Subsumption Architecture was proposed [Brooks 1986] in the context of controlling robots, and an example is shown in Figure 3.2. These robots can have the goal of mapping a room, for example. However, they must first be able to avoid objects, and then move around the room. This is similar to sub-concious reactions in humans which take over from conscious decisions in the case of danger; extreme heat for example.

2 Although arguably, other agents and humans are part of the agents environment, and so would fall into the reactive/proactive properties anyway.

Within Subsumption Architecture the agents work in hierarchical layers. Agents in each layer receive their input from a sensor, and are free to calculate an output based on its inputs. However, the output may be modified by agents in the layer above. Hence the lower layers are subsumed.

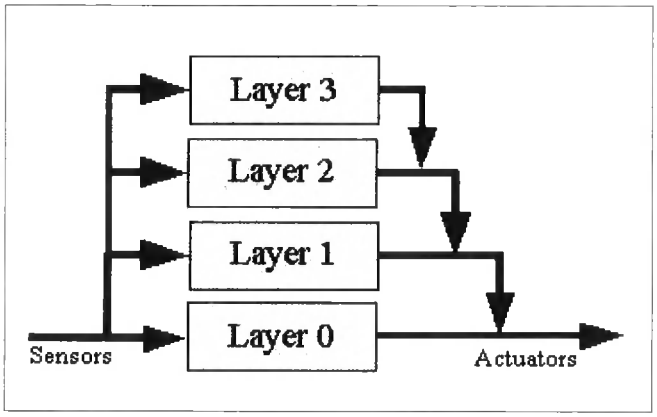


Figure 3.2: Diagram of Subsumption Architecture

For example, an agent on layer 3 can distort the input from the outside to alter the reaction of an agent in layer 2 - this in turn can change the perception at layer 1 so that it takes a different action to that which its unaltered sensors might suggest.

In this way, each layer provides an extra layer of abstraction in the sense that:

(i) the higher the layer the increasingly proactive it can be (being more and more sheltered from short-term considerations)

(ii) the lower layer is guided in matters of urgency - so is abstracted from the environment.

Again, there is a parallel between this and the models discussed in the previous chapter.

3.2 Connection Admission Control Agents

By allowing more than one agent to operate in an environment, social interaction can be

investigated. This environment may include co-operative teams of agents, altruistic agents and ruthlessly competitive agents, with everything in between. In many of these situations, agents are expected to negotiate with each other for the purpose of task distribution or buying and selling.

This area of cooperating agents maps well to the set-up process for calls in a connection-oriented network. Each call is assigned resources - bandwidth, Quality of Service level etc. - and the network knows what resources it has available for further calls. Eventually, if it has insufficient resources for a new call it must reject it.

This call-set-up process represents significant overheads to the system. The call must be assigned sufficient bandwidth and queueing resources from end to end. If a link along the route has insufficient resources, the routing must be "cranked-back" to find an alternative way round.

On ATM networks this call set-up process is called Connection Admission Control, or CAC. The commitment does not stop at call set-up: if the call proceeds, it must be monitored to ensure that it only uses the resources it requested. Indeed, the call requester agrees a contract with the network, and can be punished by having its packets being preferentially dropped if it violates that contract.

Two experimental implementations of CAC schemes are described below, both of which make use of AI techniques.

3.2.1 Hall 1998

This scheme is described in [Hall 1998]. As will be seen, it has similarities to the work described in this thesis, but differs in a number of crucial areas. The system uses a learning mechanism to evaluate the utility of each link, but this evaluation is based only on the results of previous efforts: there is no extra exploration of the network, so a sub-

optimal solution could be learned.

The learning mechanism itself is a Neural Network (NN), whose inputs are the Delay Requirement and the Arrival Rate. The output of the NN is a quotient of the amount of resources a statistical multiplexer must set aside for that flow, a figure that is interpreted as a probability[3].

The aim of Hall's work was to implement a measurement-based CAC. In the case of simple situations that were open to analysis, the NN was able to learn the correct probabilities. It is noted that such simple scenarios are likely to be rare, so the NN was also applied to more complex scenarios and was still able to learn "correct" probabilities.

3.2.2 Hayzelden 1999

The ACTS-IMPACTS project aim was to implement CAC in the ATM domain using agent technology. The structure it uses is that of Subsumption Architecture, described previously. The work described in [Hayzelden 1999] is part of this project, and a number of key concepts can be drawn from it.

An application wanting to send traffic must declare what level of service it requires (e.g. CBR, NRT-VBR), and some parameters to describe the offered traffic (e.g. PCR, DVT). The network then attempts to find a route across the ATM network with sufficient resources available to carry the traffic; the network must also decide if it wants to commit these resources to that particular application.

Because the network provides *virtual* circuits only (so traffic is multiplexed), the network can admit more traffic than it could apparently carry. For example, suppose one link in the network has a capacity of 155Mbs, but the network admits different traffic types whose PCRs sum to 200Mbs. The network can do this because statistically the

3The statistical multiplexer's resources total unity, so a flow requires some fraction of those resources.

sources are unlikely to all transmit at this rate at the same time.

Hayzelden's work and the ACTS-IMPACT project, are a proof of concept that a Subsumption Architecture based Multi-Agent system can be used to control the CAC of a telecommunications system. Results show that the implemented system is able to utilise the bandwidth on a real ATM network better than a traditional CAC policy.

3.3 Intelligent Threshold Control

A problem with using a set threshold within a system, is that there can be circumstances in which the threshold becomes unsuitable, hindering the operation of the system. For example, a speed limit on a road might be suitable when the road is dry and there is little congestion, but may be dangerous when wet and there is heavy traffic - as is the case, for example, on the south-west stretch of the M25 motorway around London. Indeed, overall throughput of the road can be increased by reducing the speed limit at such times. The following section describes attempts to apply such adaptation of thresholds to the area of Telecommunications.

3.3.1 Gaïti and Pujolle

Agents have been used to implement intelligent Congestion Control in ATM networks such as [Gaïti 1994, 1996a, 1996b]. In this work, which is simulation base, agents were placed on each port on each node. The agents measured various aspects of the traffic passing through that node. Using this information, four different thresholds were set. These thresholds were cumulative - each triggered increasingly drastic action to prevent, and then counter-act, congestion.

The agent can then use the information collated to vary the level at which action is taken, or indeed what action to take in the event of congestion,or the approach of congestion.

The agent will need to monitor:

- overall level of traffic against the total capacity,
- amount of traffic at each level of priority,
- amount of queuing (buffering) required,
- a measure of the level of discards (be it cells, packet or datagrams),

This would seem a sensible and logical step to allow a network node to watch for trends in the network, and adjust the threshold to reflect local knowledge. A static threshold is only likely to be optimal in a static environment, which is not appropriate when considering a telecommunication network. The method showed an improvement in performance when compared to using statically-assigned threshold values within the network.

3.4 Ant-Based and Intelligent Routing

Ant-Based Routing is not *strictly* part of AI, not being inspired by humans (cf the Turing Test [Turing 1950]). However, it would be classified as AI in the very broadest sense discussed earlier in the author's definition of agents. Ant-Based routing is an application of Swarm Intelligence, or more generically, Nature Inspired Solutions, which is also called Artificial Life.

Ant-Based solutions were originally applied to such problems as the Travelling Salesman Problem in [Dorigo & Gambardella 1997]. This problem is concerned with planning routes, which obviously has a relevance to the area of computer networks. Thus Ant based solutions were applied firstly by [Schoonderwoerd et al 1996, 1997] and then by [Di Caro & Dorigo 1997a, 1997b, 1998], as described next.

The basic concept behind Ant-Based Routing is the use of many small packets which are routed probabilistically around the network, reinforcing, i.e. increasing the

probability of, the routes which they take.

3.4.1 Schoonderwoerd et al

The first application of Ant Based Solutions to the area of Telecommunications is described in [Schoonderwoerd, Holland & Bruten 1997] and [Schoonderwoerd, Holland, Bruten & Rothkantz 1996].

[Schoonderwoerd et al 1997, 1996] compares Ant Based Control, ABC, to a mobile agent approach proposed by [Appleby & Steward 1994]. The network involved is a telephone network, and the measurements revolve around the call acceptance rates through the network.

Appleby & Steward's approach uses Parent Agents, that move around the network monitoring it. If an Agent decides that a node is performing sub-optimally, then it launches a Load Management Agent that investigates the best routing around that node using an adaptation of [Dijkstra [1959] to update the routing table.

It is notable, however, that in the case of congestion the generation of a traffic generating agent could be counter-productive, because it would then be desirable that control traffic would be minimised to concentrate on transporting data. It is also pointed out that should an Agent fail, it would not be a trivial task to recover; this could be seen as a disadvantage. This is in contrast with an ant system, where the failure of an ant would have little effect indeed, the system derives as much information about the network by an ant *not* arriving as it does when the ant *does* arrive.

The following equation, Equation 3.3, is given as the reinforcement given by the ant to the route it has just taken, where Δp is the change in probability applied to the route used by the ant.

$$\Delta p = \frac{0.08}{age} + 0.005$$

Equation 3.3: Ant-System Reinforcement Expression

The terms of proportionality in Equation 3.3, however, are given empirically, with no explanation as to how they were arrived at, or the precise effect of each term. When the reinforcement is applied, all the routes must be normalised, so that the probabilities sum to unity.

In the case of congestion, Equation 3.4 is used. The Ant is artificially aged according to the spare capacity available, where s is spare capacity, ranging from 0 to 100.

$$delay = 80\, e^{-0.075 s}$$

Equation 3.4: Ant-System Artificial Ageing Expression

3.4.2 Heusse et al

[Heusse et al [1998], but also originally proposed in [Guérin 1997], adapt ABC [Schoonderwoerd 1996] using an Ant, travelling more than one hop, to update the probability of all the intermediate nodes as well as its source node. This reduces the number of ants required for a given speed of convergence, and also helps to eliminate circular routes, since the ant has a record of all the nodes it has visited. However, the scheme does introduce either variable-sized ant-packets, or arbitrarily large packets.

To distinguish this system from [Schoonderwoerd's 1996], the adaptations are referred to as "ABC with Smart Ants". The other major difference is that the ants perform round trip journeys, to propagate the information back to the source node.

3.4.3 Bonabeau et al

[Bonabeau et al 1998] took the [Heusse 1997] idea and apply it to one-way ants - that is the Ants do not return to there source to propagate the information.

3.4.4 Dorigo et al

Di Caro and Dorigo, pioneers of Ant and Swarm Based methods [Dorigo & Gambarella 1997] proposed an Ant-inspired solution to telecommunications routing, AntNet, in [Di Caro & Dorigo 1997a & 1998].

AntNet is an Ant-Based Routing System for datagram networks, and uses Forward Ants and Backward Ants. The Forward Ants search the network, and are subjected to the same conditions as the actual traffic, while the Backward Ants convey information back across the network to speed convergence, and are given a higher priority.

The network measurements used in the determination of the reinforcements are all based on the time taken to traverse the network. However, the synchronisation of time across a network is not a trivial task, especially to the precision and accuracy required, and this would make the scheme hard to implement in practice.

Swarm Intelligence, the book by [Bonabeau, Dorigo & Theraulaz 1999] is a standard text on the subject. It discusses the work of Schoonderwoerd and Caro & Dorigo, together with developments to these.

3.4.5 White et al

In [White et al 1998], the problem addressed is that the solutions proposed involving Ant-Based Routing and Load Balancing generally have a large number of tunable parameters. This not only means that significant work is required to find the optimal settings, but also that the control-system can become fragile when faced with dynamic situations.

The suggested solution to these problems is to use a form of Genetic Algorithm to evolve a better solution. Individual solutions are cross-bred to produce new solutions (which usually differ from both original solutions).

3.4.6 Choi and Yeung

[Choi and Yeung 1996] suggest an improvement to the technique proposed by [Boyan and Littman 1994]. The earlier paper proposes a solution that is regarded as weak in two respects, both of which can be attributed to a lack of exploration undertaken by the learning system. The Exploitation-Exploration dilemma is a significant, and well-known, one in AI. Usually, continuing Exploration has an associated cost so that there is a compromise between using the best, known solution, and searching for another even better one.

The problem described concerns the fact that a network is a dynamic system; an optimal solution in one circumstance is not necessarily optimal in another. Often in learning techniques, the system either has a specific phase in which the system learns, or a decaying rate of learning is used, to smooth between the two phases. However, these systems usually have a stationary environment, and often seek terminal goal-states. Unfortunately, in continuing systems, and especially ones that are non-stationary, the goal-state is non-terminal, so that the agent must find the action that keeps the system in the goal state. In the case of a dynamic system, the correct action is likely to change.

3.4.7 Vittori and Araújo

[Vittori & Araújo 2001], use two Ant Systems as their control systems; ABC and ABC with Smart Agents. The ABC System is as described in [Schoonderwoerd 1996, 1997], using Equations 3.3 and 3.4. The Smart Ant system is based on [Boyan & Littman 1994] and [Kumar & Miikulainen 1997, 1999] who use the Q-Learning update rule (described in Chapter 6) to update the values associated with different routes.

The Q-Agent solution itself essentially uses packets that could be described as Ants, and for the purposes of this critique, this term will be used. The system uses Equation 3.3 to age the Ants. The Ants not only update the value for the source destination pair, but also the values for the intermediate nodes.

The values stored in the tables are interpreted as Q-Values, although they could be also viewed as probabilities. Q-Learning will not be discussed here, as it is discussed in detail in Section 6.2. The system updates in the forward direction as well as the backward direction.

Additionally, when selecting a route for the traffic, the amount of congestion on that link is consulted *as well as* the Q-Value for that link.

3.5 *Failure Recovery Agents*

This is an area where intelligent agents could offer significant benefits. The ability to adapt and reason strategically would enable the agents to handle problems associated with failure, but also enable coordination attempts to minimise the disruption caused to the rest of the network.

3.5.1 Pre-emptive and Dynamic Schemes

Although discussed in principle in [Nwana 1994], this paper merely discusses the coordination techniques possible for a failure recovery scheme. [Vilà et al. 1999] discusses a proposal to simulate an ATM network running a multi-agent system on each node which handles the reconfiguration or re-routing of Virtual Paths (VP) in the event of failure. Two different strategies are identified: (i) pre-planned schemes, where back up VPs are kept free for existing traffic to use in the event of failure, and (ii) dynamic schemes where re-routing is done wholly after failure. Obviously a sensible compromise would be to back-up VPs for high-priority traffic, but to dynamically handle lower priority traffic.

Keeping spare VPs, with the associated resources actually reserved, for traffic network-wide uses a vast amount of resources. Conversely, attempts at re-routing after a failure causes problems due to the amount of overhead traffic generated over and above the

otherwise unaffected traffic. This can then cause further congestion across the network, extenuating the original problem.

A compromise, where reservations are made for high priority traffic, whilst routes for lower priority traffic are merely earmarked, minimises the system resources tied up in backing-up a route.

3.5.2 Link and Node Disjoint Backup Routes

[McDysan 2000] describes two basic levels of resiliency, *link disjoint* where a back-up route shares no links, and the more stringent *node disjoint*, which shares no nodes (and therefore links). Having link disjoint back-up routes would mean a single point link failure would affect only one or the other of the back-up and primary route. Having node disjoint back-up routes extends the guarantee to single node-failure. There is an extra cost to using node disjoint routes, not only in terms of operational complexity, but also that the number of possible routes may well be dramatically reduced.

Both these schemes could be used in strategy (i), described in 3.5.1, of pre-defining routes to use in the event of failure, and, assuming single point failures, they will minimise disruption to a set of traffic.

3.5.3 General Suitability of Backup Routes

Since a back-up route is exactly that, there is a significant probability that it is only suitable as a stop-gap; it is unlikely to be optimal in terms of performance, since it was found under the main constraint of *not* sharing links/nodes with the primary one (rather than any measure of suitability). Once a failure has occurred, and the network has been allowed to stabilise, there may well be a case for finding a replacement primary route (and in time, re-evaluating the back-up route).

3.5.4 Garijo et al

[Garijo et al 1994a] describes work on an Agent to work with an ISO Management Information Base (MIB), and resolve fault conditions within the network. The system, called Cooperative Network-Fault Manager (CNFM), contains six classes of agents. Each agent specialises in a different task, from recognition of faults to diagnosis of them, and communicates with the user and other CNFM systems.

[Garijo et al 1994b] describes the implementation of two of the class agents, whilst the others are left to dummy-shells for development purposes. No results are presented.

3.6 Chapter Summary

This Chapter has described previous applications of Artificial Intelligence, and related techniques, to the area of Telecommunication Networks. Many areas of Network Management have been addressed, including: Connection Admission, Congestion Control, Routing, Configuration Management and Failure Recovery. Notably, all of these areas relate to natural areas of AI.

Having described the state-of-the-art in AI control for Telecommunications systems, the next Chapter describes the work undertaken in the context of this thesis.

Chapter 4: Agent System

This chapter details a novel approach to the overall management of a Telecommunications Network. Specifically in this case an ATM Test-bed is configured and controlled. The work is briefly outlined in [Legge & Baxendale 2002].

The objective of the work is to ascertain the feasibility of a system that can autonomously configure and manage a network by highly modular means and enable autonomous configuration; meaning that the network can manage itself without human intervention under ordinary conditions. Agents are used as they can be designed to be inherently modular and autonomous with the application of of Artificial Intelligence (in some form)to handle fault conditions.

The chapter firstly outlines in more detail the aims of this work, as well as some of the motivations and the major concerns that are to be addressed during the course of the rest of the chapter, as well as the rest of the thesis in some cases.

A description of the research environment is provided, including the hardware (the ATM Test-bed, and the agent toolkit which provides the framework for the software). With the scene fully set, an overview of the system is provided, before breaking down the system into the constituent agents.

A critical analysis of the system is then presented and then the chapter finishes with a discussion which also allows us to set the remaining chapters into their full context. Although this system stands alone in its own right, it provides motivation for the rest of the work presented in this thesis.

4.1 Overview of the Research

This chapter describes research work to implement an autonomous Network Management System using the BT Zeus Agent Building Toolkit, which is detailed in a following section. The network specific to this work is a six-node ATM Test-bed, which can be connected in any topology desired; however, it is intended that the management system is generic and is capable of implementation on different networks and protocols.

4.1.1 Motivations for Research

A classical paradox in the operation of telecommunications networks is the centralisation or distribution of control. A network is said to have centralised control when one node makes all decisions. A classic example is centralised routing, where a single node decided the routing tables for the whole network, and then had to communicate that across the network. Here there is a clear leader, and the assumption is that it can make impartial and co-ordinated decisions. Problems arise when the network is geographically distributed, and the central node has to make decisions with an incomplete and out-dated knowledge. Also, a link failure could cause the isolation of part of the network, or if the central node itself fails then the whole network could become inoperable.

In the other extreme, each node can make all of its own decisions. As a whole, the Internet runs on this basis; and it has shown itself to be highly resilient; for non-critical data is perfectly adequate. However, the Internet provides little if anything in the way of Quality of Service (QoS) guarantees. The whole of the Internet provides "best-effort" service in terms of QoS, and this is inherent in the protocols developed, certainly until the new Internet Protocol (IP), version 6 is fully implemented and deployed. ATM (Asynchronous Transfer Mode) however, is a protocol designed from first principles to enable guarantees on the QoS to be provided. It achieves this by tightly controlling its resources, agreeing contracts with users before admission to the network. It can then police those contracts continuously.

A perceived problem with ATM is the complexity of its configuration. Any automation of this process of configuration would be of benefit. At the same time, care must be taken to minimise the additional network traffic generated by the management system itself (referred to as Control Traffic).

4.1.2 Aims of the Research

The research described in this chapter has a number of aims. The primary aim is to build a modular network management system from autonomous software agents. These agents should be able to configure the network by themselves, and handle fault conditions automatically. The research issue is to enable this control in a realisable form.

This autonomous control system would represent a novel approach to the problems associated with hybrid-centralised/distributed control and could greatly simplify and reduce the effort required to administer a telecommunications network. It would also show that it is feasible to build such a system with off-the-shelf, standards compliant systems (Zeus), and that Agent technology has reached the level of maturity to achieve this.

The hybrid architecture enables the implementation of fully distributed control within a network, while maintaining the ability to take more holistically informed decisions (taking account of the state of the network as a whole, rather than just local conditions) for network-wide and longer-term strategies; this is achieved by communication between agents on each node.

Each node of the network will host its own agent "society" with a layered hierarchical structure. Although the lower-layer agents' actions within the society must be tightly prescribed, it is intended that higher-layer agents will take on more strategic, longer-term outlooks, and be more adaptive and autonomous. This means that the lower agents will have well defined tasks and roles, but with greater freedom higher up.

Agent societies will be able to communicate, to allow their collaboration. Initially this

communication will be restricted to be between equivalent agents on each node, that is on a peer-to-peer basis. The intention is eventually to allow generic communication between all agents.

4.1.3 Major Issues

There are two major issues, which are related, that must be considered through every stage of design and implementation. The first of these is Control Traffic.

Control Traffic is used by the network to administer itself. It can be defined as any traffic that is not directly payload (i.e. anything that is not data). Although a certain amount is essential to the running of the network, it can quickly build up and start to degrade the performance of the network by taking up resources that data-traffic should be using. Indeed, the efficiency of the network in this context is defined as in Equation 4.1.

$$Network\ Efficiency, \eta = \frac{Payload\ Traffic\ Transported}{Total\ Traffic\ Transported}$$

Equation 4.1: Possible Definition of Network Efficiency

The second issue, which is inextricably linked with the first, is scalability. This is the concept of whether a system, usually designed for a small number of nodes, can handle the demands associated with a significantly larger number of nodes. It should be said however, there is a practice of writing off good ideas very early on, before properly developed, because they are deemed to be *non*-scalable when not enough effort has been expended to optimise the processes involved to enable them to be scalable.

The two issues are linked by the fact that if a node produces a set amount of Control Traffic per node, τ, for a small network, with n nodes, the level of traffic could be $n\tau$, and in the case of a large network with m nodes, the level of traffic could be $m\tau$. This would represent a linear increase, and would not increase the average level of traffic per

node, so the system would be scalable. However, if a system exhibits exponential growth, say $m^2\tau$ in levels of control traffic, it will not be scalable, since the Control Traffic will swamp the network. Ways to deal with this include dividing the network into a hierarchical structure, just as Autonomous Systems do on the internet.

There are other properties of systems that might not be scalable, for example, time required for some task, or amount of storage required for the system to operate. These issues must be considered during the development process to ensure the system remains scalable and therefore deployable.

In reality no system is completely scalable. The Internet is constrained by the available address-space, and there have been colossal efforts to work around this constraint in IPv4, and these have extended its life considerably, but there are only so many addresses. It is notable that in IPv6 the designers quadrupled the number of address-space bits, thus increasing the number of addresses to 3.4×10^{33}-1. Perhaps this is a case of over-engineering, perhaps it is an example of extreme foresightedness.

4.2 Research Environment

This section describes the various tools, both hardware and software, available for the course of development and experimentation. Firstly, the ATM Testbed is described, which was a major capital outlay, and then the software package which provided the framework for the agents and society Zeus.

4.2.1 ATM Test-bed

The University of Durham's School of Engineering owns a six-node ATM network, dedicated to research work [website a]. This is shown in Figure 4.2 with all possible connections made. This means that any topology can be configured and any test scenario

used. All six switches, two UNIX machines and six PCs are connected by 10-Megabit Ethernet connections on a firewall protected network.

Video codecs and network analysers can be used to generate and capture traffic. This means that video can be transported across the network and the difference in quality and effects of network delays can be observed.

The ATM switches themselves have limited processing capacity in terms of running extra processes, and also suffer a proprietary operating system, which complicated the possibility of basing the system within the network itself. Instead, it was decided to use the PCs attached to each node to host these processes. This added an extra interesting aspect to the work the control being distributed in an additional sense.

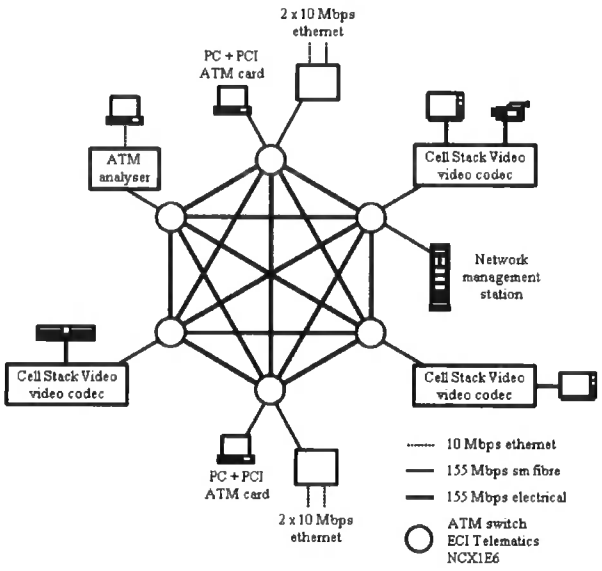


Figure 4.2: Diagram of ATM Test-bed

4.2.2 BT's Zeus Agent Building Toolkit

In searching for a toolkit to aid the building of the agent society, a great many candidates were discovered. Most of these were developed for specific research projects and as such were not adequately documented or supported. However the BT Zeus Agent Toolkit, available from website [b] seemed not to suffer from these difficulties in that it

was freely available and received ongoing support. It is also written in Java, which is useful given the mix of operating systems and machines available and the platform independence that Java provides.

Because of the difficulty of development of any software on the ATM switches, including installing a Java Virtual Machine, the PCs were used as platforms to run the agents. A low level agent then controls the switches remotely. This does not affect the functionality, as the agents can control all aspects of the switches from the PCs, but it does allow the use of a friendlier front end and Graphical User Interface (GUI). It also provides an interesting aspect of remote control of high-speed networks.

4.3 Agent Society Structure

This section describes the overall design of the Agent System, before the individual agents are described in the next section. Firstly, it is helpful to describe some of the analogies used to design the system. This then leads on to the general rules used to decide what an agent's role and responsibilities are.

4.3.1 Analogies for Design of Society

In developing a structure on which to base the hierarchy of the Agent Society, a number of issues became apparent. It also became obvious that these sorts of issues have been tackled before, but approached with different objectives. In designing a system to control a network, it is useful to consider the hierarchy of the network hardware and software itself.

Because of the inherently complex nature of the domain, protocols are designed to fill a particular level of the hierarchy. The interfaces between the levels of the structure are well defined, so that any protocol can be used with any other. One such model is the ISO OSI model discussed in Chapter 2.

The model provides defined interfaces and levels of abstraction to enable communication protocols to be developed which are interoperable with a generic higher and lower layer; effectively introducing platform independence.

Each layer in the stack considers increasingly wider-area issues, when considering an ascending order. The lower layers only consider the state of the links and nodes directly connected to the agent's node. However, higher level agents consider end-to-end connectivity with other nodes at arbitrary distance, by any metric. This is demonstrated by the most well known protocol, not in itself adhering to the OSI model, but to its own simplified structure, the TCP/IP suite [Stevens 1994]. Here, the lower level protocol (IP) concentrates on routing packets on a hop-by-hop basis, while TCP ensures that all packets arrive at the other end. The two models are shown in Figure 4.3.

<table>
<tr><th>OSI Model</th><th>TCP Model</th><th colspan="2">Example Protocols</th></tr>
<tr><td></td><td>Application Layer</td><td colspan="2"></td></tr>
<tr><td>Transport Layer</td><td>Transport Layer</td><td>TCP</td><td>UDP</td></tr>
<tr><td>Network Layer</td><td>Internet Layer</td><td colspan="2">IP</td></tr>
<tr><td>Link Layer</td><td rowspan="2">Physical Layer</td><td rowspan="2">Ethernet</td><td>AAL</td></tr>
<tr><td rowspan="2">Physical Layer</td><td>ATM</td></tr>
<tr><td colspan="2">Copper, Fibre</td></tr>
</table>

Figure 4.3 Diagram to Compare OSI model and TCP/IP models

The lowest level of the OSI model is the physical layer, which concerns the wires, symbol representation etc., so in the management structure an agent is present to represent and control the physical switch. It must carry the status of the switch (on-line/off-line etc.) and is responsible for the communication with it; this agent will be called the switch control Agent.

When it is established that the switch is operational, an agent is required to handle the next layer of the model the link layer. In this system, therefore, an agent is employed to maintain the status of all the links from a node, called the neighbour discovery agent. This agent assumes that the switch is on-line, as it is abstracted from such concerns by

the switch control agent; it merely needs to find out what links exist, and what state they are in.

Once all the links are known, then the system must find out what is at the other end of those links. This will be handled by another agent, the node agent. Again, it can assume that the links are on-line, since it has been supplied the list by the neighbour discovery agent, which monitors them constantly. The node agent interrogates any node agents which are directly connected to this switch.

Finally, an agent can be used to look further afield. By communicating with the nodes described to it by the node agent, it can exchange information as to the topology of the network to which it is connected. The agent is therefore dependant on the other agents to inform it of the other nodes.

An alternative analogy with which to view the system is provided by the time-scales involved at each level of a structure. The agents running at the lowest level, will work on a faster time scale than those higher up the hierarchy; indeed the lowest level Agents will be required to be dedicated to the task. The layered structure will help to gradate (ease in) between the fast, reactive agents at the bottom, nearest to the hardware, up to the more proactive, slower agents at the higher layers.

The higher layers will enable more adaptive behaviour, according to long-term trends, while monitoring the actions of the lower layer agents; this is a similar idea to that of Subsumption Architecture proposed by [Brooks 1986] and discussed in [Hayzelden 1999].

As well as there being analogies to the time-constants involved, there is a relationship with how local the issues are that is, an agent near the bottom must handle the state of each link from a node, which allows a higher agent to deal with a node at the other end of the link. Eventually an agent near the top can deal with end-to-end connectivity.

The structures described above provide useful parallels for determining the scope of

each agent in the society. These analogies allow a model that is layered with the aim of providing increasing levels of abstraction from the physical interface. In particular reference to the OSI model, the bottom three layers (the Physical layer, the Link layer and the Network layer) can easily be represented in our initial structure by one agent each.

In designing the agents, it was found useful to use the "Sphere of Responsibility" (SoR) test, described by [Collis 2000]. This simply says that an agent should only be responsible for a single decision.

4.3.2 Communication Between Agents and Societies

It is the ability of agents to communicate not only between agents within a society on a node, but also with other agents in other societies on other nodes that makes this approach so potentially powerful. This will allow a hybrid solution between fully-distributed and centralised management which is much more likely to be an optimal solution than either of the two extremes.

The ability to communicate enables the production of network-wide strategies that take into account conditions in all parts of the network which are impossible where no such communications exist. The layering of the agents within a society also means that each agent need only be aware of the agent layer directly above and below, simplifying the interactions (although there will be a need for "utility agents" to provide services which 'glue' the whole society together).

In the initial work, agents will only be able to communicate directly up or down the stack of agents, or horizontally to agents occupying the same layer in a different society. This is used in this system so that the Neighbour Discovery Agent can discover the node directly connected, and so that the Network Agent can exchange information to find out about the network as a whole. The resulting agent society is shown in Figure 4.4, while the roles of these agents are discussed in the next section.

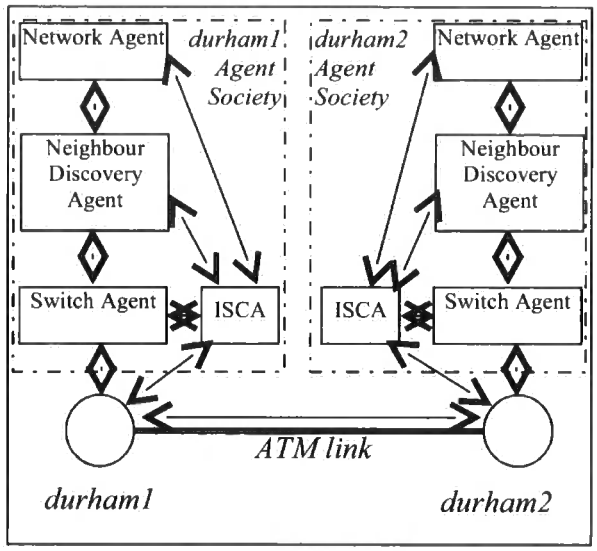


Figure 4.4: Diagram of Two Agent Societies

4.4 Outline of the Agents

This section describes the roles and responsibilities of each agent and is in order of ascending through the hierarchy. Following the agents already briefly outlined there are some other agents required for the system to function; for example an agent to act as postmaster for the node in communicating with other nodes.

4.4.1 Switch Control Agent

The Switch Control Agent provides access to the hardware resources in the switch. This enables both the discovery and configuration of switch resources, including, for example, recognising the presence of links. The agent keeps a record of what physical interfaces are present, whether they have been initialised, and a reference to the control channel used for communication on that interface which will be used for communication with the node at the other end of that link. When the agent has established communication with the switch, a message is sent to the Inter-Society-Communication Agent (ISCA, see below) to start listening for messages to the agent

society.

4.4.2 Neighbour Discovery Agent

The first task that the agent society must do is to discover the immediate connections to the host node. It was decided that a single agent on each node should handle this, and store the results. The information stored about the link by this agent is the destination node and the local port numbers at either end.

The agent has a refresh function, in case other nodes come on-line at a later point, this is manually triggered currently, but should be an automatic function depending on a time-out; however, it is this time-out that will greatly affect the amount of Control Traffic generated.

4.4.3 Network Agent

When the immediate neighbours are known, the next stage is to attempt to discover any other nodes on the network. This is done by exchanging knowledge with the nodes directly connected. In this way, knowledge of the network ripples across the network; if the agents are all refreshing at the same rate, each refresh will spread knowledge of a node by one hop. This raises the question of scalability, however without a hierarchical network structure this is inevitable.

This process effectively compiles a routing table. The agent can then make decisions on the best route to get to a particular node. The information stored in the routing table is the name of the destination node, the local port number to use and some representation of the distance to it.

It is obvious at this point that implementation described thus far has built-up to implement a Distance Vector Algorithm, or DVA. Although a DVA is seen as weaker than the alternative Link State Algorithm (LSA) it is considerably simpler; this is more fully discussed in Chapter 2.

The agent is designed to store a primary route and a back-up route in case of failure; the primary route having the lowest metric. Having a back-up route offers a balance between resilience in the event of failure, against the flooding of the network with control traffic in finding every possible route between two points. The time-out period for refreshing is another finely balanced decision, and is discussed in [Shaikh 2001] but for the purposes of this work is user prompted. When the agent is given control of this timing, a much greater level of autonomy will have been achieved, according to the definitions of agents in [Wooldridge 1997] and [Gaïti 1996b] and this is certainly intended.

Presently only peer-to-peer communication is allowed between societies; and there seems no reason to break this convention, as this simplifies the communication. This means that communication can only occur vertically - up and down the "stack" within the same society (or horizontally) to the same layer in another society. The ability to communicate with other agent societies (on adjoining nodes) requires a further agent, described next.

4.4.4 Inter-Society Communication Agent

The Inter-Society Communication Agent (ISCA) does not represent a layer of the ISO model, as it is really a utility agent, one that provides infrastructure to the society as a whole. Similarly to the Switch Agent, it maintains a connection to the ATM switch, across which messages to other societies are directed, and then forwarded by the switch on the appropriate link. This ISCA-based communication takes place over the ATM network itself.

Although an Ethernet connection is available it was felt that using this for inter-society communication was not in the spirit of the research, so the ATM links are used. Each agent across the network is uniquely identifiable by its name and the society it belongs to; thus giving it an address such as network@durham1. This is analogous to a normal email address. Figure 4.5 shows the path of a message from *network@durham1* to *network@durham2*. The overall route of the message between the two Network Agents

is shown by the dotted arrowed line, whilst the actual route is shown by the solid arrowed lines. Note that the message must pass across the ATM-link itself, shown by a solid black line.

The use of the ATM links for inter-society communications allows us to measure the aggregate Control Traffic using an ATM traffic analyser. It is also important to keep the communications to within existing protocols; by employing a standard-compliant Agent Communication Language (ACL) and not relying on a proprietary implementation; this is aided by the use of an off-the-shelf toolkit (Zeus).

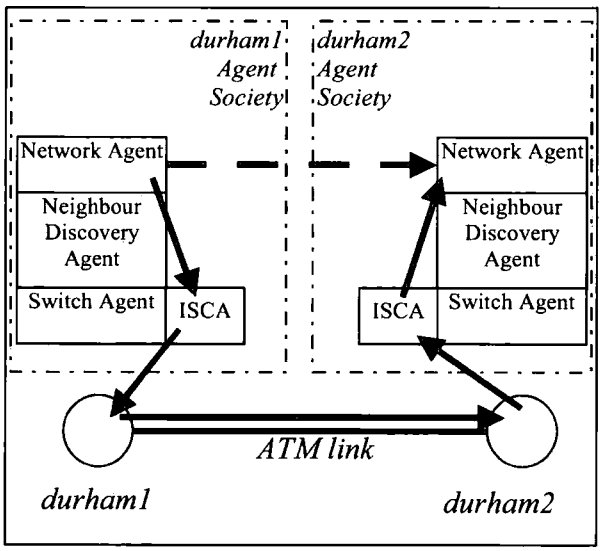


Figure 4.5: Diagram of Inter-Society Communication

4.5 System Development

Having specified the role and function of each agent within the society, this section now describes the development process and the testing procedures within that. The process of development was carried out under the iterative or prototyping model of software development [Somerville 2004]. Testing was performed throughout the procedure to identify bugs as early as possible. Rather than writing a full implementation before testing, the individual components were tested as soon as they were executable in their

own right.

This defined the order in which agents were developed i.e. Lower layer first. The structure of this section is that firstly the formal definition of the software is presented, before the testing procedures are described in more detail.

4.5.1 Agent Interfaces

The following paragraphs outline the structure and framework used in implementing the agents. Rather than listing the entire code, which would not be particularly instructive, representative Java Interface files are presented (similar to a header file in C++). Because interface files define only what methods an external object can access, only public methods are listed. Private methods are not included in Interface declarations as they are regarded as implementation specific. There is no implementation in an interface file, and so the methods are listed, followed by a semicolon instead of their bodies. Constructor methods cannot be defined in Interfaces, but were present in the implementations.

In the implementation of each agent a number of methods are inherited from the Zeus Toolkit. These provide inter-agent communication, databases, logical-rules etc. The most important of these in terms of this research is the provision of Rule and Fact Databases. The Fact Database represents the possessions of an agent, which in this case consist of the links from a node for example.

Since the Agents are self-contained from one another, but must work as a team, just as with protocol stacks, described in previous chapters, the interactions between the agents are well defined; both their interactions and agents roles are discussed in the previous section.

Figure 4.6 shows the interface declarations of the Switch Agent; this is made up three classes:

- The SwitchGUI class, which handles the interaction of the user with the

Agent,

- The SwitchInterface class which composes the messages for the switch,
- The SwitchSocket class which handles the TCP/IP socket connection.

```
public interface SwitchGUI extends JFrame implements ZeusExternal,
FactMonitor, ActionListener, WindowListener
{
// Methods prompted by buttons
  public void connectAction () ;
  public void disconnectAction () ;
  public void autoAction () ;
  public void pingAction () ;
//Methods prompted by other agents
  public void whoAction (int thisPort) ;
  public void onlineAction () ;

// Zeus Methods
  public void exec(AgentContext context) ;
  public void factAccessedEvent(FactEvent fe) ;
...
  public void windowOpened(WindowEvent event) ;
  public void windowClosing(WindowEvent event) ;
}

public interface SwitchInterface
{
   public static int connect( String IP_ADDRESS ) ;
   public static int disConnect( String IP_ADDRESS ) ;
   public static int sendPing ( StringBuffer MESSAGE_STRING ) ;
   public static int sendCardOnline( int[][] CARD_ARRAY ) ;
   public static int sendWhoAreYou ( int portnumber,
        nodeAndPort nodePort ) ;
}

public interface SwitchSocket
{
   public static int connect(String IP_ADDRESS) ;
   public static int disConnect(String IP_ADDRESS) ;
   public static int send( StringBuffer MESSAGE_STRING )
       throws IOException ;
}
```

Figure 4.6: Definition of Switch Agent Interfaces

Within the SwitchGUI class, it can be seen that the user has the option of connecting (through connectAction) and disconnecting (disconnectAction) manually, or allowing the Agent to run autonomously (autoAction). Finally there is an option to *ping* the switch for testing purposes (pingAction), which sends a TCP ICMP packet, expecting one back to confirm connectivity with the switch.

Along with the user-driven actions, triggered by traditional method calls, there are also actions requested by other agents. Zeus provides the ACL (Agent Communication Language) protocols, and through this the agents can request actions rather than use direct method calls. This keeps to the philosophy of Agent-Oriented programming; keeping the agents autonomous, modular and cohesive.

The Link Agent uses the Zeus communication process to send a request to the Switch Agent to query the status of all the possible ports on the switch. The Switch agent runs the *onlineAction* method declared in the SwitchGUI interface of Figure 4.6, which performs this query by running the sendCardOnline method defined in the SwitchInterface interface. When this is done, the *whoAction* method is used to establish the nearest neighbours by running the sendWhoAreYou method in the SwitchInterface interface. The WhoAreYou message queries the identity of the node at the other end of the link.

All of these actions rely on the SwitchSocket class which constructs messages to be sent, but leaves the socket establishment procedures to the SwitchInterface class. The SwitchSocket class only has three methods; connect, disConnect and send, which are self-explanatory. All the communication with the switch is prompted by the local agent so no process is required to listen at other times (the agent only performs synchronous communication so will never get an unexpected message).

Figure 4.7 shows the interface declaration for the Link Agent. The methods are user prompted, and the agent can be started off (startAction), stopped (stopAction), or told to refresh (refreshAction). Also included, for testing purposes, was a method to send a message (messageAction) which proved connectivity with the neighbouring nodes.

Once started, the Link Agent will query the Switch Agent (using the message passing provision of Zeus) to establish the available links, and then the identity of the nodes at the other end of those links. The agent will store the existing links in a Fact Database along with their status. The Fact and Rule Database, also provided by Zeus, can be used to trigger actions. For example, once the Link Agent knows of a link, but not the other

node connected, a query will be triggered to find the other node.

```
public interface LinkGUI extends JFrame implements ZeusExternal,
FactMonitor, ActionListener, WindowListener
{
// Agent Action Methods
  public void startAction () ;
  public void stopAction () ;
  public void refreshAction () ;
  public void messageAction () ;

// Zeus Methods
  public void exec(AgentContext context) ;
...
  public void windowClosing(WindowEvent event) ;
}
```

Figure 4.7: Definition of Link Agent Interfaces

The Rule-Base is developed in concert with the Java classes. Figure 4.8 shows an example of a rule used in the system. These Rulebases are collected into sets, and can be used in more than one agent. The rule shown, as its name suggests, is used by an agent to establish communication with the agent in the next layer down the hierarchy. The prefix of a "?" denotes a variable, which can be stored and manipulated later, whilst those without a "?" force positive matches for the rule to be fired.

The rule in English states:

IF there is

- (i) a fact of type *agentsName* currently in the fact-Database, and its *switchName* component is set to "null",

AND

- (ii) a fact of type *otherAgent* currently in the fact-Database, and its *state* component is set to "register"

THEN

- (a) modify the *switchName* to "registering"
- (b) send a message to the agent in the lower-layer with this agent's name

The lines before and after the send message command, merely cut and paste the correct names into the message and then change it back.

```
(RegisterWithLowerLayer
  ?agName <- ( agentsName ( switchName null ) ( agentName ?aN ) )
  ?oA <- ( otherAgent ( state register ) ( agentName ?lowerlayer ) )
 =>
  ( modify ?agName ( switchName registering ) )
  ( modify ?oA ( agentName ?aN ) )
  ( send_message ( sender ?aN ) ( type inform ) ( content ?oA )
    ( receiver ?lowerlayer ) )
  ( modify ?oA ( state registering ) ( agentName ?lowerlayer) )
)
```

Figure 4.8: Example of a Zeus RuleBase Rule

Figure 4.9 shows the interface declaration of the Network Agent. The methods available are all currently run at the prompt of the user pressing a button on the GUI and are to start (startAction) and stop (stopAction) the agent, as well as a refresh function (refreshAction) discussed earlier. Further to this, an ISCA-message can be sent for testing (messageAction), and a print-out of the current routing table (printAction) can be prompted.

```
public interface NetworkGUI extends JFrame implements ZeusExternal,
FactMonitor, ActionListener, WindowListener
{
// Agent Action Methods
  public void startAction () ;
  public void stopAction () ;
  public void refreshAction () ;
  public void messageAction () ;
  public void printAction () ;

// Zeus Methods
  public void exec(AgentContext context) ;
...
  public void windowClosing(WindowEvent event) ;
}
```

Figure 4.9: Definition of Network Agent Interfaces

Figure 4.10 outlines the methods implemented in the ISCA-Agent. There are only two main methods, an incoming message (incomingMessage) and an outgoing message (outgoingMessage), but also a test method, (AreYouThereAction) which tests that messages can be passed between nodes. Since the communication in this case can be prompted by any other node directly connected, a SocketListener must be established with its own thread of control to enable incoming messages to be received at any time. This has but two methods - one to start the thread (run) and one to stop it cleanly

(pleaseStop).

```
public interface ISCAGUI extends JFrame implements ZeusExternal,
FactMonitor, ActionListener, WindowListener
 {
// Agent Action Methods
  void AreYouThereAction () ;
  static public int incomingMessage ( StringBuffer MESSAGE_STRING ) ;
  static public int outgoingMessage ( Fact fact, int chanNum ) ;

// Zeus Methods
  public void exec(AgentContext context) ;
...
  public void windowClosing(WindowEvent event) ;

}

class ISCASocketListener extends Thread {
   public void run() ;
   public void pleaseStop() ;

}
```

Figure 4.10: Definition of ISCA Agent Interfaces

4.5.2 Testing Procedures

The importance of a methodical testing procedure is to establish confidence in the results. Although it is impossible to prove the absence of software faults, by comparing pre-defined or pre-calculated expected results with actual results it is possible to establish sufficient confidence in software to use it, and rely on results produced by it.

Some of the strengths of Agent Based Software Engineering are the development of the concepts established by Object Oriented Software Engineering. One such concept is that of modularity; the design of the system described in this Chapter is highly modular, with agents relying on the facilities provided by those agents below them to already be present (for example, control of the network connections on which they rely).

This fact means that, during development, the testing process falls naturally into White-Box testing as an agent is developed, but then Black-Box testing as the next agent is developed since the second agent relies on the first.

Black Box, or Functional, Testing is when only the output of the module is evaluated - the module itself is considered a Black-Box (hence the name). This is to test whether over the whole range of inputs the correct output is given. It uses the same level of abstraction as the Interface definitions shown earlier - the implementation used is not important, only the correct result.

White Box, or Structural, Testing is the opposite, investigating *inside the box*, and analyses the code itself. This itself breaks down into static techniques (examining the code) and dynamic techniques (which exercises the code).

Each agent was developed incrementally, with the code tested from early on, and then at relatively small steps - there were few constraints like cost of testing or hardware availability, just the time taken, which was short in itself. This repeated testing meant that any failures were identified and corrected quickly, as part of small iterations.

An example was in the process of enabling communication between the agents. This required use of TCP-sockets, and the incorrect command caused the computer to become unresponsive, as a hardware timeout was occurring. Because there was no distinction between the development and the testing phase, until acceptance testing at the end of the process it is difficult to quantify the number of errors. Perhaps more descriptive is that the number of compilations of the software ran into the low thousands. Each time provided some improvement in the code, added functionality and indeed the identification of of errors.

4.5.3 Unit Testing

The following paragraphs describe testing procedures unique to each Agent, which can be seen as an individual unit, or component.

The first unit to be developed, the Switch Control Agent, handles a TCP/IP socket and interfaces with the network node. A particular control protocol was in existence on the

node, so message formats were already defined. The agent had to have the full lexicon of the protocol available for the use of agents higher up the hierarchy.

The Zeus Toolkit provided a number of tools to visualise the workings of the agents. This included a listing of the all the communication between the agents - each having its own inbox and sent-mail - as well as the Facts the agent possessed. In this case, the agent kept appropriate records of its activities; e.g. the Link Agent keeps a list of available links in its fact database, the Network Agent, a routing table.

The fact database can be viewed through the Zeus GUI to confirm the agent has the correct level of knowledge expected at runtime. Confirming that messages were received when expected and that the content was correct were part of the testing procedures.

The Link Agent, was required to establish communication and use the facilities provided by the Switch Agent. To this end, the agent communication was developed within the structure of the Zeus toolkit. Written generically, this could be re-used in the communication between pairs of agents within the hierarchy. The rule presented in Figure 4.8 is part of that process infrastructure. Indeed in the case of the Link Agent, the roles are reversed in the establishment of communications with the Network Agent.

The testing involved here was by regular execution of the code throughout development. This meant that bugs were picked up early in the development, at which time they are easier to trace and remedy.

Once the communication process was established as sufficiently reliable for normal operation, the Link Agent could then request the Switch Agent to query the hardware, as described earlier. The testing involved here was to ensure the Link Agent had a full and correct list of the hardware's physical connections.

The ISC-Agent is very similar to the Switch Agent, in that it controlled a socket. The difference being that the communication involved was less well structured - messages

could arrive at any time. A new thread of control was required to listen on the socket to handle incoming messages. When a message was received, it was added to the fact Database of the Agent. A destination field included in the message was used to forward the message to the relevant agent.

The Network Agent, again, relied on those agents below it. Testing of the agent used both the Link, (and therefore Switch) and ISC-Agent as black-boxes. The result was a routing table. The Network Agent must communicate with every node it is connected to, to establish routes with every other node available.

4.5.4 System Testing

The system was constantly under test during development - using an increasing number of agents as they were developed. This ensured that there was the intended interoperability at all stages - given the dependence of higher-layer agents on those lower in the hierarchy, this was critical to the success of the system as a whole.

The system as a whole was tested in many different configurations, firstly between just two nodes, but then with a larger number. The next section describes an example of a four-node network which is designed to be generic.

The system was designed to be stable, so that the system could be started, stopped and restarted an arbitrary number of times. This level of resiliency was required to handle a node-failure or an Agent failure, but also to show the agents could remain in the environment to handle failure, and not be merely single-use.

4.6 Full System Result

This section presents and describes the results obtained. Firstly the network to be tested is described and then the results are presented in this case the output of the top-level

agent at each node. Then follows a detailed discussion of the results, including descriptions of contributing factors to the system's performance.

4.6.1 Test Network

Figure 4.11 shows the network topology being used to confirm the system works. The network is designed to show some alternative backup routes, but also have some unique routes.

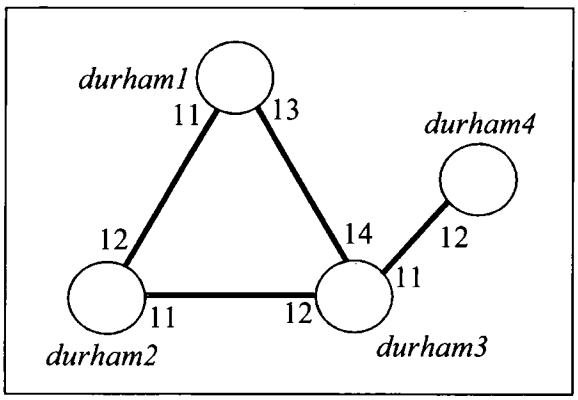


Figure 4.11: Topology of the Network Tested. The node-names and local port numbers are shown.

4.6.2 Agent Society

Figure 4.12 shows a screenshot of the Agent Society front-end GUIs (Graphical User Interface) controlling the *durham3* ATM switch. One of these societies is required for each switch being used and controlled.

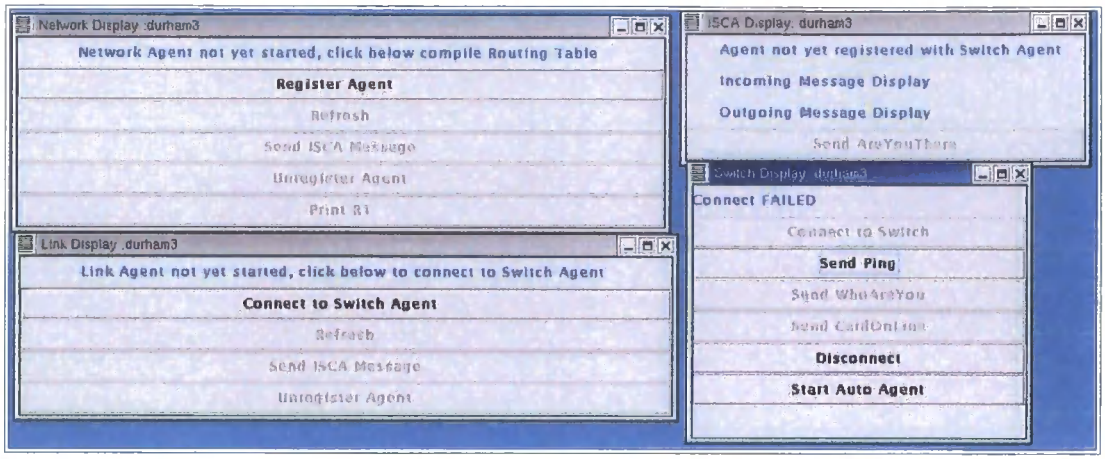


Figure 4.12: Screen Shot of Agent Society

4.6.3 Output of Agents

Figures 4.13 to 4.16 show the outputs at each node. As can be seen in Figure 4.11, all traffic for *durham4* must pass through *durham3* so any backup routes are not node or link-disjoint. The routing tables for *durham1* and *durham2* list each other as backup routes if the direct route to *durham3* fails. Nothing can be done by any routing system if the *durham3-durham4* link is down, however, if it is the *durham1-durham3* or *durham2-durham3* link that is the problem, then at least the routing system will try that. If no Backup route was provided, the system would not attempt to deliver the traffic at all.

node	port	metric	altport	altmetric
durham2	11	1	13	2
durham3	13	1	11	2
durham4	13	2	11	3

Figure 4.13: Output of Network Agent on *durham1*

node	port	metric	altport	altmetric
durham1	12	1	11	2
durham3	11	1	12	2
durham4	11	2	12	3

Figure 4.14: Output of Network Agent on *durham2*

node	port	metric	altport	altmetric
durham1	14	1	12	1
durham2	12	1	14	2
durham4	11	1	0	100

Figure 4.15: Output of Network Agent on *durham3*

node	port	metric	altport	altmetric
durham1	12	2	0	100
durham2	12	2	0	100
durham3	12	1	0	100

Figure 4.16: Output of Network Agent on *durham4*

Figures 4.13, 4.14, 4.15, and 4.16 show the routing table produced at each node. As can be seen each node learns of all the nodes on the network, not just those that it is directly connected to. It also discovers a backup route (prefixed with "alt"); although this can share links with the primary route, they do diverge somewhere. Where the metric is "100", no back-up route has been discovered.

4.6.4 Discussion of Results

The results presented here demonstrate the concept and feasibility of the system. The system was written for strict functionality rather than speed of performance at this stage which would be required to be optimised in a full system. Exact timings of how long the system took to configure itself were not possible, nor would they have meaning anyway, for the reasons laid out below.

The Ethernet connection meant that the system is not designed to perform to hard-real-time constraints, since the network access mechanism is stochastic, rather than deterministic, which would be used in hard time-constraint situations. Having stated that, it is an interesting facet that the agents are running remotely.

The Zeus environment also adds a burden to the processor, since it is designed to run GUIs and allow the user to examine the state of the agents databases etc. rather than

being designed for speed and efficiency of operation.

Finally, Java is an interpreted language. This means that the program must be compiled at run-time. Whilst this means that the code is highly portable, it is slightly slower in principle than dedicated code.

4.7 Context for Following Work

It has been proposed that by employing a society of autonomous software agents on each node, an effective Network Management system can be developed. A natural structure to this society is provided by considering the increasing geographical scale and timescales an agent must consider when ascending the stack the lower members must handle the state of individual links, while higher members can consider End-to-End connectivity; this is analogous to the ISO-OSI model.

Although the work described above is a working system, the intended functionality extends considerably further than that currently implemented. The aim of the project is to harness the advantages of agent-based software to manage and control a telecommunications network. These advantages include the ability to work autonomously, adapt to their environment and learn from situations that arise.

So far this chapter has described such a system that has been developed, and has been demonstrated to work. The next stage holds many possibilities, and these are discussed in this final section of the chapter which is provided to give background into the development of the research and the reasons for undertaking it. Firstly, possibilities for further agents in the same structure are discussed.

There are many different possible agents to complement those already described; these include ones to manage the situation of a failure in the network, and one to handle the mapping of IP over ATM. These are described next.

4.7.1 Failure Recovery Agent

An interesting situation is presented when a node or link fails. In this case rather than allowing knowledge of the failure to ripple through the network, it will be necessary to notify nodes explicitly. There are two possible situations here:

Node Failure: If a node fails then the routing tables need to be amended, to remove the failed node. The source node of any traffic destined for the failed node should be notified, and all traffic routed via that node needs to be re-routed around it. Note that the failure of a node could be seen as being equivalent to multiple link failure.

Link Failure: In the event of link failure, the traffic must be routed round the problem. Depending on the topology of the network, a link failure could isolate part of the network, making it equivalent to a (multiple) node failure.

Whichever situation it is, traffic will have to be re-routed. Initially, the furthest downstream node (i.e. next to the failure) must consult its routing table, and attempt to set up connections for the traffic around the problem. If the node can't find an alternative route, the responsibility passes back to the previous node which will try again itself. Cases where traffic ends up doubling back on itself need to be detected, and corrected.

This re-routing is a similar function to a normal routing algorithm, but there is the additional problem of dealing with traffic flows already using the network, and so having an existing traffic contract. Although a failure could be seen as a *force majeure*, the network should try to re-route if at all possible.

To achieve all of this, the Failure Recovery Agent will have to propagate failure messages to update routing tables. It must then handle the attempts at re-routing. The failure of a major link could have large knock-on effects across the network. The agents across the network will have to adapt to the new conditions, which will possibly affect many routes.

4.7.2 DiffServ Agent

Originally, it was intended that ATM would be *the* ubiquitous protocol, going straight to the desktop PC. However, with the fast growth of the Internet[1], ATM has found its niche at the backbone level of networks (i.e. high speed trunk routes).

As stated above, the Internet has very little in the way of QoS provision. There are attempts to patch some in however. One of them, the most likely to succeed, is known as Differentiated Services, or DiffServ[2] [Bernet 2002].

DiffServ associates IP packets with a number of traffic classes using "code points". The code point is identified in the IP header and the mapping of this to a traffic class and therefore QoS requirements is domain dependant. The code point aggregates traffic with similar QoS requirements. At the boundary of an ATM core network designed to carry IP traffic, decisions must be made on how to map DiffServ traffic to ATM resource requirements and hence routes.

The use of agents would allow this process to adapt to changing traffic patterns and resource availability within the core network.

4.7.3 Link State Routing Agent

The Agent System, as it stands, implements a Distance Vector Routing Algorithm. A development of this position would be to extend the system to a Link State Algorithm, LSA; this is a significantly more resilient and scalable routing system. Whilst there are good reasons for implementing an LAS, the work itself would require significant effort.

The IETF RFC describing the LSA (OSPF) protocol requires over two hundred pages, and at this stage little extra information would be gained from such a scheme in terms of having proved a concept of an agent-based network management system.

1 Which is one *very* large TCP/IP network.

2 Also referred to as DS.

4.7.4 Ant-Routing System Agent

The system as it stands performs topology discovery. A natural extension of this is the introduction of a more advanced routing system on top of this. The system as implemented represents a Distance-Vector Algorithm, which is the more basic of the two types of dynamic routing algorithms, the other being Link-State; these are both discussed in Chapter 2.

The metric used in the system was simply the number of hops between two nodes; whilst this is the best simple metric, better metrics can be described with some added sophistication.

Ant-Based Routing, as discussed in Chapter 3, and described in [Schoonderwoerd et al 1996, 1997], [Bonabeau et al. 1999] amongst others represents a method of providing a more sophisticated metric, and has been shown to perform Load-Balancing as well as finding the shortest route.

This then would be a natural extension to the system developed here, and indeed an Ant system must know of all the nodes present on a network before it can start; this information can be provided by the existing system. In return, the Ant-System can provide metrics to be used by the Routing Agent to control the network.

4.7.5 Constraints on Further Work

Whilst the system has many potential further avenues of research, it is constrained by the bareness of the existing management system on the ATM Testbed. The lack of this system would entail colossal effort to take the current developments any further.

Whilst this means the end of this line of investigation, it does not mean the work has been fruitless. The novel approach described here may well make the management system appreciably simpler to design and build, along with the potential for autonomy of configuration.

The way in which the system can be extended "off-line" is that the concept of agents working at particular layer of a modular management system, allows a level of abstraction in designing a higher layer. In this sense, the next layer can be designed knowing that the underlying layers already work. All the next layer must do is conform to the interface provided by the uppermost existing layer these are the same concepts as Object Oriented software attempts to address abstraction, encapsulation, modularity etc.

Of the areas that have already been identified in this section, the DiffServ Agent and the Network Recovery Agent would require the management functions not available on the ATM-Testbed so the project is infeasible. As for the Ant-Based Routing, although to implement the system on the ATM Testbed would require the management system, it would be possible to simulate the work so this would not be a problem.

4.8 Chapter Summary

In this chapter a system has been developed that automatically establishes all the connections available at a node, and then compiles a routing table of all nodes it can learn about. This can only be achieved by one node co-operating with its surrounding nodes. This is a first basic step towards network configuration and provides the framework for a more adaptive and dynamic network control system. There is potential for a number of other agents, including more proactive, strategic-planning agents (e.g. Failure Recovery etc.) which would fit into the framework, as already built. These have been discussed and the more realistic possibilities highlighted.

An aim of this work has been to build a fully operational network control system capable of running autonomously, and able to adapt to changing circumstances. An emphasis was placed on achieving this through using off-the-shelf components, to ensure compliance to general agent standards.

In the next Chapter, the Ant-Based Routing System is explored and developed in great detail, for reasons described previously. Although systems similar to this have already

been simulated, the specific implementation is novel. The final Chapter of novel work describes the addition of an Agent monitoring the Ant-system with the aim of taking strategic decisions to improve their performance, and adapt to rapidly changing conditions on the network.

Chapter 5:Ant-Based Routing System

This chapter describes the implementation of the basic Ant-System. First, the major issues involved in designing the Ant-System are outlined before the software tools required are described; these include the telecommunications network simulator used, as well as the programming languages behind them. Then the Ant-Based Routing system can be fully described. A further section describes the testing of routing parameters to find the optimum settings followed by a section on the testing of the final configuration as a standalone routing system. The final part of this chapter introduces the motivations which are then expanded in the next chapter.

Although Ant-Based Routing has been implemented before, this instance is a unique implementation, blending ideas from previous attempts and new ideas from the author, as well as using a standard telecommunications simulator to validate the work.

5.1 Ant-Based Routing: A System Inspired by Nature

There are many different species of ants, and they come in many different sizes and colours; their behaviour ranges from the benign to the extremely aggressive. Many of these species use a simple technique to navigate their way around, and learn the best route between their food and a food source. It is this behaviour that has this inspired this, and other work[1].

Ant colonies are constantly exploring the surroundings of the nest looking for food - some outliers can travel significant distances from the nest. As they travel, the ants are constantly leaving a trail of chemical hormones called pheromones. This chemical can be smelt by the ant and allows it to retrace its steps.

1. See Chapter 3.

If an exploring ant discovers a food source it returns to the nest laying a strong trail of the pheromone, and can communicate with other ants telling them there is a food source to be plundered is a similar process to that of bees, where the phenomena of "waggledances" are observed to alert the rest of the colony to a new source of nectar.

As other ants travel towards the food, they may lay more pheromone and so strengthen the trail to induce more ants to follow[2]. This process means that ants are extremely successful at exploiting sources of food in their environment, but that is not the full extent to their apparent intelligence.

If an obstacle falls across the path while this exploitation is taking place, it is observed that the ants, very quickly, find a way round the obstacle. But not only that, they find the shortest way around it. This is because they do not lay a constant amount of pheromone as they travel, but slowly reduce the rate of deposition. This means that a gradient is formed which can be followed; in the example of the obstacle, this would mean that the ants travelling the shorter path would be laying larger amounts of pheromone than those that travelled the longer path.

Whilst drawing on the inspiration of ants, it must be clear that the objective is an effective routing system. The rest of this chapter describes how the basic process is translated into a routing system for a telecommunications network that produces an optimal routing scheme.

5.2 The Ant-Based Routing System: Issues In The Design

This section describes the the Ant-System which is broadly based on those already discussed in previous sections. The beauty of Ant-Based Routing is the simplicity in the actions of each individual ant compared with the complexity of the behaviour of the "colony" as a whole. This Simple Micro-behaviour/Complex Macro-behaviour-makes

2. Each Ant species has its own set of rules, some lay on the way there, some on the way back, some in both directions.

the system relatively easy to program, and can result in an elegant solution.

At this point, some clarification of the terms used in the rest of the thesis is desirable, since there are many different models and analogies that could be used in a similar context.

Firstly, in the context of this thesis, the intelligence is held at the nodes of the networks, the *colonies*. The ants are the messages passed between the colonies. This is seen as more desirable than the alternative, which is to make the ants themselves intelligent by embedding executable code within them - this would greatly increase the amount of traffic generated by the system and processing burden at the nodes.

Further, to stay true to the concept behind the Ant-System, the ants should be kept as simple as possible. This approach also helps from the point of view of minimising control traffic.

Secondly, in this section the term "Probabilities" is used and discussed frequently. Some clarification is required of this term. Each ant that travels the network rewards the route it is using. This reward, or reinforcement, means that that route is more likely to be used by other ants in the future. This likelihood is expressed as a probability so that it can be compared with other available routes. Probability Tables, which could also be called Pheromone Tables, are used to store all this information for every route for each destination. Each different destination will require a separate set of probabilities.

The following sections describe some key issues relating to ant-based systems, and discuss the underlying principles on which the implementation is based.

5.2.1 Separation of Ants and Traffic

In this implementation the ants carry no data, and therefore they effectively increase the level of traffic on the network. This may seem counter-intuitive considering the previous discussions on the subject of Control Traffic. The reasons behind this decision

are set out below.

The scheme requires a large sample size of Ants using the network in order to achieve convergence and provide viable results order to minimise the additional network load, each ant is only a single packet and is kept as small as possible. An ant is sent out periodically from each node to a randomly selected destination.

The alternative to using separate transport for the ants and actual data is to superimpose ants onto the real traffic. There are a number of problems with this approach, including sparsity of information on lesser-used areas of network on top of the general restriction on the number of Ants possible, distortion of information by differing packet size and most importantly the speculative nature of some routes an Ant could take. These are all discussed in the following paragraphs.

To elaborate on the most important consideration, if a network accepts some traffic, whether it has QoS constraints or not, the network has an implied duty to make a "best-effort" attempt at transporting the traffic. Thus it must, if at all possible, send a packet on a known route (whether the actual best route or alternative due to congestion etc.).

An Ant-System essentially generates traffic which it routes experimentally. The effect of this is that all of the possible routes are measured and the optimum one identified. It would not be acceptable to send real data down a known dead-end route, nor indeed delay it unnecessarily. However, an ant-packet would not be mourned and could be artificially aged to achieve almost the same result. Another result of delaying an ant on a given route is that a shorter, less congested route could be found *quicker*; this would result in the shorter route being used more times before this longer route is used once - so the shorter route is reinforced considerably.

An interesting compromise would be to not only use traffic as Ants, but also issue extra non-data-Ants in the areas seen as sufficiently deficient in traffic volume for that to be a problem. This would have the advantage of only issuing extra control-traffic when/where necessary. However, this would accentuate the problems discussed in

relation to variable packet size - the data-ants would be significantly larger than the non-data-ants.

A further complication in measuring the response of a network to traffic results from variable sized packets. In ATM, for example, this is not a problem as all packets are split into fixed-sized cells (of 53 bytes) - however, few protocols are like this. The differing sizes of packets would affect the transmission delays as well the segmentation delays. It is worth noting that ATM was designed with fixed cells to increase the processing efficiency. The size of 48 payload bytes was a compromise between minimising the size for video streams - to minimise delay - and maximising absolute efficiency by increasing the number of payload bytes per overhead byte.

The extreme example of variable packet size is an instance of *jumbo-payload* in IP-networks - these are packets of 65,535 bytes, that take significant time to handle on each node and would distort the ant-statistics appreciably. The packet would only be attached the significance of a single ant, but would be associated with a large propagation time.

For clarity, the problem with variable ant-packet size is one of experimental control. Although it could be argued that using a variety of packet-sizes to measure the system makes the measurements more realistic, this would be left up to chance and the system measurements could be skewed by one particular application. By using standard sized ant-packets, the effects could be accounted for through calibration.

The *volume* of traffic traversing various parts of the network could also be a problem. It may be that some parts are heavily utilised (and so receive the attention of a large number of ants), while other parts are neglected as there is no traffic and therefore no ants; again noting that the scheme requires a significantly large sample size.

5.2.2 Backward Reinforcement

Although the solution of Ant-Based Control maps very well to the domain of communications networks, the largest flaw in the mapping is the problem concerning

Forward Reinforcement.

In the simple example of an Ant travelling from its nest to a food source, Forward Reinforcement would mean the ant was leaving pheromone as it travelled towards the food but before the outcome of that journey is known. Backwards Reinforcement would mean the Ant was dropping a trail of pheromone only as it returned to the nest. There is nothing to prevent using both types of reinforcement being employed.

Real ants exist in a continuous world where there exist an almost infinite number of paths between any two points, and pheromone can be deposited constantly along the route. Ants restrict *themselves* to particular paths by use of the pheromone, but with free will, or when exploring virgin territory, they could in principle wander anywhere.

On any (wired) communication network there is a limited number of paths, or links, available and it is only possible to reinforce routes at certain points - the nodes of the network.

This discretised world means that if ants were to reinforce in a forward direction they would do so blind, as the reinforcement would happen before the outcome of the next-hop was known. An ant could leave some pheromone to signify it had taken a particular link, and then perhaps be dropped as the queue at the next node was full. This could cause problems, as a route that should be punished is continually reinforced.

Reinforcing the link on arrival at the next node does not present this problem, but it is then the return route (i.e. in the opposite direction) that is reinforced. This is known as Backward Reinforcement. When used, there is an assumption that the return path is the same as the forward one. This means that the single direction of a route cannot be considered in isolation, but only in conjunction with its return route. However, this is also the case with OSPF, the recommended TCP/IP Routing Protocol, and RSVP.

An interesting observation is that by using Backward Reinforcement each node is effectively broadcasting its location out across the network using ants as the medium. The trails branch out and funnel any traffic into them and onto larger and larger routes.

5.2.3 Pheromone Trails and Evaporation

Pheromone is used by real ants to mark the route they have taken, for example between a food-source and their nest. The ants lay pheromone all the time, so they can find their way back, but is laid in greater quantity when they find a food source. The more pheromone laid in one place, the stronger the trail, but there is an ageing/decaying effect at work also; the pheromone evaporates over time. Both of these effects can be represented in the network by storing the amount of pheromone present as a probability of a link being selected; each time an ant arrives the probability is increased by some amount, as would the pheromone.

Probabilities must by definition sum to unity at all times, thus when one route is reinforced, the sum of probabilities across all routes must be normalised to unity again. If one route is not reinforced for some time, while others are, then the process of normalisation means that the probability associated with that route is decreasing. This simulates the effect of pheromone evaporating, which as well as being what happens physically, has been deliberately modelled in previous implementations; it introduces a dynamic system and a smooth historical drop-off. The initial route probabilities must be set to some non-zero value. The simplest solution is to make all the routes equiprobable to begin with.

To ensure that each and every route is explored continuously a Minimum Probability is used. This is set to 5%, but must be used with care - in the case of a node having many links, there is a danger that the sum of the probabilities that have fallen to the minimum may constitute too significant a proportion of the total.

The method of pheromone trails highlights an interesting biological mechanism for social-memory. If normalisation were not used then over time the amount of pheromone would tend to infinity, and the effect of each individual ant would tend towards zero. This would result in a system with no adaptability.

Although it would be possible to use any number arbitrarily, or indeed not to enforce a constant sum, using such a scheme as described here aids the visualisation of the system (the numbers can be thought of as probabilities). Although the act of normalisation models the process of evaporation well, it could be argued that if no reinforcement occurred at all for some time then the evaporation analogy falls down. In this instance it would be possible to use an algorithm that slowly equalises all of the probabilities so that when ants return to the network they effectively start afresh. The sum of the probabilities are always maintained at one in the implementation. To allow any other total would by definition be probabilities and complicate the algorithm required. Enforcing a sum of one also ensures the sum does not decay to zero (in the case of a dearth of ants). If any route's probability was allowed to fall to zero, it would never be explored again. If the sum was to tend to zero then the reinforcement from a single ant would have a disproportionate affect.

It is the author's assertion that normalisation of the probabilities is analogous to pheromone evaporation. To implement further modelling of evaporation, e.g. by using a further updating mechanism to gradually equalise probabilities would add extensively to the complexity of the system. A far simpler mechanism, already incorporated, is to reset the probabilities to the original uniform distribution if necessary. Combined with the option of freezing the probabilities in the event of failure of either link or node the system is flexible enough to cope with different levels of failure. The further complexity of modelling pheromone evaporation precisely would not offer sufficient benefit to warrant it except if this was the specific area of study. There is a danger here that the balance between designing a system to control a telecommunications network that works and building a model of how ants work in the real world would be lost.

5.2.4 The Ant Decides (Randomly)

The decision by an ant as to which route to take is a weighted-random process. The probability table for the ant's destination is used, and these probabilities describe the likelihood that that link is used. The mechanics of this process are similar to that of rolling a (weighted) die and the result being a particular link to be used in this instance, but this is described in more detail in Section 5.4.1.

Each link has a probability of being used, and it is these probabilities that are changed by the movement of the ant-packets. For the purposes of a dynamic system, each possible link should have a definite (i.e. non-zero) chance of being selected. In [Schoonderwoerd 1996], this is achieved by the addition of "noise" to the random process; the mechanism used is to have a probability, f, that an ant takes a purely random route, with the probability (1 - f) that the pheromone table is consulted.

Instead, the approach used here is to set a minimum value that any probability can fall to. This guarantees that every route is periodically explored. It is worth noting that this minimum value should be dependant on the total number of links available on the node.

5.2.5 Requirement for Ant Ageing

A second ageing process is required in the system. The first, evaporation of the pheromone has already been discussed, but the ants must also have an ageing function themselves, meaning they lose their virility as they travel. Real ants vary the amount of pheromone left along their path depending on how far they have already travelled. This is translated in the implementation as the ageing-function of ants. Obviously, real ants have a lifetime far greater than a single journey so the analogy falls short here (but not detrimentally), but in the proposed system, an ant's lifetime is exactly one journey, however long that journey happens to be.

In the context of this discussion it is more useful to consider the ageing of an Ant in

terms of it starting off with some amount of pheromone, and using this up as it travels and this shall be discussed here. The alternative, is that the ant is able to produce pheromone as it travels, but as it ages, this rate decreases. It is unfortunate that "age" implies a property that *in*creases over time, however as discussed above, in this thesis a property is being used that *de*creases with time. However, it provides a more compact notation than the alternative, which is a combination of the *Initial Amount of Pheromone* and *Rate of Deposition.* Since it can be argued that the detail of the implementation remains true to the overall inspiration, Ant-Age shall be referred to, until it becomes unwieldy to do so.

The concept of the Ants ageing as they travel is an important one. It is this property that enables the Ant-System to differentiate between shorter and longer routes, as demonstrated in the next example.

Consider two ants. The first travelling a direct route, shorter than the second which takes a circuitous route. If the Ants deposited the same amount at each node along a route, then at the final destination node no advantage would be gained by the shorter route over the longer route. The same amount of pheromone had been left by both and the effects have been cancelled out.

However, if the ants were ageing so that the younger ant deposited more than the older ant at the destination node, then the net effect of the two is a slightly higher probability for the shorter route (approximately - the process of normalisation will alter this figure slightly). Thus, the direct route receives greater reinforcement than the circuitous one, which is the required effect. The way in which ants age greatly affects the balance between longer and shorter routes.

A network is used that provides five different routes - with the number of hops ranging from one to five - and this is shown in Figure 5.1. This network is used throughout this chapter to validate the Ant-System. The system is programmed so that the amount of

pheromone deposited at each hop is the same - the values tested are 5%, 10% and 15%.

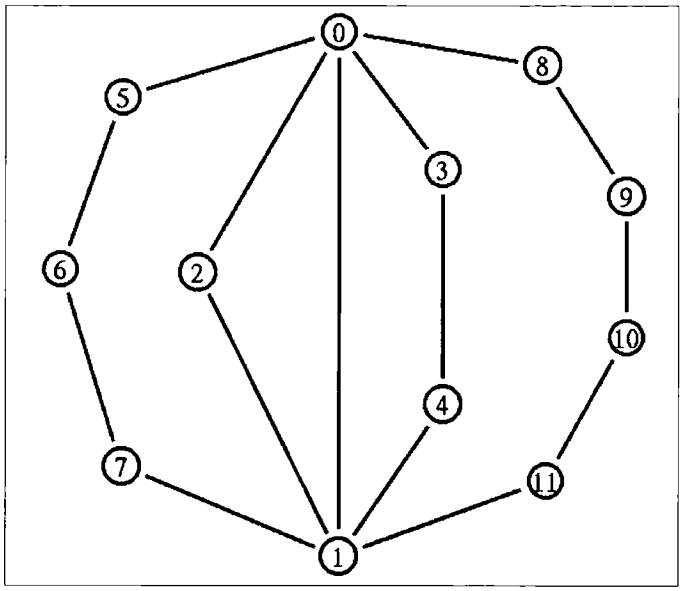


Figure 5.1: Test Network for Chapter 5

Figure 5.2 shows a standard form of simulation result for this chapter, and is the first result presented here; so a little time shall be spent describing the aspects now. It can be seen that although the figure comprises two separate graphs, they are describing the same experiment, and that they share a common time axis. The upper graph shows the number of ants using each incoming route, whilst the lower graph shows the resulting probabilities. In most graphs in this chapter the effect each ant has on the probabilities differs - i.e. the amount of pheromone it leaves changes. The key included within the graph area indicates which line represents the probabilities/ants from which node.

5.2.6 Form of Ant Ageing

The need to reduce the amount of pheromone deposited at each hop is clear. What form this reduction takes is now considered. Two broad options are available:

- A linear reduction
- A proportional reduction

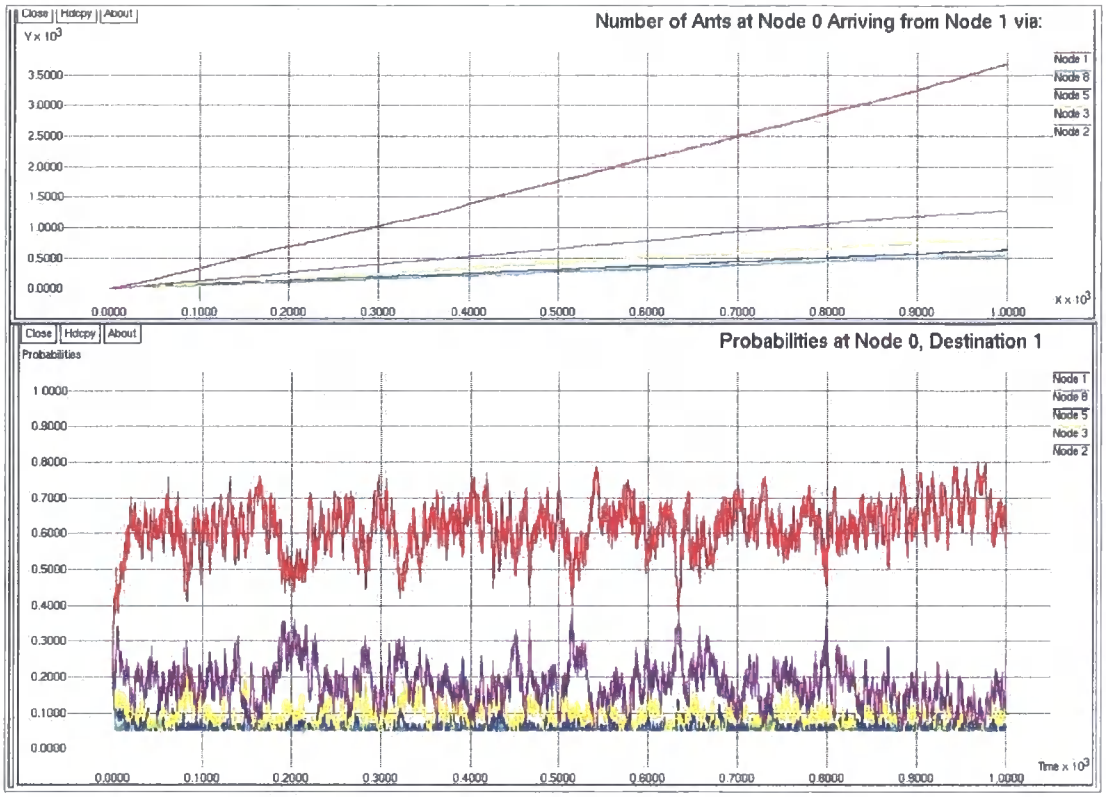


Figure 5.2: Ant-System with linear ageing

A linear reduction, e.g. along the route, 40% then 30%, 20% and 10% (Which sums to 100%) of the initial amount is deposited, would have the effect of giving a fixed limit to the range of the ants - in this example, an ant would only travel four hops - if the destination is not within four hops no route will be found. The effect of this linear drop-off is shown in Figure 5.2.

A proportional reduction is used, i.e. at each hop a set percentage of the ants current pheromone is deposited, e.g. 10% or 50%. This gives each ant potentially infinite range. This gives an exponential drop-off function when considering the pheromone left by an ant at each hop. The effect of this algorithm is shown in Figure 5.2. As can be seen, the shortest route receives significantly more reinforcement, so that the route becomes the clear and obvious choice to be recommended for actual data-traffic.

5.2.7 Ant Production Interval

The Ant Production Interval parameter is the time interval between Ants being produced. This means that it directly affects the amount of control traffic being introduced onto the network; therefore care must be taken not to set the rate too high. It should also be noted that the number of ants present on the network will also depend on the number of nodes.

5.2.8 Ant Priority

Anther issue associated with the Ant-System is the relative priority taken by the ants over the traffic. The ants could be given any of the three possibilities - higher, lower or equal priority; or the relative priority could even be manipulated.

Making the ants equal priority with the traffic would mean that the control system gets a good evaluation of the state of the network as the ants face exactly the same congestion conditions as the traffic (neither getting priority to bypass it, nor being bypassed themselves). However using equal priority also means that in times of congestion the system is transporting control traffic at the expense of payload traffic.

Giving higher priority to the ants would not be particularly useful, as they are there to measure the congestion but would bypass it; having stated that, other factors would give indications to congestion, such as queue-state. It could however be used to measure the network infrastructure's performance and not take into account loading levels, if this was a useful exercise.

Giving the ants lower priority than the traffic means that the system will be more sensitive to congestion: when congestion occurs the ants will not get through. However, if no ants are able to arrive to reinforce routes, the system may over-react, and cause unsuitable routing decisions to be made.

Therefore, in the design of this system, the ants are assigned a lower priority than the traffic, but if some level of service could be guaranteed to them (through a suitable multi-queue weighted service system for example) then the routing system could be made more robust.

5.2.9 Congestion Compensation

Since the Ant-system should take account of the traffic levels present on the network, they should be affected by any congestion that occurs. Firstly, this can be done by giving the traffic a higher priority than the ants, as discussed in the previous section. Secondly, the ant's pheromone level can be reduced depending on the level of congestion in the network, and this can be quantified by measuring how full the queues are at the nodes of the network - i.e. how much traffic is waiting to use each link.

The first measure means that if there is any traffic waiting in the direction the ant is travelling, then the ant will be delayed; this ensures that the service offered to payload traffic is not undermined by excessive control traffic. Remembering that Backward Reinforcement is being used, the second measure can be used to ensure that the state of the route in the reverse direction is taken into account by adjusting the pheromone level as a function of the reverse direction loading.

It is important to note that the ant is penalised depending on the state of the queue in the return direction - this attempts to take account of any congestion in the reverse direction being reinforced. It also more closely couples the two directions, forwards and return.

This penalisation is done by reducing the ants' pheromone level in proportion to queue occupancy; if the queue is seventy-five percent full, then the pheromone level is reduced by seventy-five percent.

5.2.10 Storage

The storage of the probability tables is a scalability issue. The amount of storage

required for any particular node is related to the number of links connected to that node, and the number of nodes on the network; the number of entries being the product of these two.

5.2.11 Relationship to Agent System

It is important to note the relationship between the Ant System and the actual routing mechanism for the network. It must be made clear that the Ant System does not make the actual routing decisions for the traffic. The Ant System is a means of measuring the metrics of a network, to allow a routing decision to be made.

In this case it can be envisaged that the Ant System could be connected to the Network Agent described in Chapter 3 to provide it with more subtle metrics than the simple hop-count. The Ant System would inform the Network Agent if the best route had changed, but the decision as to whether the traffic is redirected along the new nominated best route would be up to the more strategic agent.

For instance, perhaps the routes were constantly "flapping" [Labovitz et al. 2000, 2001] between two similarly probable routes. The agent could instigate a threshold that meant that it would only change the route being used by the traffic if one route exceeded the others probability by 5% for example.

Put simply, the Ant-System is a source of information about the network, to enable a routing agent to make a more informed decision. Figure 5.3 shows a flow diagram of the process for reference.

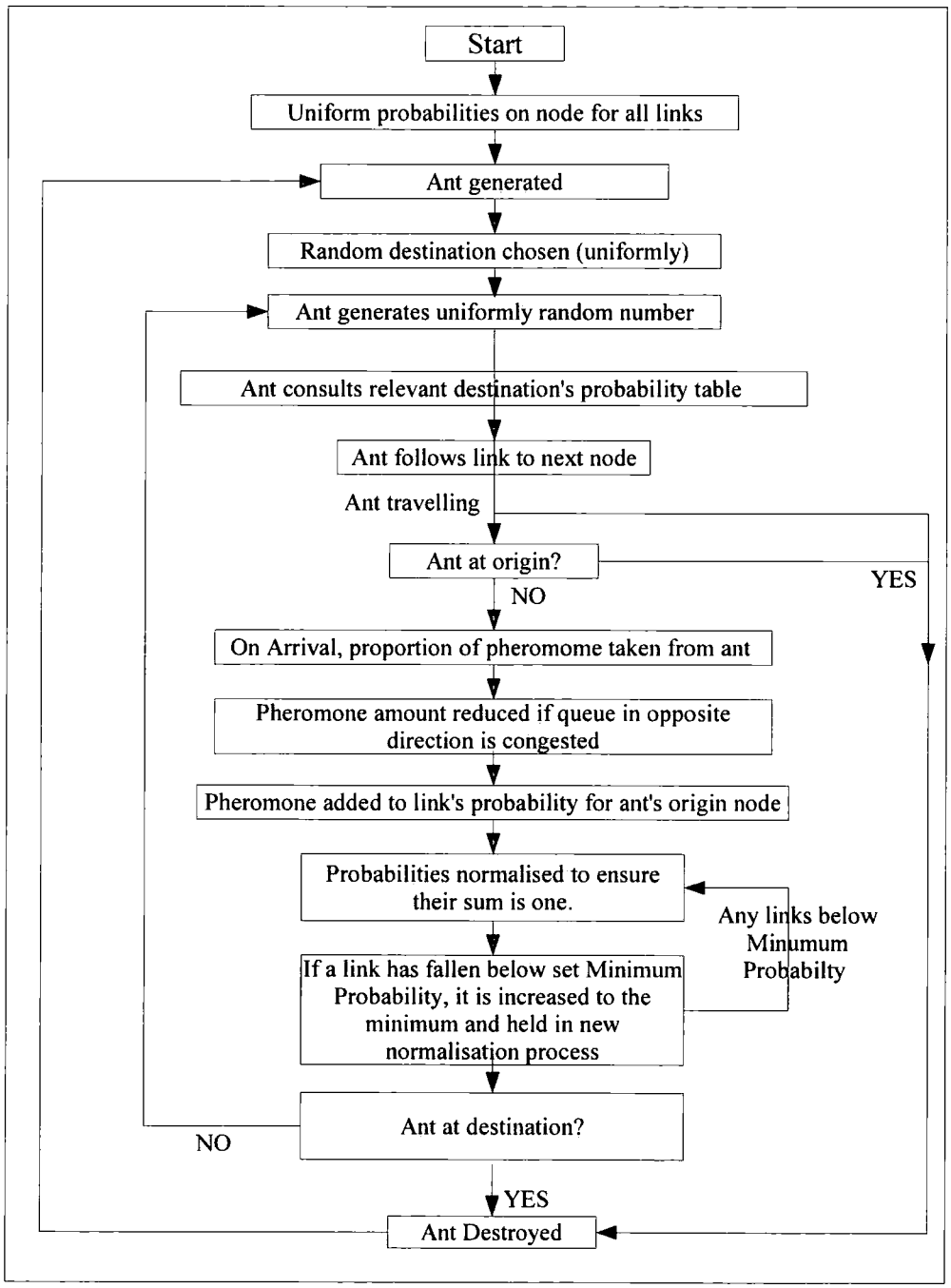


Figure 5.3: Flow diagram of the life of an ant

5.3 The Network Simulator - ns

Simulation studies were performed using the Network Simulator, or *ns*, version 2 [Breslau et al. 2000]. This simulator is the result of a number of collaborative research projects. It is an event-driven simulator which models TCP/IP networks. It is available in the from of source code, libraries, and scripts from website [e]. The program itself is implemented in the Object Oriented Programming Language C++, but it is interfaced by, and run from, TCL, Tool Control Language, which allows automated sessions to be run through its scripting language.

The Ant-System required detailed integration into the simulator code - which was not a trivial task. It required thorough knowledge of the working of *ns*, and is unlikely to have been possible with commercially-available simulators. The Ant implementation is described through the rest of this chapter.

5.4 Implementation

Having established the various issues that must be considered in designing the system, this section will now describe the detail of the implementation. First of all the overall structure is described.

5.4.1 Infrastructure

The Ant-System is operated from every node on the network. It is on the nodes that the software is run - each node runs its own ant colony. This is distinct from a system where the ants themselves are mobile code that is run at every node that is visited. This would greatly increase the amount of traffic produced by the Ant-System.

This node software, or the Ant Colony, as it will be referred to, contains the Probability-

Tables, or Pheromone-Tables, which the ants update. An alternative view of the system is that there are embedded agents on each node which pass each other simple messages constantly.

It is assumed that each node on the network is aware of the existence of every other node as it would when fitting into the overall Agent-Hierarchy described in Chapter 4. This knowledge could be provided by the implementation of an initialisation phase of the ant-system, whereby the first ants are sent out to discover nodes.

Each Probability Table is made up of a row for every destination, and a column for every direct-link (the next-hop). An example of one of these tables (not necessarily for the network used elsewhere in this chapter) is shown in Figure 5.4. If attempting to get to Node 6, then the link to Node 1 would be used, but potentially any of the links are feasible, as they all have some reinforcement (remembering that a link that has fallen to the minimum of 5% has received no, or virtually no, reinforcement).

Dest / Next-Hop	*1*	*2*	*4*
1	0.9	0.05	0.05
2	0.05	0.9	0.05
3	0.34	0.33	0.33
4	0.05	0.05	0.9
5	0.475	0.475	0.05
6	0.56	0.2	0.24

Figure 5.4: Example of probability table stored by a node

5.4.2 Queuing System

The system is based on a Class-Based Queueing algorithm, where different types of traffic have separate queues. For example, there could be a single queue for Constant-Bit-Rate traffic, another for video streaming etc. To allow the ants and traffic the correct access to resources, the payload traffic has a queue (or queues) and the ants have their

own queue. The ants' queue is very short, limiting the number of ants that can wait in the system to three. If an ant arrives and cannot fit into the queue, it is dropped. This ensures that there is not a large build up of ants that are then transported when the link has more availability, thus providing a strong reinforcement that is not justified.

Another facet of the Class Based queueing system is that the ants are guaranteed a small proportion of the bandwidth, in the region of one to three percent. This ensures that there is a continuous flow of ants avoiding extreme conditions such as the probabilities completely stagnating by guaranteeing some ants are transported at all times..

5.5 Parameter Testing and Results

This section describes the testing to ascertain the effects of each parameter individually. The parameters to be tested include

- Form of Deposition Function
- Rate of Ant Production
- Deposition Rate
- Initial Amount of Pheromone

5.5.1 Effect of Rate of Ant Production

The most obvious parameter that will affect the performance of the Ant-System is the rate at which Ants are produced. Although by increasing this rate the speed of response of the system will increase, it will also directly affect the volume of control traffic on the network.

The Rate of Ant Production was therefore set to approximately the same as the propagation delay along a link. Using the prorogation delay of the link ensures that the link is not deluged with ants, but maintains a steady stream of them and ties the systems

in a time-period of the network.

5.5.2 Effect of Deposition Rate and Initial Pheromone Level

The Deposition Rate is the proportion of the ant's instantaneous pheromone level that will be used to reinforce the route just arrived on and refers to what proportion of the ant's remaining pheromone is left at the node. The Initial Pheromone Levels (how much pheromone each ant has at the beginning of its journey) show how much the ant affects the probabilities over its lifetime. The values tested were:

Deposition Rates	***Initial Pheromone Levels***
• 10%	• 0%
• 25%	• 1%
• 36%	• 5%
• 50%	• 10%
• 63%	• 15%
• 75%	• 20%
• 90%	

These were tested in a full grid and a representative sample of these results is presented next.

5.5.3 Results of Parameter Investigation

It can be seen from Figure 5.5 that an Initial Pheromone Level of 1% produces sluggish control regardless of Deposition Rate. The highest probability, for Node 1, only rises very gradually, and still has not settled at the end of the run. Such a response would be termed *overdamped* in the context of control theory. Although this would produce a stable system in the end, the Standard Deviation of the final portion of the graph is a low 4.79E-4.

Conversely, Figure 5.6, which shows the results of Initial Pheromone Levels of 20% produces very quick control, reaching the final value in 4s, but unacceptably noisy

probabilities (with a SD of 8.89E-2 and arguably is oscillatory in nature. In terms of control theory, this would be termed *underdamped*.

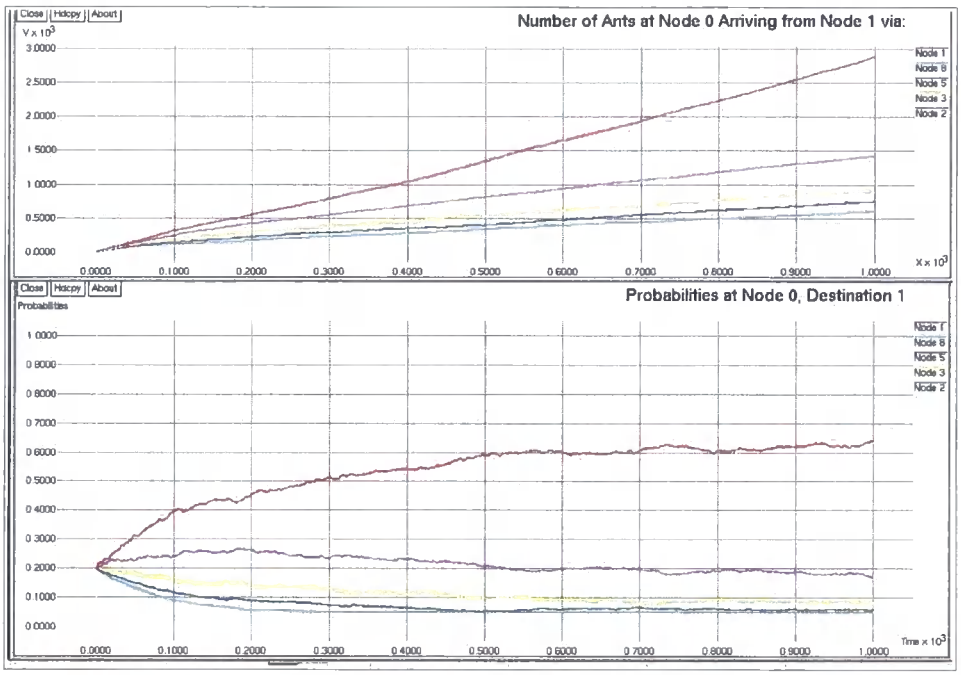


Figure 5.5: Ant-System with Initial Pheromone Level of 1%

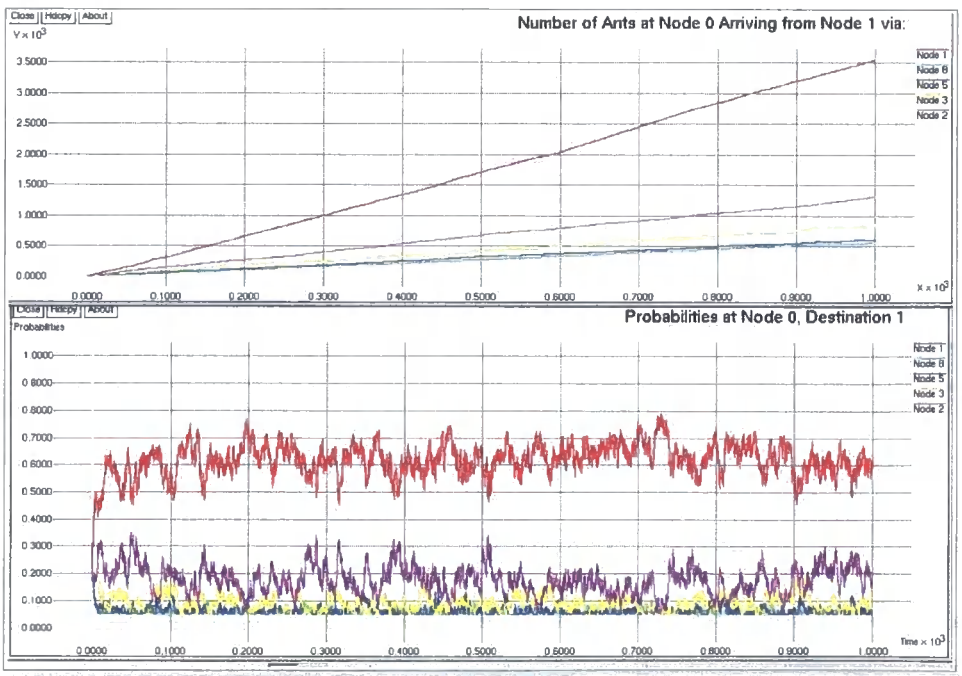


Figure 5.6: Ant-System with Initial Pheromone Level of 20%

The two Initial Pheromone Levels (IPL) in between the two extremes already described (20% and 1%) are the 5% and 10% levels. These exhibit both problems described above at either ends of their Deposition Rate (DR) ranges. Figure 5.7. and Figure 5.8. show the 5% Initial Pheromone Level (IPL) graphs. At 10% DR, with an SD of 2.49E-3 and even at 25%, (SD of 4.67E-3) the control is sluggish.

However, it is interesting to note the effect of the Deposition Rate, which can be seen to affect the equilibrium level of the probabilities. In the case of high deposition rates, this means that while short routes receive a large reinforcement, longer routes receive very little. In the case of the graphs, for 10% Deposition the graph "settles out" (although this is a little generous, given the noise) to about 50% probability with a SD of 1.12E-2. By the 50% Deposition graph however, it has almost reached the saturation point of 90% probability - remembering that it is constrained by the Minimum Probability set to 5% per route - the 63% graphs and higher show this saturation occurring. This also has the effect of reducing the SD, but this is artificial.

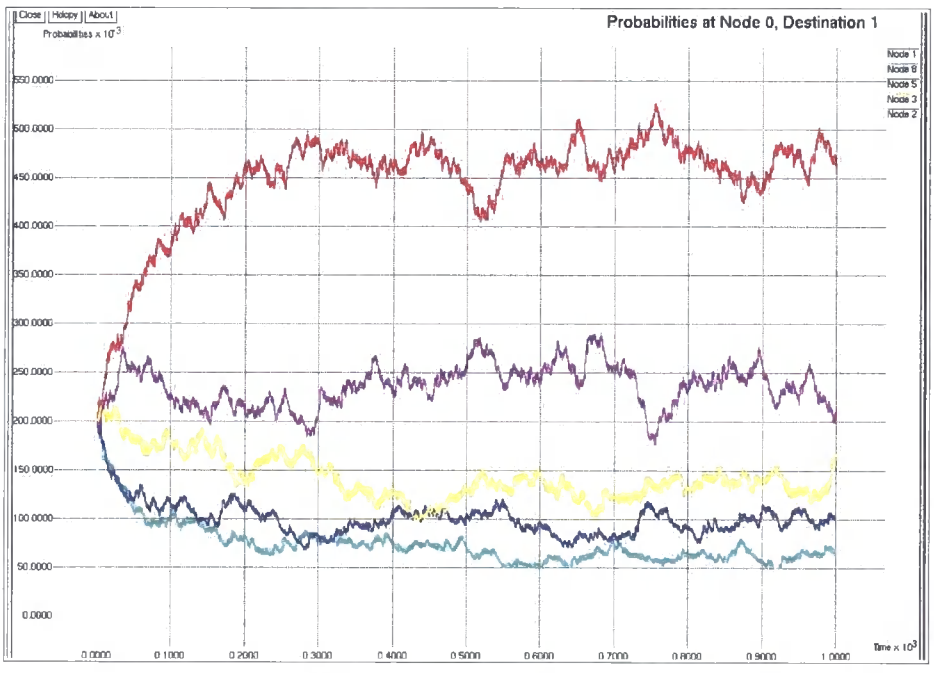


Figure 5.7: Ant-System with Initial Pheromone Level of 5% and 10% Deposition Rate

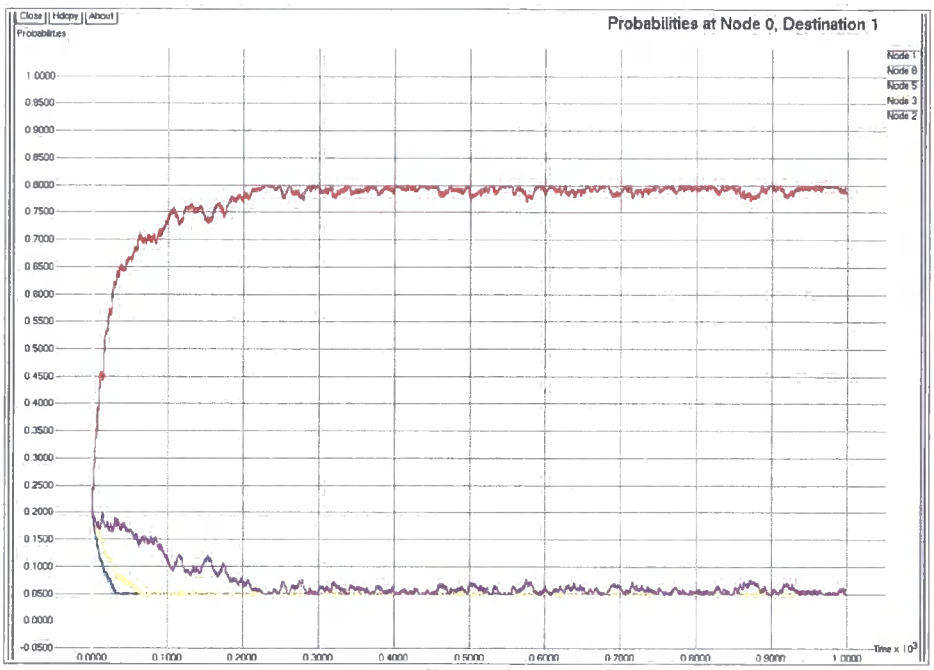


Figure 5.8: Ant-System with Initial Pheromone Level of 5% and 63% Deposition Rate

At 63% DR, and above, the saturation problem occurs. In between, 36% and 50%, reasonable control is exerted; although the 50% shows a better response, the 36% graph shows that a distinct back-up route is available - which could be important in the context of a failure.

Figures 5.9 and 5.10 show the 10% IPL graphs. These graphs show similar properties at the extremes: sluggish at a low DR, but saturated at a high DR. In this case 10% is too sluggish, and saturation starts to occur at 50%. The two in between, 25% and 36% are interesting. The lower DR (25%) seems to exhibit better control than the higher one (36%) - which has a slow oscillation near the end. The 25% graph also exhibits a clear second-best route that could be used as a backup as described earlier.

5.5.4 Final Configuration Details

It was found that there must be a compromise between maximising the speed of

response of the system and minimising the level of noise in the system. This means that under some circumstances the system does not respond as quickly as is desirable, but under others the noise does not cause problems with route flapping.

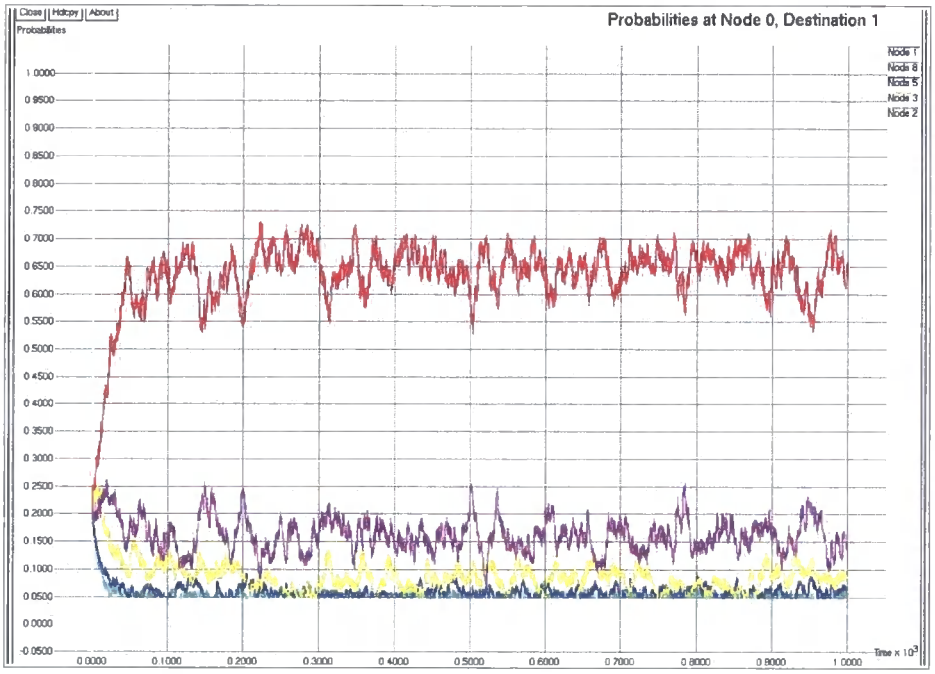


Figure 5.9: Ant-System with Initial Pheromone Level of 10% and 25% Deposition Rate

Figure 5.11 shows a summary of the results presented so far. As described in the previous paragraphs, the response of the system is decided by a combination of the amount of pheromone an ant has to begin with (the IPL) and the rate at which it uses it (the DR). There are two candidates for the best settings, a combination of 10% IPL and 25% DR or 36% IPL and 10% DR. Either of these two settings would give an satisfactory performance overall, with the potential problems already outlined. This system finds the shortest route under "normal" network operation, after an acceptable amount of settling time. Indeed, in most cases, the best route has been identified long before the probabilities have found equilibrium.

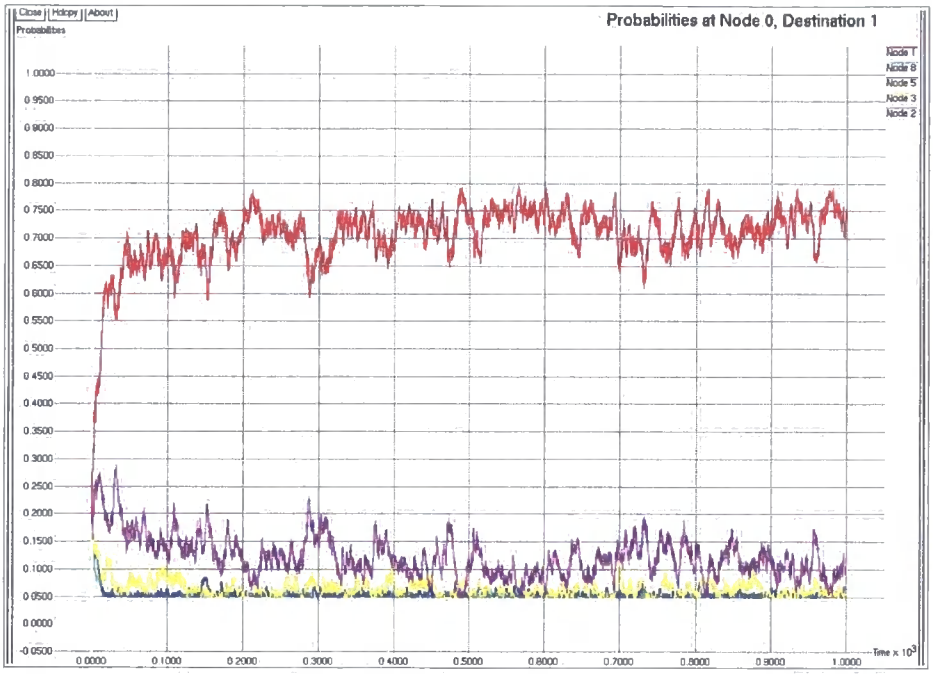


Figure 5.10: Ant-System with Initial Pheromone Level of 10% and 36% Deposition Rate

Scheme	Time to Settle	Settled SD
IPL 1% 25% DR	50	4.79E-4
IPL 5% 10% DR	30	2.49E-3
IPL 5% 25% DR	29	4.67E-3
IPL 5% 36% DR	26	6.62E-3
IPL 5% 50% DR	23	8.71E-3
IPL 5% 63% DR	22	9.87E-3 – artificially constrained
IPL 10% 10% DR	29	5.11E-3
IPL 10% 25% DR	27	6.33E-3
IPL 10% 36% DR	27	6.79E-3
IPL 10% 50% DR	25	7.02E-3
IPL 20% 25% DR	4	8.89E-2

Figure 5.11: Ant-System table of results

So far, the system has been tested under the conditions of light network loading (in terms of user traffic being offered to the network) with a mixture of constant bit rate and variable bit rate traffic present on the network, but not causing congestion The investigation just undertaken was to determine a reasonable set of parameters to use with the Ant-System rather than validate the system for all traffic conditions. To be considered any further than the confines of research, a routing system should be able to handle the situation of problems on the network caused by congestion and failures of nodes or links. The following final part of this chapter addresses congestion conditions and investigates the effect of varying loads of traffic on the system.

5.6 System Testing

Whilst the network is not congested, the Ant-Based Routing system behaves just as a conventional IP network would. If there is plenty of capacity for the traffic, then there will be plenty of capacity for the ants, so none will be dropped. Although the ants may encounter small levels of delay (while they are queuing in deference to the traffic), this is unlikely to be significant with large numbers of ants present on the network. Arguably these delays could have an effect when there are only small numbers of ants on the network (e.g. when many have been dropped), but this is also only likely when routes have very similar probabilities.

Under these normal operating conditions, the ants are unlikely to have much visible effect, since the traffic is routed according to a normal (though ant-devised) routing table. To reiterate, traffic is not routed subject to the ant's probability table (which would have the effect of routing traffic randomly, just as an ant is).

It is only when traffic levels are nearing congestion so that ants are dropped (i.e. an ant-queue they were attempting to join is full, so that the ants are discarded) that the ant-system comes into its own. When a single ant is dropped, the effect is small and the route it was trying to take is *not* reinforced, but when significant numbers are dropped at

one time or regularly (especially along one route) then the probability can drop enough for another route to be recommended by the ant-colony agent. Thus the traffic's routing table is updated and the new route is used.

Although ants will be dropped during times of congestion, ants are also liable to be dropped at levels of loading below congestion due to the bursty nature of traffic. The traffic may be taking up all but a small proportion of the resources, meaning ants are dropped, but no traffic is lost. Conversely, to flood the network with far too much traffic would also be over-simplistic and uninteresting. The experiments described in this section are designed to be in the region of this boundary between normal and over-loaded conditions but include a certain level of actual congestion as well.

The system was tested under various traffic-loading conditions using combinations of variable bit rate and constant bit rate traffic schemes, to explore the region around congestion. A scenario tested included heavy congestion, but all this proves is that an Ant-Based Routing system cannot increase the capacity of the network, only improve the efficiency with which the existing resources are utilised. When congestion occurs on a single link the system will have a finite response time until it changes. It would be advantageous to be able to speed up the response of the system, however to do this in the context of the system described would mean settling for the noisier schemes tested - this may be unacceptable, since it could result in undesirable route flapping behaviour.

Traffic level that were tested included mixtures of constant and variable bit rate traffic, these further demonstrated that settings desirable for system stability and adapting to slight changes to traffic conditions were not the best for adapting to more severe levels of congestion.

Figure 5.12 as shows an example of these results with the system being tested with a period of congestion, from 20s to 35s. It can be seen that the system does react to the shortest route being congested, as the probability drops. However, it does this at a fairly slow rate, taking 5.5s for another route become greater, especially considering the speed at which it recovers afterwards. Although the gradient of the highest probability is an

important indicator of the speed of response, it must be remembered that it is only when the probability falls below that of another route that the routing is actually switched.

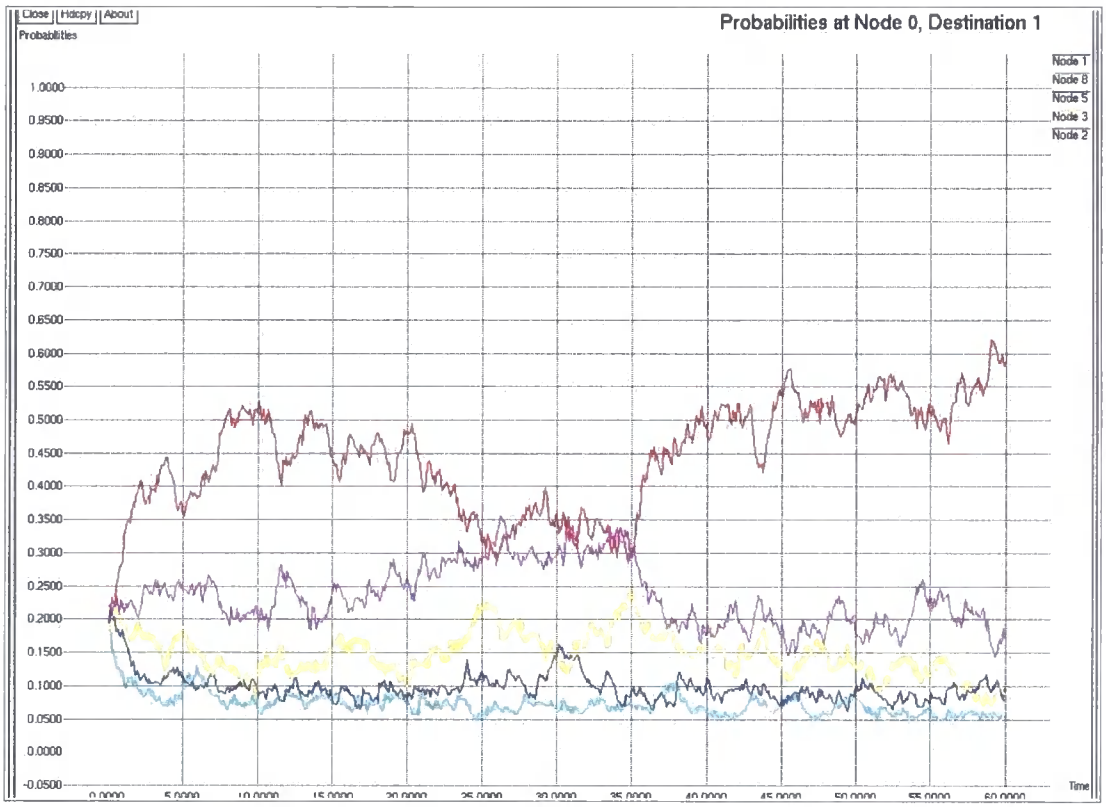


Figure 5.12: Ant-System operating during congestion

Figure 5.13 shows a summary of the different traffic scenarios tested with the final configuration detailed above. In each case there was a settling period of 20s, followed by a 15s period of congestion. The percentages of VBR is the expected average rate. It can be seen that when CBR traffic is present, the SD is lower than when VBR traffic is, due to its bursty nature. The 100% loading of CBR traffic produces a quick response, since this is the link fully utilised, so only the bare minimum of ants get through. In any of these scenarios it would be advantageous to be able to speed up the response of the system, however to do this in the context of the system described would mean settling for the noisier schemes tested - this may be unacceptable, since it could result in undesirable route flapping behaviour.

Scheme	Time to route change / s	Settled SD
CBR 90% loading	7.9	9.88E-4
CBR 95% loading	4.8	9.67E-4
CBR 100% loading	3.8	5.50E-4
CBR 85%, VBR 5%	9.7	1.98E-3
CBR 90%, VBR 5%	7.4	8.49E-3
CBR 90%, VBR 10%	6.2	4.67E-2
VBR 90%	9.5	5.43E-3
VBR 95%	7.2	8.73E-3
VBR 100%	5.1	4.32E-2

Figure 5.13: Ant-System Traffic scenarios table of results

5.7 Discussion

An important consideration in the development of routing systems, and other aspects of network control, is the degree to which they can be extended from simple test networks to work on larger sized networks. It is all too easy an assertion to make that any system developed must be able to cope with any size of network. However the implications of this assumption are far from straight forward.

The system described here is not intended to control an Internet-size network homogeneously, but is targeted at working within an hierarchical structure, such as is present within the Internet. In particular the system developed here is designed to work within a single Autonomous System (AS) - this is defined as a network within the control of a single Internet Service Provider, ISP, or equivalent. This is also true of OSPF; to do otherwise would be infeasible.

Other Ant-Routing schemes have been run on simulations of national backbone networks, such as the United States of America's (AT&T) [Vittorri 2001] with 87 nodes, and Great Britain's (BT) [Schoonderwoerd 1996] with 30 nodes. These efforts

demonstrate the scalability of Ant-Based Routing Schemes generally, and define the range of size of networks considered for this system in due course.

Considerations that may affect the scalability of Ant-Systems include the time for convergence and the number of ants required to reach this convergence. Since each node discovers a route to every other node, then for a single ant to have been sent between every node will require at least n-factorial ants. Conversely, the consideration of not overloading the network with control traffic means that attempts to speed the process up must be handled with care. These factors are extenuated by the likelihood of there being a larger number of potential routes (both viable and not) between two nodes on a network of a larger size.

A way to increase the effectiveness of each ant, so reducing the number required, would be to have the ant keep track of intermediate nodes visited, and update all the routes concerned. This is a logical extension to the basic algorithm, and is discussed in chapter 3.

The operation of an Ant-Based Routing system is fascinating to watch on a traffic simulator. However, it must be considered in the context of producing a system that could be applied to a real network. Although Ant-Based Routing systems are not currently deployable in the real environment, that does not mean that in the future they will not be a feasible choice, either as a whole, or some element incorporated into a future routing protocol, such as OSPF's successor or development. Such an element that could be taken up is that of the constant measurement of the performance of the links.

OSPF is made up of a number of constituent protocols, including the Hello Protocol. This protocol tests the link by default every 10s. In doing so, a Hello packet is deployed, and a Hello packet is required in return. This exchange of hello packets effectively measures the link in an on/off sense. OSPF only reacts to the presence of a link, as just described, not any congestion occurring on the links. The ant system performs similar measurements, with smaller packets but more regularly, but derives more information from those exchanged, including the presence of congestion. The Ant-System in this

sense is a measurement system, and the results produced could be used as the metrics required by OSPF to quantify the *cost* of a link.

An argument against routing schemes that take into account the presence of congestion on the network relates to the oscillatory behaviour that can result. This is due to the movement of traffic from one link to another affecting the measurement that caused the move in the first place. A potential way round this is to introduce intelligence that can reason about such route changes, and form proactive strategies to prevent such problems.

5.8 Chapter Summary

This chapter has described the implementation of an Ant-Based Routing and Load-Balancing System. Firstly, a detailed analysis of the workings was undertaken, followed by a description of the implementation. It was found that little published details were available on the actual parameters controlling the Ant-System. In the absence of these a series of simulated experiments was undertaken to identify the optimum parameter settings for this system.

The final parameter settings were found to be a compromise between response speed and level of noise. Thus, the Ant-System has been shown to find sensible routing configurations under static traffic conditions, but does not respond well to fast changing situations on the network. The following chapter addresses this problem with a novel solution. This will allow the Ant-System to handle both situations and adapt to the requirements of different networks.

Chapter 6: Agent Managed Ants

An Ant-Based Routing System has been constructed and it has been shown to find sensible routing strategies under static conditions; this is described in Chapter 5. With the introduction of dynamic traffic situations onto the network, the ants do not always display optimum behaviour. It has been shown that parameters cannot be set to a single value to satisfy both steady state and transitory conditions optimally. It is therefore proposed to use an Agent on each node to adapt to changing conditions on the network. This, it is hoped, can be achieved by dynamically manipulating the parameters with which the ants operate.

This Chapter describes the implementation of an Agent that is designed to manage strategically the Ant-System that is described in Chapter 5.

The chapter is arranged as follows; firstly the parameters that could be controlled are discussed along with the measurements that could be used to decide when to change them. Secondly, the Agent requires some form of intelligence to be encapsulated within it, which affects the structure of the agent, so this will be discussed next. The intelligence to be used is Reinforcement Learning (RL), for reasons outlined in the section, and a selection of different algorithms is presented.

The discussion is then broadened to include how more complex domains are handled within RL and the context of the current problem is introduced to the discussion. At this point the design and implementation can be described in full.

To conclude the chapter, results of the experiments performed are presented; this then leads to a final full discussion of the system as a whole.

6.1 Overview of Agent-Managed Ants

This section discusses the reasoning for employing an Agent to manage the Ants. It then proceeds to discuss how this might be achieved. This entails monitoring some measurements of the system to identify times of change on the network, as well as identifying some way of changing the performance of the Ants to manipulate strategically their response.

6.1.1 Motivations for This Work

Consider two nodes connected by two different routes, (a) a direct link and (b) a circuitous route, involving many hops. Under normal loading all traffic will be sent the direct route, and rightly so; in the ant-scheme this will be exhibited by the short route accumulating a large probability at the expense of the longer route.

Now consider when this short route becomes congested; under normal operation it will take a long time for the ant probabilities to change sufficiently for a route change to take place; in this case it would be advantageous for the ant-parameters to be adapted to speed up the response; and this requires Intelligent Control.

Assume the shorter route described above had probability of 80% compared to 20% for the longer one. If each Ant has the effect of changing the probability by two or three percent, it will take ten to fifteen ants to equalise the routes probabilities. During this time traffic is likely to be discarded, which can cause even more congestion through retransmission schemes.

This outlines a problem with Ant-Systems - the inertia that is built up by the best route in static conditions, and shows that their strength becomes their weakness in dynamic situations.

6.1.2 Aims of The Work Described in This Chapter

The aim of this research is to show that an intelligent agent can be used to modify the parameters associated with the Ant-system to make both the convergence from start-up, and the response to changing conditions within the network, quicker.

The parameters are changed on a per-node basis - i.e. an agent on each node is in control of the parameters used on that node. This means that, at this stage, the agents are acting independently from each other, although the ant system is an implicit, or stigmergetic, form of communication. This can be described as a fully-distributed control system, although there is this inherent communication between the nodes, in the form of ants.

The following paragraphs start to discuss how the Agent might hope to achieve this control.

6.1.3 Ant-Parameters to be Controlled

Firstly, it must be considered which of the parameters within the ant system, shall be strategically manipulated. The three immediately obvious candidates are:

- Rate of Ant Production
- Rate of Deposition
- Initial Amount of Pheromone

At first the Rate of Ant Production would seem the easiest to use. However, the problem would be that any increase in this parameter would directly affect the amount of Control Traffic on the network. This is considered undesirable - especially as it would increase the volume of traffic at the point of congestion, making it worse - and so is rejected until a more sophisticated control regime is implemented, that can take into account the subtleties of this consideration.

The rate of deposition does not suffer from such intricacies, as it merely determines how much pheromone is deposited at each node on an ant's journey. However, this parameter

does affect the relative balance between longer and shorter routes, so should also be used with care.

This leaves the third candidate parameter, which describes the amount of pheromone each ant begins its life with. This could be described, in combination with the previous parameter, as how virile each ant is. By increasing this parameter more pheromone is deposited at each hop - a set proportion is left each time since the Rate of Deposition is being kept constant. Therefore each ant will have a greater effect on the probabilities.

6.1.4 Instantaneous Probability

This is the probability of the current route in use. This can be used to give the mean a context. For example, if the mean is very different from the current weight, then the system has drifted/changed over time.

6.1.5 Windowed Statistics

A number of the meaningful statistics require more than just the present value of a measurement. However, the most useful measures will not merely incorporate every measurement from the first reading to the present one, as the system will be burdened with excessive processing as time progressed. This approach would also be unlikely to provide the most useful measurements. This implies that some sort of windowing function is required. The sort of statistics that require a window function will be: any type of average (means, medians etc.), the Standard Deviations and Variances.

The size of the window is a parameter which can affect the performance of the system, as well as affecting the perception of its performance. By introducing a window a filter is created which could both be a positive influence if designed well, but a hindrance if designed badly. In this case the mean can be used to filter out short-term variation in the probabilities (i.e. the "noise" associated with individual ants arriving) but the window must not be overly large as this will slow the effect of more recent phenomena

To this end, the window-length is set to 250 entries, meaning 250 ant arrivals. This gives a time constant of approximately three seconds under normal conditions. This three second window provides a balance between reacting to actual trends but not being reacting to instantaneous fluctuations; as described above.

The statistics derived from this window are described below:

- Mean - this is the average of the probabilities for the link measured at the time of each ant-arrival.
- Standard Deviation (SD) - this is a measure of the spread of the values. Thus the probabilities, when they are steady, will have a smaller SD than when the probabilities are changing rapidly.
- Gradient - This is calculated from the mean of the earliest 10 windowed probabilities, and the last 10. Ten samples are used at each end to form an average, rather than basing the measurement on a instantaneous reading. The function, as a whole, characterises the change of the probability over the time of the window.

6.1.6 Event Statistics

A record is kept of "events". This is primarily a record of when the last route change took place, and what the old route was. This is intended to guard against constant route-flapping. Each change of route triggers a report to the agent. These reports could be seen as decision points, at which the Agent could take action in light of an event.

6.1.7 Use of Measurements

Having outlined the measurements that could be used to control the system, how this information can be used must be considered. Some form of learning is required - it is undesirable that this learning should take place before the system goes on-line, rather it should learn over time - trying out reasonable strategies, and improving them if the situation repeats itself.

In the light of these requirements, it is proposed that the mechanism to be used is that of

Reinforcement Learning - the system will earn a reward for taking a good action, but is punished if it makes a bad decision. This branch of Artificial Intelligence is outlined in the next section.

6.2 Reinforcement Learning

The main concept behind Reinforcement Learning (RL) is the idea of reward (or Reinforcement). Consider the training of a dog; to entice it to behave as desired, treats in the form of food can be given when it is good and, more controversially, it can be punished when it misbehaves. It consequently associates being rewarded (with food) with a particular behaviour (or set of actions).

This idea is used to train software to act as required; some reward function is used to reinforce desired actions, while the absence of reward, or negative reward (punishment) is used to dissuade from undesirable actions. This is developed further by the concept of different system "states".

In this scenario, the software can be trained to observe a number of different states of the system, and associate different desired behaviours with those states.

6.2.1 Types of Reinforcement Learning

Within RL, there are a number of different algorithms, the main contenders include:

- Dynamic Programming (DP)
- Monte Carlo Techniques (MC)
- Temporal Difference Learning (TD)
- Q-Learning (QL)

The first of these, Dynamic Programming, requires the development of a model of the environment which is to be controlled. As this is a non-trivial task in this context, the technique is rejected for use here.

Monte Carlo techniques, generally converge slowly, especially compared with the final two techniques. This is because whereas TD and QL update constantly, MC updates its values comparatively infrequently.

The final two techniques are very closely linked - they are based on different forms of the same equation. TD is used when a model of the environment is available - when the effect of an action is deterministic. It was used in [Tesauro 1992, 1994, 1995, 2002] to learn to play Backgammon - and *very* successfully so. Both techniques are described in more detail in the next paragraphs.

6.2.2 Temporal Difference Learning

This method was initially developed in [Samuel 1959] and [Klopf 1972]. It was then further developed and strengthened in [Sutton & Barto 1998]. Effectively, if trying to predict the time taken to achieve a task, or the weather one day next week, TD methods use constant updates to the prediction to hone in on the correct answer, this also makes the calculations more efficient, since they can be iterated; as opposed to Monte Carlo methods which allow no updates, and must be formulated from scratch each time.

Considering Tesauro's Backgammon program [Tesauro 1992, 1994, 1995, 2002], TD-Gammon, the basic algorithm is this:

- A current game-position is presented to the agent
- The Agent calculates every possible move it can make - these are called the "after-positions" i.e. After the move is made this will be the position.
- Each after-position is assessed and assigned a value which is dependent on past experience and can be interpreted as the probability of winning from that position.
- The position with the highest value is chosen, and that is the move to be made.

Tesauro's papers follow the progression of TD-Gammon over some years, and the algorithm is augmented to look further ahead - by two or three moves. However, the basic mechanism is still the same.

The value of each after-position is notated as V(s) - that is, the Value of being in State *s*, which can be defined as: "the long-term reward that will be received by being in that state". With TD-Gammon, the system is only given a reward of one when it wins[1]. This reward is then filtered back to the states that it passed through to get to the winning-state, as is described below.

The following equation represents TD-Learning, where s is the current state, V is the Value function of a state, r is the reward, α the learning rate and γ is the future reward discount rate.

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s+1) - V(s))$$

Algorithm 6.1: The Temporal Difference Learning Algorithm

So the new value becomes the value of the current state, plus (the learning rate multiplied by) the sum of reward for the current state and the discounted value of the next state less the value of current state.

The discount-rate referred to, γ, is a similar concept to that of money - if you could have £10 now or in a years time, you would take it now as it would be worth less in a year, due to inflation etc. the discount rate can be any value between zero and one, which are the significant cases.

TD where $\gamma = 0$, is the myopic version, where only the reward at the present time is taken into account, so the equation reduces to:

$$V(s) \leftarrow V(s) + \alpha(r - V(s))$$

Algorithm 6.2: TD with zero Discount Rate

TD where $\gamma = 1$ looks for absolutely any reward in the future, from now until infinity.

$$V(s) \leftarrow V(s) + \alpha(r + V(s+1) - V(s))$$

Algorithm 6.3: TD with unity Discount Rate

1. In Backgammon, there are actually three levels of winning, an ordinary win, a gammon and a backgammon. TD-Gammon is rewarded extra for the latter two, but these outcomes are relatively uncommon. In terms of this discussion, these outcomes are ignored.

In terms of TD-Gammon, by using $\gamma > 0$, the reward for a win will reinforce all the states passed through as experience grows.

The elegance of TD-Learning is that it can be implemented incrementally, with little computation. The problem is the small number of situations where enough is known about the environment. This is where Q-Learning comes in.

6.2.3 Q-Learning

Whereas with TD-Learning the result of actions had to be deterministic (in Backgammon the immediate effect of a move could be observed by its after-position), by using the following substitution, Q-Learning does not require a deterministic relationship - where *a* represents an action:

$$V(s) = \max_{a}[Q(s,a)]$$

Equation 6.4: Relationship between V(s) and Q(s, a)

This equation states that the V-function is the maximum Q-value of all the possible Q-values connected with the available actions. This allows RL methods to be used when the immediate effect of an action is *not* known. Substituting Equation 6.4 into Equation 6.1 gives:

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

Algorithm 6.5: Q-Learning

Note that once an action has been taken, then it becomes the maximum value of a set of one. A Q-value is again interpreted as the expected long-term reward for taking action *a* in state *s*.

Q-Learning was defined and developed as a PhD Thesis [Watkins 1989], but is summarised in a Technical Note [Watkins & Dayan 1992]. It is described as assigning values to State-Action pairs. It reinforces successful actions, but punishes unsuccessful

ones. Q-Learning is an Off-Policy TD Control algorithm. Off-policy means that the system can learn about the policy whilst not actively pursuing it [Sutton & Barto 1998]. The system converges to the optimum policy if every state is visited an infinite number of times.

6.2.4 Table-Based TD-Learning and Q-Learning

Both TD and QL in their simplest form use a look-up table to store values in. This is indexed by the state and action and can easily be updated - it is analogous to the ant's storage system. Indeed, whereas the ants store a table of probabilities for each link and destination combination, Q-Tables store reward values for each state and action combination.

[Kapetanekis et al. 2002, 2004] describes two agents, who are each unaware of the other's presence, playing a game in which they must choose between two actions - *a* or *b* - and the reward they achieve is contained in a grid such as shown in Figure 6.6. In the game described the agents must learn to play the joint action *bb* to maximise their reward on each go. This game is often run iteratively, and so the agents are able to explore the different actions available to them. Because they are not aware of the other's existence, it appears to them that there is some random element to the result of their choices.

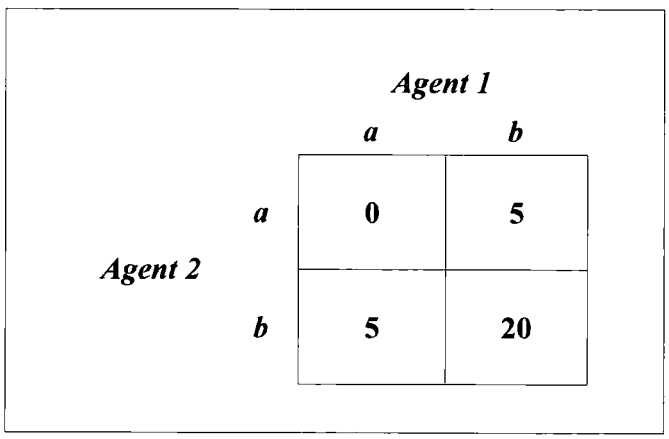


Figure 6.6: Diagram of Agent Coordination Game Reward Scheme

The games become more and more complex by the use of larger and larger grids, but also by using a stochastic element as well. The agents are tested to see if they pick the *optimal joint-action* by the use of Table-based Q-Learning and often are able to.

It is obvious that look-up tables will become unwieldy very quickly with large numbers of states and actions, so this solution is not usually practical. The problems and solutions to this are described below.

6.2.5 TD-Learning and Q-Learning in Complex Domains

As stated above it is usually not possible to just use a look-up table for most problem domains due to the sheer dimensions involved. This includes the domain concerning this thesis. The telecommunications network described is highly complex which is *why* we are attempting to apply RL.

The system is sufficiently complex that it can be abstracted to a continuous system - the time-scale on which the agent must work is a far greater time-scale than that of the system it is controlling. For example, a graph of a monitored variable will show many discrete readings, but when a long sequence of these are viewed, the graph will tend to look continuous.

The system is also characterised by a potentially infinite number of states the system must consider. For the the learning algorithm to be viable it must be able to consider a handful of states at most. Therefore a stage is required analogous to an Analogue to Digital Converter (ADC) - taking in a continuous signal, and outputting a set of binary digits.

This function-approximation would interpret the state of the system - reducing it down to a handful of states - and then a look-up table could be used to store the associated actions, as shown in Fig 6.7.

Alternatively, the function-approximator could be used to do both steps by making it

approximate the Q-values, in a similar way to [Tesauro, 1992, 1994, 1995, 2002] TD-Gammon approximates the V-function. An obvious function-approximator to use is an Artificial Neural Network (ANN, or just NN), which is indeed how Tesauro implements his system.

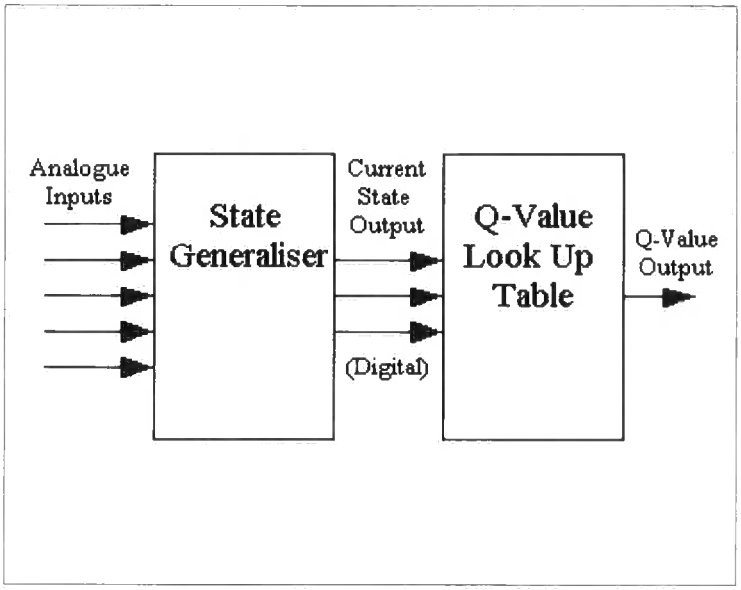


Figure 6.7: Diagram of Separate State-Approximator and LUT

6.2.6 Neural Network Based Q-Learning

Q-Learning is based on a table storing values for state-action pairs. The size of this table is dependent on the number of different states and actions (the number of entries being the product of the two). This can mean that the storage of such a table can be prohibitive in the case of complex systems, such as that discussed in this thesis. An alternative storage mechanism is to use a NN to approximate the look-up table.

A further paper by [Tesauro 1999] describes using his methodology described earlier, but implementing the Q-Learning algorithm within the same NN as the state generalisation. The application to which this is applied is unrelated, however. The technique continually updates its predictions by modifying them iteratively, so offers advantages on the volume and complexity of the calculations. Complexity is also reduced by using a single NN.

Figure 6.8 shows the configuration of a NN when being used for Q-Learning. The inputs define the state, the output is the Q-value of an single action, whilst the Q-Learning Equation shown in Equation 6.9, is used in place of the normal NN error signal.

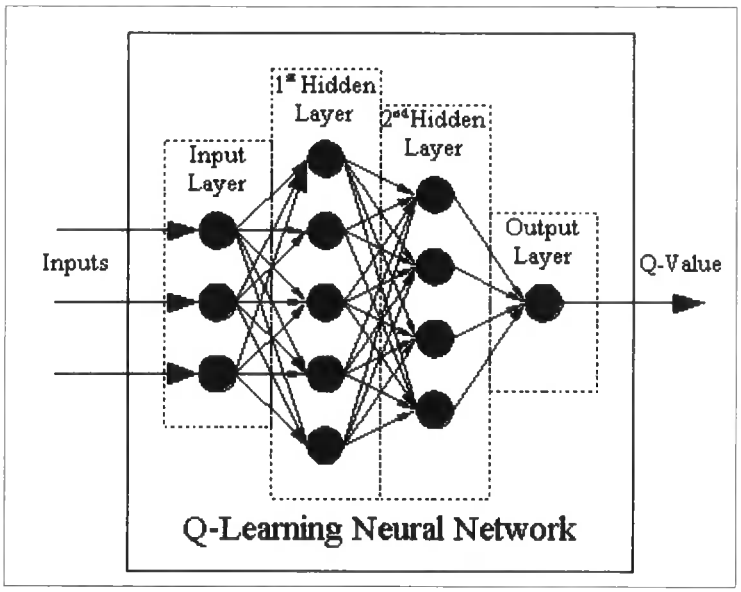


Figure 6.8: Black Box Diagram of NN Being Used for Q-Learning

$$Q-Learning\ Term=\alpha(r+\gamma \max_{a}[Q(s_{t+1},a_{t+1})]-Q(s_t,a_t))$$

Equation 6.9: Q-Learning Desired Output for an ANN

The following section will describe Neural Networks in the context of Q-Learning in more detail.

6.2.7 One-Step and Multi-Step Q-Learning

The simplest form of Q-Learning is One-Step Learning. This is where a single update is made at each time-step - i.e. The last action taken is rewarded. To aid the speed of learning, actions taken further back in time can also be rewarded. This requires the use of an *eligibility trace* [Sutton & Barto 1998] to record which state-actions to reward, and also to what extent. An *eligibility trace discount rate,* λ, is used to decay the rewards through time (i.e. The last action gets more reward than the action before that).

There is more than a single scheme to implement One-Step Learning. Watkins' Q(λ)-Learning [Watkins 1989][Watkins & Dayan 1991] stops rewarding actions backwards in time at the point where an *explorative* action is taken (when the action with the highest Q-value is *not* taken). Peng's Q(λ)-Learning [Peng & Williams 1991] ignores this consideration, but assuming (over time) the rate of explorative actions is reduced then the scheme will still tend to the optimum.

For the improved convergence speed that either of these schemes bring, the cost is significantly increased complexity in implementation, especially for NN-based Q-learning, where each state-action to be rewarded requires the entire NN-setup to be stored for reinforcement in the future.

These *eligibility trace* techniques apply equally to TD-Learning.

6.3 Artificial Neural Networks for Q-Learning

This section describes the theory behind Neural Networks and the design and implementation of the Neural Network used for Q-learning within the agent controlling the Ant-System.

6.3.1 Neural Network Theory

The theory behind Neural Networks is an entire subject in its own right; thus a detailed discussion is beyond the scope of this thesis. This section attempts to give a brief oversight of the subject, but to mainly highlight the differences of a Q-Learning NN from an "ordinary" NN. For more detail, the reader is directed to one of many standard texts on the subject, such as [Zurada 1992].

6.3.2 Forward Recall

The normal operation of an NN is in the Forward Pass, or Recall, mode. This is when

the network evaluates the inputs applied to it, and the network performs an approximation of the function it is designed to simulate, with the result read out at the output.

A simple NN is shown in Figure 6.10. A NN is made up of a number of neurons, which are grouped together in layers. A NN can be made up of any number of *layers*, accepting that fewer than two layers becomes trivial (indeed, less than three is uninteresting, for reasons explained later). Layers between the Input and Output are called Hidden Layers, as shown in Figure 6.8.

The basic operation of a neuron is the summation all of its input signals. This result is passed through a response function and the result is the output of the neuron. The response function can be linear, but can also take on other forms, as discussed below.

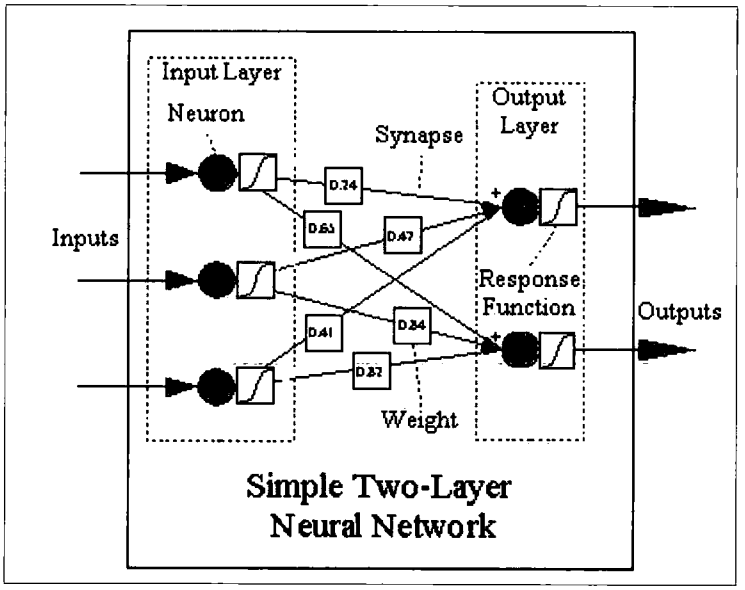


Figure 6.10: Diagram of Two-Layer Neural Network

The neurons are connected by *synapses* (see Figure 6.10), which each have a *weight,* or multiplier, associated with them. These weights can be adjusted in a Backward Pass, to affect the accuracy of the network. The neurons evaluate themselves when all their inputs have updated, so the process starts at the Input layer, and moves, in turn, through

to the Output layer. More complex NNs have recursive connections - a synapse links the output of a neuron to the input of a neuron in an earlier layer - but these are not discussed here further.

Sigmoid Functions are often used as a response function in a neuron. They are defined as strictly increasing functions that exhibit smoothness and asymptotic properties. The logistic function whose equation is shown in Equation 6.11, is a typical function used in NNs as it smoothly limits the output of the neuron between zero and one. The logistic function is used here for reasons that will be explained below.

$$f(x)=\frac{1}{1+e^{-x}}$$

Equation 6.11: The Logistic Sigmoid Function

6.3.3 Global Error

The Global Error of the network is the sum of the Root Mean Square Errors (RMSE), or Mean Square Errors (MSE) across all of the output neurons. This provides the sum of the magnitude of the errors - negative numbers are eliminated by the squaring function.

This Global Error is often discussed and quoted as a quotient of the performance of the network. It also has a second function in many NNs where the network is trained for a designated period. In that instance the Global Error is used to decide when to stop the training, and move to simple recall mode. The Global Error figure, at which training is stopped, can be a highly influential parameter, but is difficult to set optimally - set too high and the network will not produce accurate results as training will terminate too early, but set too low, and the network might never reach the target.

For the application described in this thesis, a separate training session was undesirable and so not used. No threshold was set therefore, but the Global Error is still a useful performance measure of the NN.

The Global Error is expressed as follows, where d is the desired output, o is the actual output and k ranges from zero to the total number of outputs:

$$Global\ Error = \sum_k \frac{1}{2}(d-o)^2$$

Equation 6.12: The Global Error Function

It is of note that the desired output, o, in this equation is usually known beforehand - for example in the case of the standard EX-OR function, the the outputs are well defined. In the case of NN for Q-learning this term must be generated from the reward function at run-time. The reward could be some combination of the inputs, but can also be entirely unconnected- in the case of [Tesauro 1992, 1994, 1995, 2002] a reward function is generated only when TD-Gammon wins an entire game (it is zero otherwise), and is generated externally from the NN.

6.3.4 Error Back-Propagation and Gradient Descent

Our NN is designed to be a function-approximator, however it will start with no knowledge of the domain, so it must be trained to produce the correct function. This is done by updating the synaptic weights by the method of Error Back-Propagation which uses Gradient Descent techniques. This technique is defined in [Rumelhart, Hinton & Williams 1986], but also described in many standard textbooks on NNs, including [Zurada 1992], and the reader is directed there for the full derivation. Here, the results are presented.

Gradient Descent refers to the method that reduces the error to as small as possible by searching for the global minimum in the error function. It does this by updating the synaptic weights in proportion to the rate of change of the error - its gradient. Although the Global Error is an indication that the errors are getting smaller, it is from the individual output neuron that the error signals are derived.

The error on each individual output is used to calculate the weight adjustments for the weights on the synapses directly connected to the output layer. For weights on synapses

not directly connected to the outputs, the Back-Propagation algorithm attributes proportions of the error systematically.

Since the weights are updated according to the gradient of the error, and the outputs of the NN pass through the threshold function, the threshold function used will affect the updates - or more precisely, the *derivative* of the threshold function affects the updates.

To make the calculations viable, it is best that the threshold function has a reasonably simple derivative, and that the function has no discontinuities. A function meeting this description is the Logistic Sigmoid Function, shown in Equation 6.11, which also has the advantage of constraining the output to between zero and one:

As described above, a suitable derivative is equally important to the choice of function, and this is shown in Equation 6.13 :

$$f'(x)=(1-x)x$$

Equation 6.13: The First Derivative of the Logistic Sigmoid Function

Using the Logistic Sigmoid Function and its derivative, the *Output Layer Error Signal*, δ_o, becomes the following, which is a quotient of the error *directly* attributable to the particular output neuron being considered, neuron *k* in this case :

$$\delta_{ok}=\frac{1}{2}(d_k-o_k)(d_k-o_k^2)$$

Equation 6.14: The Output Layer Error Signal

The Hidden Layer Error Signal,δ_h, shown in Figure 6.15 is as follows, where w_{kj} is the weight from that neuron to a neuron in the next layer. It is a quotient of the error being *attributed* (by the particular method, it is not directly attributable) to the non-output neuron being considered, neuron *h* in this case:

$$\delta_{oh}=\frac{1}{2}(1-o_k^2)\sum_K \delta_{ok} w_{kj}$$

Equation 6.15: The Hidden Layer Error Signal

The procedure described above shares out the error found at the output and attributes it equally to all the neurons in the previous layer - all the error could actually be due to a single neuron in a hidden layer, but there is no way to account for this.

6.3.5 Momentum

A common augmentation to the Gradient Descent method is the addition of a momentum term at the point of the weights being adjusted. The momentum term ensures that if the previous weight update was large then so will the next one - it acts like a low-pass filter to noise on the error-term. In terms of Gradient Descent it attempts to stop the solution from being caught in a local minimum - the momentum will carry the system out of it.

The momentum factor, μ, must be less than one for a stable system, and greater than zero. This means that when the update is applied the equation is this:

$$\Delta w_{kj}(t) = \alpha \delta_{ok} y_j + \mu \Delta w_{kj}(t-1)$$

Equation 6.16: The Output Layer Weight Adjustment with Momentum Term

For the Hidden Layer, the update equation becomes:

$$\Delta w_{ji}(t) = \alpha \delta_{yj} y_z + \mu \Delta w_{ji}(t-1)$$

Equation 6.17: The Hidden Layer Weight Adjustment with Momentum Term

The momentum should be set as high as possible to retain a stable system, but is typically "chosen between 0.1 and 0.8" [Zurada 1992, pp 211].

6.3.6 Incremental Learning and Batch Learning

When training, the NN can be run in two different modes. Incremental Learning is when the weights are updated after every Forward Pass of the network. Batch Learning is when the weights are updated only after a number of patterns have been presented.

Often it is perfectly valid to update the NN incrementally, and doing so has the advantage of increasing the speed of convergence. In the work described in this thesis the network is being asked to compare a number of different actions at a single time-step, and so it would be invalid to update the network before they had all been presented - since they should all have the same conditions to be a fair test. Also in Q-Learning, the weights are only updated using the errors on the chosen action output. This means that perhaps three forward passes are done before a single back-propagation is performed.

It is these differences from standard NNs that means that a standard NN-package could not be used. For example, JOONE, or Java Object Oriented Neural Engine, an Open-Source Java implementation of a Neural Network toolkit available from [c] had no facility to update the weights using the correct pattern; or indeed at the correct rate.

6.3.7 The Desired Output

In the normal operation of NNs, for instance in an X-OR function, the desired output is known and well-defined. For example, when the inputs are both on, the output should be off. The error is easily worked out - it is the outputs deviation from zero - and the weights updated.

In the Q-Learning domain however, the desired output is *not* known - it is what we are trying to find. Therefore, the best we can do is supply our best estimation of the desired output. This is the Q-Learning Term stated in Algorithm 6.5; thus our Q-values are actually $\hat{Q}$ -values since they are estimations to the *actual* value, Q, which we are attempting to find.

This was a further barrier to using JOONE, since it expected the desired output to be the form of a plain-text file. Short of the inelegant solution of writing to and reading from text files at run-time, this meant JOONE was unsuitable without heavy modification. Although a bespoke JOONE implementation had been developed for the system, given the other problems already described it was decided a purpose-built implementation of the Neural Network software was necessary; this is described in full later.

6.4 Design of the Q-Learning System

Reinforcement Learning, and the means for implementing it, have now been described. This section can now describe the implementation for this application; this will be done in the order of inputs, outputs, reward function and then the topology of the Neural Network.

6.4.1 Inputs

Firstly, the parameters and measurements that are fed into the network must be considered. These inputs must fully describe the state of the system; or at least any description of the state that is liable to change.

The "encoding scheme", meaning what information is provided to the network, and how it is stated, is an important factor in the effectiveness of the implementation. Information that is easily "understandable" will be of far greater use than that which is more abstract. Considering human sign-language, it is far more efficient to have signs for whole words than spell out every word from an alphabet.

[Tesauro 1992, 1994, 1995, 2002], in which NNs are used to play Backgammon, describes how initially the simple position of each piece was used as inputs, but later more succinct data relevant to good play was encoded and used. The downside to these encoded inputs is that the potential solution space is them limited - although the search is then directed.

In the case of this implementation, some considerable time was spent configuring access to statistics considered to be relevant to the aims of the project. The candidate inputs therefore are:

- Change in Mean of Probabilities
- Difference Between Mean and Instantaneous Probabilities
- Standard Deviation of Probabilities

- Time Since Route Change
- Number of Dropped Ants
- Rate of Ant Production
- Rate of Deposition
- Initial Amount of Pheromone

Since many of these measurements are inter-related, and it is not known exactly what combination of the above will produce the best results, it would be useful to consider different combinations of them. This then is left to the experimental section.

6.4.2 Outputs

The output of the system is the action to be taken. However, it is not the variable in its raw form. The NN is queried as to whether the parameter should be changed (the options being a reduction, an increase or neither). As already discussed in Section 6.1, the parameter to be manipulated is the Initial Amount of Pheromone.

The next issue is how this parameter will be changed by the NN. This will define the number of actions available to the agent, and it may be tempting to allow a large of number of potential actions. The NN handles a large number of states, but attempting to account for a large number of actions as well would require numerous NNs, or a more complex one which must be tested many times, once for each action.

It was decided that it would be better to minimise the number of NNs and use only one per node. This could then be tested once for each action.

6.4.3 The Reward Function

The NN needs to be rewarded in some form so that its actions are directed. This reward is a further input, although is used to adjust the output with its feedback. The reward function is used to calculate the feedback given to the NN - it defines the error in the system.

In the case of this system, the measurements that are inputs to the NN could also be viewed as describing the system. Therefore the candidates to comprise the Reward Function are:

- Change in Mean - if the mean has dropped significantly then the Change in Mean will reflect this, this is a long term measure, and is tuned to the correct time-scale. The mean itself is not used as it will have a constant offset and ensures a drop will provide a negative reward (a punishment).
- Difference Between Mean and Spot-Weight - this measurement gives another description of how much the probability has changed, and will highlight shorter-term trends. If the difference is negative, it means that the route probability is dropping.
- Standard Deviation - this measurement describes how "noisy" the probability has been, it also highlights large changes in the probability. The number will always be positive, but the smaller the better - therefore the negative of the SD should be used as a reward.
- Number of Dropped Ants - the number of ants discarded on a particular link will highlight overflowing queues. This should also be negative.
- Time Since Route Change - the system should be rewarded for being stable. This means that routes that rarely need to be changed should be rewarded.

6.4.4 Topology of the Neural Network

The design of NNs is an inexact science, but rough guidelines exist. To this end a relatively simple design is used. That is to say, the NN used here will initially have, briefly:

- The input layer made up of 8 input neurons, these inputs are described in detail elsewhere. The final inputs must represent which action is being tested - the network is then run once per action.
- A single hidden layer (between inputs and outputs). This layer enables the NN to consider non-linearly separable problems. There are 3 neurons in this layer; matching the number of action inputs.
- The output layer of the NN represents the Q-value of action being tested. This

output is then compared with the outputs for different actions, and the action that produces the largest output (Q-value) is the action that is expected to produce the greatest reward in the long-term.

6.4.5 Training the Neural Network

For the system to perform correctly, the NN needs to be trained for a significant amount of time. Unlike usual NNs this is not off-line training, but is done on-line. A strength of using Q-Learning in this system is that it is an Off-Policy method - meaning that useful information is learnt about an action even when it is not being actively pursued.

If training is not performed for sufficiently long, then the NN will not be able to distinguish between different states and actions as all the internal weights will be too similar. Whilst it is difficult to quantify how long is "long enough", it is likely to be in the tens of thousands of time-steps, or training iterations. The more complex the NN, the larger the number of hidden layers and nodes, the longer the training will take.

6.5 Agent Implementation

Having described the "Intelligence" encapsulated in the Agent, this section describes the structure of the agent.

6.5.1 Agent Structure

The structure of the agent was determined by the three main interfaces. These were: the user interface, the Ant-Colony interface, and the encapsulated Intelligence. Within this there was the storage issue; there was a requirement to be able to add extra entries (i.e. Of new nodes or links) at run-time. These issues are discussed in greater depth in the following sections. Figure 6.18 shows the agent and its required interfaces diagrammatically.

Whilst the ant-colony is embedded within the ns simulator, the Agent runs independently as a separate Java-based thread, implementing the Q-Learning. The Ant-colony and Agent systems communicate over a TCP socket, the ant-colony side driving the Java-system, but producing reports on the state of the ns system.

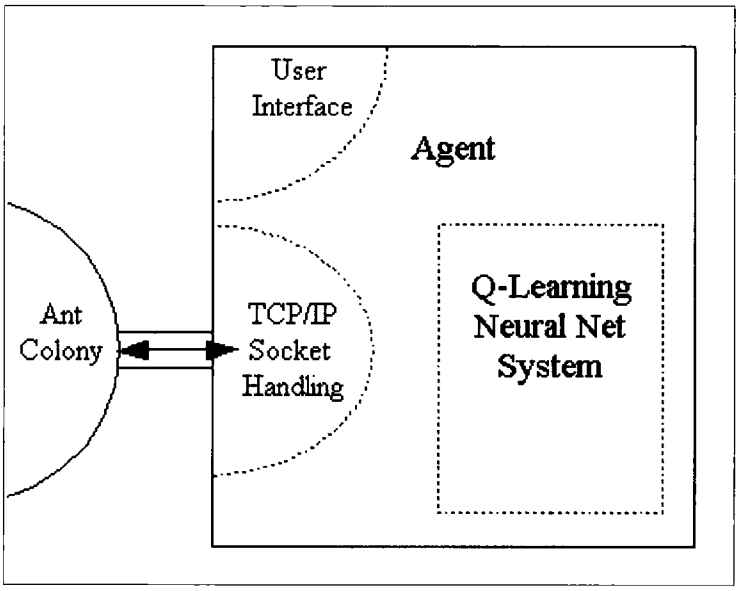


Figure 6.18: Diagram of Agent and its Required Interfaces

The socket handling was designed and developed from basics, based on the standard Socket libraries supplied with Java. A standard ontology had to be developed to allow communication between the two systems. The implementation of this communication system took a three months.

The data structure of the Agent took significant effort, ensuring it could handle an arbitrary number of links and nodes at run-time.

The Neural-Network implementation, after a false start using JOONE (described later) which took up some five months, took a week to build from nothing, with no prior experience of implementing Neural Network software. However, this was to build the infrastructure with the synapse-weight updating algorithms in a suitable way to support Q-Learning.

The Q-Learning mechanisms that then ran the NN took some four months to perfect also. This also included finding the correct reward mechanism for the system to encourage it to take the correct actions.

The agent was developed without the use of Zeus, however, it was designed to fit into the Zeus framework at a later date, and thus to fit into the Agent-System described in Chapter 4.

6.5.2 Agent Process

Figure 6.19, overleaf, shows an overview of the process the Agent goes through on each time step of the system. Note that this is different from a standard NN, as training can only be applied after the next time step has occurred, since the reward function measures the performance over the intervening time step.

6.5.3 System Simplification

Due to the level of complexity of the system described, some simplification of the system was required during development. Rather than using separate NNs for each node, a single one was used. This does not imply global knowledge of conditions across the network, but more shared learning. Each node has access to the NN in turn, at which time it performs its consultation of the network.

The benefits of this shared learning is a significant compression of the time required for simulation, since it can learn from every action on the network. The downside of the simplification is that it may affect the accuracy of the function learnt by the NN since conditions on each node are likely to differ. However this can be investigated at a later stage of the development of the system.

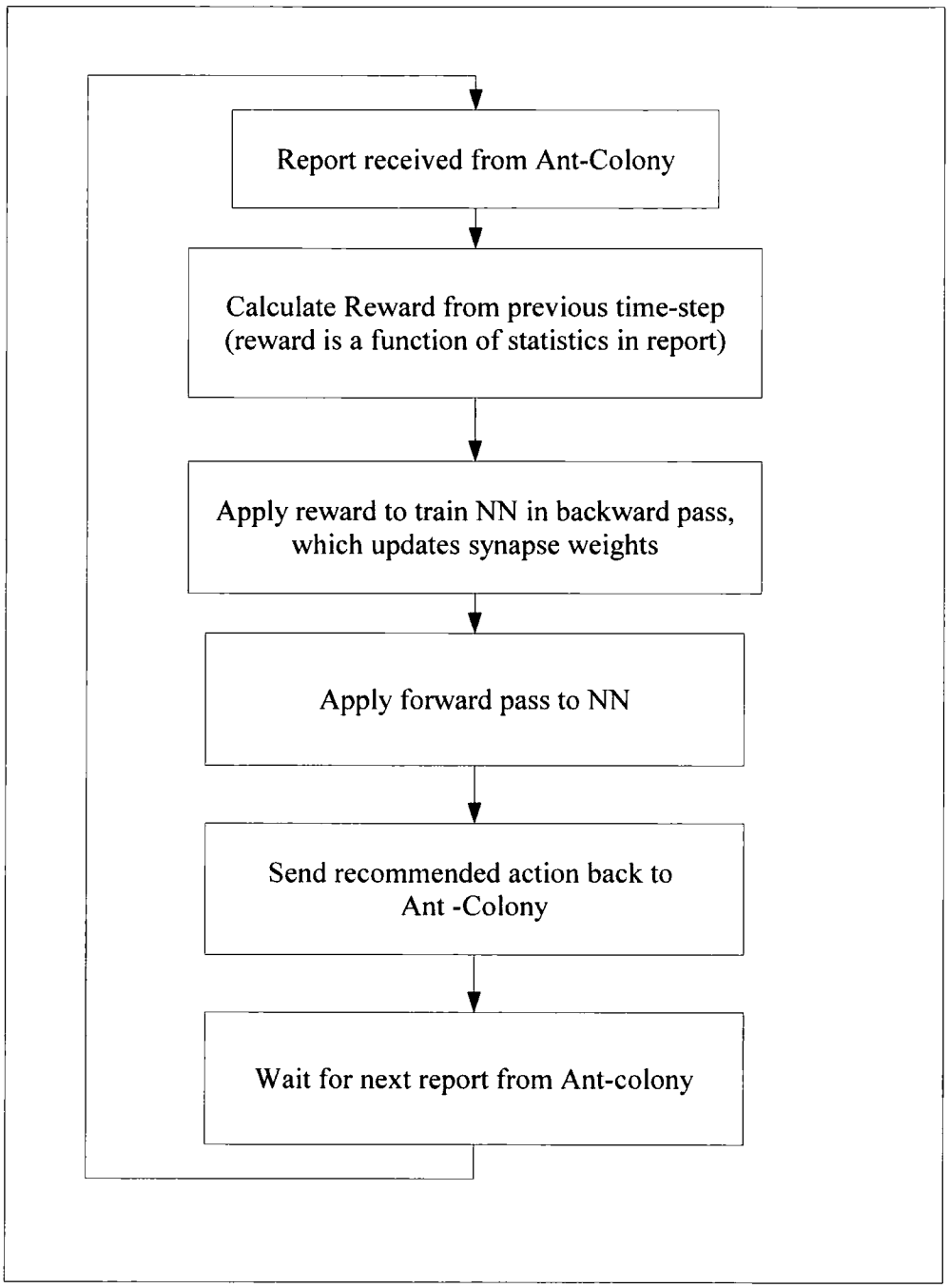


Figure 6.19: Flow Diagram of Agent process for each time step

6.5.4 Agent Interfaces

The agent itself is made up of a number of Java classes. A UML-based diagram of the structure of the Agent is shown in Figure 6.20. Only the major methods are shown and

the method parameters are omitted. The normal lines between the classes - formally *association* lines - in this context, show lines of communication as well as instantiation. The lines ending in diamond-shapes show *aggregation*, which means that they are part of the parent class.

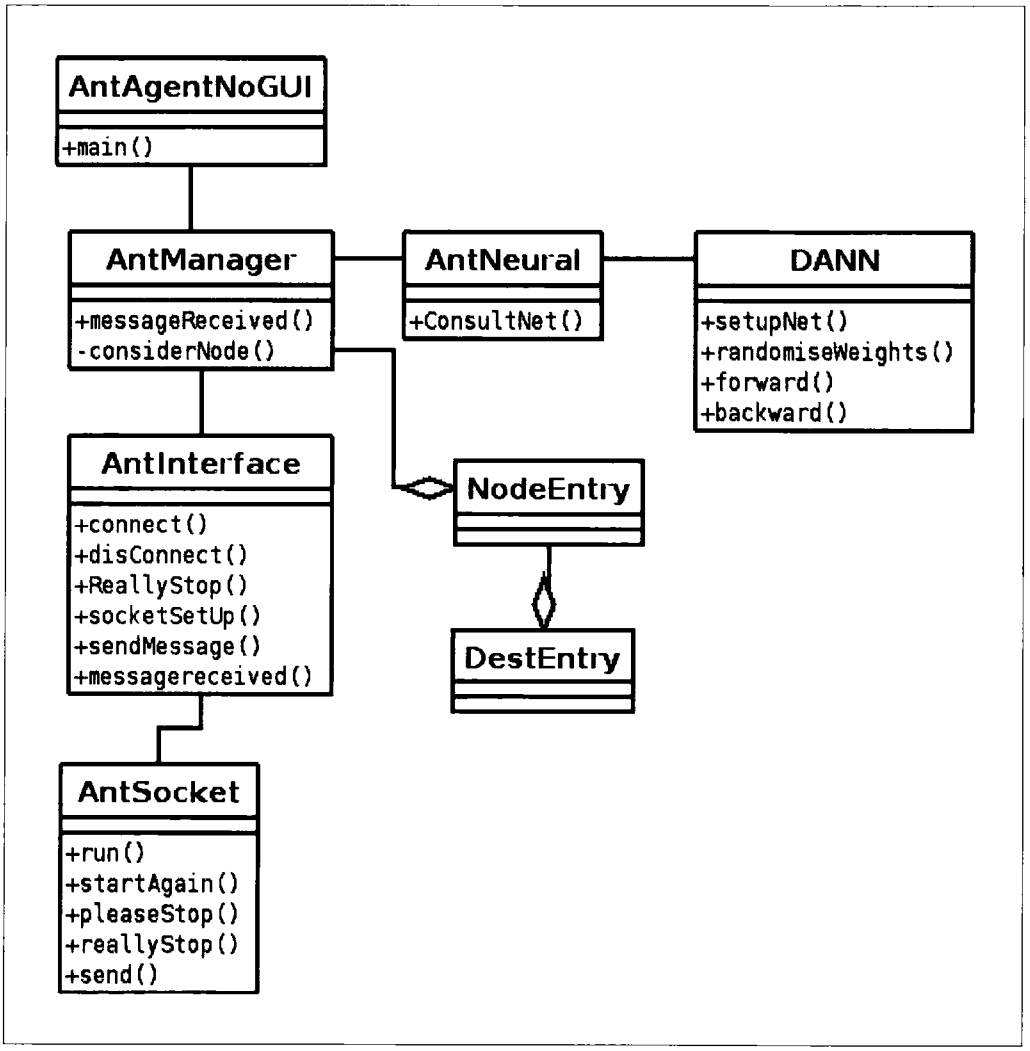


Figure 6.20: UML diagram of class structure

`AntAgentNoGUI` is a simple user interface program run from a prompt and was an alternative to `AntAgentGUI` which was not developed past the initial stages. `AntAgentNoGUI` starts a `main` thread to establish the `AntManager` class to run the Agent. The Interfaces for `AntAgentNoGUI` and `AntManager` are shown in Figure 6.22. Internal

to `AntManager` were two "class classes", `NodeEntry` and `DestEntry`, which are essentially data-structures, defined to store information about the links and nodes of the network. These are shown in the UML diagram, Figure 6.20, as aggregation classes. Figure 6.21 shows an simple example of a snapshot, taken at runtime, to give an idea of the topology of the data-structures.

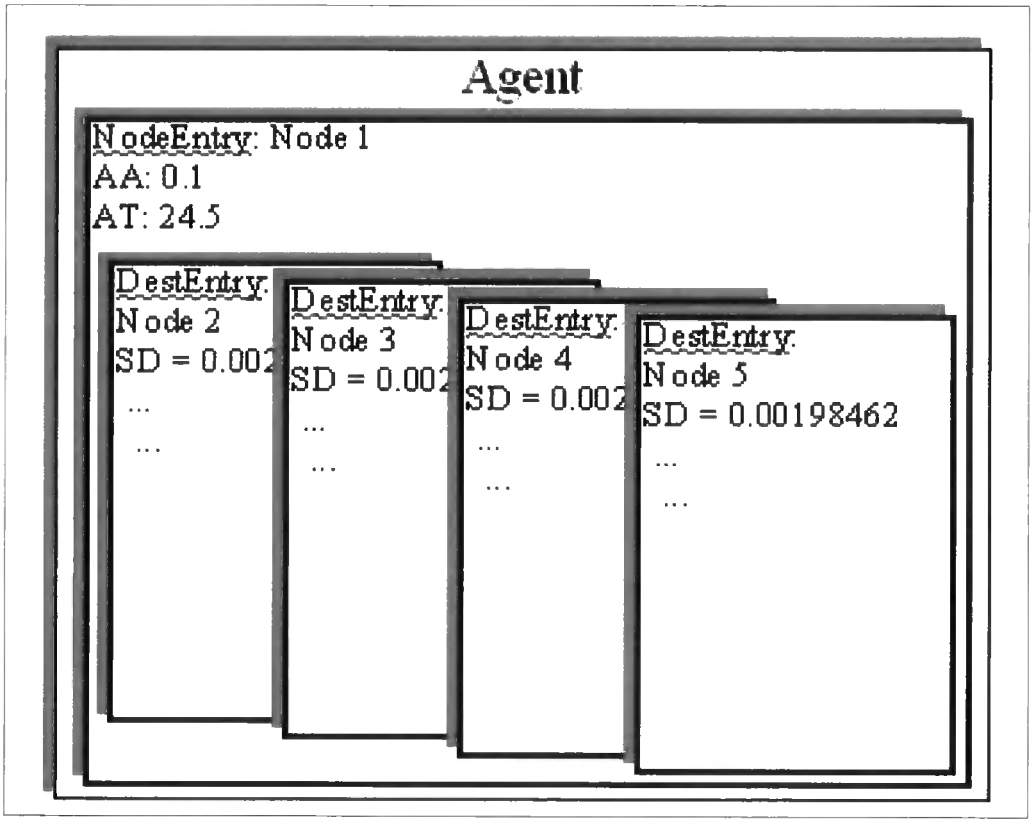


Figure 6.21: Example snapshot of `NodeEntry` and `DestEntry` Data Structures

The Java constructs these used to implement `NodeEntry` and `DestEntry` were ArrayLists. Which are more suitable in this case than a LinkedList - since they are *generally* set up at the beginning and then although referred to, not created or destroyed - or a simple Array, which requires a definition of its size too early for the this situation. Both `NodeEntry` and `DestEntry` implement the *clone* method inherited from *clonable* to allow the instances to be copied fully - i.e. to obtain a second identical instance of the original, as otherwise an apparent copy of the object was merely another handle, or

pointer, to the original instance.

The `AntManager` class has two main methods, shown in Figure 6.22, that signify the arrival of a report from the Ant-Colony, `messageReceived`, and that of consulting the NN to consider the report just received, `considerNode`. Although this is declared as `private`, it is the main function of the class, so is shown (commented out).

```
public class AntAgentNoGUI
{
   public static void version( ) ;
   public static void usage( ) ;
   public static void main( String[] argy ) ;
}

public interface AntManager
{
   class nodeEntry implements Cloneable { public Object clone( ) ; }
   class destEntry implements Cloneable { public Object clone( ) ; }

   public StringBuffer messageReceived( StringBuffer MESSAGE_STRING ) ;
// private StringBuffer considerNode ( int nodenumber,
        AntNeural antneural ) ;
}
```

Figure 6.22: Definition of Java Interface for the Agent

Below the `AntManager` two branches of the hierarchy exist. This is shown in Figure 6.20, the UML diagram. The two branches are the communication with the Ant-Colony, and the handling of the NN. Firstly, the Ant-Colony interface is described.

Since communication is prompted by the Ant-Colony generating a report during a simulation, the Agent's role is to listen to a TCP socket on a pre-defined port-number, as visualised in Figure 6.18. This is similar to the ISCA-Agent in the earlier Agent-System - a separate thread of control is required to handle this, defined in the `AntSocket` class. Another class, `AntInterface`, handles the incoming messages, and passes them onto `AntManager`. `AntInterface` also handles the return messages as well and is defined in Figure 6.23.

Once the `AntManager` class has the information from the colony, it consults the Neural Network. The `AntNeural` class handles the input and output of the NN, and is

essentially the static framework for the NN.

```
public interface AntInterface
{
   public static int connect( int PORT_NUM ) ;
   public static int disConnect( ) ;
   public static void ReallyStop( ) ;
   public static void socketSetUp( ) ;
   public static int sendMessage( StringBuffer MESSAGE_STRING ) ;
   public static void messageReceived( StringBuffer MESSAGE_STRING ) ;
}

public interface AntSocket extends Thread
{
   public void run( ) ;
   public void startAgain( ) ;
   public void pleaseStop( ) ;
   public void reallyStop( ) ;
   public static int send( StringBuffer MESSAGE_STRING ) ;
}
```

Figure 6.23: Definition of Java Interface for the Agent's Ant-Interface

The instance of the NN itself - including the weights and synapses etc., i.e. the dynamic part - was implemented in the `DANN` class (David's Artificial Neural Network). The `AntNeural` Interface is also shown in Figure 6.24. DANN was developed from nothing, after JOONE had been rejected as inappropriate, and incorporated the differences from a "normal" NN described previously.

The DANN class has two main functions, that of forward recall (`forward`), and that of backward update (of the synaptic weights), called `backward`. Further to these, there was a configuration method, `setupNet`, and a randomisation method (`randomiseWeights`) that set the weights between -0.1 and 0.1. There were methods to set all the various parameters (Learning Rate, Momentum etc.) and one to implement the Sigmoid function (`sigmoid`) which is heavily used. Finally, a `toString` function was provided, along with a method to print out the synaptic weights, `writeWeightsOut`. The `toString` method is Java-programming good-practice, and provides a standard interface to query the state of a class.

The update of the weights occurred immediately before the following forward pass, rather than straight after the previous one, as the learning rule depends on the outcome

of the action taken at the previous time-step.

```
public interface AntNeural
{
   public double[][][] ConsultNet( double[][] inputArray,
      double previousNodeValues[][] ) ;
   public String toString( ) ;
   public void writeWeightsOut( String suffix ) ;
}

public interface DANN
{
   public void setupNet( int[] numNodesInEachLayer ) ;
   public void randomiseWeights( ) ;

   public double[][][] forward( double[][] inputArray ) ;
   public void backward( double reward ,
      double[][] previousNodeValues ) ;

   public double sigmoid( double input ) ;

   public void setLearning( boolean newLearning ) ;
   public void setBatch( boolean newBatch ) ;
   public void setSeed( long newSeed ) ;
   public void setLearningRate( double newLearningRate ) ;
   public void setMomentum( double newMomentum ) ;
   public void setDiscountRate( double newDiscountRate ) ;
   public void setReward( double newReward ) ;

   public String toString( ) ;
   public void writeWeightsOut( String suffix ) ;
}
```

Figure 6.24: Definition of Java Interface for the Agents Neural Network

6.5.5 Ant-Colony Agent Communication

The communication between the agent and the ant-colony uses a simple TCP socket. Over this link a simple protocol is used, which allows concise communication, but is sufficiently flexible for development, and is robust; using acknowledgements to confirm the receipt of messages.

A message is made up of information about an entire node and the links from it. These were listed which each item having a two-letter code, e.g. AA for Ant-Age, AD for Ant-Destination etc.

The characters arrive over the TCP socket, where the `AntSocket` process is listening.

The string of characters is passed to the `AntInterface` class through its messageReceived method, which in turn calls the `AntManager messageReceived` method. Here the message is parsed and the information is logged in the correct `nodeEntry` and `destEntry` within that - an example structure being shown in Figure 6.21.

6.6 Initial Evaluation

Eventually: having developed the agent framework; developed a protocol to allow the agent to communicate with the ant-colony; designed and implemented the storage mechanism for the agent to handle an arbitrary number of links and nodes within a network; implemented a Neural Network system from scratch, including the adjustments to allow Q-learning instead of the standard feedback inputs; and established a reward function that would promote the effective operation of the network; it was possible to run the system attached to the Ant-Based Routing system.

To aid the development of the system, two simple networks were used to test the system, and it is results of this investigation that are described in this section, before a more complex, but more realistic network is used to validate the system in the next section.

The aim of the experimentation is to show that there is an improvement in performance when the parameters governing the ants are changed dynamically at run-time. In this way the system can take account of changing conditions on the network.

6.6.1 Experimentation

The results presented here are a comparison of different solutions, although each is based on the use of Ant-Based Routing. As such, graphs of the results are presented in the same form as those of the previous chapter. However, when manipulation of the parameters occurs, it is necessary to incorporate this information into the graph as well.

Although graphs exist for each of the results described, most have been provided only in the Appendix. For clarity these are cross referenced in the text.

The first solution uses the ordinary Ant-Based Routing Scheme, described in the previous Chapter. No adaptation of the parameters is involved. This is used as the control test, and it is expected that the other solutions will out-perform it.

In the second solution, the Ant-Based Routing Scheme is augmented by an *Ideal Agent*. This represents the bound on the best possible strategy that the proposed Agent would be able to achieve. Indeed, the Ideal Agent strategy is unrealisable, outside of these simulations, since the strategy is based on *a priori* knowledge of the traffic. This strategy should however provide evidence that it is desirable to be able to manipulate the parameters, in whatever method - it should show that improvements to the Ant Control are possible.

Thirdly, a *Simple Reactive* agent is implemented. This uses a simple thresholding mechanism that increases the *Initial Pheromone* level when the *Standard Deviation* exceeds a certain level, but decreases it when it is below the threshold.

This strategy should outperform the ordinary Ant-Based Routing System, but in a comparison with the proposed Q-Learning agent should not be as successful; the approach is included here to validate that the learning mechanism provides an advantage over such a simple threshold mechanism.

Finally the proposed Q-Learning agent is tested and is able to change the parameters as it sees fit at runtime, according to strategies it learns.

The results are presented in the form of graphs against time of the probabilities for each next-hop from the source node of user-traffic. Where applicable, they are presented in combination with graphs of the parameters being changed by the agent strategy.

Of interest in each test is the reaction of the ant-system (both managed and unmanaged)

to congestion; the speed of response is of importance. A further concern is that the routing should not constantly change between two or more routes - *route-flapping*.

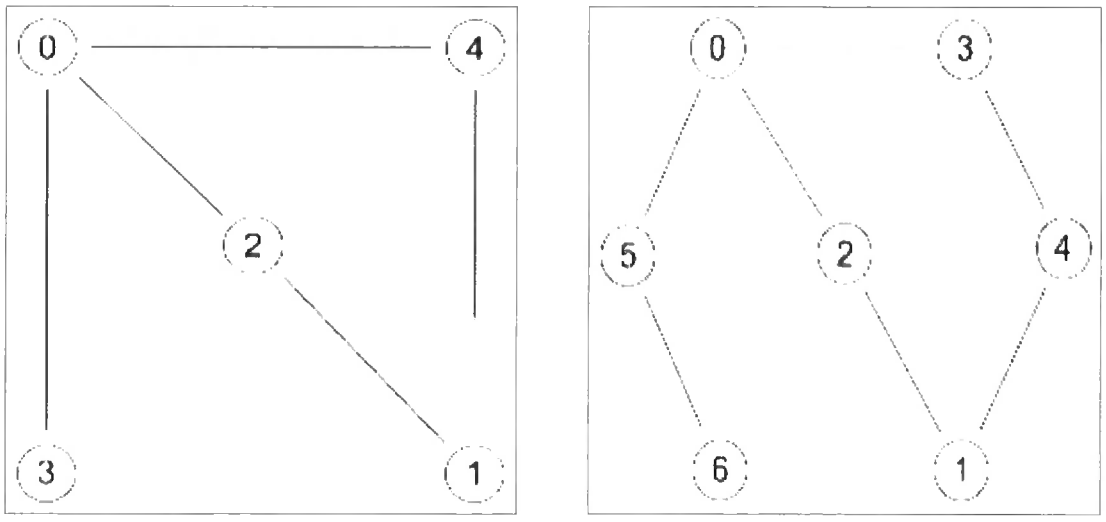
A useful measure, as already alluded to, is the Standard Deviation of the highest probability. This gives an indication of the *noise* on the probabilities and the stability.

6.6.2 Test Networks

In the development of the system, two networks were used to test the Agent-Managed Ants. This ensures that the system can adapt to different scenarios, and does not merely provide a solution by chance. In each case the results concentrate on the routing decision between two (non-adjacent) nodes, labelled Node 0 and Node 1 in both cases.

Network One is shown in Figure 6.25(a). It is a simple five node network. The important feature, as stated, is that there are three equally-long routes between Node 0 and Node 1. One of these routes will at times be congested.

Network Two is shown in Figure 6.25(b). It is a seven node network with two equal-length routes and one shorter. It is the shorter route that will, at times, be congested.



(a) (b)

Figure 6.25: Diagrams of (a) Network One and (b) Network Two

When the shortest route becomes congested, the ants must react as quickly as possible to the congestion. They must overcome a large amount of inertia, represented by the high probability built-up by the shortest route - this inertia is greater than that of the previous scenario.

6.6.3 The Traffic Scenario

Both network scenarios will be required to transport some user-data, which we shall describe as real-time, loss-sensitive data. This shall be transported from Node 0 to Node 1, in both network topologies. The traffic is offered to the network on an on-off basis for bursts of 15s duration.

Problems would be caused by the network dropping the user-data, excessive delay or excessive *jitter* - variation in the inter-arrival time of packets, which is often caused by excessive switching of routes.

The situation under which the network will be tested is as follows. The traffic causing congestion shall represent other network load and will be set at a level that will not, by itself, cause packets to be dropped (other than ants).

Two flows of traffic are involved. Firstly, a constant-bit-rate (CBR) flow, that uses the majority of the bandwidth available on the link between Node 0 and Node 2. Secondly, a smaller Variable-Bit-Rate (VBR) flow also uses the same link. Both flows stop after fifteen seconds.

This combination of traffic will result in intermittent congestion during the time when both flows are present on the network. Traffic will require storage in queues during congestion whilst it is waiting to be transmitted. Ant-packets will be dropped having been assigned a lower priority than other traffic.

This pattern of traffic is repeated, at irregular intervals, to provide both the training and the testing of the Agent-Managed-Ant System.

6.6.4 Ant-Based Routing

This set of results uses "standard" Ant-Based Routing and as such is the control test in these experiments. The details of the configuration are provided in the previous Chapter, Section 5.5.4.

Figures 6.26 and 6.27 show the probability graphs for each route from Node 1 to Node 0 for Network One and Network Two respectively. Before congestion occurs, the SD of the probabilities drops to as low as 7.06E-5. The network stabilises almost immediately after congestion starts since there is nothing to choose between the three routes. In this scenario, one route must be chosen, however arbitrarily, for the sake of network stability.

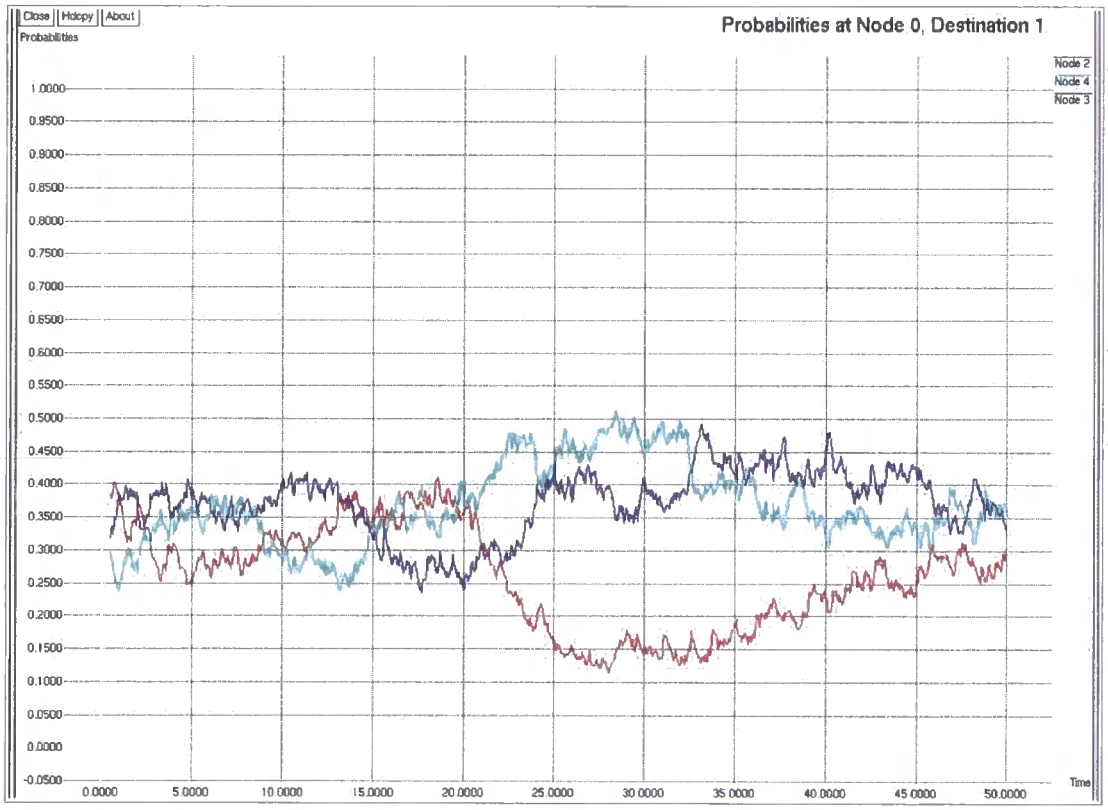


Figure 6.26: Graph of probabilities for Ant-Based Routing on Network One

When the traffic conditions change, at 20s, the ants take a little time to react but the control traffic is not using the congested route. The SD peaks at 1.17E-3 and remains

around 5.56E-3 for the duration of the congestion. However, it can be observed that the probability of the congested route (represented by the red trace) falls sharply, from this time. After 35s, the route starts to recover, showing that the system implements constant exploration of the network, in search of the best route.

In the case of Network Two, Figure 6.27, there is a definite shortest route to begin with, and the probabilities reflect this from the start. The shortest route, via node 2 (again represented by the red trace), receives the majority of the reinforcements provided by the ant-packets moving around the network. This means that an advantage is built up by this "best" route, meaning that the user-traffic will be routed by this route, as it should be. During this time, the SD falls to 1.04E-4. However, after 20s when the congestion starts to occur on the shortest route, this *advantage* for the shortest route turns into a *dis*advantage in terms of the network and traffic as a whole.

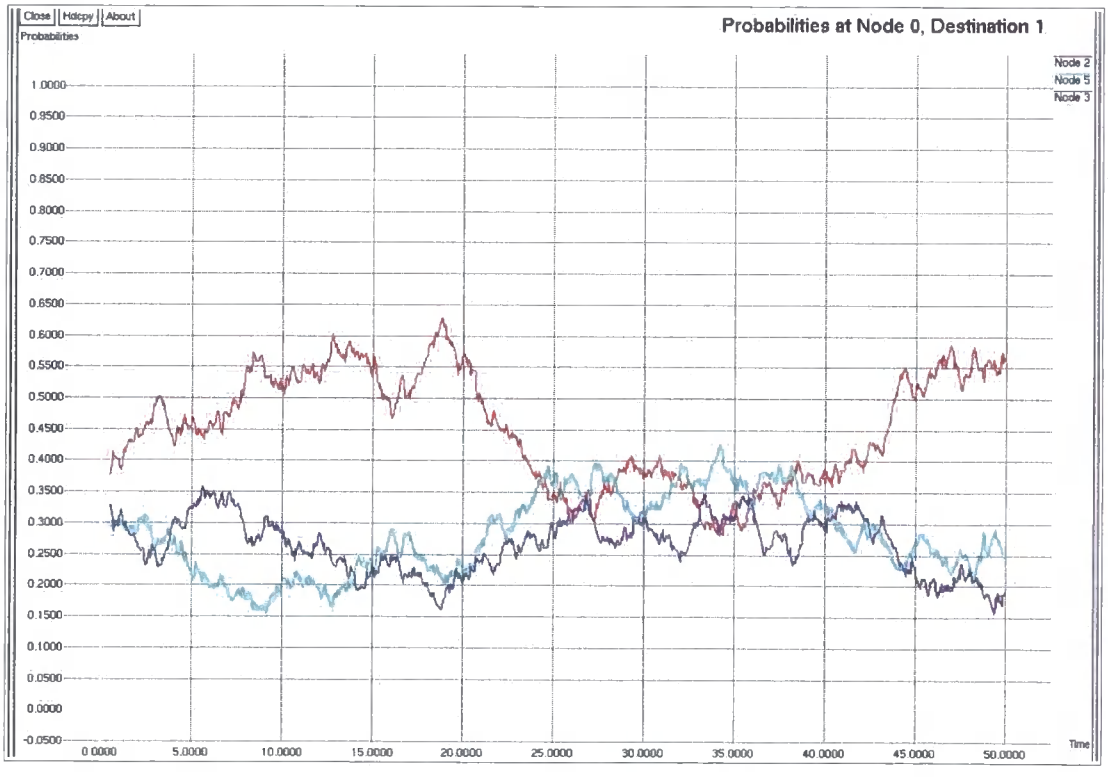


Figure 6.27: Graph of probabilities for Ant-Based Routing on Network Two

It can be seen that the other routes' probabilities in Figure 6.27 take significant time or

overcome this difference in probability. The SD of the probability reflects this by having a value of 2.24E-3 during the congestion, with a peak value 6.80E-3. The route changeover occurs after 4.30s of congestion, during which time a significant amount of traffic is dropped.

Whilst the congestion occurs, the probability of the actual shortest route is suppressed as the link is overloaded. However, this does not prevent it being the selected route for a portion of this congestion - but this is during a period when the congestion has slightly abated.

At the end of the time of congestion, at 35s in Figure 6.27, it can be seen that the probability recovers to its previous level from before the time of congestion, but takes some time in doing so.

6.6.5 Ideal Agent Solution

This set of results utilises a pre-programmed attempt at the solution to the problem. It is a simple hypothesis of what a hypothetical *ideal agent* might do. However, through having knowledge of the traffic-levels beforehand, a certain advantage is evident. Although this strategy may be effective in this particular circumstance, it is likely to be a fragile solution, being unsuited to scenarios it was not specifically designed for - i.e. every other situation apart from than the one just tested.

In both scenarios, the graphs of which are provided in Appendix A, the solution promotes exploration of other routes, and emphasises the congestion present on the network, since any ants in excess of the link capacity are discarded.

Because the routes are equally-probable from the beginning, no settling time is required by the network, but the ants change the best route frequently to begin with.

Again on Network One, the congested route is not being used by the user-traffic when the congestion starts, although it has only just fallen below another route. It can be seen

however that the probability of the congested route drops off more sharply compared to the ordinary ant-based routing, and then recovers faster when the congestion is removed. The SD over this period is 5.83E-4 before congestion occurs, and is 3.03E-3 during the whole period of congestion, peaking at 8.09E-3.

For Network Two an improvement in the response is shown by the change of route occurring after only 3.60s. This is an improvement of more than a second to the ordinary ant system's response - almost a 20% reduction in time to decisively respond. When the congestion stops, the shortest route quickly recovers a large probability, and the system finds its equilibrium again. The SD in this case shows the same pattern as in Network one; having a value of 8.89E-5 before congestion, a value of 2.62E-3 over the whole period of congestion, but a peak value of 8.09E-3 immediately after congestion starts.

6.6.6 Simple Reactive Agent Managed Ants

This solution uses an agent that simply changes the ant's parameters (the *Deposition Rate* in this case) when the *Standard Deviation*, *SD*, exceeds a threshold,

In manually setting a threshold on the SD a number of issues become apparent. The level of the threshold must be thoroughly researched, otherwise the level could be inappropriate. This level could also be overly sensitive to the topology. A node with only two links is likely to have a lower SD than a node with four links, since fewer ants are likely to arrive. Therefore this solution is not likely to be especially robust, although more robust than the previous hard-coded solution.

The solution is considered in this thesis to provide proof that an active learning mechanism is required, and actually provides an advantage over solutions such as this.

The results from the two networks are again shown in Appendix A (A.5 and A.6). These figures are made up of two graphs. Each graph has a noisy section at the beginning. This is due to a large SD caused by the initialisation of the system. This triggers the threshold mechanism, which increases the effect each ant has on the probabilities, making the

probabilities more noisy.

For Network One the SD drops to a value of 1.39E-4 in the pre-congestion period, up to 20s. The route changes after 1.2s, whilst the SD peaks at 6.78E-3, with a value of 3.03E-3 over the entire period of congestion. After the congestion the system is sluggish to recover; however, the recovered route is only as good as the other two routes so no harm is done.

The result from Network Two, anomalously gives a time of 4.3s, compared with the 4.2s of the ordinary Ant-System. However it can be seen that the agent has an effect, increasing the speed of response when congestion is present on the system. For the duration of the congestion the probability of the shortest route is suppressed, but it quickly recovers without the need of the agents manipulation. The SD shows a value of 1.34E-4 before congestion, and 2.17E-3 after, with a peak of 6.29E-3.

6.6.7 Q-Learning Agent Managed Ant Solution

Figures 6.28 and 6.29 show the final solution presented here, the Agent-Managed Ant System should provide a realisable solution which responds in a time comparable to the solutions described so far. The ordinary Ant-System should be outperformed by the Agent, but the hard-coded solution is the ideal - which the Agents should not actually be able to achieve. The Simple Reactive agent may well in some circumstances be comparable to the more sophisticated Q-Learning Agent, but as it does not adapt its strategy (it has a set threshold) it will not perform as well over varying scenarios.

Figures 6.28 and 6.29 show the graphs for the Q-Learning Agent on Network One. They are included here as well as Appendix A, as they are of a slightly different form. The lower portion shows when the agent takes the action of changing the system's parameters. Figure 6.29 is of a larger scale than those previously, hence it looks larger and more noisy; however this is necessary as the detail would be obscured if seen in the same scale. It can be seen that the agent reacts to congestion, and takes the appropriate action which is shown in the lower portion of graph, with the rising line showing the

agent manipulating the IPL. It can also be seen by the increase in noise, and steeper gradient of the probability of the congested route. The SD confirm this behaviour, having a value of 1.07E-4 before congestion, 2.60E-3 after, but peaking at 5.32E-03.

Figure 6.29 shows the agents action on Network One in more detail. It can be seen that the agent reacts to the fall in the probability of the congested route in reasonable time.

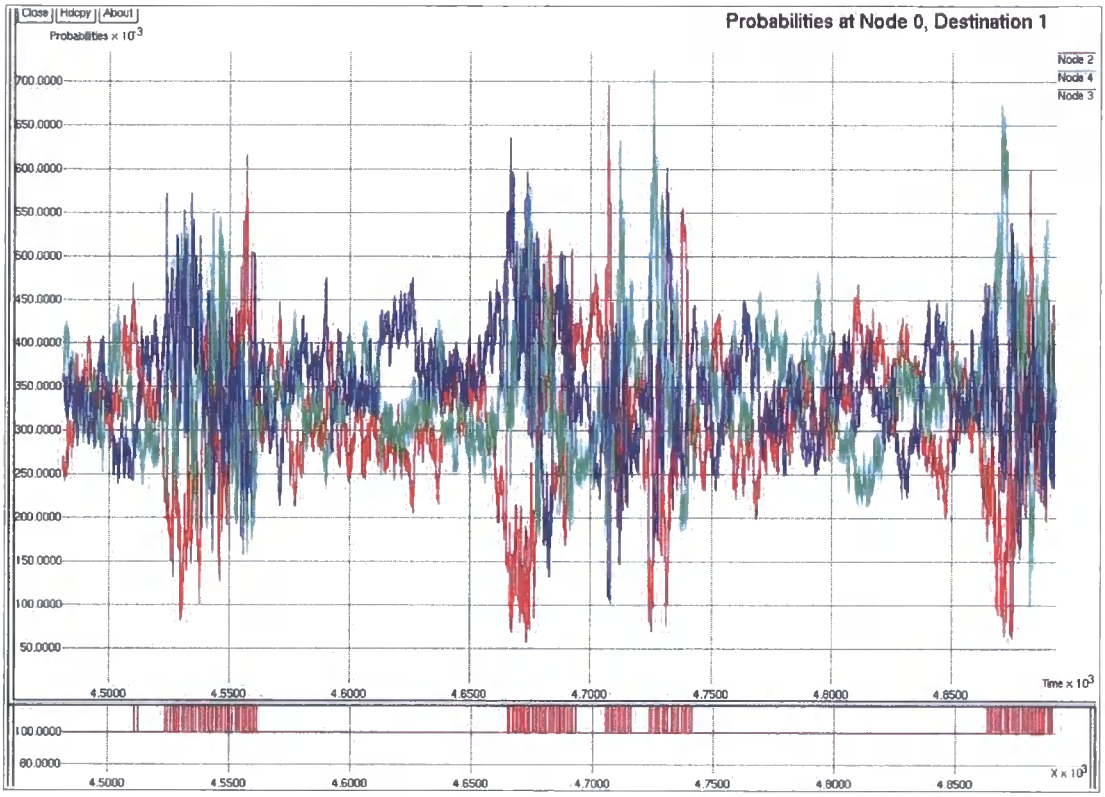


Figure 6.28: Graphs to show the Agent reacting to congestion on Network One

Figure 6.30 shows the response to congestion in the Ant-Managed system on Network Two. The route change occurs at 3.89s. This is an improvement over the 4.3s of the Ant-System, but does not improve on the hard-coded solution of 3.60s as was expected. It is noticeably more *noisy* than other graphs, not only because of the increased effect each ant has, but also because of the closeness of the probabilities and the fact that they are always similar. This is shown in the SD, which has a value of only 1.85E-4 before congestion, which peaks at 4.26E-3 after, and an overall value of 1.64E-3. These results are summarised and discussed in the next section.

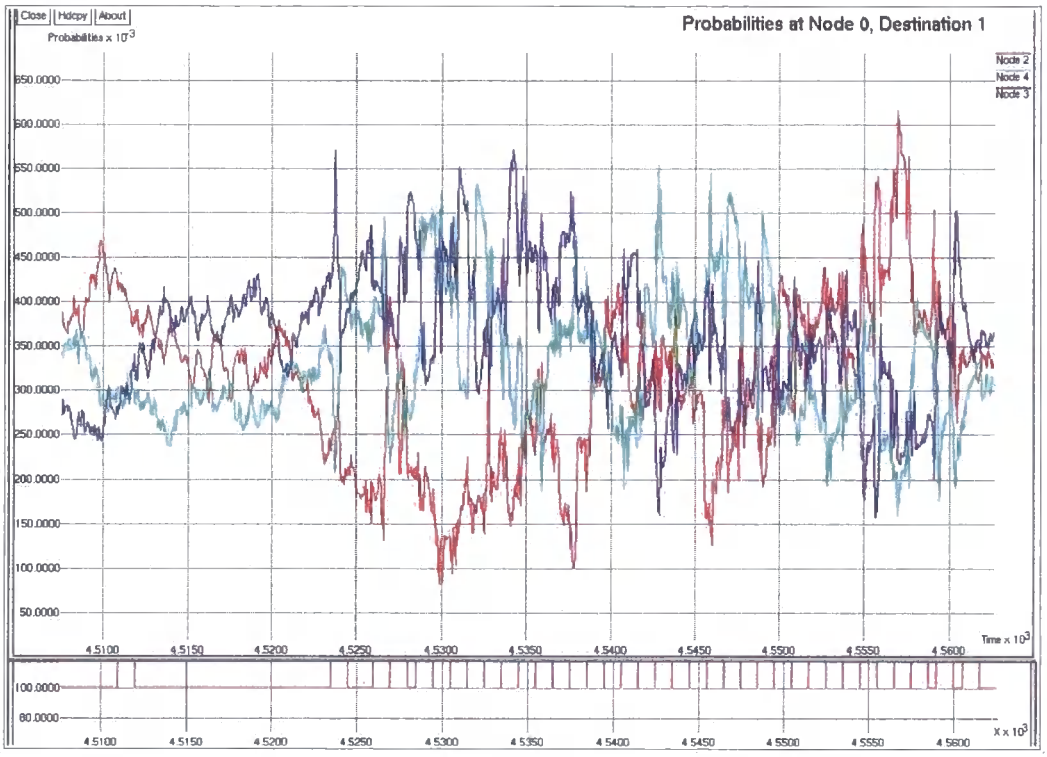


Figure 6.29: Graphs to show the Agent on Network One in Detail

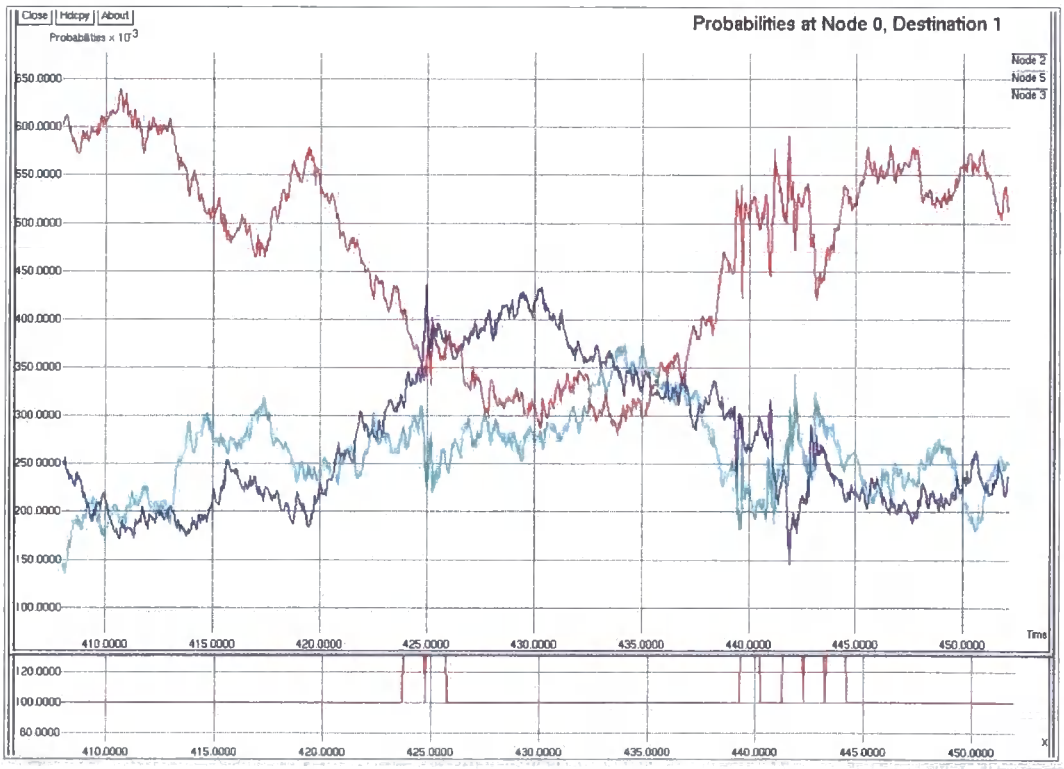


Figure 6.30: Graphs to show the Agent reacting to congestion on Network Two

6.6.8 Summary of Results

This section has described the initial testing of the Agent-Managed Ant system. The next section goes on to test the system on a larger, more complex network, which is more realistic in its topology, it is a sparsely-connected network, which is more able to be generalised.

The results presented in this section are summarised below, in Figure 6.31 and Figure 6.32. It can be seen that the reaction time on Network One is not comparable as it is rarely applicable. However, it can be seen that the standard deviation increases significantly when congestion occurs. The schemes that react to this factor - all but the ordinary Ant-Based Routing (ABR in the tables) improve the performance.

One	Reaction Time	SD low	Low Time	SD high	High Time	SD congestion
ABR	-	7.06E-5	18.5	1.17E-3	23.0	5.56E-3
Hardcoded	-	5.83E-5	11.0	8.09E-3	29.0	3.03E-3
Threshold	1.2	1.39E-4	18.5	6.78E-4	27.0	3.47E-4
Q-Learning	-	1.07E-4	18.0	5.32E-3	22.5	2.60E-3

Figure 6.31: Table to show different Agent performances on Network One

Two	Reaction Time	SD low	Low Time	SD high	High Time	SD congestion
ABR	4.3	1.04E-4	17.0	6.80E-3	23.0	2.24E-3
Hardcoded	3.6	8.89E-5	11.5	8.09E-3	22.0	2.62E-3
Threshold	4.3	1.34E-4	12.0	6.29E-3	22.5	2.17E-3
Q-Learning	3.9	1.85E-4	416.3	4.26E-3	422.5	1.64E-3

Figure 6.32: Table to show different Agent performances on Network Two

Figure 6.32 summarises the results for Network Two. It is clear here that the Q-learning Agent and the Hardcoded solution improves the performance of the Ant-Based Routing system because it improves the speed of response in times of congestion.

To summarise the results so far it has been shown that the four different solutions

perform as expected over the two networks. Each is based on the use of the Ant-Based Routing System, as described in the previous chapter, and this basic system is used as the control test for comparison with the others. Whilst a simple threshold mechanism was found to produce satisfactory results it is also a fragile solution, needing to be tuned to its environment manually (by changing the threshold). The *Ideal* agent performs the best, but is unrealisable, since it requires prior knowledge of the traffic conditions. Finally, the learning agent performs satisfactorily, and demonstrates the potential for AI techniques to be applied to real-world, complex problem domains.

The next section takes the solutions described here and applies them to a larger network, to demonstrate that the Q-Learning solution is not restricted to simple networks.

6.7 Agent Managed Ants on a Generalised Network

Having performed initial tests on the Agent system that were used in the development of the techniques, it is now required to test the system on a more complex network. Rather than apply the techniques to simulations of the topology of a national telephone network, which is the approach taken by [Schoonderwoerd et al. 1996, 1997], [Vittori 2001], the approach taken here is to apply them to a generalised network - i.e. a sparsely connected network, Network Three. The network to be investigated is a configuration where no node is a spur, but the network is not fully connected. This network is shown in Figure 6.33.

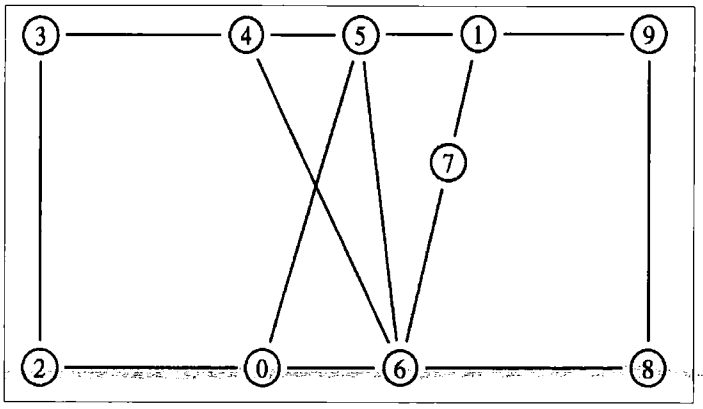


Figure 6.33: Diagram of Sparsely Connected Network

6.7.1 Traffic Scenario

The traffic scenario used here will be based upon the scenario described in the previous section. The main feature is again congestion on the network, specifically between Nodes 1 and 0. This congestion is split into specific time periods of about 15s, but is still intermittent within those periods, with a combination of Constant Bit Rate (CBR) and Variable Bit Rate)

6.7.2 Ant-Based Routing System

The ordinary Ant Based Routing System does react to the congestion, but only changes the route after 9.1s, the graph is included in the appendix, Figure A.10. This is a significantly longer time in comparison to Networks One and Two, primarily due to the size of the network, but also due to the large difference in the probabilities when in the uncongested steady state.

The SD just before congestion starts has dropped to 9.78E-5. The probability then starts to drop off at 20s, when the congestion starts, but due to the lack of adaptation of the ant parameters, the system must respond at a fixed rate which is unsuited to this situation. During this time, the SD reaches a value of 3.08E-2, but overall has a value, for congestion, of 4.03E-02.

6.7.3 Ideal Agent Solution

The Ant-System managed by the Ideal Agent shows significant improvement over the ordinary Ant-Routing. Figure A.11 shows the Ideal Agent operating on Network Three. The route change occurs after 7.1s, and in this time the probabilities have had to change by a large amount, but have been assisted by the ant parameters being manipulated to increase the speed of response. The SD values range from 9.78E-5 before congestion, to 6.14E-3 during congestion and, peaking at 8.16E-3.

6.7.4 Simple Reactive Agent

The Simple Reactive Agent takes some time before switching routes, only improving on the performance of the ordinary Ant-System by 0.1s, taking 9.0s to do so. This graph is also in the appendix, Figure A.12. The manipulation of ant-parameters is triggered frequently and not just when it should be. This shows the fragility of the system which was designed for the characteristics of Network One and Two. Due to the size of the network, the SD tends to be larger, so the simple threshold is set too low in this instance. The actual SD values are 2.38E-4 initially, 1.94E-2 at the peak of congestion, and 6.41E-3 for the congestion period overall.

6.7.5 Agent-Managed Ant System

Figure 6.35 shows the response of the Q-Learning Agent to the problem described. It can be seen that the Agent responds to the large changes on the network caused by congestion by increasing the response speed of the network. It does this consistently, and then returns the Pheromone levels to their lower values, once the congestion has gone.

It can be seen from Figure 6.34 that the parameter is manipulated (shown in the bottom graph) when a drop in the highest probability is encountered. This manipulation means that the route change will proceed faster than if unaided. This graph also shows that the system works consistently across time. There are two periods of congestion covered in this graphic, with the probabilities recovering in between time. Although the the parameter is not returned immediately after the second period of congestion, it does at the end of the graphic.

The SD of the highest probabilities range from values 7.05E-4 when settled, to 3.08E-2 peak in congestion with an overall value of 1.22E-2 for congestion. These results, for Network Three, are summarised in the following section, and are also presented cross tabulated for each network in the final section of this chapter, along with the results from Networks One and Two.

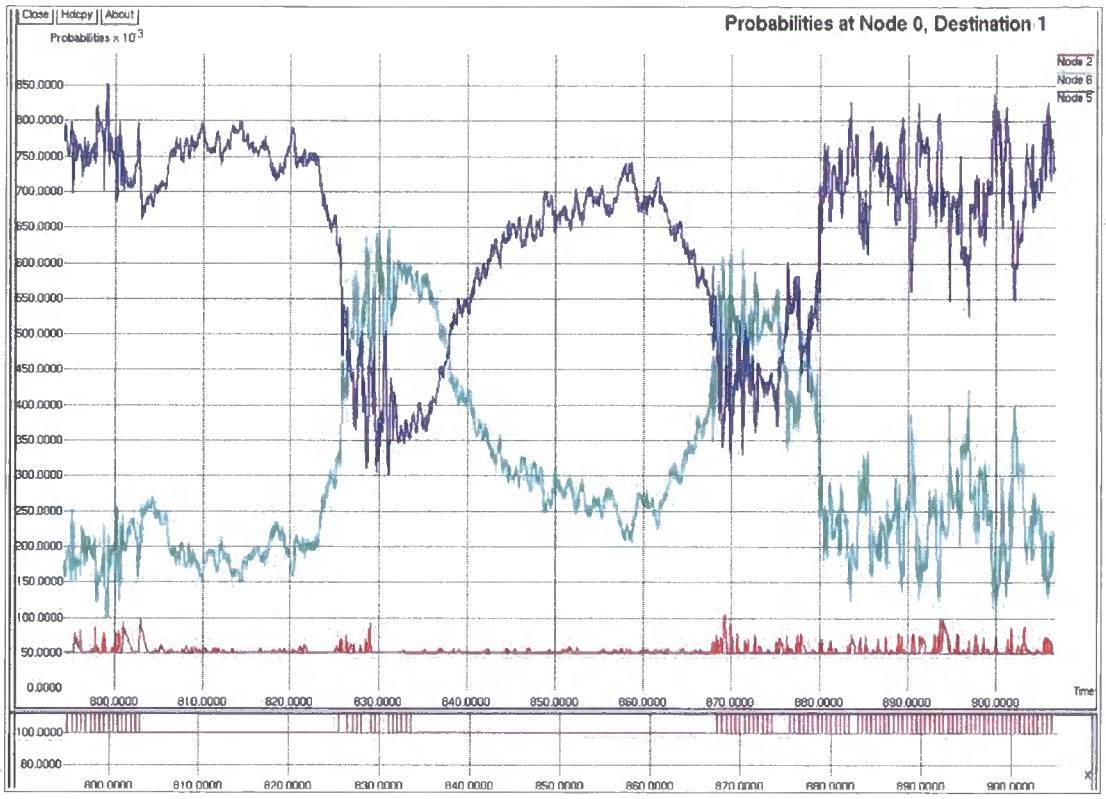


Figure 6.34: Diagram to show Agent reacting to congestion on Network Three

Figure A.14, in the Appendix, shows a period of congestion in more detail, and from this it can be seen that the route change takes only 6.5s from the beginning of the period of congestion. This compares very favourably with the other schemes described - 7.1s and 9s. The difference in level of noise between when the parameter is increased and when it is not is obvious. In this instance the parameter is returned to its original level promptly which will help in terms of system stability.

6.7.6 Network Three Results Summary

Figure 6.35 shows a summary of results for Network Three. As can be seen the Hardcoded and Q-learning systems work appreciably better than the other two, however the hardcoded solution is unrealisable.

Results across the networks are summarised and discussed in the penultimate section of the chapter.

Three	Reaction Time	SD low	Low Time	SD high	High Time	SD congestion
ABR	9.1	9.78E-5	12.0	3.08E-2	26.5	4.03E-2
Hardcoded	7.1	9.78E-5	12.0	8.16E-3	23.5	6.14E-3
Threshold	9.0	2.38E-4	17.5	1.93E-2	22.5	6.41E-3
Q-Learning	6.5	7.05E-4	815.0	3.08E-2	824.0	1.22E-2

Figure 6.35: Summary of results for Network Three

6.8 Validation Tests

A final stage to the experiments already undertaken is to validate that the experimental results are not purely dependant on the random numbers that are generated in that instance. They are generated from a constant random seed for all experiments to ensure repeatability and comparability. In this section a selection of the final tests are repeated with a different random seed to establish independence from such factors - to ensure the results are not merely an artefact.

Figure A.15 shows a graph of the repeated tests, in the same format as previously. It can be seen that the agent system again reacts to disturbances caused by congestion. In this instance the random seeds controlling the ant generation and routing, and the NN initial weights (which have been demonstrated to affect the performance of the system) have all been changed.

Figure A.16 shows another run of the validation tests, and demonstrates the Agent acting consistently to the congestion on Network Three. It can therefore be concluded that, having repeated the experiments with a new set of random numbers, that it is not just a function of the random seed used.

6.9 Discussion of Results

The results presented in this chapter have attempted to set out the case for the use of AI techniques, or more specifically Reinforcement Learning (and then Q-Learning), to improve the performance of telecommunications networks.

By the use of some simple networks, and then a more complex one that represents the generic, sparsely connected network, this has been proven. The first experiments were designed to be concerned with a single decision point, and so the networks represented different generic types of decision - i.e. many routes of equal lengths, and a number of routes with a *definite* shortest one. The system has then been run on the generic network and has been shown to improve the performance.

A comprehensive set of strategies was presented: the ordinary Ant-System, the ideal strategy, a simple threshold mechanism, and the proposed Q-Learning agent. These were used to show firstly that the adaptation of parameters by any sensible mechanism can improve the performance of the Ant-System, and secondly to show that the Q-Learning Agent outperforms that of a simple reactive mechanism - so justifying the use of the overhead and complexity of the learning system. Thirdly, an "ideal" strategy is presented to show the bound on the improvement in performance.

It is important to note that the Agent controls the Ant-System in a strategic manner, it does not interfere with the minutiae of the ants, and the actions of individual ants, but monitors the performance of the system as a whole, and acts accordingly. This architecture is similar to that of Subsumption Architecture used by [Brooks 1986] who used it to control robots in a robust and progressive manner.

The phenomena of noise in the context of the probabilities is the effect of individual ants arriving and the probabilities being updated. If each ant has a large effect on the probabilities, then they will be more "spiky" - or *noisy* in our terms. If each ant has only a small effect on the probabilities, then they will be updated in a much smoother

manner. It is not however, merely a question of aesthetics in terms of wanting a nice smooth graph. The effect of the noise in certain circumstances would be to move the probability of one route above the another. This may or may not be the correct thing to happen. If this occurs *only* because of the presence of the noise, and not due to a longer term trend (i.e. because congestion means that a different route is now better) then this is undesirable. It is also likely to be reversed when an ant arrives on the other route, making that probability larger again. The result of a constant change of best-route would be that, potentially, packets from the same flow would be spread over many routes making reassembly at their destination expensive.

The performance of the ordinary Ant-System provides the benchmark for this set of experiments. Since it is the parameters controlling the Ant-System that are being manipulated, this set of results describes what would happen to the system if no manipulation took place.

At the opposite end of the scale from the ordinary Ant-System is the use of an ideal strategy defined to be the perfect solution to the problem scenario. This strategy, although unrealisable, defines the bound of the improvement the agent would theoretically be able to make.

Thirdly, a simple reactive agent was devised that merely adapted the parameters affecting the Ant-System when a threshold on the SD was exceeded. The SD should increase when there is congestion as the probabilities will change. However it was shown that this was a fragile solution, only ideally suited to the conditions under which the threshold level was set. From this it can be concluded that the level of intelligence provided by the Q-Learning system is required if adaptation of the parameters is to be undertaken - and that the simple reactive agent was not sufficient.

The results from each network are now presented in terms of their scheme, rather than the network. This is shown in Figure 6.37.

ABR	Reaction Time	SD low	Low Time	SD high	High Time	SD congestion
One	-	7.06E-5	18.5	1.17E-3	23.0	5.56E-3
Two	4.3	1.04E-4	17.0	6.80E-3	23.0	2.24E-3
Three	9.1	9.78E-5	12.0	3.08E-2	26.5	4.03E-2

Hardcoded	Reaction Time	SD low	Low Time	SD high	High Time	SD congestion
One	-	5.83E-5	11.0	8.09E-3	29.0	3.03E-3
Two	3.6	8.98E-5	11.5	8.09E-3	22.0	2.62E-3
Three	7.1	9.78E-5	12.0	8.16E-3	23.5	6.14E-3

Threshold	Reaction Time	SD low	Low Time	SD high	High Time	SD congestion
One	1.2	1.39E-4	18.5	6.78E-4	27.0	3.47E-4
Two	4.3	1.34E-4	12.0	6.29E-3	22.5	2.17E-3
Three	9.0	2.38E-4	17.5	1.93E-2	22.5	6.41E-3

Q-Learning	Reaction Time	SD low	Low Time	SD high	High Time	SD congestion
One	-	1.07E-4	18.0	5.32E-3	22.5	2.60E-3
Two	3.9	1.85E-4	416.5	4.26E-3	422.5	1.64E-3
Three	6.5	7.05E-4	815.0	3.08E-2	824.0	1.22E-2

Figure 6.37: Diagram to show results for all networks and all regimes

The techniques described in this thesis are used on a particular routing method (Ant-Based) and a particular implementation of that method. It is intended however that the conclusions of the thesis have a broader scope than the specific application; it is intended that by using the principle of the subsumption style architecture within an agent society context, that the techniques could be applied to other problems of lesser, equal, and eventually greater complexity. This is especially true as the reinforcement learning techniques themselves become more comprehensive and mature. There are a number of learning techniques that could be applicable to the domain of telecommunications, but also in developing these techniques, new methodologies are

developed.

Having produced consistent and repeatable results, using a number of different random seeds, for both the traffic scenarios as well as the initial weights used by the neural network, the system has shown that it is able to learn strategies to compensate for congestion and improve the reliability of the network.

6.10 Chapter Summary

This chapter has described the principles behind Reinforcement Learning, and how these can be put into practice. Then the details behind the implementation, including issues in the design, have been described. Finally, the performance of the system as a whole has been evaluated, specifically in comparison with the ordinary Ant-Based Routing System, both by itself and also when managed by a number of schemes, including an idealised strategy and a purely reactive response.

The approach taken constitutes a novel solution to the control of Telecommunications Networks, and the application of Reinforcement Learning to a real problem.

The remainder of this thesis concentrates on potential developments that could be undertaken, as well as placing the work in a context that was not possible in the introductory chapters.

Chapter 7: Further Work

This penultimate chapter identifies a number of areas where the work described in this thesis could be continued and extended. These areas include direct extensions to the work undertaken, alternative approaches that could have been taken, and other applications to which these techniques could be of benefit.

7.1 Development of Current Work

A direct development of the current work is to remove some of the assumptions required in the work so far. These included using a single NN shared by all the nodes, although to reiterate - this did not imply global knowledge since each node consulted separately. Using separate NNs for each node will mean slower convergence in the learning, but more accurate values in the long run - different nodes with different topologies are likely to have distinctive characteristics, which will be learnt by individual NN rather than the "average" function learnt by the single one described here.

Further extensions are to make the state information more sophisticated - supplying more statistics on which the NN can make more informed decisions. The number of parameters that are manipulated; as already discussed the Ant-Rate could be changed but this must be done carefully as it could add to congestion.

Neural Networks have a strength of identifying relationships between inputs and the reward function -since this is their job.

It is possible to improve the performance of a Q-Learning NN by providing "value-added" data, rather than just the raw data -for example a humans opinion on how the computer is doing. But if this is done, it could be argued that this is restricting the

learning mechanism into a pre-defined rigid framework -the state-space of solutions has been reduced; this may or may not be an advantage. Perhaps it means that the NN then has a bias towards one particular strategy in the game of Backgammon. Although in this case Backgammon is very well understood, it was Tesauro's NeuroGammon that revolutionised the playing of it as it showed that one strategy that had always been ignored was actually the best one. The effect of overly "tuning" the input data to favour one strategy is that the system might then pursue that strategy for longer than it really should.

Additionally, if this information is value-added by some form of processing before the input then why not just leave it up to the NN -since this is its strength.

The exception to this suggested rule of thumb is the reward function. This must encapsulate what the system must see as "good" and what is "bad". In terms of Backgammon, this is simple -winning. In terms of a business this must simply be making a profit; but what about balancing investment to make a profit next year at the expense of this year. This balance is affected by the use of discount rates, which are generally set by the system designer -but is this set too arbitrarily for the possible effects if could have at a later date -such as the situation described.

7.2 Deterministic Ant-Based Routing

The Ant-System described here, and in the associated literature (described in Chapter 3), uses randomly generated numbers to formulate the routes and next-hops. This is analogous to each Ant rolling a dice and consulting a table to make decisions. An alternative to this would be to use the pheromone tables to determine the time-interval between each ant to each destination.

Each destination would have a separate sequence of ants with the sequences overlaid on each other. Over some period of time, say one hundred time-steps, a route with 0.74 of the pheromone will have had 74 ants, while a route with 0.26 of the pheromone has had

26 ants. Rather than waiting complete time periods to decide on the time-intervals, the pheromone levels should be calibrated to particular time intervals. Otherwise, inaccuracies would occur with time-lags being introduced to the system.

Random number generation is fairly computationally heavy, and requires validation to ensure correct distribution of the density functions; this scheme would eliminate such problems, and simplify the implementations.

7.3 Multi-Colony Ant Based Routing

Ant-Based Routing uses many simple agents to explore and quantify a network to allow optimum routing configurations to be implemented. However with the advent of integrated service networks, traffic with differing QoS requirements use the same infrastructure; for example video-traffic and file-transfer - one real-time loss-insensitive but the other time-insensitive, loss-sensitive.

Another example is having both mission-critical data but also Internet "filler" traffic. The second class may be required to be routed under a hard-constraint of not sharing a link with the first, or some softer constraint.

A solution to this problem could be achieved by using a number of Ant-Systems running concurrently over the same network. This could be achieved by using different levels of priority-queues; so that queue with ants of a lower priority in are not serviced until the queues of ants (and traffic) with higher priority have been serviced.

7.4 Ant and Traffic Priority Adaptation

A further adaptation of the Ant-System would be to manipulate the relative priority of the ants and traffic. This would mean that under normal conditions the ants have a lower priority to ensure that no traffic is disposed in preference to an ant. However, under

congestion, when the ants are needed the most they could be prioritised in the name of stabilising the system.

7.5 Connectionless and Connection-Oriented Networks

Although Connection and Connectionless networks have differing applications and characteristics, this is not an impediment to the utilisation of Ant-Based Routing on either. This is demonstrated by the two research efforts into both type of network - [Schoonderwoerd et al. 1996 and 1997] compared with that of [Dorigo et al 1998]. Since the Ant-System works on both it is therefore shown that the Agent-Managed Ant system should be equally valid on either; there is no reason to assume otherwise.

7.6 Routing Decision Agent

There is potential for a further agent to operate on the actual routing decision -note that the purpose of an Ant-System is to measure the performance of different routes, while the Agent described here changes it speed of response. The Ant-System in effect "recommends" a route and this further agent could apply reasoning to the process -to stop the route "flapping" and to provide coordination which would prevent routing-loops; as alluded to in [Kapetanakis 2004], this would represent a *"massive coordination step"*.

Current research in the area of strategic routing decision - where the usual routing mechanisms are over-ruled for the sake of stability, or some other consideration - includes [Labovitz et al 2000, 2001]. Schemes such as these maintain counters of the number of incidents of route-changes, however they then rely on hand-selected thresholds to decide that a route is flapping excessively. This would represent an ideal application for Intelligent Threshold Management, that can adapt to changing circumstances.

Once a route has been "marked" as unsuitable, the count is reduced over time, until another threshold has been reached which marks it suitable again.

7.7 Reinforcement Learning in Complex Domains

In the distant future, there is much potential for RL to be used in many domains - perhaps the most obvious example of usage at the moment is the tightly defined arena of computer games where computer controlled characters learn different reactions to human-controlled characters. Other domains that have potential in the shorter term include those described in [Stockhiem et al. 2003].

Stockhiem et al. investigate the use of Q-learning to optimize the management of supply chains in the domain of manufacturing process improvement. The research comes across similar obstacles to those described here and also use NN to reduce the state space to something manageable.

[Valckanaers et al. 2003] describes Ant-Based techniques to the domain of manufacturing process optimisation, stock control and work flow through the factory.

7.8 Implications for Road Traffic Management

According to Trafficmaster, website [f], congestion on the road costs the UK £20Billion per year, and congestion has increased by 56% in the five years to 2004. It is plain to see why the area of road-traffic management is arguably more important than data-traffic management.

There are a number of differences between data-traffic management and road-traffic management, but there are many analogies. The biggest difference is that in the event of congestion you can discard data-traffic, but not road-traffic.

Although employing numerous vehicles to be ants would not be viable, two options are available; firstly, individual vehicles could be randomly selected as *ant-vehicles* and their progress be followed. This is similar to the TrafficMaster system already in use around the UK.

The use of Ant-Vehicles would merely mean that average speeds could be monitored, the driver of the selected vehicle would introduce the random element required for the system -i.e. In deciding its destination etc.

The novel step would be then to link this monitoring system in with the in-car traffic guidance systems being increasingly available and affordable.

7.9 Chapter Summary

This Chapter has outlined some further areas for investigation in the future. These include adaptations from each of the three stages of work described within the Thesis. Other potential applications that may benefit from similar solutions have been described both in related fields, as well as other areas of Engineering.

The following chapter concludes this thesis, drawing together all the research done, with final observations and remarks.

Chapter 8: Conclusions

This thesis has explored and investigated the use of artificially intelligent techniques to improve the control of telecommunications networks with novel approaches.

The first approach was to build an innovative, highly modular control structure to break down different tasks into more easily manageable tasks than is conventional. This should also benefit the designing of higher layer control schemes, as they will be abstracted away from the concerns of lower levels of the control and configuration scheme. The system was demonstrated to work on a real ATM network.

In the implementation described, a Distance Vector Routing Algorithm was built as a first step, but this could be extended to a Link State Protocol in further developments of the system. Other avenues of potential use have been discussed, including the mapping of Quality of Service classes from one regime to another. The system was designed to autonomously configure the nodes on a network, to improve the stability, reliability, speed of configuration and startup, and reduce the liability to human-error.

Secondly, an Ant-Based routing scheme was built and described; it is broadly based upon other implementations, but also was distinctive in a number of areas. A thorough investigation into the correct parameter settings was undertaken to enable optimal performance of the system. It was found in the process that there was, unsurprisingly, a compromise between the speed of response of the system to disturbances, and the stability of the system during normal operation. This exhibited itself in the levels of noise on the probabilities caused by individual ants arriving and depositing their pheromone to reinforce the route they had just arrived on. However it was shown the the Ant-Based Routing system was able to produced sound routing strategies.

It was therefore considered desirable that the Ant-Based Routing System be improved by some method that allows adaptation of the control system strategy, to maintain stability in steady-state equilibrium, but to increase the speed of response of the system when some disruptive event occurs - such as congestion or failure.

The applicability of Ant-Routing to other domains was discussed in Further Work, particularly to Road Traffic Monitoring, along with adaptations to the Ant-Algorithm itself. The System itself was envisaged to work within a hierarchical structure, just as existing routing schemes do, and this ensures that the system is set in a scalable framework.

In the third tranche of work, a solution was sought to the problem described with the Ant-System. This involves the novel use of an intelligent entity, an agent, to manipulate the speed of response of the Ant-System. Rather than using an empirical human-programmed threshold system that would not be adaptive to changing ambient conditions, the solution developed used machine-learning techniques that are able to adapt to subtle nuances of the environment. The advantages of this approach over a simple-threshold, reactive agent, as well as the ordinary Ant-Based Routing System was demonstrated. A further comparison was made with an Ideal Agent, that had *a priori* knowledge of congestion; this approach, although unrealisable under ordinary

conditions, provided a context for how much of an improvement was provided by the Q-Learning Agent.

The technique of using a Neural-Network to implement Q-Learning, and indeed TD-Learning, means that this solution is applicable to many more problem domains than that of Table-Based Q-Learning (and TD-Learning) since it is not constrained by the dimension of the table itself.

Since the Agent is encapsulated from the Ant-System, working in a Subsumption style architecture, there is no reason why it could not monitor some other system and manipulate the strategy there also, with minimal adaptation. Indeed this work has attempted to monitor and control a system that is a considerable number of magnitudes more complex that has been tackled before.

Results from each section of work, show that these Artificial Intelligence techniques can be successfully used to automate the configuration and maintenance of networks, where human intervention can be untimely, expensive to deploy, and potentially de-stabilising. This is increasingly important in the context of the exponential growth of mobile computing devices (phones, laptops, personal organisers etc.), and the complexity of the configuration - especially given the inevitable resulting fall in the average technical knowledge of the users - of such devices. Place this also in the context of rising levels of malicious use of each new technology, and the need for robust automation is clear.

References

Books and Papers

Appleby S., Steward S., 1994, "Mobile Software Agents for Control in Telecommunications Networks", BT Technology Journal, 12 (2): 104-113 April.

Bernet Y., Blake S., Grossman D., Smith A., 2002, "An Informal Management Model for Diffserv Routers", RFC 3290, IETF, [d].

Bonabeau E., Dorigo M., Theraulaz G., 1999, "Swarm Intelligence: From Natural to Artificial Systems", Oxford University Press, 0-19-513159-2.

Bonabeau E., Henaux F., Guérin S., Snyers D., Kuntz P., Theraulaz G., 1998, "Routing in Telecommunications Networks with Smart Ant-Like Agents", Proc. of Intelligence for Telecommunications Applications, IATA'98. Berlin, Springer-Verlag.

Boyan J.A., Littman M.L., 1994, "Packet Routing in Dynamically changing networks: A Reinforcement Learning Approach", Advance in Neural Information Processing Systems, vol. 6 pp.671-678, Morgan Kaufmann.

Braden R. (Ed.), Zhang L., Berson S., Herzog S., & Jamin S.,1997, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, IETF [d]

Braitenberg V., 1986, "Vehicles: Experiments in Synthetic Psychology", MIT Press, 0-262-52112-1.

Breslau L., Estrin D., Fall K., Floyd S., Heidemann J., Helmy A., Huang P., McCanne S., Varadhan K., Xu Y., Yu H.., 2000, "Advances in Network Simulation", IEEE Computer, 33(5), pp. 59-67, May, [h].

Brooks R.A., 1986, "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, 2(1), March.

Case J., McCloghrie K., Rose M. & Waldbusser S., 1993, "Introduction to Version 2 of the Internet-Standard Network Management Framework", RFC 1441, IETF [d]

Choi SPM, Yeung P.-Y., 1996, "Predictive Q-Routing: A Memory-based Reinforcement Learning Approach to Adaptive Traffic Control", Proc. 8th Conf. on Neural Information Processing Systems, NIPS-8, 945-910, Cambridge, MA, USA: MIT Press

Collis J., Ndumu D., 2000, "The Zeus Agent Building Toolkit: The Role Modelling Guide" Release 1.02, BT plc. [b]

Deering S., Hinde R., 1998, "Internet Protocol Version 6 Specification", RFC 2460, IETF, [d]

Dijkstra E.W., 1959, "A Note on Two Problems in Connection with Graphs", Numerische Mathematic, Vol. 1, pp. 269-271.

Di Caro G., Dorigo, 1997a, "AntNet: A Mobile Agents Approach to Adaptive Routing", Technical Report IRIDIA/97-12, Université Libre de Bruxelles, Belguim.

Di Caro, G., Dorigo, 1997b, "A Study of Distributed Stigmergetic Control for Packet-Switched Communications Networks", Technical Report IRIDIA/97-20, Université Libre de Bruxelles, Belguim.

Di Caro G., Dorigo, 1998, "AntNet: Distributed Stigmergetic Control for Communications Networks", Journal Artificial Intelligence Research, 9:317-365.

Dorigo M., Gambardella L.M., 1997, "Ant Colonies for the Travelling Salesman Problem.", BioSystems, 43 pp73-81.

Droms R., 1993, "Interoperation Between DHCP and BOOTP", RFC 1534, IETF, [d]

Gaïti D., Pujolle G., 1994, "ATM Flow Control Schemes Through a Multi-Agent System"

Gaïti D., Boukhatem N., 1996, "Cooperative Congestion Control Schemes in ATM Networks", IEEE Communications Magazine, November, pp102.

Gaïti D., Pujolle G., 1996b, "Performance Management Issues in ATM Networks and Congestion Control", ACM Transactions on Networking, Vol. 4 No. 2.

Garijo M., Cancer A., Sánchez J.J., 1994, "A Multiagent System for Cooperative Network-Fault Management".

Guérin S., 1997, "Optimisation Multi- Agents en Environment Dynamique: Application au Routage danns les Résaux de Télécommunications", DEA Dissertaion, University of Rennes I, France.

Hall J.L., 1998, "Application of Learning Algorithms to Traffic Management in Integrated Services Networks", PhD Thesis, University of Durham.

Hayzelden A.L.G., 1999, "A Multiple-Agent Approach for Resource Configuration in Communication Networks", PhD Thesis, Queen Mary College, London

Hedrick C.L., 1988, "Routing Information Protocol", RFC 1058, IETF, [d].

Heusse MS, Guérin, Snyers, Kuntz,1998,"Adaptive Agent-Driven Routing and Load Balancing in Communications Networks", Technical Report RR-98001-IASC, ENST Bretagne, Brest, France

Huitema C., 2000, "Routing in the Internet", 2nd Edition, Prentice-Hall, 0-13-022647-5

Jennings N.R., 1999, "Agent Based Computing: Promise and Perils".

Klopf A.H., 1972, "Brain Function and Adaptive Systems – A Hetrostatic Theory", Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford MA.

Kumar S., Miikulainen R., 1997, "Dual Reinforcement Q-Learning: an On-line Adaptive Routing Algorithm", Proc. Artificial Neural Networks in Engineering.

Kumar S., Miikulainen R., 1999, "Confidence Based Q-Routing: an On-line Adaptive Routing Algorithm", Proc. 16th Int. Joint Conference on Artificial Intelligence.

Kapetanakis S., Kudenko D., 2002, "Improving on the Reinforcement Learning of Coordination in Cooperative Multi-Agent Systems", Proc. AAMAS-II, AISB'02, pp 89-94, Imperial College, London, April.

Kapetanakis S., Kudenko D., Strens M., 2004, "Learning to Coordinate using Commitment Sequences in Cooperative Multi-Agent Systems", AISB Journal, 1(5).

Labovitz C., Ahuja A., & Bose A., 2000, Delayed Internet Routing Convergence", Proc. SIGCOMM'00, Stockholm, Sweden, pp175-187.

Labovitz C., Ahuja A., & Bose A., 2001, "Delayed Internet Routing Convergence" IEEE/ACM Transactions on Networking, 9(3):293-306

Legge D.N., Baxendale P.R., 2002, "An Agent-Based Network Management System", AAMAS-II@AISB'02 Conference, Imperial College, London.

Legge D.N., Baxendale P.R., 2003, "An Agent-Managed Ant-Based Network Control System", AAMAS-III@AISB'03 Conference, Aberystwyth.

Legge D.N., Baxendale P.R., 2004, "The Strategic Control of an Ant-Based Routing System using Neural Net Q-Learning Agents", AAMAS-IV@AISB'04 Conference, University of Leeds.

Legge D.N., 2005, "The Strategic Control of an Ant-Based Routing System Using Neural Net Q-Learning Agents", Lecture Notes in Computer Science, Vol. 3394, pp. 147–166.

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

McDysan D.,2000,"QoS and Traffic Management in IP & ATM Networks", McGraw Hill

Moy J., 1998, "OSPF Version 2", RFC 2328, IETF, [d].

Moy J.T., 1998, "OSPF: Anatomy of an Internet Routing Protocol", Addison Wesley, 0-201-63472-4.

Nwana H., Lee L., Jennings N., 1994, "Coordination in Software Agent Systems", Intelligent Systems Research Group, BT.

Peng J., Williams R.J., 1996, "Incremental Multi-Step Q-Learning", Machine Learning, 22:283-290.

Postel J., 1981, "Internet Protocol", RFC 791, IETF [d].

Postel J., 1981, "Internet Control Message Protocol", RFC 792, IETF [d].

Rumelhart D.E., Hinton G.E., Williams R.J., 1986, "Learning Internal Representation by Error Propagation", pp318-362 in [Rumelhart & McClelland 1986]

Rumelhart D.E., McClelland J.L., 1986, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition", MIT Press, 0-26218120-7

Russell S.J. & Norvig P., 1995, "Artificial Intelligence - A Modern Approach", Prentice Hall, 0-13-360124-2.

Samuel A.L., 1959, "Some Studies in Machine Learning Using the Game of Checkers", IBM Journal on Research and Development, 3:211-229.

Schoonderwoerd R., Holland O., Bruten J., Rothkrantz L., 1996, "Ant-Based Load Balancing in Telecommunications Networks", Hewlett-Packard Laboratories Bristol.

Schoonderwoerd R., Holland O., Bruten J., 1997, "Ant-Like Agents for Load Balancing in Telecommunication Networks", Agents'97.

Shaikh A., Rexford J., Shin K.G., 2001, "Evaluating the Impact of Stale Link State in Quality-of-Service Routing", IEEE/ACM Transactions on Networking, 9(2):162-175, April.

Shenker S., Partridge C., Guerin R., 1997, "Specification of Guaranteed Quality of Service", RFC 2212, IETF, [d].

Simon H.A., 1957, "Models of Man", New York, Wiley.

Sommerville, I. 2004, Software Engineering, Addison Wesley (7th. Edition), 0-321-21026-3

Stallings W., 1993, "SNMP, SNMPv2 and CMIP", Addison Wesley, 0-201-63331-0

Stevens W.R., 1994, "TCP/IP Illustrated Volume 1: The Protocols" Addison Wesley.

Stockheim T., Schwind M, Koening W., 2003, "A Reinforcement Learning Approach for Supply Chain Management", Proc. EUMAS'03, St. Catherine's College, Oxford University.

Sutton R.S., Barto A.G., 1998, "Reinforcement Learning; An Introduction" MIT Press, 0-262-19398-1

Tesauro G.J., 1992, "Practical Issues in Temporal Difference Learning", Machine Learning, 8:257-277.

Tesauro G.J., 1994, "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master Level Play", Neural Computation, 6(2):215-219.

Tesauro G.J., 1995, "Temporal Difference Learning and TD-Gammon", Communications of the ACM, 38:58-68.

Tesauro G.J., 1999, "Pricing in Agent Economies Using Neural Networks and Multi-Agent Q-Learning", Proc. Workshop ABS-3: Learning About, From and With other Agents (IJCAI'99), Stokholm.

Tesauro G.J., 2002, "Programming Backgammon Using Self-Teaching Neural Nets", Artificial Intelligence, 134 pp181-199, Elsevier Science

Turing A.M., 1950, "Computing Machinery and Intelligence", Mind, 49, pp:433-460.

Valckenaers P., Van Brussel H, Hadeli, Bochmann O., Germain B.S., Zamfirescu C., 2003, "On the Design of Emergent Systems: an Investigation of Integration and Interoperability Issues", Engineering Applications of Artificial Intelligence, 16, pp:377-393.

Vilà P., Marzo J.L., Fabregat R., 1999, "A Multi-agent Approach to Dynamic Virtual Path Management in ATM Networks".

Vilà P., Marzo J.L., Bueno A., Massaguer D., 2002, "Network Resource Management Investigation using a Distributed Simulator".

Vittori K., Araujo F.R., 2001, "Agent-Oriented Routing in Telecommunications Networks", IEICE Transactions on Communications, E84-B, 11 Nov.

Waldbusser S., 2000, "Remote Network Monitoring Management Information Base", RFC 2819, IETF [d]

Walrand J., Varaiya P., 1999, "High Performance Communication Networks", Second Edition, Morgan Kaufman, 1-55860-654-8

Watkins, C.J.C.H., 1989, "Learning from Delayed Rewards", PhD Thesis, Cambridge University.

Watkins C.J.C.H., Dyan P., 1992, "Q-Learning", Machine Learning, 8:279-292

White, Paguwek, Oppacher, 1998, "Connection Management Using Adaptive Mobile Agent", Proc. Int. Conf. on Parallel Distributed Process Techniques and Applications (PDPTA'98), 802-809, CSREA Press,

Wimer W., 1993, "Clarifications and Extensions for the Bootstrap Protocol", RFC 1542, IETF [d].

Wooldridge M., 1997, "Agent-Based Software Engineering" IEE Proc. Software Engineering,

Zurada J.M., 1992, "Introduction to Artificial Neural Systems", PWS Publishing, 0-314-93391-3

Websites

[a] Centre For Telecommunication Networks, Durham University
www.durham.ac.uk/telecoms.networks/main.html
[b] Zeus Agent Toolkit Project Homepage
www.btexect.com/projects/agent.html
[c] JOONE
www.joone.org
[d] Internet Engineering Task Force
www.ietf.org
[e] ns – The Network Simulator, Version 2
www.isi.edu/nsnam/ns/
[f] Trafficmaster Plc.
www.trafficmaster.co.uk

IETF RFCs

RFC No.	Title, Author, Date
791	"Internet Protocol", Postel J., 1981.
792	"Internet Control Message Protocol", Postel J., September. 1981.
1058	"Routing Information Protocol", Hedrick C.L., June 1988.
1441	"Introduction to Version 2 of the Internet-Standard Network Management Framework", Case J., McCloghrie K., Rose M. & Waldbusser S.
1534	"Interoperation Between DHCP and BOOTP", Droms R., October 1993
1542	"Clarifications and Extensions for the Bootstrap Protocol", Wimer W., October 1993.

RFC No.	Title, Author, Date
2205	"Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification", Braden R. (Ed.), Zhang L., Berson S., Herzog S., Jamin S., 1997
2212	"Specification of Guaranteed Quality of Service", Shenker S., Partridge C., Guerin R., September 1997
2328	"OSPF Version 2" Moy J., April 1998.
2453	"Routing Information Protocol, Version 2", Malkin G., 1998.
2460	"Internet Protocol Version 6 Specification", Deering S., Hinde R., 1998
2819	"Remote Network Monitoring Management Information Base", Waldbusser S., May 2000
3290	"An Informal Management Model for Diffserv Routers", Bernet Y., Blake S., Grossman D., Smith A., May 2002,

Appendix A: Graphs of Results

A.1 Simple Networks

A.1.1 Ant-Based Routing

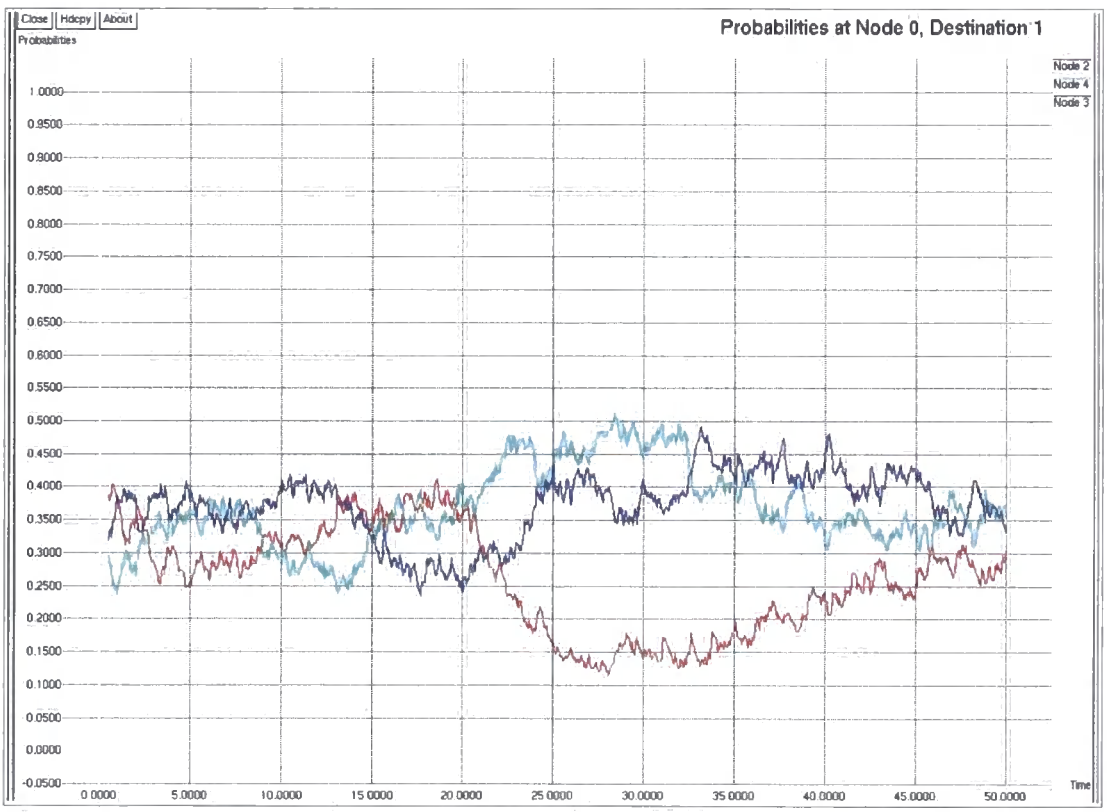


Figure A.1: Graph of probabilities for Ant-Based Routing on Network One

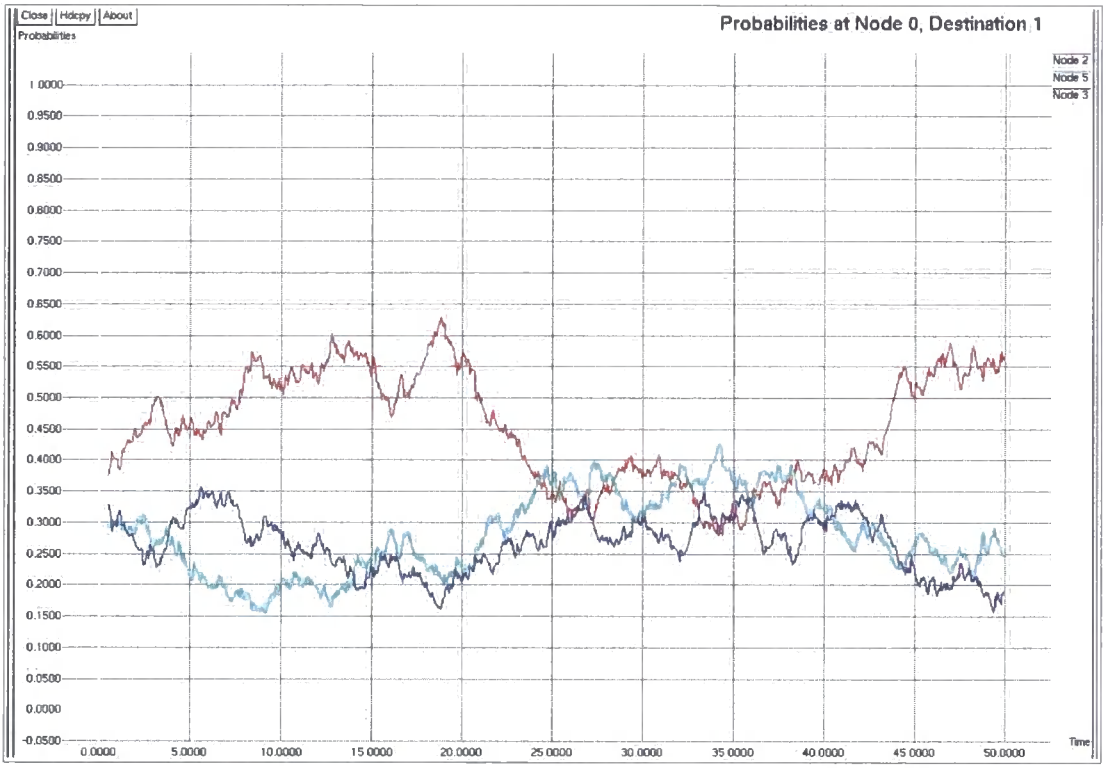


Figure A.2: Graph of probabilities for Ant-Based Routing on Network Two

A.1.2 Ideal Agent Solution

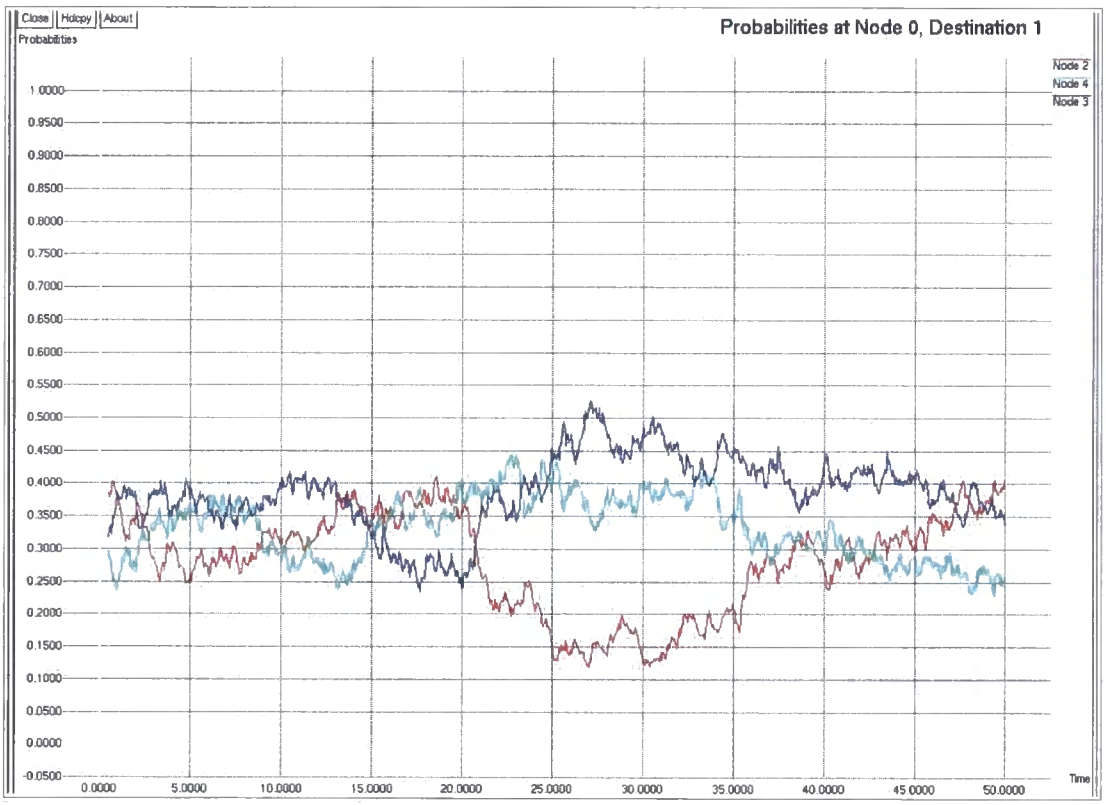


Figure A.3: Graph of probabilities for Network One with Ideal Agent

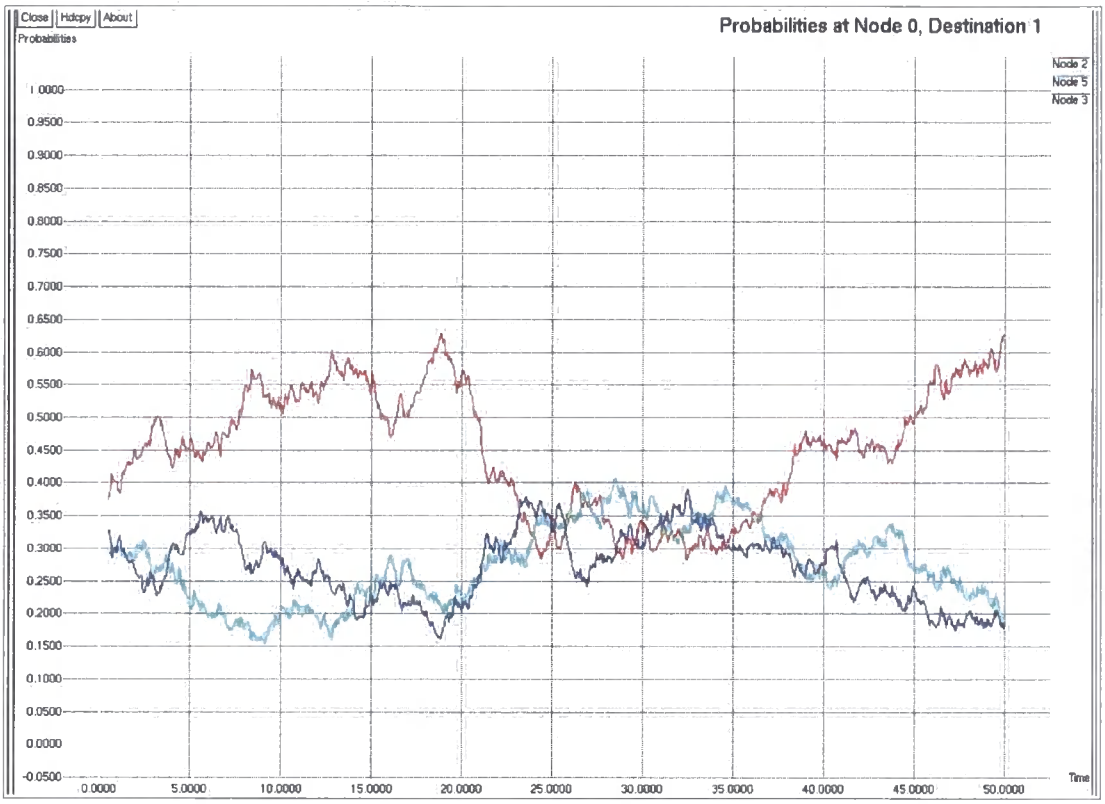


Figure A.4: Graph of probabilities on Network Two with Ideal Agent

A.1.3 Simple Reactive Agent Managed Ants

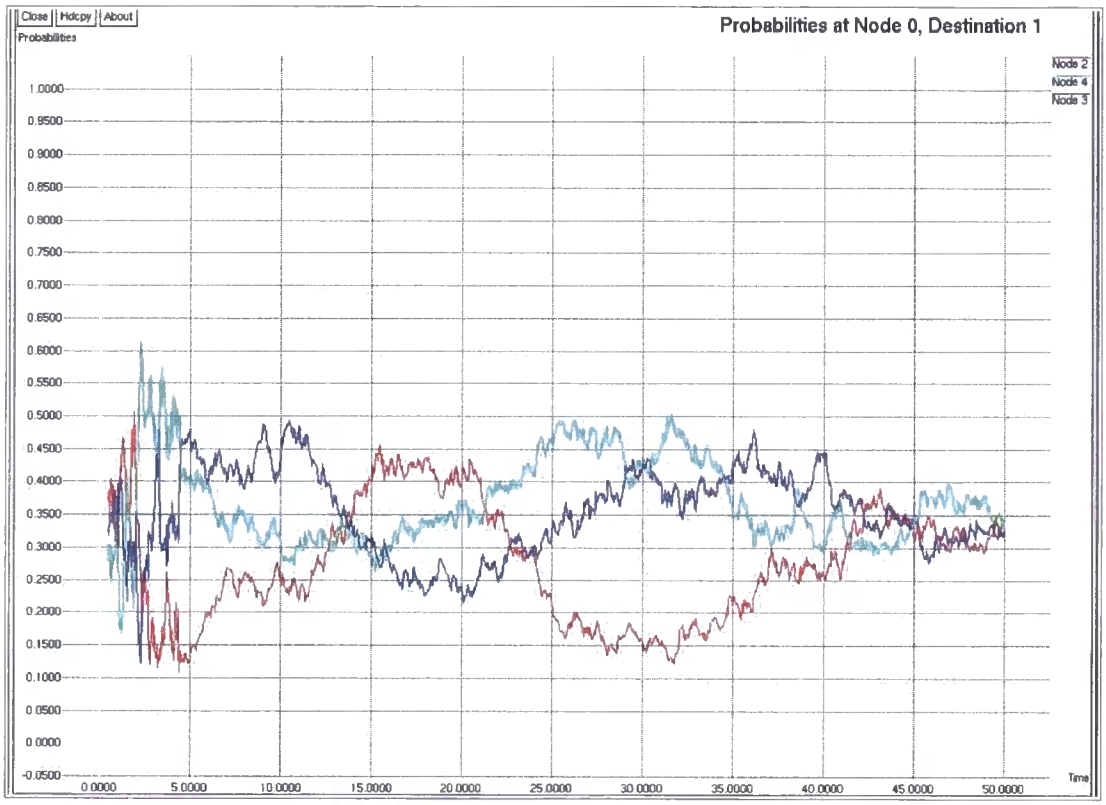


Figure A.5: Graph of probabilities for Network One with Reactive Agent

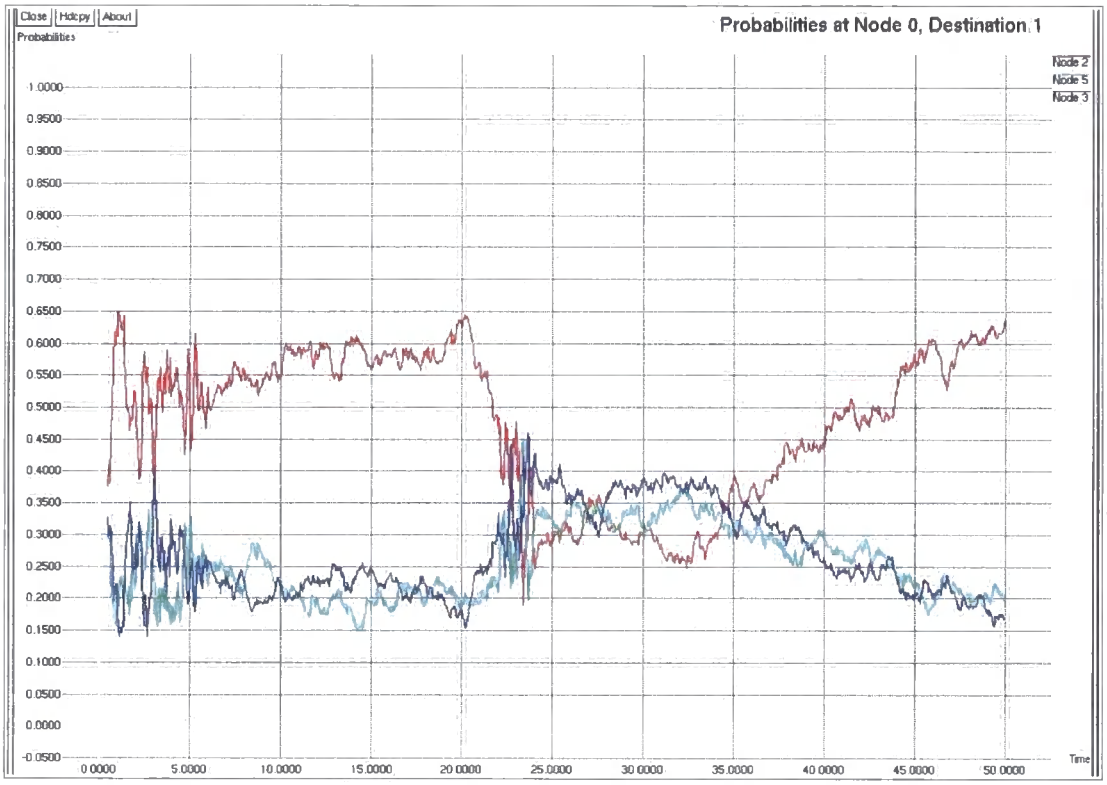


Figure A.6: Graph of probabilities on Network Two with Reactive Agent

A.1.4 Q-Learning Agent Managed Ant Solution

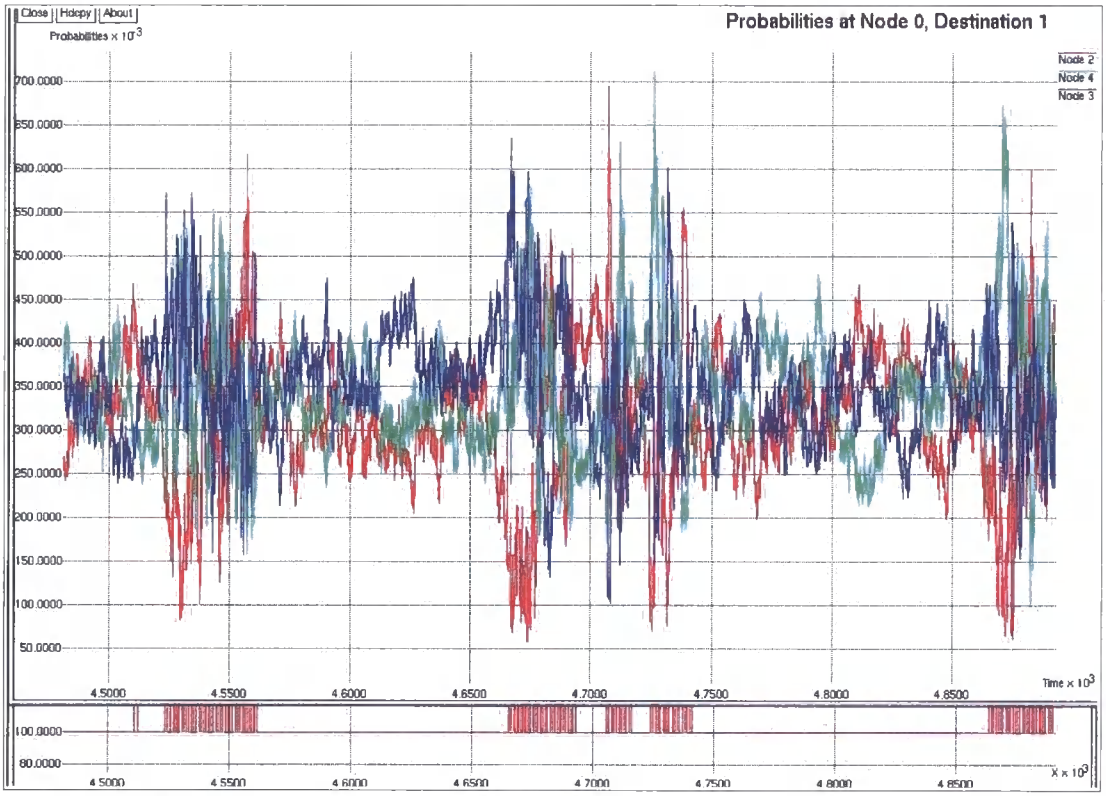


Figure A.7: Graphs to show the Agent reacting to congestion on Network One

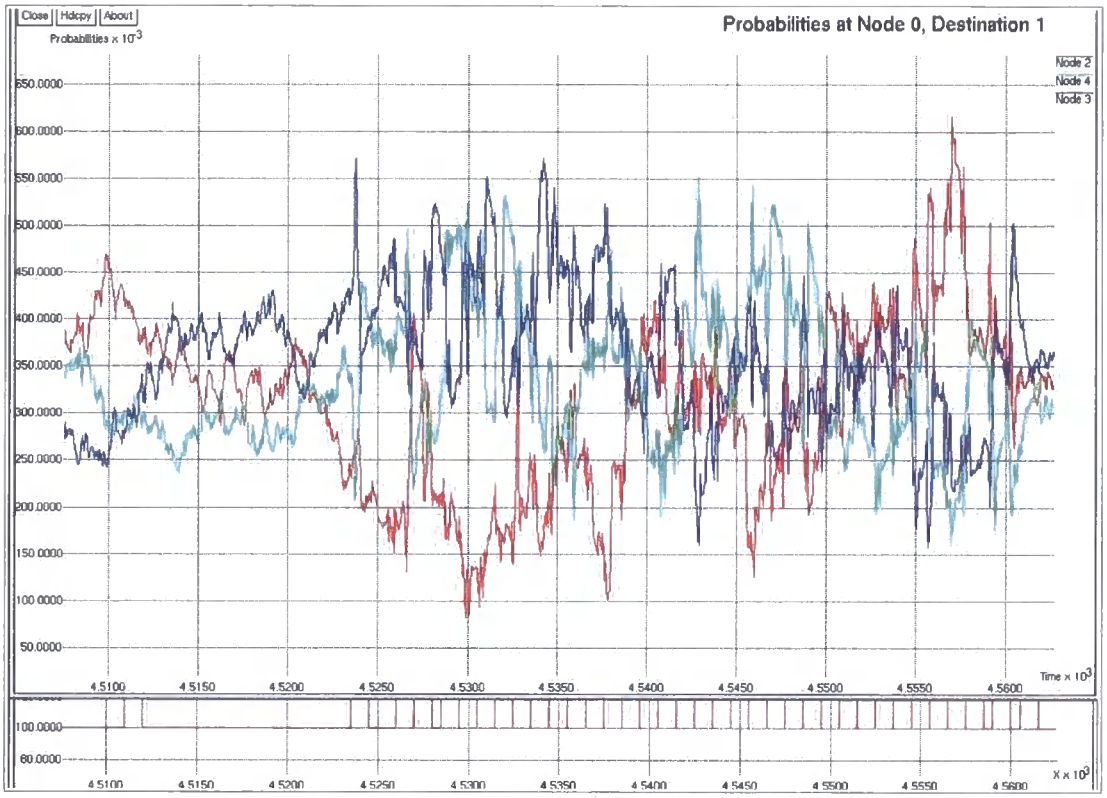


Figure A.8: Graphs to show the Agent on Network One in Detail

A.1.5 Agent Managed Ants Results

Figure A.9: Graphs to show the Agent reacting to congestion on Network Two

A.2 Generalised Network

A.2.1 Ant-Based Routing System

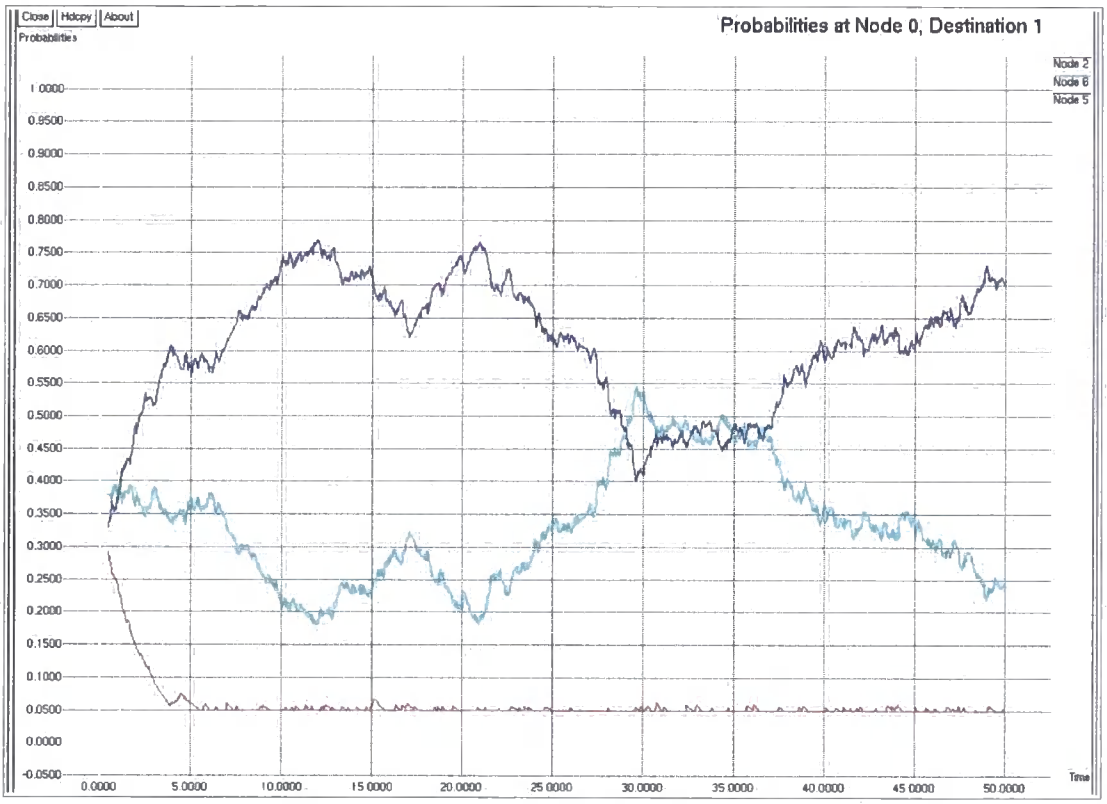


Figure A.10: Graph of Probabilities for Ant-Based Routing on Network Three

A.2.2 Ideal Agent Solution

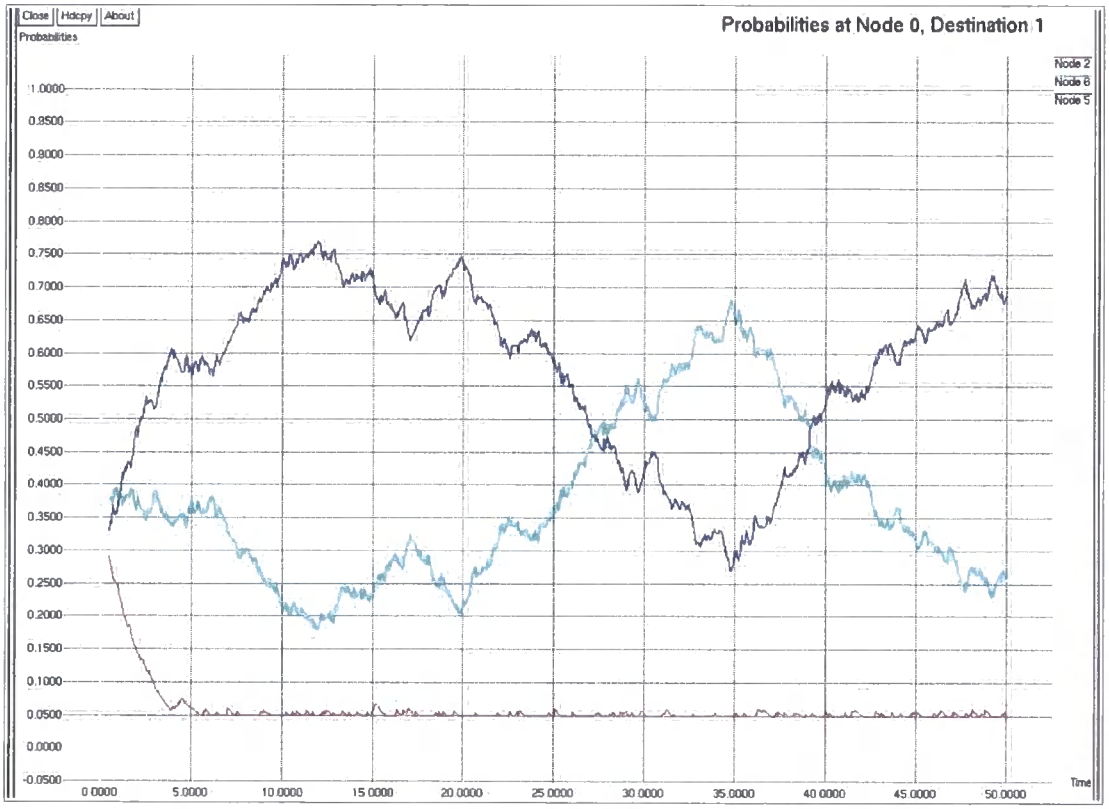


Figure A.11: Probabilities for Ideal-Agent Scheme on Network Three

A.2.3 Simple Reactive Agent System

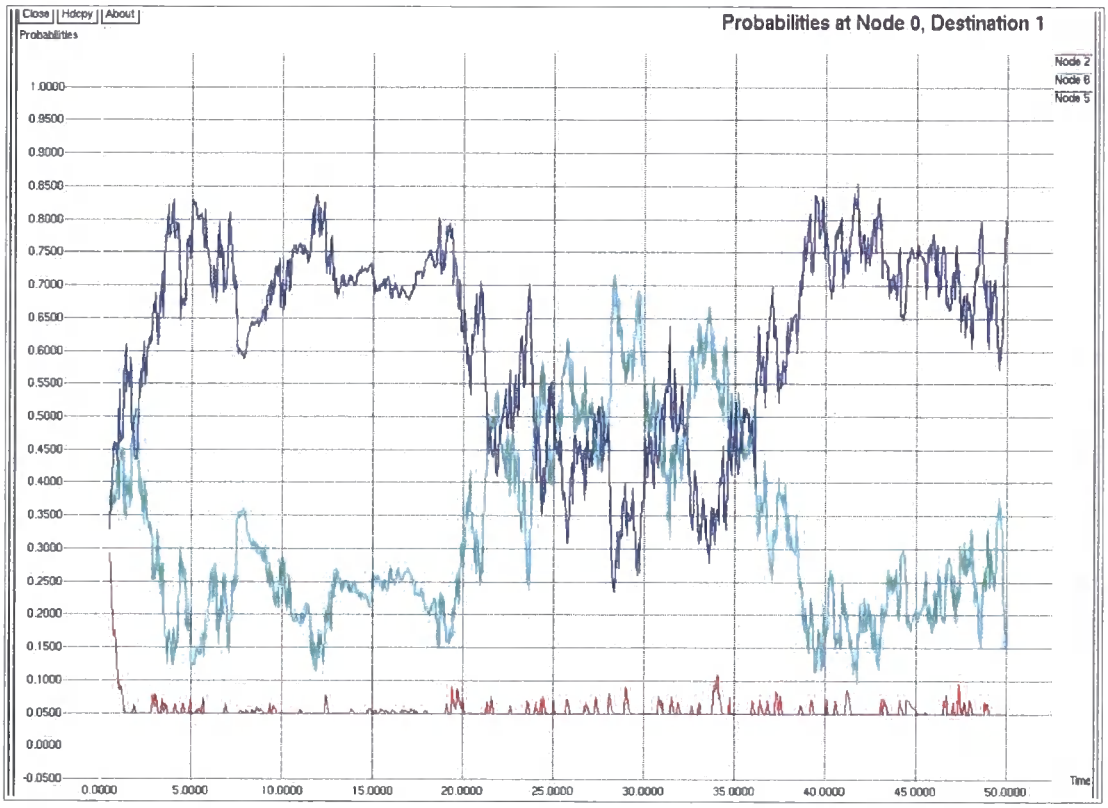


Figure A.12: Probabilities for Reactive Agent on Network Three

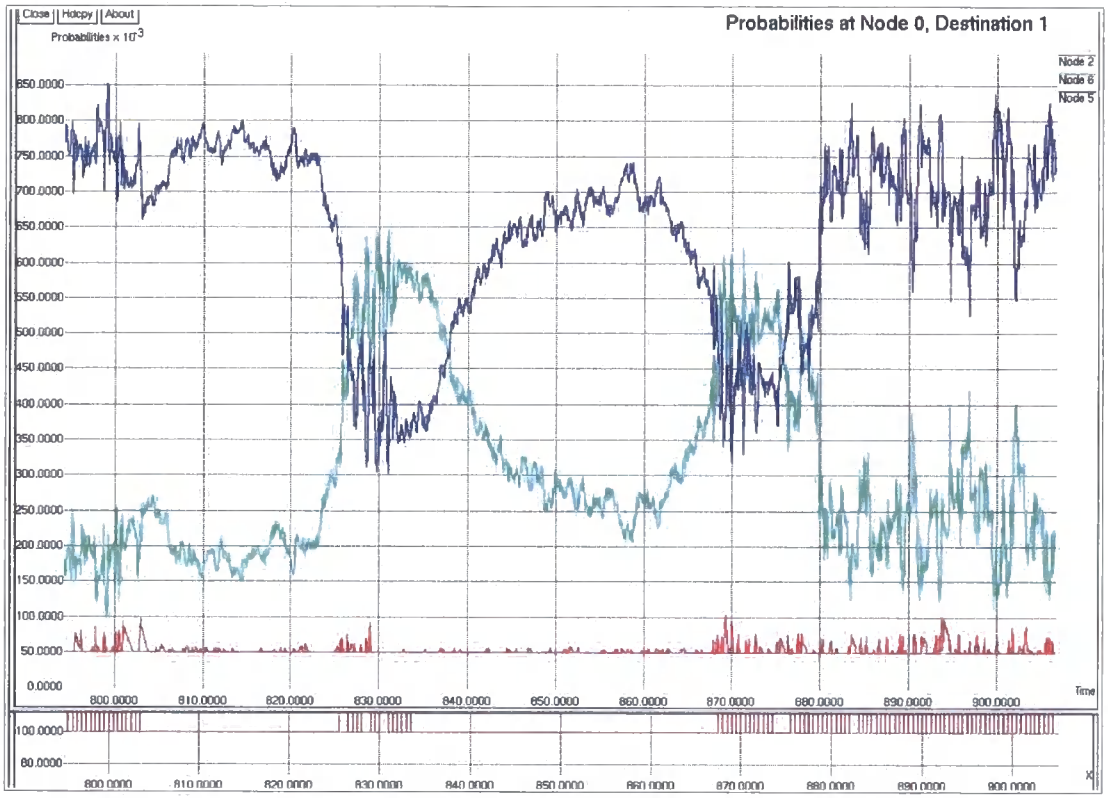


Figure A.13: Diagram to show Agent reacting to congestion on Network Three

It can be seen from Figure A.13 that the parameter is manipulated (shown in the bottom graph) when a drop in the highest probability is encountered. This manipulation means that the route change will proceed faster than if unaided. This graph also shows that the system works consistently across time. There are two periods of congestion covered in this graphic, with the probabilities recovering in between time. Although the the parameter is not returned immediately after the second period of congestion, it does at the end of the graphic.

Figure A.14: Diagram to show Agent reacting to congestion on Network Three

Figure A.14. shows a period of congestion in more detail, and from this it can be seen that the route change takes only 6.5s from the beginning of the period of congestion. This compares very favourably with the other schemes described – 7.1s and 9s. The difference in level of noise between when the parameter is increased and when it is not is obvious. In this instance the parameter is returned to its original level promptly which will help in terms of system stability.

A.3 Validation Tests

The final stage to this test is to validate that the experimental results are not purely dependant on the random numbers that are generated in that instance. They are generated from a constant random seed for all experiments to ensure repeatability and comparability. In this section a selection of the final tests are repeated with a different random seed to establish independence from such factors – to ensure the results are not merely an artefact.

Figure A.15: Diagram to show validation test of Agent on Network Three

Although the systems seems to pre-empt the congestion, this is explained by the significant change in probabilities just before this reaction. It can then be seen that the system reacts to the congestion itself. Noticeably the system prefers the more stable parameters both before and after the period of congestion (accepting the incident just mentioned).

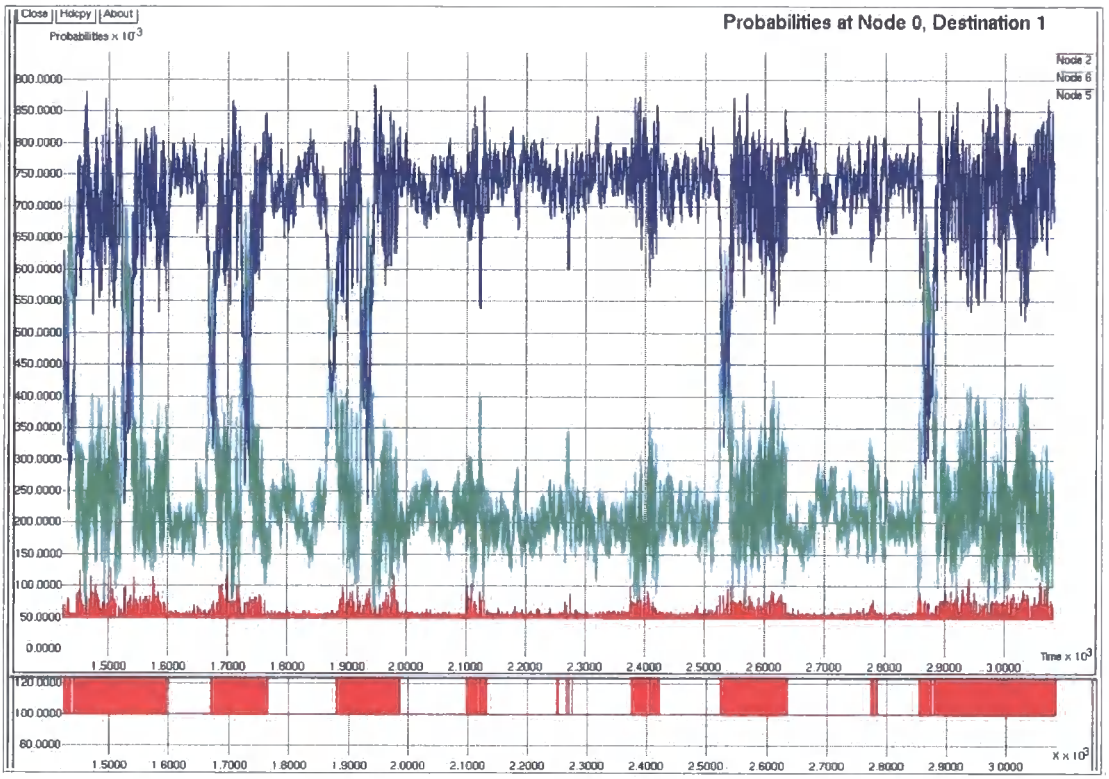


Figure A.16: Diagram to show validation test of Agent on Network Three

Appendix B: Author's Publications

The following papers describing work in this thesis have been published during the time of study:

Legge D.N., Baxendale P.R., 2002, "*An Agent-Based Network Management System*", Proc. Second Autonomous Agents and Multi-Agent Systems Symposia at the Society for Artificial Intelligence and Simulation of Behaviour Conference, (AAMAS-II@AISB'02), Imperial College, London, 4th and 5th April.

Legge D.N., Baxendale P.R., 2003 "*An Agent-Managed Ant-Based Network Control System*", Proc. Third Autonomous Agents and Multi-Agent Systems Symposia at the Society for Artificial Intelligence and Simulation of Behaviour Conference, (AAMAS-III@AISB'03), Aberystwyth, 10th and 11th April.

Legge D.N., Baxendale P.R., 2004, "*The Strategic Control of an Ant-Based Routing System using Neural Net Q-Learning Agents*", Proc. Fourth Autonomous Agents and Multi-Agent Systems Symposia at the Society for Artificial Intelligence and Simulation of Behaviour Conference, (AAMAS-IV@AISB'04), University of Leeds, 29th and 30th March.

Legge D.N., 2005, "*The Strategic Control of an Ant-Based Routing System using Neural Net Q-Learning Agents*", Adaptive Agents and Multi-Agent Systems II, Lecture Notes in Artificial Intelligence 3394, Springer Verlag, Daniel Kudenko, Dimitar Kazakov, Eduardo Alonso (Eds), pp. 147-166

Appendix C: AAMAS-IV Paper

Presented at the AAMAS-IV Symposia at AISB'04,
University of Leeds,
March 29th and 30th 2004

The Strategic Control of an Ant-Based Routing System using Neural Net Q-Learning Agents

David Legge; Peter Baxendale
Centre for Telecommunication Networks,
School of Engineering,
University of Durham
d.n.legge@durham.ac.uk; peter.baxendale@durham.ac.uk

Abstract

Agents have been employed to improve the performance of an Ant-Based Routing System on a communications network. The Agents use Neural Net based Q-Learning approach to adapt their strategy according to conditions and learn autonomously. They are able to manipulate parameters that affect the behaviour of the Ant-System. The Ant-System is able to find the optimum routing configuration with static traffic conditions. However, under fast-changing dynamic conditions, such as congestion, the system is slow to react; due to the inertia built up by the best routes. The Agents reduce this drag by changing the speed of response of the Ant-System. For best results, the Agents must cooperate by forming an implicit society across the network.

1 Introduction

Ant-Based Routing has some attractive properties for communication networks due to the distributed nature of the algorithm. Schoonderwoerd [1996] and Dorigo [1997] described the first systems, which tackle slightly different problems (telephone networks and datagram networks). There have been numerous improvements described, but all have concentrated on the algorithms controlling the behaviour of the ants.

The drawback to Ant-Systems is that they are purely reactive in nature and the probabilistic advantage built up by the best route in good conditions becomes a disadvantage if that route becomes congested.

By using ideas from the area of Subsumption Architecture [Brooks 1986] – i.e. blending purely reactive agents with strategic proactive agents to create a more balanced society – this problem could be overcome.

The Ant-Colony on each node can be viewed as an agent - the ants being only messages, rather than each ant being viewed as an agent, which is an alternative model. It is then a logical step to place a second agent on a node to perform the more strategic role.

The Agent (as distinct from the Ant-Colony), can then monitor the operation of the Ant-Colony and adjust the parameters that affect the speed of response of the Ant-System. This should be done in a way that does not increase the level of Ant-traffic, since this will add to the congestion being tackled.

The Agents across the network must coordinate to some degree, since the affect of only one taking any action would be minimal. This would ideally be done without explicit coordination since the messages could also add to congestion; thus a parallel is drawn to Blind Game-Playing Agents studied in Reinforcement Learning, including [Kapetanakis et al. 2003].

2 Aim

The aim of this research is to show that an intelligent agent can be used to modify the parameters associated with the Ant-system to make both the convergence from start-up, and the response to changing conditions within the network, quicker, whilst retaining the stability in steady-state.

A society of Agents is required however, since the parameters are changed on a per-node basis for distributed control - an agent on each node is in control of the parameters used on that node. At this stage, the agents are acting independently from each other. Conversely, both the ant system and the agents use stigmergy – where the environment is the media for an implicit form of communication.

An agent could observe how other agents are acting by checking the parameters carried by ants originating from other nodes.

3 Current Work

A significant amount of research has investigated the self-organisational properties of Ant-Based systems within telecommunications – for both load-balancing and routing purposes.

Dorigo [1997], and many others, use two

types of ants, one to explore the network, and a second type to propagate the information back across the network.

Other techniques include Vittori [2001], in which the probabilities are interpreted as Q-values, and the Q-Learning reinforcement equation is used to update them.

However, both these papers use single-valued parameters, such as rate of production of ants, rate of deposition of pheromones etc., that are empirically obtained. These are not changed at runtime, or indeed off-line.

The Vittori paper utilises Q-learning to update the routing tables and found performance gains in doing so. In this paper, it is intended to use Q-learning to modify the parameters used by the ants.

[Kapetanakis et al 2002, 2004] discusses agents participating in games with other agents, where the reward gained at the end of the round is a function of the decisions made by all the agents. e.g. Agent 1 decides on action A, while Agent 2 opts for action B. The games are taken further by the addition of a probabilistic affect, so that the the reward is f (A, B, P).

Kapetanakis' agent's decisions are made without conferring with other agent(s), indeed the agent is not aware of the presence of other agents. The agents use Q-learning, and are able to find successful joint strategies. This is analogous to the situation described in this paper.

Other research effort has concentrated on improving and fine-tuning the rules that govern the behaviour of the ants. This paper attempts to branch away from these efforts and strategically manipulate the Ant-System in a similar way to Subsumption Architecture.

4 Ant-System

This section describes the implementation of the Ant-System used. It is broadly based on the system discussed in Schoonderwoerd [1996].

The basis for the ant system is many simple-acting entities displaying complex behaviour on the macro-scale. To this end the ants are small packets of data that are kept as simple as possible. Carried within the ant are: Source, Destination, Amount of Pheromone and Deposition rate. Rather than carrying code with every ant, the processing is performed by the node.

The Rate of Ant Production is a self explanatory parameter, and at generation the ant is given a uniformly-random destination, chosen from all the known nodes on the network. It is also branded with an Initial Amount of Pheromone and a Deposition Rate (which describe its longevity and its potency). Once generated, and at each node along its journey it selects its next-hop by the generation of a uniformly random number and the consultation of the local probability table associated with the ants final destination.

On arrival at a node, and after ensuring the ant has not returned to its source, the ant deposits some pheromone. The amount of pheromone left by the ant is a simple percentage (defined by the Deposition Rate) of the current amount the ant has. If the ant has travelled in a loop, it is discarded.

Before this is added to the probability table (for pheromone table) it is multiplied by the (relative) available size of the traffic queue for the link just used by the ant.

The resultant amount of pheromone is added to the local table regarding the ant's source, i.e. The ant reinforces the return route, not the route it is currently taking. The tables are then normalised to sum to unity; this has the affect of rewarding the route just used, but also punishing all the routes not just used; a Minimum Probability is applied to ensure all routes are explored periodically. If the ant is at its destination it is then discarded.

The output of the ant-colony is the best next-hop for each known node. This is the highest probability link, except for when an extra threshold is applied – i.e. a link must exceed the probability of the current best route by some value for the recommendation to change.

The affect of the ant parameters on the probabilities have been investigated with and without traffic. It has been shown that it is possible to have a very quick response, but with a relatively noisy steady-state condition, or a smoother steady-state but a slower response. The Rate of Production also has an affect on the response.

An assumption is made that each and every link is bi-directional; the discussion of the validity of this assumption is beyond the scope of this paper. In Vittori [2001] routes are reinforced in both forward and backwards directions; although this may aid speed of convergence, the forward reinforcement has no logical basis, and so the feedback may be incorrect, leading to problems with incorrect routes.

The Ant-System is a separate entity, is reactive in nature and can operate with or without the Agent. The Agent provides a supervisory and proactive role and is described next.

5 The Agent

An agent sits on each node and monitors the ant colony on that node. The agent receives reports from the colony, both regular ones and ones triggered by any route changes that occur.

The Agent is informed of all the parameters associated with the ants (Rate of Production, and those carried by the ant), as well as the threshold value for route change.

The agent is also told the current probability of the preferred route, as well as its mean, standard deviation and gradient. Obviously, the windowing function for calculating the mean etc. is an important property, and this has been tuned to highlight important features.

The success or failure of the agent can be de-

termined by a comparison between the performance of the Ant-system by itself, and the Ant-system when supervised by the agent.

The type of learning strategy employed by the agent is discussed in the next section.

6 Agent Learning Strategies

The agent is being asked to react to different states, and should be capable of learning the correct action from any state. The environment is also non-deterministic, so simple Temporal Difference Learning (TD-Learning) would not be suitable; therefore the preferred strategy would be Q-Learning - where an agent learns state-action values.

Although Q-Learning was originally based on tables of state-action pairs, this technique becomes unwieldy when there are more than a handful of states and/or actions. Other problems occur when the system is continuous and must be approximated to a specific state before an action is chosen.

This process of function approximation and the subsequent choice of action has been performed in a single step by Tesauro [2002] and applied to the game of Backgammon with remarkable success.

In the context of Backgammon, TD-Learning is sufficient, since the environment is predictable (the "after-position" of a move is deterministic). The extension of this approach to Q-Learning is a logical next step, and has been investigated by Tesauro [1999] himself.

Tesauro's approach uses an artificial Neural Network, NN, to approximate the Q-value for each action presented to the network for a particular state. Standard NNs are trained to perform known functions - e.g. the Exclusive-Or Function – where the correct inputs and outputs are defined. The error across the outputs can be calculated easily by taking the difference between the desired results and the actual results. The NN weights can then be adjusted accordingly.

Of course, with NN-based Q-Learning the correct outputs are the unknown; but if the Q-Learning update rule is applied instead of the simple error, then the affect is directly equivalent to incrementing table-entries.

The NN-Based Q-Learning system can therefore be trained in the same way as a table-based system. Results from [Tesauro 1999] show that although the Q-values are shown to be less accurate than the table-based equivalent, the actions selected tend to be sound.

7 Implementation

The System has been implemented. The greatest challenge is to devise a Reward Function that adequately encourages desired outcomes whilst punishing undesirable results.

In the current configuration the Agent is able to adjust the Initial Age of the ants – if the Rate of Production were to be adapted there is a danger that congestion would be caused by the control system attempting to reduce it.

The Agent therefore (usually) has three actions available to it – increase or decrease the Initial Age or do nothing. The parameter is bounded as a precautionary "sanity-check", in which case an action that would break this constraint is not presented as an option.

The representation of the actions to the NN is of note.

This can either be achieved by using a single input, and using arbitrary values to represent each value (e.g. -1 for decrease, 0 for stay the same, and +1 for increase) or, using three different inputs with a "1" input to show selection (0 otherwise).

The NN must have at least a single hidden layer to be able to approximate non-linear functions. Initially for speed of convergence and simplicity, only the single extra layer will be used. The output layer is a single node – the Q-value for the action considered – whilst the input layer is composed of the data inputs and the three action inputs. The hidden layer is composed of three nodes to match the action nodes of the input layer.

In order to achieve the objectives, the system must be allowed to train for a sufficiently long time; it is important to note that there is no distinction between training and runtime – the system will be constantly learning so that it is able to adapt to conditions. There could be a danger of "over-training"; however, given the noisy nature of the environment this is not anticipated to be a problem.

Different reward functions have been tested to find the most effective. Ideally, the reward function would be composed of a smooth function of one of the (non-action) inputs – this negates the requirement to introduce an arbitrary threshold that may be suitable for some situations but not for others. This will allow the agent to adapt to the maximum number of conditions; it must be noted however, the the cost of generalising solutions is often prohibitive – in this case in terms of research time.

The two main statistical descriptions of the system that could be used as the reward function are the *standard deviation of the highest ant-probability* (referred to as *standard deviation* from now on) and the *magnitude of the change in the maximum probability*. These will be maximal when the conditions on the network are changing, so can be used as the detection mechanism.

8 Experimentation

The aim of the experimentation is to show that there is an improvement in performance when the parameters governing the ants are changed dynamically at run-time - thus taking

account of changing conditions on the network.

Because of the inherent complexity involved in testing a network, two relatively simple simulations are used. Firstly, a network where there is a number of equally long routes between the two nodes of interest and one of them becomes congested; while the second is where there is an obvious shortest route which then becomes congested (with at least one other alternative route).

Of interest in the test, is the reaction of the ant-system (both managed and un-managed) to congestion; the speed of response is of importance, as is the amount of data-traffic lost (the ants, by their nature, are expendable as information is gained by them not arriving). A further concern is that the routing should not constantly change between two or more routes.

For both scenarios, the Managed Ants performance must be compared with that of the Unmanaged Ants. As an intermediate step, a predetermined strategy, with *a priori* knowledge of the traffic conditions, is implemented that will also improve the performance. This will give a further benchmark with which to contrast the performance of the full system.

8.1 The Simulator

The simulator being used is *ns* (Network Simulator) [Breslau 2000]. It is designed to model TCP/IP networks. These networks are connectionless; that is to say packets of data are transmitted without setting up a definite route first. This means that at each hop, packets are routed according to the instantaneous best route. Packets from the same flow do not necessarily follow the same path.

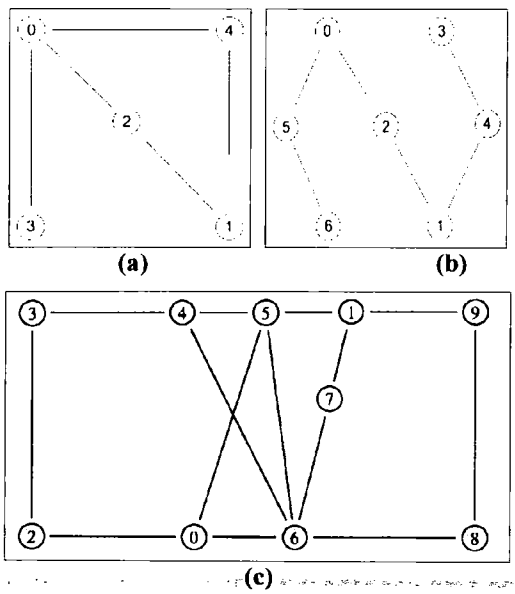


Figure 1: Diagrams of Networks One, Two & Three

Consideration of Circuit-Switched and Connection-Oriented networks are left to later discussion.

8.2 Network One

The network to be tested is shown in Figure 1 (a). It is a simple five node network. The important feature is that there are three equally-long routes between Node 0 and Node 1, one of which, as described above, will at times be congested.

8.3 Network Two

This scenario exists when there is a distinct shortest-path between two nodes as well as longer alternative routes, not sharing any links. The topology is shown in Figure 1(b).

Where the shortest route becomes congested, the ants must react as quickly as possible to the congestion. They must overcome a large amount of inertia represented by the high probability built-up by the shortest route - this inertia is greater than that of the previous scenario.

8.3 Network Three

The third network under test, Figure 1(c), is a more complex one – and is more realistic. It is a sparsely connected network, with a number of cross-links. The route between Nodes 0 and 1 still has a number of alternatives.

8.4 The Traffic Scenario

Both network scenarios will be required to transmit some user-data, which we shall describe as real-time, loss-sensitive data. This shall be transported from Node 0 to Node 1, in all network topologies. The traffic shall be offered to the network intermittently over the whole simulation, causing congestion numerous times.

Problems would be caused by the network dropping the user-data, excessive delay or excessive jitter - variation in the inter-arrival time of packets, often caused by switching of routes.

The situation under which the network will be tested is as follows. The traffic causing congestion, shall represent other network load and will be set at a level that will not, by themselves, cause packets to be dropped, other than ants.

The simulation will start with no data-traffic on the network. The user-data starts before the other sources. Firstly, a constant-bit-rate (CBR) flow, uses the majority of the bandwidth available on the link between Node 0 and Node 2. Secondly, a smaller Variable-Bit-Rate (VBR) flow also uses the same link.

This combination of traffic will result in intermittent congestion during the time when both flows are present on the network. During congestion, traffic will require storage in queues waiting to be transmitted, and ant-packets will be dropped - having been assigned a lower priority than other traffic.

These two flows both stop after a total of fifteen seconds. In the case of the Learning Agent,

the congestion is repeated in an on-off fashion to allow the NN to learn the correct response.

8.5 Comparison Tests

The Agent-Managed Ants must be compared with the ordinary Ant-System to show that an improvement has been made. Therefore the basic Ant-System by itself is used a control test.

Further comparisons could be made with both an *ideal* agent (that is not constrained by processing power/time) and a simple reactive agent (to confirm whether the overhead of the Learning mechanism is actually needed).

The simple reactive agent could just use a simple threshold mechanism, so when the standard deviation exceeds a pre-set level, the age is increased, and reduced when it falls below.

The expected shortcoming of this simple scheme is that it will undoubtedly cause excessive noise during steady state, and setting the threshold to the correct level is a non-trivial task, and it may not be the same for every network – for instance if there are numerous links from one particular node then the standard deviation will be higher on average than on a node with few links.

9 Results

Results show that the Agent is able to improve the speed of reaction of the network to dynamic traffic scenarios; by increasing the Initial Age in congestion, but reducing it otherwise. Comparisons are made with the ordinary Ant-System to evaluate the effectiveness.

9.1 Network One

Figure 2 shows two overlaid graphs for Network One. The graph in the foreground is the probability plot for Node 0 to Node 1. The graph behind shows the agent manipulating the Ant-System – the solid looking block shows that the agent has increased the parameter to increase the response-speed of the system. It can be seen that this occurs when there is a disturbance on the network.

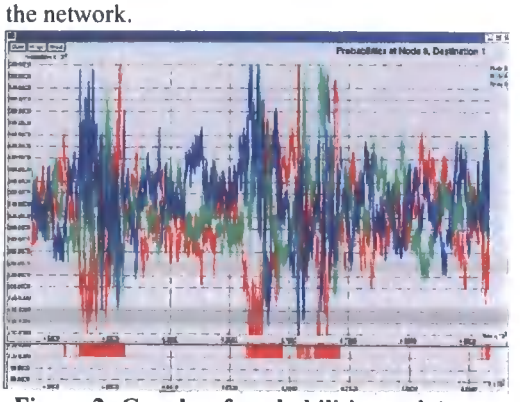


Figure 2: Graphs of probabilities and Agent manipulated Ant-parameters on Network One

At the time of congestion, the congested route is not the selected one, so no quantitative results are relevant.

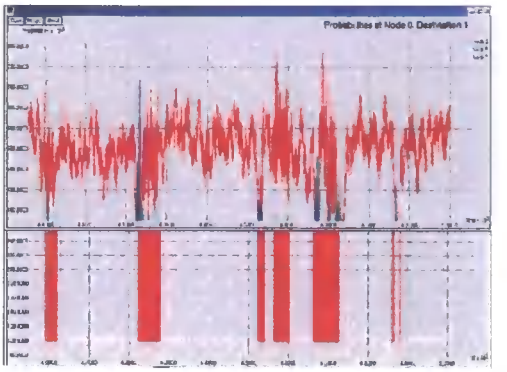


Figure 3: Graphs of probabilities and Agent manipulated Ant-parameters on Network Two

9.2 Network Two

Figure 3 shows two overlaid graphs for Network Two. Again the top one shows part of the probability graph (mainly the highest probability); underneath is shown the age function produced be the agent. It can clearly be seen that when there is major disruption in the probabilities - indicating congestion - the age is changed by the agent (which shows as a solid-looking block), speeding the convergence time.

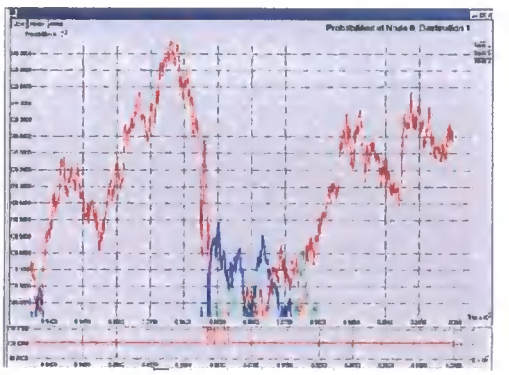


Figure 4: Graph to show improved speed of response with Agent-manipulation on Network Two

Figure 4 also consists of two graphs but shows one portion of congestion in more detail. The time before the route is switched in this case is 3.9s, as compared with 4.2s for the ordinary ants, and 3.7s in the case of the ideal-agent. The threshold solution took, anomalously, 4.3s, as it was expected to improve on the ordinary Ant-System.

9.2 Network Three

Results from Network Three show a dramatic improvement in the performance of the Agent-managed Ants over the ordinary system. Due to the larger size of the network, the route-change

takes longer in the first place – 9.1s for the simple Ant-Routing system to respond. With the Agent operating however, the changeover took only 6.4s; a significant improvement. Figure 5 shows the graphs of the result.

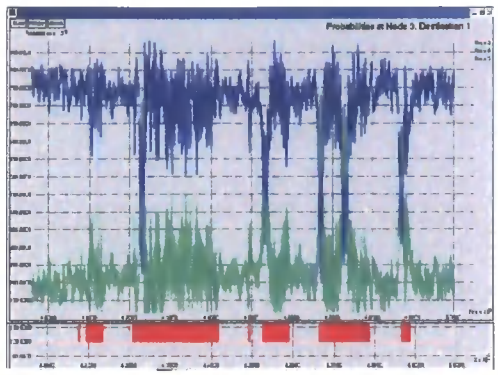


Figure 5: Graph of probabilities and Agent-manipulated Ant-parameters on Network Three

The hard-coded solution took 7.1s to respond fully to the congestion, which (on the basis of a single result) suggests that the learning system adds effectiveness to the Agent, whereas the reactive agent only improved on the Ant-System by 0.1s, at 9.0s.

9.4 Overall Results

It has been shown that the Agent is able to respond to changing conditions on the network and act accordingly. It is able to learn the correct circumstances in which to change the response speed of the ants, both increasing and decreasing it.

10 Further Work

The Agent is currently able to manipulate a single parameter strategically; this was for the sake of simplicity, but it could easily be envisaged that more parameters could also be adapted; such as the rate at which pheromone is dropped, and ultimately the rate at which ants are produced. This, as already noted is a double-edged sword, as the ants may start to contribute to the congestion itself.

Secondly, there is potential for a further agent to operate on the actual routing decision – note that the purpose of an Ant-System is to measure the performance of different routes, while the Agent described here changes its speed of response. The Ant-System in affect "recommends" a route and this further agent could apply reasoning to the process – to stop the route "flapping" and to provide coordination which would prevent routing-loops; as alluded to in [Kapetanakis 2004], this would represent a *massive coordination step.*

11 Conclusions

The Ant-Based System has been shown to find optimal routing solutions to static conditions, but has been found to show undesirable characteristics when put under the strain of dynamic loading.

A solution has been sought that utilises a software agent to adapt to these changing conditions by manipulating a parameter that controls the speed of response of the Ant-System.

The solution has been implemented, and utilises a Neural Network to perform Q-Learning and choose the optimum action for the state of the network.

This Agent has been compared with an Ordinary Ant-Based Routing System, and two similar Agent-managed Ant Systems, one representing an Ideal Agent, and the other a Simple Reactive Agent. As expected the developed Agent, outperforms all but the Ideal Agent.

References

Breslau L., Estrin D., Fall K., Floyd S., Heidemann J., Helmy A., Huang P., McCanne S., Varadhan K., Xu Y., Yu H., "Advances in Network Simulation", IEEE Computer, 33(5), pp. 59-67, May, 2000.

Brooks R.A., "A Robust Layered Control System for a Mobile Agent", IEEE Journal of Robotics and Automation, 2(1), pp:14-21, March 1986

Dorigo M. & Di Caro G., "AntNet: A Mobile Agents Approach to Adaptive Routing", Technical Report IRIDIA 97-12, 1997.

Kapetanakis S., Kudenko D., "Improving on the Reinforcement Learning of Coordination in Cooperative Multi-Agent Systems", Proc. AAMAS-II, AISB'02, pp 89-94, Imperial College, London, April 2002.

Kapetanakis S., Kudenko D., Strens M., "Learning to Coordinate using Commitment Sequences in Cooperative Multi-Agent Systems", AISB Journal, 1(5), 2004.

Schoonderwoerd, R., Holland, O., Bruten, J., "Ant-Like Agents for Load Balancing in Telecommunications Networks" Proc. 1st Int. Conf. on Autonomous Agents, Marina del Rey, pp 209-216, ACM Press, 1997.

Tesauro G., "Programming backgammon using self-teaching neural nets", Artificial Intelligence, 134 (1-2): 181-199 January 2002.

Tesauro, G., "Pricing in Agent Economies using Neural Networks and Multi-Agent Q-Learning", Proc. Workshop ABS-3 "Learning About, From and With other Agents", August 1999.

Vittori K. & Araújo F.R., "Agent-Oriented Routing in Telecommunications Networks", IEICE Transactions on Communications, E84-B(11):3006-3013, November 2001.