

Quality of Service Routing for Real-time Traffic

Michael Gellman

Department of Electrical and Electronic Engineering

Imperial College London

University of London

Supervisor: Prof. Erol Gelenbe

Imperial College London

*A thesis submitted for the degree of
Doctor of Philosophy of the University of London
and for the
Diploma of Membership of the Imperial College*

August 4, 2007

Abstract

Real-time applications such as Voice-over-IP, streaming multimedia, multi-player gaming and pervasive virtual environments account for a massive proportion of the Internet economy. These services consist of traffic flows which have requirements in terms of bandwidth, loss, delay and jitter in order to provide a usable experience to an end-user. Current approaches to providing specialised service for real-time traffic involve queueing mechanisms, resource reservation, traffic engineering or admission control. These strategies all either have scalability issues, do not work in an end-to-end fashion, or are manual and off-line in nature.

This dissertation describes and evaluates new approaches for on-line routing of real-time flows. Our framework for this work is the Cognitive Packet Network (CPN), a self-aware routing protocol which uses a user-directed, goal-based reinforcement learning technique to route flows according to the metrics which most impact their performance. We first compare CPN with an industry standard routing protocol (OSPF), demonstrating through experiments with multiple flows carrying real-time traffic that CPN can quickly converge to the optimal route in a network, and that the speed with which this occurs is proportional to its exploratory overhead. This work is the first to explore CPN's performance with multiple flows competing for the available network resources. This leads to our second contribution where we investigate the impact of competition among selfish network flows and the potential for routing oscillations. We demonstrate that these oscillations can harm overall QoS, and develop methods for eliminating them without sacrificing the benefits of CPN's exploratory nature. Towards supporting applications which have diverse Quality of Service (QoS) requirements, we develop goals in terms of two metrics (loss and delay) and show that CPN can adaptively learn paths in terms of each; in cases of moderate congestion it can decrease network losses by up to a factor of 10. Finally, we show that by using an overlay network, the benefits of CPN routing can be introduced into existing networks with low overhead and without modifying their underlying routing mechanisms.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Erol Gelenbe. He has always provided me the best advice and support, and has been an important figure in my life for many years. Through his generosity, I have been able to both live in and visit a number of different countries and cultures which has greatly expanded my outlook on life. When I first met Prof. Gelenbe I was a final year undergraduate at the University of Central Florida in Orlando. As I am writing this, I am now happily living in London which I completely owe to Prof. Gelenbe for him believing in my capabilities. He has guided the course of my research, and has always been there when I needed his expertise and wisdom.

In addition to my supervisor, I would like to thank my friends and family for their support during the long course of my PhD. My life both in the lab and outside it forms some of my most fulfilling memories. Living in Clayponds Village in the outskirts of London formed the basis for many lasting friendships which I still cherish today. In the lab, I would like to particularly thank the support of my office-mate Peixiang Liu who was always there to talk about research and anything else that might come up. I wish him and his family the best on their new adventure. All my friends from the lab have made my PhD an amazing experience and I thank them for always listening to me, and their always helpful advice. The Intelligent Systems and Networks group (formerly the Intelligent and Interactive Systems group) on the 10th floor was always filled with wonderful, supportive colleagues whom I will always remember fondly.

My family has always supported me, even when I decided to travel to another country to pursue my degree. My parents Richard and Betty, my brother Eric and my grandfather Howard have always been there for me. I would like to dedicate this thesis to my grandmother Florence who passed away this past year – she was a wonderful influence on me and implanted in me the desire to travel to new places and to learn all I could about other cultures, and I'm sure it was her that inspired me to make this wonderful journey I began with my PhD.

Contents

List of Figures	vii
List of Tables	ix
List of Author Publications	x
List of Acronyms	xi
1 Introduction	1
1.1 Research Problem	1
1.2 Contributions	2
1.3 Outline	3
2 Real-time Traffic	5
2.1 Definition of Real-time Traffic	6
2.2 Real-time Applications	7
2.2.1 Categorizing Real-time Traffic	8
2.2.2 Traffic Models	8
2.2.3 Voice Traffic	9
2.3 QoS Requirements for Real-time Traffic	10
2.3.1 Bounds	10
2.3.2 Metrics	11
2.4 Servicing Real-time Traffic	13
2.4.1 Building-blocks for QoS Support	13
2.4.2 QoS Architectures	15
2.5 Summary	16

3	The Cognitive Packet Network	18
3.1	The CPN Routing Algorithm	18
3.1.1	Reinforcement Learning and Routing	19
3.1.2	Keeping Track of the Best Route	21
3.2	Implementing CPN	22
3.2.1	The Original Implementation	22
3.2.2	CPN 2.0	24
3.2.3	Other advantages	28
3.3	Inter-operating with IP Networks	29
3.3.1	IP Applications	29
3.3.2	Tunnelling IP packets through CPN	32
3.4	Summary	34
4	Adaptation and Optimal Routing	36
4.1	Realistic Networks?	36
4.1.1	Wide-area Networks	37
4.1.2	Resource Limitations	39
4.1.3	Choosing a Topology: Receiving real topology data	41
4.1.4	Implementing the Topology	44
4.1.5	Validating our Configuration	45
4.2	Using a First-person Shooter	46
4.3	Experiments	49
4.3.1	Sorting vs No Sorting	49
4.3.2	Discovering Optimal Routes	50
4.3.3	CPN's Response to Network Dynamics	52
4.4	Future Work	53
4.5	Summary	53
5	Eliminating Routing Oscillations	55
5.1	Introduction	55
5.2	Related Work	56
5.3	Oscillations in CPN	57

5.3.1	CPN and the self-load effect	58
5.3.2	Synchronization-caused Oscillations	59
5.4	Oscillations under stationary load	60
5.4.1	Experimental setup	60
5.4.2	Impact of Switching on Performance	63
5.4.3	Packet Desequencing	66
5.4.4	Reward Threshold	67
5.5	Future Work	69
5.6	Summary	70
6	Loss and Delay-based Routing	72
6.1	Related Work	73
6.1.1	Pareto Optimality and the Pareto Frontier	74
6.1.2	Approaches to Multi-objective Routing	75
6.1.3	Summary	80
6.2	Adaptive Routing based on Loss and Delay	81
6.2.1	Formulating a QoS Goal	81
6.2.2	Rewards for Loss	84
6.3	Measurement Results	85
6.3.1	Experimental Methodology	85
6.3.2	Initial Testbed	86
6.3.3	Increasing the Size of the Testbed	88
6.4	Future Work	95
6.5	Summary	95
7	QoS-aware Overlay Routing	97
7.1	Routing Overlays	98
7.2	Peer-to-Peer Overlay Networks	99
7.3	A P2P Routing Overlay	102
7.3.1	Routing Overlay Construction	102
7.3.2	CPN-based Routing Overlay	103
7.3.3	Implementation	107

7.4	Evaluation	109
7.4.1	Testbed	113
7.4.2	Scenario	114
7.4.3	Results	116
7.5	Future Work	119
7.6	Summary and Conclusions	120
8	Conclusion and Future Work	122
8.1	Summary	122
8.2	Future Work	124
	Bibliography	126

List of Figures

3.1	Original CPN Header Layout	23
3.2	The new Cognitive Packet format	25
3.3	Common socket operations	30
3.4	Using LD_PRELOAD to force an application to open a CPN socket	31
3.5	Algorithm used to tunnel an IP packet through a CPN network	33
3.6	Using a CPN cloud to bridge two separate IP networks.	34
4.1	Geographic representation of the Swiss Education and Research Network	42
4.2	The 46-node testbed topology	43
4.3	Quake 2 traffic profile at the client	48
4.4	Quake 2 traffic profile at the server with 10 clients	48
4.5	CDF of packet sizes at Quake II server	48
4.6	Comparing the effect of sorting	50
4.7	IP vs. CPN for different percentages of Smart packets	51
4.8	How CPN reacts to a disturbance along the optimal route	52
5.1	The self-load effect	59
5.2	Topology for the experiments demonstrating synchronization-caused oscillations	60
5.3	Oscillations due to Synchronization	61
5.4	Experimental parameters	62
5.5	Comparing simulated and experimental performance	63
5.6	Oscillation Frequency as a function of FSP	65
5.7	Impact of switching on desequencing	67
5.8	Impact of reward threshold on delay	69
5.9	Reward threshold's impact on oscillation rate	69

6.1	The Pareto Frontier for a two-criteria optimization problem	74
6.2	A convex Pareto Frontier	77
6.3	Small CPN testbed.	86
6.4	Loss and delay in the small testbed with 0% artificial loss rate	87
6.5	Loss and delay in the small testbed with 1% artificial loss rate	89
6.6	Loss and delay in the small testbed with 5% artificial loss rate	90
6.7	Larger CPN testbed.	91
6.8	Loss and delay in the large testbed	92
6.9	Measured reward values at the user-level	93
6.10	Observed reward for different values of T . Note that T is a constant in formula 6.6, not a real time-out.	94
7.1	The two-tier overlay used by Gnutella	101
7.2	Process of receiving and validating a CPN Packet by an overlay router	110
7.3	Routing a CPN packet in the overlay	111
7.4	Set of actions an ingress server would take to utilize the overlay	112
7.5	The 46-node testbed topology	115
7.6	Learning to use the underlying network	118
7.7	When there exists a better route than provided by the underlay, the overlay finds it	119

List of Tables

2.1	Characterisations of popular real-time applications	12
3.1	Ratio of Packet Overhead to Packet Size (PS) for CPN and IP	27
7.1	Parameters for overlay experiment	116

List of Author Publications

M. Gellman. "Quality of Service Routing in Peer-to-Peer Overlays". *2nd Conference on Future Networking Technologies*. Extended Abstract. December 2006. Lisbon, Portugal.

M. Gellman and P. Liu. "Random neural networks for the adaptive control of packet networks". *International Conference on Artificial Neural Networks*. September 2006. Athens, Greece.

Laurence A. Hey, Peter Y. K. Cheung, and Michael Gellman. "FPGA based router for cognitive packet networks". *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology*. pp 331-332. December 2005.

E. Gelenbe, M. Gellman, and G. Loukas. "An autonomic approach to denial of service defence". *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. pp 537-541. June 2005.

E. Gelenbe, M. Gellman, and G. Loukas. "Defending networks against denial of service attacks". *Proceedings of the Conference on Optics/Photonics in Security and Defence (SPIE), Unmanned/Unattended Sensors and Sensor Networks*. Vol 5611 pp 233-243. October 2004.

M. Gellman and P. Su. "Using Adaptive Routing to Achieve Quality of Service". *Performance Evaluation*. Vol 57/2 pp 105-119. 2004.

E. Gelenbe, M. Gellman, P. Su, P. Liu, and R. Lent "Autonomous Smart Routing for Network QoS". *International Conference on Autonomic Computing*. May 2004. New York, USA.

E. Gelenbe, R. Lent, M. Gellman, P. Liu, and P. Su "CPN and QoS Driven Smart Routing in Wired and Wireless Networks". *Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. October 2003. Orlando, USA.

E. Gelenbe, M. Gellman, and P. Su "Self-Awareness and Adaptivity for Quality of Service". *Eighth IEEE International Symposium on Computers and Communications*. June 2003. Kemer-Antalya, Turkey.

E. Gelenbe, M. Gellman, and P. Su "Using Loss and Delay as QoS Goals". *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. July 2003. Montreal, Canada.

List of Acronyms

Ack	Acknowledgement Packet (either Dumb or Smart)
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
CM	Cognitive Map
CPN	Cognitive Packet Network
DAck	Dumb Acknowledgement
DP	Dumb Packet
DPRR	Dumb Packet Route Repository
IP	Internet Protocol
MB	Mailbox
MPLS	Multi-path Label Switching
OSPF	Open Shortest Path First
QoS	Quality of Service
RTT	Round-trip Time
SAck	Smart Acknowledgement
SP	Smart Packet

Chapter 1

Introduction

Real-time applications such as Voice-over-IP, streaming multimedia, multi-player gaming and pervasive virtual environments account for a massive proportion of the Internet economy. These services consist of traffic flows which have requirements in terms of bandwidth, loss, delay and jitter in order to provide a usable experience to an end-user.

In this chapter, we introduce the research problem that we tackle in this thesis (section 1.1); we highlight the important contributions that we have made in this thesis (section 1.2); and we summarize the structure of the remainder of this thesis (section 1.3).

1.1 Research Problem

This thesis deals with issues involved in routing real-time traffic, which is an increasingly more important class of traffic that is not well-served by best-effort networks such as today's Internet. Applications such as Voice over IP or Video on Demand are becoming more and more prevalent in the marketplace, becoming more popular as they add value over more traditional services offered over dedicated telecommunications networks. However, while moving to the Internet offers a number of advantages for applications, it also introduces a number of issues, especially in terms of the effort required to route this real-time traffic on links shared with other traffic.

Current approaches to providing specialised service for real-time traffic involve queueing mechanisms, resource reservation, traffic engineering or admission control. These strategies all either have scalability issues, do not work in an end-to-end fashion, or are manual and off-line in nature. No integrated solution that works from sender-to-receiver has received widespread support in the marketplace, leaving most real-time traffic left to best effort service, and requiring network operators to simply overprovision their networks to accommodate the needs of their traffic. The framework for this thesis is the Cognitive Packet Network (CPN), which is a routing protocol which uses a reinforcement learning-based approach to route traffic according to the needs of its flows. We set out to show that our online routing protocol CPN could adapt to the needs of real-time traffic – in the process expanding on previous work by increasing the size and scope of our analysis.

1.2 Contributions

This thesis makes the following contributions:

- We have updated the implementation of CPN, creating a robust implementation that interacts well with IP legacy applications, is easier to maintain and update, and scales better than the previous implementation through the use of memory caches and more efficient data structures.
- We have experimented with CPN in a real-world scenario with multiple real-time flows, showing that without any a priori information about the network it can perform just as well as an industry-standard routing protocol (OSPF) which requires administrator-determined costs.
- We have brought into question the assumption that load-sensitive metrics result in poor performance along with routing oscillations. Our own experiments with CPN show that even in the face of routing oscillations and frequent route switching, CPN is still able to provide good performance to its real-time flows (although desequencing is a natural consequence, which we show).

- We have investigated CPN's ability to use its reinforcement learning algorithm to find paths that are optimal in terms of metrics which most impact flow performance. These formulations for both single and dual metrics have been used in both wired and ad hoc networks as a generic way to find paths in terms of specific QoS metrics.
- We have developed a method for providing the benefits of CPN routing without the need to replace the current networking infrastructure through the use of a routing overlay. We provide initial evidence through the use of a prototype that this approach can quickly and with low overhead learn better routes than provided by the underlay.

1.3 Outline

The remainder of this thesis is organized as follows:

Chapter 2 completely defines what we mean by real-time traffic. We also give examples of applications which we consider to be real-time and their requirements from the network. Finally, we survey the existing solutions towards providing QoS to traffic, discussing why these are deficient in servicing the needs of real-time traffic.

Chapter 3 introduces the Cognitive Packet Network, a QoS-driven routing protocol that uses reinforcement learning mechanisms to choose paths for flows according to the metrics which most impact their performance. In addition to describing the protocol mechanics, we also go in-depth into the implementation, describing how it has evolved since its beginnings and how it is able to interact with legacy (i.e. IP) networks and applications. The material in this chapter is necessary towards understanding the background for each of the experiments in the following chapters.

Our first set of experiments is presented in Chapter 4. We explore CPN's performance in a scenario with real-time traffic with multiple flows in a large-scale testbed, comparing it to an industry-standard routing protocol (OSPF). These experiments show CPN's capacity for learning the optimal route in a network without any a priori knowledge.

These experiments lead into our work looking into routing oscillations in Chapter 5. When multiple flows co-exist in a network, we show that there is the potential for oscillating behavior, as flows switch back and forth in search of better routes. We investigate methods for controlling these oscillations, and actually demonstrate that the impact of these oscillations is minimal on the performance of our CPN flows.

In chapter 6, we explore CPN's ability to learn paths that are optimal in terms of two metrics: loss and delay. We show that CPN can optimize each metric individually, and can also learn paths that perform well in terms of both. Furthermore, the goal formulations which we have developed in this work have been used by another researcher to combine delay and power in a mobile ad-hoc network, with positive results in terms of extending the network life-time.

Finally, in chapter 7 we explore a method for extending CPN routing to a network without modifying the underlying routing processes through the use of an overlay network. We investigate methods for structuring our overlay using peer-to-peer networking techniques. Our experiments with a prototype implementation show a high level of potential for allowing the benefits of CPN routing to reach a wide audience without overhauling the existing network architecture.

Chapter 2

Real-time Traffic

Real-time traffic represents an important class of applications that is not adequately served by modern “best-effort” networks such as the Internet. As opposed to circuit-switched networks such as the phone system which provide performance guarantees and dedicated resources to a connection, the Internet is a packet-switched network, where packets from a single flow may be forwarded along different paths, receive different levels of service, or may be discarded entirely. One of the key distinguishing features of a packet-switching and circuit-switched network is that resources are not dedicated to a flow; rather there exist queues into which arriving packets from different flows are placed, and subsequently forced to share (possibly unequally) the available routing resources, potentially causing uneven forwarding delays and losses. While applications written to operate over these networks are aware of these inherent limitations, they can still have a large impact over their performance. It is especially problematic for real-time traffic which requires guarantees over its maximum delay, jitter, loss and bandwidth.

In this chapter, we explore real-time traffic in detail: we present our definition of real-time traffic (Section 2.1); overview some of the specific applications that produce and rely on real-time traffic in order to function properly (Section 2.2); discuss their specific QoS needs (Section 2.3); and finally we survey how those needs are currently met, and deficiencies in available solutions (Section 2.4).

2.1 Definition of Real-time Traffic

Real-time is defined in (30) as “a computer communication service whose clients are allowed to specify their performance requirements and to obtain guarantees about the fulfilment of those requirements.” This definition uses four terms “computer communication service”, “clients”, “performance requirements”, and “guarantees” which we specify in more detail.

While “computer communication service” at its most general can mean simply any telecommunication network, in our case we define it as the entire set of routers and links that connect a sending and receiving client. Thus, when we speak of real-time requirements, we are referring to those of the entire communications path, rather than only one segment. This is a requirement for supporting real-time services because they require *end-to-end* performance guarantees, and any segment of a path which does not participate could violate the needs of an application.

The term “client” refers to the application end-points of a communication channel, and can include the needs of the human operators, or those of an application, which may overlap, or could be distinct. For instance, an interactive terminal session may be successful from an application point-of-view as long as each state update is (eventually) successfully delivered; however, the human observer has certain requirements for delay for the application to be considered responsive and well-behaved. The performance requirements of the client are dictated according to its perception of a successful connection – in order to perceive a phone-call as interactive it may be necessary for the delay to be less than 100ms, for example. Thus, it is the needs of the client which dictate the desired performance from the underlying network.

We generally mean “performance” to be a broad term, encompassing any of a client’s needs in order for it to succeed. This can include low-level network metrics such as delay, jitter, bandwidth and loss; but may also include more high-level characteristics such as security and (economic) cost. Specifying performance can be done either quantitatively (a path with 100ms delay), or even qualitatively – an application may desire a path with *good* delay, or

high security. Indeed, the networks that we use for our work attempt to find the *best* network path rather than using an absolute threshold for a performance metric. This is a balance between best-effort networks with no support for selecting paths according to quality, and those which enforce rigid quality requirements.

This leads to specifying the term “guarantee”. In some networks this can be as strict as a contract (e.g. a service-level agreement or SLA) which can be referred to as *hard* QoS guarantees. Other networks relax this requirement and simply allow applications to specify their most-sensitive QoS metrics, which are then used as routing criteria. In this case, the network is not offering a strict guarantee – rather it is performing *soft* QoS routing. Throughout the rest of this chapter, when we talk about real-time routing, unless otherwise specified, we are talking about *soft* QoS.

2.2 Real-time Applications

A fairly comprehensive list of existing Internet-based real-time applications is given in (29): “video-on-demand, IP telephony, VoIP, Internet radio, multimedia WWW, teleconferencing, interactive games, distance learning, remote medical surgery, High Definition TV, and large-scale distributed computing using data grids.” Each of these requires various degrees of service beyond best-effort from the network in order to function at their highest quality, or even to work at all.

If we consider networks beyond the Internet, we can find even more applications that fall under our definition of real-time traffic such as those that operate in sensor networks. These networks typically consist of (possibly many) very low-power devices which are measuring some parameter, and transmitting it wirelessly to one or more data sinks. Simply by the nature of their task, they have real-time requirements (97). The measurements may need to arrive within a certain time-limit for them to remain valid and useful for their purposes. Especially in large-scale deployments, this can involve multi-hop paths for data, which requires some degree of QoS routing.

2.2.1 Categorizing Real-time Traffic

Networked applications can be classified using a number of different categories, some of which overlap, and some of which are orthogonal (79; 29). A general categorization of real-time traffic is that it is *interactive* and *inelastic*. Interactive traffic involves a series of human-human or human-machine interactions; it is the responsiveness requirements which dictate the QoS needs of the application. Elastic traffic is able to perform its duty under a wide variety of network conditions; TCP for instance is elastic in that it adapts its sending rate according to the amount of available bandwidth in the network¹. On the other hand, real-time traffic has a set of QoS requirements which dictate the range of network parameters under which it will perform; if these are violated, then the application will fail.

Real-time traffic can be further categorized as either (in-)tolerant or (non-)adaptive. Many real-time applications can be seen as tolerant to violations in their QoS bounds; for instance, streaming video may be still able to display an image in the case of packet losses. These can be further classified as either adapting to degraded networking conditions (*adaptive*) by reducing their sending rate, for example. Nonadaptive applications may be able to tolerate QoS degradation, but the impact is perceivable by the end-user. Adaptivity and tolerance are considered to be orthogonal categorizations for real-time applications (21).

2.2.2 Traffic Models

As we have discussed, there are many different types of real-time applications; yet it would be useful to be able to categorize the traffic that they generate in order to have some models with which to understand the impact they will have on the network. Broadly speaking, we can divide them into two categories: *constant bit-rate* (CBR) and *variable bit-rate* VBR (3).

Constant bit-rate applications generate data in packets of fixed size at fixed intervals. This can include applications where sensors periodically probe the environment and transmit their data

¹Of great importance to issues such as stability and fairness within TCP is the works of Frank Kelly (such as (59; 60)). We mention them for completeness of our discussion, but highlight that since we are chiefly concerned with real-time traffic, the discussion therein would have to be adapted to the needs of our traffic.

to some central processing server (such as air-control systems or sensor networks). Another class of real-time application that can be considered CBR is first-person shooters, where the client transmits its decision-making to a central server at fixed intervals (see Section 4.2 for our own measurements of a networked game demonstrating this characterization).

Taking the category of variable bit-rate, we can further specify traffic as either *on/off* or *periodic with variable packet sizes*. On/off traffic describes an application which alternates between periods of activity and silence, with the active period typically characterized by generating fixed-size packets at a deterministic interval. An example is voice traffic, which we discuss in the next section in more detail. Another type of VBR traffic is generated by video, which for the same quality level can generate packets of variable size during each fixed period, depending on the compression ratio and content of the set of frames (3).

2.2.3 Voice Traffic

Voice traffic is one of the oldest real-time applications, existing since the invention of the telephone. It is also one of the most studied due to its commercial importance, and has had a large impact on networking research and output over the decades. For example, the 48-byte cell for ATM “was chosen primarily for the benefit of voice applications – in particular, to avoid the use of echo cancellation equipment on continental connections (3).” We include an extended discussion of voice traffic here as an example of an important real-time application, and because of the large body of research that exists towards characterizing its nature.

As we discussed above, voice traffic is typically characterized as on/off variable bit-rate traffic. This is due to the nature of the application, namely a conversation between two people. A typical conversation consists of alternating periods where one party talks and the other listens.

The distribution of these periods of silence and activity have been modelled using Markov models either for both participants or a single one. In these models, the period of talking is taken to be exponentially distributed, followed by an exponentially distributed period of silence. These periods are assumed to be independent from each other. Characterizing voice

traffic has been a research topic for decades, with many models originating out of Bell labs in the late 60's. Many references to this pioneering work are listed in (3).

2.3 QoS Requirements for Real-time Traffic

Real-time traffic is differentiated from other types of data traffic in that it depends on the networking layer satisfying their QoS requirements. While there are many different applications, each with their own requirements, these typically encompass the same set of networking metrics, or at least can be mapped onto the ones we list. We begin by discussing the ways in which QoS requirements can be specified by an application, and then list the most common QoS metrics that impact networked application performance.

2.3.1 Bounds

When strict bounds need to be specified by an application (i.e. hard QoS), this can be done in different manners. Here, we discuss bounds which are boolean, deterministic, and stochastic (30). Boolean bounds are a simple form of QoS specification where a network requirement is given as a true/false statement.

A more widely used bound, and that which we focus on in the rest of this section, is a *deterministic bound*, where a specific upper-limit (or lower limit) on a metric is given that the network must not violate in order for the application to successfully function. This can be represented as $var \leq bound$ where *bound* is specified by the client, and *var* is controlled by the network.

We can also express a performance bound probabilistically $Prob(var \leq bound) \geq P$, in which case we have a *stochastic bound*. The client must specify both a *bound*, and a probability P . It should be noted that deterministic bounds can be seen as stochastic bounds with probability 1.

2.3.2 Metrics

The requirements for networked applications can normally be expressed in a small number of common metrics. These can include:

- **Bandwidth** There are two concepts that are involved in bandwidth, *capacity* and *available bandwidth*, of which the latter is the relevant metric to a real-time application. Capacity is the “maximum rate that the path can provide to a flow, when there is no other traffic [on the path] (57).” On the other hand, available bandwidth is “the maximum rate that the path can provide to a flow, without reducing the rest of the traffic [on the path] (57).”

Real-time applications can specify their bandwidth needs as either a constant bit-rate, or something more sophisticated such as a “leaky-bucket model”, where they specify both an average and a burst rate. The model they use to specify their bandwidth needs is directly correlated to the profile of the traffic they generate (see Table 2.1).

- **Delay** Delay is the most common QoS requirement of a networked application (consider Table 2.1 where only application group 3 does not specify that delay is an important factor in its performance). It is important that messages generated by a sender are delivered in a timely manner to the destination. Especially in the case of real-time applications, a delayed packet may count to an application the same as a lost packet if the information it carries is no longer useful when it arrives. For this reason, many of the experiments that we conduct in this dissertation consider delay before any other QoS metrics.
- **Jitter** Jitter can be measured in a number of ways, but it is typically expressed as the variance of delay. As an example of the fact that no common criteria exists for expressing variance, 10 different methods for expressing variance are given in (63). Regardless of how it is measured however, variance is still a very important factor for real-time traffic. Applications such as real-time voice require a steady stream of data packets to ensure regular playback in a timely manner. As the rate at which packets

Application Group	Application Type	Throughput	Delay	Jitter	Error	Loss
1	IP telephony		$\leq 150ms$	**		*
1	MPEG 1/2 A/V	*	$\leq 150ms$	**		*
2	MPEG 4/7 A/V	**	$\leq 150ms$	**	*	**
3	Audio on Demand	*		*		*
3	Video on Demand	**		*		*
3	Internet TV	**		*		*
4	Online Banking		*		**	**
4	Online Games		**		**	**
4	Online Trading		**		*	*
5	Chat		*			
5	Web Browsing		*			
5	Telnet		*			*
5	Classical e-mail		*			

Table 2.1: Characterisations of popular real-time applications. (**) denotes a strong dependency whereas (*) indicates importance with a limit of flexibility. Table taken from (27)

arrive at the destination varies more and more, it becomes harder and harder for the destination to predict when the next packet will arrive. Techniques such as buffering can compensate for a degree of jitter, but they increase the processing overhead at the destination, and can only go so far towards compensating for a high amount of jitter (e.g. how large to size the buffer).

- **Loss** Getting data reliably delivered is an important factor for many applications, where the amount of loss they can tolerate depends on the application (as can be seen in Table 2.1). In order to decrease their sensitivity to loss, many applications employ encoding schemes to reduce the impact of loss at the cost of additional processing requirements and network-layer overhead (e.g. Forward Error Correction). However, loss is still important to consider when routing traffic, and our work in Chapter 6 considers this metric in detail.

Other QoS requirements include Sequencing, Absence of Duplications, Failure Recovery, and Service Setup Time (30).

2.4 Servicing Real-time Traffic

A number of different methods have been proposed towards providing QoS to real-time traffic. As with any technical solution, these approaches have seen various levels of success and commercial adoption depending on their complexity and perceived benefits. Additionally, some of these approaches can be implemented at a single router while others need to be in place in an end-to-end manner from source to destination adding to the cost of adaptation – we highlight where this is the case in our description. We first introduce two building blocks for QoS support that are present in nearly all QoS architectures, and then describe two architectures that have been proposed for IP networks, but which have never seen widespread adoption. Many of these techniques are listed in a survey paper by El-Gendy, et al (29).

2.4.1 Building-blocks for QoS Support

The IP protocol does not have a true notion of QoS as a routing protocol, and this absence has hampered many applications which have strict requirements such as real-time traffic. While there does exist the IP precedence and Type of Service (ToS) field in the IP standard, it has not seen great adoption due to “the absence of strict rules for processing the IP ToS field in IP routers (29).” This lack of cohesiveness across administrative domains made it difficult to agree on common actions for packets marked in this way, leading to the field being virtually ignored by many routers. This lack of direct support for QoS in IP has led to a number of proposals that “bolt on” additional mechanisms for routing traffic that needs better than “best-effort” service.

Flow Scheduling

One way to provide improved quality to certain types of traffic is prioritize some flows over others. The most basic queueing mechanism is First in-First out (FIFO) which does not provide any QoS to traffic – a packet from a real-time flow may arrive to a long queue and

wait for all the packets ahead of it, increasing its overall delay. However, other disciplines such as priority scheduling, generalized processor sharing, weighted fair queueing, and class-based queueing offer either multiple queues or prioritize some traffic over others to reduce its queueing delay and drop probability (29).

These service disciplines can be adopted at any router in the network, and do not require any coordination among other routers for them to make a difference. However, it is this lack of coordination which reduces the overall impact that a queueing discipline can make to the end-to-end QoS that a flow receives from the network. If the bottleneck router along a path does not support these mechanisms (for instance, if it only uses a FIFO service discipline), then the impact of any other router on the path implementing these service disciplines is limited.

Admission Control and Policing

A network has a finite set of resources (routers and links) with finite capacity; if this capacity is exceeded, then congestion sets in, having a negative impact on QoS. For this reason, one common feature of many QoS architectures is that it implements some form of admission control where a new flow must first inform some control mechanism of an estimate of its size and requirements (e.g. bandwidth and traffic profile, in addition to its delay and loss requirements) which are evaluated in terms of the current state of the network and a decision is made as to whether the flow can be admitted to the network.

Hand-in-hand with Admission Control is the issue of policing the active flows in the network. The initial admission of a flow is based on the flow's estimate of its bandwidth and traffic nature. If these estimates are violated, then a flow may request more service from the network than it can provide, and adversely impact the other flows. For this reason, it becomes necessary to implement some form of enforcement mechanism, typically in the form of a *token bucket* which is a policing mechanism which can control the peak and average rate of a flow (29).

2.4.2 QoS Architectures

While admission control, policing and scheduling mechanisms all have use on their own, when they are combined in a QoS architecture, they can provide guaranteed levels of service to traffic. These service contracts allow real-time traffic to coexist with other flows knowing that they will be able to operate successfully. However, while architectures with these guarantees have been proposed, they still have not seen widespread adoption due to a number of factors which we discuss in the following subsection when we introduce IntServ.

IntServ and RSVP

IntServ was proposed by the IETF (13) as a proposal to compensate for IP's lack of end-to-end QoS support. It combines the building blocks which we introduced above (scheduling, admission control and policing) into a cohesive framework that provides assured and deterministic QoS guarantees to network flows. Unfortunately, for reasons we discuss later, this architecture never saw wide adoption due to number of deficiencies.

A key facet of the IntServ model is that it uses the Resource Reservation Protocol (RSVP) to reserve resources on every router a flow passes through from the source to the destination. This is done at the beginning of a connection, and is done per-flow on each router in the path. This per-flow state was one of the largest disadvantages of the IntServ framework, as it limited its ability to scale to large networks and into the network core. Additionally, this reservation procedure introduced an additional overhead into connection establishment which for small transactions was prohibitive. For more details about the IntServ architecture we refer interested readers to (29).

The IntServ framework never saw widespread adoption in large networks such as the Internet because of these scaling issues. Going back to the drawing board, the IETF proposed a more scalable alternative which is the topic of our next subsection.

DiffServ

As a successor to IntServ, the differentiated services (DiffServ) (10) framework was proposed to address IntServ's scalability issues. As opposed to strict end-to-end dynamic signalling for each flow, DiffServ operates on traffic aggregates and does not require an explicit signalling mechanism. Instead of explicit resource reservation, flows indicate which class of service their packets fall into (decided not by the source but by the network operator at the ingress router) and are prioritized accordingly at the other routers in the network.

DiffServ has also not seen widespread adoption. It was implemented into an experimental testbed called the QBone as part of the Internet2 initiative. This concluded that there are difficulties involved in its wide-spread deployment. Again, interested readers can refer to (29) for more detailed information on DiffServ.

Other architectures

In addition to IntServ and DiffServ which were specifically designed to interoperate with IP networks, a number of other architectures have been developed which incorporate various levels of QoS support. Existing between layers 3 (network) and 2 (link) are ATM and MPLS, two architectures that have seen varying levels of adoption and success. ATM did not see wide-spread adoption (outside the backbone) mainly due to its complexity while MPLS is a new architecture that, while it does not explicitly include end-to-end QoS support, it enables operators to do traffic engineering to manage the level of congestion on their networks. We mention them here for completeness, but because they are not strict layer 3 solutions, we do not go into details, referring readers to (29) for more details.

2.5 Summary

In this chapter we have defined what we mean by real-time traffic, why it is an important class of traffic, its needs and existing approaches to dealing with it in today's networks. Today's

networks are increasingly being used to carry real-time traffic, as applications such as Voice over IP, Video on Demand and distributed gaming become more and more popular. As the proportion of real-time flows in the Internet has increased, the fact that the specific requirements of these flows are not taken into account by the best-effort forwarding mechanisms in the Internet has become more and more important. The large number of proposed QoS architectures highlights the fact that some form of support is necessary, yet their failure to see wide-spread adoption means that a hole exists where an approach is needed. In the next chapter we introduce the Cognitive Packet Network protocol, which we see as fulfilling the needs of real-time traffic.

Chapter 3

The Cognitive Packet Network

The Cognitive Packet Network (CPN) has been proposed as a QoS-driven routing protocol that allows its users to define the path selection criteria (44; 42; 41; 46). This approach has been shown to be effective for a variety of uses, including: traffic balancing (43), power-based routing in mobile ad-hoc networks (40), and Denial of Service protection (38).

The material in this chapter forms the basis upon which the rest of this thesis is based. It starts with a brief overview of the concepts involved in CPN routing (Section 3.1). Then, why and how the current CPN implementation has evolved from its initial implementation is presented (Section 3.2). Finally, we discuss the various methods in which IP applications are able to take advantage of CPN routing (Section 3.3).

3.1 The CPN Routing Algorithm

CPN includes three types of packets which play different roles:

- Smart packets (SP) are used to discover routes and update network snapshots for connections; at each hop they are routed according to the experiences of previous packets (see Section 3.1.1 for details). In order to enhance the explorative nature of an SP, at some hops, with a small probability (normally 5%), it makes a *random* routing decision.

This helps to avoid over-learning a particular route. As SPs travel the network, they store QoS data (such as timestamps, counters, etc.) in a special data storage area of the packet header known as the cognitive map (CM).

- When a (Smart or Dumb) packet arrives at its destination, a corresponding acknowledgement (Ack) packet is generated which stores the route taken by the original packet, and the measurement data it collected during its journey. The Ack will then return along the *reverse route* (using a loop-removal algorithm to avoid circuits). At each hop an Ack visits, it deposits information in a special short-term memory store (referred to as *mailboxes*). At the source, the route carried by an Ack, along with its QoS data is cached in a table we refer to as the Dumb Packet Route Repository (DPRR).
- Dumb packets (DPs) carry user data using routes brought back by Smart Acknowledgements (Sacks) for source routing. They also collect measurements during their trip through the network, which are brought back to a source node by a Dumb Acknowledgement (DAck) packet.

For further information on mailboxes, and the loop-removal algorithm used by Acks, the reader is referred to (46). To clarify our terminology further, when we say Ack, we can mean either a Dumb or Smart Ack.

3.1.1 Reinforcement Learning and Routing

Different approaches to learning could in principle be used to discover good routes in a network, including Hebbian learning, back-propagation (89), and reinforcement learning (RL) (100). Hebbian learning was shown to be slow in the simulation studies that were conducted at the beginning of the CPN project (36; 50; 45) and was excluded from further consideration. Simulation experiments conducted on a 100 node network (36; 46) showed that RL is far more effective, and that it provides significantly better QoS than shortest-path routing (41). In order to guarantee convergence of the RL algorithm to a single decision (i.e., selecting an output link for a given smart packet), CPN uses the random neural network (RNN)

(35; 49; 39) which has a unique solution to its internal state (35) for any set of *weights* and input variables. CPN's RL algorithm uses the observed outcome of a decision to *reward* or *punish* the routing decision, so that future decisions are more likely to increase or maintain the desired QoS reward R which is defined in terms of the metrics which characterize the success or failure of the outcome; e.g. in terms of packet delay, loss, jitter, or some composite metric.

To clarify the role of the RNN within CPN, we describe its use in a general way. The RNN has a number of neurons equal to the number of routing decisions, with each neuron connected to every other one in a fully-recurrent mesh. The neuron which has the greatest excitation value is chosen by the routing algorithm as the next-hop for a Smart Packet. There are both excitatory and inhibitory weights on these connections which allow us to reward and punish routing options as we now show.

As an example, the reward equation for the delay QoS class is:

$$R = \frac{1}{D} \quad (3.1)$$

where D represents the round-trip delay measurement of the most recent Ack packet. Similarly, the reward equation for loss is:

$$R = \frac{1}{L + \epsilon} \quad (3.2)$$

where L is the most recent loss measurement, and ϵ represents some lower bound on the loss value (typically 0.01). As an example of composite reward equations, see Chapter 6.

In (46; 41) the RL algorithm that is used by CPN is described in more detail for the interested reader; here we provide a brief summary. Each arriving Ack has a corresponding reward, successive values of which are denoted by R_l , $l = 1, 2, \dots$. These, in turn, are used to update a *threshold* or historical value of the reward:

$$T_l = \alpha T_{l-1} + (1 - \alpha) R_l \quad (3.3)$$

where α is some constant close to (but less than) 1.

Thus T_{l-1} represents our *expectation of reward* so that if $R_l > T_{l-1}$, we *reward* the previous decision by increasing the excitatory weights leading to its corresponding neuron, and slightly increase the inhibitory weights of the other neurons. On the other hand, if the reward is below the threshold, $R_l < T_{l-1}$, then we *punish* the decision by increasing its inhibitory weights, and also slightly increasing the excitatory weights of the other neurons so that the other choices for output ports can compete more effectively in the next round of decision making.

3.1.2 Keeping Track of the Best Route

In the originally proposed algorithm, it stated that the route brought back by a SACK would be used for data traffic immediately, under the assumption that it contains the most up-to-date information about the network, and would be the best route decided upon by the network. However, this does not take into account the fact that, in order to better explore the network, each router will (with a small probability) sometimes choose a random next hop for an SP, rather than the one decided upon by the RNN. This decision may or may not result in better QoS for the route, and thus we propose to compare the quality of routes brought back by a SACK against the current route before switching.

When a SACK arrives at its destination, we compare the reward for this route to the currently active one. Only if the newly arrived SACK's reward is greater than our current route do we switch. DACKs update the reward stored with their route, but do not cause the active route to change. In the remainder of this thesis we refer to this optimization as *CPN with Sorting*.

While at first this may seem to deprive us of some of the exploratory benefits of the CPN algorithm, we demonstrate in Section 4.3.1 and 4.3.3 (also published in (51)) that this is not the case because of DACKs. Because each DACK also triggers the RNN-RL algorithm at each router, a sudden decrease in quality along a route will punish the decision, and cause the next SP to choose an alternate next hop, which will enhance its chance of discovering a new best route; thus, the network is still able to quickly respond to changes in network conditions, and

it also ensures that data traffic always uses the best available path.

3.2 Implementing CPN

While the basic CPN routing algorithm has seen little change over the course of its use, the actual implementation of the protocol has undergone a great deal of development through various levels of involvement of CPN researchers. This section serves to document a number of important developments that have taken place since the original CPN publications.

3.2.1 The Original Implementation

Originally, the CPN code was based on the IPX code within the Linux kernel, and was primarily authored by Ricardo Lent and Zhiguang Xu at the University of Central Florida. It served its purpose of demonstrating that the CPN ideas for routing were viable, but it quickly began to show some growing problems as new features were requested.

The primary problem with the CPN code has to do with the IPX code it was derived from. There was no clear (in fact, any) separation between the network and the transport layer, making the addition of a stream-oriented protocol like TCP a very difficult undertaking. In addition, the IPX implementation had seen little support, and a lack of testing due to the gradual market failure of the Novell networking stack with which it was designed to be compatible.

The old CPN Header

Another problem with the implementation was the format of the packet header. As can be seen in Figure 3.1, the Cognitive Map (CM) and the route are intermingled. This comes from the fact that the original QoS metrics used in CPN routing were delay-based, and the CM storage was used for time-stamp information. However, the format is wasteful for QoS metrics such as simple shortest-path routing, because SPs do not need to estimate delays, and thus

timestamps are not needed. In addition, the forwarding process at each node requires a linear search of the route (in the packet header), an operation which is $O(N)$ if N is the number of hops in the route. Due to the different header formats for different type of CPN packets, the forwarding process is further complicated by requiring multiple decoding routines. All of these issues pointed to a need for a change.

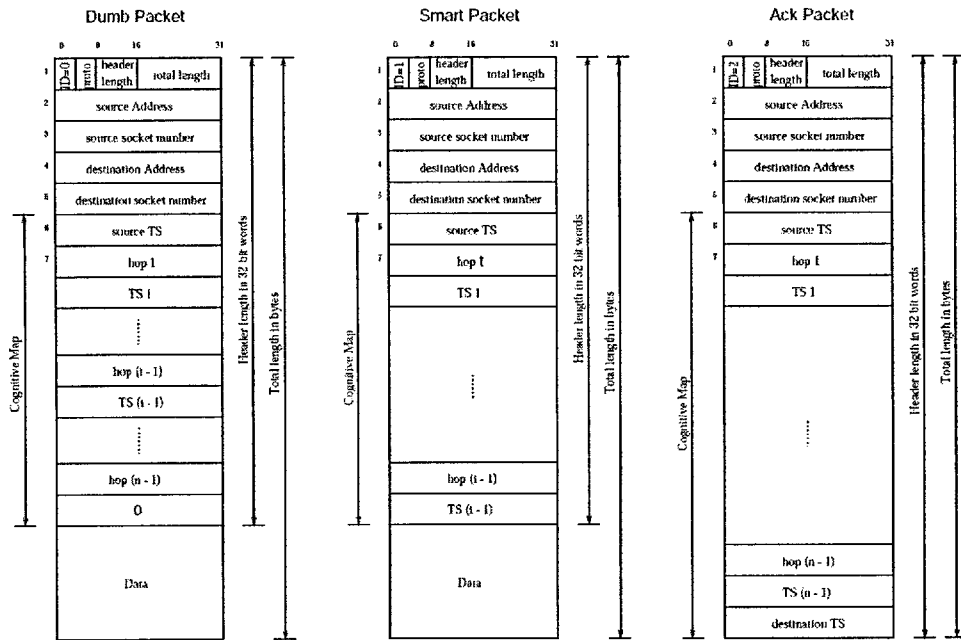


Figure 3.1: The old Cognitive Packet format. Notice that time-stamp information has been hard-coded into the Cognitive Map, and it is not obvious where to store any additional QoS data. Figure copied from (42)

Manual Neighbor Discovery

A final attribute of the CPN code that was insufficient was the need to hard-code the MAC addresses (the Ethernet identifiers) of each neighbor at each router. The MAC address information is required in order for two CPN nodes to communicate at the physical layer; they need to know the address to which to address the packet. Hard-coding of the address was necessary due to the lack of a neighbor discovery protocol. This also meant that it was impossible to know when a neighbor had been disconnected, meaning that CPN would continually forward

Smart Packets to an inactive host; resulting in sub-optimal behavior from the algorithm.

3.2.2 CPN 2.0

The next version of CPN needed to start with a different code-base. Writing a new network protocol from scratch was out of the question due to time constraints and lack of resources. However, this was not required because we already had the source code for a very well-exercised and optimized networking stack, IPv4. We decided to use the IPv4 networking code from Linux 2.4.24 as a base from which to start our CPN code.

By using IPv4 as a starting point, we immediately gained the benefit of true network and transport layer separation. In fact, the UDP and TCP layers could be included in CPN with little modifications. An immediate benefit of this is that CPN and IP-enabled hosts can directly communicate through the use of network tunnelling (see section 3.3).

A New CPN Header

One of the first design decisions that was made during the re-write was to overhaul the design of the CPN header to overcome some of the problems mentioned above, including: creating a unified format for each type of packet, simplifying the decoding logic; disentangling the CM and the route, and adapting the fields to support a more generic separation between the network and the transport layer. The new CPN header that accomplishes these goals is seen in Figure 3.2.

The fields in the header are as follows:

1. *version*: The version of the protocol. Currently 1.
2. *flags*: Additional field for flexibility and expansion. Currently unused.
3. *header length*: The length of the CPN header in words².
4. *checksum*: Verify the integrity of the CPN packet with a checksum.

²A word is defined as 32 bits

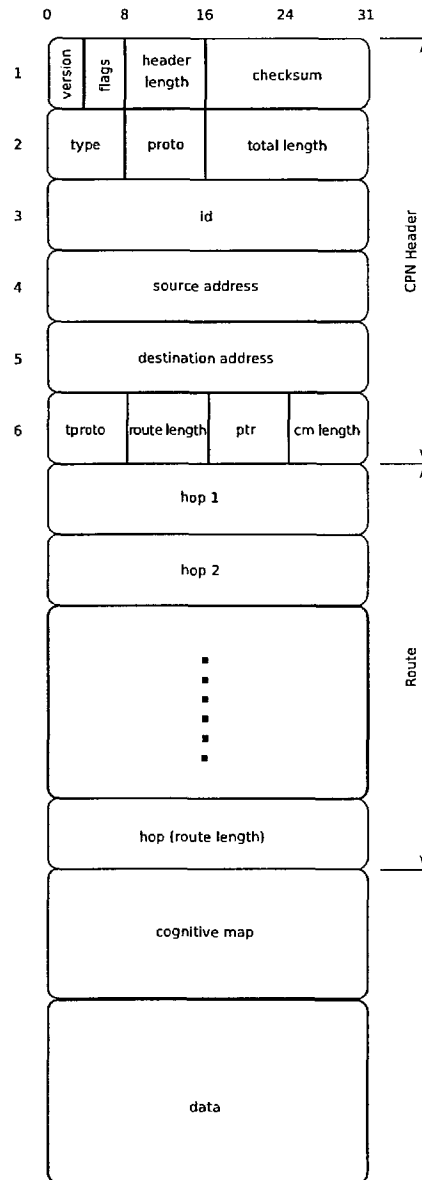


Figure 3.2: The new Cognitive Packet format. The new header separates the Cognitive Map from the route. In addition, the header is the same for all types of CPN packets, streamlining the decoding process.

5. *type*: Indicates whether the packet is an SP, DP, or an Ack.
6. *proto*: The type of QoS required (i.e. the reward type).
7. *total length*: The total length of the packet (in bytes).
8. *id*: A unique identifier for each packet.
9. *source address*: The original sender of the CPN packet.
10. *destination address*: The destination of the CPN packet.
11. *tproto*: The transport protocol identifier (i.e. whether UDP or TCP).
12. *route length*: The length of the attached route in words.
13. *ptr*: An index (*pointer*) into the attached route to tell the current node where it is located in the path.
14. *cm length*: The length of the CM in words.
15. *route*: An attached route with *route length* hops.
16. *cm*: Contains reward-specific data of size *cm length* words.
17. *data*: The data portion of the packet.

With the unified packet architecture, a number of fields have been added/modified to be more flexible. The *ptr* field avoids a linear scan of the route at each hop; forwarding nodes simply increment (or decrement if an Acknowledgement packet) the *ptr* field at each hop to know the next hop for a packet. The addition of *cm length* and *route length* allows each reward to have a variable-sized CM space in the packet, avoiding the pre-calculated nature of the old header.

CPN Header overhead

One of the key differences between CPN and IP is that the packet header is much larger, due to the additional route and QoS data. Consider the data displayed in Table 3.1. With small

Hops	CPN – PS 100	CPN – PS 1500	IP – PS 100	IP – PS 1500
0	0.52	0.03	0.4	0.03
1	0.6	0.04	0.4	0.03
2	0.68	0.05	0.4	0.03
3	0.76	0.05	0.4	0.03
4	0.84	0.06	0.4	0.03
5	0.92	0.06	0.4	0.03
6	1	0.07	0.4	0.03
7	1.08	0.07	0.4	0.03
8	1.16	0.08	0.4	0.03
9	1.24	0.08	0.4	0.03
10	1.32	0.09	0.4	0.03

Table 3.1: Ratio of Packet Overhead to Packet Size (PS) for CPN and IP

packet sizes, as the number of hops increases the overhead quickly becomes larger than the data that is being carried. However, for data packets approaching the maximum packet size, this reduces quite a bit.

One way to reduce this overhead is to limit the number of hops put into a packet header, as has been looked at by Liu, et al in (72). They introduce the idea of *recursive routing* whereby if intermediate CPN routers along a path know how to get to the destination, they are able to generate an Ack with the path information directly without going all the way to the destination. Combining this approach with some kind of hierarchal routing mechanism would greatly reduce the number of addresses and QoS data needed to be placed into packet headers.

Neighbor Discovery

By incorporating the Address Resolution Protocol (ARP) code from IP (which is responsible for resolving an IP address to an underlying physical-layer MAC address), and making small modifications to the protocol, a dynamic neighbor discovery protocol was fashioned. The protocol works according to the following algorithm:

1. When a node wishes to participate in a CPN network, it broadcasts a HELLO message

to its neighbors, signifying that it is available for routing.

2. Neighbors receiving the message add the new host to their list of neighboring nodes, for the interface that the request was received on.
3. The receiver then sends a HELLO message back to the new node to complete the discovery process.

In order to detect a node failure, if no packets have been received from a neighbor after some period of idleness (currently 60 seconds), a HELLO packet may be broadcast in order to confirm that a neighbor is still alive. If, after 3 HELLO packets, the neighbor does not respond, its entry is deleted from the neighbor tables, and it is unavailable for future route requests. At some later time, if the node reappears, it will use the above algorithm to re-establish a routing partnership.

This protocol works for our existing environment, but still lacks a few features that would preclude its use in an unsafe environment. First and foremost is security - any node which is directly attached to a CPN node is assumed to be available for routing. This could allow malicious attackers to easily intercept all data routed through a node, especially if the attacker manipulated the QoS information to appear to lie on the optimal QoS route. Security could be greatly enhanced through the use of a Public-key Infrastructure, but this is left to further research.

3.2.3 Other advantages

In retrospect, a number of other advantages have come about by using the IPv4 code. We found that this increased performance, development speed, and maintainability.

Because it is so wide-spread, IPv4's implementation and code structure has been stress-tested and optimized for a variety of operating conditions; by using it as our base-line, we immediately gained the experience of many years of maintenance. By rewriting a number of kernel structures to use hash tables instead of LRU (least-recently used) stacks, and utilizing

memory caches instead of dynamic memory, the performance of the module greatly improved (compared to version 1 of the implementation). We also benefited early on by removing all floating point operations in favor of fixed point arithmetic.

There exist a number of books and articles dedicated to the IPv4 code-base, which greatly accelerated the development progress. In addition, the number of bugs in the code-base can be considered to be close to 0 for all of the important cases. This meant that development and debugging efforts could be tightly focused on new features that had been added.

Another benefit of using IPv4 was that when it came time to upgrade our testbed kernel from 2.4 to 2.6, it was a half-day task to update the kernel module. The process amounted to reading the 2.6 IPv4 code and comparing it to the older version, and making the corresponding changes to our module to account for new locking schemes, structure changes, and so forth. Had we written our own module from scratch, this process would have taken much longer, and would have required a much deeper understanding of the underlying kernel and the changes in the two versions.

3.3 Inter-operating with IP Networks

Integrating with IP³ applications and end-hosts is an important characteristic of any networking protocol. As IP is the *de facto* standard in any modern networked environment, it is critical to maintain some kind of compatibility when designing any new protocol. In this section, we describe how IP applications can be made to use CPN, as well as a method for interconnecting IP sub-networks with CPN.

3.3.1 IP Applications

Modern operating systems typically use what is called a socket to implement network programming. A socket can best be described as an endpoint for communication - two connected sockets form a connection for data to be sent back and forth. Sockets typically are divided

³When we say IP, we refer to version 4 of the protocol unless otherwise specified.

```
int socket(int socket_family, int socket_type, int protocol);
int bind(int s, const struct sockaddr *name, socklen_t namelen);
int listen(int s, int backlog);
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
int close(int fd);
```

Figure 3.3: Common socket operations. An IPv4 application needs to change the `socket_family` value to `AF_CPN`, and any `sockaddr_in` structures to `sockaddr_cpn`.

into two types: stream and datagram. Stream sockets guarantee that the data at the source will be successfully delivered to the destination, and in the same order as it was sent. On the other hand, datagrams make no such guarantees, and are more appropriate for applications that do not need these requirements such as interactive applications. We can use either type of application over CPN with a few small modifications.

Use the source

On operating systems such as Linux, a great majority of applications have their source code available. When this is the case, adding CPN support can amount to a few very simple changes to the code which creates and connects a socket.

When a socket is first created (see `socket` in Figure 3.3), the first parameter is the `socket_family`. For an IP application, this is set to `AF_INET`. By changing this to `AF_CPN`, the kernel will instead create a CPN socket, and subsequent operations on the socket such as establishing a connection and sending data will use CPN-specific operations. Because the size of the address in CPN is kept at 32 bits, CPN and IP addresses are completely compatible, again greatly simplifying converting an IP application to CPN.

The only other step that must be taken to complete the conversion to CPN is to replace usage of the `sockaddr_in` structure with `sockaddr_cpn`. This guarantees that the `sa_family` is properly set to `AF_CPN` and the CPN-specific fields are properly filled in. By making a few small changes to the source code of an application, most apps never written with a notion of CPN can be modified to take direct advantage of the new routing protocol.

When source code is not available

Even for an application for which the source code is not available, there are still ways to force the usage of CPN. One method uses the LD_PRELOAD environment variable, and the other is to use *tunnelling*. While these are surely not the only techniques that would work, they cover the scenarios whether root access is available or not.

The LD_PRELOAD environment variable affects how the run-time linker locates the shared libraries that are specified by an application at compile time, providing a middle layer between a program and its needed libraries. By intercepting the various socket functions and redirecting them to their CPN-specific counterparts, an application which has been written to use IP can use CPN without it ever needing to know.

```
int (*real_socket)(int, int, int);

void _init (void)
{
    const char *err;

    real_socket = dlsym (RTLD_NEXT, "socket");
    if ((err = dlerror ()) != NULL) {
        fprintf (stderr, "dlsym (socket): %s\n", err);
    }
}

int socket(int socket_family, int socket_type, int protocol) {
    socket_family = AF_CPN;

    return real_socket(socket_family, socket_type, protocol);
}
```

Figure 3.4: Using LD_PRELOAD to force an application to open a CPN socket

An example section of C code for over-riding the `socket` call is shown in Figure 3.4. Whenever an application linked with this library attempts to open a socket, it will be returned a CPN socket, and subsequent socket operations will use the CPN module. Other calls that would need to be modified are any that use `sockaddr_in`, converting these to use `sockaddr_cpn` and the `AF_CPN` address family.

When root access is not available, the LD_PRELOAD method described above is a good way to force a single application to use CPN. However, for more complicated applications or where maintaining some degree of IP support is desired, the wrapper library can become a complicated solution.

3.3.2 Tunnelling IP packets through CPN

Another approach to bridging IP and CPN (for which root-level access is required) is tunnelling. This involves modifying the IP routing table so that before an IP packet is sent out on the wire, it is first encapsulated into a CPN packet. The process requires the designation of a *tunnel source* and a *tunnel edge* by an administrator. Choosing whether or not to encapsulate an IP packet is done at the granularity of IP forwarding rules, which allows for specific hosts, subnets, or every packet on the host to be specified. In addition, multiple CPN routers can be eligible for de-encapsulating a given IP packet, and it is up to the CPN algorithm to find the best route. The full algorithm is outlined in this section.

Definition 1 *A tunnel source is the CPN node where an IP packet is encapsulated into a CPN packet.*

Definition 2 *A tunnel edge is a CPN node capable of de-encapsulating a tunnelled IP packet.*

Definition 3 *The de-encapsulation route entry is the IP forwarding rule that was used at the tunnel edge to de-encapsulate the IP packet. It is specified as a pair of the form <destination,destination-mask> (e.g. <192.168.1.0,255.255.255.0>).*

Tunnelling an IP packet through CPN requires a few steps (the full algorithm is shown in Figure 3.5). First, the CPN router has to find a tunnel edge capable of forwarding the IP packet along to its intended destination, which is accomplished through Smart Packets (steps 1-3). The delay imposed by this connection setup phase can be prohibitive if it must be done for every new IP destination encountered; we overcome this requirement through caching the *de-encapsulation route entries* used by previous connections. If we find a match, then we can

1. The network begins in a state where each router has no knowledge of global structure or the quality of any paths. In addition, the location of *tunnel edges* and their *de-encapsulation route entries* are unknown. Each router is only aware of who are its immediate neighbors in the network.
2. An IP packet enters the *tunnel source*. If an administrator has designated (using the IP forwarding table entries) that this packet should use CPN, it will be forwarded to the local CPN module.
3. It will then be encapsulated into a CPN packet where the fields are set as follows:
 - *Source address* of the CPN packet will be the tunnel source.
 - *Protocol type* (reward) will uniquely identify this as an encapsulated IP packet.
 - *Destination address* of the CPN packet is chosen according to the following algorithm:
 - (a) If the *tunnel source* has a CPN route in its DPRR to the IP destination, then the destination IP address will be used, along with the current route.
 - (b) Else, the *tunnel source* consults a table of *de-encapsulation route entries* to see if any other CPN nodes have route entries that could be used to route this packet. If such a node exists, then that CPN node address is chosen as the destination for the packet. This table is constructed from SACK messages (see 5).
 - (c) Else, the destination IP address will be used.

A flag is set in the header of the packet to indicate whether the destination of the CPN packet was chosen is a CPN node, or is the IP destination address.
4. The *tunnel source* then proceeds to use the regular CPN connection-establishment algorithm, sending SPs to discover routes to the destination. These SPs carry user data to reduce the setup costs of the path. If this optimisation were not used, it would take a full extra RTT for the SACK to reach the source, and a corresponding Dumb packet to carry the user data to the destination.
5. An SP reaching an eligible *tunnel edge* has the IP packet it carries de-encapsulated. As per the normal CPN process, a SACK is generated which contains copies of the SP's route and CM with two caveats:
 - The *de-encapsulation route entry* used by the node is copied into the SACK's CM.
 - The destination IP is appended to the end of the route list for the packet.
6. Upon reception of the Ack at the *tunnel source*, it can then successfully route DPs. It encapsulates IP packets into DPs, and sends them to the *tunnel edge*, where they are de-encapsulated and forwarded along. Smart Packets are also generated at random to refresh the snapshots of the network state, and learn new routes to the destination.

Figure 3.5: Algorithm used to tunnel an IP packet through a CPN network

search our local DPRR for the corresponding CPN node, and avoid the random exploration phase of Smart Packet routing.

Tunnelling example

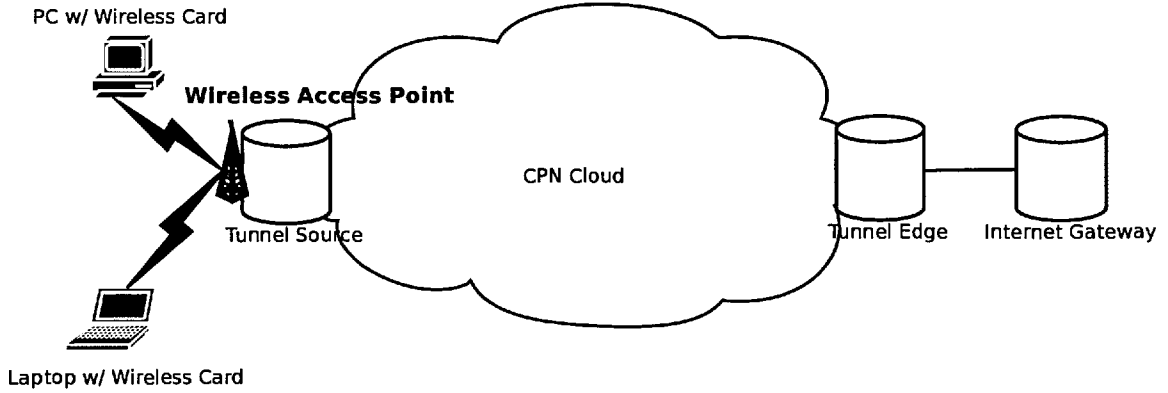


Figure 3.6: Using a CPN cloud to bridge two separate IP networks (the wireless network is one IP network, and the Internet is another, completely separate, network). Clients of the Wireless Access Point use CPN to access the Internet.

As an example of the usefulness of tunnelling, consider the scenario shown in Figure 3.6. Clients of the access point have no direct route to the Internet, but, by using the CPN cloud, they can locate a node which does have one. In a scenario like this, there may be more than one *tunnel edge* with different access links to the Internet; CPN will autonomously learn the best egress, and, if the network changes (e.g. access link goes down or degrades in quality), it will discover the next best one. Interconnecting IP networks in this manner allows us to extend the benefits of CPN routing to legacy clients and can extend their capabilities.

3.4 Summary

In this chapter, we have given an overview of the CPN algorithm, as well as how we have implemented it in our networking testbed. The implementations we have described are the ones with which we have conducted the experiments in the remainder of this thesis, and thus serve as an important foundation for understanding our results.

We concluded the chapter with a discussion of various methods available for inter-operating legacy networks (i.e. IP) with CPN. These extend the usefulness of CPN from a simple research tool to one that can interact with real-world applications. As one of the objectives of our research has been in enhancing the reality of our experiments and scenarios, we have used the described techniques frequently to use real applications for our experiments.

Chapter 4

Adaptation and Optimal Routing

The work in this chapter is an exploration into CPN's performance in a realistic network using real applications. These are the first experiments which provide a meaningful comparison of CPN to an existing routing protocol (OSPF). We show that the routes CPN computes are as good as those computed *a priori* using administrator-defined costs, and we show that CPN can quickly learn these routes. We also demonstrate the benefits of CPN's on-line learning by showing that it can quickly adapt to changes in the network environment. In a shortened form, some of this work has been published in (51).

4.1 Realistic Networks?

Before embarking on a comparative study, we first needed to come to grips with one of the key issues of our goal; namely, what is a *realistic* network, and how do we use the networking testbed that we have to construct a reasonable approximation of one? There is no definitive answer for all situations, but there exist good approximations which we will discuss. In answering this question, we are concerned with determining the *topology* of the network; in other words, we are looking to specify a *graph* G for our network, consisting of the set of vertices V , and edges E .

The question "What is a realistic network?" is not specific enough to be answered right away.

The fact is that there are many “realistic” networks, but not all of them are appropriate for our study. Networking textbooks sometimes classify networks in a hierarchical manner depending on their size and scope. This broadly breaks down into the classifications: Personal Area Networks (PAN); Local Area Networks (LAN); Metropolitan Area Networks (MAN); and Wide Area Networks (WAN). Because we are doing research into routing algorithms, we choose WANs which consist of large numbers of routers. However, we then need to ask what type of WAN can we use?

4.1.1 Wide-area Networks

The most basic WAN is typically referred to as an Autonomous System (AS). Each AS is typically owned by a single corporate entity and can consist of a few 10’s of routers to more than 10,000. In order to compute the routes to each subnetwork in the AS, an interior gateway protocol (IGP) is normally used such as Open Shortest Path First (OSPF). The routes computed by an IGP are typically those that minimize some cost function defined in terms of an administrator-determined metric. These metrics can be as simple or as complicated as the network calls for, ranging from shortest number of hops, to using traffic matrices computed based on historic observations of network usage.

Higher levels of abstraction are obtained when individual AS’s peer with each other in order to obtain greater connectivity, such as what has happened in the Internet. The standard protocol for computing these routes is the Border Gateway Protocol (BGP). Instead of cost metrics, BGP routes are typically based on more economic principles, where administrators define *policies* where they advertise to their peering partners which networks they are willing to carry traffic for.

Depending on the size and scope of a WAN, there may be more than one way its topology can be represented. The most detailed representation of a network is at the router level, where each vertex in the graph represents a single router, and the edges denote all available connections. As WANs grow in size and scope, the router-level representation may become too complex necessitating a higher level of abstraction, where routers in close geographic

proximity (in the same so-called Point of Presence or POP) are collapsed into a single graph vertex. As networks grow further in scope towards Internet-level scales, all routers belonging to a single Autonomous System (AS) are collapsed into a single vertex, and peerings form the graph edges. Here we discuss the pros and cons of each of these approaches, particularly focusing on the applicability to our own problem domain and our resource limitations.

Router-level graphs

The most specific representation of a WAN is a router-level graph, where each vertex represents a single router, and each edge represents a connection between routers. These edges typically have a *cost* associated with them which is determined by an administrator with knowledge of the network, and these costs, in turn, are used in the calculation of routes. Operators regard router-level maps of their networks as proprietary data, and are hesitant or flatly unwilling to openly publish this data, yet these serve as an invaluable resource to networking researchers wishing to ensure that their topology models resemble real networks.

Towards this end, various approaches have been taken to gather topology data. Some researchers establish a relationship with a network operator and are given access to sensitive data with the understanding that it will not be released to the public domain (as discussed later, we have taken such an approach). Another option is to gather maps of an ISP through measurement - one such study done in this manner was the Rocketfuel (96) project which mapped 10 large ISP networks using their publicly available BGP maps and router naming conventions to reduce the number of probes to give a complete topology. This data is publicly available, but, because of the size of the measured networks, it was infeasible to recreate these networks in our networking testbed. Also, as this data is based on indirect measurement, it is possible that errors could exist in the results since validation of these maps is difficult since the real topologies are confidential.

POP and AS-level graphs

In order to reduce the size and complexity of the graph, topologies representative of large-scale networks typically do not include individual routers; instead, the vertices in the graph are grouped into Points of Presence (POP), or, in the case of larger networks (such as the Internet), all routers belonging to the same AS are collected into the same graph vertex. Because of the way that these large networks are structured, these abstractions greatly reduce the number of vertices in the graph, while still retaining a great deal of information about available routing options at each point in the network.

Analysing the structure of large networks has been an active research area for many years, with many different data sets available, as well as tools able to artificially generate topologies. Some topology sets of ISPs have been released by large ISPs (e.g. (54)), or have been gathered at Internet-scale (e.g. (17)). A study in (54) looked at three different topology generators: GT-ITM (102), BRITe (77), and TIERS (101) testing how well they were able to generate topologies matching some real-world POP-level graphs with good results for all three tools. The generation of topologies representative of the Internet has also been looked at (e.g. (18)).

4.1.2 Resource Limitations

After the above discussion of different types of WAN structure, we were in a much better position to determine a topology to implement in our network, taking into account our resource limitations. The biggest limitation was the size of the topology; we had to choose one big enough that it was meaningful, but not so large that it exceeded the number of routers we had in our networking testbed. In addition to the network size we also had to ensure that the *degree* of the routers did not exceed that available number of network interfaces in our PCs.

One of the first limitations that we were subjected to was the number of routers in the topology. Since router in the topology corresponds to a PC in our networking testbed, then at some point we will run out of resources. In addition, large numbers of routers impose other costs in terms of maintenance and administration. Thus, we must have enough routers

to make the topology “realistic”, but at the same time limit ourselves to a manageable (and affordable) number.

In addition to topology size, the degree of the individual routers is an important consideration when recreating a network in our testbed. Each of our testbed PCs is limited in the maximum degree it can support; we have 2U rack-mount units which can only support up to 8 connections. Similarly, we have few stand-alone PCs which also have a maximum degree (albeit larger than the rack-mount units). This became an important factor in being able to recreate a real-world topology in the laboratory.

Pros and Cons of Emulation

On a side-note, we discuss the technique of emulation, which is an alternative (that we have not employed in our experiments) to having a one-to-one mapping of router to PC, where one physical PC is able to logically take on the responsibility of multiple routers. In recent years, as Moore’s Law has seen the speed of processors increase rapidly, software has been created (such as User Mode Linux) whereby a single computer can host multiple copies of an operating system, all of which appear (from outside the system) to be wholly separate entities. The potential for this technology is that with a limited number of physical resources, complex environments could be created, wholly in software.

However, there are a number of drawbacks to this approach which is why we have not yet taken advantage of it for our experiments. The biggest down-side to incorporating emulation is timing. Because emulation has multiple operating systems running on a single processing unit, there is inevitably the issue of scheduling and contention for resources which is non-existent when a single operating system is running. This not only introduces overhead, as possibly multiple system layers are introduced into the processing path of a network packet, it can also introduce problems for repeating experiments. As the measurements we take are typically measured in milliseconds, the interaction of multiple operating systems could vary them dramatically from experiment to experiment. This is an additional source of uncertainty that we wanted to avoid if possible. Another complication is the administrative

cost of configuring and maintaining a setup of virtual machines. Because we are dealing with a complex environment, this “hidden cost” becomes an additional constraint. If there were methods for automatically configuring a complex emulation environment it would make it a much more attractive solution.

4.1.3 Choosing a Topology: Receiving real topology data

In browsing some of the publicly available topology databases (54), we came across data gathered from the Swiss Education and Research Network (see Figure 4.1 for a geographic representation). In order to ascertain the date of this information, we contacted administrators of this network directly, who then provided us with an up-to-date snapshot (as of February 2006) of how all the 46 routers comprising their network were connected, along with OSPF costs (proportional to propagation delay) and bandwidth for each link (Figure 4.2). With the completeness of this data set, and because it can be fully represented in our networking testbed given our resource limitations, we have decided on this topology for our experiments.

In addition to satisfying our resource limitations, this topology also has a number of positive aspects from the point of view of routing protocol experimentation. First of all, because the topology represents a geographically distributed, heterogeneous network, there is a large deal of variability in terms of link costs, link speed, and node degree. This variability is a key component to “real-world” networks, and by subjecting our routing protocol to this environment, the conclusions which we will be able to draw will be able to generalize to other networks of similar scale and scope. As an aside, we also note that this topology has nodes with larger degree (maximum degree is 15 in this topology, which we achieve through the use of a desktop PC with 4 quad-port Ethernet cards) than any of the other topologies previously used in CPN experiments; this high degree corresponds to a high number of routing decisions and is an important test of CPN’s ability to scale to a real-world topology.

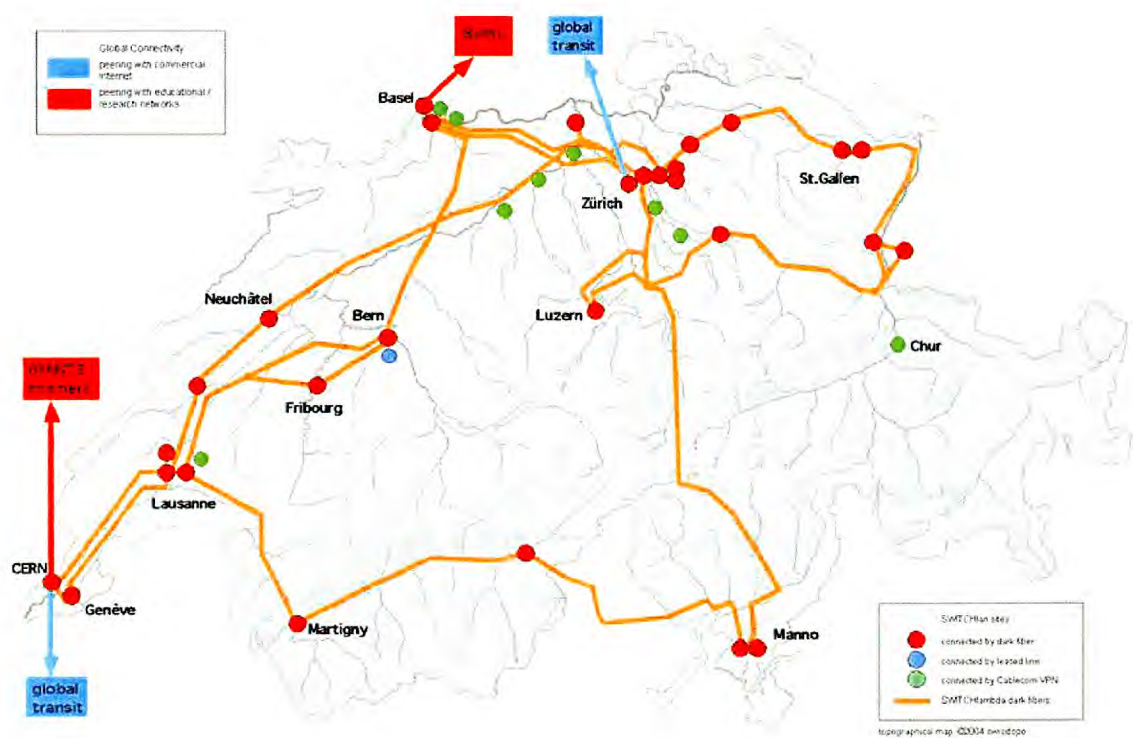


Figure 4.1: Geographic representation of the Swiss Education and Research Network that we use for our experiments.

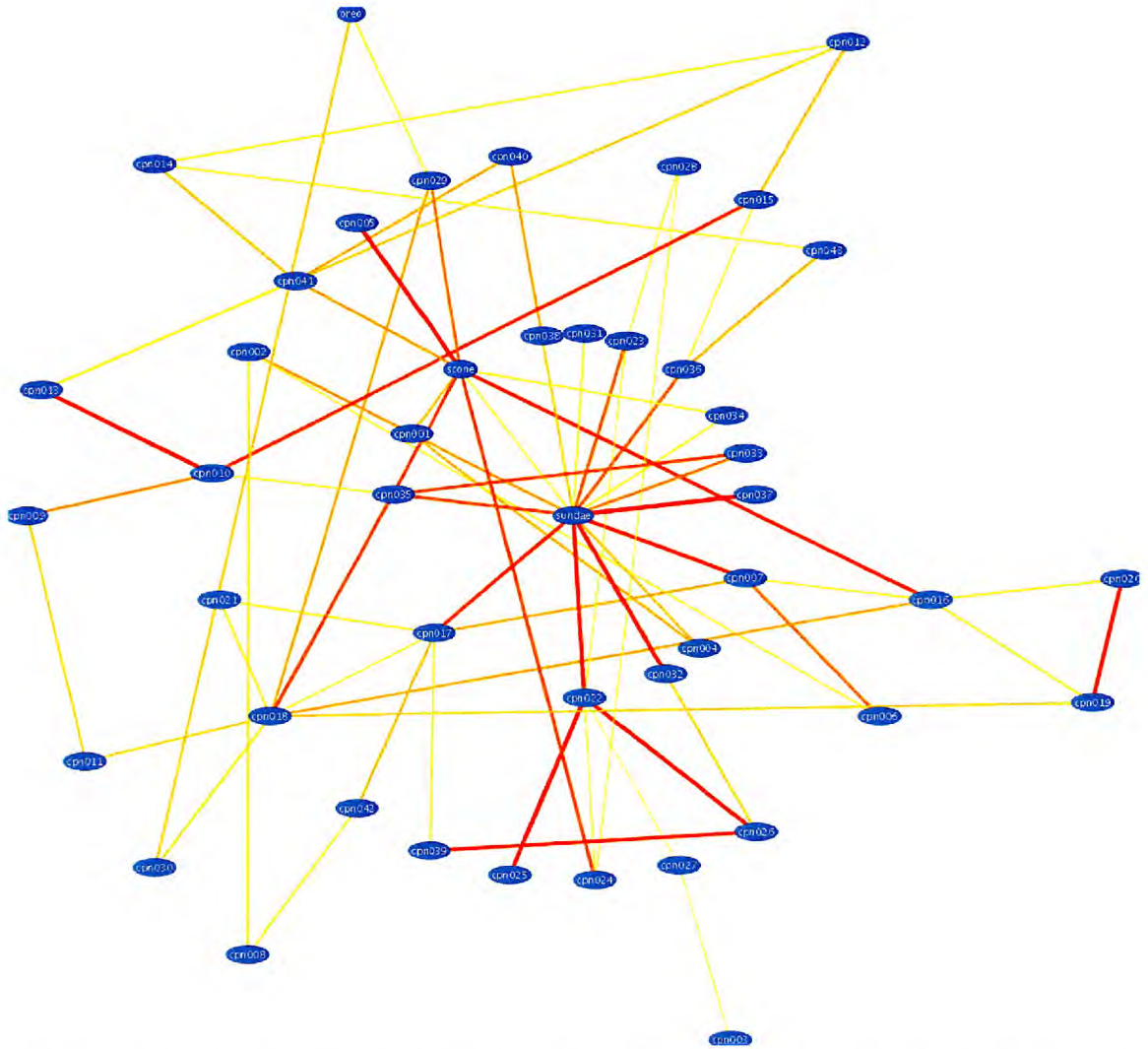


Figure 4.2: The 46-node testbed topology. Links between nodes are coloured and emphasized in terms of the delay of the link; lighter thinner lines are low-delay links, while darker, thicker ones denote higher delays.

4.1.4 Implementing the Topology

We have implemented this topology in its entirety using our networking testbed. Our testbed consists of standard Pentium IV-class PCs which run Linux 2.6.15 using the Debian Sarge distribution. Each PC contains one or more quad-10/100 Ethernet cards (manufactured by D-Link, model number DFE-580TX). Most of the PCs are 2U rack-mount units, with 2 available PCI slots, yielding a maximum of 8 connections. Because the topology has 2 routers with more than 8 connections (labelled as `sundae` and `scone` in Figure 4.2), we also included desktop-style cases in the topology which have more expansion slots available.

The method for establishing links between routers deserves some explanation. Connections were made between computers using point-to-point twisted pair crossover cables. An alternate approach which we considered, but ultimately chose not to use was to create the connections wholly in software using a Virtual LAN (VLAN) Switch. These switches allow all the cables to be connected to a single networking device, and creates logically separate network domains through software configuration. While this greatly improves the programmability of the network, making topology changes relatively painless, it comes at some additional costs. The first cost is the expense of the device itself; large VLAN switches cost thousands of pounds and implementing a large topology such as the one we want would require more than one switch. Additionally, these switches have the potential for introducing timing artifacts that are not present in a simple cable. Packets entering a switch must inevitably undergo processing which has the potential for introducing delays into forwarding that may vary from experiment to experiment. Using the same argument that we used above for emulation technologies, any variable delays that we can eliminate from our results improve our ability to draw meaningful conclusions from the results.

In addition to establishing the physical connections between routers, we also attempted to include the bandwidth and delay data present in the topology data we received (which is freely available from SwitchLAN to networking researchers). The bandwidths in the data set were either 1Gbps or 10Gbps - we scaled these values down by a factor of 100 in order to implement them using our available networking hardware. In order to implement the

physical-layer delays given in our data set (these range from 0-40ms) we have used Linux's NetEm module to introduce artificial delays on the output queue of each interface.

After the physical layer was properly configured, we needed to focus on getting the networking layer running. CPN has minimal configuration; it needs only to have its local address specified, and the interfaces to listen on. Getting IP routing was more complicated. We used the OSPF cost data (which is proportional to the artificial delays we configured above) and an OSPF routing daemon (quagga 0.99.3) to compute all the routes in the topology.

Finally, we comment about the management interface of our testbed which has been invaluable. In addition to the 4-port NIC in each router, it also contains an additional Ethernet port which is connected to a common switch. This allows each PC to exist on the same IP networking subnet, and makes common administrative tasks such as startup, configuration, and running experiments much easier. Because the management interface is completely separate from the routing interfaces, administrative traffic does not interfere with experimental traffic, reducing the potential for interference.

4.1.5 Validating our Configuration

We validated our setup by confirming that the routes calculated between sets of source-destination pairs used the shortest path as computed using the costs we were given. This ensured that the routes we were using were exactly the same as those on the real Swiss network.

Another more exhaustive method for validation would have been to obtain traffic matrices for each of the links, as well as the actual physical layer delays. Using this on our testbed would allow us to make measurements that would have timing data much more in-line with what we would hope to see were we to run CPN on the real Swiss network.

4.2 Using a First-person Shooter

After configuring a network topology, the next step became choosing a traffic source. While there are many types of applications that generate network traffic, we decided to focus on real-time applications, which, as we discussed in Chapter 2, have a number of QoS requirements. Furthermore, we would like to use an application that incorporates multiple traffic sources, as the interaction of multiple flows in a complex CPN network has not been looked at in an experimental context.

An application that meets these requirements is a multi-player networked game. This application has a client/server architecture where at regular intervals clients send action data to a server who then computes a new game state and sends that back to each client. For applications such as these, the QoS metric that most impacts them is their *ping* or round-trip time to the server, thus making them ideal candidates for CPN using its delay metric.

Further focusing our application choice, we needed to find an application that we would be able to run automatically from a script, and make repeatable measurements. With this in mind, we immediately wanted to avoid the use of any human players in our scenarios. Using real people would create logistical problems in terms of scheduling time for enough participants, and would also make it difficult to ensure that one experiment would be identical to another. For this reason, we wanted to find a game with the ability to include “bots” or computer-generated players powered by some form of AI engine to act as the players.

Meeting all of our requirements is the Quake II multi-player game. This game was originally released in 1997, and has since had its source-code released, which makes it an ideal candidate for our purposes. Because we are interested in scripting our experiments, and running them without any graphical or multi-media capabilities, we were able to modify the Quake II client code to create a null graphics and sound driver which, in essence, allowed us to create a client whose sole purpose is to generate and receive network traffic. On the server end, we were able to embed bot code which allows us to add a number of artificial players who ensure that there are always updates sent and received by each client during the session of play.

Finally, we were able to use the techniques described in Section 3.3.1 to allow Quake II use CPN as its network protocol. We analyzed various aspects of this modified Quake client, plotting an individual client's bandwidth usage (Figure 4.3), the server's bandwidth usage when hosting 10 clients, both with and without the CPN overhead (Figure 4.4), and also a cumulative density function (CDF) of packet sizes seen at the Quake II server (Figure 4.5).

In our experiments, we use a single server (cpn008 in Figure 4.2) with 10 clients. The clients were randomly selected from throughout the topology. Since the server has two ingress interfaces, in the case of one link becoming overloaded there will exist an alternate route in the network – the question is whether CPN will be able to find it (see Section 4.3.3).

Using a networked game has the additional bonus that, because it is very sensitive to network performance, it already measures statistics such as latency in the application code. Players typically monitor these so-called *ping* times, and will only play on a server with respectable (and consistent) pings, making them a good candidate for our measurement study to evaluate how various parameters affect CPN's ability to provide good-performing network routes.

Finally, we note that an additional goal of our evaluation was to produce results that were statistically significant. We wanted to ensure that each experiment was of sufficient length so as to gather meaningful data, so we settled on an experiment length of 15 minutes (because the traffic profile of 4.4 is very consistent, running the experiment for longer than this will not yield different results). Because we record the ping times of each client every second during the experiment, we gather 900 data points for each run. In addition, because there is a certain amount of randomness in each experiment (e.g. which path is used first, which paths are discovered by Smart packets, and in what order, etc.) we needed to repeat each one a number of times (we settled on 20 repetitions of each experiment). In examining the data we gathered, the variance was minimal (which is good because it ensured consistent delays for the clients) except under certain conditions where it is highlighted.

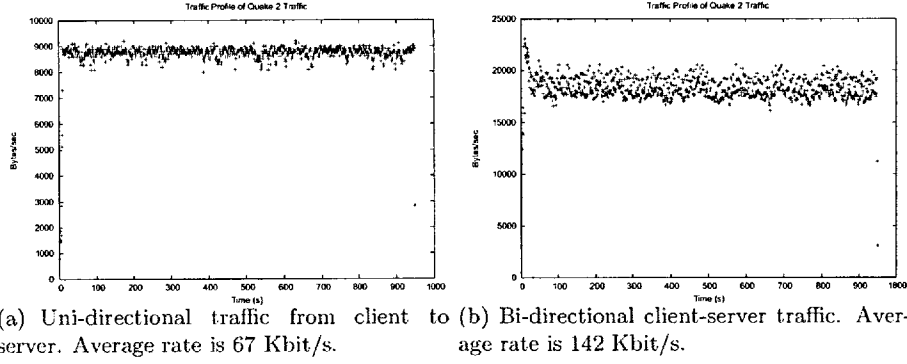


Figure 4.3: Quake 2 traffic profile at the client

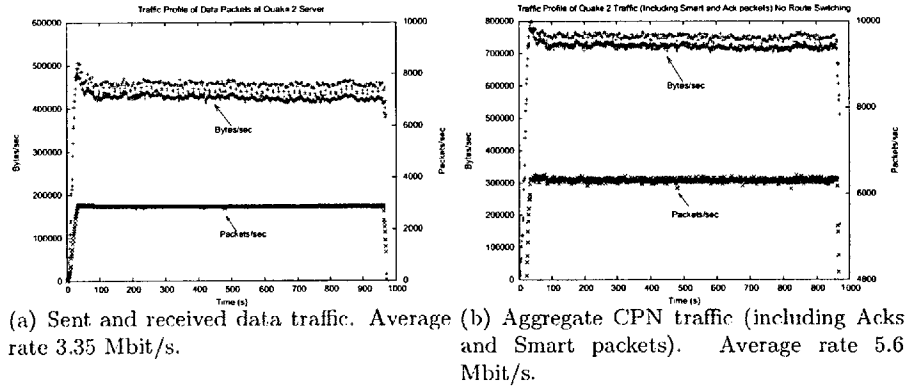


Figure 4.4: Quake 2 traffic profile at the server with 10 clients

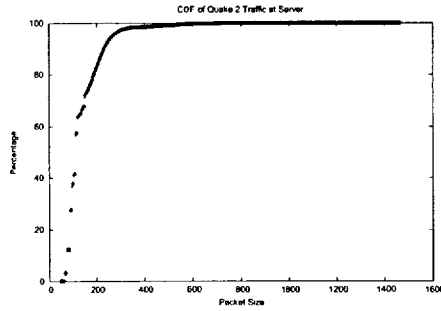


Figure 4.5: CDF of packet sizes at Quake II server. Average size 136 bytes.

4.3 Experiments

In this section, we present the results from three different sets of experiments that we conducted on our testbed. The first set compares the performance of CPN with the “sorting” optimization that we described in section 3.1.2. Following this experiment, we investigated the impact of varying the amount of exploration overhead on CPN’s ability to quickly discover the optimal path in the network. Our final set of experiments demonstrates the quickness with which CPN is able to react to a performance degradation along its optimal route.

4.3.1 Sorting vs No Sorting

When we first experimented with CPN’s ability to find the optimal route in the network, we noticed that it would sometimes route Dumb packet traffic along non-optimal paths for short periods of time before returning to the best path (Fig. 4.6). Upon closer investigation, we discovered that this was due to Smart packets which, with a small probability⁴ at each hop, would randomly choose their next hop as opposed to using the RNN-RL algorithm. While this is an integral component of the CPN algorithm because it enhances our ability to discover new paths, it was also harming the performance of data traffic that would use these new routes without considering how they compared to their current route. Thus, we implemented the optimization described in section 3.1.2.

The outcome of the optimization is shown in Fig. 4.6, where we can see it leads to a significant improvement. Once CPN has discovered the optimal route in the early stages of the experiment, it continues to use it whereas without the optimization, CPN subjects the data traffic to a number of other routes with worse delay performance. This optimal path has the same delay as the IP path. Throughout the remainder of this thesis, all experiments take place with this optimization enabled.

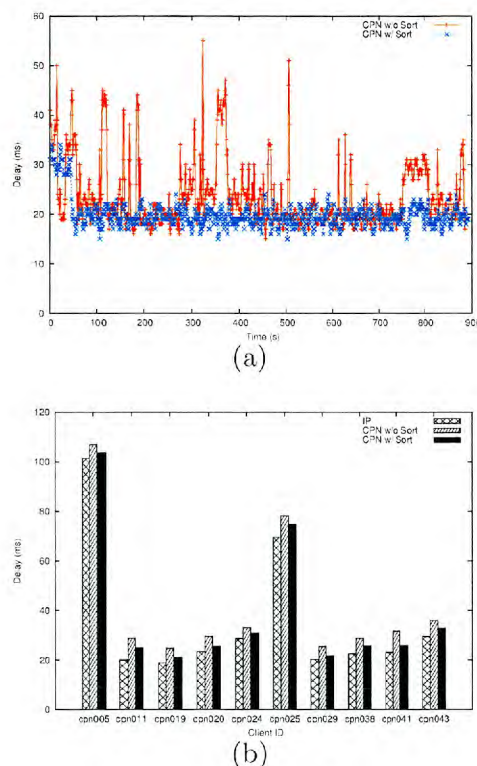


Figure 4.6: Comparing the effect of sorting. In (a), where we have plotted the measured delay during the experiment for one selected client, we can see the variance in the non-sorting case is much higher, as some of the Dumb packets use a route with higher delay before switching back to the optimal one. The results for every client are summarized in (b), where we see that the average delay is lower when using the sorting optimization. IP shows up as best, but this is due to the startup costs of CPN’s random exploration phase. As soon as CPN discovers the best route, it has the same delay as the IP path.

4.3.2 Discovering Optimal Routes

In order to demonstrate CPN’s ability to quickly find the optimal route in the network, we compared its performance with that of IP using OSPF to determine its routes. Because the cost of each link is proportional to its delay, OSPF routing converges to the minimal delay path, giving us a baseline for comparison.

Our experiments were conducted for different percentages of Smart packets⁵, which controls

⁴In our experiments, the probability of randomly choosing a next hop is 0.05.

⁵While we refer to it as a percentage, it is more accurately described as the *probability* that, when sending a DP, that we also send an SP.

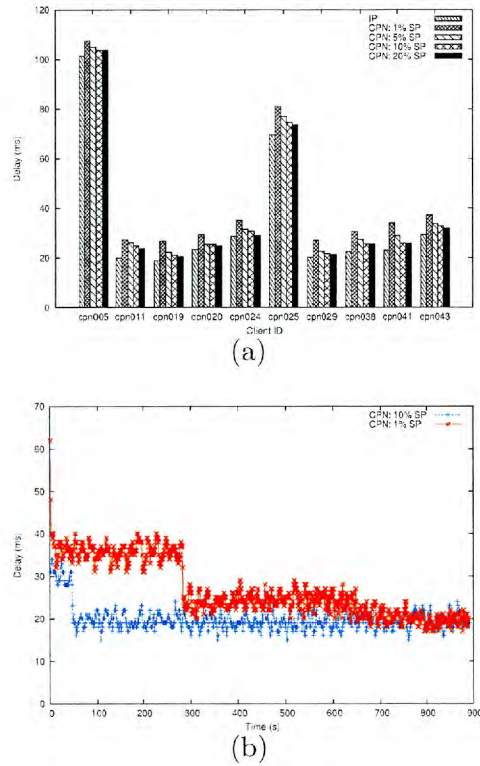


Figure 4.7: IP vs. CPN for different percentages of Smart packets. In (a), we can see that the average delay for CPN is very close to optimal. Also, as the number of SPs increases, the delay decreases. The reason the average is a bit higher can be seen in (b) where at the beginning of the experiment, CPN has not yet found the optimal route, increasing the overall average. In addition, the impact of the SP percentage on the rate of discovering the optimal route is highlighted.

the overhead of network exploration. Recall from Chapter 3 that after CPN has discovered at least one route, the only way for this route to be updated is for a Smart packet to be generated, it must locate the destination, its SACK must be received, and it must represent a path which has better quality than the active one. Thus, the rate at which Smart packets are generated strongly influences the speed with which CPN is able to locate the optimal route in the network since it starts with no knowledge of the network whatsoever. However, the percentage cannot be too high – sending too many Smart packets not only increases the amount of network overhead, it may also lead to performance degradation as they cause increased congestion and require more computation by network routers.

The results of this experiment can be seen in Figure 4.7. Looking at them, we can see that CPN can always find the optimal route in the network – it just takes a longer time when the exploration overhead is low (Figure 4.7.b). We can also see that the difference in delay performance between 10% and 20% SPs is minimal for many of the clients, leading us to decide to use the former for the remaining experiments.

4.3.3 CPN's Response to Network Dynamics

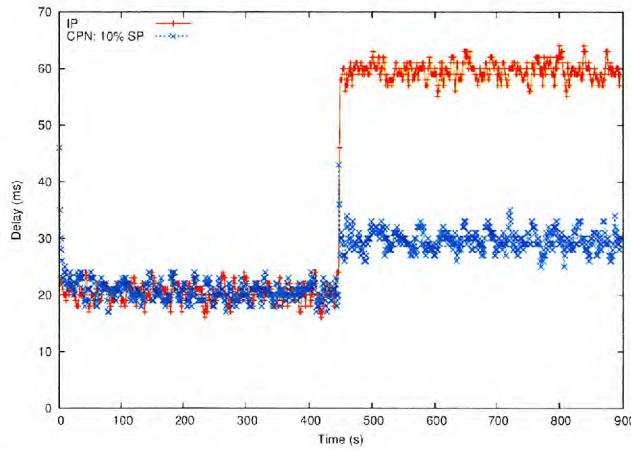


Figure 4.8: How CPN reacts to a disturbance along the optimal route. At time $t = 450s$, the delay along one link in the optimal route is increased by 40ms. Within 1 second, CPN has switched to another route.

In our final experiment, we wanted to see how quickly CPN could adapt to a sudden change along the optimal route. We decided to test this by manually increasing the delay along one of the links in the network (the link between `cpn017` and `cpn042` in Figure 4.2) midway through the experiment (at time $t = 450s$). The results of this experiment are shown in Figure 4.8, where CPN was able to react to the change within 1 second, and find a route around the hotspot.^a

This is one of CPN's key advantages over a routing protocol like OSPF. While OSPF can adapt to changes in the network, usually this requires manual intervention to alter the administrator-determined costs of a link, and happens a much longer time-scales than with CPN, which is able to automatically adapt within seconds to any changes in the network.

4.4 Future Work

The work in this chapter has gone a long way towards exploring the performance of multiple CPN flows. However, there are still areas rich for exploration in this space, especially in the area of dynamics.

The network configuration that we used for our experiments was static the whole time, which makes a good initial scenario, but it is not entirely realistic. Two factors could improve the realism: background traffic and switching nodes off and on during the experiment. Adding background traffic to the network would allow some links to be more congested than others, and allow us to better study how the overhead of CPN impacts the capacity of the network. Additionally, altering the network configuration during an experiment would allow us to better look even more in-depth into CPN's adaptability, one of its key advantages over IP.

4.5 Summary

In this chapter, we have explored how CPN performs in realistic network scenarios, showing the effect of various parameters over CPN's ability to discover and utilize optimal paths in the network. We began with a discussion of what it means to have a realistic network, finally settling on up-to-date data provided by the Swiss Research and Education Network. In addition to the network topology, the choice of network application is also important – after weighing all the options, we decided to use the first-person shooter Quake II.

Our experiments have demonstrated a number of positive qualities of CPN. One of the most important is that even with no *a priori* knowledge of a network, it can still discover and utilize the optimal route, even when the network conditions change. In addition, we showed that the exploration overhead is directly proportional to how quickly the optimal route in the network is discovered. Finally, we demonstrated one of the key benefits of CPN routing over OSPF in that it can adapt to a change in path quality on the order of seconds.

These were the first published experiments (51) to test CPN's performance with a realistic

network topology and a real network application with multiple flows. When we ran these experiments, we noticed that the individual flows interacted with each other, accounting for many of the route switches. In fact, we noticed that there was the potential for *route oscillations* where a client frequently switched its active route back and forth. This led us on to our next chapter, where we examine these oscillations, and their impact on CPN's performance.

Chapter 5

Eliminating Routing Oscillations

5.1 Introduction

Routing oscillations are a networking phenomenon that have been observed since some of the earliest packet-switched networks (such as the ARPANet) where they caused performance to suffer under medium to high load (62). These oscillations were due to the use of a load-sensitive routing metric, which has led to a commonly taken-for-granted assumption among researchers that routing protocols should ensure that oscillations do not occur, and furthermore, load-sensitive metrics should not be used in such a way that they could occur. We set out to test this assumption using an adaptive routing protocol, with some surprising results which question whether it still holds.

In this chapter, we challenge the assumption that routing oscillations always result in poor network performance. Using experiments conducted with CPN on a networking testbed configured with real-world topology data, we show that increased switching does not always result in poor performance, and can actually result in improved QoS for network flows. While an increase in route-switching results in a corresponding increase in packet desequencing ⁶, we demonstrate two factors which function to trade-off between network performance (i.e. delay

⁶We define this term later, but it refers to the arrival of packets at the destination in a different order than they were sent.

and loss) and desequencing.

We begin in Section 5.2 by overviewing the relevant literature about routing oscillations, including the relevant studies which have concluded that they invariably lead to poor performance. Then, we demonstrate in Section 5.3 that CPN suffers from routing oscillations due to two causes outlined in (32), namely the *self-load effect* where the impact of a flow on a path is not taken into account when probing, which can lead to permanent oscillations, and secondly when the measurement periods of multiple flows overlap causing them to switch in groups chasing perceived areas of better performance. While we do observe routing oscillations in CPN, in our larger scale experiments the results seem to contradict some of the conclusions of (32) when the background traffic is stable. In Section 5.4, we show that CPN in effect can *benefit* from frequent route switching, yielding its best performance when it is allowed to switch whenever it finds a better-performing route. This observation seems to indicate that myopic and selfish behaviour of users in a network, leading to frequent “changes of mind” can under certain circumstances result in better performance for all parties concerned. This affirmation needs to be tempered when oscillations can have other side effects, such as packet desequencing for flows that are highly sensitive to sequence, such as real-time video or voice, or even TCP traffic (Section 5.4.3).

5.2 Related Work

Routing oscillations are hardly a new phenomenon, and there have been many contributions to the subject in the field of networks. The ARPANET, one of the predecessors of the modern Internet was susceptible to routing oscillations under heavy load (62) which led to poor performance. The effects of Internet routing instability were discussed in (70; 71) which summarized their impact by saying that “Overall, instability has three primary effects: increased packet loss to unstable destination, delays in the time for network convergence, and additional overhead (memory, CPU, etc.) within the Internet infrastructure.” A discussion of the negative impact of synchronization of periodic routing messages is found in (31) who show that even if traffic sources are initially not synchronized, they can abruptly become synchronized.

In (91) route instability is discussed in the presence of network congestion, causing increased packet loss and latency.

The negative impact upon TCP performance of disturbances at the routing layer is examined in (88) which considers both the case of routing oscillations due to the self-load effect and link failures. TCP is affected by these events because of how it responds to asymmetric paths (i.e. the path that its data packets take is different from those of its Acks), and out-of-order packet delivery.

Routing oscillations in overlay networks are analyzed in (61). They deal with the negative consequences of synchronization of multiple overlay networks, and analytically determine an upper bound for the length of a period of synchronization.

5.3 Oscillations in CPN

Routing oscillations in Intelligent Route Control (IRC) systems have been explored in (32) through simulation. They define IRC systems as aiming “to optimize the cost and performance of outgoing traffic, based on measurement-driven dynamic path switching techniques”. CPN also carries out path switching as needed for QoS purposes based on dynamic measurement and control, and (as previously described) it constantly explores the network state using SPs.

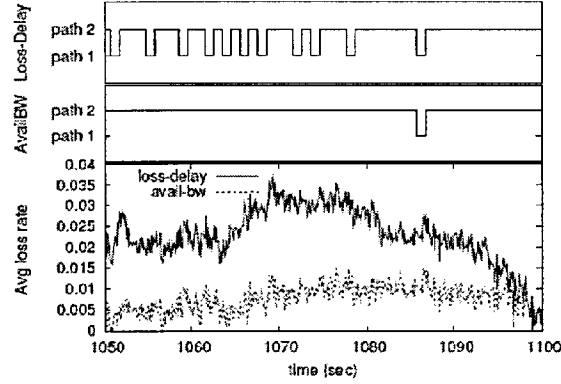
In (32), two causes of the oscillations are identified. The first is what they term the *self-load* effect, where the impact of a flow on the metric that is measured is not taken into account in the probing process so that once a path whose load appears to be light is actually used, the resulting load is significantly higher than the one that was previously observed. This first phenomenon has also been studied in (4) with respect to the use of CPN in admission control algorithms. Of more difficulty is the second cause they outline: that routers’ measurement windows can overlap, leading to persistent switching as flows interfere with each other, preventing the network from stabilizing, and causing uneven utilization and poor QoS. In this section we provide measurements to show that CPN can also suffer from both of these causes of path oscillations.

A solution proposed in (32) is to use knowledge of a path’s available bandwidth to help reduce or eliminate oscillations. This is because, if an estimate of a flow’s bandwidth and the available bandwidth of a path are known, then the impact of switching a flow to a path can be estimated before switching to the path. However, very often the measuring of available bandwidth occurs at longer timescales than the routing process itself. A recent comparison of tools that may be used in this process (93) found that the two most accurate tools had running times of $5.5s$ (*pathchirp*) and between 7 and $22s$ (*pathload*). Another more recent study (92) concluded that a time of $5s$ was needed for accurate bandwidth estimation. These times indicate that it would be difficult to satisfy the need to have routing intervals that are of similar or greater length than the measurement time (i.e. $T_m = T_r$ from (32) when $T_r = 1s$).

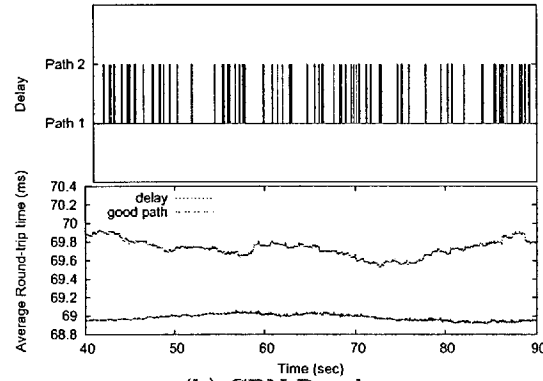
5.3.1 CPN and the self-load effect

We have examined the extent to which CPN suffers from the self-load effect and have tried to match as closely as possible the experiment described in (32) (Section III.A). We have set up a simple 4-router square topology with one source S and one destination D on opposite corners (i.e. there are two paths to the destination, p_1 and p_2). Four flows were started where three are statically routed using the Internet Protocol (IP), with two along path p_2 and one on path p_1 . The fourth flow uses CPN with delay as its routing metric so that it generates SPs which are constantly searching for paths with lower end-to-end delay. The traffic generated by each flow is constant bit-rate (CBR) with rate equal to 25% of the capacity of a path (2.5 Mbps).

In order to compare the effect of routing oscillations on the performance of the CPN flow, we used two different configurations. In the first, CPN was configured to optimize delay, while in the second, we disabled path switching and forced CPN to stay on the optimal path p_2 (labelled *good path*). The results are shown in Figure 5.1, where the round-trip packet delay is a sliding window average over 30s of measurements per packet. While the time-scales of the two data sets differ by an order of magnitude (e.g. CPN switches paths on the order of hundreds of milliseconds, while their simulations take seconds), we clearly see that CPN also suffers from the self-load effect, to the detriment of performance.



(a) Figure from (32)



(b) CPN Results

Figure 5.1: **The self-load effect** Path switching (above graph) shows that oscillations are occurring, and the resulting QoS (loss in the above set of measurements, and delay in the bottom set) shows the negative impact that they can have on performance.

5.3.2 Synchronization-caused Oscillations

In addition to the self-load effect, routing oscillations can be caused as a result of multiple flows in the network making synchronized decisions or decisions are made quasi-concurrently.

To determine whether CPN can also suffer from oscillations due to concurrent decisions, we conducted an experiment using 10 CPN flows with the topology shown in Figure 5.2. We artificially extended the measurement period within CPN to be 1s and started each of the flows approximately 0.1s apart so that their initial, and presumably subsequent, measurement periods would overlap. In addition to the CPN flows there are also 10 IP flows (5 per path) which function as background traffic. Oscillations caused by synchronization are shown in

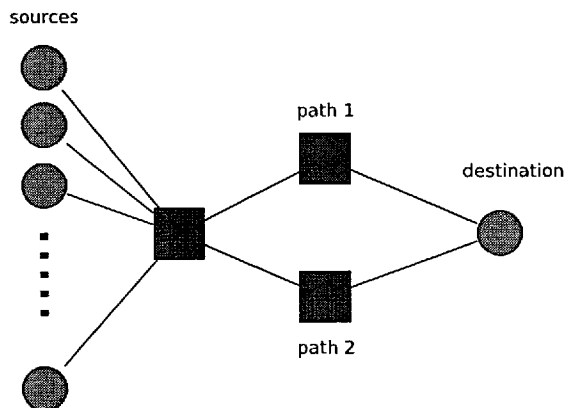


Figure 5.2: Topology for the experiments in Section 5.3.2. There are 10 CBR sources all with the same destination.

Figure 5.3. Their period is close to $2s$, matching well the simulations in (32) where the decision time is also $1s$. Of course, these CPN oscillations are indeed somewhat artificial because the observation and decision times in CPN tend to be much shorter (of the order of 50 to $100ms$) and this kind of synchronous behaviour may be difficult to observe in practice⁷.

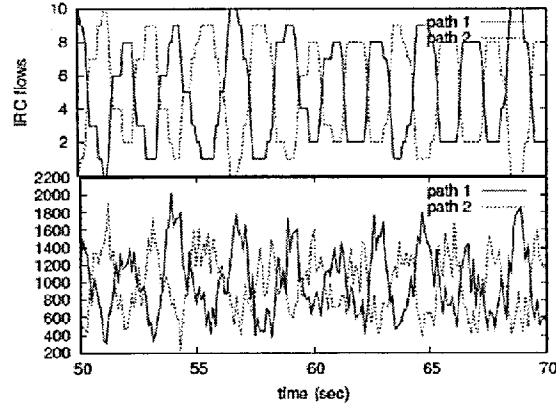
5.4 Oscillations under stationary load

Now we have shown that CPN can suffer from similar oscillations as have been seen in other adaptive networks, we wanted to scale up our analysis and conduct experiments using a realistic environment where routing oscillations would be observed and their impact on the performance of a large number of flows could be observed. In this section we experimentally investigate the assumption that oscillations result in poor performance, with surprising results.

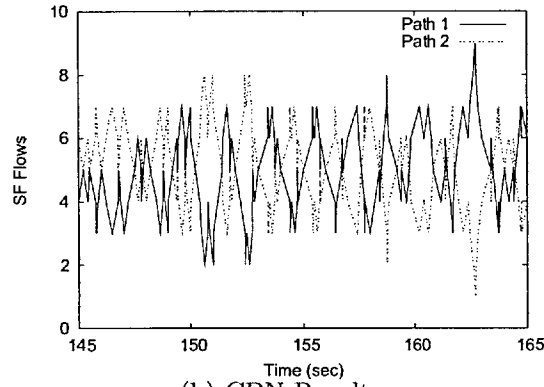
5.4.1 Experimental setup

Our networking testbed consists of 46 Pentium IV-class machines, each equipped with one (or more) 4-port 10/100 Ethernet interfaces. The Operating System is Linux 2.6.15, where CPN is implemented as a loadable kernel module. Each link is full-duplex, and is configured

⁷These oscillations *do exist*, they are just more difficult to demonstrate.



(a) Figure from (32)



(b) CPN Results

Figure 5.3: Oscillations due to Synchronization

to run at 10Mb/s. The topology we use is based on information we received about the Swiss Education & Research Network shown previously in Figure 4.2.

Routes at the source

In the original CPN algorithm, a route brought back by the ACK would be immediately used, as it contains the most up-to-date snapshot of the network state. However, (51) showed that this can lead to poor data packet performance due to the random component of SP exploration. Thus we now switch routes only when the reward of the new route is greater than that of the current route. ACK packets are also generated at the destination when a DP reaches it (in which case it is called a DACK), and these can also serve to refresh the QoS

information related to the paths that are stored at the source.

To complement this strategy, we incorporate two additional parameters which influence when the source switches to a new route. The first of these enhancements (also described in (32) though not explicitly used to control oscillations) we call a *reward threshold*, where we specify that a newly discovered route must be better than the active route by a fixed percentage in order for a switch to occur. This is a similar approach to the one used in (48) where a group of routes within a given quality margin were all considered as viable routing options, and a round-robin policy was used for selecting a route. The second control we use is a *fixed switching probability* (FSP) which was also used in (32). When a new route is discovered which is better by the reward threshold than the active one then with some probability it will cause a switch. As an example, in traditional CPN the FSP is equal to 1, whereas if no switching were allowed, the FSP would be 0.

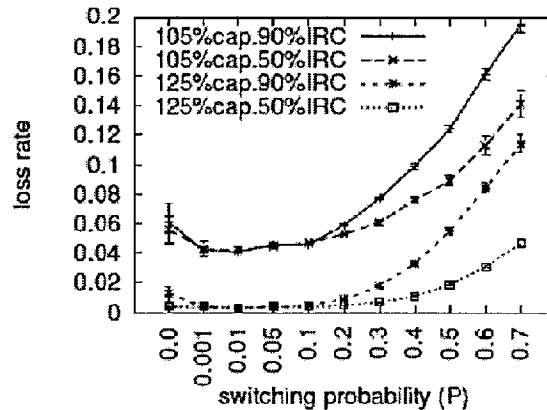
Parameter	Value
Flows	24
Rate per flow	1.66 Mb/s
Ingress Links at Destination	4
Available Ingress Bandwidth	40 Mb/s
Smart Packet %	10%
Experiment Length	15 minutes
Reward Threshold	12.5%

Figure 5.4: Experimental parameters

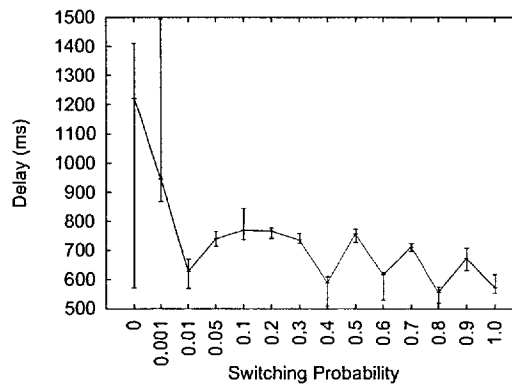
Our experiments use 24 constant bit-rate flows, each of 1.66Mb/s , which makes for (just above) 40Mb/s of application data (not including SP and ACK overhead). This yields similar parameters to the 105% of capacity case studied in (32)⁸. Each client sends traffic to the same destination, which has 4 ingress interfaces that provide it with 40Mb/s of available incoming bandwidth. For each Dumb packet sent, the probability of generating a Smart packet is 0.1. In order to estimate the impact of unresponsive traffic, we configured this probability to be 0 for two of the flows (this matches the 90% case in (32)). Each CPN flow is configured to optimize delay and each individual experiment lasted 15 minutes.

⁸This amount of traffic can be injected into the network given its size and complexity; each source injects a small amount with the same destination which eventually arrives to overwhelm it. It is under these severe circumstances that we want to evaluate CPN's behaviour and performance.

5.4.2 Impact of Switching on Performance



(a) Simulated performance when varying the switching probability (from (32))



(b) CPN's performance when increasing switching. The average delay of each client is computed individually, and the median is plotted in the graph. Error bars indicate the 1st and 3rd quartile.

Figure 5.5: Comparing simulated (above) and experimental (below) performance. While in the simulation increasing the switching probability results in poorer performance, the opposite trend is observed in our experiments. The metrics in the graphs are different (yet both are valid for estimating performance, and are entwined) – we have chosen to present delay as this was what CPN was told to optimise in the experiments.

In order to study the extent to which CPN's performance is affected by routing oscillations, we began by experimenting with the fixed switching probability (FSP) parameter. The most striking difference between our experiments and the simulation results of (32) is shown in Figure 5.5. In the simulations, as the switching probability increases, the performance of the system quickly decreases. The exact opposite trend is observed in CPN. When the switching probability is zero (i.e. the path of every flow is chosen at random at the beginning of the

experiment, and then never switched), the delay is at its highest. As CPN is allowed more freedom to switch routes, it is able to better optimize its performance, yielding a median value half that obtained without switching.

Differences between Experiments and Simulations

Despite our efforts to match the experimental conditions to that of the simulator described in (32), we highlight that the following items that could account for the differences we observe:

1. CPN probing occurs much more rapidly than what is reported in the simulations. Indeed, any information gathering scheme must sample the environment at a high rate in order to obtain an accurate picture of what is going on. In the simulations, one measurement period consists of 10 samples each of which lasts for $100ms$, and an average over the 10 samples is used as an estimate of the path's performance. Our system uses exponential averaging over a relatively short memory span (8 samples on average) at each hop in the RL algorithm, and samples each node of the network roughly every $1ms$, which is a factor of 100 times faster.
2. The reinforcement learning algorithm used by CPN differs from the straightforward rules used by the IRC flows in (32) – the sophistication of the RL algorithm may allow CPN to benefit from increased switching.
3. CPN uses delay as its QoS metric in order to adapt towards better performance, whereas the simulation uses bandwidth.
4. Finally, the simulations in (32) only include the effect of the ingress links at the destination node, while our measurements cover a more realistic and complex network topology which includes the effects of possible bottlenecks and capacity sharing throughout the network.

In addition to these more conceptual differences, there are also some other differences at the experimental level that may impact our results:

1. It uses a *Fractional Gaussian Noise* (FGN) model for traffic. Each 100ms, the rate of a flow changes according to this model which attempts to reproduce the self-similar traffic that may be seen in a backbone network. We do not have a FGN traffic generator, and used constant bit-rate traffic instead. While at first this seems like a large difference, we feel that if the simulation's results were only due to this complicated traffic model then it is surely not a portable result.
2. It simulates 100 sources each sending out one flow, while we conduct measurements on the test-bed with 24 connections. This is due to resource limitations.

While these 2 factors could impact our results, it is our feeling that they are not enough to account for the dramatically different performance trends that we have observed.

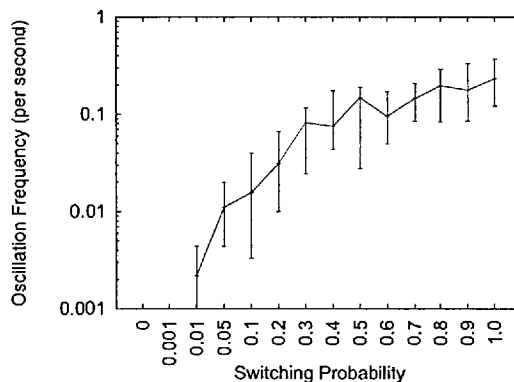


Figure 5.6: Oscillation Frequency as a function of FSP

We wanted to confirm that the increase in the switching probability corresponded to an increase in not only switching, but also oscillations. We defined an oscillation as a special pattern of route switching where a route is used, then another, then back to the original route. For instance, if route *A* is used, followed by route *B*, and then by route *A* again, we define that as a single oscillation. The rate of these occurrences over the experiment length is plotted in Figure 5.6. As the switching probability increases so does the rate at which the routes oscillate.

When taking Figures 5.5 and 5.6 together, we can see some interesting relationships. The difference in performance between probabilities 0.01 and 1 is not very dramatic, yet their rate

of oscillation differs by a factor of 100. Thus, our results are not indicating that oscillations are *necessary* for good performance – rather we are saying that they do not degrade performance in our network.

5.4.3 Packet Desequencing

One of the disadvantages of frequent path switching is that it can cause packets to arrive at the destination in a different order from which they were generated. This desequencing⁹ negatively impacts many traffic types' performance. The negative impact of reordering on a TCP connection is discussed at length in (9), and it is an on-going research activity to design a TCP implementation that can withstand desequencing without suffering a performance penalty (11). It is also well-known that real-time traffic such as Voice over IP needs its packets to be received in-order, or else they will be buffered, increasing the memory and processing requirements at the destination, and, in cases of excessive desequencing, it leads to dropped packets and poor performance.

Because of the negative impact on application performance, we measured the amount of desequencing as a result of increasing the switching probability. We make the assumption that our client streams each represent a single flow of real-time traffic. While in our traces, the application has not used a monotonically increasing sequence number, we can deduce the order in which each packet was generated by looking at the *id* field, which is the time-stamp at the source when the packet was originally sent. By considering the sequence of received *ids*, we can reconstruct the order in which the packets were generated, and, in turn, the amount to which it became desequenced by the network.

In order to quantify the level of desequencing, and in keeping with our focus on real-time traffic, our metric uses a *reordering buffer* that acts to store out-of-order packets. When an out-of-order packet arrives at the destination¹⁰, it is put into a buffer until it can be freed by

⁹This has also been referred to as *packet reordering* and *out-of-sequence packets* in the literature. We use the term *desequencing* to reflect the fact that we are measuring the degree to which the arriving packets are out-of-order.

¹⁰Any arriving packet having a sequence number greater than the expected packet is out-of-order.

the arrival of the expected sequence number. If an attempt is made to add an arriving packet to a full buffer, then it is assumed that the expected packet has been lost, and the packet with the lowest sequence number in the reorder buffer becomes the new expected packet. This acts to timeout packets which arrive excessively late. Packets arriving with sequence number less than the expected one are simply dropped (i.e they are very late). The reorder buffer has the ability to smooth out small amounts of desequencing, but, under higher levels it can lead to packet drops. This approach is similar to the Reorder Buffer Density metric proposed in (7), except we are concerned with the number of packets dropped by the buffer - not its size distribution.

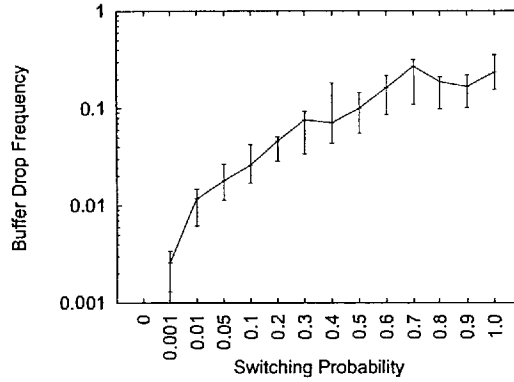


Figure 5.7: Impact of switching on desequencing. The number of packets which would be dropped by a reorder buffer increases with the switching rate.

For a reorder buffer of size 10, we obtained the drop frequencies in Figure 5.7. We use this buffer size as a reflection of the real-time nature of the traffic – larger buffer sizes would mean that even if a very late packet were successfully delivered to an application, its information would no longer be of value. As the switching frequency increases, the drop rate quickly exceeds the tolerable limits of many applications.

5.4.4 Reward Threshold

So far, we have demonstrated the impact that the switching probability has on performance, both in terms of delay and desequencing. Another factor which has a large impact on CPN's ability to select the best-performing routes is the reward threshold which we introduced in

Section 5.4.1 (which we refer to simply as *threshold* for the rest of the thesis), which is how much better the estimated reward for a newly discovered route must be than the active route in order to result in a route switch. We express the threshold in terms of a percentage; thus, the scenario where a new route would have to be twice as good as the current route corresponds to a threshold value of 100%. Here, we discuss the impact that this threshold has both on performance and oscillations.

The threshold serves as a parameter which can be tuned to make CPN more or less sensitive to improvements in route quality. If the value is too low, any improvement in route quality, no matter how small, will result in a route change. This can lead to unnecessary route switches due to, for instance, the *self-load effect* where the Smart packets of a flow do not impose the entire load of the flow on a path, thereby seeing improved levels of QoS that are not experienced by the flow after it switches, subsequently leading to further switches and high rates of oscillation. Similarly, if the threshold is too high, CPN will not be sensitive enough and will not take advantage of routes which offer improved performance. Thus reduced switching may also lead to reduced performance. Both of these phenomenon are observed in Figure 5.8. There appears to be an optimal value of the relative threshold, in this case around 14.3% of the current value, which provides better performance than with greater or smaller values.

When Figure 5.8 is considered along with the rate of oscillations in Figure 5.9 we can confirm our analysis above, but also draw a stronger relationship between oscillations and performance. When the threshold is 0 the oscillation rate is nearly 2 times per second, a factor of 10 increase over all the other switching thresholds. Similarly, at the highest threshold value, the oscillation rate is at its lowest. It is when we analyse these two graphs in tandem however, that we see some interesting trends. For oscillation rates that are very nearly identical, we see significant performance differences. This tells us that the rate of oscillation is not by itself enough to impact performance, and that there are other factors at work, corroborating our analysis of the results in Section 5.4.2.

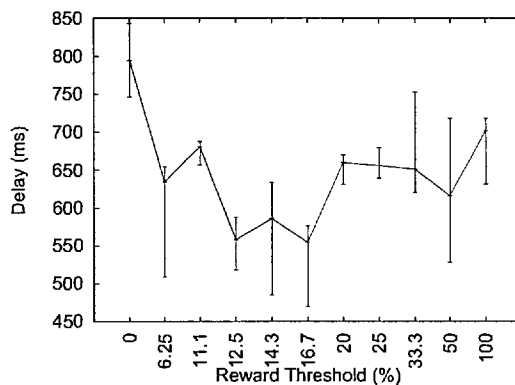


Figure 5.8: Impact of reward threshold on delay

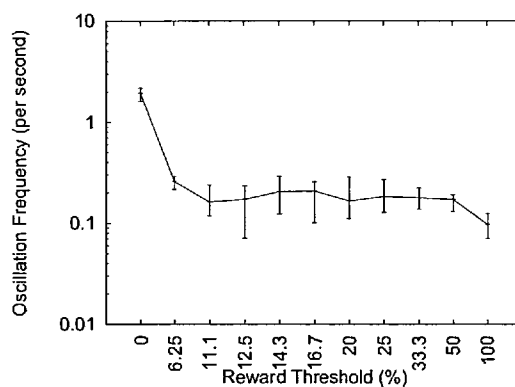


Figure 5.9: Reward threshold's impact on oscillation rate

5.5 Future Work

All of the results that we have presented in this chapter have been experimental in nature, demonstrating that oscillations do not always lead to poor performance. One way to enrich this work would be to examine the underlying causes of oscillations in a mathematical context with the idea of providing a clear explanation of the processes that bring forth our results.

One way to begin this analysis is to examine the reasons that CPN switches routes, namely that it senses that another path will provide better quality than the current one. Taking the parameters that we have introduced such as the switching probability and the reward threshold gives us more of a framework for understanding how often and how much better a given route must be in order to cause a switch. Taking these ideas even further, we can look

at routing oscillations as unnecessary switching, where there is no net gain made by switching to one route, and then back to the previous one. It is our hope that by looking at these issues in a theoretical context we can better understand the reason and impact of routing oscillations.

5.6 Summary

This chapter has examined routing oscillations and their impact on performance. We first confirmed that CPN indeed suffers from routing oscillations as a result of different factors, similar to other real networks and to results which have been observed by others in simulation experiments. We then turned our attention to studying the impact of routing oscillations on performance. Our measurements indicated that the CPN routing protocol provided somewhat different results from those that we would have expected based on long-held views (dating back to the ARPANet), that path adaptation will lead to oscillations and that this will result in poor performance under medium to high load. Our results indicate that routing oscillations do not severely degrade performance as would be expected; rather we show that even when they are present we can still obtain high performance.

We also examined how oscillations can be controlled in the network, and tested two different parameters which largely impact the rate at which oscillations are observed. In particular we studied the use of probabilistic path switching can be used both to make path switching more asynchronous, and to vary the rate at which switching decisions are made. We also examined the value of introducing a decision threshold which will only allow path switching if the gain expected from switching exceeds a certain minimal value. Both of these controls schemes are easy to implement, and provide an effective way to limit oscillations and their negative consequences.

More generally, we feel that this work brings into question whether some long-held assumptions in the networking community regarding the viability of load-sensitive routing metrics are applicable for new classes of routing protocols which are based on self-monitoring and

adaptation such as CPN.

Chapter 6

Loss and Delay-based Routing

In the previous chapter, we looked at how multiple flows interact in CPN to cause routing oscillations, and explored to what extent these actually impact performance. In this chapter, we consider a single flow which wishes to find an optimal path in terms of multiple objectives. Namely, we consider a flow that has both loss and delay requirements, and wishes to find a path which is optimal in terms of both. In addition to showing that CPN can do this, this work also shows that CPN's goal functions allow it to find optimal paths in terms of the metrics most important to an application.

While in this section we only deal with the combination of loss and delay, these are not the only metrics that have been shown to be able to successfully work with the methodology we present in this chapter. The goal formulations we use here were also used with delay and battery life in mobile ad-hoc routing (40) with very good results (it was shown that a trade-off between min-delay routing and network lifetime was achieved). The formulations that we develop in this chapter could also be used with other QoS metrics.

We begin this chapter in Section 6.1 with a discussion of existing approaches to selecting routes based on multiple objectives, and why this is such a difficult problem to consider. Then, in Section 6.2 we introduce our own approach to routing loss and delay sensitive flows using CPN. We present the results of this work in Section 6.3 where we demonstrate that not

only can CPN be configured to individually find loss and delay-optimal paths, but it can also optimize combinations of the two.

6.1 Related Work

Multi-objective optimization is a difficult problem that is encountered in many disciplines including Chemical Engineering, Aerospace Engineering, Biology, Economics, and so on. Many times, these problems require heuristic approaches due to the fact that as the number of input variables increase, a *combinatorial explosion* of the search space occurs, making it computationally infeasible to enumerate all of the candidate solutions. The specific problem considered in this work is the multi-objective routing problem, which is represented as selecting an optimal path in terms of many, possibly conflicting, cost metrics. This chapter begins with a precise definition of the problem, and then presents an overview of existing approaches.

Multi-objective routing (also known as the Multi-Constrained Optimal Path, or MCOP, problem) is a difficult problem that has been shown to be NP-complete¹¹ meaning that as the number of routers and edges increases, a *combinatorial explosion* of the number of possible paths occurs. This section precisely defines our multi-objective routing problem and the goals of our proposed solution.

A general multi-objective optimization problem is posed as follows (74):

$$\begin{aligned}
 &\text{Minimize for } \mathbf{x} \in \mathbf{X} \quad \mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), \dots, F_k(\mathbf{x})] \\
 &\text{subject to} \quad g_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \dots, m \\
 &\text{and} \quad h_l(\mathbf{x}) = 0, \quad l = 1, 2, \dots, e
 \end{aligned} \tag{6.1}$$

where k is the number of objective functions, m is the number of inequality constraints, and e is the number of equality constraints. $\mathbf{x} = [x_1, x_2, \dots, x_n]$ is the set of n independent variables, and $\mathbf{F}(\mathbf{x})$ is a vector of objective functions. We define an ordering on $\mathbf{F}$ such that $\mathbf{F}(\mathbf{x}) < \mathbf{F}(\mathbf{y})$ iff $F_1(\mathbf{x}) < F_1(\mathbf{y}), F_2(\mathbf{x}) < F_2(\mathbf{y}), \dots, F_k(\mathbf{x}) < F_k(\mathbf{y})$.

¹¹ Although, as we will see, at least one researcher feels the difficulty of the problem has been a bit exaggerated for many practical cases.

The *feasible decision space* $\mathbf{X}$ is defined as the set $\{\mathbf{x} | g_j(\mathbf{x}) \leq 0, j = 1, 2, \dots, m; \text{ and } h_i(\mathbf{x}) = 0, i = 1, 2, \dots, e\}$. The *feasible criterion set* $\mathbf{Z}$ is defined as the set $\mathbf{F}(\mathbf{x}) | \mathbf{x} \in \mathbf{X}$

Applying this definition to the MCOP, $\mathbf{X}$ represents the set of all feasible paths (*i.e.* those that satisfy the user's criteria), each metric that the user wishes to be optimized is represented as a function in $\mathbf{F}(\mathbf{x})$, and $g(\mathbf{x})$ and $h(\mathbf{x})$ are the user's constraints.

6.1.1 Pareto Optimality and the Pareto Frontier

The predominant concept in defining an optimal solution (path) is that of *Pareto optimality* (83), which is defined as follows:

Definition 4 *Pareto Optimal: A point, $\mathbf{x}^* \in \mathbf{X}$, is Pareto optimal iff there does not exist another point, $\mathbf{x} \in \mathbf{X}$, such that $\mathbf{F}(\mathbf{x}) \leq \mathbf{F}(\mathbf{x}^*)$, and $F_i(\mathbf{x}) < F_i(\mathbf{x}^*)$ for at least one function.*

All Pareto optimal points lie on the boundary of the feasible criterion space $\mathbf{Z}$. The set of all Pareto-optimal solutions forms the *Pareto Frontier* (see Figure 6.1). Defined in another manner, the set of Pareto-optimal solutions contains those points which are *not* dominated by any other solutions, making them of particular importance to a decision-maker.

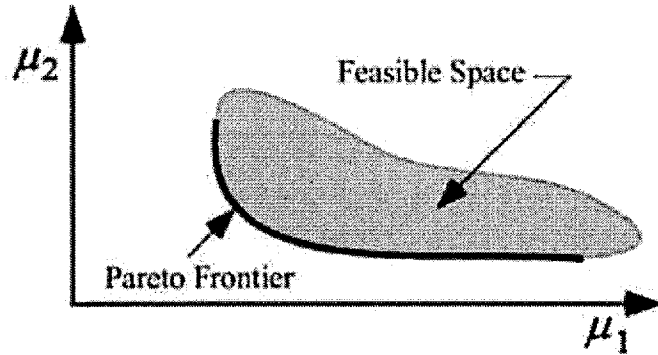


Figure 6.1: The Pareto Frontier for a two-criteria optimization problem (Figure from (76)). For the MCOP problem, μ_1 and μ_2 represent two criteria that the user wishes to optimize.

As can be seen in Figure 6.1, there does not exist a *single* solution which is optimal for the user's metrics μ_1 and μ_2 . Instead, for different levels of importance of the two metrics, there

are different paths which can be used. It will be our goal to discover a set of paths, and take user preference into account when determining which one to take.

6.1.2 Approaches to Multi-objective Routing

Among the literature, there exist a number of approaches towards multi-objective routing. One key distinguishing characteristic among these approaches is whether they attempt to work in an on-line dynamic environment where information is always changing, or in an off-line manner, where the information that they operate on is assumed to be static. This difference is key because it dictates how difficult the problem is; as we will see, the off-line case has a sizable body of work which shows that, for many practical instances, the NP-complete nature of the problem has been over-emphasized. While our specific implementation will only consider the more difficult, on-line scenario, it is still useful to review some of this analysis for the off-line case.

Static, Off-line Algorithms

Beginning our discussion of static algorithms, it is necessary to discuss some of the recent work of F. Kuipers and P. Van Mieghem. In (68), they argue that even though the MCOP problem is NP-complete, it can be *exactly solved* for many of the practical cases, making heuristic approaches superfluous. They show that it's NP-complete nature depends on four conditions: (1) the topology, (2) the granularity of link weights, (3) the correlation between link weights, and (4) the constraints. They thoroughly demonstrate that these four conditions are necessary for the MCOP problem to fulfil its NP-complete nature. They also postulate that many practical instances of the MCOP problem do not fulfil these requirements, and thus can be exactly solved. The authors own approach to exact QoS routing is given in (78), and a thorough comparison to other multi-objective routing algorithms is given in (69) which demonstrates that their SAMCRA algorithm performs better than any other proposed algorithms. In (6), they modify their algorithm to achieve optimal network utilization (*i.e.* traffic engineering) through different path length functions.

Some of the earliest work on the multi-criteria routing problem in computer networks was performed in (20), which presents modified Dijkstra and Bellman-Ford algorithms to work on the optimal delay-cost routing problem. By solving two new routing problems, each one giving one of the cost metrics *coarser resolution* (i.e. integer values), the overall problem can be solved in polynomial time. This approach does not guarantee a solution will be found, even if one exists, but obtained solutions will always be solutions of the original problem.

While the above cited body of work enhances our understanding of the MCOP problem, it should not imply that it has been fully solved. The largest limitation in the above methods is that they assume that link weights remain stable throughout the running of each algorithm, and an exact algorithm still requires a significant time to converge to a solution. Thus, they cannot immediately be applied to an on-line environment where cost metrics are frequently changing; thus, we feel the above algorithms will converge to a solution, which will then be out of sync with current network conditions. This is something which has yet to be fully addressed by researchers.

Dynamic On-line Algorithms

Following our discussion of static algorithms, we now turn our focus to those algorithms which have been presented in the literature for our specific problem at hand. All of the algorithms which we discuss are heuristic algorithms which act on the most-current network state available, and attempt to make timely routing decisions.

Weighted Combination of Objective Functions

Given multiple objective functions, a simple method for enumerating the Pareto frontier is to combine each of them into one objective function, giving each one a different weighting. These weights can either be user-specified, which will yield a single Pareto optimal solution, or the weights can be varied during the search, in an attempt to enumerate the Pareto frontier. Formally, this can be defined as:

$$U(\mathbf{x}) = \sum_{i=1}^k w_i F_i(\mathbf{x}) \quad (6.2)$$

where $[w_1, w_2, \dots, w_k]$ is the vector of weights typically set by the decision-maker such that $\sum_{i=1}^k w_i = 1$.

The main limitation of weighted sum methods such as these is their inability to yield solutions that lie on non-convex regions of the Pareto frontier (80; 23; 65; 5) (see Figure 6.2). In addition, it can be a time-consuming process to evolve a Pareto frontier, because it requires fixing the weight values, obtaining a solution, and then modifying the weights to obtain another one (unless this is done in parallel). On the other hand, if the user's preferences are known in advance, then we can directly obtain the route that best-suits their needs.

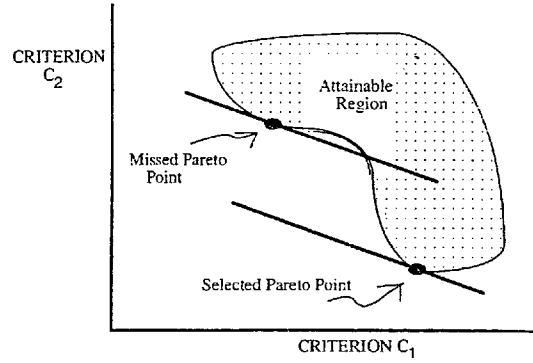


Figure 6.2: A convex Pareto Frontier where weighted sums methods would fail to enumerate all solutions (Figure from (5))

One method to overcome these limitations is discussed in (67), where the authors provide heuristic solutions to the MCOP problem using the non-linear cost function:

$$U_\lambda(p)(\mathbf{x}) = \sum_{i=1}^k (w_i F_i(\mathbf{x}))^\lambda \quad (6.3)$$

where $\lambda \geq 1$. They show that as $\lambda \rightarrow \infty$, the likelihood of finding a feasible path increases because the cost function is able to closer map to the Pareto frontier, discovering solutions that might be missed with a linear function for reasons discussed above. Based on this method, they present a heuristic approach based on Dijkstra's method that performs well.

Genetic Algorithm-based approaches

Holland (55) first introduced the concept of *Genetic Algorithms* (GA), which are a biologically-

inspired computation technique based on the method in which genes encode phenotypes, and the way in which natural selection tends to evolve organisms which are better suited to their environment than the parents. In its implementation, solutions to a problem are encoded as chromosomes, and a simulation of survival takes place, where genetic operators such as mutation, crossover, and fitness are used to create new generations of solutions from its parents. GAs are particularly attractive for multi-objective optimization because they can evolve a set of solutions, which can converge to the Pareto frontier.

In (66), the authors expand their earlier work with GA-based routing for single QoS metrics into multi-objective routing. Utilizing a novel, tree-based gene coding for routes, they are able to represent their population of routes using chromosomes all of the same length, greatly simplifying genetic operations such as crossover and mutation. While the authors only examine delay and loss, there is a discussion of ways to extend to multiple objectives. While a good first step, this work suffers from a number of drawbacks. First, while there exist a number of GA-based multi-objective optimization algorithms (such as (24)), the authors make no attempt to take advantage of this approach, and instead use a single-objective GA whose fitness is evaluated using a weighted sum approach such as equation 6.2. In addition, no comparative results exist which compare their approach to other well-known approaches, so the method's efficiency and run-time complexity can not be evaluated.

GA-based approaches to the multi-objective routing problem are also presented in (82). As opposed to the above work, they emphasize the use of a multi-objective GA to find a set of solutions well-distributed along the Pareto frontier. While their approach seems promising, the work appears in its preliminary stages, as results have not been presented as to how well they compare with other approaches such as we have discussed.

One interesting hybrid approach for the MCOP problem in mobile ad-hoc networks is presented in (75) using fuzzy set theory, with weightings derived through the use of a GA. The work utilizes four metrics: path length, battery capacity, queue length, and link stability to evaluate proposed paths.

In general, a GA-based approach to on-line routing needs to take into account the dynamic

nature of the problem, and the fact that convergence of a population to optimal solutions may take sufficient time that the obtained decision is based on data that has sufficiently changed to make it invalid. Many of these approaches do not directly address this problem, making it unclear as to how well they are actually suited to the problem at-hand. These same comments apply to many of the algorithms we discuss here.

Genetic algorithms have been used with CPN as well – albeit with a single metric – with positive results for low traffic rates, but poor performance under medium to high load (37). This was attributed to the fact that the GA runs at time-scales which are sufficiently different from the rate at which network dynamics operate that the decisions of a GA are no longer relevant by the time they are made.

Ant Colony Optimization

Ant Colony Optimization (ACO) is based on observations of how ants locate food. When ants travel from their nest to a food source, they deposit a chemical substance, called *pheromone*, on the ground. An ant makes a decision of which way to go in a probabilistic manner depending on the concentration of pheromone they smell. In the beginning of a simulation, there is no pheromone, and the ants take any path with equal probability. However, due to the fact that ants taking shorter paths will reach the food source quicker, they will be more likely to return along the shorter path because of the higher concentration of pheromone. While this process iterates, more pheromone is deposited along the shorter path than the longer path until, eventually, all of the ants end up using the shortest path (22; 12; 25).

The work in (14; 15) presents an ant-based routing algorithm in computer networks that performs well against other popular network routing protocols in simulations, while Bean, et al (8) develop an analytical model for ant-based routing. They find deficiencies that they conclude are based on the fact that “the ants employ the same policy for exploration as they do for decision-making”. They consider this a problem because ants simultaneously explore the network according to a different policy from data packets, while at the same time trying to measure trip times for data packets. Their analysis suggests that the measured times are in fact *perturbed* with respect to the delays experienced by data packets. To correct

this deficiency, the authors suggest de-coupling the process of network exploration and delay estimation by only using ant-based routing on the first router in the network, and using a modified data routing procedure for subsequent hops that still maintains some sense of exploration. This is a criticism that we will take into account for our own methods.

There have been a number of ACO mechanisms presented in the literature for optimizing multiple objectives. A common thread among many of these is the use of multiple ant colonies, each focusing on different objectives, or on different weightings of the objectives to evolve solutions in different areas of the Pareto front. There have been routing algorithms suggested which attempt to optimize multiple QoS metrics, in both the wired environment ((94)), and in ad-hoc networks ((26)). A survey of these routing algorithms is presented in (95), while (33) presents a comprehensive evaluation of numerous multi-objective ACO algorithms, in addition to two state-of-the-art multi-objective GA algorithms, with promising results of the superiority of the ACO's derived solutions.

Ant-based algorithms share a lot in common with our own approach to routing. Decisions are made in a distributed manner based on the experiences of previous packets. However, one of the biggest differences between our approach and these is the way in which data packets are routed. We choose to source-route data packets according to the best path that we have discovered; whereas in ant-based approaches, a more stochastic routing procedure is used which uses paths in proportion to the level of pheromone. We feel that our approach should use the best paths, and yield better performance, which we hope to exhibit through our empirical evaluations.

6.1.3 Summary

In this section we have introduced the multi-objective routing problem, drawing attention to why it is so difficult, especially in on-line routing scenarios. The existing approaches either do not work at time-scales relevant to the changes in the network (e.g. Genetic Algorithms) or are not able to exploit enough of the best paths in the network as they explore the realm of available paths (e.g. Ant-based routing). We feel that CPN provides a solution to the

problem that accounts for both of these deficiencies which we address in the next section.

6.2 Adaptive Routing based on Loss and Delay

For this work, we consider a flow which has both loss and delay requirements. As discussed in Chapter 2, this could include a variety of real-time applications such as voice or video applications. We begin by formulating a QoS goal which incorporates both metrics. Then, by using two different networking testbeds we show that not only is CPN able to use its goal function to optimize an individual metric, it is also able to combine those metrics to find paths which are good in terms of both.

6.2.1 Formulating a QoS Goal

For an application which has QoS needs that can include both loss and delay, the QoS goal that may be used to route packets will have to combine in one single goal function both the forward loss (L_f) and delay (D_f) incurred from source to destination. We are chiefly concerned with optimizing both metrics in the forward direction *only* (i.e. the path from the source to the destination). We do this for a number of reasons, the first and foremost is that the application running on the sender is only concerned with how fast and how reliably it can transmit its information to the destination. We say this with the caveat that in the case of a two-way transaction, the inverse process is also running at the destination for its communication with the source. Because network paths are not symmetric, it is important that we try to measure only the forward direction metrics.

The simplest method for combining these objectives into a unified goal function G is by using a weighted combination:

$$G = w_L L_f + w_D D_f \tag{6.4}$$

with user-determined weights w_L and w_D . This is comparable to the approach employed in (47), where the authors use a combination of delay and shortest-path routing. One particular

problem with this scheme is that the units of the two metrics differ by quite a large margin – loss is expressed as a probability with values ranging from 0 to 1 whereas delay is unbounded, taking any value greater than 0. One method to deal with this scenario is to measure the values encountered in practice and scale these by a constant factor in the goal equation. In combination with the user weightings, this could be useful, but it is still very dependent on constant factors which may differ for different networks. In addition, the discussion given in Section 6.1.2 highlighted a number of other drawbacks to this simple scheme, thus leading us to looking for an alternative formulation.

As an alternative strategy, we can formulate a goal by considering the scenario in the terms of a flow with both loss and delay requirements. In this case, a sender has a packet of data that they wish to transmit to the destination. If the packet is lost, then the sender must wait a time-out period before it considers the packet lost and then must repeat the scenario again. On the other hand, if it is not lost, then it will want the packet to arrive to the destination with the lowest amount of delay. This scenario can be formulated in a goal function G as follows:

$$G = (1 - L_f)D_f + L_f(T + G) \quad (6.5)$$

where D_f is the forward delay, L_f is the forward packet loss probability, and T is the additional time incurred by a packet which is retransmitted after a loss, including the time-out delay before a non-acknowledged (and presumably lost) packet is retransmitted, and any additional overhead resulting from the retransmission of the lost packet. Note that G appearing on both sides of (6.5) is written under the assumption that the subsequent packet sent out to replace the lost packet will on the average incur the same total cost G , since it too may be lost and could be retransmitted. This expression simplifies to yield the reward $R = 1/G$:

$$R = \frac{1}{T \frac{L_f}{1-L_f} + D_f} \quad (6.6)$$

In order to use R we must obviously be able to estimate D_f and L_f . In Section 3.1 we describe how ACK packets deposit an estimate of ‘delay to the destination’ into the mailboxes of nodes

that they visit. In order to select a particular path in the network based on composite path QoS metrics, CPN also needs to estimate path packet loss ratios. We will discuss how to estimate path loss in the following section.

Estimating Path Loss

In order to use loss as a QoS metric, we first need to form an estimate of the current loss within CPN. It is necessary to have some sort of estimate, because it is impractical to have the destination nodes keep a count of the number of packets received for each possible route from every possible source; we need to find a scheme that will reduce the amount of data that is stored. This requires us to make a simplifying assumption based on the idea that forward and reverse routes generally use the same set of nodes and links. Thus, we assume that the DP loss ratio L_f from the source S to the destination D is proportional to the ACK loss ratio L_b in the opposite direction:

$$L_b = \alpha L_f \quad (6.7)$$

Let N be the number of DPs sent from S to D , and A be the corresponding number of ACKs received by S . We can write:

$$A = (1 - L_f)(1 - L_b)N \quad (6.8)$$

Combining this with Equation 6.7 yields:

$$\frac{A}{N} = 1 - L_f - \alpha L_f + \alpha L_f^2 \quad (6.9)$$

Assuming that forward and reverse loss rates are identical (i.e. we set $\alpha = 1$) yields:

$$\frac{A}{N} = 1 - 2L_f + L_f^2 \quad (6.10)$$

Using the quadratic formula yields solutions:

$$L_f = \frac{2 \pm \sqrt{4 - 4(1 - \frac{A}{N})}}{2} \quad (6.11)$$

This simplifies to:

$$L_f = 1 \pm \sqrt{\frac{A}{N}} \quad (6.12)$$

Because we are only interested in the solution where $0 \leq L_f \leq 1$, we obtain:

$$L_f = 1 - \sqrt{\frac{A}{N}} \quad (6.13)$$

6.2.2 Rewards for Loss

If routing only selects paths which offer the lowest packet loss, there are several ways in which we can construct the reward R . One approach is to set $D = 0$ in expression (6.6), obtaining:

$$R = \frac{1 - L_f}{TL_f}, \quad (6.14)$$

so that $1/T$ acts as a constant multiplier. In practice, since we do not want R to be infinite when $L_f = 0$, we set:

$$R = \frac{1 - L_f}{T(L_f + \epsilon)}, \quad (6.15)$$

where ϵ is a constant representing some minimal value for the loss. A simpler approach is to use R of the form:

$$R = \frac{\beta}{L_f + \epsilon}, \quad (6.16)$$

which relates loss directly to the reward. This is the approach we have taken in our experiments when we just deal with loss (rather than loss and delay). In the experiments we report, we have used the following numerical values of the constants: $\epsilon = 10^{-5}$ and $\beta = 0.5$.

6.3 Measurement Results

We wanted to test how the formulations in the previous section for rewards based on loss and a combination of loss and delay would impact CPN's performance, and whether CPN's learning algorithm would be able to optimize different QoS metrics simply by varying the Goal equation. The measurements that we report in this section were performed under a variety of conditions on the two network testbeds in Figures 6.3 and 6.7. These experiments were conducted in the early days of CPN development (and also with the CPN 1.0 implementation) which is why they are different from the testbeds we have used in earlier chapters – they were conducted at a time when not as many CPN routers were available. However, since the results we obtained on the two testbeds showed the same trends, and because this formulation has been used for other QoS metrics (e.g. battery life and delay in (40)) we are confident that these results would be seen on other topologies.

6.3.1 Experimental Methodology

It was our goal to experiment with the impact that the goal equation had on CPN's loss and delay performance. For that reason, all tests were conducted using a flow of UDP packets entering the CPN network with constant bit rate (CBR), varying the sending rate from very light to very high in an attempt to see how CPN's flow switching can compensate for congestion-based performance degradation.

In wired networks, both packet forwarding delays and loss increase along with congestion. Because they are correlated, optimizing one metric can invariably effect the other. However, in an environment where loss may be unrelated to congestion, such as in a wireless setting, minimizing delay will not necessarily minimize loss. To study the effect of packet loss when it is not just a consequence of congestion, an “artificial loss ratio” was introduced in one of the routers (Node 1 in Figure 6.3), which was an additional source of loss in the experiments beyond pure congestion.

Completing our experiment description, each router ran the CPN code (at the time these

measurements were taken it was running the “old code” described in Chapter 3) on a Pentium III-class PC running Linux kernel 2.4. All flows had a fixed packet size of 1024 bytes, and all inter-connects used 10Mbps point-to-point Ethernet links.

6.3.2 Initial Testbed

The experiments in this section were all conducted on the “small” testbed in Figure 6.3. We used this environment to do our initial explorations of CPN’s ability to optimize various QoS metrics before we scaled up our measurements to a larger testbed. Though we only used 6 routers, we were still able to generate results and trends which were also observed in the larger experiments.

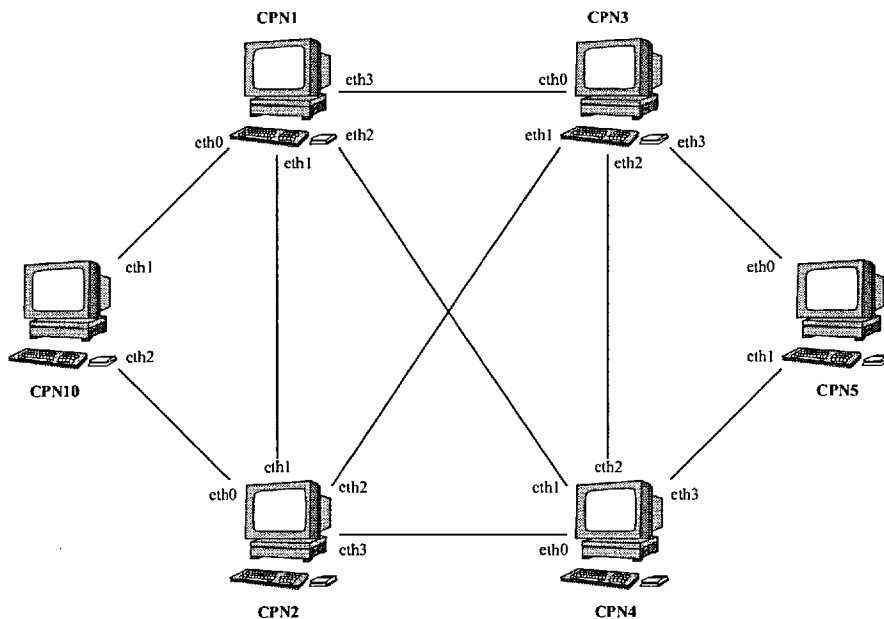


Figure 6.3: Small CPN testbed.

We first conducted experiments with the artificial loss rate set to 0. In Figure 6.4, we can see that both delay and loss performance when using delay as the QoS goal performs best. Thus routing based on delay appears to minimize *both* observed loss and delay.

When we introduce a very small (1%) artificial packet loss rate (see Figure 6.5) delay does not seem to change, but the loss performance curves show that using loss as the goal function

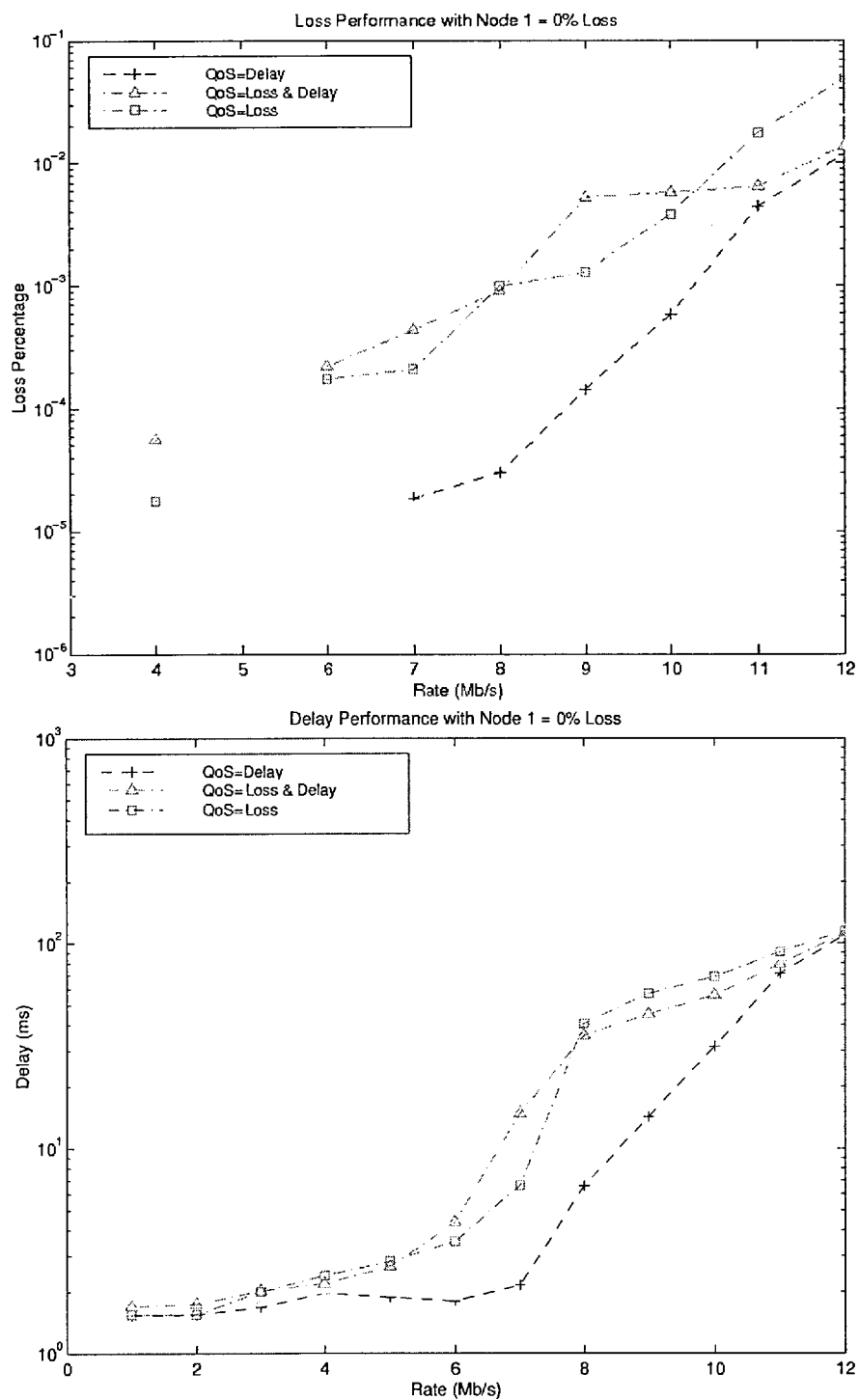


Figure 6.4: Loss (top) and delay (bottom) in the small testbed with 0% artificial loss rate at Node 1.

is clearly the best as long as the network is not heavily loaded (i.e., link traffic under 8Mbps). When traffic is close to saturation levels, losses due to congestion dominate and (as one would expect) the results become similar to those of Figure 6.4. Finally, in Figure 6.6 we increase the artificial packet loss ratio to 5 and observe once again that using loss as the goal in the routing algorithm provides the best overall results.

6.3.3 Increasing the Size of the Testbed

To provide further evaluation of composite goal functions, we conducted another set of measurements on the larger CPN testbed consisting of 26 nodes shown in Figure 6.7. The same UDP packet stream with CBR, as before, was sent into the larger testbed into Node 10 as the source, for forwarding to Node 7 as the destination. The CPN protocol was run in the usual manner with paths being discovered by SPs, while DPs were used to carry and source route the payload UDP packets. No artificial losses were introduced.

As shown in the top figure of Figure 6.8, the reduction in packet loss rate due to the use of loss alone, appears even more significantly in the larger testbed. Here, using the goal function based only on loss results in the lowest observed loss rates. The curves in the bottom figure show that the lowest delay is obtained by using only delay (or loss *and* delay) in the goal function. The linear scale used for delay (y-axis) in this figure clearly shows a peak for traffic in the range of 11 to 12Mb/s, with a reduction in delay above that value, when loss or only delay are used in the routing goal function. The drop in delay at higher traffic values is due to the significant loss of packets which results in lower congestion.

Figure 6.9 summarizes measurements on the larger testbed, with packet loss only resulting from congestion (i.e. no artificial packet loss) from the perspective of QoS perceived by the user. The purpose of these figures is to see whether the end user is indeed obtaining the outcome in QoS that it is requesting. On the y-axis of the three figures we show the reward function value at the end user (and not the numerical value that is used by the SPs, which operate at the lower network level). The reward is computed using the formulas given in equations 3.1, 6.6 and 6.16, using the *measured* values of loss and delay. In the upper curves,

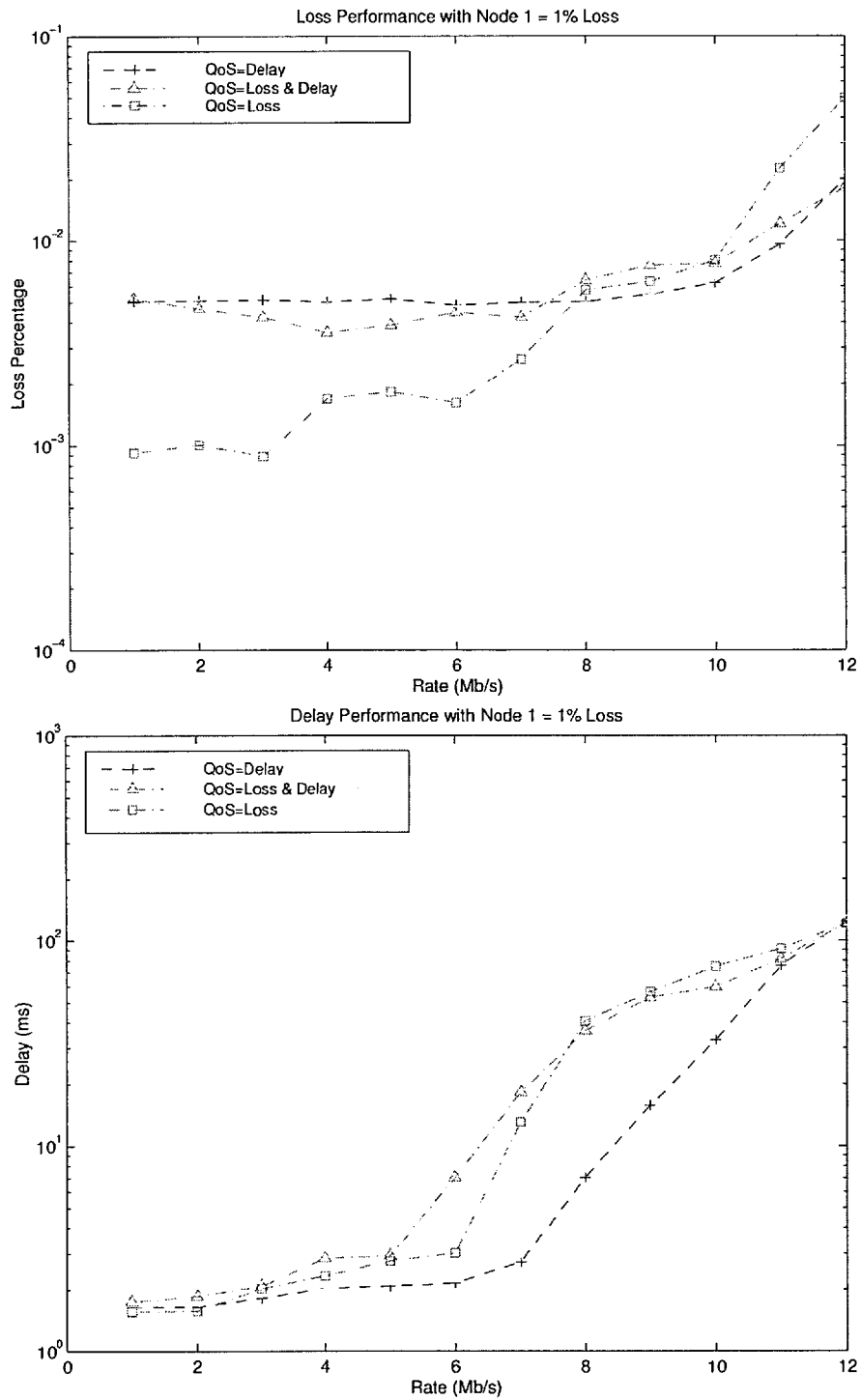


Figure 6.5: Loss (top) and delay (bottom) in the small testbed with 1% artificial loss rate at Node 1.

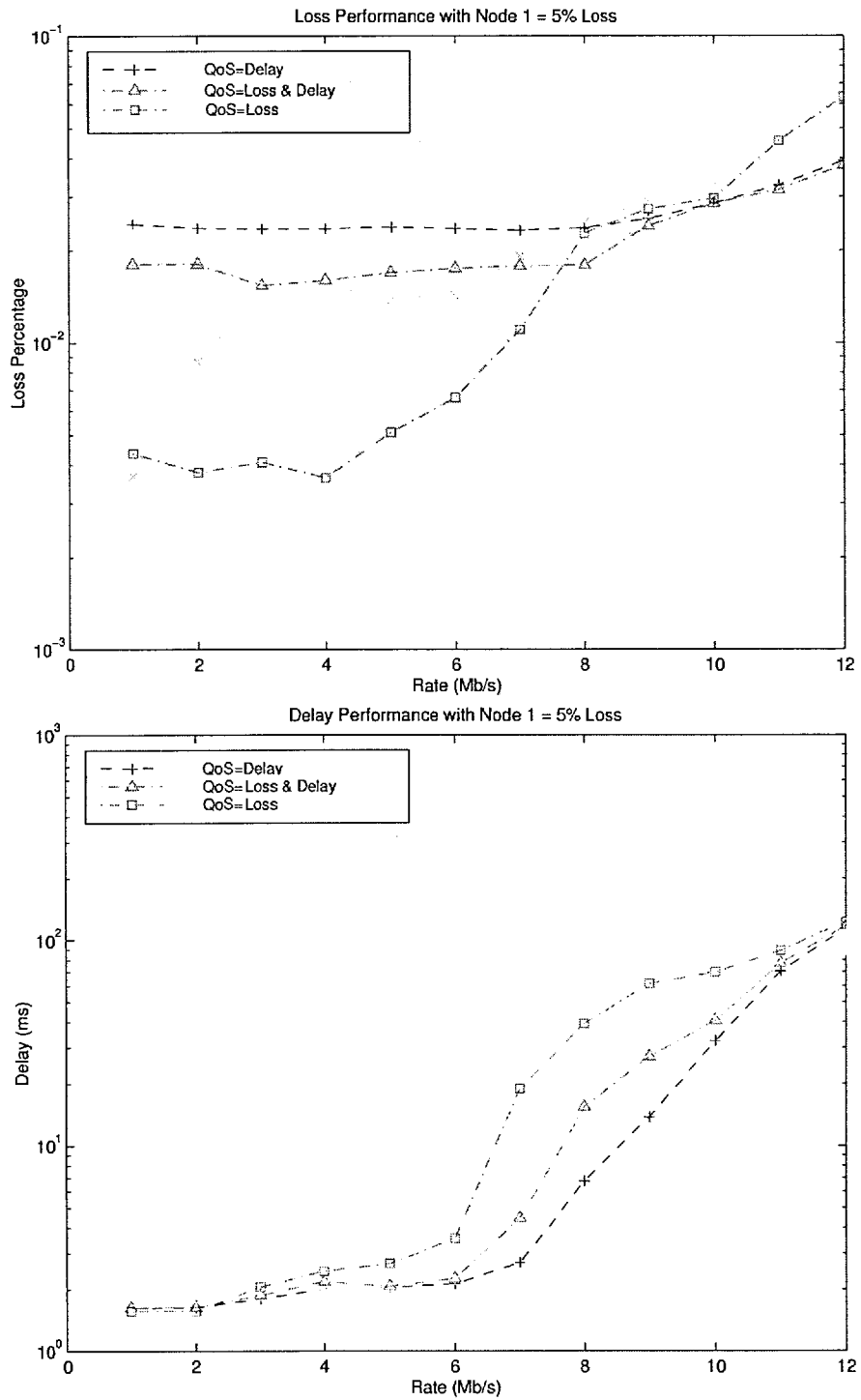


Figure 6.6: Loss (top) and delay (bottom) in the small testbed with 5% artificial loss rate at Node 1.

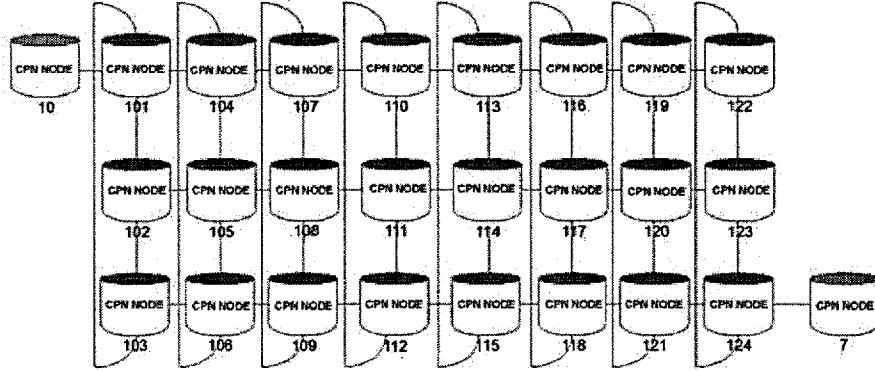


Figure 6.7: Larger CPN testbed.

we show the “loss based” reward for the user, as a function of user traffic. We see very clearly that the highest (therefore the best) reward is obtained if we use *only loss* in the goal function, while using *delay* and *loss and delay* are worse but equivalent from the user’s perspective. In the middle and lower figures, we see that using *delay* as the QoS goal will provide the user with the best reward at low traffic; all three goal functions provide equivalent QoS at higher traffic values. The two lower figures are nearly identical. The reason is that when we combine loss and delay to compute the reward function, the effect of the loss rate for (1) low loss values (i.e. less than 10%) is negligible in the reward function, while for (2) high loss values all of the congestion avoidance methods embodied in the three goal functions provide equivalent performance and QoS. However in view of the upper set of curves, we do see that using a goal function which is specifically tuned to the user’s needs is justified since in some cases it will have a definite effect, while in other cases it will not be detrimental to user perceived QoS.

One more set of measurements on the larger testbed are reported in Figure 6.10. The purpose is to evaluate the impact of the parameter T used in the Goal functions. The curve shows that varying T between 10^2 and 10^5 has little impact on the values of the reward value experienced by the user; a more significant difference is only observed at high traffic rates (hence high loss rates) when $T = 10^7$.

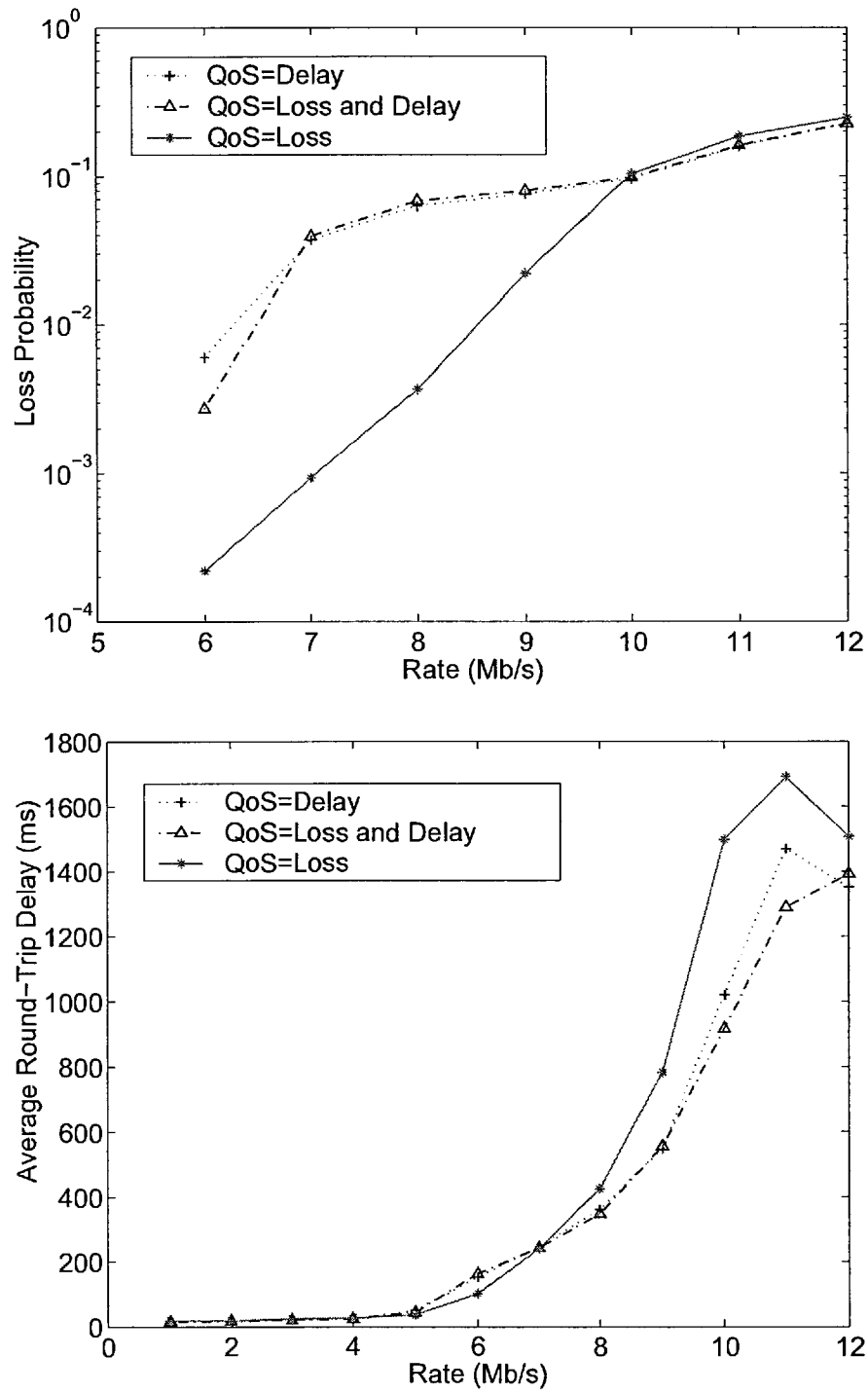


Figure 6.8: Loss (top) and delay (bottom) in the large testbed.

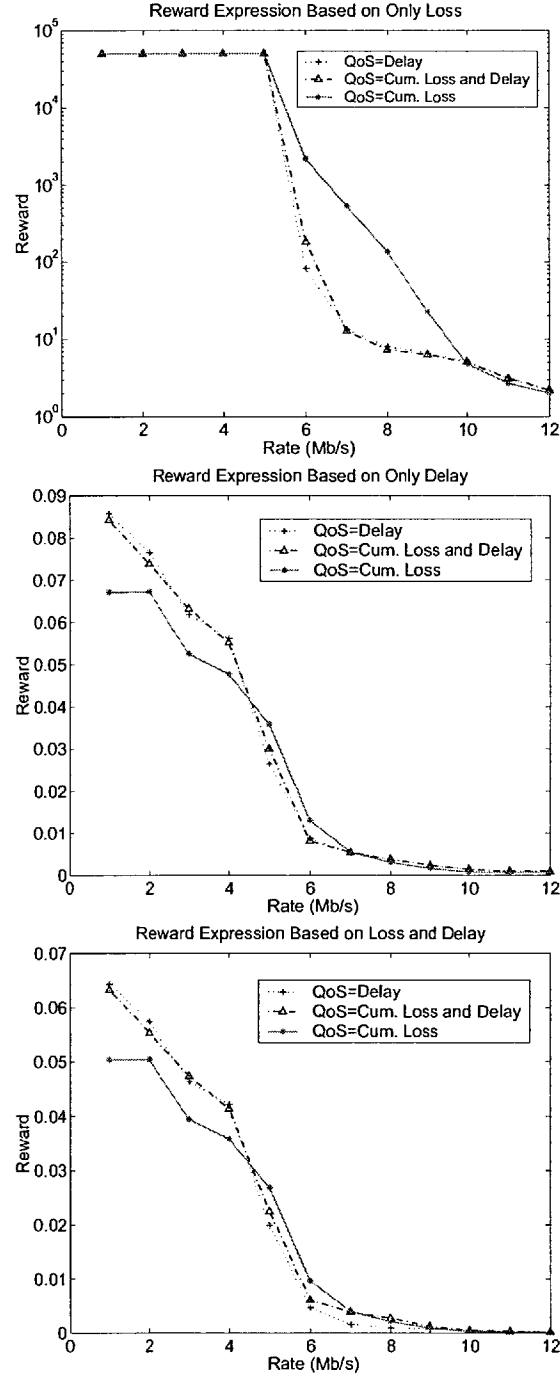


Figure 6.9: Measured reward value at user-level when RL is based on Loss (upper), Delay (middle), and combined Loss and Delay (lower) in the large testbed.

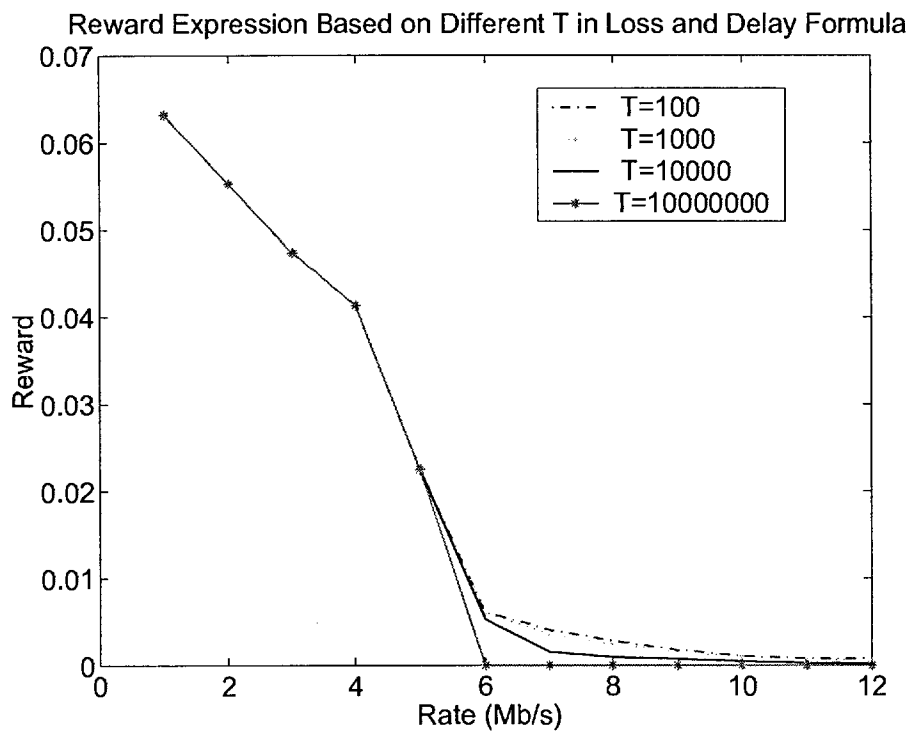


Figure 6.10: Observed reward for different values of T . Note that T is a constant in formula 6.6, not a real time-out.

6.4 Future Work

The biggest way in which this work could be expanded upon would be to include more than two QoS metrics. As we discussed in chapter 2, there are a variety of metrics that impact application performance and these are but two. This approach was already used by Gelenbe, et al in (40) to route based on power and latency with promising results. However, it is unclear how routes could be selected based on three or more metrics. Environments such as mobile ad-hoc networks, sensor networks, or inter-planetary networks typically have many metrics that must be taken into account when routing and a generic way of combining an arbitrary number of metrics would go a long way towards enabling CPN routing in these environments.

One way in which multiple QoS metrics could be combined into a single decision maker would be to use multiple random neural networks. Given N metrics, we could have N RNNs, each focused on finding the best path in terms of each. Then, an algorithm would be needed in order to combine the decisions of each neural net into one overall *best* decision for routing.

6.5 Summary

We set out to show that CPN could find paths optimal in terms of both loss and delay, which are both metrics which greatly impact the performance of a real-time flow. This showed the benefit of CPN's flexible reinforcement learning-based approach to evaluating and choosing routes. By changing the evaluation criteria (which is simply a goal function), we showed that CPN switches the way it utilizes routes and is able to discover (and take advantage of) routes which best fulfil the reward criteria.

In the measurements on the small testbed, we have observed that when packet loss is only due to congestion, the use of delay by itself in the routing goal function results *both* in the smallest delay and the smallest loss. This is a strong indication that measuring delay is a good sufficient indicator for congestion in the small testbed. However, when non-congestion related losses are introduced in a node, using loss alone results in the smallest loss. Nonetheless, when the

network is saturated, we see that all goal functions (with or without both metrics) essentially result in the same delay and loss. In the larger testbed, we observe when the network is operating close to congestion, the use of *delay* or *loss and delay* in the goal function results in better delay characteristics for DPs. Similarly, using loss alone in the goal function provides the smallest loss rate for DPs. Thus our experiments support the claim that there is a good correspondence between the QoS Goal that the SPs use to find paths, and the resulting QoS observed by the users' payload.

While our results for combining loss and delay were not conclusive, this was mainly due to the fact that loss and delay performance are tightly correlated with congestion. This same approach was later used in mobile ad-hoc routing by Lent, et al (40) as a combination of delay, and *path availability* which was proportional to battery life. It was shown that this leads to a good trade-off between prolonging the lifetime of a network by utilizing nodes with larger amounts of remaining battery and shortest path routing.

In the preceding chapters, we have explored CPN's performance in a number of different aspects with very positive results. However, one disadvantage of these experiments is that they required the replacement of the legacy IP routing infrastructure in order to be realized. In the next chapter, we investigate methods for implementing a CPN overlay on top of an existing IP network, which provides an intermediate step towards implementing CPN routing.

Chapter 7

QoS-aware Overlay Routing

The previous chapters have demonstrated and explored the Cognitive Packet Network's (CPN) ability to adaptively learn the best path in a network, in terms of user or application-driven Quality of Service (QoS) needs. Up to this point, this work has taken place in the context of a wired network, where routers are connected to each other using point-to-point links. In terms of the layered model of networking, this work has been done at the networking layer. It is the goal of this chapter to develop a framework for which CPN-like routing can use existing routes computed by an underlying routing protocol (e.g. OSPF or BGP in IP networks) in order to achieve the benefits of CPN routing without the need to completely replace an existing routing architecture (e.g. the Internet).

The organization of the chapter is as follows. Section 7.1 introduces the concept of a *routing overlay* which is what we construct in order to influence the path packets take in the network. We have chosen to structure our overlay using Peer-to-Peer principles, which we introduce in Section 7.2, and describe how we use them for constructing our overlay in Section 7.3. We evaluate the proposed overlay in section 7.4, showing it is able to adaptively learn the network path that provides the best level of QoS.

7.1 Routing Overlays

After Internet routing was shown in a number of classic measurement papers (85; 90) to result in paths that are sub-optimal with respect to a number of metrics, routing overlays were proposed as a method for improving performance, without the need to re-engineer the underlying network. By adding intermediate routing hops into the path a packet takes, they influence its overall route without modifying the underlying mechanism for computing routes.

It should be asked at this point why we cannot just take a *clean-slate* approach, doing away with the existing architecture and having another go with the benefit of hind-sight at its existing deficiencies? There is some merit to this line of thinking; if we were able to replace the current architecture, we could enumerate all of the various issues and problems that have arisen over time, and develop solutions to each of them, and in the end we would have a much stronger infrastructure. Unfortunately, time and again, proposals to overhaul an entrenched protocol or technology (even with clear technical improvements) have met with failure. In the networking context, this can be seen in attempts to introduce QoS routing, upgrading to IPv6, or the fact that SPAM and Denial-of-Service attacks run rampant because of fixable flaws in the IPv4 stack. It seems a fact of life that for the foreseeable future IPv4 is here to stay, so developing approaches that sit atop IP (or run alongside it) have a great deal of merit.

With this in mind, one of the most promising approaches to overcoming the limitations of Internet routing is an overlay network. Resilient Overlay Network (RON) (2) was the first to be implemented in a wide-area network, demonstrating that adding an extra (overlay) routing hop could benefit an application in terms of improved delay and reachability. However, its main drawback is that it did not scale very well: as the number of participating routers increased, its costly $O(n^2)$ probing overhead became a limiting factor. This overhead was a direct result of the fact that RON used all-pairs probing, which has the advantage that the routers are all connected as a full mesh, but the downside is that a reasonable RON overlay could support only about 50 routers before the overhead became too much.

The work of RON has inspired many approaches to create a routing overlay. When considering only the QoS metric of availability, (53) showed that a simple one-hop overlay routing scheme could mitigate 56% of the network failures in their study. By using a subset of RON and PlanetLab (86) nodes, Hu, et al (56) empirically demonstrated that the use of an intermediate node can improve the available bandwidth along a path. More recently, Nakao, et al (81) have proposed reducing probing overhead by structuring an overlay using AS-level information, pruning redundant logical connections and creating a topology similar to the underlying physical links between neighbors. However, no existing work has tackled the problem of building an overlay that could be widely deployed over a sizable population of routers. This is one issue that from the outset we wish to incorporate into our solution.

7.2 Peer-to-Peer Overlay Networks

Real-world large-scale peer-to-peer (P2P) overlay networks operate in harsh conditions: nodes appear and disappear without warning, routers offer widely different performance criteria, and are subjected to rapidly changing workloads. In spite of this, they must still provide a level of service that fulfils their users' expectations for locating content quickly. Taking this into account, we feel they are a strong candidate for structuring our routing overlay.

Overlays can be divided into two broad categories: *structured* which “assign keys to data items and build a graph that maps each key to the node that stores the corresponding data” (e.g. distributed hash tables), and *unstructured* which “organize nodes into a random graph and use floods or random walks to discover data stored by overlay nodes” (16). While structured networks typically guarantee the location of a data item in $O(\log N)$ overlay hops (where N is the number of overlay nodes), they can suffer from long lookup delays as the overlay construction does not typically take into account network proximity (73). Because the peerings (and amount thereof) in the overlay are of critical importance in discovering good-performing routes, we choose to use an unstructured overlay as a basis to allow us a much greater level of control.

Kazaa (FastTrack) (58) and Gnutella (52) are two of the most well-known and popular unstructured overlay networks. They both boast network sizes in excess of one million users, and have seen terabytes of data transferred through them. While a majority of the press that they receive is negative due to the fact that copyrighted works of art are being illegally exchanged, they serve as examples of scalable, large-scale Internet overlays.

In its original form, Gnutella possessed a flat structure where queries were issued using flooding. This was found to be largely unscalable, and also failed to take into account the heterogeneity of participating clients. Gia (19) was proposed to address a number of these shortcomings, incorporating a node hierarchy to allow more powerful peers more responsibility, and also introduced a random walk-based query method. These improvements, among others have been integrated into the Gnutella protocol, which today boasts in excess of 3 million users, while maintaining short routing distances among peers and good connectivity (99).

Since its debut, the protocol has been upgraded to improve its scalability. In a comprehensive study of the modern Gnutella network, (99) showed that it now utilizes a 2-tier hierarchical structure where higher-powered peers (known as superpeers or ultrapeers) become responsible for a subset of overlay peers, and bear the brunt of in-network processing (see Figure 7.1). Ultrapeers must meet minimum resource requirements such as available memory, processing power, available bandwidth, and uptime. The final requirement for promotion is that a node has to sense that there is a shortage of superpeers in the network, which is based wholly on local information obtained through gossiping (e.g. finding a large number of super-peers who have few free slots for new peers). When they are promoted, they assume more responsibility in the network, answering and forwarding searches made by their peers. According to the protocol, leaf peers maintain a connection with 3 ultrapeers, and ultrapeers maintain 30 children and 30 ultrapeers.

The sum total of the various parts of the Gnutella protocol results in a large-scale, completely distributed overlay consisting of millions of nodes, even in the face of a great many challenges. One is that there is no one Gnutella implementation; in fact, it is simply a standard to which

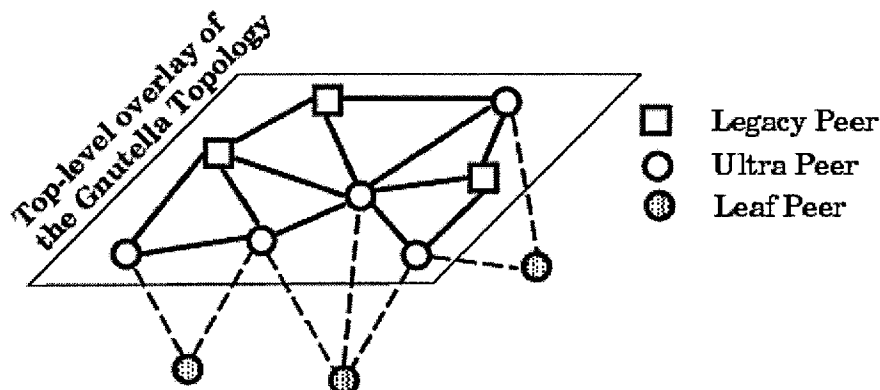


Figure 7.1: The two-tier overlay used by Gnutella (diagram from (99)). Legacy peers are those which run old versions of the software and do not understand the two-tier hierarchy. Figure used with permission from author.

there are many different implementations, some of which are open-source. This means that any client can be changed, even in ways that could destabilize the overlay if they were to become widespread. Another challenge faced by a real-world overlay is the dynamics of peer participation, also known as *churn*. This is carefully studied in (98) for multiple P2P networks, finding that while “a majority of participating peers in the system are long-lived peers, . . . the remaining small portion of short-lived peers join and leave the system at such a high rate that they constitute a relatively large portion of sessions (98).” One final obstacle that these overlays have overcome is Network Address Translation (NAT) which makes it impossible to send peers unsolicited packets, yet one measurement study found that a full 73% of all peers are behind a NAT device. A real-world overlay operates in harsh conditions.

Even in the face of these challenges, (99; 28) found that the overall overlay structure exhibits “onion-like” routing where long-lived peers are more likely to be connected to each other, *forming a core that is very robust to many forms of failure*. More astonishingly, even in a network in excess of 3 million peers, the majority of nodes can be reached in only 5-6 hops. Another interesting characteristic of the overlay is that even though it does not attempt to find optimal peerings in the network (e.g. those with good network properties), the connections that result exhibit a strong intra-region bias through other means such as trying to maintain a connection with users with the same locale setting, and initiating more connections than

needed and keeping the quickest respondents. This mechanism provides many overlay-level peerings which may be exploitable for routing.

7.3 A P2P Routing Overlay

We look to the unstructured overlay Gnutella as a candidate for structuring our routing overlay. We do this because we identify scalability and dealing with churn as two of the most important characteristics of a successful overlay network, and we believe that Gnutella has fulfilled these requirements by scaling to over 3 million active participants, even in the face of a great many obstacles as detailed in the previous section.

7.3.1 Routing Overlay Construction

For the remainder of this chapter, we assume that our overlay construction and maintenance protocol mimics that of Gnutella. Namely, we assume it meets the following criteria which have been identified as key factors in its construction:

- *Two-tier overlay*: “A subset of peers, called *ultrapeers*, form a top-level overlay while other participating peers, called *leaf peers*, are connected to the top-level overlay through one or multiple ultrapeers (99)” (see Figure 7.1).
- *Gossip protocol*: As a distributed and decentralized overlay, it is of vital importance how peers discover participating overlay members. In the beginning this is typically accomplished through a boot-strapping procedure (e.g. using UDP Host Caches or well-known seed peers). After this initial list of nodes however, the discovery of overlay hosts is done using only local interaction with peers. Whenever an overlay peer initiates a connection with a new peer, it is required to share information about leaf and ultra peers it knows to be of good quality (i.e. it has recently been in contact with them). This creates a de-centralized mechanism through which the overlay can scale.

- *Distributed promotion/demotion algorithm*: As a direct consequence of the gossip protocol, leaf and ultra peers are able to estimate the network's need for (or lack thereof) ultrapeers. For instance, if a leaf peer is not able to locate an ultrapeer within a short time interval, it may infer that the network lacks enough ultrapeers and decide (if it meets the necessary resource requirements) to promote itself. In turn, if two ultrapeers communicate, and one of them has few leaf peers, it may inform this to the other node who may decide to demote itself to leaf status.
- *Threshold-based connection maintenance*: According to the common implementations, ultrapeers maintain 30 connections in the top-level overlay, and host between 30-45 leaves. Leaves have on average 3 ultrapeers. As we will discuss later in this section, these values work very well for the CPN routing mechanism, as they provide a variety of routing decisions.

7.3.2 CPN-based Routing Overlay

Given that we have an overlay construction protocol as detailed above, we now turn our attention to how we can use the overlay for routing. We feel that the CPN paradigm of using on-line reinforcement learning, which has shown so much promise in wired and wireless networks, is an ideal candidate to learn routes in the overlay, and adapt to changing network conditions.

Previously, CPN has been restricted to wired and wireless networks; the shift to overlay networking presents unique challenges that must be overcome. We first detail how we propose to use the two-tier hierarchy constructed by our overlay for routing. We then pose the question of *what to do when the overlay introduces more overhead than it can compensate for with a better path*, proposing a unique solution which compactly represents the decision to leave the overlay as it does any other routing option. Finally, we deal with the question of how to send user data using the overlay.

Using the two-tier hierarchy for routing

In a physical network, neighbors are those peers who are directly reachable at the physical layer via a given network interface, whereas in an overlay context, *everyone* is connected to each other via the underlying network¹². This can lead to the case where the number of routing options in the overlay exceeds our capacity to obtain feedback and make sensible decisions. However, this problem can be solved by using the two-tier hierarchy constructed by our overlay construction and maintenance protocol.

The first architectural decision that we make is that *leaf peers should not be responsible for routing traffic*. Leaf peers maintain connections to 3 ultrapeers, and are limited to these 3 links for routing. Similarly, ultrapeers connect to (on average) 30 other ultrapeers who serve as their routing options. At any stage in the routing process, the decision can be made to exit the overlay (see next subsection) and go directly to the destination using the route computed by the underlying routing protocol. We allow any peer to be the originator of a traffic flow, and refer to the source node as the *overlay ingress*. Similarly, the last overlay router in a path we refer to as the *overlay egress*.

This two-tier hierarchy has two immediate benefits which make it ideal for a routing overlay. First of all, it limits the number of routing choices at any given node to a small subset of the overall overlay. It does this while (as discussed in Section 7.2) keeping the overlay fully connected with short paths (on the order of 5-6 overlay hops) between nodes. Because the number of routing decisions is limited, the overlay has the ability to maintain meaningful estimates of link and path quality, allowing it to make informed routing decisions. A second benefit of a two-tier hierarchy is that it exploits the heterogeneity of participating peers. It is not the case that all members of the routing overlay will be able to offer the same level of service to other peers. Potentially, some participants may be home users with slow links, while others may be located close to the network core, able to offer a great deal of route diversity at the networking layer, and making them an ideal candidate for contributing alternate routes

¹²For this discussion, we make the assumption that all peers are *virtually connected* to each other via the underlying network. We omit nodes who are not directly addressable (*i.e.* those behind Network Address Translation firewalls).

to the overlay. By separating the “wheat from chaff”, we allow higher-powered nodes to contribute more to the overlay.

Because in CPN each router maintains a fully recurrent RNN Neural Network (see Chapter 3) with the number of neurons equal to the number of routing decisions, limiting the number of routing decisions is an important step in defining the overlay. In our overlay, each peer maintains a connection to 3 ultra-peers, meaning that its RNN will have 3 neurons. Similarly, each ultra-peer connects to (on average) between 30 and 45 other ultra-peers, and each RNN will have a corresponding number of neurons. Because ultrapeers are responsible for routing traffic for their peers, they have a more diverse set of flows to be routed, and should maintain more routing options. In addition, because of their higher demands, they are more likely to be able to maintain up-to-date state information about a larger set of overlay links.

While at first this could be seen as somehow limiting a leaf peer to a very small portion of the overlay, we maintain that this is actually positive. First of all, by having a limited number of choices, they can build up a comprehensive view of the quality they are offered by all of their ultrapeers in addition to the underlying network. Also, since they do not contribute their resources to others in the network, the amount that they can take should be limited; if they are able to support more overlay connections, they can promote themselves to ultrapeer status.

Learning to use the Overlay

Even though Internet measurement papers have identified that some paths computed by the underlying routing protocol are suboptimal, it cannot be inferred that this is always the case. Thus, we include as a critical component of our overlay the ability to *learn when to use the overlay*, and when to bypass it completely.

We choose to accomplish this by adding an extra routing option to each router in the network (including the source) which corresponds to the decision to exit the overlay. This extra routing decision is represented as an extra neuron in each node’s RNN, and is reinforced using the

same RL algorithm as other routing decisions. If this neuron becomes the most-excited in the neural net, it is interpreted as the decision to leave the routing overlay, and use the underlying network to go directly to the destination (i.e. the overlay egress). Thus, the decision to bypass the overlay entirely is made at the source using *exactly the same process* as deciding a next hop. *The quality of the underlying network is compactly represented at each router in the network, and no special treatment for the decision to exit the overlay need be given.*

Handling User Data

Finally we remark on how our routing overlay should handle user data. Recall that in CPN, at the beginning of a connection a Smart Packet (SP) is generated to search for a suitable route to the destination. An Acknowledgement packet is generated when the SP finds the destination, and travels the reverse route of the SP which generated it, causing the RL algorithm to be run at each network hop (see Chapter 3 for more details).

In a wired network, the SP exploration process has the potential to not find the destination; in a routing overlay this is not the case. When an SP in a wired network exceeds its maximum hop-count or time-to-live (TTL) value, it is destroyed by the network to keep it from infinitely looping. However in an overlay, since we have assumed that every router is connected through the underlying network, it is never the case that an SP will not be able to locate a path; rather, it is only a question as to how well the path performs.

Using this observation, we note that depending on the QoS type, SPs may or may not be used to carry user data. The advantage to using SPs to carry data is that the network overhead of the routing overlay amounts to the extra header added to each CPN packet, and the Acknowledgement packet. Still, an application which is affected by out-of-order deliveries (*e.g.* TCP) may wish to not have its packets encapsulated in SPs, as this may cause data packets to travel different paths and arrive in a different order in which they were generated. Other applications (*e.g.* UDP) may be less sensitive, and may even benefit since their data packets will begin to use optimal routes as they are discovered as opposed to waiting for feedback (although the random component of CPN's path exploration policy may cause them

to occasionally use a poorer performing route). Our preliminary experience with the overlay implementation has confirmed that UDP benefits from SP encapsulation.

7.3.3 Implementation

We have initially implemented our overlay using the Click modular router (64) in C++. It runs as a user-level process, with multiple configuration files. One core file specifies all of the overlay-specific operations common to all applications (*e.g.* sending Smart and Dumb packets, storing routes, and performing the reinforcement learning operations), which is then combined with smaller configurations which contain any of the ingress and egress-specific operations like capturing an application's packet, possibly changing the source and destination address, and injecting it into the CPN overlay.

Routing Decision-Makers (DM)

In order to make the implementation more generic, and to evaluate the trade-off between different learning algorithms, we have abstracted this learning process into the idea of a Decision-Maker (DM). Currently, this is the default RNN neural net using reinforcement learning to update its weights; yet it need not be the only approach. In the past, we have looked at multiple-criteria decision-making which could incorporate multiple neural nets reinforced in parallel; this approach could be easily implemented and evaluated in our architecture. Similarly, an all-together different decision-making scheme could also be evaluated and compared to our RL scheme.

As we have previously described, the “decisions” to be made are the next hop for a Smart packet. The available choices depend on the connections in the overlay that have been established among peers. A given leaf peer can choose a next hop from among their set of ultra-peers. Similarly, ultra-peers choose from their set of neighbor ultra-peers. At any hop in the overlay, the decision can be made to discontinue using the overlay and go directly to the destination. It is important that this decision always be evaluated so that unnecessary

overlay hops are not introduced into the forwarding path.

CPN overlay routing

The overlay routing actions can be broadly broken down into the following stages:

1. *Receiving and validating a CPN packet:* We listen on a pre-determined port (currently this is UDP-based, but it could also use TCP depending on the application) for new IP packets. We then perform necessary reassembly and checksum-verification tasks, and, if it is a valid CPN packet we begin processing. The Click elements and packet flow are depicted in Figure 7.2.
2. *Routing the CPN packet:* Depending on the packet's destination, a different series of operations is carried out (Figure 7.3):

- *Packet for local host:* If the packet is an SP or a DP, its route and CM information will be used to create an Ack packet, which will travel the reverse route through the overlay. Any data carried by the packet will be delivered to the user-space application.

An arriving Ack has its route stored in the Dumb Packet Route Repository (DPRR) along with its reward. It then runs the RL algorithm at the local host, and is discarded.

- *Packet for remote host:* **Dumb** packets have their CM updated according to their reward protocol, and are forwarded using the route in their header. **Acks** calculate a reward based on the local state of the node and their CM, and then run the RL algorithm before they are forwarded. **Smart** packets consult the Decision Maker to determine their next-hop.

3. *Client and server end of the overlay* Figure 7.4 One key element of the process is the IPRewriter Click element which performs Network Address Translation (NAT) on packets both arriving from the overlay and being injected into the overlay by the server.

We use this to make it appear to the server that clients are arriving from the local host (albeit from different transport ports than the server is listening on). Thus, responses from the server are transparently directed to the Click overlay; the server is never made aware of the fact that it is using a routing overlay. A similar procedure is used at the client side of the overlay to mask to a client that it is not talking directly to the server. The client end is nearly identical, but may require an additional socket depending on the application and run-time environment.

Overlay endpoints

So far, we have used the overlay implementation with a variety of both UDP and TCP-based applications. As we currently focus on optimizing path latency, many of the applications we have chosen are real-time, interactive applications, as these stand to benefit a great deal from finding a path that delivers their packets as quickly as possible.

We have used a variety of real-time applications including multi-player games and Voice over IP (VoIP). In the realm of games, we have successfully been able to run our overlay with Quake 2 & 3 and Battlefield 2. We also were able to use our overlay with a popular VoIP server (TeamSpeak). In addition to running our overlay on local computers where we have full control of the operating system, we have also successfully run it on PlanetLab nodes (84) which, because of security precautions, is much more restricted in terms of the packets that can be generated. As a result of verifying our implementation with a number of clients and server applications, we are confident that our overlay would easily integrate with any UDP application using a client/server networking paradigm.

7.4 Evaluation

As a first step in evaluating our proposed architecture, experiments have been conducted to show the effectiveness of our overlay in improving QoS. Our goal in this evaluation is

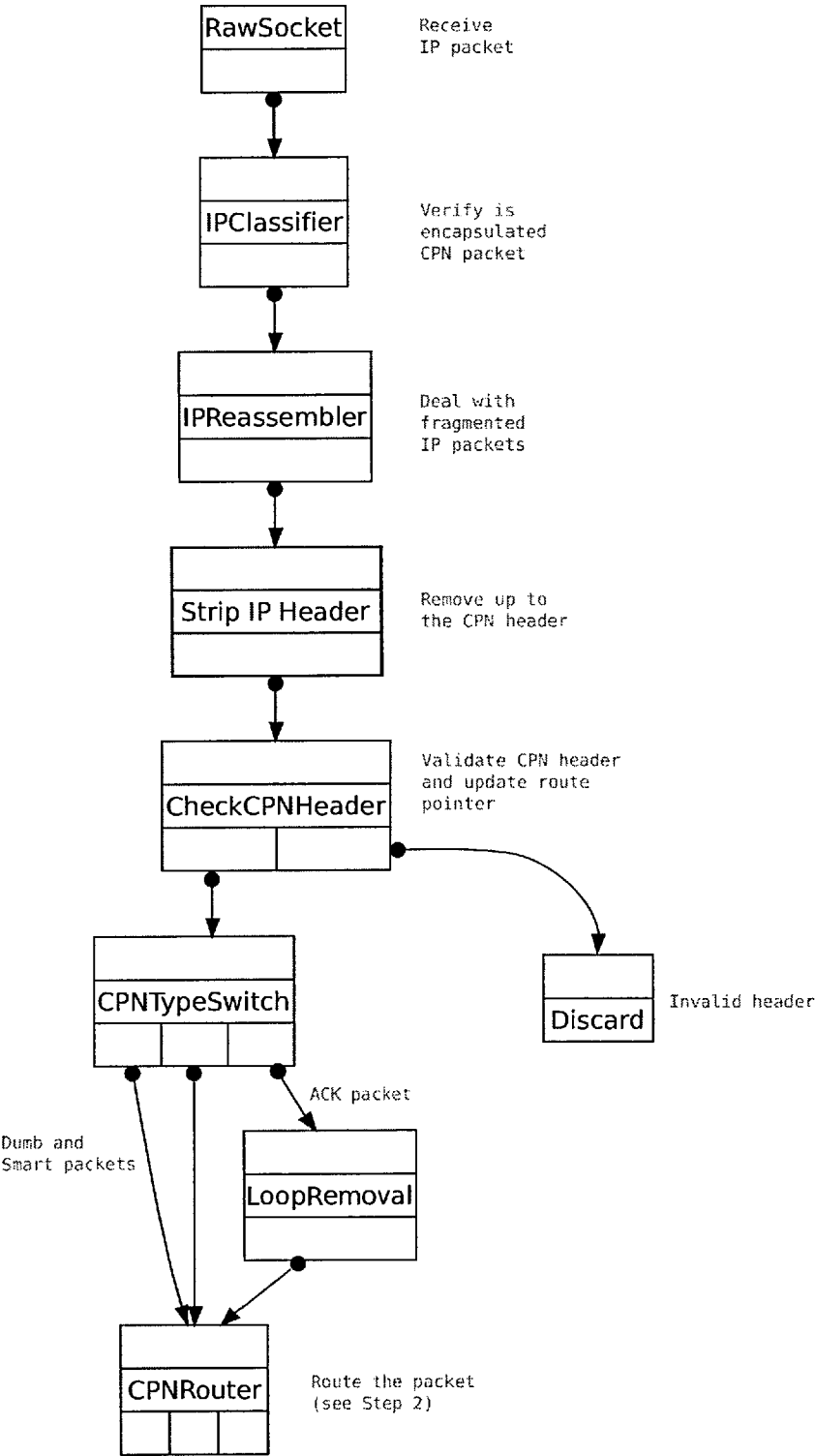


Figure 7.2: Process of receiving and validating a CPN Packet by an overlay router

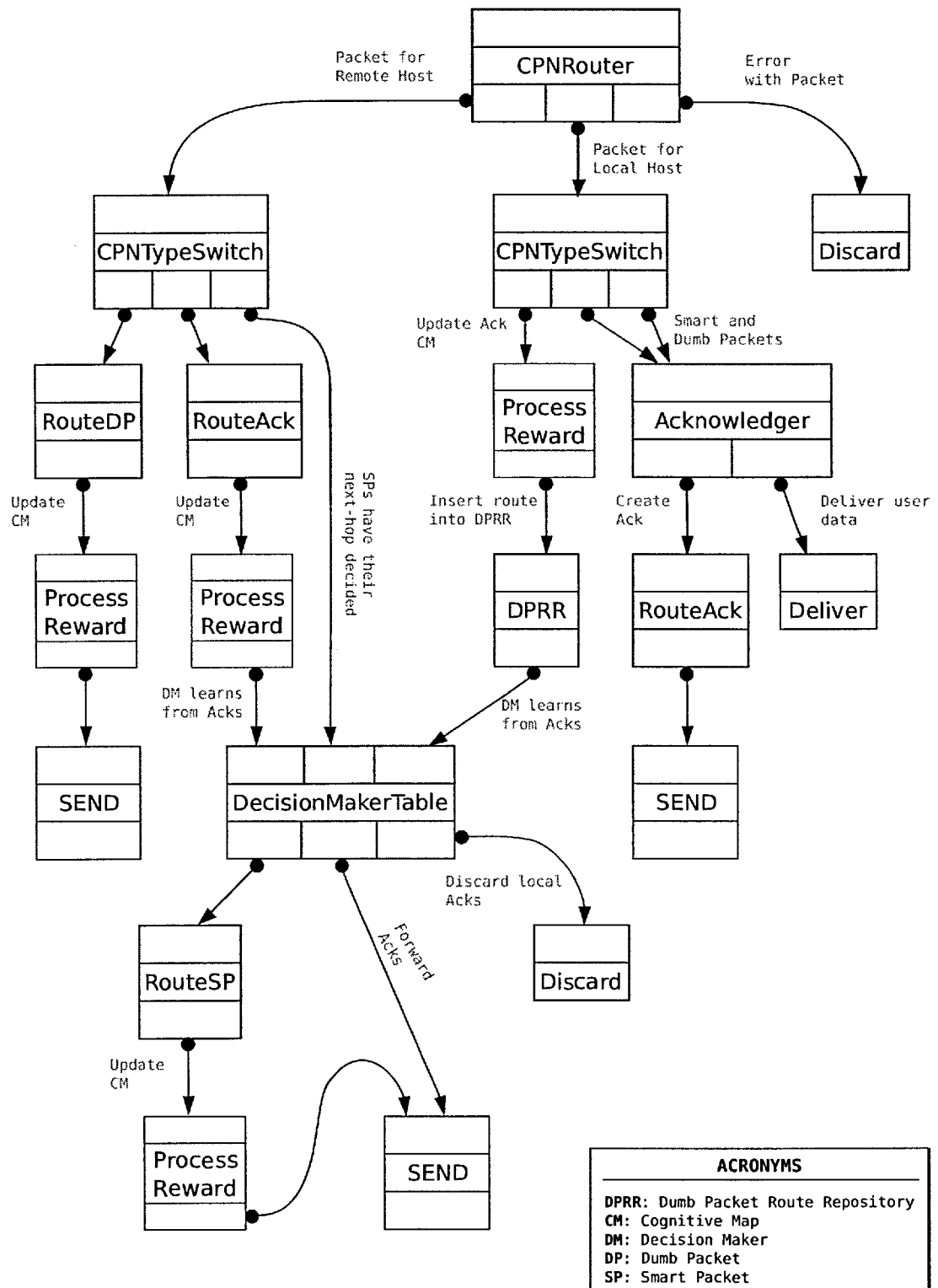


Figure 7.3: Routing a CPN packet in the overlay

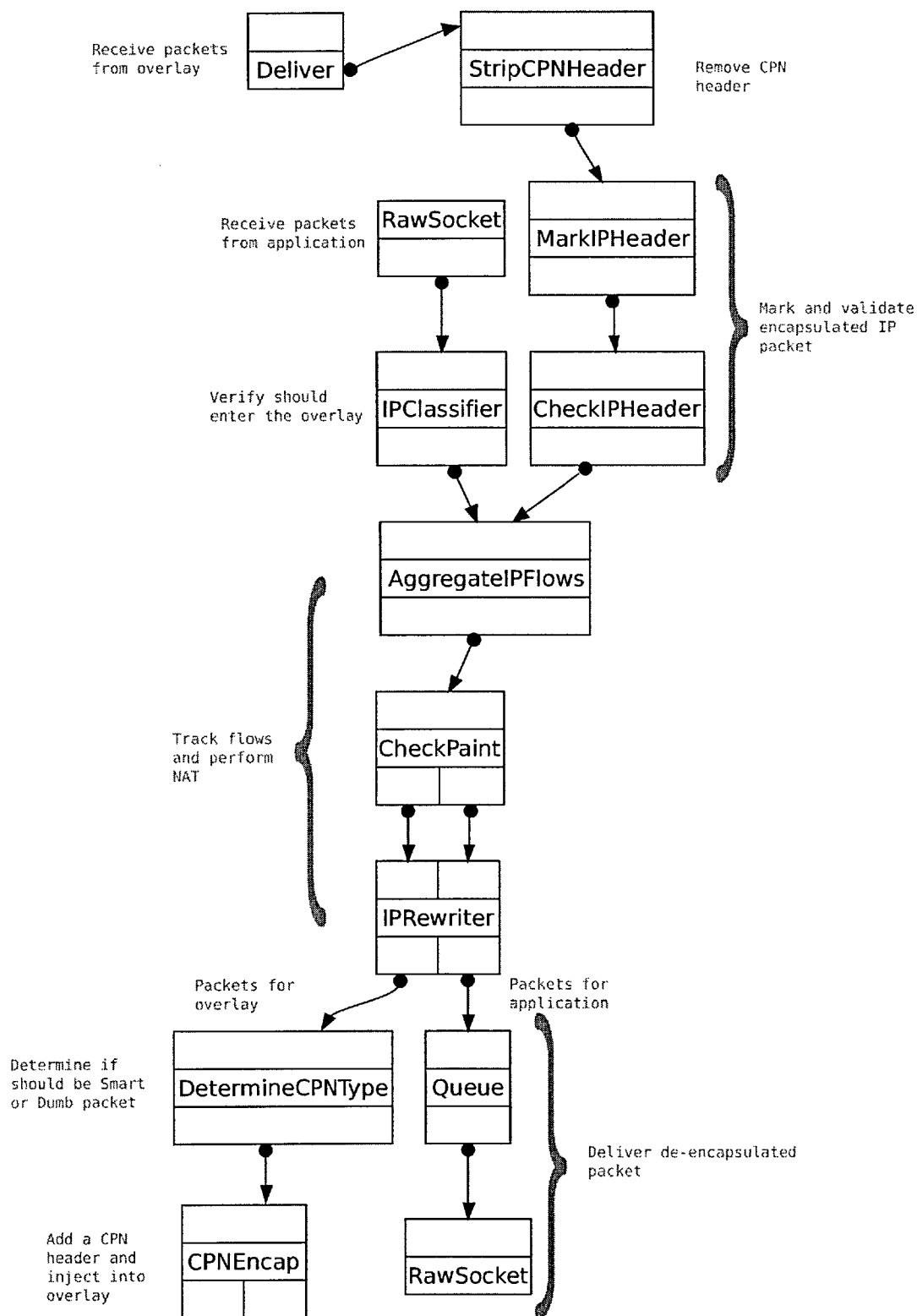


Figure 7.4: Set of actions an ingress server would take to utilize the overlay

to demonstrate that the routing overlay we have outlined is able to discover paths that are optimal in terms of an application's QoS needs.

In this section, we first describe the scenario under which our experiments were conducted. Following that, we demonstrate two critical aspects of our overlay: whether it is able to learn when it is best to use the underlying network; and that we can actually improve the QoS of traffic using the overlay when a better route exists than that offered by the underlying routing protocol.

7.4.1 Testbed

We believe that in order to generalize our conclusions to larger environments, our choice of network topology is important: for that reason we worked to find a topology that we could construct in the lab that resembled an existing network. As we discussed in Section 4.1.4, we contacted the Swiss Education and Research Network, who provided us details on their 46-router backbone, complete with bandwidth, OSPF costs, and link-level delays (Figure 4.2). We have implemented this topology in its entirety in our lab¹³ using off-the-shelf PCs running Linux 2.6.12 using 10/100 point-to-point Ethernet for the physical connections. We use OSPF as our IP routing protocol, achieved using quagga 0.99.3 (87). It uses the OSPF costs given to us; thus, the routes in our testbed are exactly the same as those used in the real Swiss backbone. In addition, because the OSPF costs are directly proportional to the artificial delays on each link the routes calculated between IP destinations will result in delay-optimal routing (disregarding the effects of congestion).

An overlay topology

After a physical and network-layer topology had been decided upon, the next question became how to structure the overlay topology. In the interests of repeatability, and being able to demonstrate the performance of our routing protocol in a stable environment, we opted to

¹³With the following differences: we have scaled the bandwidth down by a factor of 100, and have used artificial delays to replicate the physical layer delays in the data set (from 0-40ms).

not include the effect of overlay dynamics (i.e. nodes joining and leaving the overlay) in this evaluation. This is not to say that this is not an important parameter in an overlay's functioning; rather, we feel that the Gnutella membership protocol (discussed in Section 7.2) has been shown to handle dynamics very well in creating a stable overlay. In addition, our experiments in this section are on the order of 15 minutes in length, and we feel it is a reasonable assumption that the overlay should be moderately stable on time-scales of this nature.

Thus, we have manually (and statically) created an overlay topology for our routing protocol to run atop. The first step in creating the topology was to designate a set of ultrapeers. The percentage of ultrapeers in the Gnutella network is given (99) as 15% so our goal was to find a criterion that ensured the set of ultrapeers would be of a similar percentage. Using a node's degree in the topology graph as our criterion, choosing those with degree greater than 4 to be ultrapeers resulted in 13% of the nodes being made ultrapeers. This has the added advantage that high-degree nodes are made to be ultrapeers which is in line with their definition of being "stronger" than other nodes in the network.

After the assignment of ultrapeers, each remaining peer was assigned to a cluster using physical layer proximity. The result of this process can be seen in Figure 4.2. Each ultrapeer is shown as a red node, with each coloured region showing its set of ultrapeers. Each ultrapeer is fully connected at the overlay layer with each of the other ultrapeers, whereas each peer is only configured with one overlay neighbor (the ultrapeer in its cluster).

7.4.2 Scenario

Our experiments use the open-source Quake II multi-player game (which uses UDP as its transport protocol) as a traffic source. Due to Quake II's open-source nature, we were able to modify the application to better suit it to network experimentation which involved: removing all video, sound, and input code to allow clients to run without user input; and embedding *bot* code into the server so that our generated traffic resembles that of a real game complete with movement, player kills, and so forth. We use a single server (cpn008 in Figure 7.5), and

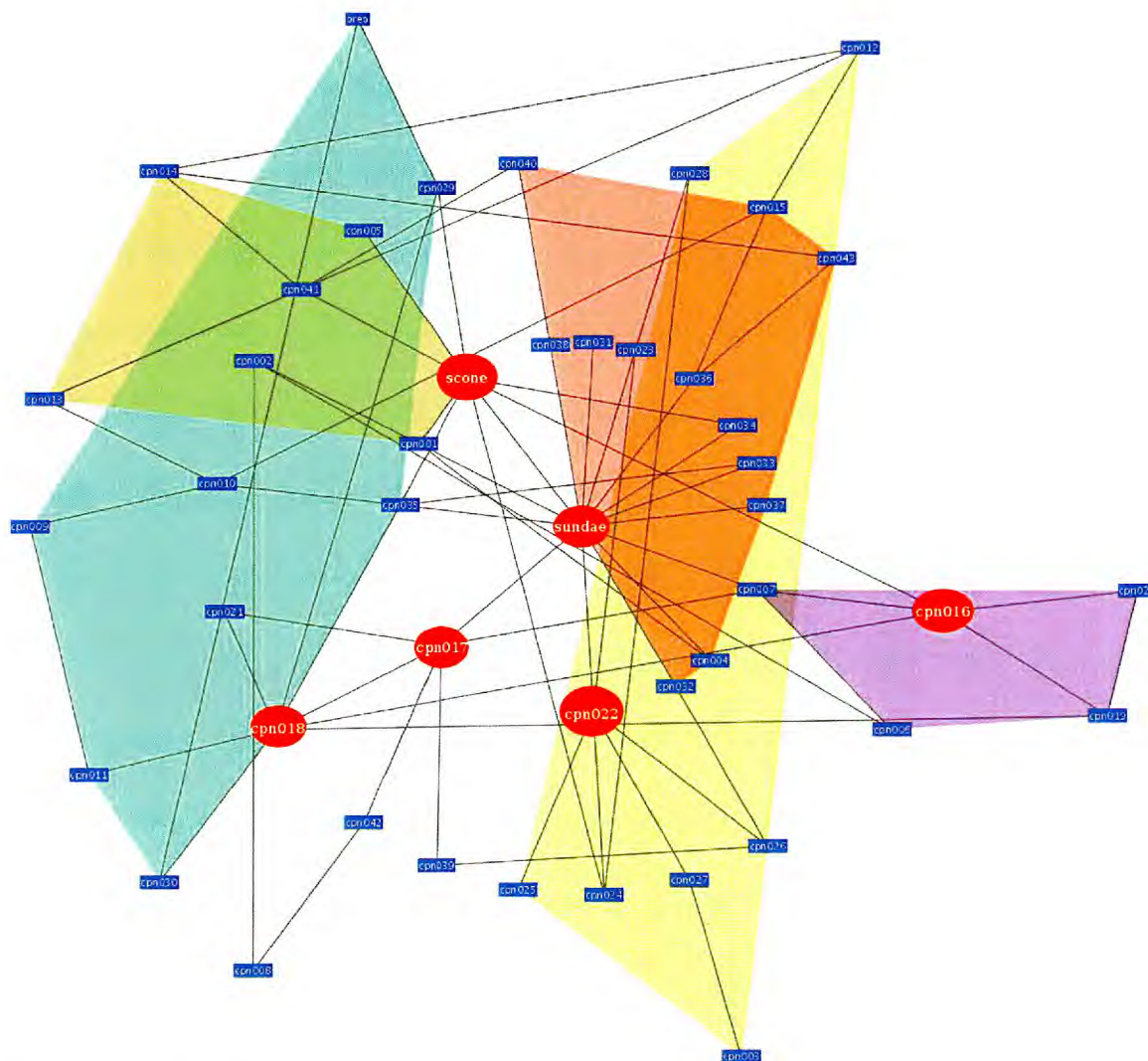


Figure 7.5: The 46-node testbed topology. Each of the 6 shaded clusters corresponds to a different overlay sub-network, where each peer is *logically connected* only to its ultrapeer (denoted by a circle), who are in turn connected to each other using a full mesh. Figure generated using GUESS (1).

Smart Packet Probability	0.2
Data encapsulated in SPs	Yes
Quake II Server Bots	8
Experiment Length	15 minutes
Repetitions	4

Table 7.1: Parameters for overlay experiment

a selection of clients from each remaining cluster (for a total of 9 clients).

In order to compare our overlay's performance with the underlay, we have used the *maximum number of overlay hops* (MAX_HOPS) as an independent variable which controls how many overlay hops may be visited by a Smart packet before it is forwarded to the destination. We emphasize that this is the *maximum* number of hops that may be traversed in the overlay - the actual number of hops used may be *learned* to be less, depending on network conditions. Recalling the discussion in Section 7.3.2, each overlay peer has an RNN decision maker with the number of neurons (*choices*) equal to the number of overlay connections plus an additional choice which is interpreted as the decision to choose the destination as the next (and final) hop. Thus, MAX_HOPS=0 means that the overlay receives the application's packet, encapsulates it in a CPN packet, and then forwards it directly to the destination without consulting the reinforcement learning algorithm for a routing decision. Thus, it gives an indication of the amount of processing overhead introduced by the overlay software because it will always use the route provided by the underlay. On the other hand, 2 hops allows the Smart packet to visit up to (but possibly less depending on the learning algorithm) two ultrapeers, thus fully exploring the possible routing options provided by this topology.

7.4.3 Results

Here, we present two different sets of results: the first demonstrates the overlay's ability to learn to use the underlay, while the second demonstrates how it can improve the QoS of the metric it is assigned to optimize.

The delay results that we report are the mean of application-level *ping* results as given by

the Quake II server. Each experiment lasts 15 minutes, and was repeated 4 times. Unless otherwise noted, there was minimal variance in the results.

Using the underlay when appropriate

The first experiment took place in a network where the underlying routing protocol (OSPF) computed a set of routes that are optimal in terms of delay (not including the effect of congestion). The goal of this experiment is to determine whether the CPN overlay can *learn not to use the overlay, and use the route provided by the underlying network*. We can verify that our overlay uses this route, as opposed to some other route, as additional overlay routing hops would increase the average delay unnecessarily.

The results are shown in Figure 7.6 where we can see that, excepting cpn020¹⁴ the use of the overlay had minimal effect on the perceived application-level delay. In fact, the difference between IP and using the overlay is statistically insignificant for many of the clients (using a 95% confidence interval), including for cpn024 where the delay *decreased* slightly when using the overlay.

The first conclusion that we can draw from these results is that the application-level overhead of our overlay is minimal. This value is the difference between the first and second bars, and is on the order of a few milliseconds at most in almost all cases.

The second conclusion that we draw is that the overlay indeed has learned to use the path provided by the underlying routing protocol. We see as the MAX_HOPS parameter increases (i.e. the overlay is given more freedom to try longer paths) the delay does not increase noticeably. If the routing overlay were instead routing a majority of packets using overlay hops not on the optimal path, we would instead see the delay steadily increase as it was given the ability to use longer and longer paths. Because we do not see this phenomenon, we can conclude that the path being used by the overlay has equal quality to that of the underlying

¹⁴This client appears as an outlier in our experiment. The fact that the difference between 0 overlay hops and 2 is negligible implies that it is not using additional underlay hops. However, the 10ms increase between the underlay and overlay cannot be explained by overhead alone. We include it in the data set for completeness, but do not feel that it impedes our ability to generalize our results.

network.

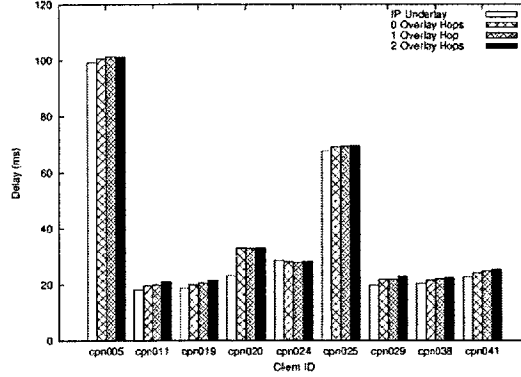


Figure 7.6: This graph shows that even as the overlay has more freedom to explore the underlying network (shown by the increase in the number of hops) it is still able to learn to use the optimal routes provided by the underlay (the difference between the second, third, and fourth bar for each client is small to none). We can also see that the overhead of using our overlay is quite small (shown by the difference between the first and second bar).

Finding Better Routes than those Given by the Underlay

Our second experiment's goal was to measure the overlay's ability to route around a *hotspot* in the underlying network. To create the hotspot, we increased the artificial delay on one link (connecting node 17 and 42) to 100ms. As shown in Figure 7.7, this affected 4 clients who used this link to reach the server.

The hotspot affected clients until the routing overlay was given enough freedom to explore the full ultrapeer mesh (MAX_HOPS=2). In this scenario, each client initially is affected by the hotspot, but it is periodically probing the overlay for better paths. As this probing process is a combination of learned behaviour and (semi-random) exploration, it just takes one Smart Packet to discover an ultrapeer who has an alternate underlying route that avoids the hotspot. As soon as that happens, an Acknowledgement packet will be generated which rewards the discovered route, and the overlay will begin to use the new path. This entire procedure took less than a minute in our experiment for each client, but this time is mainly a function of the Smart Packet probability (0.2 in our experiment). Decreasing this probability

will discover paths more slowly, but will impose less overhead on the network. On the other hand, increasing the probability too much increases measurement overhead, and may also decrease application QoS as some data packets are embedded in Smart Packets, which may explore areas of the overlay that are poorer in quality.

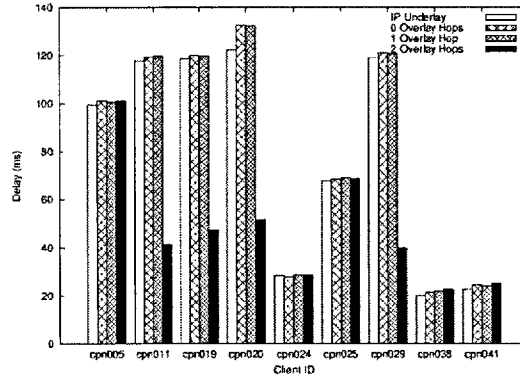


Figure 7.7: When there exists a better route than provided by the underlay, the overlay finds it, and learns to use it (clients 11, 19, 20, 29).

7.5 Future Work

This work has introduced a scalable, adaptive method for structuring a routing overlay based on peer-to-peer principles. Our initial prototype of this idea has demonstrated a number of positive qualities inherent in the approach. However, this could be expanded upon in a number of ways.

The next logical step for this work would be to expand its scale and scope to consider network dynamics. While we used 46 routers, this could still be greatly scaled up. One attractive avenue for the next stage would be the PlanetLab distributed testbed, which encompasses over 500 nodes distributed worldwide among different commercial, governmental and educational campuses. These have the advantage that they allow researchers to load their own code, making it an excellent testbed for networking experimentation. In addition to the size of the testbed, it also introduces a great deal of dynamics that were missing from our initial experiments, as these nodes connect to each other using Internet links.

Another interesting avenue for exploration would be to examine how the underlying routing protocol impacts overlay performance. If we ran similar experiments as those in this chapter using CPN to calculate underlying routes, we could then compare the performance of pure IP against a CPN overlay against full CPN, perhaps leading to interesting conclusions about both our overlay, and CPN as a routing protocol.

7.6 Summary and Conclusions

In this chapter, we have presented the steps we have taken for implementing the CPN algorithm in a routing overlay. This is an exciting concept as it allows applications the benefits shown for CPN routing without the need to completely replace an entrenched routing infrastructure. However, it is not a simple matter of simply taking the existing implementations and running them over IP, rather it requires a few new ways of looking at the problem, as we have discussed.

The first major difficulty we have outlined is in how to structure the overlay in a scalable manner that deals with nodes leaving and joining. Towards this goal we have highlighted an existing real-world Peer-to-Peer overlay, Gnutella, and shown how it has met these challenges and more, resulting in a stable overlay of (currently) more than 3 million participants. We have looked at its underlying membership and overlay maintenance protocol and concluded that it would be feasible for us to use this to construct our routing overlay.

Turning our attention away from the overlay construction problem, we then tackled the issue of adapting the CPN algorithm to run over an overlay. We first decided that leaf peers should not be responsible for routing traffic, thus limiting the scale of the routing problem. We also integrated into the routing process the idea that it must be possible to learn when to exit the overlay, and that this decision should always be an option to reduce the chances of the overlay hindering an application's performance.

Using this as a base, we presented our current implementation of the CPN routing overlay in the Click modular router. We have tested this implementation under various operating

conditions, and with multiple applications to ensure that the approach works with many real-world applications in use today. Our conclusion is that any UDP-based client/server architecture would be able to make use of our routing overlay.

Using this implementation, we conducted experiments using our local testbed to answer two questions of the routing overlay's performance. First, we asked whether the overlay could learn to use the underlying routes as opposed to the overlay when appropriate. We then asked if the overlay could improve upon the QoS offered by the underlying routes when such a route exists. In both cases we found the answer to be yes, providing strong support for our implementation and architecture, as the processing overhead of the approach was found to be minimal.

Chapter 8

Conclusion and Future Work

8.1 Summary

This thesis has explored CPN's ability to route real-time traffic according to its performance needs. This has encompassed a large breadth of work, which started with re-implementing the CPN protocol and experimenting with how it compared with a standard routing protocol; continuing to an exploration of the effect that multiple real-time flows have on each other and how to control the presence and effect of routing oscillations, and then showing; how we can use CPN's reinforcement learning-based routing mechanism to easily switch the way in which CPN chooses how to route flows in the network. Concluding the thesis, we explored and experimented with a prototype implementation of a method for extending CPN to a large-scale network with large amounts of dynamics without replacing the underlying routing infrastructure.

We began by extending the existing implementation of CPN to make it more robust, more easy to maintain, and more friendly to existing networked applications that use IP as their network protocol. This formed the basis of our experimental work – without this step we would not have been able to easily experiment with multiple flows, multiple applications, and multiple topologies. It removed many of the performance problems of the previous implementation (both in terms of code structure and memory utilization), and provided a robust framework

for working with some of the many *real* applications that have been shown to work across CPN.

After the implementation was completed, a natural question became how CPN compared with the state-of-the-art, both in terms of overhead and performance. We set out to test this with a real multi-player game with multiple clients connecting to a single server. The results of this work were very promising: we showed that not only does CPN find optimal routes in the network without any a priori knowledge, we also were able to demonstrate that the processing overhead and route exploration overhead is minimal. Furthermore, we also showed the benefits of CPN's route overhead when we explored the impact of network dynamics – when the optimal route in the network suffered from performance degradation, CPN was very quickly able to route around it.

During these experiments with multiple flows, we noticed that the CPN flows did not behave independently – their traffic influenced each other, causing many routing oscillations. We explored these oscillations, seeing their impact on real-time flows. The results of this work were very interesting, bringing into question whether some long-held views on the impact of routing oscillations on load-sensitive metrics hold on a protocol like CPN. We showed that even though CPN can frequently switch routes, this does not have a severe impact on its performance. In addition, we also developed two controls which impact the oscillation rate, and the quality of the routes provided which allow CPN to trade-off between performance and switching. Even though we were able to demonstrate the lack of a performance penalty from switching, we did show that the amount of desequencing that this introduces has to be taken into account, and has to be weighed against the improvement in quality that is seen.

Switching focus slightly from work with multiple flows to work with multiple QoS metrics, we set out to show that CPN could optimize metrics which most impact a flow's performance using loss and delay which are both metrics which greatly impact the performance of a real-time flow. This showed the benefit of CPN's flexible reinforcement learning-based approach to evaluating and choosing routes. By changing the evaluation criteria (which is simply a goal function), we showed that CPN switches the way it utilizes routes and is able to discover

(and take advantage of) routes which best fulfil the reward criteria.

The work that we completed up to this point was always in the laboratory, and always assumed that the only networking protocol available was CPN – our final goal was to determine if there was a way to obtain the benefits of CPN routing without calling for the replacement of the dominant routing protocol used today, IP. With this in mind, we looked to overlay networks, especially unstructured peer-to-peer networks which have already scaled to many millions of participants in the face of unpredictable network dynamics. The networks demonstrate a way to use CPN-like routing without removing the underlying network infrastructure. Using this idea, we constructed an overlay topology in the laboratory and were able to demonstrate that without modifying the IP routing tables, we could route around a performance fault, with a very low degree of overhead.

8.2 Future Work

The work of chapter 6 could be expanded upon to include more than two QoS metrics. As we discussed in chapter 2, there are a variety of metrics that impact application performance and these are but two. This approach was already used by Gelenbe, et al in (40) to route based on power and latency with promising results. However, it is unclear how routes could be selected based on three or more metrics. Environments such as mobile ad-hoc networks, sensor networks, or inter-planetary networks typically have many metrics that must be taken into account when routing and a generic way of combining an arbitrary number of metrics would go a long way towards enabling CPN routing in these environments.

Additionally, the loss calculations proposed in chapter 6 could be expanded upon to make them more robust to asynchronous losses by using the counters on both edges of a link to better estimate the drop rate. One method which could be used which we be resilient to many forms of cheating is proposed in (34).

Our work has always used Random Neural Network based decision making for routing, with very positive results; yet it is still unclear that this is the only, or indeed the best method.

Combining CPN decision making with other more complicated approaches was looked at in (37) (genetic algorithms) which pointed out that the decision-making algorithms must operate in a timely manner so that their decisions are still relevant for the prevailing network conditions. Looking at the impact of the decision maker is left for future research.

We showed that oscillations can lead to a high degree of desequencing, while still having good performance. Designing transport protocols that can deal with a high level of desequencing would allow these applications to take advantage of CPN's performance gains while not suffering any performance penalty from this frequent switching.

The overlay construction mechanism discussed in chapter 7 was prototyped in a laboratory setting, but has not yet been experimented with in a large scale environment. One way to more fully demonstrate the benefits of the approach would be to use the distributed testbed PlanetLab which consists of a number of geographically distributed PCs of varying Internet connectivity. Using these nodes would allow for larger-scale, more realistic experiments than we have been able to conduct in the laboratory. The only down-side of using PlanetLab is that it is a popular resource with the research community and many of the nodes are overloaded.

Bibliography

- [1] Eytan Adar. Guess: a language and interface for graph exploration. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 791–800, New York, NY, USA, 2006. ACM Press.
- [2] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. *Operating Systems Review (ACM)*, 35(5):131–145, Dec 2001.
- [3] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne. Real-time communication in packet-switched networks. *Proceedings of the IEEE*, 82(1):122–139, January 1994.
- [4] M. D’Arienzo, E. Gelenbe, and G. Sakellari. Admission control in self aware networks. In *49th annual IEEE Global Telecommunications Conference, GLOBECOM 2006, San Francisco*, November 2006.
- [5] T.W. Athan and P.Y. Papalambros. A note on weighted criteria methods for compromise solutions in multi-objective optimization. *Engineering Optimization*, 27:155–176, 1996.
- [6] S. Avallone, F.A. Kuipers, G. Ventre, and P. Van Mieghem. Dynamic routing in QoS-aware traffic engineered networks. In *Proc. of Eunice 2005*, 2005.
- [7] T. Banka, A.A. Bare, and A.P. Jayasumana. Metrics for degree of reordering in packet sequences. In *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on*, pages 333–342, November 2002.
- [8] Nigel Bean and Andre Costa. An analytic modeling approach for network routing

- algorithms that use “ant-like” mobile agents. *Computer Networks*, 2005. Article in Press.
- [9] Jon C. R. Bennett, Craig Partridge, and Nicholas Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Netw.*, 7(6):789–798, 1999.
- [10] S. Blake, D. Black, M. Carlson, E. Davis, Z Wang, and W. Weiss. An architecture for differentiated services. IETF, RFC 2475, December 1998.
- [11] S. Bohacek, J.P. Hespanha, Junsoo Lee, C. Lim, and K. Obraczka. A new TCP for persistent packet reordering. *IEEE/ACM Transactions on Networking*, 14(2):369–382, April 2006.
- [12] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Théréaluz. Routing in telecommunication networks with ‘smart’ ant-like agents. In *Proceeding of IATA’98: Second Int. Workshop on Intelligent Agents for Telecommunication Applications*, volume 1437 of *Lecture Notes in AI*. Springer Verlag, 1998.
- [13] R. Braden, D. Clark, and S. Schenker. Integrated services in the internet architecture: an overview. IETF, RFC 1633, June 1994.
- [14] G. Di Caro and M Dorigo. Mobile agents for adaptive routing. In *Proceedings of the 31st Hawaii international conference on system sciences*, pages 74–83, 1998.
- [15] G. Di Caro and M. Dorigo. Two ant colony algorithms for best-effort routing in datagram networks. In *Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 98)*, page 541?546. IASTED/ACTA Press, 1998.
- [16] Miguel Castro, Manuel Costa, and Antony Rowstron. Peer-to-peer overlays: structured, unstructured, or both? Technical Report MSR-TR-2004-73, Microsoft Research, Cambridge, 2004.
- [17] Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott J. Shenker, and Walter Will-

- inger. Towards capturing representative AS-level internet topologies. *Computer Networks*, 44(6):737–755, apr 2004.
- [18] Hyunseok Chang, Sugih Jamin, and Walter Willinger. To peer or not to peer: Modeling the evolution of the internet’s as-level topology. In *Proceedings of IEEE Infocom*, 2006.
- [19] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM ’03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, New York, NY, USA, 2003. ACM Press.
- [20] S. Chen and K. Nahrstedt. On finding multiconstrained paths. In *Proc. IEEE Int. Conf. Communications (ICC’98)*, pages 874–879, 1998.
- [21] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *SIGCOMM*, pages 14–26, 1992.
- [22] David Corne, Marco Dorigo, and Fred Glover, editors. *New Ideas in Optimization*. McGraw Hill, 1999.
- [23] I. Das and J. Dennis. A closer look at drawbacks of minimizing weighted sums of objective for pareto set generation in multicriteria optimization problems. *Structural Optimization*, 14:63–69, 1997.
- [24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
- [25] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.
- [26] Frederick Ducatelle, Gianni Di Caro, and Luca Maria Gambardella. Ant agents for hybrid multipath routing in mobile ad hoc networks. In *Proceedings of the Second Annual*

- Conference on Wireless On-demand Network Systems and Services (WONS'05)*, 2005.
- [27] Gerald Eichler, Fabio Ricciato, Stefano Salsano, Anne Thomas, and Ralf Widera. AQUILA network services. In *8th International Conference on Advances in Communications and Control*, Crete, Greece, jun 2001.
- [28] Amir Hassan Rasti Ekbatani, Daniel Stutzbach, and Reza Rejaie. On the long-term evolution of the two-tier gnutella overlay. In *9th IEEE Global Internet Symposium*, April 2006.
- [29] M. El-Gendy, A. Bose, and K. Shin. Evolution of the internet QoS and support for soft real-time applications. *Proceedings of the IEEE*, 91:1086–1104, 2003.
- [30] D. Ferrari. Client requirements for real-time communication services. *IEEE Communications Magazine*, 28(11):65–72, November 1990.
- [31] Sally Floyd and Van Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Trans. Netw.*, 2(2):122–136, 1994.
- [32] Ruomei Gao, Contantine Dovrolis, and Ellen Zegura. Avoiding oscillations due to intelligent route control systems. In *IEEE Infocom*, 2006.
- [33] Carlos García-Martínez, Oscar Cordon, and Francisco Herrera. An empirical analysis of multiple criteria ant colony optimization algorithms for the bi-criteria TSP. In *ANTS 2004*, 2004.
- [34] Ignacy Gawedzki and Khaldoun Al Agha. Proactive resilience to dropping nodes in mobile ad hoc networks. In *AINTEC*, volume 4311 of *Lecture Notes in Computer Science*, pages 139–158. Springer, 2006.
- [35] E. Gelenbe. Learning in the recurrent random neural network. *Neural Computation*, 5:154–164, 1993.
- [36] E. Gelenbe, Şeref, and Z. Xu. Simulation with learning agents. *Proceedings of the IEEE*, 89:148–157, 2001.

- [37] E. Gelenbe, M. Gellman, R. Lent, P. Liu, and Pu Su. Autonomous smart routing for network QoS. In *Proceedings of the First International Conference on Autonomic Computing*, pages 232–239, New York, NY, May 2004.
- [38] E. Gelenbe, M. Gellman, and G. Loukas. Defending networks against denial of service attacks. In E.M. Carapezza, editor, *Proceedings of the Conference on Optics/Photonics in Security and Defence (SPIE), Unmanned/Unattended Sensors and Sensor Networks*, volume 5611, pages 233–243, London, UK, October 2004.
- [39] E. Gelenbe and K. Hussain. Learning in the multiple class random neural network. *IEEE Transactions on Neural Networks*, 13(6):1257–1267, November 2002.
- [40] E. Gelenbe and R. Lent. Power-aware ad hoc cognitive packet networks. *Ad Hoc Networks Journal*, 2(3):205–216, July 2004.
- [41] E. Gelenbe, R. Lent, A. Montuori, and Z. Xu. Cognitive packet networks: QoS and performance. In *Proceedings of the IEEE MASCOTS Conference*, pages 3–12, Ft. Worth, TX, Oct 2002. Opening Keynote Paper.
- [42] E. Gelenbe, R. Lent, A. Montuori, and Z. Xu. Towards networks with cognitive packets. In K. Goto, T. Hasegawa, H. Takagi, and Y. Takahashi, editors, *Proceedings of the International Conference on Performance and QoS of Next Generation Networking*, pages 3–17, Nagoya, Japan, Nov 2002. Springer Verlag. Opening Invited Paper.
- [43] E. Gelenbe, R. Lent, and A. Nunez. Smart WWW traffic balancing. In *Proceedings of the IEEE Comm. Society Workshop on IP Operations and Management*, pages 15–22, Oct 2003.
- [44] E. Gelenbe, R. Lent, and A. Nunez. Self-aware networks and QoS. *Proceedings of the IEEE*, 92(9):1478–1489, September 2004.
- [45] E. Gelenbe, R. Lent, and Z. Xu. Towards networks with cognitive packets. In *Proc. IEEE MASCOTS Conference*, San Francisco, Sep 2000.

- [46] E. Gelenbe, R. Lent, and Z. Xu. Measurement and performance of a cognitive packet network. *Journal of Computer Networks*, 37:691–791, 2001.
- [47] E. Gelenbe and P. Liu. Qos and routing in the cognitive packet network. In *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Jun 2005.
- [48] E. Gelenbe, P. Liu, and J. Laine. Genetic algorithms for autonomic route discovery. In *Proceedings of IEEE 2006 Workshop on Distributed Intelligent Systems*, June 2006.
- [49] E. Gelenbe, Z.-H. Mao, and Y. Da-Li. Function approximation with spiked random networks. *IEEE Transactions on Neural Networks*, 10:3–9, 1999.
- [50] E. Gelenbe, E. Seref, and Z. Xu. Towards networks with intelligent packets. In *Proc. IEEE-ICTAI Conference on Tools for Artificial Intelligence*, Chicago, Nov 1999.
- [51] M. Gellman and P. Liu. Random neural networks for the adaptive control of packet networks. In *Submitted for publication, International Conference on Artificial Neural Networks, Athens*, September 2006.
- [52] Gnutella. <http://www.gnutella.com>.
- [53] Krishna P Gummadi, Harsha V Madhyastha, Steven D Gribble, Henry M Levy, and David Wetherall. Improving the reliability of internet paths with one-hop source routing. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, 2004.
- [54] Oliver Heckmann, Michael Piring, Jens Schmitt, and Ralf Steinmetz. On realistic network topologies for simulation. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 28–32, New York, NY, USA, 2003. ACM Press.
- [55] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

- [56] Shun-Yun Hu and Guan-Ming Liao. Scalable peer-to-peer networked virtual environment. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 129–133, New York, NY, USA, 2004. ACM Press.
- [57] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 295–308, New York, NY, USA, 2002. ACM Press.
- [58] Kazaa. <http://www.kazaa.com>.
- [59] Frank Kelly. *Mathematical Modelling of the Internet*, pages 685–702. Springer, 2001.
- [60] Frank Kelly. Fairness and stability of end-to-end congestion control. In *European Journal of Control*, number 9, pages 159–176, 2003.
- [61] R. Keralapura, C.-N. Chuah, N. Taft, and G. Iannaccone. Can coexisting overlays inadvertently step on each other? In *Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on*, 2005.
- [62] A. Khanna and J. Zinky. The revised arpanet routing metric. *SIGCOMM Comput. Commun. Rev.*, 19(4):45–56, 1989.
- [63] Hendrik Knoche. Übersetzung von dienstguteparametern in verteilten multimediadiensten unter berucksichtigung der menschlichen wahrnehmung. Master’s thesis, U. Hamburg, 1999. Obtained through private correspondence.
- [64] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.
- [65] J. Koski. Defectiveness of weighting methods in multicriterion optimization of structures. *Communications in Applied Numerical Methods*, 1:333–337, 1985.
- [66] Akio Koyama, Leonard Barolli, Kazunori Matsumoto, and Bernady O. Apduhan. A GA-based multi-purpose optimization algorithm for QoS routing. In *18th International Conference on Advanced Information Networking and Applications (AINA'04)*, 2004.

- [67] T. Korkmaz M.M. Krunz. Routing multimedia traffic with QoS guarantees. *IEEE Transactions on Multimedia*, 5(3):429–443, 2003.
- [68] F. A. Kuipers and P Van Mieghem. The impact of correlated link weights on QoS routing. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, 2003.
- [69] F.A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem. Performance evaluation of constraint-based path selection algorithms. *IEEE Network*, 18(5):16–23, 2004.
- [70] C. Labovitz, G.R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6(5):515–528, 1998.
- [71] C. Labovitz, G.R. Malan, and F. Jahanian. Origins of internet routing instability. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 218–226, New York, NY, 1999.
- [72] P. Liu and E. Gelenbe. Recursive routing in the cognitive packet network. In *Proceedings of 3rd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, May 2007.
- [73] Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72–93, 2005.
- [74] R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.
- [75] S. Marwaha, D. Srinivasan, C K Tham, and A Vasilakos. Evolutionary fuzzy multi-objective routing for wireless mobile ad hoc networks. In *IEEE Congress on Evolutionary Computation*, 2004.
- [76] Christopher Mattson and Achille Messac. Pareto frontier based concept selection under uncertainty, with visualization. *Optimization and Engineering*, 6:85–115, 2005.

- [77] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. *SIGCOMM Comput. Commun. Rev.*, 30(2):18–28, 2000.
- [78] P Van Mieghem and F. A. Kuipers. Concepts of exact QoS routing algorithms. *IEEE/ACM Transactions on Networking*, 12:851 – 864, 2004.
- [79] D. Miras. A survey on network QoS needs of advanced internet applications, 2002.
- [80] A. A. Mullur, C. A. Mattson, and A. Messac. Pitfalls of the typical construction of decision matrices. In *AIAA 41th Aerospace Sciences Meeting and Exhibit*, number AIAA-2003-0466,, 2003.
- [81] Akihiro Nakao, Larry Peterson, and Andy Bavier. Scalable routing overlay networks. *SIGOPS Oper. Syst. Rev.*, 40(1):49–61, 2006.
- [82] M.S. Osman, M.A. Abo-Sinna, and A.A. Mousa. An effective genetic algorithm approach to multiobjective routing problems (MORPs). *Applied Mathematics and Computation*, 163:769–781, 2005.
- [83] V Pareto. *Manuale di Economica Politica, Societa Editrice Libreria*. Macmillan, Milan, 1906. translated into English by A.S. Schwier as Manual of Polical Economy, edited by A.S. Schwier and A.N. Page, 1971. New York: A.M. Kelley.
- [84] Larry Paterson and Timothy Roscoe. The Design Principles of PlanetLab. Technical Report PDN-04-021, PlanetLab Consortium, June 2004.
- [85] Vern Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Trans. Netw.*, 5(5):601–615, 1997.
- [86] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.
- [87] Quagga. <http://www.quagga.net>.

- [88] U. Ranadive and D. Medhi. Some observations on the effect of route fluctuation and network link failure on TCP. In *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 460–467, Scottsdale, AZ, 2001.
- [89] D. E. Rumelhart and J. L. McClelland. *Parallel distributed processing Vols. I and II*. Bradford Books and MIT Press, 1986.
- [90] Stefan Savage, Thomas Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: Informed internet routing and transport. *IEEE Micro*, 19(1):50–59, 1999.
- [91] Aman Shaikh, Anujan Varma, Lampros Kalampoukas, and Rohit Dube. Routing stability in congested networks: experimentation and analysis. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 163–174, New York, NY, USA, 2000. ACM Press.
- [92] A. Shriram and J. Kaur. Empirical study of the impact of sampling timescales and strategies on measurement of available bandwidth. In *PAM*, March 2006.
- [93] Alok Shriram, Margaret Murray, Young Hyun, Nevil Brownlee, Andre Broido, Marina Fomenkov, and Kimberly C. Claffy. Comparison of public end-to-end bandwidth estimation tools on high-speed links. In *PAM*, pages 306–320, 2005.
- [94] Kwang Mong Sim and Weng Hong Sun. Multiple ant-colony optimization for network routing. In *Proceedings of the First International Symposium on Cyber Worlds*, 2002.
- [95] Kwang Mong Sim and Weng Hong Sun. Ant colony optimization for routing and load-balancing: Survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics*, 33(5):560–572, 2003.
- [96] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, February 2004.
- [97] J.A. Stankovic, T.E. Abdelzaher, Chenyang Lu, Lui Sha, and J.C. Hou. Real-time com-

- munication and coordination in embedded sensor networks. *Proceedings of the IEEE*, 91(7):1002–1022, July 2003.
- [98] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM Press.
- [99] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In *Internet Measurement Conference*, 2005.
- [100] R. S. Sutton. Learning to predict the methods of temporal difference. *Machine Learning*, 3:9–44, 1988.
- [101] TIERS, 2007. TIERS. Tiers topology generator. <http://www.isi.edu/nsnam/ns/ns-topogen.html>.
- [102] Ellen W. Zegura, Kenneth L. Calvert, and Michael J. Donahoo. A quantitative comparison of graph-based models for internet topology. *IEEE/ACM Trans. Netw.*, 5(6):770–783, 1997.