INAUGURAL - DISSERTATION

zur
Erlangung der Doktorwürde

der
Naturwissenschaftlich-Mathematischen Gesamtfakultät

der
Ruprecht - Karls - Universität
Heidelberg

vorgelegt von	Diplom-Informatik **Ali Al-Shabibi** aus Baghdad, Irak
Tag der mündlichen Prüfung	6. Juli 2011

MultiPaths Revisited -
A novel approach using OpenFlow-enabled devices

Ali AL-SHABIBI

Gutachter: Prof. Dr. Volker Lindenstruth
Prof. Dr. Ulrich Brüning

To my Iraqi family, the ones I know,
the ones I will never know,
and the ones I have yet to meet.

Acknowledgements

The road has been rather long — not to mention somewhat winding.

Over the past three years, it has been my good fortune to have and met many friends/people who saw past the whining and still decided to stick around and listen to me grovel about my thesis ad eternum. They have given me their time, companionship, professional and personal help, and above all: patience than was perhaps warranted by my seeming determination to indefinitely position the deadline for finishing this thesis at "next year".

I would first like to thank my advisor, Volker Lindenstruth, who had the courage to take me on as a Ph.D student even though he did not yet know me very well. Even though, he has a busy schedule he still found time to listen to my problems and advise me appropriately.

Brian Martin was simply instrumental to the completion of this thesis. He, unfortunately for him, had to share an office with me. As if that isn't already torture enough, he graciously sacrificed himself to correcting, proof reading, and criticizing my articles and this thesis. His finely tuned BS-detector is about as sensitive as the ATLAS detector on the LHC and without it this thesis would have been full of it. I shall be eternally indebted to him.

I would also like to thank my fellow researchers and colleagues. Catalin Meirosu's support and watchful eye as I was embarking on the implementation phase of this thesis. Stefan Stancu for his help in deploying my testbed. Michael Levine's critical encouragements in moments of doubt.

Rather than listing everyone, I wish to send out an aggregated thank you to all my family and friends without whom this thesis would have never been possible. Thank you all for the support and the availability to follow me into any random place I thought was "cool". I would also like to give a special shout out to the Bums, who have always been there for me whenever I needed

them. I am truly honoured to call these people my friends.

I am, of course, particularly indebted to my parents and my brother Ridha for their monumental, unwavering support and encouragement on all fronts. They have truly always been there for me, and without them none of this would have been even remotely possible.

Ali AL-SHABIBI
Geneva, December 2010

Abstract

This thesis presents novel approaches enhancing the performance of computer networks using multipaths. Our enhancements take the form of congestion-aware routing protocols. We present three protocols called MultiRoute, Step-Route, and finally PathRoute. Each of these protocols leverage both local and remote congestion statistics and build different representations (or views) of the network congestion by using an innovative representation of congestion for router-router links. These congestion statistics are then distributed via an aggregation protocol to other routers in the network.

For many years, multipath routing protocols have only been used in simple situations, such as Link Aggregation and/or networks where paths of equal cost (and therefore equal delay) exist. But, paths of unequal costs are often discarded to the benefit of shortest path only routing because it is known that paths of unequal length present different delays and therefore cause out of order packets which cause catastrophic network performances. Further, multipaths become highly beneficial when alternative paths are selected based on the network congestion. But, no realistic solution has been proposed for congestion-aware multipath networks. We present in this thesis a method which selects alternative paths based on network congestion and completely avoids the issue of out of order packets by grouping packets into flows and binding them to a single path for a limited duration.

The implementation of these protocols relies heavily on OpenFlow and NOX. OpenFlow enables network researchers to control the behavior of their network equipment by specifying rules in the routers flow table. NOX provides a simple Application Programming Interface (API) to program a routers flow table. Therefore by using OpenFlow and NOX, we are able to define new routing protocols like the ones which we will present in this thesis.

We show in this thesis that grouping packets together, while not optimal,

still provides a significant increase in network performance. More precisely we show that our protocols can, in some cases, achieve up to N times the throughput of Shortest Path (SP), where N is the number of distinct paths of identical throughput from source to destination. We also show that our protocols provide more predictable throughput than simple hash-based routing algorithms.

Todays networks provide more and more connections between any source-destination pair. Most of these connections remain idle until some failure occurs. Using the protocols proposed in this thesis, networks could leverage the added bandwidth provided by these currently idle connections. Therefore, we could increase the overall performance of current networks without replacing the existing hardware.

Diese Arbeit präsentiert neuartige Ansätze, um die Leistung von Computernetzwerken durchmultipathszu verbessern. Unsere Verbesserungen haben die Form von congestionaware routing protocols. Wir präsentieren drei Protokolle mit den Bezeichnungen MultiRoute, Step-Route und nally PathRoute. Jede dieser Protokolle lokale und entfernte Verkehrsstatistiken und bilden Repräsentierungen (oder Abbildungen) des Netzwerkverkehrs durch das Nutzen einer innovativen Representation des Verkehrsaufkommens von Router-Router Verbindungen. Diese Verkehrsstatistiken werden dann durch ein Aggregationsprotokoll zu anderen Routern im Netzwerk verteilt.

Lange Jahre wurden multipath routing Protokolle nur in einfachen Situationen, so wie Link Aggregation benutzt und/oder Netzwerken bei denen Pfade mit selben Kosten (und deshalb die selben Verzögerungen) vorherrschen. Jedoch sind Pfade mit unterschiedlichen Kosten oftmals zumnutzen des kürzesten und einzigroutenden Pfades ausgesondert. Dies geschieht da es bekannt ist, dass Pfade von unterschiedlichen Längenanderezeigen und deshalb katastrophale Netzwerk Leistungen bewirken. Desweiteren werden multipaths hochgradig nützlich, wenn alternative Pfade basierend auf Netzwerkauslastung ausgesucht werden. Jedoch wurde keine realistische Lösung für auslastungsbewusste multipath Netzwerke. Wir presentieren in dieser Arbeit eine Methode, die alternative Pfade basierend auf Netzwerkauslastung selektiert und das Problem von out-of-order Packeten durch Gruppierung von Packeten in fliesst umgeht und sie für eine begrenzte Zeit in einen einzelnen Pfad bindet.

Die Implementierung dieser Protokolle ist stark von OpenFlow und NOX abhängig. OpenFlow ermöglicht Netzwerk Forschern das Verhalten ihrer Netzwerkausrüstung durch Festlegung von Regeln in den fliesst Tabellen der Routern zu kontrollieren. NOX bietet eine einfaches Application Programming Interface (API), um fliesst Tabellen von Routern zu programmieren. Aufgrund dessen sind wir in der Lage neue Protokolle wie die, die wir in dieser Arbeit presentieren werden zu definieren.

Wir demonstrieren in dieser Arbeit, dass die Gruppierung von Packeten eine signifikante Netzwerkleistungssteigerung liefert, auch wenn diese nicht optimal ist. Genauer gesagt zeigen wir auf wie unsere Protokolle in einigen Fällen bis zu N mal den Durchlauf des Shortest Path erreichen, wobei N die Zahl der verschiedenen Pfade des identischen Durchlaufes von Quelle zum Ziel sind. Desweiteren zeigen wir, dass unsere Protokolle mehr vorhersehbahren Durchlauf als einfache hash-basierte Leitweg Algorythmen liefern.

Heutige Netzwerke liefern immer mehr Verbindungen zwischen sämtlichen Quell-Ziel Paaren. Die meisten dieser Verbindungen bleiben ungenutzt bis irgend ein Fehler stattfindet. Mit den Protokollen die in dieser Arbeit vorgeschlagen werden, könnte sich der zusätzlichen Bandbreite dieser derzeitig ungenutzten Verbindungen bedient werden. Folglich könnten wir die Gesamtleistung von bestehenden Netzwerken steigern, ohne die hardware zu ersetzen.

Contents

Acknowledgements i

Abstract iii

List of Figures xi

1 Introduction 1

1.1 Today's Internet . . . 2
1.2 The Benefits of Multipath . . . 3
1.3 The Applications of Multipath Routing . . . 5
1.3.1 Load Balancing . . . 5
1.3.2 Quality of Service . . . 5
1.4 Thesis Contributions . . . 5
1.5 Outline . . . 7

2 History of Routing 9

2.1 Networking Fundamentals . . . 9
2.1.1 Reference Models . . . 9
2.1.2 Foundations of Ethernet. . . . 13
2.1.3 Evolution of Ethernet - The fast and the faster. . . . 17
2.2 High Performance Networks - InfiniBand . . . 20
2.3 Routing - Where to next? . . . 21
2.3.1 Some Issues in Routing . . . 22
2.3.2 Properties of Routing Algorithms . . . 24
2.4 Classification and the Evolution of Routing Algorithms. . . . 25
2.4.1 Static Algorithms . . . 25

2.4.2 Adaptive Algorithms 26
2.4.3 Distance Vector Routing 26
2.4.4 Link State Routing 30
2.4.5 Optimal Routing 32
2.5 Summary 33

3 Multipath Routing **35**
3.1 Why Multiple Paths: Some justifications 35
3.2 Multipath Routing Schemes 37
3.2.1 Proposed Extensions to Single Path routing 37
3.2.2 Multipath Routing Algorithms 42
3.3 Multipath Routing in current IP networks 44
3.3.1 OSPF Extensions 44
3.3.2 Alternative Methods 46
3.4 Summary 48

4 Multipath Theory and Models **49**
4.1 Graph Algorithms 49
4.1.1 Dijkstra's Algorithm 49
4.1.2 Bellman-Ford Algorithm 50
4.2 Selfish Routing 54
4.3 Analytical Models 55
4.3.1 Queue Theory 56
4.4 Summary 61

5 The MultiRoute Family **63**
5.1 Path Discovery 63
5.1.1 MultiRoute Path Construction (MRPC) 64
5.1.2 Algorithm Sketch 66
5.2 MMP - MultiRoute Monitoring Protocol 67
5.2.1 MMP Packet Structure 68
5.2.2 MMP Header 69
5.2.3 Status Packet 70
5.2.4 Update Packet 70
5.3 The Protocols 72
5.3.1 The Classical - MultiRoute 72
5.3.2 The Organizer - StepRoute 81
5.3.3 The Know-it-all - PathRoute 85
5.3.4 Implementation Experience 89
5.4 Commodity Protocols 90
5.4.1 Shortest Path 90

5.4.2 Equal Cost MultiPath 90
5.5 Theoretical Delay Model for MultiRoute-based protocols 92
5.6 Summary 94

6 Materials & Methods **95**
6.1 OpenFlow & NOX 95
6.1.1 OpenFlow 95
6.1.2 NOX 98
6.2 TestBed Installation 99
6.2.1 Routing Hardware 99
6.2.2 Network Installation 101
6.2.3 Machine deployment 104
6.3 GETB Network Testers 104
6.4 Experimental Technique 105
6.4.1 Parasite Traffic 105
6.4.2 Fully Meshed Traffic 105
6.4.3 Latency 105
6.5 Summary 105

7 Results & Discussion **107**
7.1 Test Topologies & Associated Routing Tables 107
7.2 Parasite Traffic 109
7.2.1 Parasite Traffic on the Flower Topology 111
7.2.2 Parasite Traffic on the Pentagon 114
7.3 Fully-Meshed Traffic 118
7.3.1 Fully-Meshed Traffic on the Flower 119
7.3.2 Fully-Meshed Traffic on the Pentagon 120
7.4 Latency & Packet Loss Tests 121
7.4.1 Latency & Packet loss on the Flower 122
7.4.2 Latency & Packet loss on the Pentagon 124
7.5 Summary 124

8 Conclusions & Future Work **127**
8.1 Achievements 127
8.2 Summary of Results 128
8.3 Future Work 129

A Collection of derivations **131**
A.1 Disruptions in the Hash Threshold variant of ECMP 131
A.2 Backpressure messages given in AMP 132

A.3 Dependence of Message Length and Interarrival times 133
A.4 The Price of Anarchy in Multipath networks with linear cost functions 135

B Collection of Pseudocodes **139**
B.1 Dijkstra's Algorithm Pseudocode 139
B.2 Bellman-Ford's Algorithm Pseudocode 140
B.3 MultiRoute Path Construction - Pseudocode and Proof 141
B.3.1 Proof 141
B.4 MultiRoute Protocol Family 143
B.4.1 The Classical - MultiRoute 143
B.4.2 The Classifier - StepRoute 145
B.4.3 The Know-it-all - PathRoute 147

References **149**

List of Figures

1.1 Internet users per 100 inhabitants 1997-2007 1
1.2 Illustration of Path Quantity and Independence. 4

2.1 Inter-networking models 10
2.2 The TCP/IP Model 12
2.3 The concept of Ethernet, as drawn originally by Bob Metcalfe. 14
2.4 A simplified algorithm sketch for CSMA/CD 15
2.5 Structure of an 802.3 Ethernet Frame. 16
2.6 A bridged Ethernet network. 18
2.7 A switched Ethernet network. 19
2.8 Interaction between Transport and Network Layer 23
2.9 Conflict between fairness and optimality. 24
2.10 Evolution of Distance Vector Algorithms 27
2.11 Evolution of Implemented Routing Algorithms 30
2.12 Evolution of Optimal Routing Algorithms 33

3.1 A complex network 36
3.2 The Evolution of multipath algorithms 38
3.3 An example of MPA. 39
3.4 An example of SPF-EE 40
3.5 An example of AMP. 41
3.6 An example of Shortest Multipath constructed with DASM. . . 43
3.7 The flow classifier. 47

4.1 Running Example of Dijkstra's Algorithm 51
4.2 Running Example of Bellman-Ford's Algorithm 53
4.3 Pigou's Example 54

4.4 A standard queue. 56
4.5 Average delay as a function of ρ. 59
4.6 A tandem network. 60

5.1 Output of the MRPC algorithm 65
5.2 Step-by-Step of MRPC according to the algorithm. 67
5.3 A *Routing Mask* with its associated routing table 69
5.4 MMP header 70
5.5 MMP status packet 71
5.6 MMP update packet 71
5.7 The reference topology. 74
5.8 Comparing a *Routing Mask* with its an Update Vector 75
5.9 The transfer function used in Classical MultiRoute. 76
5.10 *Routing Mask* Length as a function of N and C. 77
5.11 Plot of the *Routing Mask* Limitation 79
5.12 The transfer function used in StepRoute. 82
5.13 Flowgraph of the distance diffusion computation. 87
5.14 Inheritance/Dependency graph for MultiRoute-based protocols 91
5.15 Flow delay as function of the load ($= \frac{\lambda}{m\mu}$) and number of servers. 93
5.16 Number of flows in each queue 94

6.1 The OpenFlow flow signature. 96
6.2 The OpenFlow switch. 97
6.3 An explanation of routing processes. 100
6.4 The testbed as it is implemented at CERN's computing center. 102
6.5 The connectivity map of the Testbed. 103

7.1 The Flower topology. 108
7.2 The Pentagon topology. 110
7.3 Shortest Path versus MultiRoute in the presence of Parasitic traffic. 111
7.4 Shortest Path versus StepRoute in the presence of Parasitic traffic. 112
7.5 Shortest Path versus PathRoute in the presence of Parasitic traffic. 113
7.6 A comparison of the predictability of MultiRoute versus Hash Threshold 115
7.7 Shortest Path versus MultiRoute in the presence of Parasitic Traffic. 116
7.8 Shortest Path versus StepRoute in the presence of Parasitic Traffic. 117
7.9 Shortest Path versus PathRoute in the presence of Parasitic Traffic. 118

7.10 Fully-Meshed Traffic running on the Flower. 119
7.11 Successive Fully-Meshed Traffic running on the Flower. 120
7.12 Fully-Meshed Traffic running on the Pentagon. 121
7.13 Latency plots for protocols running on the Flower. 122
7.14 Packet Loss for protocols running on the Flower. 123
7.15 Latency plots for protocols running on the Pentagon. 123
7.16 Packet Loss for protocols running on the Pentagon. 124

A.1 Before and After the deletion of the third next hop possibility. 131

Chapter 1

Introduction

Over the past two decades the Internet has grown at an unprecedented rate. Estimates show that one in five inhabitants of the planet are now connected to the Internet. That number is expected to grow rapidly with the emergence of mobile devices and increasing penetration in developing countries as depicted in Figure 1.1. This has, and will, lead to the deployment of more networking infrastructure and therefore there will be an increase in connectivity worldwide.

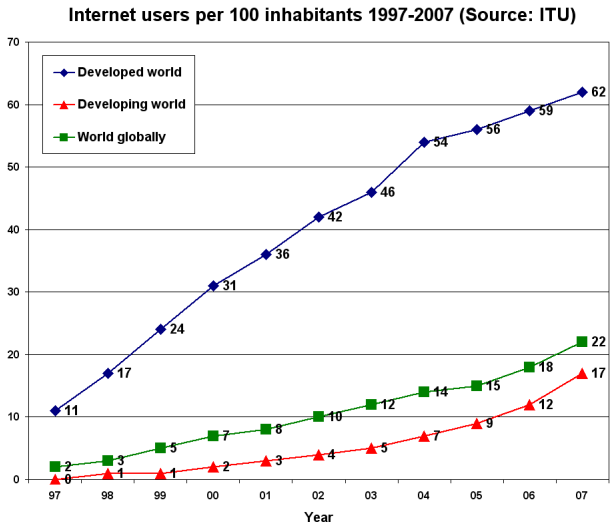


Figure 1.1: *Internet users per 100 inhabitants 1997-2007*

This increase comes with extra challenges for ISPs to provide a continued good level of service to customers, achieve their operational objectives and differentiate their service offerings. The increased infrastructure and connectivity brings new problems in traffic engineering, namely in terms of congestion and network performance. One of the major challenges to Internet traffic engineering is constructing the ability for automated control that adapts rapidly to a changing network state as mentioned in [ACE+02]. This challenge is precisely the topic of discussion in this thesis.

Besides these challenges, the increase in connectivity and infrastructure brings a new opportunity to the Internet by providing many redundant connections between networks. Each of these redundant connections can be seen as an alternative path for networks, which are currently unused by routing protocols. Moreover, current technologies do not use congestion information as a criteria for their routing decision process. For these reasons, research in multipath routing protocols has become very active. By embracing the Internet's growth and using appropriate technologies to measure and distribute congestion indications, it is our belief that a viable multipath routing protocol will emerge.

1.1 Today's Internet

Packet switched networks rely quasi-entirely on efficient routing algorithms. Current protocols have been designed to mainly deliver network robustness and resiliency in the event of a failure. While this has been a noble and just cause, the research community and interests are shifting towards traffic engineering, ie. the optimization of networks in terms of performance. Current technologies have not been designed for traffic engineering as they do not provide a simple solution to allocate network resources with respect to changing traffic demands.

Nowadays, the Internet is built mainly on shortest path first protocols such as OSPF in [Moy98], RIP in [Hed88] , etc. Such protocols only consider the shortest path between two networks and either discard alternative paths or they will only consider alternatives which are of equal cost to the shortest path. This, of course, has the disadvantage of not only limiting the maximum throughput between any two networks; but also, and much worse, increasing the probability of congestion.

Current operational networks suffer significantly from issues related primarily to congestion. A network device is said to be congested if it experiences a sustained overload over an interval of time. Congestion almost always causes a degradation of the throughput offered by network connections.

In an effort to solve these problems, much research has been done employing multipath solutions. One interesting single path approach by [FT00] was to use the current technologies, mainly OSPF in multipath scenarios, and modify

the connections weights based on the projected network resources demands by users. Unfortunately the optimization problem was shown to be computationally intractable (NP-hard), and moreover it was very difficult to obtain timely and accurate traffic values from a live network. Since such solutions will always require the use of heuristics, a solution involving a multipath protocol may be far simpler.

1.2 The Benefits of Multipath

Multipath technology attempts to exploit the underlying physical network resources more efficiently by providing multiple paths between source and destination pairs. By providing multiple paths, multipath technologies have the ability to aggregate the bandwidth offered by the various paths and thereby enable the network to support higher data rates than that supported by a single path. Moreover, since multipath technologies function over several paths simultaneously, we can expect increased resiliency to failure due to the fact that as one path fails others are still in operation. Furthermore, if there is a network failure re-computation can be done off-line while the multipath protocol is still transporting traffic, thereby rendering failures virtually transparent to users.

A multipath protocol needs to guarantee two essential aspects. The first being the computation of multiple loop-free paths and the second is the ability to split traffic amongst these paths efficiently. When considering a multipath scheme, the device implementing such a scheme must first calculate the set of paths available between the source and destination. There are two concepts which characterize a path set:

- *Path Quantity* defines the number of available paths between the nodes. A higher number indicates that there is a better chance at load distribution.
- *Path Independence* describes the freedom of each path set, ie. does this path set contain paths which share one or more links with other path sets? Evidently, independent path sets are ideal but complete independence can be very difficult to achieve.

Figure 1.2 illustrates these two concepts. Consider the following paths set for node A sending data to node F, $\wp_1 = \{(A-B-E-F), (A-C-E-F)\}$ and $\wp_2 = \{(A-B-D-F), (A-C-E-F)\}$. Both path sets contain the same number of paths, but for path set $\wp_1$, we can easily see that the paths are not independent due to the fact that they share link E-F. Path set $\wp_2$ is independent and therefore would lead to better usage of the network resources and would be less likely to get congested. Multipath protocols which

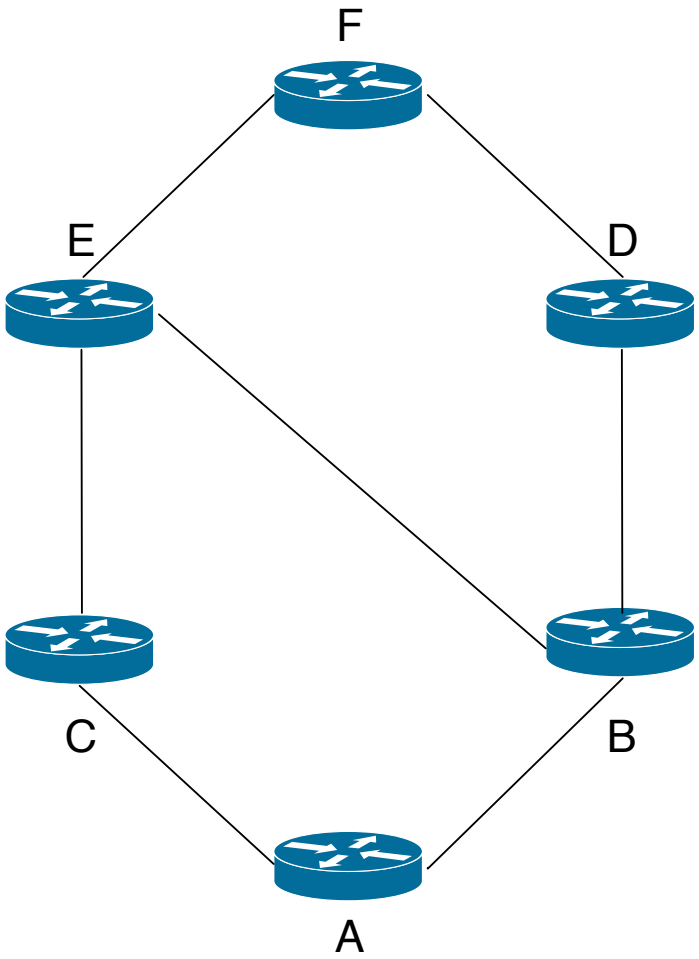


Figure 1.2: *Illustration of Path Quantity and Independence.*

try to optimize these concepts will therefore deliver higher performance. While these concepts are essential, we also believe it is vital to inform the multipath protocol about the network status and also it needs to be aware of the network topology, these concepts will be introduced in Section 1.4.

1.3 The Applications of Multipath Routing

1.3.1 Load Balancing

Load balancing, in the context of Multipath routing, is geared towards minimizing the risk of congestion by making more use of the available network resources. By minimizing congestion, the idea is to reduce packet loss to a minimum but, on the other hand, if alternative paths are inaccurately chosen then we would obtain additional propagation delay. We therefore have a trade-off situation, namely throughput versus delay, as some network applications are sensitive to packet loss whilst others are affected by delay. As we will see later, it is very important to be able to control this trade-off.

The ability to control the traffic flow is essential when deploying a load balancing solution. In traditional systems, the network administrator had to control the link metrics taking care not to disrupt the overall traffic flow. In a multipath system, the routing algorithm is responsible for monitoring the metrics and taking action based on them thereby resulting in rapid adaptation to changing network conditions.

1.3.2 Quality of Service

Quality of Service has long been a feature which has been difficult to implement. The IETF's (Internet Engineering Task Force) Integrated Services has shown scalability problems when faced with the large amount of memory required to store routing states and to maintain consistency. Another IETF effort, called Differentiated Services has also proved to be non feasible as it would impose a significant overhead to cope with link failures and maintain consistency [BMG00].

Multipath offers a scalable solution to Quality of Service, as various paths can be reserved for a specific type of service. Flows of similar types can be aggregated, and therefore the quality for an aggregated flow can be guaranteed which in turn provides guarantees to an individual flow.

1.4 Thesis Contributions

This thesis presents a new approach to multipath routing, called MultiRoute, which is media independent and this solution can co-exist with current routing

technology. This protocol breaks down the underlying global optimization problem of resource allocation into a set of local problems. The next-hop for each destination is computed off-line and stored in the routers forwarding table, enabling a rapid next-hop lookup. Traffic is then grouped into a flow identified by several parameters, and assigned to the port corresponding to its next-hop. A flows assignation is immutable for the duration of its lifetime, thereby avoiding path oscillations and thus providing a guarantee for the protocol's stability.

The first contribution is a method for computing alternative paths based on a modified Dijkstra algorithm. The multiple path discovery process relies on the existence of a shortest path between source and destination points. After establishing the shortest path cost (the reference cost), each alternate path is computed whose cost is within a reasonable delta of the reference cost. This ensures that the latency versus throughput trade-off is respected.

The second contribution is an In-Network monitoring protocol to distribute the statistical information, eg. counter values or congestion representations, to other routers. The statistics are polled locally by the router and sent to neighboring routers, this process is performed in-band and not by an external monitoring process. A one bit representation of congestion is used, thereby reducing to a minimum the size of the monitoring information and therefore its impact on the network load. Disseminating the congestion information is achieved by using an aggregation protocol, therefore allowing for real-time statistics to be distributed within the network efficiently.

A third contribution is a novel topology representation which allows routers to enhance their routing decision based on the destination of a flow. Each router maintains its own routing vector, consisting of congestion representation of its paths to different networks. Routing vectors are then exchanged with neighboring routers using the In-Network monitoring. The neighboring router is required to interpret this information according to its own forwarding table, therefore each router must be aware of the forwarding table of each of its neighboring routers. To achieve this, we introduce the concept of a *routing mask* which represents the structure of forwarding tables in multipath enabled routers.

The fourth and final contribution of this thesis is the deployment and installation of a real world testbed. This test bed was used to develop, test and perform experiments on our routing protocols. The testbed was implemented using NOX and OpenFlow technology using commodity ethernet hardware.

Experimental results will be presented to demonstrate both the correct functioning of the routing protocol and the improvements over existing technologies. Representative topologies presented, and in each case a multitude of traffic patterns, will be used to test each protocol. The routing protocol will be submitted to several scenarios in order to establish its effect on delay

and throughput. Results show that in most cases our protocol outperforms shortest path and other multipath technologies in terms of throughput and delay.

1.5 Outline

A brief and elementary introduction to networking will be presented in Chapter 2. Chapter 2 will also define routing and why it exists, followed by an overview of existing technologies and their classification.

The state-of-the-art is discussed in Chapter 3, where several other multipath protocols designed for different situations will be presented. Chapter 4 will introduce theoretical aspects of multipath routing and discuss the limitations of current models, and therefore demonstrate the mathematical complexity in building an accurate model. Next in Chapter 5, we discuss the important components that have to be taken into account when building a multipath system.

Chapter 6 will be describing the experimental setup and detail the implementation of this routing protocols and those used as comparison. Finally Chapter 7 will present the results of this protocol when compared with other multipath protocols under varying topologies, scenarios and metrics.

Chapter 2

History of Routing

"Not all those who wander are lost."
– J.R.R. Tolkien

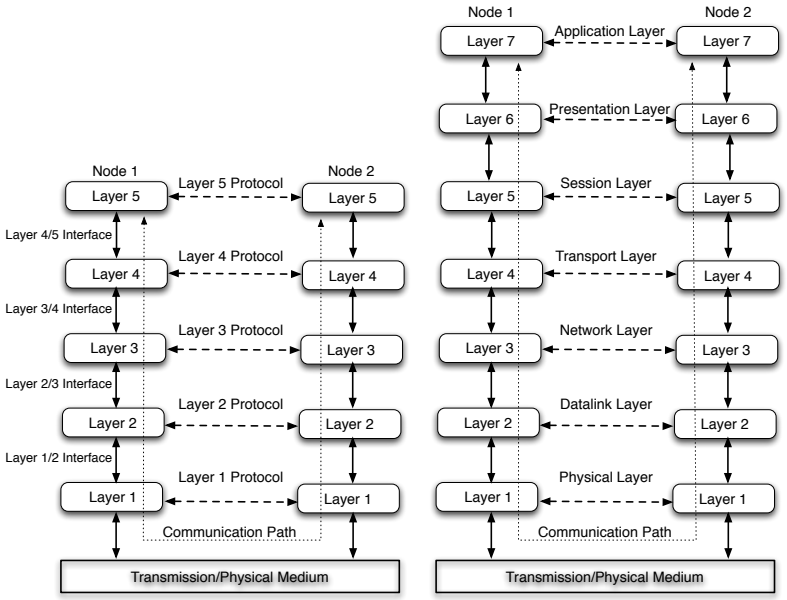
2.1 Networking Fundamentals

One of the most significant evolution of computer systems is the merging of computers with communication systems [Tan03]. The early model of a single mainframe computer serving just local users has been replaced with a globally distributed set of interconnected commodity machines. This new model is called a Computer Network. The interconnecting media, and the way they are exploited have also evolved.with time giving rise to countless network protocols.

Many communication solutions have emerged over the decades in response to different requirements and therefore there are innumerable network technologies used to support or create them. The main drawback of this situation is that there are also countless network protocols, and thereby it becomes virtually impossible to understand all of them.

2.1.1 Reference Models

Networking software has been designed as layers, as shown in Figure 2.1a, in an effort to reduce its complexity. The goal of each layer, in all networks, is to expose services to the higher layers via interfaces between each layer, thereby abstracting the implementation complexities. Such a design is extremely powerful, not only because each layer is free to define how it performs its objective



(a) *The Reference Model.* **(b)** *The OSI Model*

Figure 2.1: *Inter-networking models*

but also it can be optimized or completely replaced without affecting other layers. Each layer on a host uses a protocol to communicate with the same layer on another host. A protocol is defined as an agreement by both communicating parties on how the information exchange should proceed.

Figure 2.1 depicts the layers, protocols and interfaces used to communicate between two nodes. Horizontal lines depict logical communication path, as no actual physical communication occurs horizontally, but rather physical communication occurs as shown by the dotted line, ie. via all the layers.

In the next subsections, we will present two models; the OSI and the TCP/IP models. While the OSI model is general and the features addressed by each layer are still important, it is rarely used because of the bad timing of the standard ratification and its complexity. However, it is still the theoretical reference model for networks. On the other hand, the TCP/IP protocols are widely used, but the model is particular, and does not always provide a clear separation of interfaces and protocols.

The OSI Model

The Open Systems Interconnection (OSI) reference model was originally developed by the International Standards Organization in an effort to standardize the protocols used in the model. Figure 2.1b, shows the seven layers which make up the OSI stack.

Each layer in the OSI model performs a well defined function, and the boundaries are chosen to minimize the amount of information which has to be exchanged across the interfaces.

We will now present the function of each layer:

1. *The Application Layer* is the layer which is closest to the user. Essentially this means that both this layer and the user interact with the software application.

2. *The Presentation Layer*, contrarily to the other lower level layers which are designed to shift bits around reliably, attempts to add syntax and semantics to to the information. It does this by providing function to operations that are performed often.

3. *The Session Layer* enables users to establish sessions between themselves. For example, it would be used to login to a remote system or transfer a large document.

4. *The Transport Layer* has the basic role of obtaining data from the above session layer and splitting it up into smaller data units. These data units are then passed to the underlying network layer and the transport layer then makes sure the data arrives correctly at the other end.

5. *The Network Layer* handles and controls all the operations concerning a subnet. A particular issue is how a packet is routed from its source to its destination. These routes can either be determined statically or allocated dynamically. In this thesis we discuss different algorithms which are implemented in this layer.

6. *The Data Link Layer* has the task of obtaining the raw data from the physical layer and turning it into an input stream that seems error-free. This is accomplished by imposing that the sender split up the data into frames and transmit them sequentially.

7. *The Physical Layer* is responsible for transmitting the raw bits over the physical medium. It is responsible for the actual translation of bits into electrical voltages or optical levels and vice versa.

The TCP/IP Model

TCP/IP was designed for the ARPANET which was a project sponsored by the US Department of Defense (DoD). TCP/IP was built with a high degree of resiliency due to the fact that the DoD could have its installation attacked at any moment, and therefore wanted to keep communications alive as long as the end-hosts were still alive. The TCP/IP reference model, shown in Figure 2.2, was first defined by Cerf et al. [CI05], and consists of four layers which we are going to detail below:

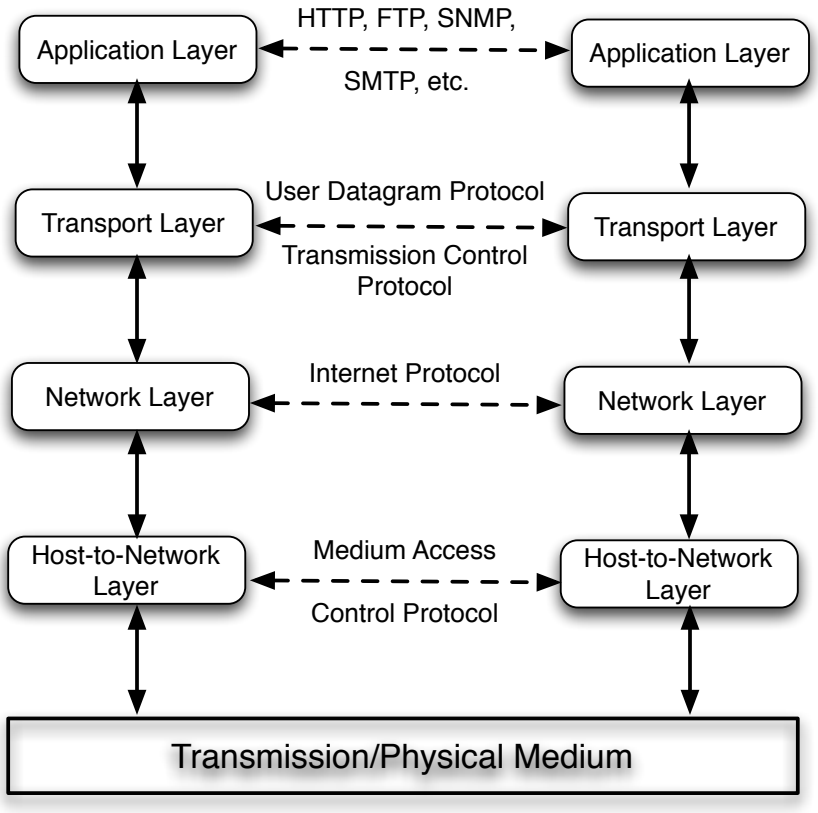


Figure 2.2: *The TCP/IP Model*

1. *The Application Layer* contains the same high level protocols described for the OSI application layer. The protocols used at this layer are denoted

as Layer 7 protocols (analogy with the OSI layers), even if the TCP/IP stack only has four layers (five, depending how you consider the Host-to-Network Layer).

2. *The Transport Layer* provides the same functionality described by the OSI Transport layer, and its protocols are frequently denoted as Layer 4 protocols. Two end-to-end protocols are defined by the model: TCP (Transport Control Protocol) and UDP (User Datagram Protocol). While TCP is connection-oriented and provides a reliable (error free) end-to-end communication, UDP is a connectionless protocol which does not provide any guarantee on the integrity of the transmitted data. However, UDP is lightweight compared with TCP and also provides multicast capabilities (i.e. one-to-many transmission).

3. *The Internet Layer* defines the IP (Internet Protocol) protocol and packet format. This protocol is designed to deliver the IP packets over a network with multiple paths. As packets from the same conversation may follow different routes, experiencing different delays, there is no guarantee for preserving the order of IP packets. As the IP protocol provides functionality similar with that of the OSI network layer, it is often denoted as a Layer 3 protocol.

4. *The Host-to-Network Layer* is not precisely defined in the TCP/IP model except to state that the host must connect to the physical medium in order to send IP packets. The protocol, commonly, found here is the Medium Access Control Protocol, which provides addressing and medium access control mechanisms that make it possible for several hosts to communicate within a network.

2.1.2 Foundations of Ethernet.

Ethernet is a technology which spans layers one and two of the OSI model, namely the physical layer and the datalink layer. Actually, the datalink layer has never changed since the introduction of Ethernet. The real beginning to Ethernet took place on the island of Hawaii in the early 1970s with a system named the ALOHA developed by the University of Hawaii [Abr70]. This system was constructed to allow radio communications between distant machines scattered over the island and a central IBM mainframe.

The ALOHA system allowed for a data rate of 9600 bits per second with fixed size frames transmitted sequentially. The mainframe would use a separate channel to convey acknowledgements of a successful transmission to sending hosts. If a terminal did not receive an acknowledgement after a fixed amount of time, it would timeout. Upon a timeout, the sending station would wait a random amount of time and attempt to retransmit. If two stations attempted

to transmit simultaneously, the data arriving at the mainframe would appear corrupted due to the overlapping communications from the source stations. Such an event was referred to as a *collision* causing both data packets to be lost and therefore neither station would receive an acknowledgement from the mainframe. The *collision* and timeout mechanisms resulted in poor performance by the ALOHA protocol, only about 18% of the bandwidth was ever usable [Tan03].

In 1973, a researcher named Robert Metcalfe in Xerox's research laboratory in Palo Alto developed an improved version of the ALOHA system. This system, named Ethernet (Figure 2.3), was designed to interconnect computers in the laboratory. Ethernet was named after *luminiferous ether*, which was thought to be the universal transmission medium for light at the end of the 19th century.

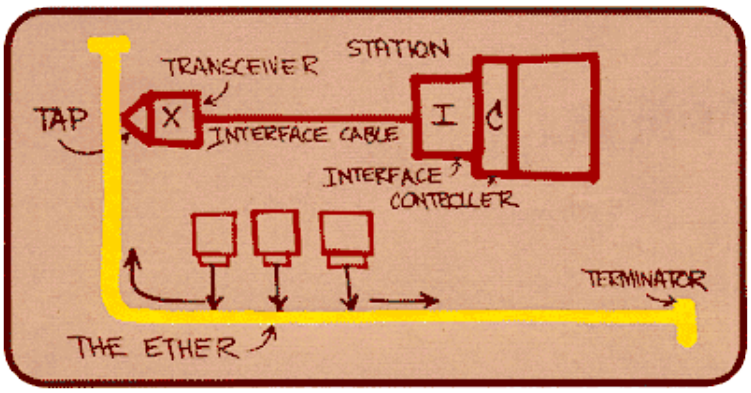


Figure 2.3: *The concept of Ethernet, as drawn originally by Bob Metcalfe.*

With Ethernet came a new transmission medium, a thick yellow coaxial cable. Still the idea was the same as ALOHA, as the coaxial cable was the shared medium. Ethernet did not have a control channel as ALOHA did, so Metcalfe devised the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol [Tan03], in order to better make use of the available bandwidth offered by this cable.

Figure 2.4 shows a simplified flowchart of the CSMA/CD algorithm. Initially, a station must listen on the shared medium before taking a decision to transmit, as it can only transmit when the medium is free, otherwise it will cause a collision. Due to the propagation delay imposed by the shared medium, two stations may consider that the medium is free and start their transmission, thereby inadvertently causing a collision. The stations detect this collision and stop their transmissions immediately and wait a random amount of time before

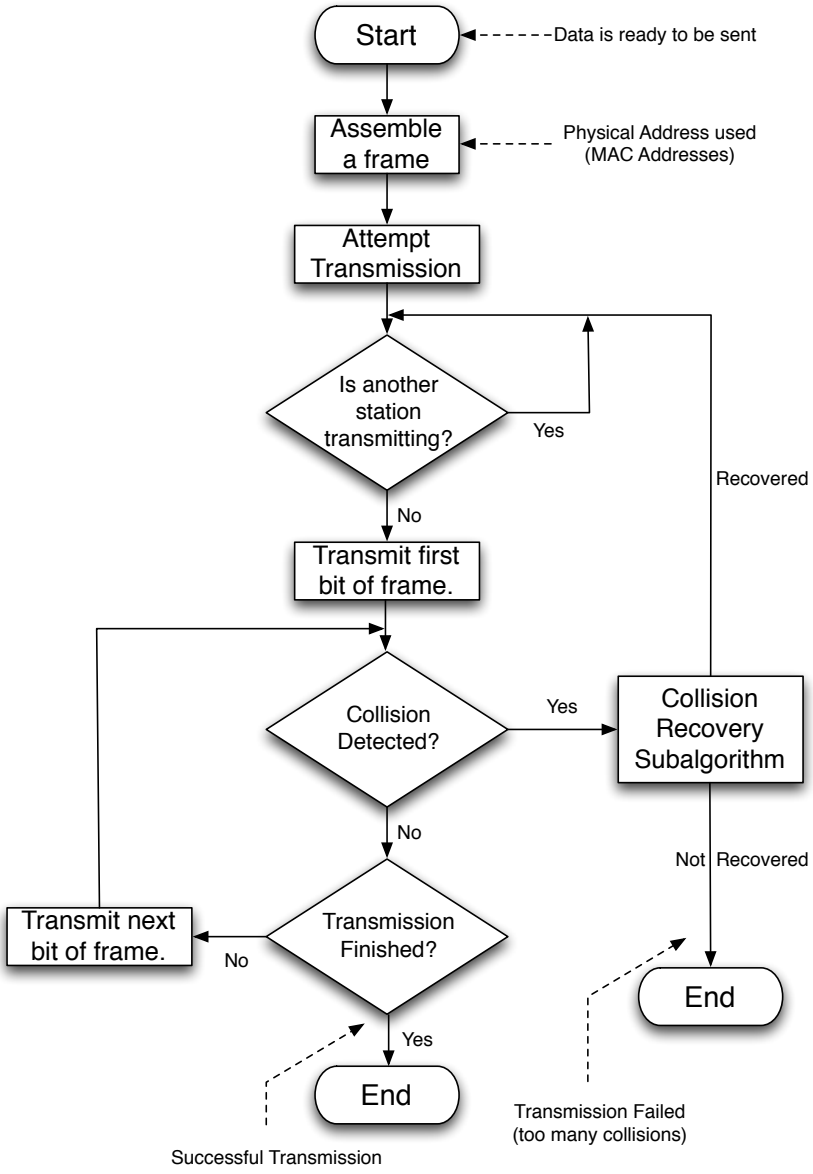


Figure 2.4: *A simplified algorithm sketch for CSMA/CD*

attempting a retransmission. The interval in which a collision may happen is referred to as the collision window.

The maximal collision window is double the propagation time between the two ends of the network. In order for the collision mechanism to work correctly, the transmission time for the smallest frame has to be greater or equal to the collision window, if it is less then we obtain what is known as a collision fragment. It is therefore clear that a small collision window associated with the transmission of large frames would yield the maximum efficiency of the transmission medium. The Ethernet standard, however, defines the smallest and largest frame sizes possible. This in turn determines the theoretical efficiency of the network transmission, while real networks have more overhead as shown in [BMK88].

The Ethernet developed by Metcalfe and Xerox was so successful that Xerox, DEC and Intel drew up a standard for it which later became the base for IEEE 802.3 [IEE85; Spu00]. The transmission rate was set to 10 Mbps, the maximum and minimum frame sizes where set to 1518 and 64 bytes respectively. The official denomination 10BASE5 was adopted (10 Mbps data rate, base band transmission with network segments of 500 meters maximum). The shared medium was used for both directions thereby creating a half duplex network. In order to increase the maximum size of the network, devices called repeaters were introduced, which would clean and amplify the signal.

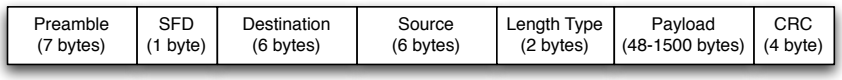


Figure 2.5: *Structure of an 802.3 Ethernet Frame.*

The Ethernet IEEE standard [IEE85] defined the structure of the frame as shown in Figure 2.5. The frame starts with a preamble, which is a special pattern allowing the receiver clock to synchronize with the incoming stream. The preamble is followed by the *Start of Frame Delimiter (SFD)* which indicates the immediate arrival of the frame. The source and destination fields are numbers which uniquely identify a device connected to the Ethernet network, although the destination field can contain special numbers that indicate that this communication is either multicast or broadcast. The length field is used to communicate the amount of data in the payload if it is less than 1500 otherwise this field is used to give information to higher layer on how to interpret the payload. If the payload carried is less than 46 bytes then it is padded with zeros in order to reach the minimum length of an Ethernet frame of 64

bytes. The *Cyclic Redundancy Check (CRC)* enables the receiver of the frame to determine if the frame was corrupted during transmission. Every frame is followed by an idle period of 96 bit times, which indicates the *End of Frame Delimiter (EFD)*.

Essentially, Ethernet was a broadcast communication protocol as all network members are listening on the cable at the same time. To this end, the IEEE standard introduced a special broadcast address (0xFFFFFFFFFFFF) to take advantage of this natural feature of Ethernet. A frame sent out with this destination address would be processed by all stations on the network. Another special group of addresses, which all have their first bit equal to 1, were used for multicast. Multicast frames where only processed by a subset of stations on the network.

As the popularity of Ethernet networks grew, and the number of stations exchanging data grew as well, it became apparent that CSMA/CD's limitations on the size of the network were a constraint. As traffic on the network increased, a phenomenon called *network congestion* appeared which caused the useful throughput to drop significantly below what was theoretically expected. Protocols of higher levels which included their own congestion control mechanism, for example TCP [Pos81], caused further degradation of transfer rate as the number of stations increased.

The Ethernet bridge was introduced by IEEE in 1990 with the IEEE 802.1D standard [IEE91]. Bridges have the feature that traffic local to a collision domain are not passed over it, shown in Figure 2.6. If the network was well designed most of the traffic remained local to a collision domain, thereby increasing the useful bandwidth available on the two networks as opposed to a network with a single large collision domain.

2.1.3 Evolution of Ethernet - The fast and the faster.

In the same year as the introduction of the network bridge came IEEE 802.3i [IEE90] which presented a new type of media for Ethernet Transmission: the Unshielded Twisted Pair cable (UTP). The cable enabled full duplex transmission and changed the topology of networks from a shared bus to a star. Devices were placed around a hub or a switch which was at the center of the network.

A hub is a special type of Ethernet repeater, it is in fact just a collapsed segment. While its earliest version consisted of only two ports that linked two Ethernet segments together, it quickly became possible to obtain hubs with many ports. It provided no extra functionality over the shared coaxial cable and therefore the network was still viewed as a single collision domain. Moreover, transmissions were still limited to half duplex and limited to 10 Mbps.

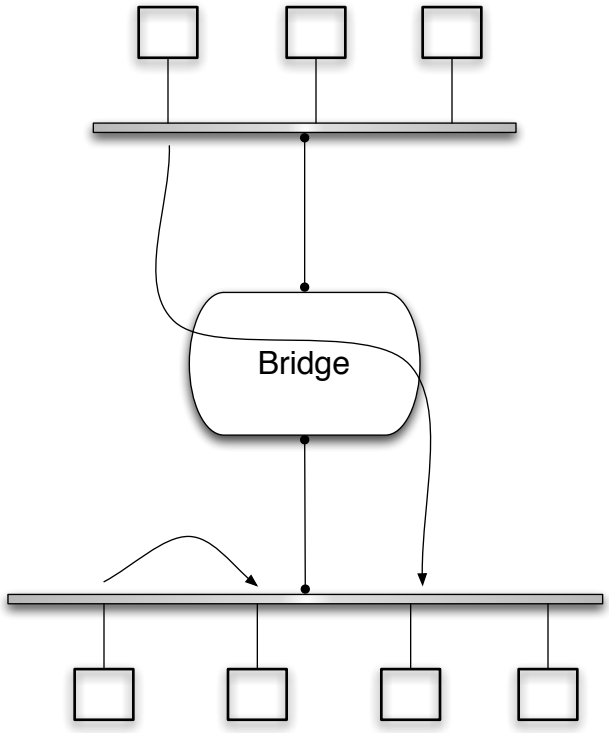


Figure 2.6: *A bridged Ethernet network.*

A major development in Ethernet Networks was the introduction of the switch. A switch is capable of connecting any of its ports to any other one for a brief instant via rapid configuration changes as shown in Figure 2.7. This enabled full duplex communications and therefore practically doubled the bandwidth available on the network. For a network switch of N ports the number of concurrent conversations is $\frac{N}{2}$. If the internal speed of the switch fabric is sufficient to service all ports concurrently it is said to be non-blocking. A blocking switch can still handle conversations on all ports provided that the total bandwidth does not exceed its internal capacity. At that point it is said to be oversubscribed.

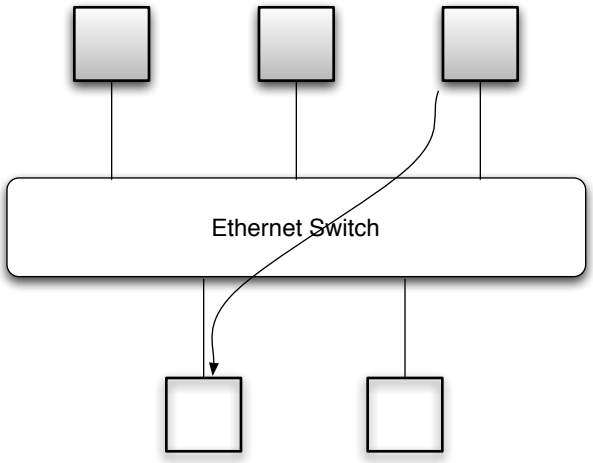


Figure 2.7: *A switched Ethernet network.*

Nowadays, Ethernet networks are built using switches. A switches internal fabric does not necessarily provide enough bandwidth to satisfy any configuration. In the situation where the incoming traffic surpasses the available internal bandwidth the switch is said to be oversubscribed. This is common practice amongst network manufacturers in order to provide relatively inexpensive solutions to customers who do not need the full bandwidth.

Fiber optics were standardized in IEEE 802.3j [IEE93] in 1993. Later in 1995, transmission speed was increased to 100 Mbps in IEEE 802.3u [IEE95]. Finally, in 1998, Gigabit Ethernet was introduced in IEEE 802.3z [Fra98]. More recently, in 2002, 10 Gbps was introduced.

Arguably, during the first decade of Ethernet there were other technologies that provided much better quality services. For example, Token Ring [IEE89] which provided a collision-free protocol and more bandwidth than the origi-

nal Ethernet. The Asynchronous Transmit Mode (ATM) was able to provide guarantees on the bandwidth that could be shared between different types of traffic, whereas Ethernet has no such guarantee and can only be considered as 'best effort'. These technologies proved themselves to be too expensive and much more complicated than Ethernet.

As stated earlier, Ethernet is mainly a technology located at layer one and two of the OSI model, which has the modest goal of, essentially, moving frames from one wire to another. As Ethernet networks grow, it becomes more complicated to transport packets from their source to destination as there may be intermediary devices and several possible routes between them. This is what the Network Layer (Layer 3) and more precisely network routing is responsible for.

2.2 High Performance Networks - InfiniBand

Before moving to routing techniques, we are going to detail an alternative communication technology, named InfiniBand, as its structure resembles modern Ethernet and its control structure is similar to OpenFlow (see 6.1.1). Therefore, we believe that the novel congestion protocols could be adapted to function in an InfiniBand network.

Infiniband has established itself as the industry standard for inter-server I/O. It was established by a consortium of about 180 companies under the InfiniBand Trade Association (IBTA). Membership to the IBTA is open to universities and research labs. Infiniband was designed to provide high levels of scalability, availability, performance and reliability. Achieving this goal does not come easy, the InfiniBand Architecture (IBA) has a 1500 page long specification [Ass].

The length of the specification allows the IBA to scale up and down depending on the required solution for the problem. On the other hand, this has lead to the development of many options, such as multiple link widths, multiple MTU, optional major features, and many more. This has become confusing and therefore IBA supports profiles which contain predefined options in order to simplify the deployment of IBA.

The IBA was developed due to the fact that processing power was starved by slow I/O, mainly because modern computer still rely on buses to transport data (at least until the adoption of PCI-Express). Buses are inherently shared mediums and therefore require an arbitration algorithm to mediate access to the said bus but this comes with a significant overhead. On top of this, buses are memory-mapped and therefore are accessed via the CPU's store and load operations. A store operation does not pose much of a problem as it can overlap with computation, but load operations are costly as the CPU will probably need the result of the operation rather sooner than later and therefore the

CPU will have to wait until this information becomes available. Finally, a third reason is that busses do not deliver the level of availability or reliability required by high performance systems. Indeed, a single failed device on a bus may disrupt some or all other devices attached to the bus.

As a solution to the problems encountered by buses, IBA defines point-to-point connections, and therefore data transfer is not bussed. Moreover it defines message semantics and therefore command and data exchanges as achieved via messages and not as memory operations. As many modern communication systems, the IBA is defined as a stack containing a physical, link, network, and transport layer.

The smallest functional IBA system is a subnet which can be joined together by router to form larger systems in a similar manner to modern IP networks. Switches route data from a source to a destination based on the routing tables computed either at the network's initialization and/or network modification. The exact structure and format of the routing tables depends entirely on the subnet manager as it is the entity responsible for maintaining the tables. Subnet managers send messages to the devices they control (ie. the agents). Each IBA system must contain at least one subnet manager that can either reside on an endnode or a switch. The subnet manager discovers all the devices on the network and assigns them local IDs, and computes the routing tables to be loaded a each device. There has been some study of routing techniques used in IBA systems [GFR+02] as well as some extension proposals [FLS+06].

The architecture of an IBA is similar to the solutions developed in this document to construct network protocols. Therefore, it is the author's belief that even though the implementation presented in this thesis employed Ethernet, that the same approach could be used in IBA systems. Indeed, a central aspect of an IBA system is the subnet controller which is quite similar to an OpenFlow controller (see Section 6.1.1), which is used to control Ethernet device behavior.

2.3 Routing - Where to next?

At the end of the previous section we discussed primarily Layer 2 networks, which only provide means for transport frames from wire to wire over an Ethernet hub or switch. Layer 3, the network layer, is concerned with transporting packets from the source to their final destination [BG92]. This process, called *routing*, achieves this goal by using devices called routers. This goal may seem simple, but the algorithms they employ are complex. Moreover, there is a variety of routing protocols which define how routers can communicate to determine the best routes to all destinations.

The origins of routing can be traced back to the late 1950s, where various

shortest path algorithms where proposed by Ford [Jr.56], Bellman [Bel58], Floyd-Warshall [Flo62] and Dijkstra [Dij59]. These algorithms compute the shortest path between a source and destination pair in a graph, and they make up the core of current routing protocols implemented nowadays. Each link in a network is given a cost, and therefore the cost of a path is the sum of the constituent link costs, and trivially the shortest path is the path with the lowest cost. Since these algorithms provide remarkable simplicity and generality, they have been in use since the early days of the ARPANET [KZ89] to networks nowadays.

Routing is a complex process, which involves several independent algorithms exchanging information between themselves. There are a several reasons for this complexity:

- Routing requires all the network nodes to coordinate to offer a coherent routing strategy.
- Routing must be capable of surviving link and node failures, this requires the algorithms to provide alternative paths and therefore to keep their routing databases up-to-date.
- In order to achieve high performance, a routing strategy may be required to modify its routes as areas of the network become congested.

2.3.1 Some Issues in Routing

Routing affects mainly two performance metrics, the first is *Throughput* (Quantity of Service) and the second is *average packet delay* (Quality of Service) [BG92]. These metrics are determined by the flow control mechanism provided by the Transport Layer as shown by Figure 2.8.

When the network is lightly loaded we can easily say that,

$$Throughput = OfferedLoad \tag{2.1}$$

As network load increases the flow control scheme will reject some traffic, due to congestion. We therefore have Equation 2.2,

$$Throughput = OfferedLoad - RejectedLoad \tag{2.2}$$

Traffic within the network will also receive an extra delay due to time required for the routing scheme to select the appropriate path. This (and other indirect factors resulting from the rejected load) will significantly affect the throughput as flow control schemes attempt to strike a balance between throughput and delay. Therefore, we can say that the more the routing scheme keeps delays low, the more the flow control scheme will allow traffic on the network.

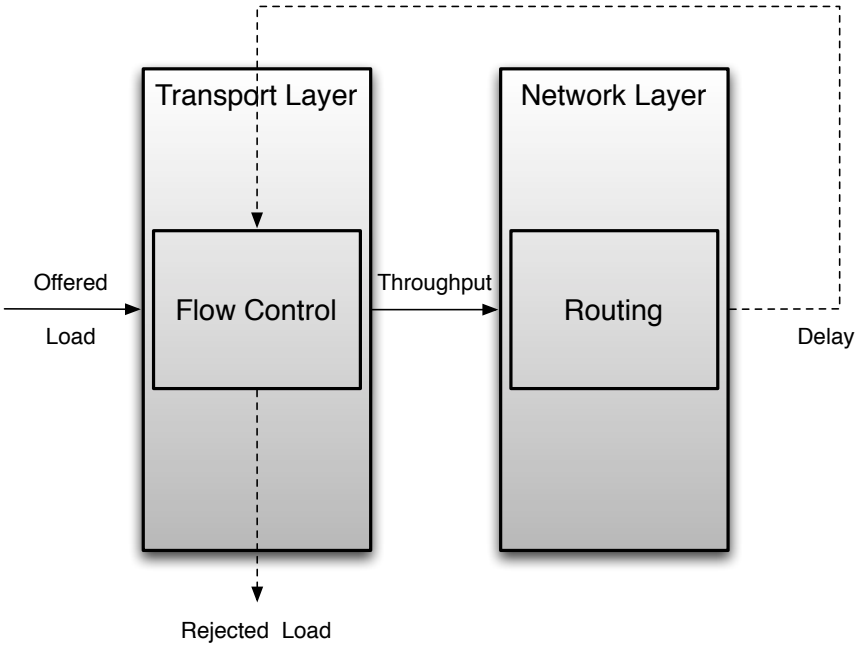


Figure 2.8: *Interaction between Transport and Network Layer*

2.3.2 Properties of Routing Algorithms

Before entering into the classification of routing algorithms let us first describe some desirable properties for routing algorithms: correctness, simplicity, robustness, stability, fairness, and optimality [Tan03].

- *Simplicity* and *Correctness* are trivial. All routing protocols should provide a simple solution to the routing problem, and maybe more importantly they should be correct.
- *Robustness* - When a routing algorithm is brought online, it may be expected to run for many years. Therefore, it must be able to withstand failures, changes in topology and traffic patterns.
- *Stability* is one of the most fundamental aspect of a routing algorithm simply because an algorithm which does not converge to some equilibrium could cause unpredictable network results.
- *Fairness* and *Optimality* - While they are enormously desirable properties, they are often opposed. As shown in Figure 2.9, suppose the horizontal links between A and A', B and B', and C and C' are saturated. In order to maximize the total flow, traffic between X and X' should be virtually non existent, but from X and X' point of view this is unfair. Therefore there is a tradeoff between fairness and optimality.

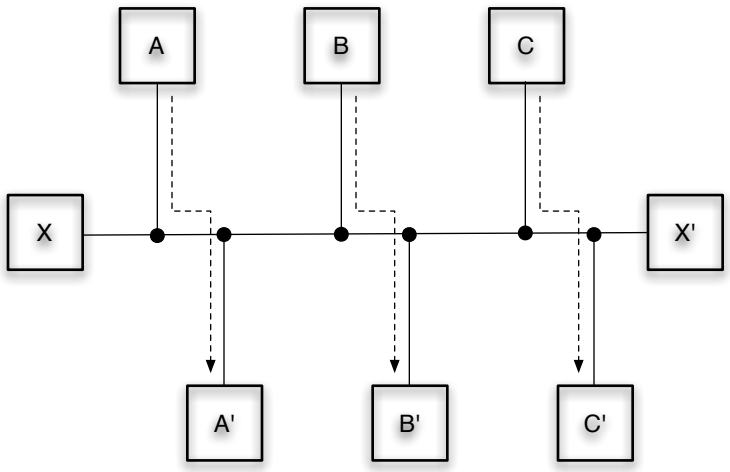


Figure 2.9: *Conflict between fairness and optimality.*

One last important concept before we move onto the classification of routing algorithms is a statement about optimal routes regardless of topology and traffic. This is known as the *Optimality Principle*. In its simplest form it states that if a router B lies on the optimal path between routers A and C, then the optimal path between B and C falls on the same route. The major consequence of this principle is that all the routes from any source to a given destination comprise a tree rooted at the destination. This tree is called a *Sink Tree*.

2.4 Classification and the Evolution of Routing Algorithms.

In this section we will attempt to classify routing algorithms while presenting their evolution from the original graph algorithms. We will not enter into the details of the graph algorithms, for more information see Section 4.1. Two classes are commonly used to group routing algorithms: *Static* and *Adaptive*. In static algorithms, all routes are computed off-line and this configuration is then downloaded to the routers. This is sometimes referred to as *Source Routing*. Adaptive algorithms react to changes in topology and network state (such as congestion, delay, or other metrics), these protocols are the most common in packet switched networks [BG92]. Another class of algorithms we will discuss are called *Optimal Routing* Algorithms. Although they have never had a real world implementation, they have an influence on multipath routing algorithms which will be discussed in Chapter 3.

2.4.1 Static Algorithms

The most basic static routing algorithm is to compute all the possible source-destination shortest paths off-line using a Shortest path algorithm [Dij59; Flo62; Bel58]. Then, use the output of this computation and install static routes at each router in the network. The details of these algorithms will be given in Section 4.1.

Another static algorithm is *Flooding*, where each incoming packet is sent on every outgoing line except the one it arrived on. Clearly, this algorithm generates many duplicate packets, and it can be shown that an infinite number of duplicates will persist in the network. To avoid these duplicates, each packet contains a hop count field which is incremented at each router. When the counter reaches some predefined limit the packet is dropped. Flooding is actually used in current protocols, such as OSPF [Moy98], in order to advertise a routers connections.

The two previous algorithms only consider topology. *Flow-based routing* considers both topology and network load. The assumption here is that the capacities and the average load for each link in the network is known. If these

values are known and so is the topology, it is possible to compute the mean delay from queuing theory. Finally, from the mean delay for each link it is then simple to calculate the mean packet delay for the entire subnet.

2.4.2 Adaptive Algorithms

Nowadays networks mainly use dynamic routing algorithms rather than static ones. The two main classes of algorithms are presented here: *Distance Vector* and *Link State* routing.

2.4.3 Distance Vector Routing

Distance Vector Routing algorithms function by having each router maintain its best known distance to each destination and which outgoing link to use in a routing table. Each entry is composed of the outgoing line and the metric estimate for each destination. These tables are then exchanged with each of the neighboring routers. The metric used to compute the distance may either be the number of hops, delay, or the number of packets queued along the path (queue depth). The local router knows its distance to all its neighbors, if the metric is hops then its distance is one hop. If the metric is queue depth, then the router simply measures it. Otherwise if the metric is delay, the router can send a special echo packet in order to measure the delay.

As an illustration of Distance Vector algorithms, assume that the metric is hop count and therefore the router knows that all its direct neighbors are within one hop. Periodically, every router sends its hop counts (exchange vector) for each destination to each of its neighbors, while also receiving one from every neighbor. Consider that router Y receives an exchange vector from router A containing the hop count A_B, which is the hop count from router A to B. If Y knows its hop count to A, denoted Y_A, then it can conclude that B is $Y_A + A_B$ hops away, and update its routing table accordingly. If this is done for all neighbors, a router will have an estimate of distances to each other router in the network.

Distance Vector Algorithms are sometimes called distributed Bellman-Ford routing algorithms or the Ford-Fulkerson algorithm, because of the researchers who developed it [Bel58; Jr.56]. Other implementations of distance vector algorithms include RIP [Hed88] and the original routing algorithm of the ARPANET [MW77] as shown in Figure 2.10. Distance Vector algorithms still exist in some current routing algorithms as shown in *EIGRP* [AGLAB94], as shown in Figure 2.11.

While Distance vector works very well in theory and reacts quasi perfectly when a newer and shorter route appears, there are situations where these algorithms can take a a very long or even infinite amount of time to converge. Consider the situation when a router X, only knows of a very long path to Z.

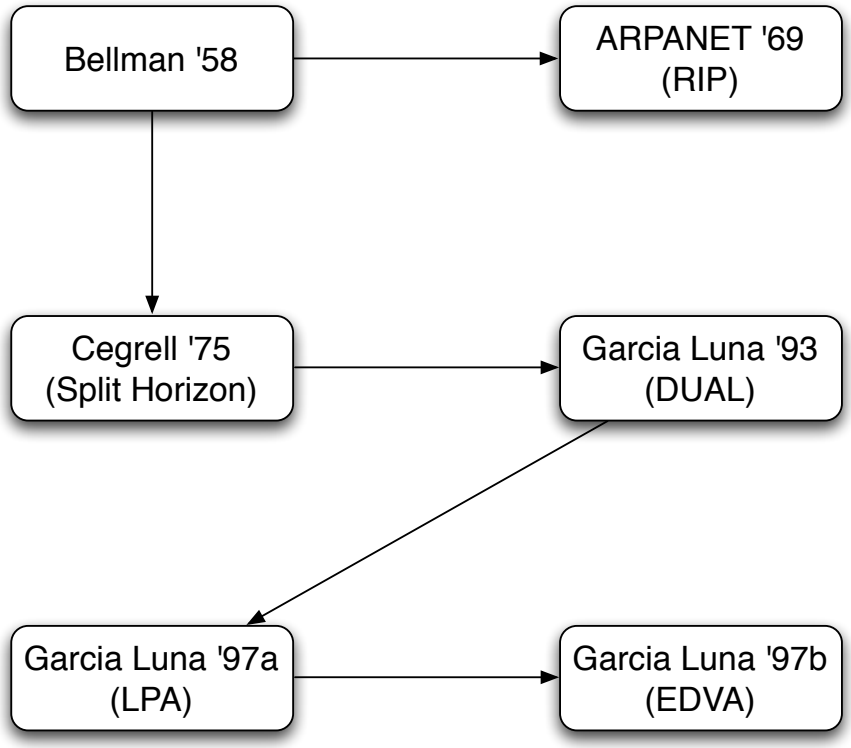


Figure 2.10: *Evolution of Distance Vector Algorithms*

If now a router Y tells X that it knows of a shorter route to Z, X will directly change its forwarding table to reflect this situation. This situation describes the normal and optimal operation of a Distance Vector algorithm.

A	B	C	D	E	
0	∞	∞	∞	∞	initial situation
0	1	∞	∞	∞	one exchange
0	1	2	∞	∞	two exchange
0	1	2	3	∞	three exchanges
0	1	2	3	4	four exchanges

(a) *A good news situation*

A	B	C	D	E	
	1	2	3	4	initial situation
	3	2	3	4	one exchange
	3	4	3	4	two exchange
	5	4	5	3	three exchanges
	5	6	6	6	four exchanges
				$\vdots$	
	∞	∞	∞	∞	

(b) *A bad news situation*

Table 2.1: *Count-to-infinity problem.*

Consider a topology where for the simplicity of argument, all the routers (A through E) are arranged in a line. Figure 2.1 illustrates the distance tables at each router to A. Figure 2.1a depicts the situation when all the routers are up and functioning correctly. Figure 2.1b shows the situation when A has suddenly gone down for some reason (link failure or broken router). At the first exchange, router B does not receive a message from A, but C tells B that it has a path to A of length 2. The problem here is that B does not know that the path advertised by C passes through itself. Therefore B thinks it can reach A via C with a length of 3. Next C notices that all the routers around have a path to A with length 3, and therefore it must update its table. By doing so it causes the same sequence of messages to occur again as shown in Figure 2.1b. Therefore all the routers never cease to update their tables, and will keep increasing their path length to A ad eternum. This issue is known as the *count-to-infinity* problem.

In 1975, Cegrell introduced the concept of *Split Horizon* [Ceg75] to counter the effect of the count-to-infinity problem. Split horizon operates on a very

simple premise, if a node A is routing traffic to a destination X via node B, it makes no sense for B to try to route to X via A. Therefore it is useless for A to announce to B that X is close to A. Split Horizon is elegant as it only involves a small change to the routing algorithm. Indeed, instead of a router always broadcasting all its routes on all its links, it simply modifies the exchange message on the the links that some destinations are actually routed through.

Unfortunately, split horizon does not solve the problem entirely. Consider two connected routers X and Y who both connect to T via Z. Suppose that initially both X and Y have a distance 2 to T, and Z has a distance 1. If the link between Z and T goes down for some reason, both routers X and Y report that they cannot get to T and Z reports that T is unreachable. Unfortunately in the meantime, X and Y both exchange their routing information where they have a route to T. Causing X and Y to think they can reach T via Y and X respectively, with length 3. After this, for every exchange they increment their length to T by one and therefore we are back to the count-to-infinity problem. This is known as a three-way loop.

Some important Algorithms

In this section we will discuss some algorithms, developed by Garcia-Luna-Aceves, which had an influence on some of the multipath algorithms discussed in Chapter 3.

In early 1993, Garcia-Luna-Aceves proposed a family of innovative routing algorithms, named *Diffusing Update Algorithms* (DUAL) [GLA93], which computed shortest paths using diffusing computations based on the idea presented in [DS80]. Diffusing computations is a computation that takes place over a set of networked machines. Dijkstra proposed method [DS80] for detecting the termination of the computation. DUAL provides loop-free operation at every instant throughout a route computation. This allows routers involved in a topology change to synchronize at the same time, while not involving routers that are unaffected by the change. DUAL is currently used by *EIGRP* [AGLAB94], which is a Cisco proprietary routing protocol.

A few years later in 1997, Garcia-Luna-Aceves presented an algorithm for *Loop-free Path-finding Routing* named LPA [GLAM97]. This algorithm eliminates the temporary formation of routing table loops during a topology change more efficiently than DUAL. A routing table loop is a path which is specified by the routers' routing tables at a specific time (usually during a topology change), which causes a packet to visit the same node more than once before reaching its destination.

Finally, later that same year, Garcia-Luna-Aceves presented a more *Efficient Distance Vector Algorithm* (EDVA) [XDla97]. EDVA uses the enhancements of the two previously described algorithms to reduce the convergence

time and the number of control packets needed by the routing algorithm. As we will see later, EDVA has had some influence on multipath routing algorithms.

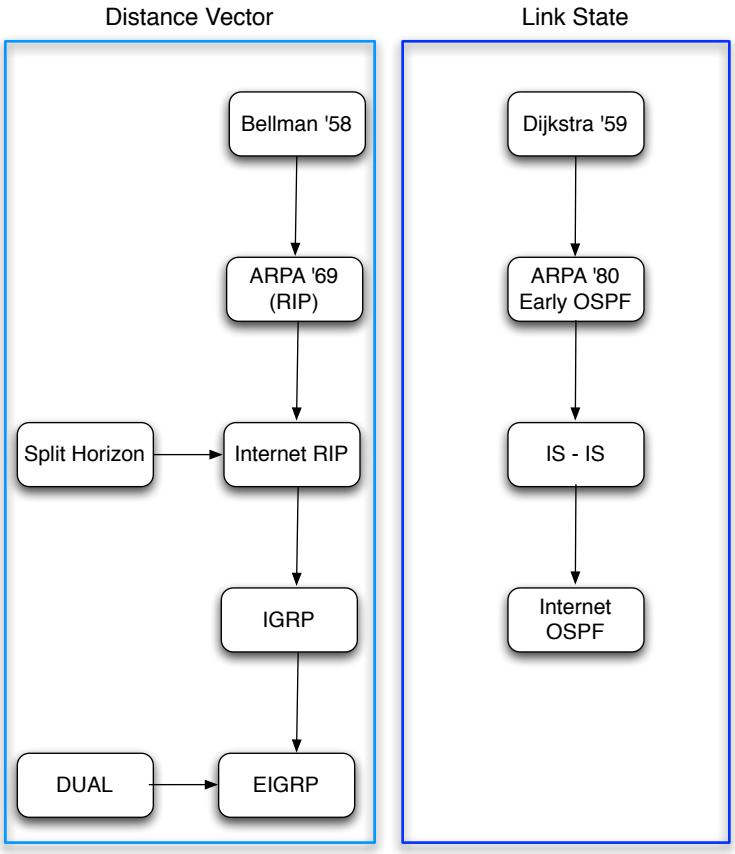


Figure 2.11: *Evolution of Implemented Routing Algorithms*

2.4.4 Link State Routing

Link State algorithms came about as a replacement to Distance Vector algorithms and provided the major improvement of being free of the count-to-infinity problem. While this improvement is highly desirable, it came at a price, Link State algorithms need to maintain a complete and up-to-date vision of the network topology at every participating node. In order to achieve this goal, Link State algorithms must initially discover all their neighbors. Then, they

must measure the cost (in terms of hops or delay) to each discovered neighbor. Next, this discovery must be advertised to all routers in the network, as we will see this is achieved via Link State packets. Only once all this information is gathered can the algorithm compute the shortest path to every other router using Dijkstra's Shortest paths algorithm [Dij59].

Link State algorithms provide several advantages over Distance Vector algorithms as is shown below:

- **Loopless and Fast Convergence** - As we have seen with Distance Vector schemes, the number of steps required to achieve convergence can be large and even sometimes infinite. Link State solves this by distributing information rapidly via a flooding protocol, and then performing a computation which is local to every router. Immediately after the computations all the routes are guaranteed to be valid and loopless.

- **Multiple Metrics** - Distance vector schemes do not cope well with multiple metrics. Indeed, if a single metric can have several orders of magnitude in variation, introducing others will probably cause the algorithm to converge very slowly as the solution space is expanded. Whereas Link State algorithms perform their computation with full topology knowledge, there can be as many metrics as desired.

- **Multiple Paths** - Nowadays, complex networks provide paths which are either identical or very similar to the shortest path. Distance Vector algorithms cannot support multiple paths in their current form because the routing table can only accommodate one entry per destination [Hui95]. In Link State, a simple modification to shortest path computation can be added and this will provide us with a list of candidate equal cost paths for a given destination. Thereby, this has the potential for increasing the overall performance of the network.

Neighbor discovery is achieved by sending specially crafted *HELLO* packets on each of the router's outgoing lines. Any router receiving such a packet must respond immediately indicating its identity. This introduces an, albeit minor, constraint to Link State protocols which is that router identities must be unique. A line's cost can easily be estimated by sending a *ECHO* packet. As with the *HELLO* packet, once a router receives a *ECHO* packet it sends it back immediately. This provides an estimate for the round trip time on a given line, which is then divided by two to obtain the line delay.

Once the information described above has been collected, the router builds the Link State packets. The distribution of the link state packets must be achieved reliably in order to guarantee that each router has the same view of the network. This distribution is achieved via a flooding algorithm similar to the one described in Section 2.4.1.

Nowadays, most of the Internets routes are governed by OSPF [Moy98]. OSPF grew from an early link state algorithm which was designed to replace RIP [Hed88] in the ARPANET [MRR79]. It took over most of the functionalities developed in IS-IS [RFC04]. This evolution is shown in Figure 2.11.

2.4.5 Optimal Routing

Optimal Routing aims at optimizing the average global delay of a network, instead of only finding a shortest path. Therefore the problem of computing optimal routes in a network can be expressed as a optimization problem (more precisely, a nonlinear multicommodity flow problem) [CG74], this will be explained in more detail in Chapter 4.

Performance and costs are usually measured by metrics like delay, maximum line utilization, packet loss rate, stability of the system, and convergence time after failures. Current routing methods used in the Internet are not optimal in terms of any of these metrics. For instance, most of the routing protocols use hop-counts, or weights (derived from metric estimates) assigned to the lines in order to derive the routing tables which obviously cannot optimize any of the metrics mentioned above.

A direct way to improve the routing process (ie. optimize in some sense the metrics described above) is to use multiple paths between source-destination pairs. Current protocols, such as OSPF [Moy98] or IS-IS [RFC04] do this in a limited sense, by splitting the load to a destination along multiple shortest paths if these exist. This is an improvement, but still it is not optimal.

Optimal Routing has been researched and studied for many years. Its simplest form is Flow-Based Routing (see 2.4.1) where a network is modeled by a set of nodes which are connected by links. The capacity and amount of traffic on each link is known, and using this information the total cost to transport messages is minimized.

In 1974, Cantor and Gerla [CG74] modeled this problem as a optimization problem and used separation techniques to solve it. In 1977, Gallager [Gal77] proved the necessary and sufficient conditions for having a minimum-delay routing and introduced a distributed multi-path routing algorithm for this purpose that is loop-free at each iteration under the assumption that the network traffic is stationary or slowly changing. In 1999, Garcia-Luna-Aceves proposed a new algorithm, called Near-OPT [GLAVZ99], which constructs multiple loop-free paths, and then applies a similar method to the one proposed by Gallager. This algorithm is the basis of a multipath algorithm which will be described in Chapter 3. Figure 2.12 depicts the evolution of Optimal Routing algorithms. None of these methods were used in practice due to their complexity as well as low performance of processors and low capacity of links at the time these methods were proposed.

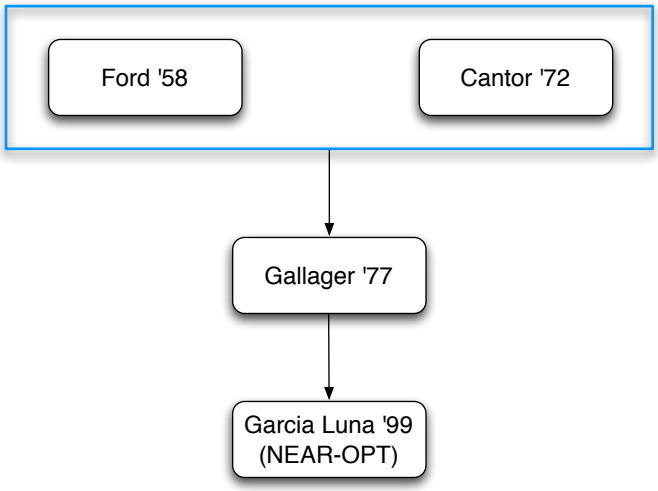


Figure 2.12: *Evolution of Optimal Routing Algorithms*

2.5 Summary

In this section, we have presented the fundamentals of networking along with a history of routing algorithms. Over time the OSI model has turned into a reference only model, while the TCP/IP model is virtually ubiquitous. Ethernet has evolved from 9600 bits per second beginnings to 10 Gbps with 100 Gbps at our doorstep.

Distance Vector algorithms have been mostly replaced by Link State ones. Primarily due to their shortcomings in terms of convergence time and their inability to support sophisticated metrics. Optimal Routing algorithms have for long been a mathematicians toy and a networks researchers dream, due to the fact that they were only implementable in a centralized manner.

In the next chapter, we will discuss the state-of-the-art in multipath routing. These algorithms attempt to reconcile the best from Link State or Distance Vector routing and Optimal routing.

Chapter 3

Multipath Routing

"If you want to succeed you should strike out on new paths, rather than travel the worn paths of accepted success."
– J. D. Rockefeller

3.1 Why Multiple Paths: Some justifications

Computer networks have grown over the past decades, this growth has lead to a rise in complexity of their topology. While increased complexity is often viewed negatively, it may be a blessing in disguise to networks because it can provide multiple paths towards a destination. If we are able to utilize these alternative paths intelligently, we can expect ta significant increease in network performance.

Figure 3.1 shows a complicated network consisting of multiple paths between a given source-destination pair. For example, consider source A and destination C: we have four possible paths from A to C (A-B-C, A-D-C, A-D-B-C, A,D-E-C). Most protocols nowadays, only consider a single path at any given instant. While such a decision reduces the diffculty in building correct routing tables, it comes with a major drawback: instead of using the entire capacity of the network, one will only use a relatively small fraction (in this case, only a fourth is used).

It has been proven that splitting traffic over multiple paths reduces delay while increasing he efficiency of the network [JMG93; Hui95]. If all packets of a given communication can be assigned to one path, then we will obtain higher network throughput as several communication can occur simultaneously and lower delays in both absolute terms and delay variations. Moreover, using

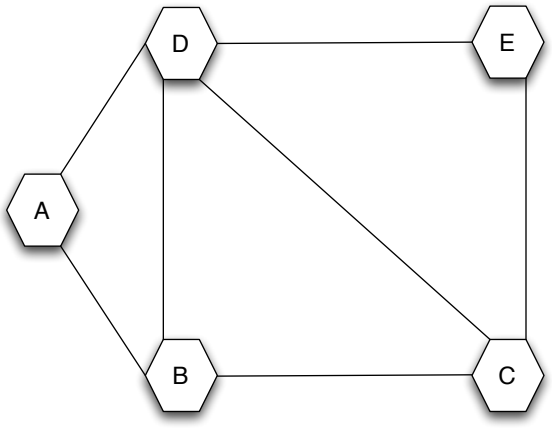


Figure 3.1: *A complex network*

multiple paths solves another problem which is encountered in single paths routing: link failure. If a link fails in a multipath scenario, then the traffic can simply be rerouted via an alternative path while a recalculation occurs or the failed path is re-established contrasting with single path routing where the routing tables must be recalculated and all traffic is lost in the meantime.

On the other hand, splitting traffic over multiple paths comes with its own problems. First, it is possible that packets on different paths encounter different path MTU (Maximum Transmission Unit) and therefore would require a router to break up packets according to the path they take, thereby negating the advantage of an advertised MTU. Second, debugging tools, such as traceroute or ping, are made inaccurate or even incorrect due to the use multipaths. Finally, a much more serious problem, is the phenomenon of variable delays encountered by packets on different paths which cause them to arrive out of order. In many TCP (Transmission Control Protocol [APS99]) implementations, out of order packets cause retransmissions which drastically reduce the performance of a network. A solution to this problem is suggested in Section 3.3.1.

Before we continue, we must introduce the concept of *flow*, which in this document is similar to the definition of a microflow in [NBBB98].

Definition. *A* flow *is a sequence of packets which is identified by its characteristics, such as source and destination address, source and destination port. Where characteristics could for example be same header values.*

There are two features which are desirable for multipath protocols:

- **Minimal Disruptions** - Since multipath protocol route traffic on many paths, the probability of one path failing or removed for an arbitrary reason is much greater than in the case of single path. It is therefore desirable to minimize the number of flows affected by a link removal.

- **Fast Computation** - The overhead required to compute the next hop to forward a packet or flow should be small when compared with single path protocols.

Multipath routing, essentially, asks the question of how to best split network traffic on a set of paths given some cost function. This is known as a flow maximization problem and is a very difficult problem to solve. Moreover, we are not looking for a centralized solution, nor a distributed solution where routers make their own decisions, but a solution based on a feedback loop where decisions are influenced by the evolving network conditions. The centralized solution is NP-Complete, but if we retain some network paths we can solve the problem in a bounded time by assigning a price to each link of the network. The price is the first derivative of the cost function evaluated at the offered traffic level. In order to estimate the offered traffic we must first be aware of all the routing decisions taken by all routers, and therefore this makes such algorithms difficult, if not impossible, to implement in a real network. Moreover, such approaches will inevitably suffer from scalability problems and therefore convergence problems. These problems make up a very active field of research.

3.2 Multipath Routing Schemes

As it has been shown in the previous section, single path algorithms, such as OSPF, RIP, EIGRP [Hed88; Moy98; AGLAB94], make inefficient use of the overall network bandwidth. In this section we present some research multiple paths algorithms whose evolution is shown in Figure 3.2.

3.2.1 Proposed Extensions to Single Path routing

Multiple Path Algorithm (MPA) [NST99] is an extension that could be added to OSPF [Moy98]. MPA constructs a subset of paths in a given network which satisfy a condition for loop freeness. This algorithm introduces the notion of a *viable next hop*, where initially the shortest paths are computed. A router then computes the difference between its neighbor's path length to the destination and the distance to get to its neighbor, if this distance is **less than** the distance from the router to the destination then the path is considered viable. More formally, for every neighbor N of router R the test in Equation 3.1 is performed:

$$D_N(R, X) - d(R, N) < D(R, X) \tag{3.1}$$

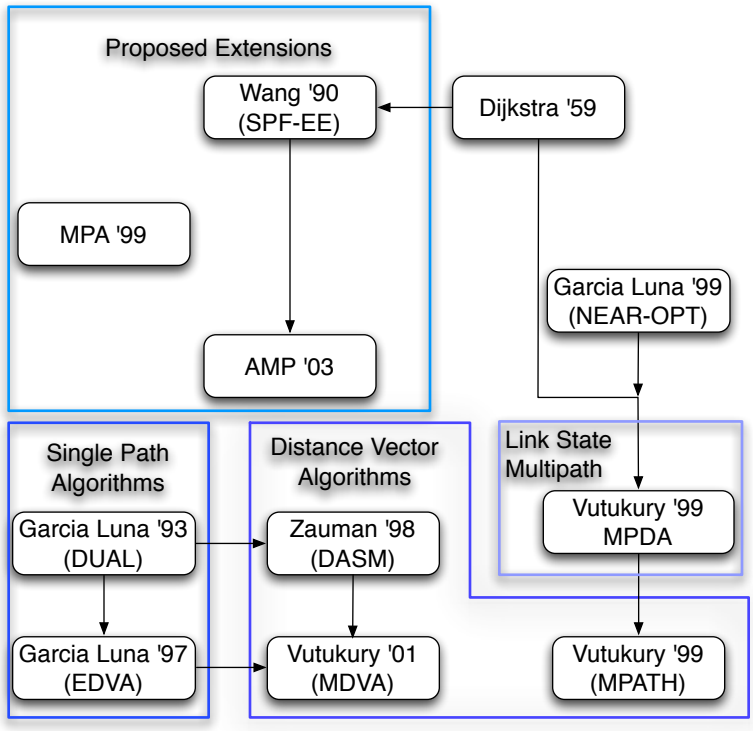


Figure 3.2: *The Evolution of multipath algorithms*

Where,

- $D_N(R, X)$ is the distance from neighbor N to destination X
- $c(R, N)$ is the cost of the link between R and N
- $D(R, X)$ is the distance from R to X.

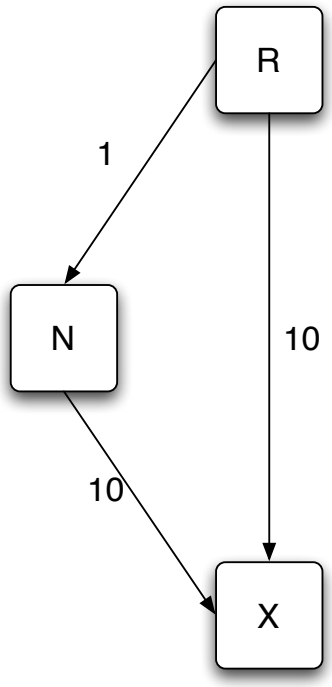


Figure 3.3: *An example of MPA.*

If this test is successful then the neighbor is considered a viable next hop. As shown in Figure 3.3, the path R-N-X is not the shortest path because its total length is 11 whereas the direct path R-X has a length of 10. According to the viability test in Equation 3.1, the path R-N-X can be used as an alternative and it does not create any loops.

The problem with classic routing algorithms is their inability to re-route traffic when one of the paths becomes congested or fails. In 1990, Wang proposed the idea of adding emergency exits for traffic when faced with congestion. Shortest Path First with Emergency Exits (SPF-EE) [WC90] behaves like a

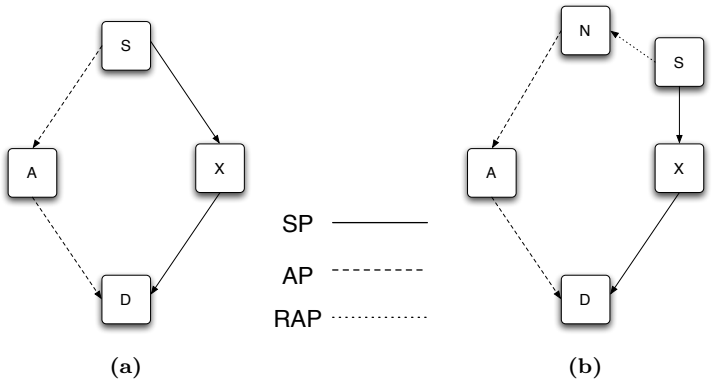


Figure 3.4: *An example of SPF-EE*

classic algorithm under light traffic conditions, but provides alternate routes when congestion or failures occur rather than updating the routes which is an expensive process.

Figure 3.4 illustrates the idea behind SPF-EE. If the network load is low then a source router S forwards packets along its shortest path to the destination D (denoted SP in Figure 3.4). In the event of some failure, two cases have to be distinguished:

1. The alternative path's next hop is downstream from the source router S.
2. All the neighbors to S, other than the shortest path next hop, are upstream.

In the first case, the packet will simply travel from S to the alternative path (denoted by AP in Figure 3.4) next hop (A) and then follow the shortest path from A to the destination as shown in Figure 3.4a. The second case is more complicated as there are no downstream possibilities (assuming that all routing table updates have taken place, and therefore S knows that its shortest path is down) shown in Figure 3.4b. Router S therefore sends a control packet to all its neighbors asking if they have an alternative path to the destination. If a neighbor router N finds a path then it sends the control packet back to S who then establishes a reverse alternative path (RAP), otherwise the neighbor propagates the request to its neighbor routers excluding S.

The use of a control packet in SPF-EE is problematic because it increases the packet delay significantly while an alternative is found. Moreover, if the source router has no other neighbors other than the shortest path, the algorithm simply fails. A better solution would be to precompute the alternative path set in order to minimize the delay in the event of a failure.

SPF-EE only re-routes traffic in the event of an overload or a failure, another approach would be to take advantage of the path redundancy continuously. Adaptive MultiPath (AMP) [GZRR03] does precisely that. AMP is based on providing routers with local network status information by exchanging so-called *backpressure messages* (BM). Thus, enabling routers to adapt continuously to the changing network loads.

When AMP detects congestion on a link, it attempts to shift traffic away from this link. In order to achieve this, a router R connected to the congested link sends to its neighboring routers a BM indicating their contribution to the congested link. The neighboring router which receives this BM then notifies its neighbors of their contribution and so on. The routers then attempt to forward the congestion-inducing traffic onto other routes. Figure 3.5 illustrates the dispersion of the BMs when congestion originates from the Y routers.

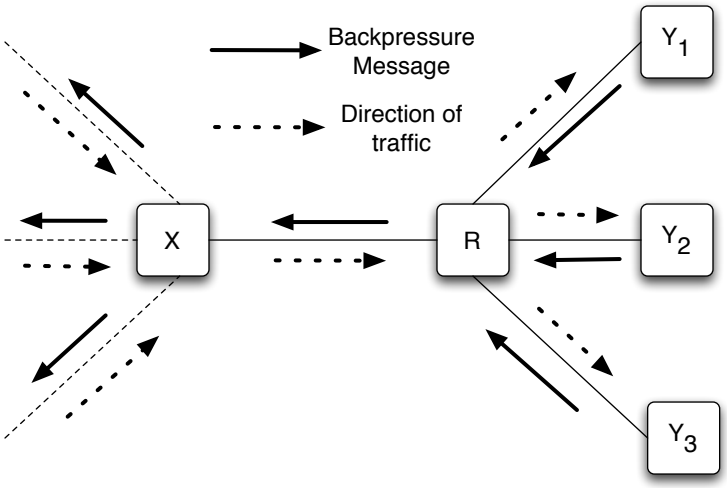


Figure 3.5: *An example of AMP.*

The issue with such an approach is that while router R knows how much traffic is sent to the individual Y routers, it generates a single value which is then sent back to router X in the form of a BM. Thereby the information which router X is interested in, namely its contribution to the congestion on a given path, is lost. By analyzing the structure of the BMs, we notice that every router needs to maintain a counter for each flow and store a reference of the flows origin because each router needs to compute a BM containing a value which is composed of the contributions of each individual flow, this clearly poses a scalability problem. The derivation of the BMs can be found

in Section A.2.

3.2.2 Multipath Routing Algorithms

In 1998, Zauman introduced a new algorithm for the computation of multiple loop free paths from a source to a destination called Diffusing Algorithm for Shortest Multipath (DASM) [ZGLA98]. DASM is based on a generalization of the Diffusing Computations introduced by Dijkstra [DS80] and therefore it is also a generalization of DUAL [GLA93] described in Section 2.4.3. DUAL only maintains a single next hop for any destination whereas DASM maintains a set of possible next hops. DASM introduces the concept of the *Shortest Multipath*, which is an extension of DUAL's loop free routing along shortest paths. In DASM, the Shortest Multipath is guaranteed to be acyclic at the end of the diffusing computation.

Definition. *A* shortest multipath *is defined by the directed acyclic graph obtained by the entries of the routing tables at each router in all paths from a source to a destination.*

In order to ensure loop freeness, DASM constructs the Shortest Multipath graph by forcing routers to select next hops whose distance to the destination is less than their own. Figure 3.6, illustrates the concept of Shortest Multipaths which contains the shortest path along with longer paths. It should be noted that each router computes the possible paths to the destination, therefore the shortest and longer paths at each router represent the shortest path and longer as computed from the current router.

A subsequent extension of DASM is presented in 2001 by Vutukury which provides loop freeness at every instant. Multipath Distance Vector Algorithm (MDVA) [VGLA01] is based on EDVA [XDla97] algorithm presented in Section 2.4.3. To provide loop freeness at every instant MDVA makes sure that at every instant during a diffusing computation, a node who reports a distance via a next-hop K must keep K as its next hop at least until the end of the computation. Simply put, by using the loop free conditions enunciated by EDVA and the method to build multiple paths in DASM coupled with the invariance of reported distances during a computation, MDVA is capable of computing loop free multipaths at every instant.

In link state algorithms, Vutukury proposed MPDA [VGLA99c] which provides multiple paths of unequal costs to every destination which are loop free at every instant. The idea is to first compute multiple loop free paths using the same method as Near-OPT [GLAVZ99], but then use heuristics built on Gallager's Theorem [Gal77] to allocate flows to the computed paths and thereby approximating the optimal result presented in Near-OPT. MPDA sends updates to its neighbors and awaits their acknowledgement before sending another update, contrasting with previous algorithms who used diffusing compu-

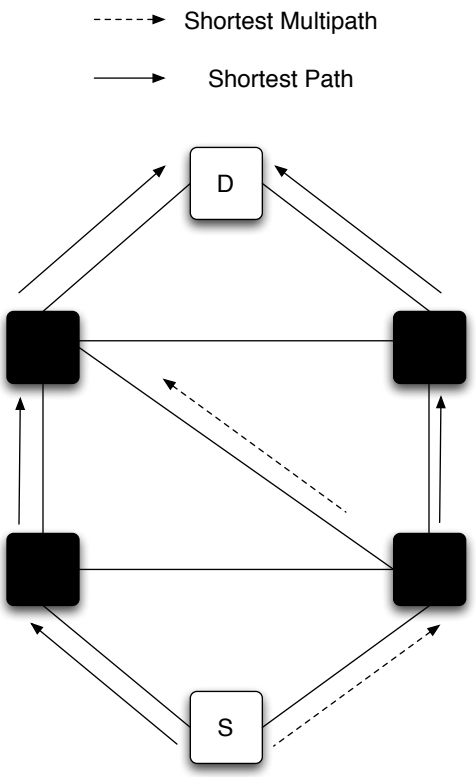


Figure 3.6: *An example of Shortest Multipath constructed with DASM.*

tations which span the entire network, MPDA only sends out information to its neighbors and therefore its synchronization only spans a single hop.

Along the same lines as MPDA, Vutukury proposed MPATH [VGLA99a; VGLA99b] which is a distance vector algorithm providing multiple loop free paths using only neighbor information. MPATH exchanges distances to destinations along with the predecessor to the destination, thereby allowing it to compute multipaths using only predecessor information. Similarly to MPDA, MPATH synchronizes only with its neighbors and therefore performs single hop synchronization. MPDA and MPATH differ only in the type of information their participating nodes exchange.

3.3 Multipath Routing in current IP networks

3.3.1 OSPF Extensions

Equal Cost Multipath

OSPF defines a multipath protocol, called Equal Cost Multipath (ECMP) [Moy98], which only considers alternative paths of equal length to the shortest path, ie. each router maintains the set of possible next hops whose subsequent paths are of equal cost to the shortest solution. ECMP suffers from out of order packet arrival if routing decisions are taken packet per packet. To solve this problem several solutions are proposed by [TH00], which each define flows and then each flow is assigned to an outgoing port.

1. **Modulo-N Hash** - The router performs a modulo-N operation over the hash of the packet headers which identity a flow. This method results in $\frac{N-1}{N}$ flow disruptions upon a link failure.

2. **Hash-Threshold** - The router computes a hash of the packet headers used to identify a flow. The output range of the hash function is mapped against the number of possible next hops such that a the value of the computed hash will indicate which next hop to use. This method results in between $\frac{1}{4}$ and $\frac{1}{2}$ of all flows to be disrupted upon a link failure. This method is analyzed in [Hop00]. The analysis of the disruptions is given in Section A.1.

3. **Highest Random Weight** - The router computes a key for each next-hop by computing a hash over the fields of a flow, as well the over the address for the next-hop [TR98]. The key which obtains the highest value is selected and therefore so is the next-hop. This method involves N times the number of computation as in Modulo-N hash, but only $\frac{1}{N}$ flows are disrupted upon a link failure.

An alternative approach is proposed by Vutukury [VGLA00] in which a key is appended by the source router to each packet belonging to a flow. This key is then used to by each router along the path to the destination to determine the next hop.

OSPF Weights

Fortz and Thorup [FT00] and similarly Wang [WWZ01] propose methods to optimize the weights assigned to OSPF links. This centralized approach requires that the weights be precomputed assuming that the required load is known beforehand and that the network is quasi-static. Such assumptions make this solution unrealistic for an arbitrary network.

A typical and naive approach is to include a load metric with the standard distance metric [Hui95]. The combination of these two metrics is then used to compute the shortest paths. This process therefore creates a feedback loop:

1. A router receives an update for the different link loads and computes the new shortest paths.

2. The traffic is now re-routed according to the new shortest paths. This causes the link loads to change and therefore,

3. A new update message is generated and distributed.

Feedback loops cause routes to oscillate and therefore create out of order packets, which as we have seen cause a significant performance loss. The frequency of the oscillations depends on the duration of the feedback loop. If the loop is slow, the number of oscillations is small but very little is gained over standard OSPF. On the other hand, if the loop is fast and many oscillations occur, we can expect many out of order packets and therefore very poor network performance.

Optimized Multipath OSPF

We have seen in the previous subsections techniques which attempt to divide traffic more or less equally among the available path sets. In 1998, Villamizar proposed OSPF Optimized Multipath (OMP) [Vil99] which takes into account path loads to distribute traffic efficiently among paths. OMP uses mechanisms already available in OSPF to flood the link information within a subnet [Col98].

Initially the algorithm computes the initial set of possible paths. Then the load on each link is iteratively adjusted according to the load information received from other routers. The following steps describe the process of building the path sets:

1. Using a standard shortest path first algorithm the shortest paths to all destinations are computed.

2. Construct a set of possible alternative paths by inspecting paths from neighbors to destinations and combining them with the links from a given router and these neighbors.

3. Paths which then have equal costs or are within an acceptable margin of the minimum cost are kept. In order to avoid loops, paths which are longer than the minimal cost will only be kept if their immediate neighbor closer to the destination than the origin router.

Once the possible path sets are established, the router can share load onto the paths. Routers will monitor the links and flood load information when the load changes. Depending on the overall network load, the flooding interval can be as large as 20 minutes if the load is low and 30 seconds otherwise. Every time a router receives new load information it will reexamine its load sharing as explained below:

1. Examine all links and determine the critical links, and therefore establish the links with the highest load.

2. Reduce the load on paths containing critical links.

3. Increase the load on paths which do not contain critical links.

OMP is still considered an experiment and has not yet been deployed in a production environment. However, it is very promising. In the next section we will discuss other load sharing algorithms which are research experiments.

3.3.2 Alternative Methods

In [BEZ92], the authors introduce a routing scheme which is similar to the idea presented by SPF-EE [WC90]. Under light traffic conditions, traffic is routed along the shortest path but as the load increases and congestion becomes significant the algorithm attempts to shift traffic onto less congested paths. To achieve this the algorithm uses two main concepts. The first is the classification of link states to identify more easily congested links. The second, is the computation of acceptable alternative paths which are within an acceptable margin of the shortest path in terms of hop count.

Load sensitive routing has always been hampered by the signaling overheads. Moreover, oscillations occur frequently in the path selection because the routing decisions are taken on out dated information. In [SRS99], the authors suggest a hybrid approach to solve the efficiency and stability issues, which relies on identifying long lived traffic flows and routing them separately.

The algorithm provides load sensitive routing for long lived flows while forwarding short lived flows on preprovisioned static paths. In order to identify long lived flows and ensure stability, routers relate the detection of long lived flows to the timescale of signaling messages.

A similar approach is suggested in [LC01], where packets are also grouped into flows to avoid out of order arrivals and long lived flows are also differentiated. Although in [LC01], the authors suggest that a minimum of flow states should be maintained by the router. This results that a flow is either routed on the primary (shortest) path or on the secondary (alternate) path depending on whether it is long lived or not. Figure 3.7 shows this load control mechanism: the flow classifier, identifies long lived flows and stores them in the flow table. Then the packet forwarding module forwards the packets according to the contents of the flow table.

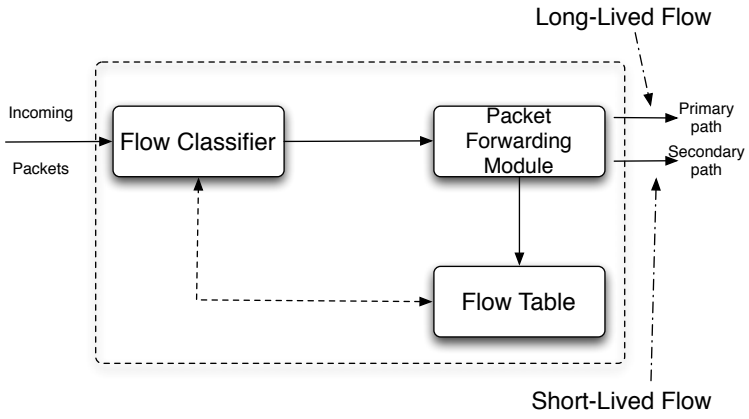


Figure 3.7: *The flow classifier.*

A dynamic multipath routing (DPMR) scheme to improve resource utilization of a network carrying real time traffic by re-routing on going flows through shorter routes is proposed in [DD01; DDT01]. These approaches alleviate instantaneous congestion by allowing rerouting of a flow through a longer route. In contrast, in this paper, rerouting of an ongoing flow is allowed only if the new route is more preferable (i.e., shorter) than the current one. DMPR scheme works based on the route length, and is independent of the network congestion.

3.4 Summary

In this section, we have presented the current state of the art in terms of multipath protocols. We have seen that the current protocols do not use multipath in a dynamic sense. Some protocols only consider paths of equal length, but these paths may not always be available. Other protocols introduce mechanisms for quick failover in case a path breaks. Finally, some protocols are simply not implementable as they will require too many resources. In the next sections, we will introduce models for multipath protocols and describe our congestion-aware protocols.

Chapter 4

Multipath Theory and Models

"In theory, theory and practice are the same.
In practice, they are not. "
–Albert Einstein

In this chapter we will start by detailing the various graph algorithms used to compute shortest paths both in the Link State and Distance Vector scenarios. Next, we will describe the concept of Selfish routing which will provide a formal definition of the consequences of congestion insensitive routing. Finally, we will give a brief introduction and present the most remarkable results of Queue Theory which will be used later in this document to model our Multipath routing protocol.

4.1 Graph Algorithms

The algorithms presented in this section do not provide multipath routes but only shortest paths. That said, they can be used, with some minor modifications, to build multipath routes as we will see in Chapter 5.

4.1.1 Dijkstra's Algorithm

Dijkstra's Algorithm was proposed by Edsger Dijkstra in 1959 [Dij59], it is a graph search algorithm which given a graph composed of a set of edges, nodes and edge weights, finds the shortest path from a given source to all other nodes. It is therefore obvious that this algorithm is fundamental in Internet routing.

The idea behind this algorithm is simple, consider a city's road system with intersections. You wish to find the shortest path between two points

in this city. At your start point, you build a list of intersections which are directly connected to it. Then, select the intersection that is closest to your destination and mark the road and the starting intersection used. Next, repeat these steps considering the current intersection as your start point, considering one intersection at each iteration. Once the list of intersections is empty, the algorithm ends, and your shortest path is the the one consisting of all the marked intersections.

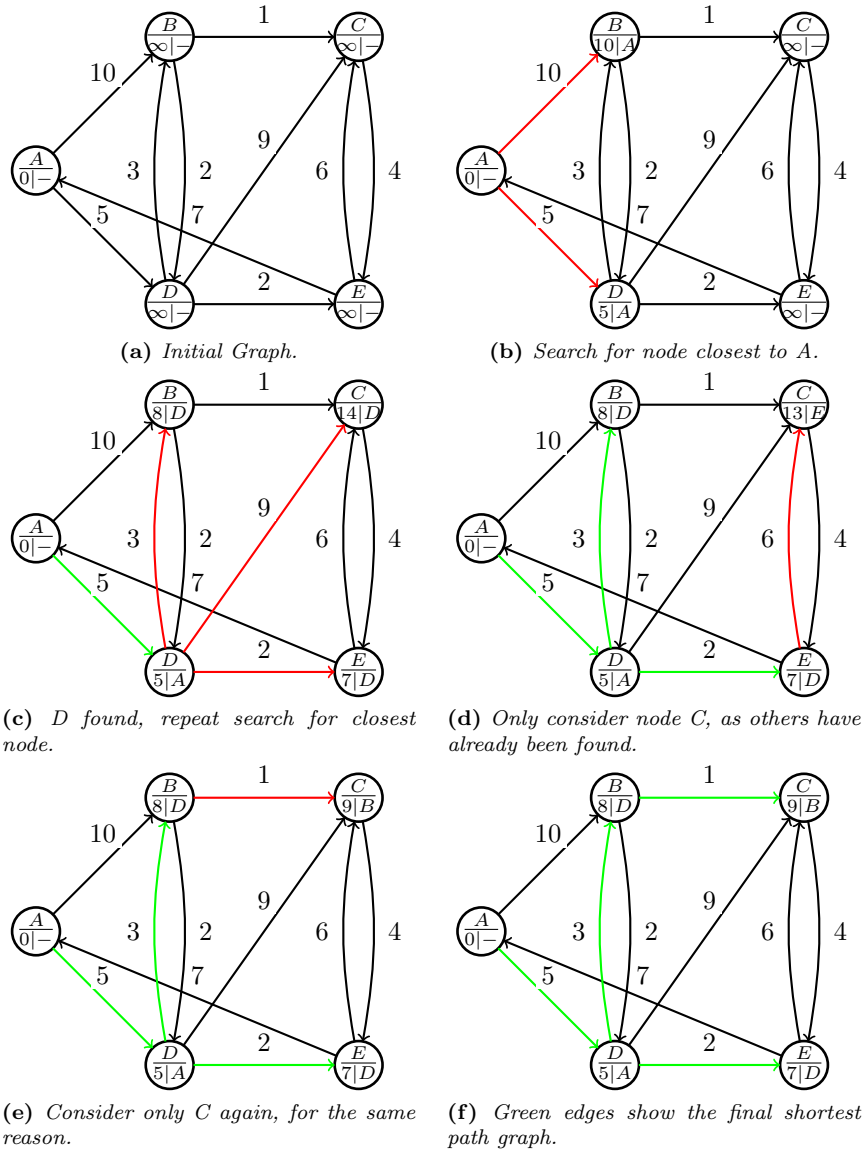
We will now give the algorithm description in textual form, the pseudo-code can be found in Section B.1, followed by a running example of the algorithm in Figure 4.1, the notation $\frac{x}{y|z}$ is used where x is the node name and y is the distance to the initial node and z the previous node.

1. Set the distance of the initial node to zero and infinity for all the others.
2. Set all nodes to unvisited except the initial node (current node).
3. For unvisited neighbors of the current node, compute their distance from the initial node. If this distance is less than their current distance, replace their current distance with the computed one.
4. Once the distance for all unvisited neighbors have been computed, mark the current node as visited.
5. If no unvisited nodes remain, then the algorithm is finished. Otherwise pick the node with the smallest distance from the initial node and set it as the current node and repeat from step 3.

As is shown by Figure 4.1, Dijkstra's Algorithm grows a tree from a given source to all the other nodes for which the distance from the source to all the other nodes is minimal. We initially start with the original graph at node A which has a cost of zero. In the second step, we start at A and search for its closest neighbor (red arrows), we than remove A from the list of unvisited nodes (Q). Next, having found D as the closest neighbor, we can safely mark the edge from A to D as part of the shortest path (green arrow). We then repeat the search for D and find E and remove D from Q. At E we only consider the edge to C as the others link to nodes which are no longer in Q. At the end, we obtain the green graph which is the shortest path from A to all the other nodes. Table 4.1 shows the intermediate steps in the algorithm, the notation x/y is used where x represents the distance from the initial node and y is the precedent node in the shortest path.

4.1.2 Bellman-Ford Algorithm

Bellman-Ford's algorithm [Bel58] is the basis for Distance Vector routing protocols. It was proposed simultaneously by Bellman and Ford in 1958. It produces the same result as Dijkstra's algorithm but it is more flexible, namely it



(a) *Initial Graph.*

(b) *Search for node closest to A.*

(c) *D found, repeat search for closest node.*

(d) *Only consider node C, as others have already been found.*

(e) *Consider only C again, for the same reason.*

(f) *Green edges show the final shortest path graph.*

Figure 4.1: *Running Example of Dijkstra's Algorithm*

I	A	B	C	D	E	Q	Path Nodes
0	0/-	∞/−	∞/−	∞/−	∞/−	A-E	∅
1	0/-	10/A	∞/−	5/A	∞/−	B-E	A
2		8/D	14/D	5/A	7/D	B,C,E	A,D
3		8/D	13/E		7/D	B,C	A,D,E
4		8/D	9/B			C	A,D,E,B
5			9/B			∅	A,D,E,B,C

Table 4.1: *Step by Step of Dijkstra's Algorithm*

allows for negative edge weights[1]. It is a decentralized algorithm which only requires nodes to inform its neighbors of their distances to other nodes. Then, each node receiving this information picks the shortest advertised weight.

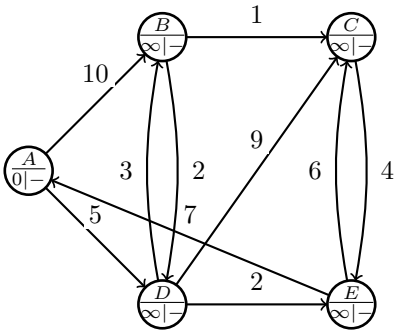
In a network, routers which use this algorithm maintain a distance table containing an entry for each of the nodes in the network. By looking up in the distance table, a router will know where to send traffic intended for a particular destination. Below we give a textual description of the algorithm, whose pseudo-code can be found in Section B.2 followed by a running example of the algorithm.

- Initialize the graph by setting all distances to ∞ and the source's distance to zero.
- If the distance from the source to a neighbor using a particular edge is shorter than the current distance, overwrite the current distance with the new one. Repeat this step for all nodes and reconsider all edges in the graph.
- Finally check for negative edge cycles by computing the distance from a node to its neighbor, if this distance is smaller than the stored distance then the graph contains negative edge weights.

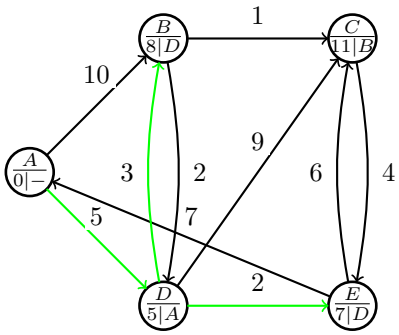
The order in which the edges are visited is not specified by the algorithm, therefore we will opt to visit, from left to right, the top horizontal edges first, then the vertical ones, followed by the bottom horizontal ones, and leave the diagonal ones last. The notation is the same as in the case of Dijkstra.

It may seem in Figure 4.2, that Bellman-Ford's algorithm is extremely efficient. This is not the case, because for each node it analyses every edge in the network, this is not shown for reasons of brevity and clarity. Bellman-Ford therefore requires O($|V||E|$) operations, whereas Dijkstra's algorithm requires

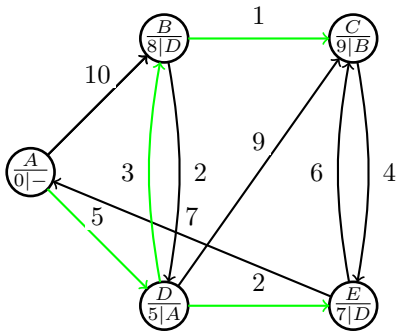
[1]While this is an interesting fact, it is useless for internet routing. Indeed, negative weights have no physical meaning.



(a) *Initial Graph*



(b) *At node B, update distances according to edge traversal order*



(c) *Update distance for node C, since it was missed in the last iteration.*

Figure 4.2: *Running Example of Bellman-Ford's Algorithm*

$O(|E| + |V|log|V|)$, where $|V|$ is the number of vertices's and $|E|$ the number of edges.

4.2 Selfish Routing

Selfish routing describes the overall negative effects on the performance of a network when routing decisions are taken selfishly. It is clear that the travel time between two points is highly dependant on the number of users using a given route. Yet, most of us will always opt for the shortest path that gets us to our destination fastest, regardless of the effect this decision has on other users. This is referred to as Selfish Routing [Rou05].

Routing on the Internet is insensitive to congestion and therefore suffers from the phenomena explained by Selfish routing. Indeed, shortest path routing is analogous to commuters traveling to work each choosing the shortest path and thereby delaying everyone. Figure 4.3 shows a network (referred to as Pigou's Example [Pig20]), in which the total traffic is represented by one, with two paths where the upper one has a delay of one time unit regardless of congestion and the lower path's delay is a function of the congestion. It is reasonable to assume that all traffic will be selfishly routed along the lower path, and therefore that it will be delayed by one time unit. On the other hand, if we consider that the traffic is split equally between the two paths, the overall delay is $\frac{3}{4}$. Indeed, the traffic on the upper path suffers a delay of one time unit but now the traffic on the lower path is only delayed by half a time unit. Therefore, we easily see that no traffic is worse off and moreover half of the traffic is delivered significantly faster. Clearly, the phenomenon illustrated in the first situation could be offset by increasing the link capacities but this is not a tenable nor scalable solution in the longterm.

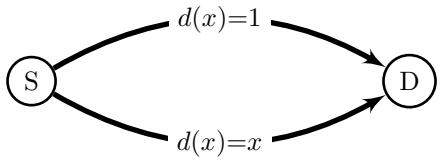


Figure 4.3: *Pigou's Example*

Selfish Routing also provides us with a further justification of deploying multipath routing protocols. Before we develop further this justification we must first proceed to the definitions of a few concepts which are more formally detailed in Section A.4. First, we must redefine the notion of a flow which will be limited to this section only.

Definition. *A* flow *f is interpreted as the aggregated routes chosen by the traffic, with f_p measuring the amount of traffic on route p.*

Note: The term flow used here differs significantly from our previous definition. This definition only hold for this section and during the rest of this document we will revert to our previous definition.

- *Nash Flow* can be seen as a flow routed selfishly, as in the first case given in Figure 4.3.
- *Optimal Flow* is the flow with minimum possible delay, as in the second case of Figure 4.3.
- *Price of Anarchy* is the *worst-possible* ratio between the delay of a Nash flow and that of an optimal flow.

With these definitions at hand and if we consider multipath networks with only linear delay functions, Selfish routing tells us that the upper bound[2] on the *Price of Anarchy* is $\frac{4}{3}$ (see proof in Section A.4). This result further justifies our exploration of multipath networks, because in a multipath network with selfish users it would be expected that additional problems in congestion would arise and therefore deteriorate the performance of the network. This result shows us that this is not the case, and that even in a multipath network with only selfish users there is "only" about a 33% loss in performance, which gives us hope that we may improve the performance of a multipath network significantly while paying a relatively small price in the worst-case scenario (ie. only shortest paths available). Indeed, as we will see in the next chapter, our protocol attempts to leverage all feasible alternative paths and defaults to the shortest path (selfish route) if the alternative paths are not available. As the delay functions increase in degree (ie. polynomial delay functions), the Price of Anarchy increases as well.

4.3 Analytical Models

In this section we present some analytical models based on Queuing Theory which will allow us to model multipath protocols. It should be stated that an exact analysis of multipath communication is too complex [Kle72], however the independence assumption of Section 4.3.1 simplifies the calculations and provides a reasonable approximation.

[2] An upper bound also exists for all polynomial delay functions, which is given in Section A.4, but for the sake of clarity we only refer to linear cost functions here.

4.3.1 Queue Theory

A queuing system [Kle75], as shown in Figure 4.4 is characterized by the following parameters:

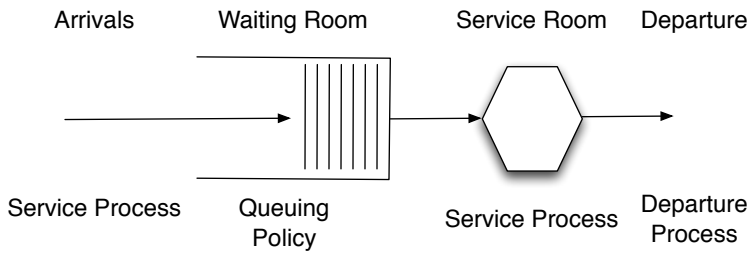


Figure 4.4: *A standard queue.*

- *Arrival process (A)* is defined by the distribution of the inter-arrival time of clients into the queue. The only interesting arrival process in this thesis is exponential (Markovian).
- *Service time distribution (B)* represents the time required to serve a client present at the top of the queue. We will only deal with Markovian distributions.
- *Number of servers (m)* present in a single queue. We will deal with models containing one or more servers.
- *Capacity (K)* of the queue is the maximum number of clients which can be present in the system at any given moment.
- *Population size (N)* represents the potential number of clients whether finite or infinite. We will consider situations with infinite populations.
- *Queuing policy (D)* defines in which order the next client is selected from the queue, eg. first in, first out (FIFO).

Kendall's Notation is often used to identify the type of queue used which is denoted by A/B/m/K/N/D. The parameters K, N, and D are often omitted indicating that they are either infinite or that a FIFO policy is used.

The arrival and service time processes can be either of the following:

- *Markovian* - A Poisson distribution [Kin93] where the inter-arrival times are independent and exponentially distributed.

- *Deterministic* - constant inter-arrival/service time.
- *General Distribution* where the underlying probability density function is arbitrary.

It is worth presenting some general results and notations which apply to all queuing systems:

- λ is the average inter arrival rate of clients into the systems.
- $\frac{1}{\mu}$, the average service time per server.
- ρ is the utilization factor of the queue ($\frac{\lambda}{\mu}$ and $\frac{\lambda}{m\mu}$ in the multi-server case). For the queue to be stable, we should have $0 \leq \rho < 1$.
- T is the total average time in the system (queue + service).

$$T = W + \frac{1}{\mu} = \frac{1}{\mu - \lambda} \tag{4.1}$$

- W is the average waiting time in the queue.

$$W = \frac{\rho}{\mu - \lambda} \tag{4.2}$$

- $\bar{N}$ is the average number of clients in the system given by Little's Result [Lit61].

$$\bar{N} = \lambda T \tag{4.3}$$

- $\bar{N}_q$ is the average size of the queue.

$$\bar{N}_Q = \bar{N} - m\rho \tag{4.4}$$

- P_N is the probability that there are N clients in the system.

In the next sections we will present remarkable results for both the M/M/1 and M/M/m queues. We will also describe Kleinrock's Independence Assumption.

M/M/1 Queue

The M/M/1 queue is composed of exponentially distributed inter-arrival process and service times along with a single server model. The M/M/1 queue is the simplest form a queue can take, while still remaining interesting. While being simple its behavior is similar to other more complex cases.

Since the inter arrival distribution is Poisson, the arrival rate is given by λ and the service time is $\frac{1}{\mu}$, which gives us directly the following relations:

$$\bar{N} = \frac{\rho}{1-\rho} \tag{4.5}$$

Using Little's Result (Equation 4.3) and Equation 4.4, we obtain the following relations for W and T.

$$W = \frac{\rho}{\mu(1-\rho)} \tag{4.6}$$

and,

$$T = \frac{1}{\mu(1-\rho)} \tag{4.7}$$

All the relations given for $\bar{N}$, T, and W demonstrate common behavior with respect to the utilization factor ρ, more precisely they behave inversely to $1-\rho$. Therefore, as ρ tends towards one the average delays and queue sizes tend towards infinity as shown in Figure 4.5.

M/M/m Queue

We will now consider a generalization of the M/M/1 queue to m servers. In this model, a single queue forms the entry to the system and a collection of servers will handle the first client at the head of the queue. As previously, λ is the arrival rate and $\frac{1}{\mu}$ is the average service time. Also, we have here that $\rho = \frac{\lambda}{m\mu}$.

Before presenting the steady state relations relative to an M/M/m queue, we must first distinguish the situations where the queue is busy and when it is not, which help establish the steady state relations. This is given by the probability of finding that all servers are busy when a new client arrives:

$$P_Q = \frac{p_0(m\rho)^m}{m!(1-\rho)} \tag{4.8}$$

where,

$$p_0 = \left[\sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!(1-\rho)}\right]^{-1} \tag{4.9}$$

P_Q is known as the *Erlang C formula* [Erl17] and is widely used in telephony. Now we give the following relations for the number of clients in the system and in the queue, respectively. The complete derivation of these relations can be found in citeKLEVol1 and other Queuing Systems literature, the author only wishes to remind them here.

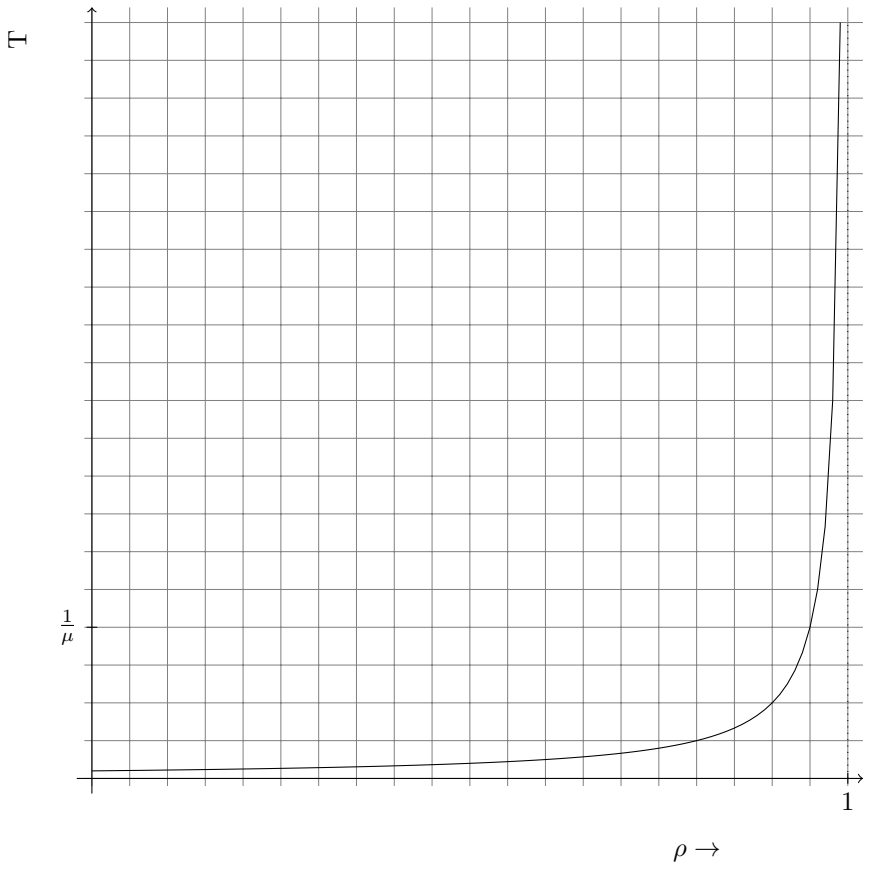


Figure 4.5: *Average delay as a function of ρ.*

$$\bar{N} = m\rho + \frac{\rho P_Q}{1-\rho} \tag{4.10}$$

and,

$$\bar{N}_Q = \frac{\rho P_Q}{1-\rho} \tag{4.11}$$

Using Little's Result (Equation 4.3) and Equation 4.4, we obtain the following relations for the average time (network delay) and the average waiting time (forwarding and processing time) spent in the system, respectively.

$$T = \frac{1}{\mu}\left(1 + \frac{P_Q}{m(1-\rho)}\right) \tag{4.12}$$

and,

$$W = \frac{P_Q}{m\mu\,(1-\rho)} \tag{4.13}$$

Kleinrock's Independence Assumption

In the two models, presented above, we have assumed that both the arrival and service times respect the Markovian property, ie. the future state of the system depends only on the present state. Moreover Burke's Theorem [Bur56] states:

Theorem 1. ***Burke's Theorem.*** *The steady-state output of a queue with N channels in parallel, with Poisson arrivals and message lengths chosen independently from an exponential distribution is itself Poisson-distributed.*

Consider now the situation, which exists within data networks, where many queues interact in the sense that the output from one queue is the input of another (or possibly several others). Given Burke's Theorem and the models we have presented, one might think that delays in data networks are simple to obtain via such models. Unfortunately this is not the case, the message (or packet) inter-arrival time and the message lengths beyond the first queue in the network become strongly dependent. Due to the fact that the service times at each queue are a function of the message length and therefore the arrival time at each subsequent queue is no longer Markovian [Kle72].

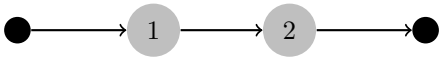


Figure 4.6: *A tandem network.*

Figure 4.6 illustrates two queues in tandem where packet lengths are exponentially distributed and independent of each other as well as the inter-arrival times at the first queue. We can therefore state that the first queue follows the M/M/1 model, but we cannot say the same about the second queue because the inter-arrival times at the second queue are strongly correlated with the packets lengths. Indeed the inter-arrival time at the second queue is **equal** to the transmission time at the first queue. As an analogy to this situation, consider a truck traveling on a narrow busy road along with several fast cars. Typically the truck will see an empty road ahead of it while it is being closely followed by a line of cars. The dependence between inter-arrival and message lengths (and therefore service time) is a source of great mathematical complexity for the analysis of queuing networks, in which even the simple case of the tandem network has no known exact solution [BG92]. A full mathematical demonstration of this dependence is given in Section A.3.

As previously shown the dependence appears for queues which are internal to the network. Therefore, one might ask: Why is there a difference between the initial (network entry) queue and internal queues? The answer is straightforward, the initial queue receives its messages from an external source consisting of many subscribers (in our case people or computers) which are all generating messages. The overall message generation by the subscribers exhibits an independence [Kle72] since one message is different from one person to the next. In a general network a similar situation exists. Indeed, more than one queue can deliver messages to any given queue, similarly any given queue is receiving messages from many other queues. If we accept this observation then we can define the following assumption:

Theorem 2. ***Independence Assumption.*** *Each time a message is received at a queue within the net, a new length v is chosen for this message from the following distribution:*

$$P(v) = \mu e^{-\mu v}$$

Obviously, such an assumption does not correspond to the reality with a general network. Nevertheless, it results in a far simpler mathematical modeling of networks, while still maintaining an acceptable degree of accuracy [Kle72].

4.4 Summary

This section has presented the modeling tools that we have used to describe our protocols. As we have shown, it is clearly beyond the scope of this document to attempt to describe multipath networks accurately due to the sheer mathematical complexity involved.

We have also discussed and given justification for deploying multipath protocols. While the use of multipath may seem evident, using longer paths rather than only the shortest path is less clear. We have seen that routing selfishly (ie. always onto the shortest path), causes the performance to drop significantly.

Chapter 5

The MultiRoute Family

"There is always a better strategy than the one you have;
you just haven't thought of it yet"
– Sir Brian Pitman

We now present the MultiRoute suite of congestion-aware inter-gateway routing protocols. But first, we should define the path discovery strategy employed by MultiRoute variants as it is identical and fundamental to all protocols. Then, we will describe MultiRoute Monitoring Protocol (MMP) which is responsible for the dissemination of congestion information throughout the network. While it differs slightly from protocol to protocol, its essential basis stays unchanged within each protocol.

Next, we will follow RFC 1264 [Hin91] which describes the procedures for creating and documenting standards on routing protocols. While this RFC has been obsoleted for practical reason, the author still believes that some aspects of the RFC can be used.

5.1 Path Discovery

Modern computer networks may offer many paths from any source to any destination but current protocols only employ the shortest path or make use of multiple paths of minimum cost if they exist.

The first goal of any routing protocol is to discover the shortest path, while a multipath protocol must also discover the possible alternative paths. We have employed a technique similar to the one present in IGRP [Bos92], which allows a router to forward packets through paths whose length is less than the product of the shortest path length and the variance factor. Since such an

approach may lead to routing loops[1], we have employed a slightly modified Relaxed best path criterion [Vil99].

5.1.1 MultiRoute Path Construction (MRPC)

MultiRoute's path construction algorithm relies heavily on Dijkstra's Shortest Path algorithm [Dij59]. However, the idea here is not only to discover the shortest paths between any source-destination pair, but also find all the available paths whether they are of equal length or within an acceptable delta of the shortest path length. This is achieved through the use of two parameters to the algorithm, the first represents the tolerable cost deviation (δ) which we accept, the second being the maximal hop count deviation (H). Both of these parameters are set by the network engineer, and should be chosen carefully as they directly affect the latency versus throughput trade-off.

The MRPC algorithm can be explained as a two phase algorithm. First, the algorithm constructs a set of paths identical to OSPF, ie. the shortest paths, each shortest path cost is then set as the reference cost. Second, considering the δ parameter, the algorithm now computes the potentially longer alternative paths. Let the value α to be the shortest path from some node A on route to a destination node B, and set β as the length of the alternative path which passes through node C. The algorithm, then, visits the alternative path possibilities, by checking whether $\beta \leq \alpha + \delta$, and if so it adds this alternative path to the list of available paths. By the end of the algorithm we are guaranteed to obtain a set of alternative paths whose lengths are no longer than δ plus the shortest path length (See proof in Section B.3).

Figure 5.1 shows the output of the algorithm run with $\delta = 3$ *and* $H = 3$ and initial source 1 and destination 6. The shortest path is 1-8-4-7-6 whose cost is 15. It can be easily seen that the two other paths, 1-2-3-5-6 and 1-8-4-5-6, have costs 18 and 16 respectively. These results respect the property that no alternative is longer than the reference cost plus δ.

It is clear that such a modification to Dijkstra's Shortest path algorithm introduces loops as can be seen in Figure 5.1. A rather simple solution to this problem is to only select paths whose next-hop is closer to the final destination. This is called the Relaxed Best Path Criteria and guarantees that no loops can appear as we route only onto shorter paths. We apply a slight modification here, and state that we shall only route onto a path whose next-hop is closer in terms of distance to the final destination with respect to the ingress port. Since, a router may have multiple distances to a given destination associated to different paths (and therefore ports), we perform computations with respect to the ingress port a packet came on rather than the ingress router.

[1] Routing onto paths which are longer than the shortest path is known to lead to routing loops.

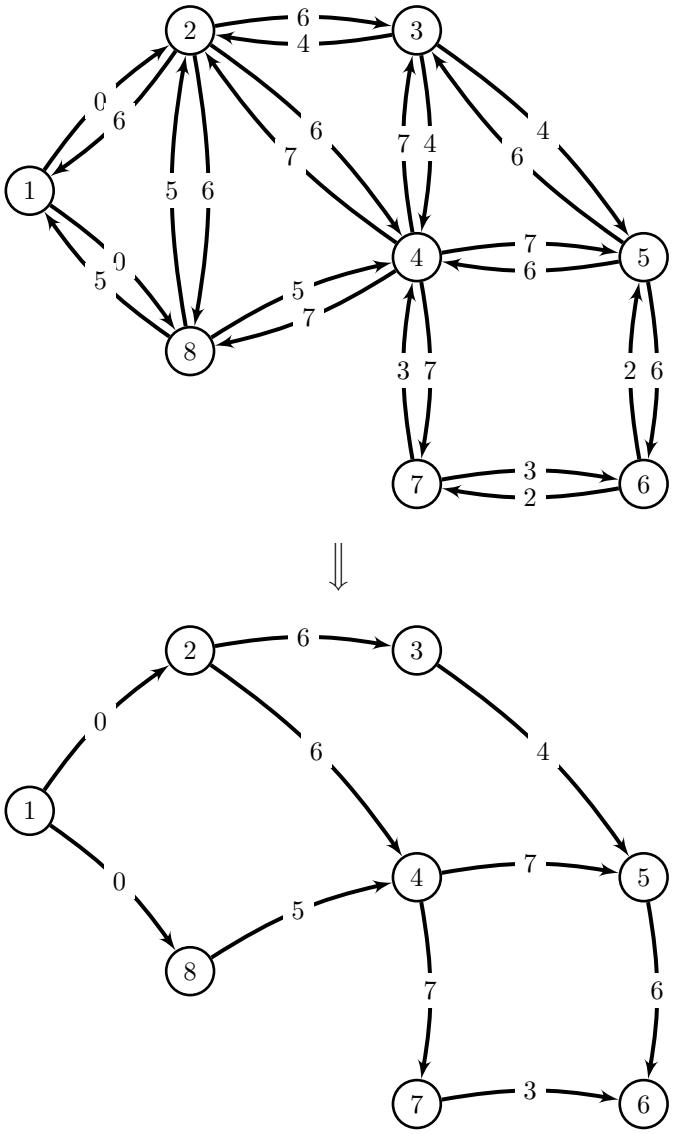


Figure 5.1: *Output of the MRPC algorithm*

In order to avoid pathological cases, the hop count must be kept within a reasonable value compared to the shortest path's hop count. For example, consider a graph where the shortest path is made up of links whose costs are much greater than one and an alternative path is made of many links whose costs are one. It could happen that, the alternative path was selected, much longer in terms of hop count but close in terms of cost. This situation must be avoided as the latency for communication will be heavily affected[2]. The process for pruning such paths is similar to the one described above.

5.1.2 Algorithm Sketch

In the previous section, we mentioned that the MRPC algorithm can be seen as a two phase process. While this is a practical way to visualize and understand the algorithm, the implementation involves only a single pass over the network graph.

Starting with a network graph and the set of associated vertices and an initial vertex S, we will detail the step taken by the algorithm:

1. Set the distance of the initial node to zero and infinity for all the others.

2. Set all nodes to unvisited except the initial node (current node).

3. For unvisited neighbors of the current node, compute their distance from the initial node. If this distance is less than their current distance, replace their current distance with the computed one and reinitialize the neighbor list of predecessors and append the current node to its predecessors.

4. If the alternative distance is less than the current shortest distance plus the tolerance value δ, then append the alternative node to the predecessors list, as well.

5. Once all the unvisited nodes have been processed, mark the current node as visited.

6. If no unvisited nodes remain, then the algorithm is finished. Otherwise pick the node with the smallest distance from the initial node and set it as the current node and repeat from step 3.

[2]Practically, latency is significantly affected by the delay in the router

By running the algorithm on Figure 5.1, we obtain the following output:

V_{min}	1	2	3	4	5	6	7	8	Q
0	0/-	∞/−	∞/−	∞/−	∞/−	∞/−	∞/−	∞/−	1-8
1	0/-	0/1	∞/−	∞/−	∞/−	∞/−	∞/−	0/1	2-8
2,8		0/1	6/2	{6,5}/{2,8}	∞/−	∞/−	∞/−	0/1	3-7
3,4			6/2	{6,5}/{2,8}	{10,12}/{3,4}	∞/−	{13,12}/4		5-7
5,7					{10,12}/{3,4}	{15,16}/{7,5}	{13,12}/4		6
6						{15,16}/{7,5}			∅

Figure 5.2: *Step-by-Step of MRPC according to the algorithm.*

The pseudocode and the proof for the algorithm can be found in Section B.3

5.2 MMP - MultiRoute Monitoring Protocol

With each congestion-aware routing protocol is associated a monitoring protocol which provides routers with crucial network status information. MMP achieves this goal by employing an innovative representation of a routers routing table, which then enables each router to represent the actual congestion value as it wishes. In this section, we will describe the fundamentals of MMP along with its associated packet structure.

MMP is an in-band monitoring protocol as opposed to an out-of-band protocol, such as SNMP [CFSD90] or sFlow [WLL04]. Out-of-band protocols are not adapted to the design of a congestion-aware routing protocol, due to the simple fact that they require a central entity (or management station) to collect the statistics from the monitored devices. Such a management station creates a single point of failure, which is obviously unacceptable for a routing protocol which has to be resilient to failures.

Moreover, a centralized approach poses a major timing problem. Network statistics are all sent to the management station, from there they are sent back to the concerned router in a format the router can interpret and take an appropriate action. It is clear that the time required for the statistics to travel from the router, be processed at the management station and finally back to the relevant router would violate the requirement a congestion-aware routing protocol has for fresh and timely statistics.

We have therefore developed MMP, which is a decentralized in-network monitoring protocol. The basic premise is that each router polls its own local counters. Based on these values, each router applies a function called the transfer function, to generate a representation of the polled value. The actual transfer function used by the router at this point is unspecified, it is up to the

specific routing protocol to supply it. The output of the transfer function is then packed into a data structure called the Update Message. So far we have described the general idea behind MMP, but we have not said anything about how a router interprets the information it receives. This is exactly what a *Routing Mask* (RM) does.

A *Routing Mask* allows remote routers to make sense of the values received in Update Messages from other routers. By relying on the order of the routing tables, we construct the *RM*. The actual ordering relation is irrelevant as long as the same one is used by all routers in the network. *RM*s consist of a sequence of zeros separated by ones which corresponds to the structure of the routing table as shown in Figure 5.3. A set of one or more zeroes define the field containing the output of the transfer function, which will be filled in by the transfer function when constructing the update messages. It indicates the set of links which can be used to send packets to the next destination. A one is the delimiter which indicates the transition to the next network in the routing table. Still, this does not explain how a router understands the meaning of this sequence of bits. The solution to this is simple, initially, each router computes its *RM* including the area reserved for the output of the transfer function and sends it to its neighboring router. Once a router receives the initial *RM* from its neighbor, it compares it with the *order* (not the contents) of its own routing table, therefore allowing the router to interpret which parts of the RM refer to which network and the length of the Update Messages to expect. Concretely, this means that for every one encountered in the *RM*, the router determines that the next entries (ie. the zeroes) correspond to the next network in the routing table. Basically, ones in the *RM* can be seen as an instruction which tells the router to move on to the next network entry in the routing table. Clearly, if a new network appears then all the routers must recompute their *RMs*. The interpretation of the output of the transfer function is left unspecified and is completely dependent on the specific routing protocol. It should also be noted that an Update is only send to neighboring routers when it is different to the previous one.

Routing Masks provide a simple method for representing all a routers surrounding congestion while retaining the flexibility to enable a protocol designer to develop his own representation of congestion. Moreover, *RM* provide a lightweight mechanism to exchange detailed congestion information. As we will see in Section 5.3, by only modifying the transfer function, we can obtain protocols which have very different characteristics.

5.2.1 MMP Packet Structure

MMP consists of three types of packets, one header packet and two data packets. MMP packets can either be encapsulated in Ethernet or IP packets as they define an Ethernet type and an IP protocol.

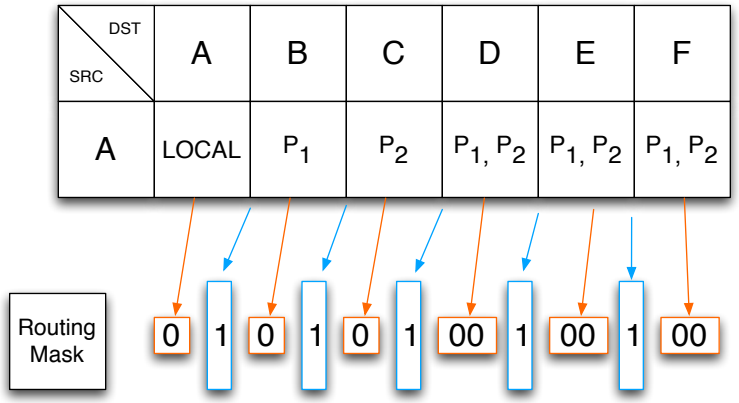


Figure 5.3: *A Routing Mask with its associated routing table*

5.2.2 MMP Header

The MMP header consists of three fields, which are packed with the most-significant bit first (ie. big-endian format).

- The 8-bit *Type* field indicates the type of MMP packet which is encapsulated by this header. It can either be STATUS or UPDATE.
- The *Code* field is used to determine the status of this packet. This field is relevant to both STATUS and UPDATE MMP types. This field is 8-bits
- The 16-bit *Checksum* field is used to store the checksum of the entire MMP packet. When a router receives an MMP packet, it computes the checksum for the packet. If the computed value is different to the checksum field, the packet is discarded.
- The 48 bit *Data* field which will contain extra information about the router or port status. It is intentionally large to accommodate for many states.

The header packet is 32 bits long and contains a data field which, strictly speaking, is not part of the header as it will contain the encapsulated packet (ie. the status or update packets).

Figure 5.4: *MMP header*

5.2.3 Status Packet

A MMP status packet is used to convey information about a router or some port on a router. The packet contains 3 fields and one data field which is used to provide a detailed explanation of the status, if one is available from the router.

- The *Chassis Address* is the mac address of the router which is concerned by this message.
- An optional 32 bit *Port Number* if the message is about a port status.
- The 16 bit *State* field is made up of 6 different possible types:
  - *Fail* - Failure of a router or port
  - *New* - New router/port announcement
  - *Update* - Status update of router/port (details are contained in the data field).
  - *Root* - Used for specific algorithms (eg. Spanning Tree, Diffusion Algorithms) to designate the Root of a computation.
  - *Init* - Indicates the beginning of a computation, all routers must clear their variables.
  - *Terminate* - Ends the protocol or the computation.

5.2.4 Update Packet

The MMP update messages are designed to send information to routers on the collected statistics or to enable a computation (eg. Diffusion Computation). It is 160 bits long and contains 4 fields.

- The *Chassis Address* is the MAC of the sending node/router of this update message (48 bits).
- If relevant, the *Port* is the originating port of this message (32 bits).

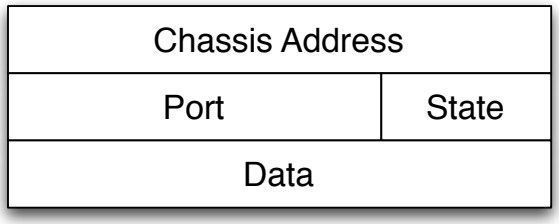


Figure 5.5: *MMP status packet*

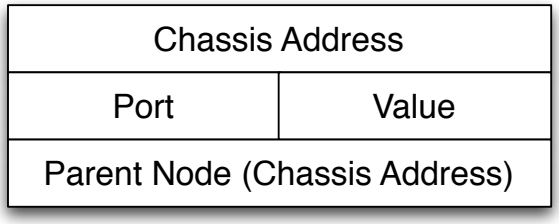


Figure 5.6: *MMP update packet*

- *Value* can signal the distance from one node to another (eg. current node vs. source node) when used in the context of a distance computation. Otherwise, when the protocol is running, initially it contains the *Routing Mask* and all subsequent packets contain the update Update Message. Currently, this field is 128 bits long, which is more than sufficient for our tests.

- *Parent Node* is used to store the MAC of the parent node of an aggregation step. Concretely, this refers to the node which actually performs the aggregation is stored in this field (48 bits).

5.3 The Protocols

In this section we will detail three routing protocols built on top of the path discovery method described above, and the MMP protocol. Each protocol has its own specific manner for describing congestion. MultiRoute describes congestion as a single bit value, whereas StepRoute provides a method to define congestion classes. Finally, PathRoute describes congestion from any source to the desired destination. It should be also noted that once a next-hop is chosen for a flow it remains so for the duration of the flow's lifetime. In other words, a routing decision is final and immutable as long as the selected link does not fail, if the link does fail another decision will be taken.

All the protocols described below have been implemented using NOX [GKP+08] and OpenFlow [MAB+08] which is described in Section 6.1.1. OpenFlow enables the programmer to control the behavior of a switch or router via a single API, and NOX gives simple access to this API using several wrapper functions. While enabling innovation, OpenFlow and NOX, also impose their overhead since ultimately the controller (which is running on a PC somewhere) decides where traffic should be directed. Hence, initially a switch/router does not know where to direct traffic and therefore it must query the controller to obtain this information. This caused an experimental delay in route setup which is caused by the experimental technique. This will be described in more detail in Section 6.1.1.

The examples given in this Section refer to the topology in Figure 5.7 and to the connectivity suggested in Table 5.1.

5.3.1 The Classical - MultiRoute

The classical implementation of MultiRoute is based on a single bit representation of congestion, which allows it to leverage local and remote congestion information. It is based on an approach similar as the one used in Random Early Marking [ALL00] (REM).

		Destination					
		A	B	C	D	E	F
Source	A	L	1	1	2	2	2
	B	1	L	1	1	1	2
	C	1	1	L	1	1	2
	D	1	1	1	L	1	1
	E	2	1	1	1	L	1
	F	2	2	2	1	1	L

Table 5.1: *The connectivity table. Each entry shows the number of possible paths.*

Protocol Detail

As stated above, MR is modeled on REM. Each router probabilistically marks a link as congested, this congestion information is then sent to neighboring routers allowing them to modify their route selection algorithm. Links are marked congested following an exponential measure of the link congestion (ie. the transfer function), doing so ensures that the marking probability is also exponential for the path congestion measure.

Achieving the behavior described above requires that each router to obtains both the available paths for routing and their associated statistics. Conveniently, the Path Discovery algorithm described in Section 5.1.1 delivers MultiRoute (MR) [ASM10] with a set of valid multipaths available for routing. Based on this set of links, MR builds a preliminary routing table indicating the set of paths available for each destination. This, then, allows MR to construct the *Routing Mask* (shown in Figure 5.3) representing the structure used by the congestion updates. The local and remote congestion information received from neighboring routers allows the current router to build the final routing table which only contains the best (in terms of congestion) next-hop to all destinations. The algorithm by which this table is built is detailed in Section 5.3.1.

The update message received from a neighboring routers is compared to the *Routing Mask* obtained initially from that same router. By aligning the *Routing Mask* and the update message, as shown in Figure 5.8, it is clear to the current router how to interpret the update message and therefore it is evident which paths for each destination are congested. Thereby, enabling a simple method to compute the final routing table.

Actually, the final routing table can be pre-computed as all the information to determine the next-hop for each destination is available before packets to these destinations arrive. Pre-calculating the final routing table is justified by the fact that in the worst case update messages will be received at one second intervals as this is the minimal polling interval router support for counter

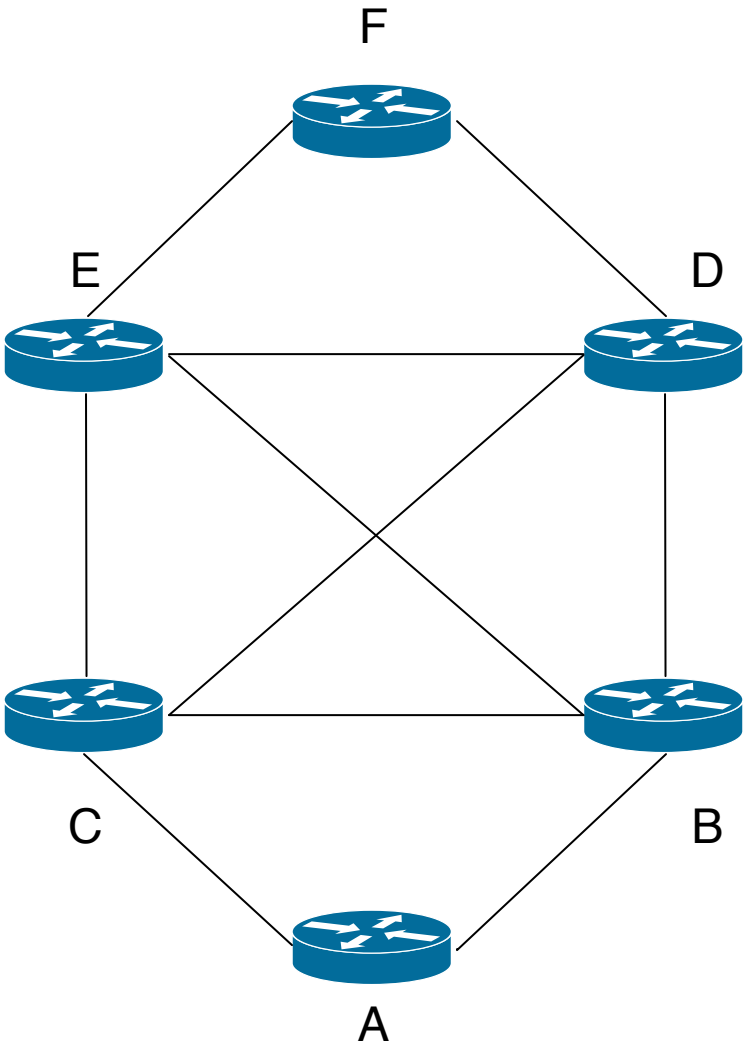


Figure 5.7: *The reference topology.*

updates [ASCSA07]. Moreover, it minimizes the flow setup time as the router now only needs to perform a lookup in the routing table to obtain the best next-hop.

MultiRoute's Monitoring Protocol delivers the MR with raw statistics from the interface counters. MR then uses the transfer function shown in Equation

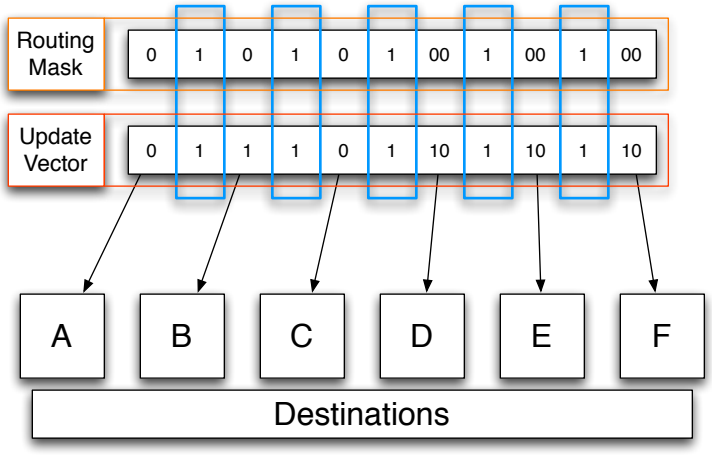


Figure 5.8: *Comparing a Routing Mask with its an Update Vector*

5.1 to compute a 1-bit probabilistic marking of the router-router links, where Δ is the difference between two consecutive counter values and Γ is the capacity of the router-router link. Finally, Φ is a scaling factor set by the network operator. Basically, the larger this factor is the more probable a link will be marked.

$$P[Link\ is\ congested] = 1 - \Phi^{-\frac{\Delta}{\Gamma}} \tag{5.1}$$

As the link utilization ($\rho = \frac{\Delta}{\Gamma}$) increases, the probability to mark the link increases exponentially, as shown in Figure 5.9, where the dotted lines show the curve for different values of Φ. The 1-bit marking is derived by random variable who value is compared to the output of the transfer function, if the random variable is smaller than the output of the function then the linked is marked as congested. The rational behind this is that we want to rapidly mark a link as congested in order to shift traffic from it as early as possible.

Another factor that is important is the *Routing Mask* length obtained when using this transfer function with the classical implementation of MultiRoute. Trivially, the *Routing Mask* length, given in bits, is shown in Equation 5.2.

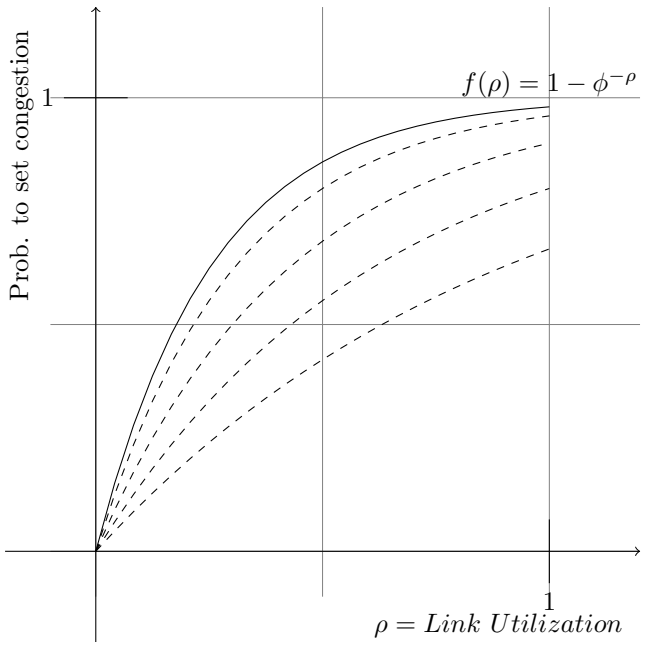


Figure 5.9: *The transfer function used in Classical MultiRoute.*

$$\begin{aligned} RM\ Length &= (N-1) + N\sum_{i=1}^{N}\frac{C_i}{N} \\ &= (N-1) + N\bar{C} \\ &= N(1+\bar{C}) - 1 \end{aligned} \tag{5.2}$$

Equation 5.2 gives the derivation of *Routing Masks* where N is the number of networks, C_i is the number of uplinks per destination. It is difficult, if not impossible, to predict the C_i values as they are linked to the topology therefore we define $\bar{C}$ as the average number of uplinks per destination which is equal to the sum of C_i divided by N. The first term, $(N-1)$, represents the number of separator bits in terms of number of networks and the second term represents the number of bits required to represent the congestion per destination. Based on this equation we have the 3D plot in Figure 5.10 of the length of the *Routing Mask* as a function of the average number of uplinks and the number of networks.

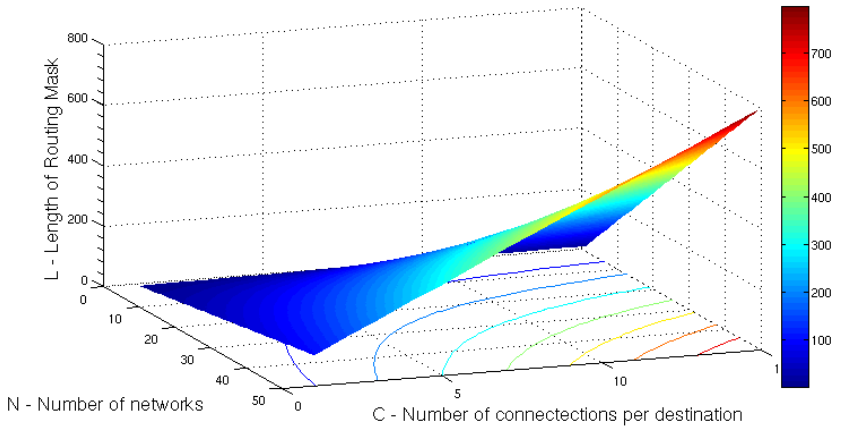


Figure 5.10: *Routing Mask Length as a function of N and C.*

Features and Limitations

MR provides an extremely lightweight mechanism to represent congestion within a multipath network. The crucial information describing which link to

which destination is congested is maintained by the Update Messages relying on the *Routing Masks*. Therefore, rather than aggregating congestion information at each router, MR provides congestion information for each router-router link and represents them clearly to all routers in the network using a single bit. Due to the use of only a single bit to represent congestion, MR is best suited for networks in which flows are long-lived and each flow consumes the entire bandwidth offered by the router-router link.

Moreover, MR uses a minimal amount of processing power at flow setup time as it pre-computes the best next-hop for each destination. The selection algorithm which builds the final routing table requires itself few resources. First, it does not create any extra data structures, thereby requiring no extra memory. Second, it is essentially a comparison algorithm and the elements of the comparison are delivered in a simple form (Update Message). Therefore the computation is straightforward as shown in Section B.4.1. Overall, the protocol maintains just one extra table more than legacy protocols (such as Shortest Path protocols). All protocols maintain a routing table which contains the next-hop to any destination, MR also maintains the preliminary routing table which contains all the possible paths to any destination obtained from the path construction component.

As MR functions by receiving signals from neighboring routers, it is clear that the main problem is the signal propagation time. Due to the congestion on the paths, signals may either be delayed significantly or in the worst case, lost. MMP, which is responsible for delivering the monitoring messages, does not verify if messages arrive at their destination (thereby behaving like UDP). As congestion increases, the probability to delay or lose packets increases significantly but as MR monitoring messages only contain an Update Message plus an MMP packet header they are very small (typically less than 200 bits as shown in Figure 5.10). Therefore, the risk of delaying or losing packets is mitigated by the size of the packets themselves.

Another limitation is given by the *Routing Mask* length itself. It is known that the Maximum Transmission Unit (MTU) for an IPv4 path is 576 bytes, and the MMP protocol does not support fragmentation[3]. By setting *RM Length* to $576 * 8$ bits in Equation 5.2, we can see that there is an inverse relationship between the number of networks and the average number of uplinks per destination as shown in Figure 5.11. Considering that MR is an interior-gateway routing protocol, an unreasonably large number of networks and/or parallel uplinks would be required to exceed the MTU limit.

[3]Moreover, fragmentation is nowadays considered harmful [KM95]

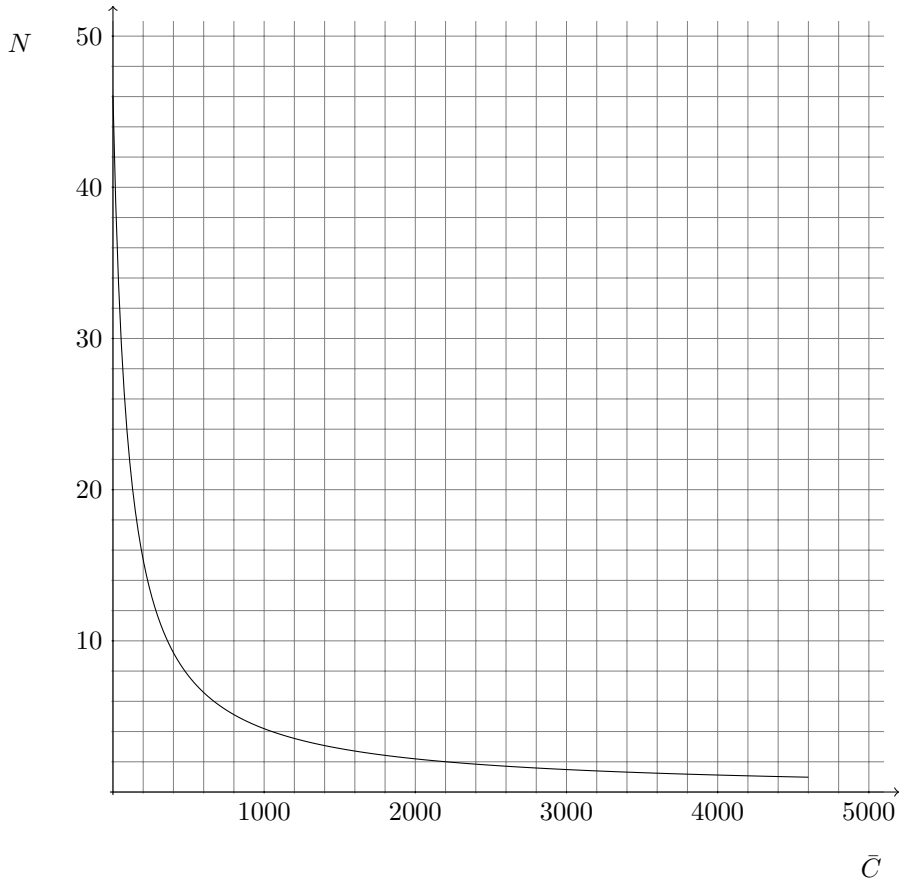


Figure 5.11: *Plot of the Routing Mask Limitation*

Routing Algorithm

The Routing Algorithm in MR, given all the congestion indications, constructs the best (or least congested) next-hop based on the set of available paths obtained from the preliminary table. The pseudocode is given in Section B.4.1. The Routing Algorithm considers both local and remote congestion information and we distinguish three cases in which the algorithm operates:

- *Case I - All local paths uncongested.* In this case, we consider the remote congestion statistics relevant to the destination under consideration. If there are multiple local paths available, for each potential next hop the number of uncongested paths are counted. The local path associated to the next hop with the least congested paths is selected, thereby maximizing the probability that the traffic will be forwarded unhindered. On the other hand, if all the remote congestion counts are equal or if there are multiple candidates, the shortest path is selected.

- *Case II - Some local paths are congested while others are not.* Here we first consider the uncongested local paths in ascending path lengths. Then the associated remote paths are inspected, if one is found that is not congested then it is selected as a next hop. Otherwise, the shortest path is selected. This case will attempt to fill available and uncongested paths, thereby increasing the overall throughput of the network.

- *Case III - All local paths are congested.* In this last case, we perform a search which is similar to the first case. The idea being that we want to find a path that is well suited to accommodating more traffic. If none is found, we select the shortest path again.

It is clear from the outline of the algorithm that in the worst case the algorithm will always select the shortest path. Moreover, the algorithm favors the shortest path first, and only once this path is considered congested does the algorithm switch to an alternative path. This approach guarantees that the algorithm cannot perform worse in terms of throughput than the current shortest path algorithms.

Consider now the network shown in Figure 5.7 and the associated routing table given in Table 5.1, and multiple flows originating from the network under router F with destination network at router A. For the purpose of this example, let us assume that the flow's inter-arrival time at the source router is greater than the update interval at neighboring routers[4]. We also assume that each flow is long lived and immediately consumes the entire bandwidth of the path it utilizes. There are four paths between networks F and A, namely F-D-B-A (1),

[4]In practical terms, these routers poll their local statistics at one second intervals, therefore the update time is one second plus the update message transmission time.

F-D-C-A (2), F-E-C-A (3), and F-E-B-A (4), path (1) is considered to be the shortest followed by path (2) and so on. As there are only two distinct paths we can expect to double the network throughput with respect to a shortest path algorithm.

When the first flow arrives, it is bound to path (1) as this path is considered to be the shortest, and due to Case I, this then causes an update from the routers F, D, and B indicating that their links are congested. When the next flow arrives, the decision is taken by Case II of the algorithm, and therefore the algorithm will consider the paths with next hop E as this link is not congested. The path (4) is excluded, because the link between B and A is congested due to the first flow. Therefore the second flow is bound to path (3). Finally, for each subsequent flow, Case III of the algorithm will apply. The next two flows will be bound to paths (2) and (4) respectively, and the rest will be sent along the shortest path.

5.3.2 The Organizer - StepRoute

StepRoute [ASM11] is a variant of MultiRoute which enables the network administrator to define congestion classes. These classes allow for a finer control of the congestion of the network and therefore enable for a more efficient use of the network resources.

Protocol Detail

StepRoute contrasts with MultiRoute by first extending the *Routing Masks* to accommodate the definition of congestion classes. Second, it uses a linear transfer function rather than an exponential one in order to simplify the definition of the congestion classes. Based on a predefined value, a link is deemed to be part of one of the congestion classes.

By classifying congestion into classes and using *Routing Masks*, we are able to deliver timely information to routers while maximizing the accuracy of the delivered statistics. Each router has to define the same number of congestion classes, otherwise neighboring routers will not be able to interpret the *Routing Masks* received correctly.

$$\gamma = \frac{\Delta}{\Gamma} \tag{5.3}$$

In a similar fashion to MR, StepRoute obtains its raw statistics from MMP. StepRoute then classifies the link into the corresponding congestion class based on the transfer function, shown in Equation 5.3, and the congestion classes defined in Table 5.2 where the α values are the congestion class boundary value. As the link utilization increases the link is classified into a higher congestion

classes, thereby simplifying the process by which the Routing Algorithm will select the next-hop, ie. favor links in lower congestion classes.

Class 1	$0.0 < \gamma \leq \alpha_1$
Class 2	$\alpha_1 < \gamma \leq \alpha_2$
Class i	$\alpha_{i-1} < \gamma \leq \alpha_i$
Class P	$\alpha_{P-1} < \gamma \leq \alpha_P = 1.0$

Table 5.2: *Congestion Classification.*

Figure 5.12 depicts the linear transfer function used in StepRoute. In this variant of MultiRoute, it is preferable to use a linear transfer function as we wish to distribute traffic equally amongst all the links we have at our disposal. If an exponential transfer function had been used then links would quickly fall into the highest congestion class and links with unused bandwidth would not be used.

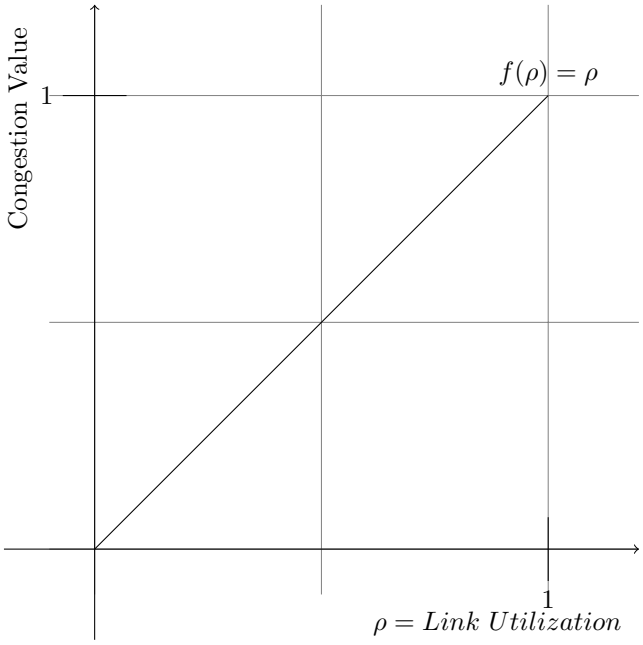


Figure 5.12: *The transfer function used in StepRoute.*

Using such a classification mechanism we can simply encode the congestion class number in the Update Message. Therefore if we would like to rep-

resent m classes of congestion where m can be expressed as 2^n, then we only need n bits per router-router link are needed to represent all the congestion classes. Therefore, as the number of congestion classes grows exponentially, the space required to represent them, in the Update Message, grows linearly. This method allows us to describe many congestion classes while employing a lightweight, and therefore easily distributable, representation.

On the other hand, using multiple bits to represent the congestion class will require longer *Routing Masks* as each entry for each destination in the *Routing Mask* is now n bits long. We therefore have Equation 5.6 which represents the length of the *Routing Masks* in StepRoute.

$$\begin{aligned} L &= (N-1) + N\sum_{i=1}^{N}\frac{nC_i}{N} && (5.4)\\ &= (N-1) + Nn\bar{C} && (5.5)\\ &= N(1+n\bar{C}) - 1 && (5.6) \end{aligned}$$

Equation 5.6 is identical to Equation 5.2 if we set n, the number of bits required to represent the congestion classes, to one. Therefore, all the other terms in Equation 5.6 represent the same thing as for MR.

Features and Limitations

As StepRoute is built on top of MultiRoute it inherits all the functionality that MR had, but also it also suffers from the limitations of MultiRoute. Namely, it suffers from the same problem of signal propagation time and the potential *Routing Mask* length. Due to the fact, that StepRoute uses more bits in the *Routing Masks* than MultiRoute, it supports less networks or less connections per destinations. In other words, the inverse relationship between number of networks and number of connections in StepRoute is steeper than in Multi-Route.

StepRoute uses slightly more memory than MultiRoute as it requires three tables. The first two are also found in MultiRoute, the third table is used to convert the raw congestion value into a congestion class. The size of this table depends entirely on the number of congestion classes and therefore it is safe to say that it contains $n-1$ entries.

StepRoute allows for a finer control of the congestion on the network. By defining multiple congestion classes, StepRoute can represent the current level of congestion to neighboring routers and therefore attempt to install flows onto paths which have enough bandwidth to support the new flow. This approach leads to much better usage of the network resources especially in the presence of flows which do not consume the entire bandwidth of router-router links.

Typically, if the the router uplinks have a much larger capacity than the underlying network, StepRoute will load balance this traffic much more efficiently than MultiRoute, as flows will always consume a fraction of the capacity of the uplink and therefore the congestion classes will describe more precisely the partial congestion present on each uplink.

Routing Algorithm

Similarly to MultiRoute, StepRoute distinguishes three cases which represent the state of the congestion local to the router. Then, based on these three cases it selects the current best next-hop from the preliminary routing table and constructs the final routing table. The pseudocode is given in Section B.4.2. We should note here that the algorithm considers the lowest congestion class as already being congested and therefore attempts to avoid the potentially lightly loaded link.

- *Case I - All local paths uncongested.* In this case, the algorithm only looks at the statistics received from neighboring routers. Assuming there are multiple paths available, the router searches for the least congested path which is simple due to the classification of the congestion values discussed in Section 5.3.2. Clearly, if the algorithm finds a remote path with is not at all congested, it immediately selects this path for forwarding. On the other hand, if all the remote congestion counts are equal or if none is found, the shortest path is selected.

- *Case II - Some local paths are congested while others are not.* This case is slightly more complex because a local path, even if it is carrying some traffic, may still be amongst one of the better options. This is due to the fact that remote paths, which lay beyond a completely uncongested link, may be completely congested. In this case, the algorithm ranks the candidate paths by summing their local congestion with the remote congestion. The path with the lowest congestion value is then selected. As with Case I, if there are multiple candidates, the shortest one is selected.

- *Case III - All local paths are completely congested.* This case is very much similar to the first case. The idea here is to look at the congestion values of remote routers and determine the least congested path, in an effort to use up all the available bandwidth. Again, if multiple candidates are found, the algorithm defaults to the shortest path.

Let us consider, as an example, the network given in Figure 5.7 and its associated Table 5.1, when multiple flows enter at router F destined for network A. We also assume that each flow is long lived and that it immediately consumes

half of the available bandwidth. There are four paths between networks F and A, namely F-D-B-A (1), F-D-C-A (2), F-E-C-A (3), and F-E-B-A (4), path (1) is considered to be the shortest followed by path (2) and so on. As there are only two distinct paths we can expect to double the network throughput with expect to a shortest path algorithm. It is important to note that, with respect to the real implementation the routing tables are pre-computed as statistics become available and not when a flow arrives.

When the first flow arrives, it is bound to path (1) as this path is considered to be the shortest, and due to Case I, this then causes an update from the routers F, D, and B indicating that their links are partially congested. When the next flow arrives, the decision is taken by Case II of the algorithm, and therefore the algorithm will consider the paths with next hop E as this link is not congested. The path (4) is excluded, because the link between B and A is congested due to the first flow. Therefore the second flow is bound to path (3). Upon arrival of the third flow, Case II will rank the available paths according to the congestion level and will choose path (2) as the link between D and C is not congested. Similarly, when the fourth flow arrives, Case II ranks the available paths again and picks path (4). As subsequent flows arrive at router F, Case III attempts to find available bandwidth to send the flow on and if this is not possible it sends it onto the shortest path.

Similarly to MultiRoute, the algorithm defaults to the shortest path as well. This default is to reduce the delay experienced by the packets in the situation where no best option exists.

5.3.3 The Know-it-all - PathRoute

While MultiRoute provides a link by link measure of the path congestion, PathRoute provides the actual congestion over the entire path from a given source to any destination. MultiRoute could in certain circumstances lead traffic into a congested area of the network due to a lack of knowledge of the effective distant congestion. PathRoute, by reporting entire path congestion at every router, avoids this situation and will never lead traffic into a congested area to the extent of the possible of course.

While the previous two protocols limited themselves to distributing update vectors to their direct neighbors only, PathRoute concatenates the information contained in update vectors to deliver the path congestion for each router. To achieve this objective PathRoute uses the same transfer function as MultiRoute and also represents congestion on a single bit. The difference resides in the fact that now the protocol analyzes the information received from its neighbors to derive the path congestion. In order to distribute path congestion, PathRoute needs to know two pieces of information:

1. The set distances from a given router to all the other routers in the

network. This will allow PathRoute to define the length of the *Routing Masks* and therefore construct the initial mask which will be used to inform neighboring routers about the structure of the update messages.

2. The set of congestion information that should be concatenated at each router and sent to neighboring ones. A router needs to determine which part of an update message is relevant to it, then concatenate this information and send it to neighboring routers.

The first problem is solved using Diffusion Algorithms with termination detection [DS80]. Each router initiates a computation by outputting an Update MMP of code INIT on each of its router-router links for each destination and keeps track of the number of messages it has sent and received. Next, when a router receives an Update MMP with code INIT, it checks whether the packet is destined for it, if it is then the router increments the distance counter (contained in the packet), changes the code of the packet to NEW and replies to the source over the shortest path. Otherwise, the counter is incremented and the packet is forwarded to the destination, if the current router has multiple paths to the destination it duplicates the packet and sends it down each of the uplinks for that destination. Then, if a router encounters an MMP with NEW code, it checks whether its number of sent and received messages is equal, if yes it sends the reply towards the source along the shortest path otherwise it simply stores it. Doing so, guarantees that if router sends a reply it is either the destination of the query or it has received as many queries as it has sent and therefore by the time the source obtains a reply we know that the computation has terminated. Finally, when the source receives a reply it updates its distance table. The flowchart in Figure 5.13. Once all the routers have computed their relative distances, each router can now construct and send their *Routing Mask* to their neighbors. This process occurs at the initial state of the network and therefore we assume that no other traffic can interfere with this process.

The second issue can be divided into two sub-problems. First, knowing that each entry for a destination in the Update Message contains the entire path congestion to that destination, the protocol must first know where to insert its local congestion information. Second, when receiving an update message the router must identify the parts of the update message which are of interest to it, ie. the parts which describe the congestion to one or more of its destinations.

The first sub-problem can be solved quite easily. Knowing the distance and the number of uplinks for any destination, given by the distance table and the preliminary routing table respectively, the router can easily identify the first bit of a path description which will be the congestion value for its local uplink. The following sub-problem relies on the structure of the *Routing Mask*. When a router receives an update message, it first determines if the port on which

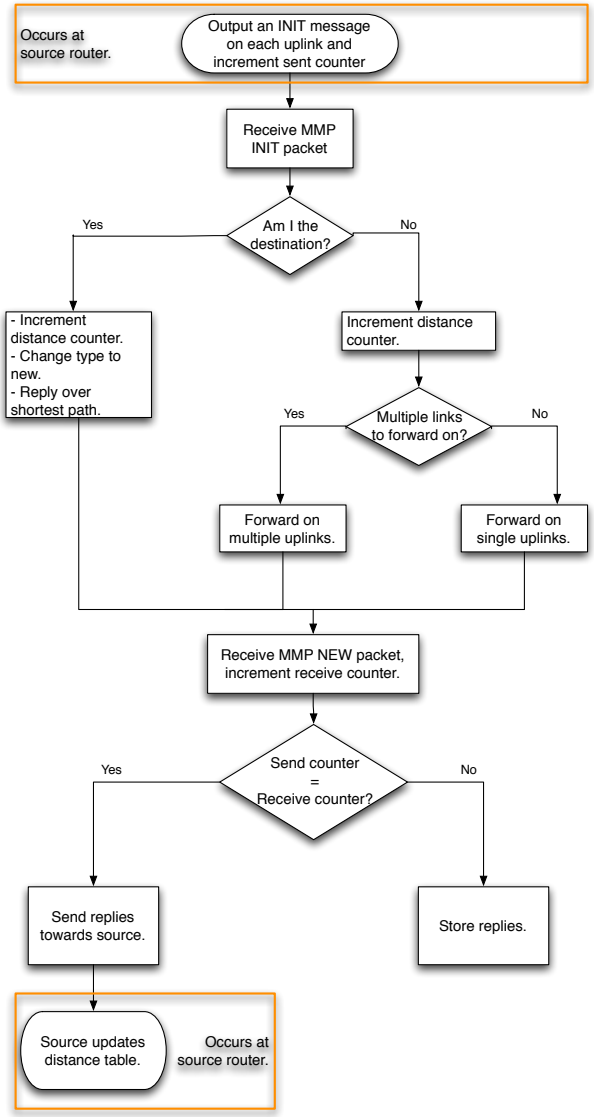


Figure 5.13: *Flowgraph of the distance diffusion computation.*

the message is received is used for any destination and if so, is the distance to the destination on this port greater than one? Then by comparing the message with the *Routing Mask*, just like in MultiRoute or StepRoute, it determines which parts of the update message contain information about one or more of its destinations. The router can simply copy the interesting parts of the update message into its own update message, by concatenating the local congestion on the uplink with information received in the update message from the neighbor. This new update message is then propagated to the neighbors which therefore distributes the congestion information around the network and enables routers to build their view of the path congestion. This process is referred to as the aggregation step of PathRoute.

The major side-effect of representing the entire path congestion within the update messages is that their length increases significantly. While this is intuitively trivial, it is not so clear analytically since it is impossible to formalize the path length as it is intimately coupled to the network topology. Therefore we can only predict the message size by specifying an average path length, $\bar{P}$, as shown in Equation 5.9.

$$
\begin{aligned}
L &= (N-1) + N \sum_{i=1}^{N} P \frac{C_i}{N} && (5.7)\\
&= (N-1) + N\bar{P}\bar{C} && (5.8)\\
&= N(1+\bar{P}\bar{C}) - 1 && (5.9)
\end{aligned}
$$

Features and Limitations

PathRoute delivers an extremely space and computation efficient mechanism for representing the path congestion at each router. By computing the distance from any router to another, PathRoute is able to extend its *Routing Mask* with space to describe every link along the path to a destination and therefore deliver full path congestion information to every router. Finally, since each router running PathRoute knows all the information about the status of the path to the destination, the resulting routing algorithm is very simple and only requires information which is locally stored which therefore enables rapid flow installation.

By design, MultiRoute may unknowingly route traffic into congestion since it only has congestion information about its local links and its neighbors links. Therefore, if congestion lies beyond the next-hop andthe next-hop has no other option then MultiRoute may exacerbate the congestion in some areas of the network. PathRoute, on the other hand having full knowledge of path congestion, will never route traffic into congestion unless there is no other option available to it.

Even though PathRoute provides deep information about the status of the network to each router, it requires no more memory than StepRoute. Three tables are required, a preliminary routing table, a final routing table, and a distance table. On the other hand, initially, PathRoute requires slightly more CPU power as it must send MMP packets to compute the relative distances from one router to another.

As PathRoute is built on top of MultiRoute, it suffers from the same problems. Moreover, since update messages may be aggregated, the propagation delay for an update message is potentially multiplied by the path length. Therefore, the probability of delay or drop of the update message is also increased, but again this is mitigated by the relatively small size of these messages.

Routing Algorithm

As each router now has all the information about the path congestion, there is no longer a need to distinguish three cases for the routing algorithm. PathRoute's routing algorithm takes the simplest form. More precisely, it simply finds the least congested path and forwards packets through the associated next-hop. As shown in B.4.3, the routing algorithm is straightforward and simply counts the number of links which are not congested. The path which contains the least number of congested links is selected.

Consider, once again, the network given in 5.7 and its associated connectivity Table 5.1, when flows originate at multiple routers in the network. Let us also assume that each flow directly consumes the entire bandwidth proposed by the path. We assume that the source routers are F, D, and E and a single destination A. Therefore, the available paths are F-D-B-A (1), F-D-C-A (2), F-E-C-A (3), F-E-B-A (4), D-B-A (5), D-C-A (6), E-C-A (7), and E-B-A (8).

Considering the situation described above, two flows arrive at router F. Based on PathRoute the first flow will select path (1) and the second path (2). Now, a flow arrives at router D, since path (5) has now two congested links (D-B and B-A) caused by the first flow. Router D selects path (6) which only has one congested link. The same occurs for the flow that originates at Router E and path (8) is selected. We can see that PathRoute will attempt to use all the links of the networks before sharing full paths.

5.3.4 Implementation Experience

The entire MultiRoute family was designed and implemented within NOX (see Section 6.1.1). NOX is a controller API for OpenFlow whose main advantage is the exposition of a simple API which enables the rapid prototyping of research networking protocols. NOX introduces the concept of components which can be viewed as black boxes computing a certain function, for example, compute

the set of shortest paths within a network. NOX components communicate via events and direct message passing.

The design of the MultiRoute family is based on components. Each function described the previous section is implemented with a component. The component itself may be further segmented into various files. The code which makes up the MultiRoute suite is highly modular. Each protocol depends on different components. Changing these components modifies the behavior of the protocol. For example, switching the transfer function in MultiRoute for the one which computes congestion classes yields the StepRoute protocol. Figure 5.14 show the dependency maps for each protocol, notice that each protocol differs only in some of the components used. Since similar components expose identical API or produce the same events, most of the code from one protocol to another is re-used. This design accelerates the implementation cycle and guarantees that the code is stable from protocol to the next. Moreover, the implementation was almost exclusively done using Python [Pyt] which means that the implementation itself was simple and is easily understandable.

5.4 Commodity Protocols

The protocols detailed below were implemented to enable us to test MultiRoute-based protocols against the protocols which are used today in industry. Each of these protocols were implemented on our OpenFlow-enabled testbed.

5.4.1 Shortest Path

Shortest Path Routing is a direct application of Dijkstra's Shortest path algorithm [Dij59]. In essence, it is a destination-based, load-insensitive routing algorithm which is built based either on the minimum hop count or sum of link weights. Therefore, given a destination the routing algorithm performs a lookup in its routing table, which contains the next-hops to every destination, and returns the next-hop of the shortest path to the requested destination.

This protocol is made to resemble OSPF [Moy98] as much as possible without having to re-implement the totality of OSPF. This is justified, as we are only interested OSPF's routing model but not its distributed nature nor all its extra functionality.

5.4.2 Equal Cost MultiPath

Equal Cost MultiPath (ECMP) [TH00] is similar to Shortest Path routing in that it is also destination-based but it differs in the fact that multiple best paths may be used to forward packets. Therefore, as it name indicates, if the

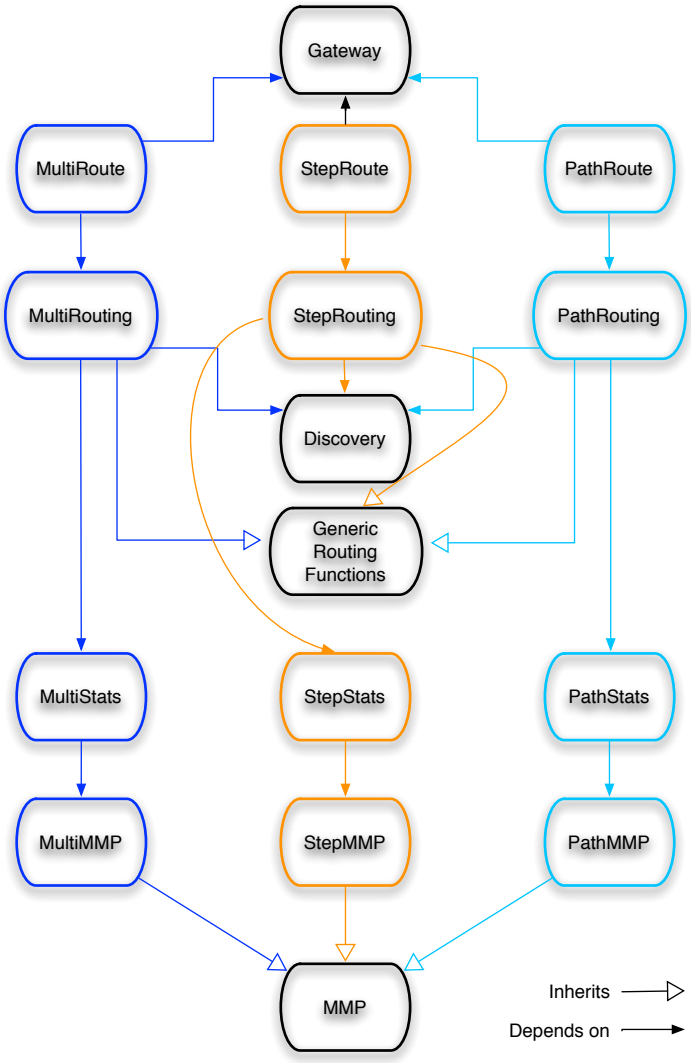


Figure 5.14: *Inheritance/Dependency graph for MultiRoute-based protocols*

network contains multiple shortest paths between any source-destination pair then they are all used to forward packets.

Normally, ECMP uses a hash which is computed based on the source and destination mac-address contained in the packets. In our case, out network testers and machines connected to our testbed only have a limited number of mac addresses. Therefore, to increase the diversity of the hash, we have computed based on the source and destination port numbers which are random for every flow.

Section A.1 derives and explains the number of disruptions to expect when a link of an ECMP set goes offline.

5.5 Theoretical Delay Model for MultiRoute-based protocols

In this section, we will use the model detailed in Section 4.3.1 to derive a delay model for Congestion-Aware Routing protocols. Furthermore, we also evaluate the average number of flows present at any node in the system.

A multipath network can be seen as a system with multiple servers. Each server, therefore would represent an uplink between nodes in the system. Routers are then referred to as nodes. We now need to define the arrival (input) and departure (output) models. We can safely assume that the arrival model is Markovian since packets sent are sent from end nodes (ie. computers) are independent and memoryless, the same goes for the output model since it only depends on the actual packet length of the router which is constant. Therefore we can model our first queue in the System as M/M/m queue (see 4.3.1). But, as explained in Section 4.3.1, we cannot simply say the same thing for any other queue in the system as the message length and the inter-arrival times become highly dependant. Fortunately, Kleinrock's Independence assumption 2 and in particular Burke's Theorem, allow us to select a new message size at the output of each queue independently and therefore model our entire system as independent M/M/m queues.

Therefore, we can now say that the delay created by a certain node is the average time spent in the system and for an M/M/m queue is given by:

$$T = \frac{1}{\mu}\left(1 + \frac{P_Q}{m(1-\rho)}\right) \tag{5.10}$$

Where P_Q is given by Equation 4.8 and $\frac{1}{\mu}$ is the service time and $\rho = \frac{\lambda}{m\mu}$ is the utilization ratio with λ the arrival rate and m the number of servers.

Since, we claim that each queue is independent we can state that the overall delay in the network is simply a function of the number of hops a packet must traverse and therefore we have:

$$\bar{T} = \bar{n}T \tag{5.11}$$

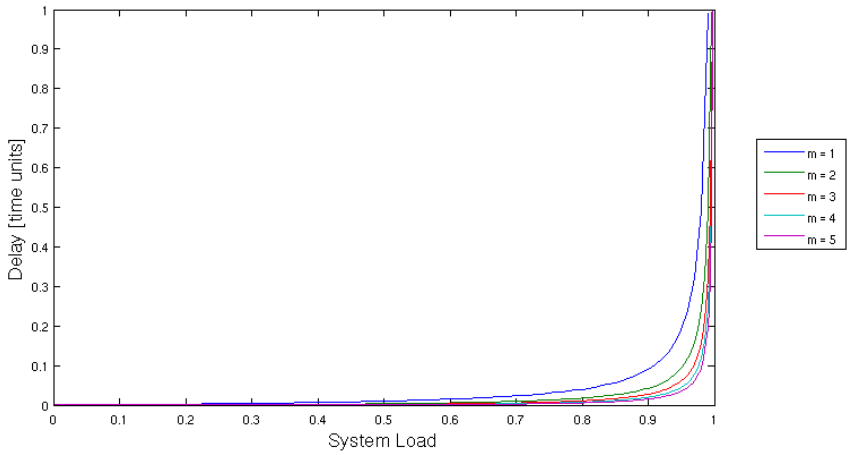


Figure 5.15: *Flow delay as function of the load* ($= \frac{\lambda}{m\mu}$) *and number of servers.*

Figure 5.15 shows that as the arrival rate increases the overall load on the system increases and eventually leads to ∞ which is to be expected because the expression for delay is dominated by $1 - \rho$

Another interesting quantity is the number of clients (ie. flows) that are present at any queue. This quantity will allow us to determine the expected disruption should a queue become unavailable for some reason.

$$\bar{N} = m\rho + \frac{\rho P_Q}{1 - \rho} \tag{5.12}$$

Since Equation 5.12 is also dominated by $1 - \rho$, we would have the same behavior than in Figure 5.15. Namely, as the system load increases the number of flows in queue will increase as well. Figure 5.16 shows the number of flows waiting at each queue. We can see that as the arrival rate increases, the number of flows in the queue increases significantly. This value can also be seen as the number of flows that would be disrupted if an uplink where to go offline, thus giving us an approximation on the number of communications that would need to be re-established should there be a problem. We should note that if we pick $m = 1$, then we have a model for a single path system. We can see that in such a system the delay increases significantly than in multipath systems.

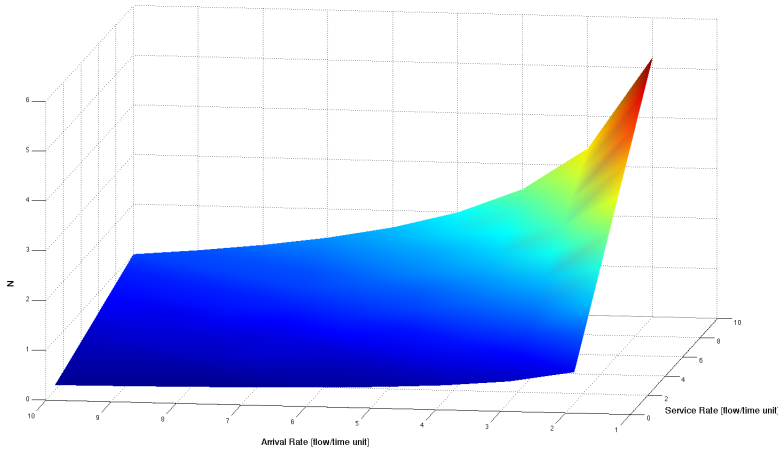


Figure 5.16: *Number of flows in each queue*

5.6 Summary

In this section, we have described three protocols by using the outline given in RFC 1264 [Hin91] then we gave several theoretical predictions which, while not fully accurate, provide us with an idea of the performance we can expect of our protocols.

The first protocol, MultiRoute, describes congestion as a single bit. While the second, StepRoute, defines congestion classes. Finally the third protocol, provides a congestion measure for the entire path from source to destination.

Using multipaths does not change the rules of the game in networks, by this we mean that the latency still increases as the load on the network increases and queues still get more and more full as the load increases. That said, all these phenomena happen when the load is higher than in the single path settings. More importantly, we also see that even though each of our protocols routes flow rather than individual packets, we can still expect a significant increase in performance while not optimal it is still quite a desirable increase over single path routing.

Chapter 6

Materials & Methods

"If you are afraid of change, leave it here."
– Seen on a tip box, Mountainview, CA

6.1 OpenFlow & NOX

OpenFlow is an open standard which enables a network engineer to control the behavior of their networking equipment programmatically. It represents a radical shift for the networking status quo. First, manufacturers need to embrace OpenFlow and implement it[1]. Second, it signals the end of distributed network algorithms and a return to simpler centralized algorithms.

OpenFlow and NOX are the basis for the implementation of the work in this Thesis. While the author believes that the future of computer networks lie alongside OpenFlow, he has implemented his work in a distributed manner in order to resemble current day network protocols.

6.1.1 OpenFlow

OpenFlow [MAB+08] offers a way for network researchers to try their experimental protocols on real hardware and on the networks they use every day. Until now there has been an extremely high barrier to entry for new networking ideas due to the face that there is a rational reluctance within the community to experiment with production traffic. The installed base of networking hardware has been largely built to conform with the extant RFC's and as such offers few, or no external interfaces. The result of this has been to exclude innovation since there is no mechanism to deploy or test alternative strategies to the ones

[1] So far, HP, NEC and Broadcom are known to have hardware implementations.

laid down in silicon. Up until quite recently the only way that experimental progress could be made was work with specific experimental installation such as GENI [GEN]. GENI is an nation-wide (in the United States) network which provides programmable network elements via virtualization and is capable of processing packets for multiple isolated experiments. The disadvantage with such installations is that they are very costly and take years to implement. Another example of such systems, while smaller, are Emulab [Emu] and StarBed [Sta].

There also exists several software solutions. Many OSs can route packets between their interfaces and software implementation of routing protocols exist, for example from XORP [HHK03] (eXtensible Open source Routing Platform). Moreover, the CLICK [MKJK99] router project allows researchers to control and manage the packet processing. While these solutions are interesting, they do not provide the performance or the port density required to test news ideas efficiently. Hardware implementations are also available, the largest one is the Advanced Telecommunications Computing Architecture Supercharged PlanetLab platform [TCD+07] but it is targeted for large deployments and it is extremely costly. There is also a NetFPGA [NETb] solution, but this only provides four ports per card.

The question is why can we not find a way to use the hardware we currently have and the networks we have deployed. This is exactly what OpenFlow is designed to do.

Figure 6.1: *The OpenFlow flow signature.*

OpenFlow exploits the flow-tables contained in modern switches. Flow-tables are built from TCAMs[2], which allow functions like QoS, NAT, firewalls to run at line rate. OpenFlow stores the signature of a flow, shown in Figure 6.1 into the flow-table, and associates one or more actions to be applied to

[2]A TCAM is a Ternary Content Addressable Memory. This allows the operating system to match a third state, "X." The X state is a "mask," meaning its value can be anything. This lends itself well to networking, since netmasks operate this way. Routers can store their entire routing table in these TCAMs, allowing for very quick lookups.

the flow. The signature then allows the switch to match subsequent packets onto that flow. The overall effect of the actions determines the behavior of the network. Figure 6.2 shows an OpenFlow-enabled switch which is made up of a secure channel and a flow-table. The secure channel connects the controller to the switch enables the controller to send commands to the switch and manipulate its flow-table.

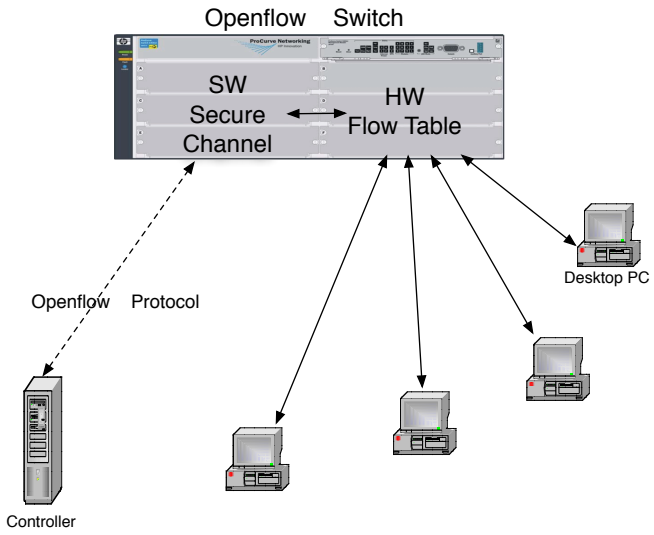


Figure 6.2: *The OpenFlow switch.*

OpenFlow then provides an open and programmable interface to control the contents of the flow-table. The set of actions which can be applied depend on whether the switch implements all the actions or only the required ones. Moreover, some manufacturers may not be able to implement certain actions due to their underlying hardware as we will see in Section 6.2.1. The set of actions supported by an Openflow switch are summarized below:

- **Set VLAN ID** - Specify or overwrite the VLAN identifier.
- **Set VLAN priority** - Define the priority of the packet with the VLAN.
- **Strip VLAN header** - Remove all VLAN information.
- **Modify source or destination MAC address** - Overwrite the Ethernet header information.

- **Modify source or destination IP address** - Overwrite the IP header information.
- **Modify ToS bits** - Modify the Type of Service bits.
- **Modify Transport source or destination ports** - Overwrite the TCP header information.

OpenFlow decouples the control from the datapath and exports it to a controller, such as NOX, which will control the contents of the flow table. As an OpenFlow device is usually referred to as a switch even though technically it is not a stereotypical switch. OpenFlow can implement the function of a switch, router or even both; but it may also implement any other function. The term switch here is used to refer to the hardware on which OpenFlow is deployed.

6.1.2 NOX

NOX [GKP+08] is an OpenFlow controller whose goal is to provide a programmatic interface so that other applications can manage the underlying network. NOX presents applications with a centralized programming model. NOX, programs are written as if the network were contained on a single machine. NOX abstracts low-level network details away from the developer. It essentially provides a framework to manage OpenFlow-enabled switches and routers by handling the low-level OpenFlow API provided by the enabled hardware.

A NOX based network is made up of openflow-enabled switches and one or more instances of NOX where each instance provides the exact same network view to the developer(s). Network applications are written on top of NOX and control the behavior of the network by manipulating the flow-table. During normal operation, NOX receives flow initiations (first packet of an unmatched flow), these are passed by NOX to the network application which decides what to do with such a flow. In the end, a new flow rule may be added to the flow table and therefore subsequent matching flow will not be sent to NOX but rather the switch will apply the action specified in the added flow rule.

NOX's programmaable interface is driven by events, the namespace and the network view. Traffic patterns in modern networks are constantly shifting therefore flows come and go, links go up and down. NOX handles this by allowing developers to register for events. When an event occurs NOX calls the handler passed at the registration.

NOX builds a network view based on messages obtained from the Openflow-enabled switches as they connect. Applications use the network view to perform computations or decide how to handle a certain type of packet, etc. Each

NOX instance constructs a list of loaded applications and from this point onwards each loaded application can exchange messages with each other. This is known as the NOX namespace.

6.2 TestBed Installation

6.2.1 Routing Hardware

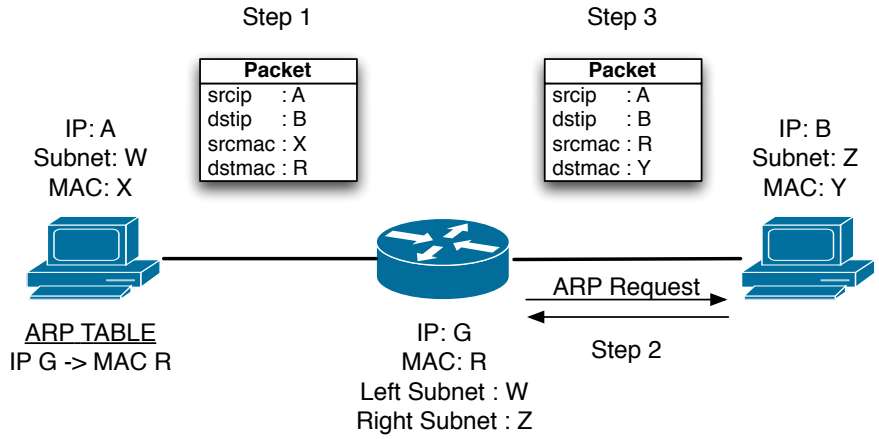
The testbed is implemented on HP Procurve switches [procurve] running an experimental OpenFlow-enabled firmware. OpenFlow on the HP platform is instantiated per VLAN[3]. We have exploited this by defining different VLANs on each network device which are then used to identify the network address range controlled by the device. The VLAN ID is then advertised to the controller who uses it to assign a virtual gateway address for each router in the network.

The HP implementation of OpenFlow has several limitations which are mostly due to the underlying hardware. For example, HPs cannot perform mac address rewriting in hardware because their routing support is monolithic and the re-writing functionality cannot be isolated from the other routing functionality. It is actually more limiting, HPs can only re-write mac-addresses to a single mac address (the base address of the device) and therefore specifying a custom mac address is impossible. Since the hardware does not support some OpenFlow actions, they are implemented by the OpenFlow firmware in software. As the processor within switches is a shared resource between all the ports and is not very powerful in the first place, the software path is a very slow path. Our measurements show that the software path provides roughly 1.25 MB/s overall bandwidth per switch, this is equivalent to 10Mb/s cross sectional bandwidth.

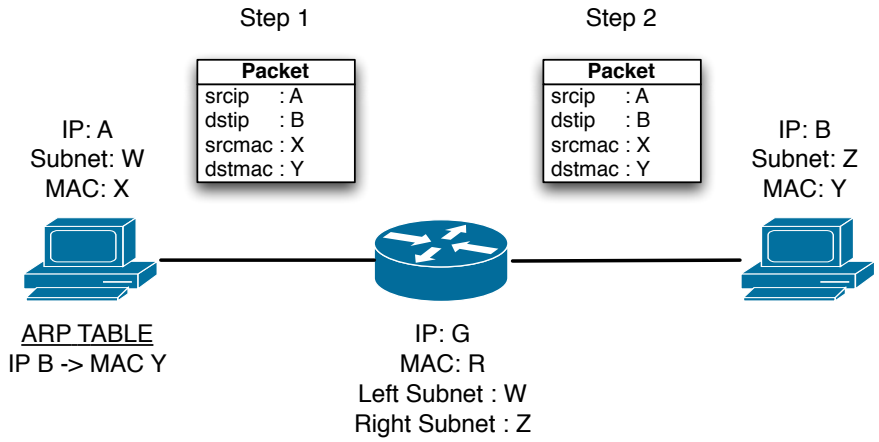
Since we are designing a new routing protocol which is supposed to surpass all existing ones in terms of performance, using our OpenFlow devices in software path (ie. invoking the switch CPU) is counter productive. Therefore we have employed several workarounds in order to build a routing protocol using HP's OpenFlow implementation.

Our goal is to develop a protocol which achieves its routing function without replacing the Link Layer information, which is mainly how a router behaves [RFC95]. The Link Layer information is replaced by a router in order to be able to direct a packet to its next-hop whilst maintaining the source and destination IP fields intact. This is easily solved in an OpenFlow implementation as we can direct packets out of any interface at any moment. The more crucial problem arises when the packet is arriving at its destination. Because we will not be able to re-write the MAC address, which is set to the MAC address of the

[3]Virtual Local Area Network



(a) *Standard routing model.*



(b) *HP Openflow Routing.*

Figure 6.3: *An explanation of routing processes.*

last-hop router, the destination host will reject the packet. The workaround here is rather cumbersome, we prepopulate each host ARP table with the IP to MAC address mappings, and therefore when a host sends out a packet to any other host it can already set the destination MAC address to its final value. These two workarounds allow us to design a routing protocol albeit slightly simplified but the main routing function, of routing packets through a network via the most efficient path, is still guaranteed. Figure 6.3a depicts the normal behavior of a router and Figure 6.3b shown the behavior of routing in our OpenFlow testbed. In Figure 6.3a, a host with IP address A sending traffic to another host with IP B would first notice that due to its subnet mask that the destination IP is not on the same local area network (LAN). Therefore, it builds a packet with source and destination IP A and B respectively and source and destination MAC address X and R respectively (Step 1). When the packet arrives at the router, the router performs an ARP request to determine the MAC address associated to IP B (Step 2). Once the MAC address is resolved, the router rewrites in the packet and sends the packet to the destination (Step 3). In Figure 6.3b, since the HP Openflow firmware is not able to do any rewriting (efficiently, at least), we prepopulate the ARP table at every host for all other hosts. This results in that the packet built at Step 1 already contains the correct MAC address for the destination IP. Therefore, the router performs no MAC address lookup and simply forwards the packets to the destination (Step 2).

6.2.2 Network Installation

The testbed, shown in Figure 6.4, is designed to provide isolated connectivity for OpenFlow experiments while providing hosts and network testers (see Section 6.3 for experimental evaluation). This is achieved by deploying connectivity via three networks. First, comes the connection to CERN's General Purpose Network (GPN), then there is the Control Network (CN) and finally the OpenFlow network (OF).

Figure 6.5 shows the block connectivity map used with our testbed. The GPN provides connectivity to the world which allows us to install software on the hosts and update the firmware running on the switch devices. The CN is the command and control network, which provides the Network FileSystem (NFS) where the data and systems are stored for the hosts. The GETB Machines (see 6.3 for a complete description) are installed as individual machines each connected to the GPN via its own interface. The OF is the actual connectivity which defines the topology of our test network, as shown in Figure 6.5.

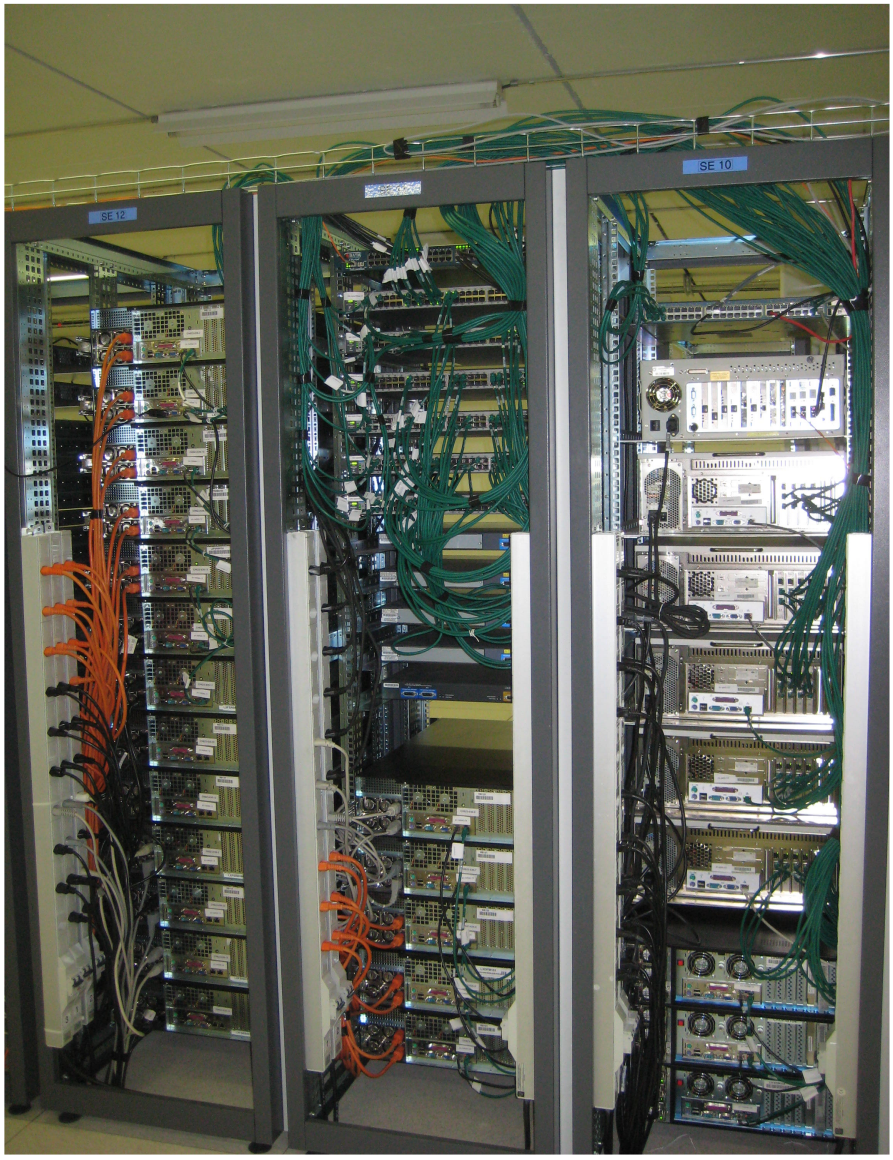


Figure 6.4: *The testbed as it is implemented at CERN's computing center.*

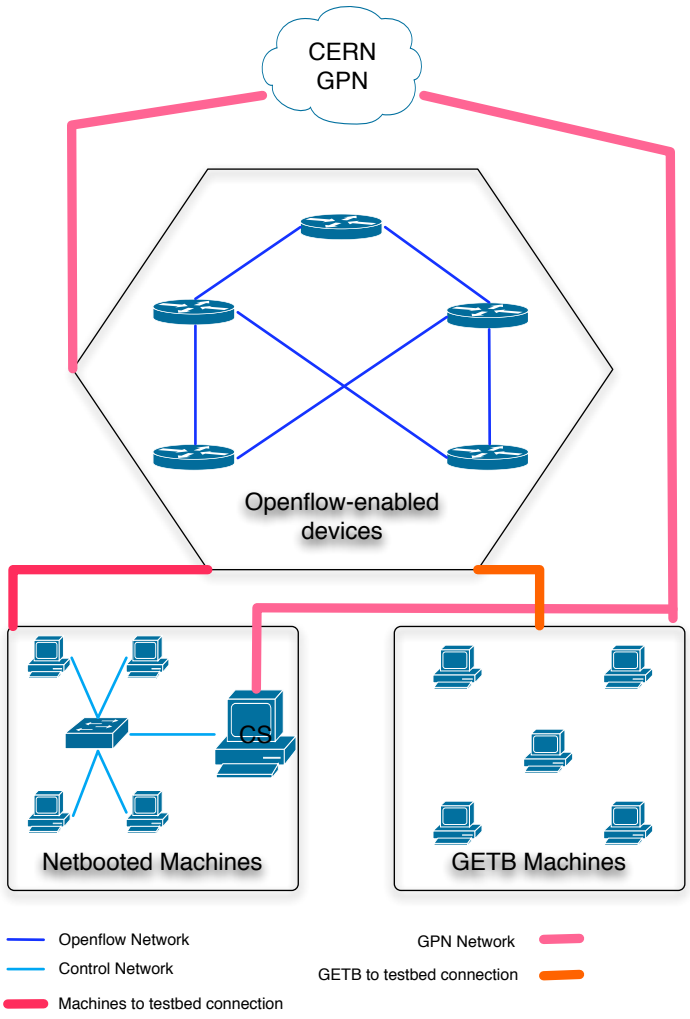


Figure 6.5: *The connectivity map of the Testbed.*

6.2.3 Machine deployment

The hosts are netbooted[4] from the Central Server (CS). Each is configured to obtain an IP address at boot time via the Dynamic Host Configuration Protocol (DHCP). DHCP provides the hosts with extra information such as the gateway address, and the netbooting server. The host downloads a bootloader from the netbooting server and boots as if it had the bootloader locally. The bootloader instructs the host to mount an NFS volume where all the boot and data for the host to run is contained. For more information on netbooting, the reader is referred to [Neta].

Each host has a dedicated directory on the CS which contains the specific configuration of that particular host. Therefore, each host behaves as a diskless machine and pulls its root filesystem from the CS. This approach has several advantages. First, the configuration of the hosts is simplified as they can all be done on the CS. Second, deployment of a new machine is accelerated as we can simply copy another hosts' directory and modify the configuration. Third, since the directory containing all the applications used by the hosts is shared by all the hosts, we just need to install application into this directory and it is directly available on all the hosts.

6.3 GETB Network Testers

The Gigabit Ethernet TestBed [CSLM05] (GETB) is a FPGA-based platform designed to provide a flexible and fully programmable device testing environment. The GETBs were designed to deliver full 1Gb/s wire speed full duplex for two ports running concurrently.

Modifications to the FPGA firmware are difficult and time consuming. This is why the GETB was programmed to offer services. A set of relatively simple functions were implemented in the FPGA. These were then used as building blocks for more complex behavior the hardware functions were accessible to the user through the GETB control software (GETB/CS).

The aim of the GETB/CS was to provide a common control infrastructure for all the GETB-derived projects. We needed a system flexible enough to allow complete customization for the different applications. We also wanted to be able to easily automate the actions of the GETB cards. This requirement practically excluded the development of a Graphical User Interface specifically for the GETB (which we thought would limit the functionality and would be difficult to maintain and further improve). Finally, our decision was to build almost all the GETB control system using the Python scripting language [Pyt].

[4] Netbooting consists in booting machines from a server over the network. These machines can either be diskless or not, but all the boot information is contained on the network server.

The GETB/CS is divided into two main components: the server component which runs on the computers hosting GETB cards; and the client component running on any other machine which needs to access the resources of the servers. Each machine which contains GETB cards runs the server component. Then, the client is run on a different machine and controls the operation of the GETB cards.

The GETB cards enable you to generate traffic at various rates and in a controlled fashion. Therefore, we are able to devise test scenarios with a relatively large port density as we can connect up to 32 GETB simultaneously.

6.4 Experimental Technique

Now that we have described our experimental testbed setup and and the associated network tester, we are going to briefly define the methods used to test our protocols.

6.4.1 Parasite Traffic

In this test, we measure the capacity of the network device to forward traffic in the presence of interfering traffic. This test will indicate the capacity of the router to avoid areas with pre-existing congestion.

6.4.2 Fully Meshed Traffic

Fully meshed traffic, in which traffic is transmitted from all active ports to all other active ports. This will allow us to determine the drop-rates and throughput.

6.4.3 Latency

In this scenario the idea is to measure the time required to transmit a packet from one host to another while the network is transporting traffic at different rates. Doing so will bring out the capability of the routing protocol to route traffic efficiently and through the current best path.

6.5 Summary

In this section, we have presented the tools which have enabled us to develop our congestion-aware protocols. Moreover, we have introduced the technologies which allow us to deploy our testbed and end nodes. Finally, we have described our tester boards which will generate the traffic we need to test our protocols. The next section will present the results obtained from our test environment.

Chapter 7

Results & Discussion

"Insanity: doing the same thing over and over again and expecting different results."
– Albert Einstein

In this chapter we will present the most notable results which were obtained from our real-world testbed implementation. First, we will present a scenario in which our protocols perform exceptionally well. Then, we show that even in the extreme worst cases, our protocols still present an improvement over the legacy protocols. Finally, we show how the latency and packet loss are affected by our protocols. But before presenting the results we should introduce the topologies on which the results where generated.

7.1 Test Topologies & Associated Routing Tables

The Flower, shown in Figure 7.1, is the first topology we present. It consists of a fully connected square with two networks at either side of this network. This topology is quite common in campus networks as it provides redundant paths between the exterior networks (SW1 and SW6 in this case). While it contains relatively many links it only delivers a maximum of two absolutely independent paths from any source to any destination.

The low number of independent paths means that the opportunity for leveraging the alternative paths is reduced. Therefore, the full-mesh runs should display similar results regardless of the protocol used. It's associated routing table is shown in Table 7.1.

The next topology used is called the Pentagon due to its pentagonal shape. This topology presents many more alternative paths from a given source to

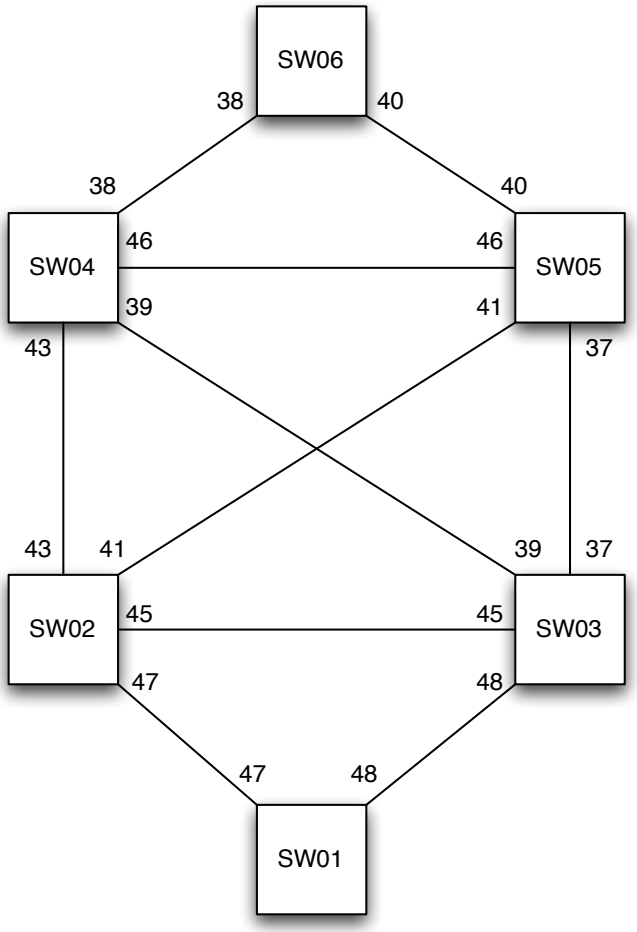


Figure 7.1: *The Flower topology.*

		Destination					
		SW1	SW2	SW3	SW4	SW5	SW6
Source	SW1	L	47	48	[47,48]	[47,48]	[47,48]
	SW2	47	L	45	43	41	[43, 41]
	SW3	48	45	L	39	37	[37, 39]
	SW4	[43, 39]	43	39	L	46	37
	SW5	[41, 37]	41	37	46	L	40
	SW6	[38, 40]	[38, 40]	[38, 40]	38	40	LOCAL

Table 7.1: *The routing table associated to the Flower topology.*

any destination. This topology is also quite common in campus networks as it consists of a starpoint centered on SW6, neighboring networks are then connected to each other to provide redundancy. The routing table is shown in Table 7.2.

		Destination					
		SW1	SW2	SW3	SW4	SW5	SW6
Source	SW1	L	26	27	[26,25]	[27,25]	25
	SW2	26	L	[26, 27]	28	[27, 28, 26]	27
	SW3	27	[27, 29]	L	[29, 30, 27]	30	29
	SW4	[28, 31]	28	[31, 32, 28]	L	32	31
	SW5	[30, 33]	[32, 33]	30	32	L	33
	SW6	25	27	29	31	33	L

Table 7.2: *The routing table associated to the Pentagon topology.*

It should be also noted that the entries in the routing tables are sorted by distance to the destination. Therefore, the leftmost entry is the shortest to the destination. Moreover, each link in the experimental network runs at 1Gb/s.

Each of these topologies are connected to the GETB machines. There are 32 connections to the GETB machines and there are six switches, therefore we have chosen to drop two GETBs and have five connections per switch. The communication path for the GETBs is determined programmatically and depending on the scenario employed.

7.2 Parasite Traffic

In this section we will present results for both topologies when faced with parasite traffic. The idea of these tests is to run a UDP stream (ie. The

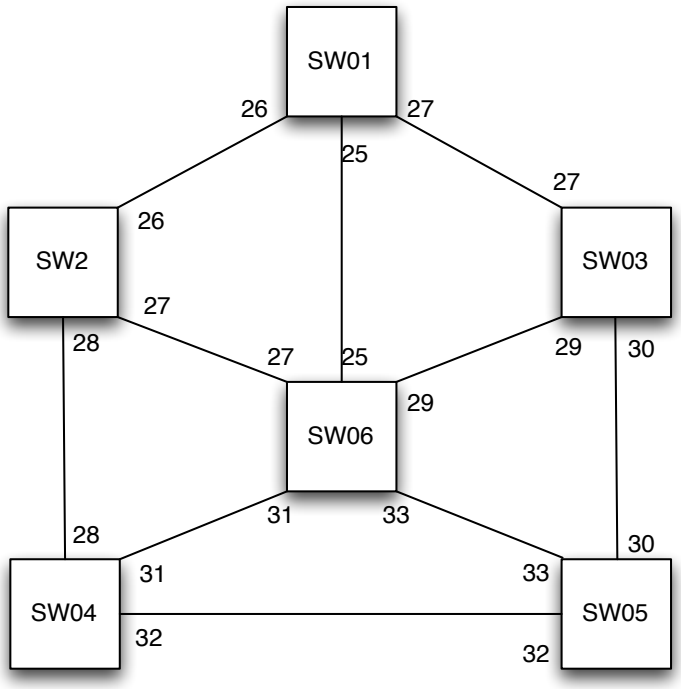


Figure 7.2: *The Pentagon topology.*

parasitic traffic) on a segment over the entire shortest path between two nodes. Then, two TCP flows are sent between the two same nodes. We then compare our results with Shortest Path (SP) only routing and the Hash-Threshold variant of ECMP (HTE).

7.2.1 Parasite Traffic on the Flower Topology

In this setup, parasitic UDP traffic is sent from SW2 to SW6 via SW4. Then, two TCP flows are sent from SW1 to SW6 and the behavior of each protocol is observed.

Shortest Path versus MultiRoute

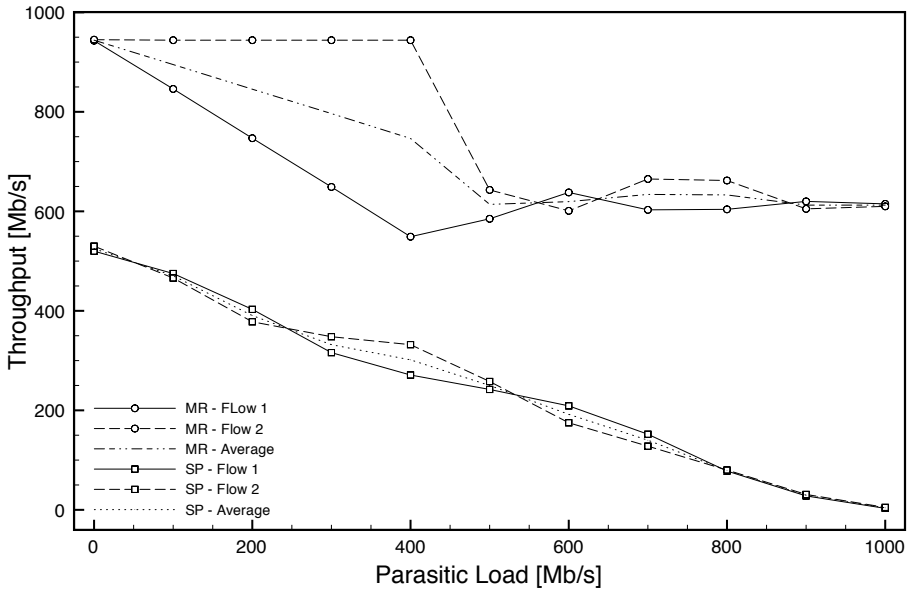


Figure 7.3: *Shortest Path versus MultiRoute in the presence of Parasitic traffic.*

Figure 7.3 shows MultiRoute's behavior in the presence of parasitic input load. We can see that as the parasitic traffic rate increases, one of the flows gradually ramps down while the other is running at line rate. This situation indicates that during these runs MultiRoute did not mark the link running the parasitic traffic as congested, because the associated transfer function was

not sensitive enough. Once this link is marked, the two flows share links along the path to the destination, and this causes TCP's congestion control to rate limit the traffic and therefore we observe that both flows share the bandwidth between themselves. Figure 7.3 also presents the results when running a shortest path only protocol. As expected, the two TCP streams share the available bandwidth between themselves and as the parasitic traffic rate increases, we observe the throughput for both flows being significantly affected.

Shortest Path versus StepRoute

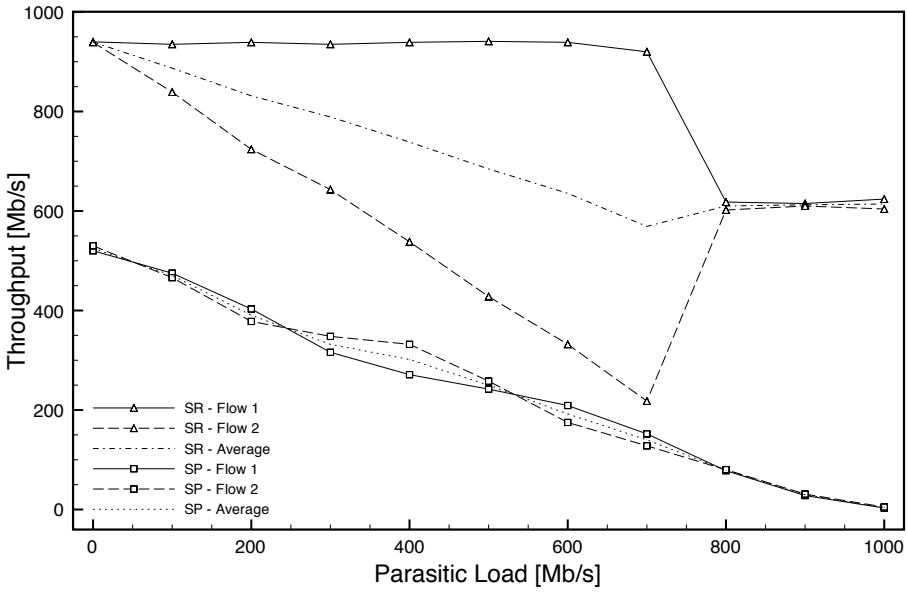


Figure 7.4: *Shortest Path versus StepRoute in the presence of Parasitic traffic.*

When running StepRoute, classes of congestion are used, and therefore as the parasitic load increases it is classified into different congestions levels. As a TCP flow consumes the entire bandwidth available, StepRoute directly classifies the paths used by the first TCP stream into the highest congestion class. Therefore the paths used by the TCP stream are considered to be a worse option than the ones carrying the parasitic load as the parasitic traffic classifies into lower congestion levels. Thus, on Figure 7.4 we see that one TCP flow

runs unhindered at line rate while the other shares with the parasitic traffic until the parasitic traffic is classified into the highest congestion level, at which point the TCP flows share the same paths and therefore share the available bandwidth,

The issue with this behavior is that UDP does not cooperate with TCP (or with anyone for that matter) as it does not implement the congestion control algorithms that TCP does, which means that the UDP traffic will continue to to use all the bandwidth it needs and leaves the remainder for the effective TCP traffic. Therefore, on Figure 7.4 we see that the second TCP flow is affected significantly by the parasitic traffic but it is never worse than the shortest path.

Shortest Path versus PathRoute

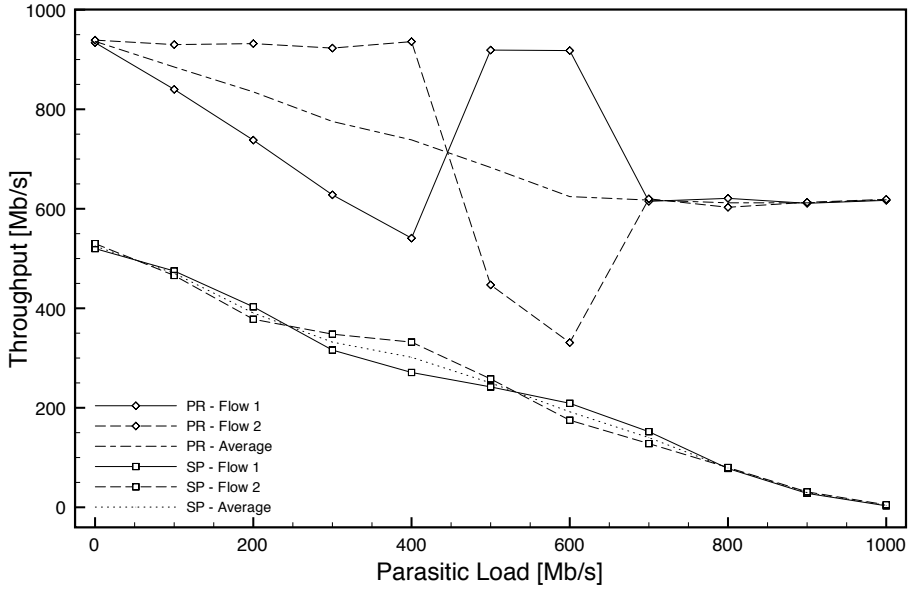


Figure 7.5: *Shortest Path versus PathRoute in the presence of Parasitic traffic.*

PathRoute uses the same congestion marking scheme as MultiRoute. Therefore, the first part of Figure 7.5 behaves similarly to MultiRoute where one flow goes along the alternative path and the other flow shares the traffic with the parasitic traffic. But as the parasitic traffic increases in intensity, the

segments which it uses are eventually marked as congested, but as the first segment of the path SW1 to SW6 is not congested at all, the first flow uses it and then reroutes via SW5, this causes the segment SW1 to SW2 to be congested. When the second flow arrives it avoids the congested area totally and therefore takes the option containing less congested segments, this causes the second flow to share bandwidth with the parasitic traffic. This explains the seemingly exchange in roles between the two TCP flows. Eventually, all the segments used by the parasitic traffic are marked as congested causing both TCP flows to avoid the congested area and share bandwidth.

Hash-Threshold versus MultiRoute

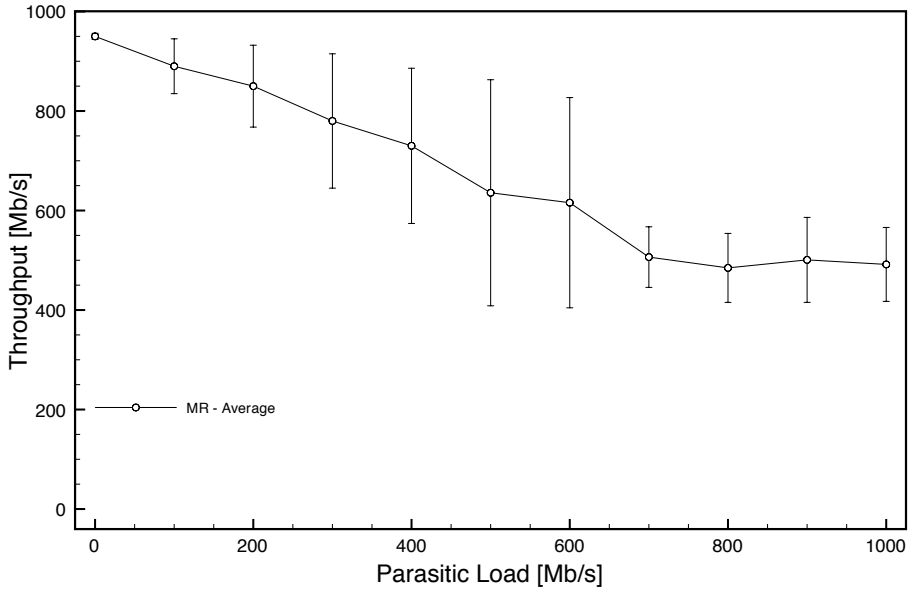
Figure 7.6 shows the average throughput with error bars which represent the variation in throughput observed when running our tests for MultiRoute (Figure 7.6a) and HTE (Figure 7.6b) . These measurements where done while varying the parasitic traffic rate. Considering only the result for HTE, we observe a large variation of minimum and maximum values. This is explained by the fact that HTE does not consider congestion statistics, and therefore flows may in one case share links and not in others. This phenomenon is especially apparent when the parasitic traffic is high, in some cases all flows will share the same path as the parasitic traffic, causing quasi null throughput for these flows, and in others one flow will be alone on an uncongested path resulting in line rate throughput. These results show that in such situations, HTE throughput can become difficult or nearly impossible to predict accurately. Consider now the results for MultiRoute, the variance between the maximum and minimum values is significantly smaller than in the case of HTE, especially when the parasitic traffic is high. As the parasitic traffic increases in intensity, MultiRoute will route all TCP flows onto the same path causing, like in Figure 7.3, to share the available bandwidth while avoiding the path used by the parasitic traffic.These results show that we can predict with a significantly higher degree of confidence the expected throughput of MultiRoute in the presence of other traffic flows. It is also interesting to notice than in average MultiRoute performs better than HTE.

7.2.2 Parasite Traffic on the Pentagon

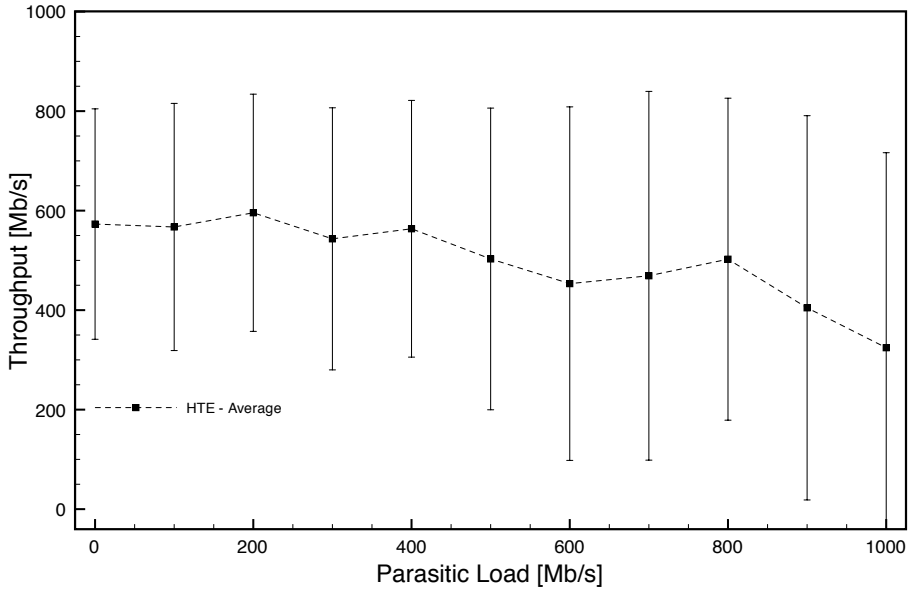
In this setup, parasitic UDP traffic is sent from SW2 to SW5 via SW6. Then, two TCP flows are sent along that same path.

Shortest Path versus MultiRoute

In this situation, there is more than one alternative path to the destination. Therefore, it is expected that all the congestion aware protocols will eventually



(a) *MultiRoute's throughput variation*



(b) *Hash Threshold's throughput variation*

Figure 7.6: *A comparison of the predictability of MultiRoute versus Hash Threshold*

use all the available paths and therefore avoid the parasitic traffic altogether. Indeed, as we can see in Figure 7.7, one TCP flow shares the shortest path to the destination with the parasitic traffic but then switches to an alternative path when the shortest path is marked as congested.

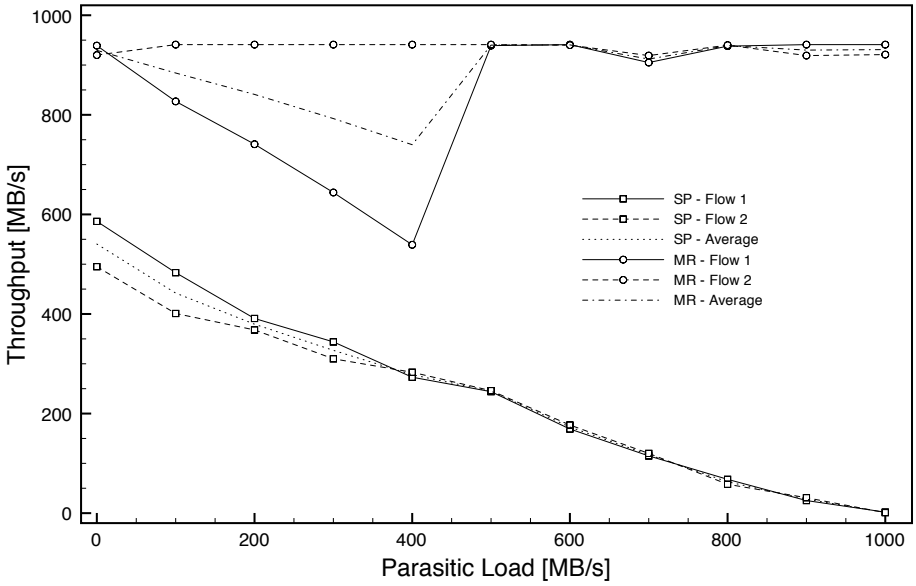


Figure 7.7: *Shortest Path versus MultiRoute in the presence of Parasitic Traffic.*

Shortest Path versus StepRoute

When running this test with StepRoute, we observe the same behavior except that both TCP flows quickly run at line rate as can be seen in Figure 7.8. This can easily be explained by the fact that StepRoute uses congestion classes and not a single bit only. Therefore, as StepRoute quickly marks the link used by the parasitic traffic as lightly congested, it directly uses the alternative path to the destination. We can therefore say that StepRoute is quite sensitive to the congestion status of the various paths.

Shortest Path versus PathRoute

Figure 7.9 shows the same test when running PathRoute. Since MultiRoute and PathRoute use the same mechanism to detect and represent congestion we

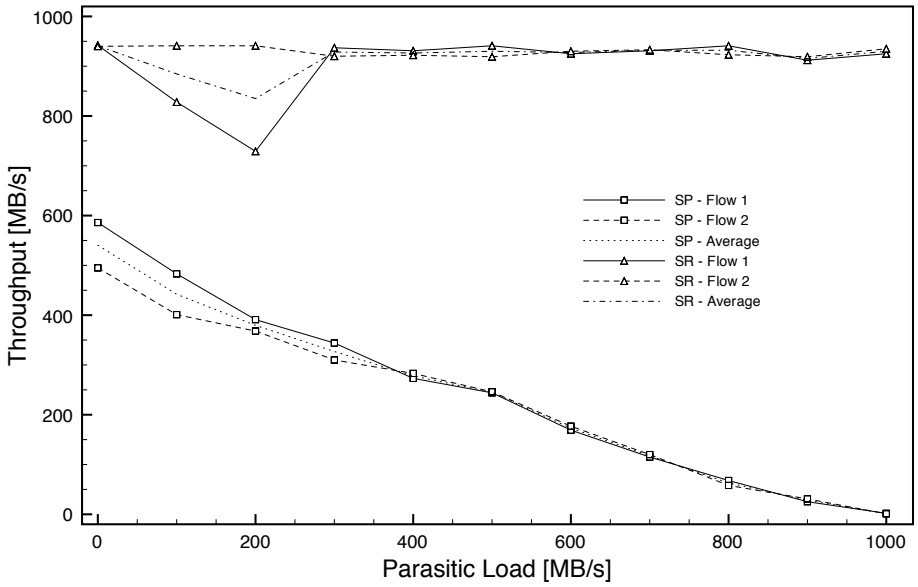


Figure 7.8: *Shortest Path versus StepRoute in the presence of Parasitic Traffic.*

observe a similar result to the one presented in Figure 7.7 with MultiRoute. But the performance of PathRoute is significantly lower than MultiRoute or StepRoute. This can be explained due to the longer feedback loop which is introduced by PathRoute's complexity (ie. the knowledge of congestion for a source to any destination). The long feedback loop causes PathRoute to take decisions which are out of sync with the current network status.

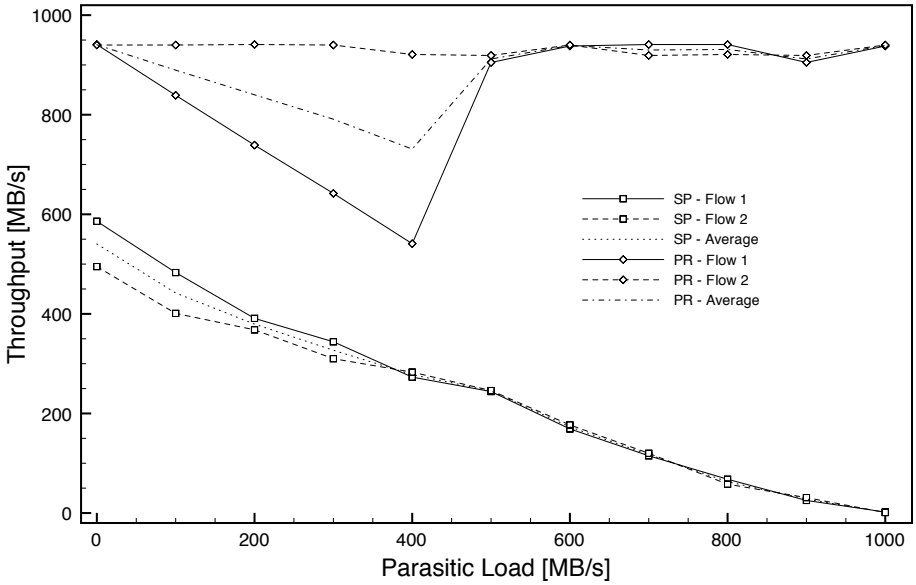


Figure 7.9: *Shortest Path versus PathRoute in the presence of Parasitic Traffic.*

7.3 Fully-Meshed Traffic

Fully-Meshed traffic is the worst possible situation for our protocols as they represent conditions where congestion is equal and omnipresent throughout the network. These tests were performed using the GETB boards. Four end nodes were connected to each switch then each end node sends traffic to each other end node at varying loads from 1% to 25% (as this sums up to 100% for each switch) and the behavior of each protocol is observed.

For both topologies, one might expect to see a linear throughput until some asymptotic limit, but this is not what was observed in either case. This can

be explained by the fact that effective throughput is essentially a function of a switch's packet drop probability. The higher this probability is, the lower the throughput will be. Therefore, when traffic is run through one switch we observe linear increase in throughput as the load increases, because the drop probability is a linear function of the load. Unfortunately this is not the case when there are multiple switches connected to each other. Indeed, while the packet drop probability is still a function of the load, it is now also conditional on the packet drop probability of the previous switch. This conditional probability manifests itself in the following plots as variations (or "kinks") in the measured throughput curves.

An attempt was made at modeling this conditional probability using Bayesian Networks [Hec97], but the model turned out to be vastly too complex and resulted in a combinatorial explosion for the expression of this probability.

7.3.1 Fully-Meshed Traffic on the Flower

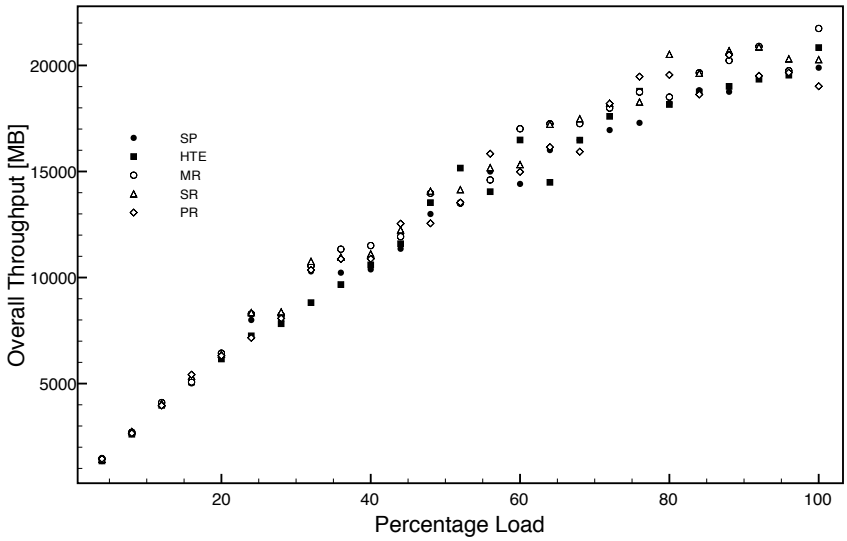


Figure 7.10: *Fully-Meshed Traffic running on the Flower.*

As is shown in Figure 7.10, all the protocols seem to perform similarly. We can explain this by the fact that all the paths are congested and therefore our congestion aware protocols quickly default to the shortest path. That said, there is a slight (albeit marginal) increase in performance for our protocols. This increase in performance can be better observed when the tests are run

successively in order to eliminate the effect of the variations in the throughput as shown in Figure 7.11. We observe that even though the throughput for both protocols is similar, we clearly notice that as the load increases the difference in the throughput between the shortest path protocol and the congestion-aware increases in average. Actually, there is about a 2GB difference on average, this equates to roughly 66MB every second which means that our protocols have added $\frac{2}{3}$ of a Gigabit link over a shortest path protocol. We believe that this a significant achievement in a Fully-Meshed environment.

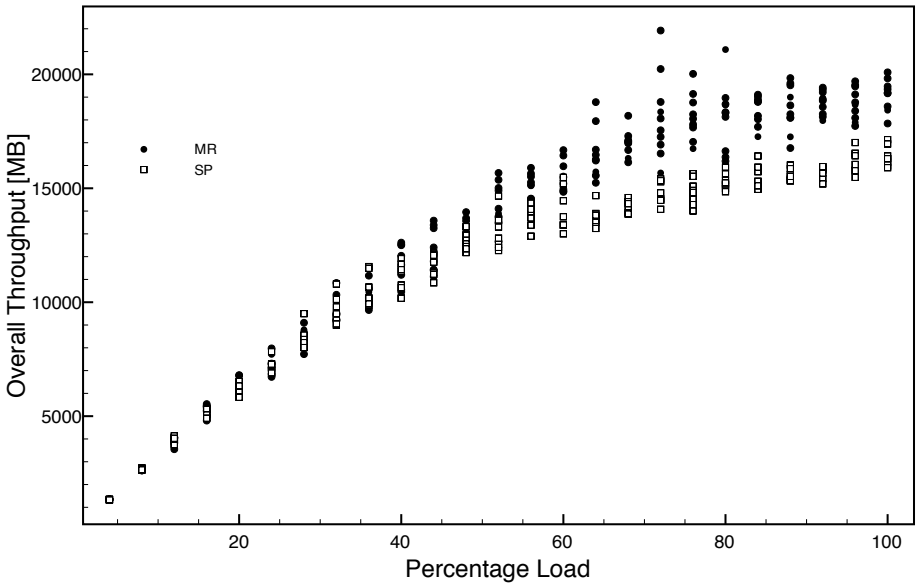


Figure 7.11: *Successive Fully-Meshed Traffic running on the Flower.*

7.3.2 Fully-Meshed Traffic on the Pentagon

In a topology which offers more alternative paths, we clearly see that our protocols consistently outperform legacy protocols. It is particularly interesting to observe that the Hash Threshold variant of ECMP performs least well. We can explain this by the manner in which HTE selects its next-hop (based on the source and destination MAC addresses), source-destination pairs will always use the same path regardless of the load they impose on the network and therefore not only does HTE overload those paths it also uses longer congested

paths to the destination which significantly decreases its performance.

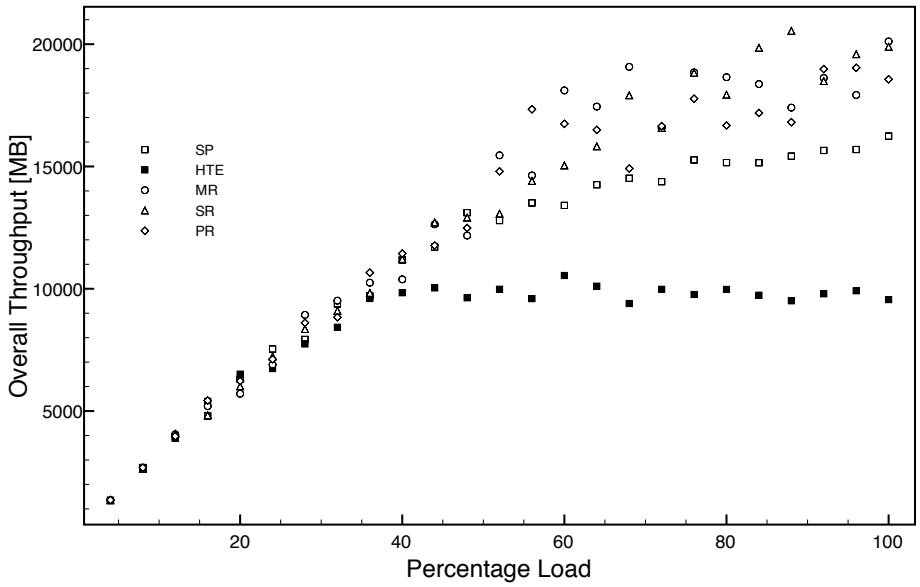


Figure 7.12: *Fully-Meshed Traffic running on the Pentagon.*

7.4 Latency & Packet Loss Tests

These final results were also obtained using the GETB boards. Full-Mesh traffic was run on each topology, then a stream of probe traffic was sent between two nodes on the network. The time of arrival of the probe traffic as measured and the number of packets lost was counted.

Before explaining the results in this section, we should comment the pseudo step function that is observed in both latency plots below. The linear part of the curve can be seen as the buffers at a router filling, once full we see a significant increase in latency as the switch requires more time to let a packet through its full buffers. From the theory we would expect a linear increase as the router buffers fill up followed by a flat line when all the buffers are full. Unfortunately, this is where our theory and practice differ as it is mathematically complex to construct a queuing model which can tolerate losses. If we consider a single router situation we would observe a linear increase followed

by a flatline as the buffers start to drop packets. But now, as we have several routers in a row, we observe a step function which is explained by the fact that a router who is dropping packets has an influence on the loss probability of the next router in the chain who now has a lower loss rate due to a lighter input. Hence, we observe a step function as the routers in the chain have differing rates of buffer saturation.

7.4.1 Latency & Packet loss on the Flower

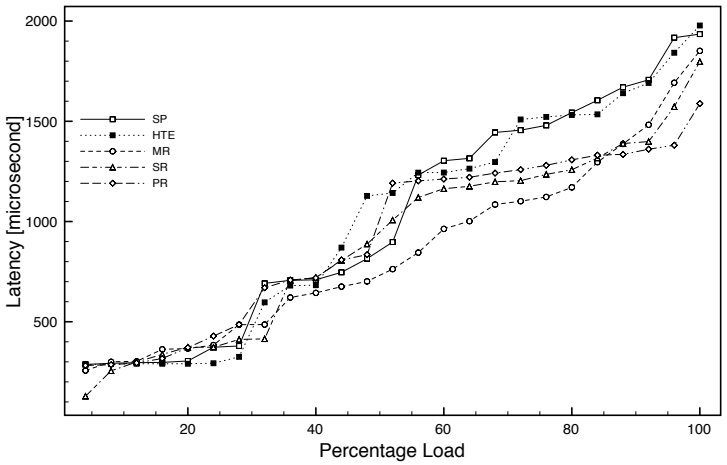


Figure 7.13: *Latency plots for protocols running on the Flower.*

Figures 7.13 and 7.14 show the latency and packet loss experienced by the different protocols respectively. In the case of latency we can see that our protocols present paths which provide less latency, which is to be expected because in this topology we have alternative paths of identical lengths and therefore our latency should either be equal to the shortest path or less. For packet loss, we see that both MultiRoute and StepRoute perform better than the other protocols, this is consistent with our previous results (see Figure 7.10) as they present increased performance in terms of throughput. On the other hand PathRoute performs less well, but again this is consistent as PathRoute does not perform well in terms of throughput during a full-mesh.

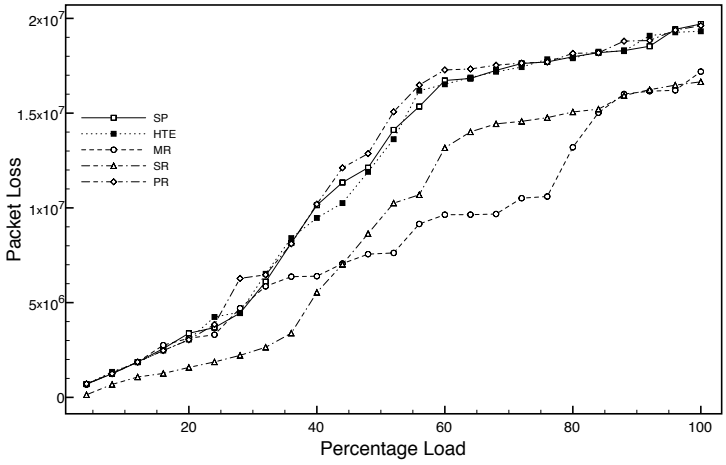


Figure 7.14: *Packet Loss for protocols running on the Flower.*

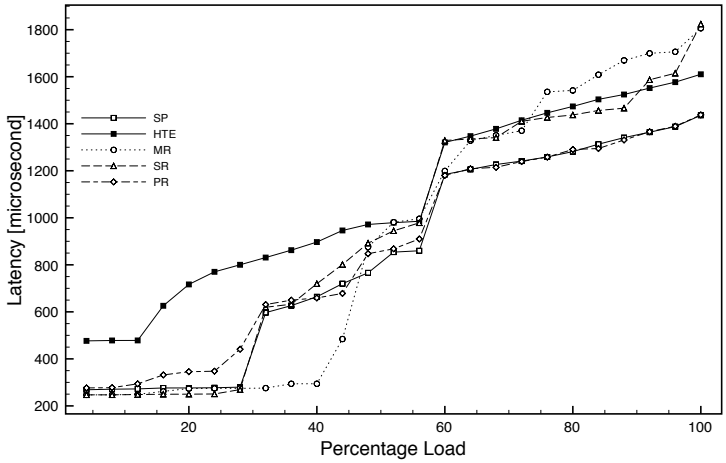


Figure 7.15: *Latency plots for protocols running on the Pentagon.*

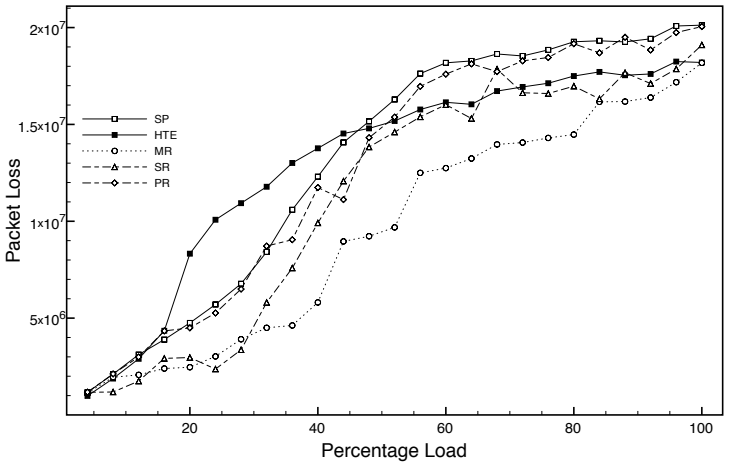


Figure 7.16: *Packet Loss for protocols running on the Pentagon.*

7.4.2 Latency & Packet loss on the Pentagon

In this case, it is interesting to see that protocols, namely MultiRoute and StepRoute, present a higher latency as shown in Figure 7.15. We can explain this by simply noticing that the alternative paths for this topology are longer that the shortest path and therefore require a higher transit time. Figure 7.16 confirms this observation as the packet loss presented by StepRoute and MultiRoute is lower than the Shortest Path protocol, which means that an alternative path must have been used else the packet loss would have been similar to the shortest path.

7.5 Summary

As these results have shown, our congestion-aware protocols have outperformed legacy protocols. Both, in worst case situations (Full-Mesh) and rather advantageous positions (Parasitic Traffic). Nevertheless, deployment of such protocols requires a rigorous study of the type of services which will de offered by the network. For example, if a network is supposed to carry mainly Voice-over-IP traffic then the network engineer should make sure to provide paths of identical lengths to all destinations in an effort to keep the traffic latencies constant.

Naively, one would expect that a protocol which has quasi network wide knowledge, such as PathRoute, would deliver optimal results. Unfortunately

this is not the case, from the results presented in this chapter we clearly observe that PathRoute performs significantly worse than its peers (MultiRoute and StepRoute). This phenomenon is exacerbated when we consider fullmesh situations. The reason for this lack in performance can be found in control theory's standard problem, ie. the duration on the feedback loop. While all the congestion-aware routing protocols presented in this thesis suffer from a delay between observing congestion and adapting the network state to reflect this change, the impact on PathRoute is more significant that on its peers. Indeed, any update in any area of the network may, in the worst case, trigger updates at all the routers in the network and therefore cause some routers to have opposing network views thereby causing poor routing decisions to be taken.

Chapter 8

Conclusions & Future Work

"Finally, in conclusion, let me say just this. "
– Peter Sellers

8.1 Achievements

The objective of this thesis was to revisit the concept of congestion-aware routing. This idea was introduced in the early days of the ARPANET but quickly abandoned due to out of order delivery of packets which caused the effective throughput of the network to be catastrophic. To achieve this goal, we have divided the problem into three sub-problems, namely:

1. Path Construction - The multiple path discovery process relies on the existence of a shortest path between source and destination points. After establishing the shortest path cost (the reference cost), each alternate path is computed whose cost is within a reasonable delta of the reference cost.

2. Network Monitoring - The statistics are polled locally by the router and sent to neighboring routers, this process is performed in-band and not by an external monitoring process. Then, depending on the transfer function used, a representation of the congestion is derived.

3. Topology Representation - Each router maintains its own routing vector, consisting of congestion representation of its paths to different networks. Routing vectors are then exchanged with neighboring routers using the Monitoring protocol .

Using this approach we have been able to build three different congestion-aware protocols which contrast with previous multipath protocols in several aspects. First, they treat alternative paths as another possibility rather than a route to take in case of failure. Second, they do not require full knowledge of the network topology as they obtain the congestion status of their direct neighbors only. Finally, they avoid out-of-order packet distribution by using flows as a basis for routing rather than packets. This coupled with the fact that routing decisions are immutable completely side-steps the problem of out-of-order packets. On the other hand, this raised the following question: *Can a protocol with such restrictions still provide a benefit in term of performance?* We believe that this thesis shows that the answer to this question is **Yes**!

Another aspect of this thesis was the implementation a real world testbed in which our congestion-aware protocols were deployed. The design, construction and evaluation of this testbed was carefully studied. First, it was designed so that it could easily be extended, for example adding an end node to the test bed is as easy as entering a few configuration lines in the file server and then booting the desired end node. This end node would then have all the functionality previously existing nodes have. Second, the testbed was constructed in such a was that it would be simple to interconnect and physically simple to extend. Finally, by designing and constructing the system with the knowledge that the system would have to be tested, it was simple to provide connectivity to the test machines and the traffic generators.

8.2 Summary of Results

Based on the results presented in Chapter 7 we believe that we can say that our congestion aware protocols provide an significant increase in performance when compared to legacy protocols. This results have been tested on two representative topologies.

Our protocols perform best when asked to route around a congested area. We show that is some cases our protocols provide line rate performance when shortest path or Equal Cost MultiPath (ECMP) provide close to zero throughput. We also show that even though our protocols are congestion-aware, they provide a predicable throughput where ECMP could not.

We have also shown that in the face of fierce adversity, ie. Full-Mesh traffic, our protocols provide an increased level of performance. This worst case scenario demonstrates that these protocols can be expected to deliver increased performance on non-traffic engineered networks.

While we show that our protocols present increased performance, a network engineer deploying such protocols should study the requirements of the network he is implementing. We believe the questions that need to be answered are:

1. Does the expected traffic tolerate loss?
2. Does the expected traffic tolerate delay?
3. Does the expected traffic consist of short or long lived flows?
4. Is the traffic on this network expected to be traced?

The answers to these questions will determine which protocol to use and/or the type of topology to deploy (either equal cost lengths or variable alternative path lengths).

8.3 Future Work

As with any thesis, many lines of research remain. We shall state a few here in the form of open ended questions:

- How does one trace traffic through a multipath network? Is debugging even possible?
- What are the effects of using other metrics as a basis for congestion detection?
- Do our congestion-aware routing protocols scale to networks with hundreds of routers?
- How can the feedback loop required for congestion-aware routing protocols be made as small as possible?
- Will it ever be possible to mathematically model multipath networks accurately?

Appendix A

Collection of derivations

A.1 Disruptions in the Hash Threshold variant of ECMP

TCP protocols perform superbly in situation where the path they flow does not change during the connection. Disruptions is the measure of how many flows will be affected by a path change. Consider now a hash based next hop selection, if our hash function is uniform then each next hop has an equivalent probability to be selected. Therefore, we can say that the hash-space is equally distributed between next-hops (or regions) as shown in Figure A.1

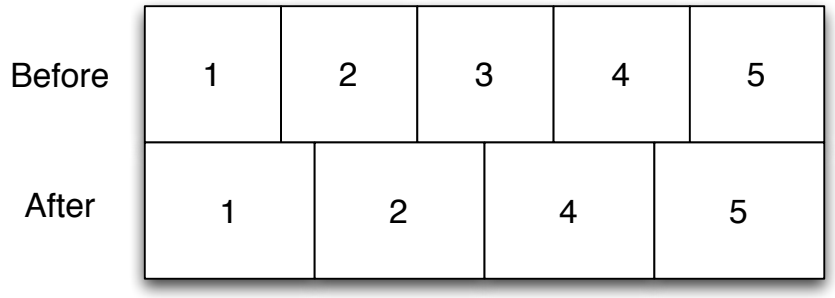


Figure A.1: *Before and After the deletion of the third next hop possibility.*

We can formalize this by considering the removal of region K in the presence of N regions. This causes the remaining $N-1$ regions to fill the $\frac{1}{N}$ space left by the removal of region K. Therefore each remaining region grows by

$\frac{1}{N(N-1)}$. But now, this means that every non-end region has to be moved to accommodate for its now grown neighbor. So the first regions grows and causes the second region to shift towards towards region K by $\frac{1}{N(N-1)}$, and in turn the third region is caused to move by $\frac{2}{N(N-1)}$ and so on until both ends meet at the borders of K. These moves causes $\frac{i}{N(N-1)}$ flows in region $i+1$ to to be taken over by region i. Therefore, we have the following equations:

$$disruptions = \sum_{i=1}^{K-1} \frac{i}{N(N-1)} + \sum_{i=K+1}^{N} \frac{i-K}{N(N-1)} \tag{A.1}$$

$$= \frac{1}{N(N-1)} \left(\sum_{i=1}^{K-1} i + \sum_{i=K+1}^{N} i - K \right) \tag{A.2}$$

$$= \frac{(K-1)(K) + (N-K)(N-K+1)}{2N(N-1)} \tag{A.3}$$

Equation A.3 formalizes the number of flows which would be disrupted if a link on a path to a next-hop fails.

A.2 Backpressure messages given in AMP

A backpressure message in AMP is given by two factors; consider the link $\overline{XY}$:

1. The load on the links directly connected to X; and
2. The contribution of the traffic routed at X on the links from Y to Z ($\overline{YZ}$).

The first point is immediate as the router can directly measure the congestion on their direct links, this is given by ρ. The remaining value describes the messages sent from the neighboring routers called $B_{Z \to Y}$ which is itself computed based on the two factors described above. Therefore, we can simply note BMs as $B_{Z \to Y}$, formally $BM(Y, X) = B_{Y \to X}$. $BM(Y, X)$ is the backpressure message sent from node Y to node X. We can now define $B_{Y \to X}$ as a function of the directly measurable congestion values and the $B_{Z_i \to Y}$, therefore we have:

$$B_{Y \to X} = f(\rho_{\overline{XY}_i}, B_{Z_i \to Y}), \forall i = 1, \ldots, n \tag{A.4}$$

As the contributions to congestion from traffic routed at X vary at each neighboring router, since only a fraction of the overall traffic is routed to a

particular neighbor, AMP proposes to define these varying contributions as a weighted average where the weight for link $\overline{YZ_i}$ corresponds to the ratio of traffic on the link $\overline{YZ_i}$ that has arrived from X via Y and the total traffic on the link $\overline{YZ_i}$. Therefore, we have:

$$B_{Y\to X} = \sum_{Z_i\in\Omega_Y\setminus X} \frac{\beta_{\overline{YZ_i}}(X)}{\beta_{\overline{YZ_i}}} max(\rho_{\overline{XY}_i, B_{Y_i\to X}}) \tag{A.5}$$

Where Ω_Y is the set of all neighbors of node X, $\beta_{\overline{YZ_i}}(X)$ is the number of bytes sent from node X via Y to Z_i, and finally $\beta_{\overline{YZ_i}}$ denotes the total number of bytes sent from any node $\in \Omega_Y\setminus X$ via Y to Z_i.

A.3 Dependence of Message Length and Interarrival times

Let us derive the joint probability $P(v_n, a_{2n})$ between the message length and the inter-arrival time at some queue. For this consider the tandem network given in Figure 4.6, and the following quantities:

v_n = message length of the n^{th} message.

a_{in} = the inter-arrival time between the $(n-1)^{th}$ message and the n^{th} at node i.

Now we recall the assumptions of queuing systems with Markovian properties of Section 4.3.1 where both v_n and a_in are described by exponential distributions, given by:

$$p(a_{1n}) = \lambda e^{-\lambda a_{1n}} \tag{A.6}$$

$$p(v_n) = \mu e^{-\mu v_n} \tag{A.7}$$

Due to these facts, Theorem 1 holds and thereby we have that the inter-departure times at the first node in the tandem are identical to the inter-arrival times at the second node. Therefore we have:

$$p(a_{2n}) = \lambda e^{-\lambda a_{2n}} \tag{A.8}$$

When the n^{th} message leaves the first node:

1. it is separated by a time gap g_n from the $(n-1)^{th}$ message. Or,
2. transmitted immediately after the $(n-1)^{th}$ message is done transmitting.

It is clear that Case 1. only occurs if the first node emptied out before receiving the n^{th} message, which happens with probability $1-\rho$. The second

case occurs when the first node is busy which has probability ρ. Where $\rho = \frac{\lambda}{\mu}$ is the utilization factor as defined in Section 4.3.1. We now have,

$$p(v_n, a_{2n} = \rho P(v_n, a_{2n}|node\ 1\ busy) + (1-\rho)P(v_n, a_{2n}|node\ 1\ empty) \quad \text{(A.9)}$$

Trivially, the probability that message n has length v_n and arrives at a_{2n} such that node 1 is busy is equal to the probability that $v_n = a_{2n}$, otherwise the queue would be idle. Thus,

$$\begin{aligned} P(v_n, a_{2n}|node\ 1\ busy) &= P(v_n = a_{2n}) \\ &= \mu e^{-\mu v_n} \mu_0(a_{2n} - v_n) \end{aligned}$$

Where μ_0 is the unit impulse function.

Alternatively, the probability that message n has length v_n and arrives at a_{2n} such that node 1 is empty implies that $a_{2n} - v_n \neq 0 = g_n$, obviously otherwise the queue would always be busy, combined with the fact that the message length permits the queue to become empty. Due to the fact that g_n follows the same distribution as $p(a_{2n})$, we therefore have,

$$\begin{aligned} P(v_n, a_{2n}|node\ 1\ empty) &= P(a_{2n}|v_n,\ node\ 1\ empty)P(v_n|node\ 1\ empty) \\ &= P(g_n = a_{2n} - v_n)P(v_n|node\ 1\ empty) \\ &= P(g_n = a_{2n} - v_n)p(v_n) \\ &= \lambda e^{-\lambda(a_{2n} - v_n)} \mu e^{-\mu v_n} \end{aligned}$$

And finally the overall expression becomes:

$$p(v_n, a_{2n} = \mu e^{-\mu v_n} \mu_0(a_{2n} - v_n) + \lambda e^{-\lambda(a_{2n} - v_n)} \mu e^{-\mu v_n} \quad \text{(A.10)}$$

By analysis of Equations A.10, A.7 and A.8, we can clearly see that:

$$p(v_n, a_{2n}) \neq p(v_n)p(a_{2n})$$

which, by definition independent events, illustrates the deep dependence between the message lengths and the inter-arrival times.

A.4 The Price of Anarchy in Multipath networks with linear cost functions

First let us define more formally the delay of a flow as a function of the links l it traverses as,

$$C(f) = \sum_{l} c_l(f_l) f_l \tag{A.11}$$

where,

- c_l is the cost induced by a flow on the link l,
- f_l is the amount of flow using link l.

Due to the fact that the model for Selfish routing is a static one, we only consider flows which are in a state of equilibrium. Therefore it is clear that a flow is not in a equilibrium state if it can shift some traffic to another link while reducing the overall delay of the network. Thus, we have the following definition:

Definition. *A flow f is a* Nash flow *if for all paths $P_1, P_2 \in P$ and amounts $\delta \in]0, f_{P_1}]$ of traffic, we have:*

$$c_{P_1}(f) \leq c_{P_2}(\tilde{f})$$

where,

$$\tilde{f} = \begin{cases} f + \delta & \text{if } P = P_1 \\ f - \delta & \text{if } P = P_2 \\ f & \text{if } P \notin \{P_1, P_2\} \end{cases}$$

Alongside the definition of a Nash flow, we have the straightforward definition of an optimal flow, which is simply:

Definition. *A flow f is considered optimal if it is the flow with minimum possible delay. It is then denoted by f^*.*

As an example, a Nash flow is illustrated by the first situation in Figure 4.3, whereas an optimal flow is depicted by the second situation.

Finally, we can define the *worst-possible* ratio between the cost of a Nash flow and that of an optimal flow. This ratio is known as the *Price of Anarchy* and is given by the following relation and Equation A.11:

$$\rho = \frac{C(f)}{C(f^*)} \tag{A.12}$$

Let us know postulate that optimal and Nash flows are the same things, just with different delay function. In order to establish this we must establish a relationship between optimal and Nash flow as a function of the delay functions.

Definition. *If c is a differentiable delay function, then the corresponding marginal delay function c^* is defined by:*

$$c^* = \frac{d}{dx}(xc(x)) \tag{A.13}$$

By considering definitions A.4 and A.4, we can conclude that:

Definition. *Let G be a network defined by continuously differentiable delay functions and corresponding marginal delay functions c^*, then a flow f for G is optimal if and only if it is at Nash equilibrium under the marginal delay functions.*

Now let us consider a network with linear delay functions of the form $ax+b$, therefore by equation A.11 we have the expression for the delay in such a network:

$$C(f) = \sum_l a_l f_l^2 + b_l f_l \tag{A.14}$$

and thus trivially we have that:

Definition. *Suppose a network has linear delay functions and f is a flow at Nash equilibrium. Then,*

1. *$c_l^*\left(\frac{f_l}{2}\right) = c_l(f_l)$ for any link l;*
2. *The flow $\frac{f}{2}$ is optimal for the same network with have the link capacity.*

The relations in definition A.4 are simply a result of applying Equation A.14.

We now can express the delay caused by an optimal flow as:

$$C(f^*) \geq C\left(\frac{f}{2}\right) + \sum_l c_l^*\left(\frac{f_l}{2}\right)\frac{f_l}{2} \tag{A.15}$$

By definition A.4 and equation A.11 we have that:

$$\sum_l c_l^* \left(\frac{f_l}{2}\right) \frac{f_l}{2} = \frac{1}{2} \sum_l c_l \left(f_l\right) f_l \tag{A.16}$$

$$= \frac{1}{2} C(f) \tag{A.17}$$

All we need to solve now is $C\left(\frac{f}{2}\right)$, this is simple if we recall that we are considering linear cost functions and that therefore we can apply Equation A.14:

$$C(\frac{f}{2}) = \sum_l \left(\frac{1}{2} a_l f_l + b_l\right) \frac{f_l}{2} \tag{A.18}$$

$$\geq \frac{1}{4} \sum_l \left(a_l f_l + b_l\right) f_l \tag{A.19}$$

$$= \frac{1}{4} C(f) \tag{A.20}$$

By introducing equations A.20 and A.17 into Equations A.15, we obtain:

$$C(f^*) \geq \frac{3}{4} C(f) \tag{A.21}$$

Equation A.21 allows us to conclude that the Price of Anarchy for a multipath network with linear delay functions is at most $\frac{4}{3}$. This result can be extended to polynomial delay functions, and we obtain the following Price of Anarchy as a function of the degree of the polynomials:

$$\rho = \frac{(p+1)\sqrt[p]{p+1}}{(p+1)\sqrt[p]{p+1} - p} \overset{p\to\infty}{=} \frac{p}{ln(p)} \tag{A.22}$$

Appendix B

Collection of Pseudocodes

B.1 Dijkstra's Algorithm Pseudocode

Algorithm 1 Dijkstra's Algorithm

Require: A network $R = (V, E, c), |V| = n, |V| = m$, where $c : E \mapsto \Re_+$ is a weighting of arcs on the graph $G = (V, E)$. A particular vertex $S \in V$ called the source.

Ensure: For all vertex $v \in V$, the lengths $D(v)$ are guaranteed to be minimal.

$D(S) = 0$
$D(v) = \infty, \forall v \neq S \in V$
$\forall v \in V, P_v = \varnothing$
$Q = V$
while $Q \neq \varnothing$ **do**
  $u \leftarrow REMOVE - MIN(Q)$
  for each $v \in Adj(u)$ **do**
    if $D(v) > D(u) + c(u, v)$ **then**
      $D(v) \leftarrow D(u) + c(u, v)$
      $P_v = v$
    end if
  end for
end while

B.2 Bellman-Ford's Algorithm Pseudocode

Algorithm 2 Bellman-Ford's Algorithm

Require: A network $R = (V, E, c), |V| = n, |V| = m$, where $c : E \mapsto \Re$ is a weighting of arcs on the graph $G = (V, E)$. A particular vertex $S \in V$ called the source.

Ensure: For all vertex $v \in V$, the lengths $D(v)$ are guaranteed to be minimal.

```
for each vertex u ∈ V do
    d[u] = ∞
    p[u] = u
end for
d[s] = 0
for i = 1 to |V| − 1 do
    for each edge (u, v) ∈ E do
        if (w(u, v) + d[u] < d[v]) then
            d[v] = w(u, v) + d[u]
            p[v] = u
        end if
    end for
end for
```

B.3 MultiRoute Path Construction - Pseudocode and Proof

Require: A Network $R = (V, E, c), |V| = n, |V| = m$, where $c : E \mapsto \Re_+$ is a weighting of arcs on the graph $G = (V, E)$. A particular vertex $S \in V$, and ε which is the tolerance to the optimal path length

Ensure: For all vertex $v \in V$, the lengths $D(v)$ of a alpha-shortest independant paths within ε of the optimal path length (the ε-property). From S to v ($D(v) = \infty$ if there is no path from s to v in G) along with the immediate predecessor(s) P_v of vertex v in such a path.

$D(S) = 0$
$D(v) = 0, \forall v \neq S \in V$
$\forall v \in V, P_v = \varnothing$
$Q = V$
while $Q \neq \varnothing$ **do**
  $u \leftarrow REMOVE - MIN(Q)$
  $K \leftarrow \{k \in Q | |D(u) - D(k)| \leq \varepsilon\}$
  if $u = \varnothing$ **then**
    Graph disjoint, QUIT
  end if
  for each $v \in Adj(u)$ **do**
    if $D(v) > D(u) + c(u, v)$ **then**
      $D(v) \leftarrow D(u) + c(u, v)$
      $P_v = P_v \cup \{v\}$
    end if
    if $D(v) > D(k) + c(k, v)\, and\, |D(k) + c(k, v) - D(v)| \leq \varepsilon$ **then**
      $D(v) \leftarrow D(k) + c(k, v)$
      $P_v = P_v \cup \{k\}$
    end if
  end for
end while

B.3.1 Proof

We will now proove the correctness of the MRPC algorithm. This proof is heavily based on Dijkstra's algorithms own correctness proof and the Optimality Principle.

- **Proposition -**
  The algorithm terminates with with:

$$\forall v \in V, \delta(S, v) = D(v) < \delta(S, v) + \varepsilon,$$

where $\delta(S, v)$ is the shortest path between S and v.

- **Case I:**

$$\delta(S, v) \leq D(v)$$

This is given directly by proof of Dijkstra'a algoritm stating that the shortest distance between S and v is the one recorded at v.

- **Case II:**

$$D(v) < \delta(S, v) + \varepsilon$$

We only consider an arc if:

$$D(v) > D(u) + c(u, v) \, and \, |D(u) + c(u, v) - D(v)| \leq \varepsilon$$

We know: $\delta(S, v) = D(u) + c(u, v)$ because of the optimality principle and Dijkstra's algorithm.

From the equations above we have that:

$$|\delta(S, v) - D(v)| \leq \varepsilon$$

This gives us three possible cases detailed below.

1. $\delta(S, v) - D(v) = 0$
   This case is covered by Case I.
2. $\delta(S, v) - D(v) > 0$
   In this case we obtain that:

$$\delta(S, v) - \varepsilon \leq D(v)$$

   Which is not possible, because we know from Dijkstra that $\delta(S, v)$ is the shortest path.
3. $\delta(S, v) - D(v) < 0$
   Here we have:

$$D(v) < \delta(S, v) + \varepsilon$$

   Which is indeed what we wanted.

B.4 MultiRoute Protocol Family

B.4.1 The Classical - MultiRoute

Algorithm 3 The Classical Algorithm

Require: The local network identifier $locNet$, the destination network $dstNet$, the sorted set (from shortest to longest path) of available ports, P, for this destination. The set of routing masks, RM associated to each possible next hop and the local routing mask LRM. The latest update vectors for each possible next hop and the local router, UV and LUV respectively.

Ensure: The output port, R, associated with the path containing lowest congestion or the path along the shortest path.

$LocalCongestion = ObtainCongestionStatus(LUV, LRM, dstnet)$
$(max, port) = (0, ports[0])$
if LocalCongestion contains all False or LocalCongestion contains all True **then**
  for each $p \in P$ **do**
    $RemoteCongestion = ObtainCongestionStatus(UV_p, RM_p, dstnet)$
    if RemoteCongestion contains all False **then**
      return p
    else
      if $max < RemoteCongestionStatus.getNumFalse()$ **then**
        (max, port) = (RemoteCongestionStatus.getNumFalse(), port)
      end if
    end if
  end for
else
  for each p associated to an uncongested link **do**
    $RemoteCongestion = ObtainCongestionStatus(UV_p, RM_p, dstnet)$
    if RemoteCongestion contains all False **then**
      return p
    else
      if $max < RemoteCongestionStatus.getNumFalse()$ **then**
        (max, port) = (RemoteCongestionStatus.getNumFalse(), port)
      end if
    end if
  end for
end if
return port

ObtainCongestionStatus compares the local update vector with the routing

mask, and using the destination network id returns the bitset of congestion indications which matches the order of the set of ports P.

A link is considered uncongested when the corresponding entry in *Local-Congestion* is False.

B.4.2 The Classifier - StepRoute

Algorithm 4 The StepRoute Algorithm

Require: The destination network $dstNet$, the sorted set (from shortest to longest path) of available ports, P, for this destination. The set of routing masks, RM associated to each possible next hop and the local routing mask LRM. The latest update vectors for each possible next hop and the local router, UV and LUV respectively.

Ensure: The output port, R, associated with the path containing lowest congestion or the path along the shortest path.

$LocalCongestion = ObtainCongestionStatus(LUV, LRM, dstnet)$
$(max, port) = (0, ports[0])$
if LocalCongestion contains all False or LocalCongestion contains all True **then**
  for each $p \in P$ **do**
    $RemoteCongestion = ObtainCongestionStatus(UV_p, RM_p, dstnet)$
    if RemoteCongestion contains all False **then**
      return p
    else
      if $class > RemoteCongestionStatus.getCongestionClass(p)$ **then**
        (class, port) = (RemoteCongestionStatus.getCongestionClass(), port)
      end if
    end if
  end for
  return port
else
  for each $p \in P$ **do**
    $localCongestionClass = LocalCongestion.getCongestionClass(p)$
    $RemoteCongestion = ObtainCongestionStatus(UV_p, RM_p, dstnet)$
    $remoteCongestionClass = RemoteCongestion.getCongestionClass(p)$
    $overallCongestion = localCongestionClass + remoteCongestioClass$
    if $max > overallCongestion$ **then**
      (max, port) = (overallCongestion, port)
    end if
  end for
  return port
end if

ObtainCongestionStatus compares the local update vector with the routing mask, and using the destination network id returns the bitset of congestion

indications which matches the order of the set of ports P.

getCongestionClass(p) returns a value representing the congestion class for the link associated to the port p.

B.4.3 The Know-it-all - PathRoute

Algorithm 5 The PathRoute Algorithm

Require: The destination network $dstNet$, the sorted set (from shortest to longest path) of available ports, P, for this destination. The RM for the local routing mask LRM. The latest update vectors for the local router, LUV.

Ensure: The output port, R, associated with the path containing lowest congestion or the path along the shortest path.

{**ObtainCongestionStatus compares the local update vector with the routing mask, and using the destination network id returns the bitset of congestion indications which matches the order of the set of ports** P. **In this case, we only obtain the local congestion as it will contain the congestion for the entire path.**}

$(max, port) = (MAXINT, ports[0])$
congestion = ObtainCongestionStatus(LUV, LRM, dstnet)
for each $p \in P$ **do**
  $congOnPath = congestion.getUncongestedLinksOnPathStartingAt(p)$
  if $congOnPath = Distanceto(dstnet)$ **then**
    return p
  end if
  if $max > congOnPath$ **then**
    $(max, port) = (congOnPath, port)$
  end if
end for
return port

References

[Abr70] Norman Abramson. The aloha system: another alternative for computer communications. In *AFIPS '70 (Fall): Proceedings of the November 17-19, 1970, fall joint computer conference*, pages 281–285, New York, NY, USA, 1970. ACM.

[ACE+02] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and principles of internet traffic engineering, 2002.

[AGLAB94] Bob Albrightson, J.J. Garcia-Luna-Aceves, and Joanne Boyle. Eigrp - a fast routing protocol based on distance vectors. In *Proc. Networld/Interop*, 1994.

[ALL00] Sanjeewa Athuraliya, Steven H. Low, and David E. Lapsley. Random early marking. In *QofIS '00: Proceedings of the First COST 263 International Workshop on Quality of Future Internet Services*, pages 43–54, London, UK, 2000. Springer-Verlag.

[APS99] M. Allman, V. Paxson, and W. Stevens. RFC 2581: TCP congestion control, 1999.

[ASCSA07] A. Al-Shabibi, Meirosu C., Stancu S., and Tupurov A. Yatg: Yet another traffic grapher. https://edms.cern.ch/document/839606/1, mar. 2007.

[ASM10] A. Al-Shabibi and B. Martin. Multiroute - a congestion-aware multipath routing protocol. In *High Performance Switching and Routing (HPSR), 2010 International Conference on*, pages 88–93, jun. 2010.

[ASM11] A. Al-Shabibi and B. Martin. Steproute - a multiroute variant based on congestion intervals. In *Tenth International Conference on Networking (ICN 2011), 23-28 January, 2011, The Netherlands Antilles*. IEEE Computer Society, 2011.

[Ass] InfiniBand Trade Association. Infiniband architecture specification volume 1, release 1.2.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[BEZ92] Saewoong Bahk and Magda El Zarki. Dynamic multi-path routing and how it compares with other dynamic routing algorithms for high speed wide area network. *SIGCOMM Comput. Commun. Rev.*, 22(4):53–64, 1992.

[BG92] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, second edition, 1992.

[BMG00] Yoram Bernet, Yoram Bernet Microsoft, and S. Giordano. The complementary roles of rsvp and differentiated services in the full-service qos network. 2000.

[BMK88] D. R. Boggs, J. C. Mogul, and C. A. Kent. Measured capacity of an ethernet: myths and reality. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 222–234, New York, NY, USA, 1988. ACM.

[Bos92] CA) Bosack, Leonard (Atherton. Method and apparatus for routing communications among computer networks. http://www.freepatentsonline.com/5088032.html, February 1992.

[Bur56] Paul J. Burke. The Output of a Queuing System. *OPERATIONS RESEARCH*, 4(6):699–704, 1956.

[Ceg75] T. Cegrell. A routing procedure for the tidas message-switching network. *Communications, IEEE Transactions on*, 23(6):575 – 585, jun 1975.

[CFSD90] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. Simple network management protocol (snmp), 1990.

[CG74] D.G. Cantor and M. Gerla. Optimal routing in a packet-switched computer network. *IEEE Transactions on Computers*, 23:1062–1069, 1974.

[CI05] Vinton G. Cerf and Robert E. Icahn. A protocol for packet network intercommunication. *SIGCOMM Comput. Commun. Rev.*, 35(2):71–82, 2005.

[Col98] R. Coltun. RFC 2370: The OSPF Opaque LSA Option. http://www.ietf.org/rfc/rfc2370.txt, July 1998. Obsoleted by RFC 5250, updated by RFC 3630.

[CSLM05] M. Ciobotaru, S. Stancu, M. LeVine, and B. Martin. Getb, a gigabit ethernet application platform: its use in the atlas tdaq network. page 6 pp., jun. 2005.

[DD01] Swades De and Sajal K. Das. Dynamic multipath routing (dmpr): An approach to improve resource utilization in networks for real-time traffic. In *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 23, Washington, DC, USA, 2001. IEEE Computer Society.

[DDT01] S. De, S.K. Das, and O. Tonguz. Dynamic multipath routing in networks and switches carrying connection-oriented traffic. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 10, pages 3130 –3134 vol.10, 2001.

[Dij59] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DS80] Edsger W. Dijkstra and Carel S. Scholten. Termination detection for diffusing computations. *Inf. Process. Lett.*, pages 1–4, 1980.

[Emu] EMULAB. http://www.emulab.net.

[Erl17] A. K. Erlang. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *P.O.E.E Journal*, 10:189, 1917.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

[FLS+06] J. Flich, P. Lpez, J. Sancho, A. Robles, and J. Duato. Improving infiniband routing through multiple virtual networks. In Hans Zima, Kazuki Joe, Mitsuhisa Sato, Yoshiki Seo, and Masaaki Shimasaki, editors, *High Performance Computing*, volume 2327 of *Lecture Notes in Computer Science*, pages 363–368. Springer Berlin / Heidelberg, 2006.

[Fra98] H. Frazier. The 802.3z gigabit ethernet standard. *Network, IEEE*, 12(3):6 –7, may/jun 1998.

[FT00] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing ospf weights. In *INFOCOM*, pages 519–528, 2000.

[Gal77] R. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications*, 25(1):73–85, 1977.

[GEN] Global Environment for Network Innovations. http://www.freepatentsonline.com/5088032.html.

[GFR+02] M. Gmez, J. Flich, A. Robles, P. Lpez, and J. Duato. Evaluation of routing algorithms for infiniband networks. In Burkhard Monien and Rainer Feldmann, editors, *Euro-Par 2002 Parallel Processing*, volume 2400 of *Lecture Notes in Computer Science*, pages 159–162. Springer Berlin / Heidelberg, 2002.

[GKP+08] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *Computer Communication Review*, 38(3):105–110, 2008.

[GLA93] J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Trans. Netw.*, 1(1):130–141, 1993.

[GLAM97] J. J. Garcia-Luna-Aceves and Shree Murthy. A path-finding algorithm for loop-free routing. *IEEE/ACM Trans. Netw.*, 5(1):148–160, 1997.

[GLAVZ99] J.J. Garcia-Luna-Aceves, S. Vutukury, and W.T. Zaumen. A practical approach to minimizing delays in internet routing. volume 1, pages 479 –483 vol.1, 1999.

[GZRR03] I. Gojmerac, T. Ziegler, F. Ricciato, and P. Reichl. Adaptive multipath routing for dynamic traffic engineering. volume 6, pages 3058 – 3062 vol.6, dec. 2003.

[Hec97] David Heckerman. Bayesian networks for data mining. *Data Min. Knowl. Discov.*, 1:79–119, January 1997.

[Hed88] C. L. Hedrick. RFC 1058: Routing information protocol, June 1988.

[HHK03] Mark Handley, Orion Hodson, and Eddie Kohler. Xorp: an open platform for network research. *SIGCOMM Comput. Commun. Rev.*, 33(1):53–57, 2003.

[Hin91] R.M. Hinden. Internet Engineering Task Force Internet Routing Protocol Standardization Criteria. RFC 1264 (Historic), October 1991. Obsoleted by RFC 4794.

[Hop00] C. Hopps. Analysis of an equal-cost multi-path algorithm, 2000.

[Hui95] Christian Huitema. *Routing in the Internet*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[IEE85] Ieee standards for local area networks: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications. *ANSI/IEEE Std 802.3-1985*, 1985.

[IEE89] Ieee standards for local area networks: Token ring access method and physical layer specifications. *IEEE Std 802.5-1989*, 1989.

[IEE90] IEEE. *802.3i-1990 IEEE Supplement to Carrier Sense Multiple Access with Collision Detection CSMA/CD Access Method and Physical Layer Specifications: System Considerations for Multisegment 10 Mb/s Baseband Networks (Section 13) and Twisted-Pair Medium Attachment Unit (MAU) and Baseband Medium, Type 10BASE-T (Section 14)*. 1990. IEEE product number SH13763.

[IEE91] Ieee standards for local and metropolitan area networks: Media access control (mac) bridges. *IEEE Std 802.1D-1990*, page 1, 1991.

[IEE93] Ieee standards for local and metropolitan area networks: Supplement to carrier sense multiple access with collision detection (cswcd) access method and physical layer specifications fiber optic active and passive star-based segments, type 10base-f (sections 15-18). *ANSI/IEEE Std 802.3j-1993*, page 1, 1993.

[IEE95] Ieee standards for local and metropolitan area networks: Supplement to carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications media access control (mac) parameters, physical layer, medium attachment units, and repeater for 100 mb/s operation, type 100base-t (clauses 21-30). *IEEE Std 802.3u-1995 (Supplement to ISO/IEC 8802-3: 1993; ANSI/IEEE Std 802.3, 1993 Edition)*, pages 0 – 398, 1995.

[JMG93] Alain Jean-Marie and Levent Gün. Parallel queues with resequencing. *J. ACM*, 40(5):1188–1208, 1993.

[Jr.56] L.R. Ford Jr. Network flow theory. Paper P-923, The RAND Corperation, Santa Moncia, California, August 1956.

[Kin93] J. F. C. Kingman. *Poisson Processes (Oxford Studies in Probability)*. Oxford University Press, USA, January 1993.

[Kle72] Leonard Kleinrock. *Communication nets; stochastic message flow and delay*. Dover Publications, Incorporated, 1972.

[Kle75] L. Kleinrock. *Queueing Systems, Volume 1: Theory*. Wiley, 1975.

[KM95] Christopher A. Kent and Jeffrey C. Mogul. Fragmentation considered harmful. *SIGCOMM Comput. Commun. Rev.*, 25:75–87, January 1995.

[KZ89] A. Khanna and J. Zinky. The revised arpanet routing metric. *SIGCOMM Comput. Commun. Rev.*, 19(4):45–56, 1989.

[LC01] Youngseok Lee and Yanghee Choi. An adaptive flow-level load control scheme for multipath forwarding. In *ICN '01: Proceedings of the First International Conference on Networking-Part 1*, pages 771–779, London, UK, 2001. Springer-Verlag.

[Lit61] John D. C. Little. A proof for the queuing formula: $l = \lambda w$. *Operations Research*, 9(3):383–387, 1961.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[MKJK99] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The click modular router. *SIGOPS Oper. Syst. Rev.*, 33(5):217–231, 1999.

[Moy98] J. Moy. Ospf version 2, 1998.

[MRR79] John M. McQuillan, Ira Richer, and Eric C. Rosen. An overview of the new routing algorithm for the arpanet. In *SIGCOMM '79: Proceedings of the sixth symposium on Data communications*, pages 63–68, New York, NY, USA, 1979. ACM.

[MW77] John M. McQuillan and D.C. Walden. The arpanet design decisions,. In *Computer Networksi Vol. 1, No.5,*, 1977.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers, 1998.

[Neta] Netbooting a Linux workstation. http://http://aput.net/~jheiss/netboot_linux/.

[NETb] NetFPGA: Programmable Networking Hardware. http://www.netfpga.org.

[NST99] P. Narvaez, K.-Y. Siu, and H.-Y Tzeng. Efficient algorithms for multi-path link-state routing. In *in Proceedings of ISCOM*, 1999.

[Pig20] A. C. Pigou. *The economics of welfare*. Macmillan, London :, 1920.

[Pos81] Jon Postel. Rfc 793. http://www.ietf.org/rfc/rfc793.txt, September 1981.

[Pyt] The Python Programming Language. http://www.python.org.

[RFC95] Requirements for ip version 4 routers, 1995.

[RFC04] Recommendations for interoperable ip networks using intermediate system to intermediate system (is-is), 2004.

[Rou05] Tim Roughgarden. *Selfish Routing and the Price of Anarchy*. The MIT Press, 2005.

[Spu00] Charles E. Spurgeon. *Ethernet: the definitive guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.

[SRS99] Anees Shaikh, Jennifer Rexford, and Kang G. Shin. Load-sensitive routing of long-lived ip flows. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 215–226, New York, NY, USA, 1999. ACM.

[Sta] StarBed. http://www.starbed.org.

[Tan03] Andrew S. Tanenbaum. *Computer Networks*. Pearson, Upper Saddle River, NJ, 4. edition, 2003.

[TCD+07] Jonathan S. Turner, Patrick Crowley, John DeHart, Amy Freestone, Brandon Heller, Fred Kuhns, Sailesh Kumar, John Lockwood, Jing Lu, Michael Wilson, Charles Wiseman, and David Zar. Supercharging planetlab: a high performance, multi-application, overlay network platform. *SIGCOMM Comput. Commun. Rev.*, 37(4):85–96, 2007.

[TH00] D. Thaler and C. Hopps. Multipath issues in unicast and multicast next-hop selection, 2000.

[TR98] D.G. Thaler and C.V. Ravishankar. Using name-based mappings to increase hit rates. *Networking, IEEE/ACM Transactions on*, 6(1):1 –14, feb 1998.

[VGLA99a] S. Vutukury and J.J. Garcia-Luna-Aceves. An algorithm for multipath computation using distance-vectors with predecessor information. pages 534 –539, 1999.

[VGLA99b] S. Vutukury and J.J. Garcia-Luna-Aceves. A distributed algorithm for multipath computation. volume 3, pages 1689 –1693 vol.3, 1999.

[VGLA99c] Srinivas Vutukury and J. J. Garcia-Luna-Aceves. A simple approximation to minimum-delay routing. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 227–238, New York, NY, USA, 1999. ACM.

[VGLA00] S. Vutukury and J.J. Garcia-Luna-Aceves. A traffic engineering approach based on minimum-delay routing. pages 42 –47, 2000.

[VGLA01] S. Vutukury and J.J. Garcia-Luna-Aceves. Mdva: a distance-vector multipath routing protocol. volume 1, pages 557 –564 vol.1, 2001.

[Vil99] Curtis Villamizar. OSPF Optimized Multipath. http://tools.ietf.org/html/draft-ietf-ospf-omp-02, February 1999.

[WC90] Z. Wang and J. Crowcroft. Shortest path first with emergency exits. *SIGCOMM Comput. Commun. Rev.*, 20(4):166–176, 1990.

[WLL04] M. Wang, B. Li, and Z. Li. sflow: towards resource-efficient and agile service federation in service overlay networks. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 628–635, 2004.

[WWZ01] Yufei Wang, Zheng Wang, and Leah Zhang. Internet traffic engineering without full mesh overlaying. In *INFOCOM*, pages 565–571, 2001.

[XDla97] Zhengyu Xu, Sa Dai, and J. J. Garcia luna aceves. A more efficient distance vector routing algorithm. In *in Proceedings of IEEE MILCOM*, page 97, 1997.

[ZGLA98] W.T. Zaumen and J.J. Garcia-Luna-Aceves. Loop-free multipath routing using generalized diffusing computations. volume 3, pages 1408 –1417 vol.3, mar-2 apr 1998.