Budapest University of Technology and Economics
Department of Telecommunications and Telematics

Design and Performance Analysis of Routing Algorithms in Data Networks

Balázs Szviatovszki

Ph.D. Dissertation

Advisors:

Dr. András Faragó
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas

Dr. Gyula Csopaki
Department of Telecommunications and Telematics
Budapest University of Technology and Economics

Dr. Gyula Sallai
Department of Telecommunications and Telematics
Budapest University of Technology and Economics

Budapest, Hungary
2002

Budapesti Műszaki és Gazdaságtudományi Egyetem
Távközlési és Telematikai Tanszék

Útvonalválasztási Algoritmusok Tervezése és Teljesítményvizsgálata Adathálózatokban

Balázs Szviatovszki

Ph.D. disszertáció

Tudományos vezetők:

Dr. Faragó András
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas

Dr. Gyula Csopaki
Budapesti Műszaki és Gazdaságtudományi Egyetem
Távközlési és Telematikai Tanszék

Dr. Sallai Gyula
Budapesti Műszaki és Gazdaságtudományi Egyetem
Távközlési és Telematikai Tanszék

Budapest,
2002

Contents

A bírálatok és a tézisfüzet a BME Villamosmérnöki és Informatikai Kar Dékáni Hivatalában megtekinthetők.

Acknowledgements

I wish to thank Dr. Miklós Boda, head of Ericsson Research and Development Hungary as well as Hans Eriksson, head of Traffic Analysis and Network Performance Laboratory at Ericsson Hungary for their continuous support and encouragement.

I would also like to thank Dr. Tamás Henk, head of High Speed Networks Laboratory at the Technical University of Budapest for his valuable comments and all his efforts to support me as a Ph.D. student.

I am also very grateful to my advisors, Dr. András Faragó, Dr. Gyula Csopaki and Dr. Gyula Sallai for guiding me throughout the time of my research.

And finally, special thanks to Dr. Áron Szentesi, Alpár Jüttner, Dr. András Frank, Dr. Zoltán Király and Ádám Magi who provided the closest support to the work, and also thanks to all the colleagues I worked together with at Ericsson Traffic Laboratory and High Speed Networks Laboratory, especially to Dániel Orincsay, Balázs Józsa, Ildikó Mécs, Zsolt Rajkó, Gábor Rózsa, László Németh, Gábor Privitzky, Dr. Gábor Magyar, János Harmatos, István Szabó, David Gabbitas, Dr. Tibor Cinkler and Dr. András Valkó.

List of abbreviations

AS	Autonomous System
A-INI	ATM Inter-Network Interface
AAL	ATM Adaptation Layer
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
AT&T	American Telephone & Telegraph Inc.
BGP	Border Gateway Protocol
B-ICI	Broadband Inter-Carrier Interface
CAC	Connection Admission Control
CBP	Call Blocking Probability
CBR	Constant Bit Rate
CNN	Cable News Network
CPU	Central Processing Unit
CR-LDP	Label Distribution Protocol for Constrain-based Routed LSPs
CSPF	Constrained Shortest Path First
C&W	Cable & Wireless Inc.
DAR	Dynamic-Alternative Routing
DS	Digital Signal
DSCP	Differentiated Services Code Point
ECMP	Equal Cost MultiPath
EDR	Event Dependent Routing
FIFO	First–In–First–Out
FSM	Finite State Machine
GMPLS	Generalized MultiProtocol Label Switching
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
ILP	Integer Linear Programming
IP	Internet Protocol
ISIS	Intermediate System to Intermediate System
ISP	Internet Service Provider
IUT	Implementation Under Test

MAM	Maximum Allocation Multiplier
MCF	MultiCommodity Flow
MPLS	MultiProtocol Label Switching
LAN	Local Area Network
LDP	Label Distribution Protocol
LER	Label Edge Router
LP	Linear Programming
LSA	Link-State Advertisement
LSP	Label Switched Path
OC	Optical Carrier
OMP	OSPF Optimized Multipath
OSPF	Open Shortest Path First
PNNI	Private Network–to–Network Interface
POP	Point Of Presence
PSTN	Public Switched Telephone Network
PTSE	PNNI Topology State Element
QoS	Quality of Service
RB-CSPF	Residual Bandwidth based Constrained Shortest Path First
RCC	Routing Control Channel
RFC	Request For Comment
RIP	Routing Information Protocol
RSVP	Resource ReserVation Protocol
RSVP-TE	Resource ReserVation Protocol Traffic Engineering extensions
SDH	Synchronous Digital Hierarchy
SPVC	Soft Permanent Virtual Connection
SPF	Shortest Path First
SDR	State Dependent Routing
TDR	Time Dependent Routing
TE	Traffic Engineering
TE-DB	Traffic Engineering Database
VBR	Variable Bit Rate
VP	Virtual Path
VPC	Virtual Path Connection
VPN	Virtual Private Network
WAN	Wide Area Network
WSPF	Widest Shortest Path First
WWW	World Wide Web

Chapter 1

Introduction

The recent trend of moving from dedicated telephone and data networks to multiservice backbone networks imposes new requirements on Internet Protocol (IP) based network architectures. Many *service features* of PSTN have to be supported on top of IP that was designed for a different purpose. The idea of sharing resources between flows generated by different applications, despite promising lower cost networks, also exposes difficult technological issues. Various quality of service (QoS) and grade of service (GoS) requirements must be taken into account when designing e.g., scheduling algorithms for queues, restoration scenarios or routing algorithms.

The second greatest challenge of multiservice networks comes from the fact that until recently routers inside an IP network did not have very high availability. Instead the network was designed to provide restoration in a distributed fashion. Nowadays businesses running on top of IP require very high service availability. In certain situations the time it takes for a re-route message to proceed to the head-end router is larger then the required restoration time. Therefore, router vendors are aiming to build devices with availability requirements almost as high as PSTN switches. It follows that there are an increasing number of algorithms implemented inside the routers. In order to provide uninterrupted service operation, the importance of *network control* has increased as well.

Ensuring service quality and efficient use of network resources is vital for Internet Service Providers (ISPs) in order to generate *operator revenue*. Proper capacity planning and traffic engineering with the help of routing optimization can make one network more economical than another. In this dissertation we address some of the research and design issues involved in building stable, optimally provisioned IP based backbone networks.

1.1 Research overview

The basic building block of the internet is an Autonomous System (AS). An AS is managed and administered by a single authority or organization. Typically an AS uses a single routing protocol within its boundaries. There is a difference between what is propagated inside an AS (domain) and outside of it. The operational requirements for routing protocols differ significantly, depending on the targeted environment. Between domains, political and economical issues play the most important role, while in intradomain routing protocols path optimality is one of the highest priorities. The flexibility of the AS concept and the well defined interface between inter- and intradomain protocols enable organizations (e.g., Internet Service Providers, ISPs) to choose routing and traffic engineering (TE) solutions for their network, irrespective of what other organizations use in their network.

This dissertation concentrates on a single AS and therefore on intradomain routing protocols. We investigate performance optimization problems, seek new routing algorithms and evaluate them in order to analyze their benefits and drawbacks compared to other proposed methods in the literature. The two most important concepts of network operation that are relevant for this work are *traffic engineering methods*, used to provide optimal network utilization and *routing algorithms* ensuring stable network operation. The primary area of investigation is link-state routing enabled Multiprotocol Label Switching networks. The study concentrates on the advantage of MPLS in replacing ATM in multiservice backbone networks. When limiting investigation to traffic engineering and routing optimization in MPLS networks, there are still numerous research topics. Therefore the scope has to be be narrowed further. In the remainder of this introductory chapter there follows an overview of research areas upon which this work focuses. Research issues are not discussed in great detail, but rather the most important problems in each area are introduced. Subsequently Chapter 2 draws a more complete picture of the routing and traffic engineering background, helping to properly position the problems discussed in this dissertation. Chapter 2 is followed by the discussion of each thesis in separate chapters, with a short introduction and review of past work for each topic.

1.1.1 Bandwidth constrained path computation methods

Optimally provisioned networks have enough capacity on the shortest path for all demands that arise between source and destination node pairs. However, internet traffic is not constant, moreover, capacity can not be added to every part of the network. Therefore, it may be desirable to route a demand on longer paths than the topologically shortest path (or the shortest path based on the current link weight settings).

In the case of MPLS there is the possibility to assign bandwidth requirement

to a traffic demand and reserve resources in the network for e.g., a VPN customer. Routing of flows with bandwidth requirement differs from traditional shortest path routing used in IP networks. In order to compute the explicit path for a flow, the current state of the network should be known. Depending on where path computation takes place, local and global algorithms are proposed for MPLS [1]. Local algorithms provide efficient and automated service provisioning implemented inside routers. However, distributed routing decisions may result in a sub-optimal paths set. Therefore global optimization is carried out periodically to maintain a healthy network state [1], i.e. force back most of the LSPs onto the possible shortest paths by a major restructuring.

In Chapter 3 of this dissertation we propose a new approach for routing bandwidth constrained paths, in which we allow the re-routing of an already established demand when there is no other way to route the new one. The proposed algorithm is aimed to be implemented primarily in a network operation center. However, with the restriction that only paths starting from the optimizing routers can be re-routed, the algorithm can be implemented directly in edge routers themselves in a distributed fashion.

1.1.2 Restoration path computation

When resources are reserved on a particular path for e.g., a VPN customer, there are still many things to do for the service provider. One of the most important tasks when service quality is at stake, is to provide 99.99% or 99.999% availability. This requires protection or restoration mechanisms inside the network. The hottest research area concerns mesh based restoration models. It is not an easy task to compute working and restoration paths in such a way that network resource utilization stays close to optimal even after restoration. For this problem there are number of proposals in the literature [2], [3], [4].

Chapter 4 of this dissertation compares the effectiveness of distributed restoration path computation methods that use head-end pre-computation and do not use backup path sharing. Our aim is to provide statistics on primary and secondary path lengths of simple disjoint path computation methods in real-world network topologies. Moreover, we propose a path computation method that can be used when paths are precomputed, but not pre-established.

1.1.3 Path differentiation

In multiservice networks where bandwidth pipes can be reserved for different service needs, it is important to be able to differentiate between the flows. At packet level diffserv is used to limit access to e.g., queuing resources. At path setup, the priorities defined in MPLS can be used to ensure that certain applications get the best quality path in the network (in terms of path length, thus transmission

delay). By using different priority levels, even if best effort flows are routed first, subsequently established VoIP flows can be given the chance to get the best quality paths. To enable this, preemption procedures are proposed in MPLS networks. Path computation taking into account the eight priority levels is again such an area, where little previous research has been carried out.

In Chapter 5 of this dissertation, we study the effect of distributed path selection on the dynamics of LSP preemption in Multiprotocol Label Switching (MPLS) networks. We propose new distributed bandwidth constrained path computation algorithms for *minimizing preemption of lower priority LSPs*, without requiring any enhancements to the recently proposed link-state parameters.

1.1.4 Performance evaluation of routing algorithms

All the above discussed protocol or algorithm enhancements do not provide any benefit for the service providers or equipment manufacturers if the proper operation of the algorithms can not be tested on realistic network situations before mass deployment. Simulation represents the first very important step of evaluation of different algorithms. However, algorithms implemented in routers can reveal unexpected situations in real life networks. Emulation with multiple software instances of real equipment code is one solution to test development. However, these interconnected code-pieces are hard to run, monitor and configure to analyze the desired situation.

In Chapter 6 of this dissertation we introduce a novel method to examine the behavior of a device implementing a link-state routing protocol and its path selection components. Although three of the four main chapters of the dissertation deal with MPLS networks, this last one concentrates on ATM networks running the Private Network-Network Interface protocol. Since the basic principle of both MPLS and PNNI networks is to distribute resource reservation information with the help of a link-state routing protocol, the method presented in this chapter is also applicable to MPLS networks using the OSPF routing protocol. Thus this last chapter provides a method for testing routing algorithm implementations of real products. The concept outlined in Chapter 6 eliminates the need for building large, expensive and hardly configurable test networks made of real equipment in order to inspect thoroughly the behavior of an implementation under test. By extending a simulator with functions to interconnect with real equipment, the method enables the creation of various test scenarios by simple editing of a single configuration file. This concept idea fits both MPLS and ATM environments, despite the fact that the specific solution discussed in the dissertation was developed and tested for ATM network equipment running the PNNI protocol only.

Chapter 2

Routing and traffic engineering in data backbone networks

2.1 Introduction

This chapter aims to provide a snapshot of the networking architectures, protocol choices and algorithmic solutions of contemporary data backbone networks. Based on the findings of a recent study, Section 2.2 presents how Autonomous Systems in the Internet are interconnected. Section 2.3 investigates the traffic patterns of ASs at different levels of the Internet hierarchy, while Section 2.4 and Section 2.5 provide an overview of traffic engineering methods used inside an AS.

2.2 AS hierarchy of the Internet

In order to investigate routing and traffic engineering issues in a network operated by a single organization, it is important to know how individual networks are interconnected to form the 'network of networks': the Internet. In the previous chapter the concepts of ASs, inter- and intradomain routing was already introduced. In this section, we draw a picture of the internet at the AS level based on the findings of a recent study by Subramanian, Agarwal, Rexford and Katz [5]. In this work to obtain the AS graph of the internet, interdomain routing table data was gathered from multiple vantage points by using the Border Gateway Protocol (BGP) [6]. The study identified more then 10.000 ASs and as the main part of the work it classified ASs into 5 levels based on the observed interconnection patterns among them (e.g., customer-provider, peer-peer).

Level 0: At the highest level, referred to as the 'Dense Core', there are 20 ASs. These ISP networks are densely connected (in-degree and out-degree at least $N/2$, where N is the number of ASs at this level). Fifteen ASs out of the

twenty almost form a clique. The dense core has direct connections to about 7000 customers, or small regional ISPs, which means that they provide transport for a huge number of ASs out of the whole AS space.

Level 1: This level is identified as the 'Transit Core'. The 162 ASs at this level are mostly interconnected with each other, or with 'level 0' networks. Half of the nodes have no relationship to lower levels.

Level 2: Authors of [5] found multiple disconnected groups of ASs, having less then 100 links to higher levels. They named this as the 'Outer Core'. There are 675 ASs at this level.

Level 3: The 'Small Regional ISP level' includes 950 ASs. This represents almost 9% of the whole AS space. These ISPs have one or more customers. At this level about 60% of the ASs have only two, three or four neighbours.

Level 4: 'Customer ASs' (8852 ASs) are at the bottom most layer of the Internet hierarchy. These are stub networks which originate and terminate traffic flows but do not carry any transit traffic. According to [5] 82.5% of the ASs in the Internet are in this group. At this level 40% of the ASs have 1 neighboring AS, 45% have two, about 10% have three neighbors, while the remaining 5% have less then 20 neighbors.

The place of an AS in the above hierarchy, moreover its peering relations and policies has significant effect on the AS path statistics. Uhlig and Bonaventure [7] determined AS path length statistics of a dial-up ISP. The study found that the average AS path length is around 4.2. Since a dial-up ISP usually does not provide transit services, it is most probably situated at the 3^{rd} or 4^{th} level of the AS hierarchy. We could also suppose that the investigated AS has a direct connection to either a level-0 or level-1 AS, since from the detailed AS path length statistics it turns out that by using routes having no more then three AS hops, 80% of the whole IP address space is reachable. Traffic statistics also support the above conjecture, since 64% of the traffic is carried by an AS that is at only one AS hop distance from the investigated AS. The authors of [7] deduce from the results that a very small number of BGP peers provide connectivity for almost all the interdomain traffic of the investigated dial-up ISP.

2.3 Traffic patterns of an AS

In this section we investigate traffic related issues, namely the effect of applications on the special distribution of traffic in the internet, measurement methodology a network operator can use to determine the traffic flows of its networks, and finally time effects such as periodic and unexpected traffic volume changes.

2.3.1 Role of applications

The dominant part of traffic in current IP based backbone networks is generated by client-server interactions, especially web-based applications. Therefore the location of major servers determines largely traffic patterns within the internet [1]. Today the largest sites are located in the US, which makes web traffic US centered. In case of special events with world-wide interest, huge sites (e.g., CNN [8] or BBC News [9]) get so high number of download requests that makes the central data storage concept inappropriate. In such cases traditional broadcast media, namely television and radio works much more efficiently. However, interactivity of web-based services represents additional value. To cope with the huge amount of web downloads a number of content distribution techniques have been proposed for web-sites. Techniques based on replicated information servers and dynamic load balancing methods improve web-server performance by distributing load among different sites. Akamai [10] and Digital Island [11] are two companies that are running a globally distributed network and provide services like content transformation and dynamic content delivery. These techniques are useful for multinational company home pages, media companies and software distribution sites. Other types of web-site will most probably continue to function in the same way as they do today.

New applications and their media servers —if they become dominant— might represent a major change in internet traffic patterns. Real-time voice traffic carried between media gateways of a 3G mobile network will result in such a traffic pattern that is much more concentrated on a geographic location, in an extreme case on only a few ASs. This is true for any kind of new application that stresses the importance of person-to-person communication, because people tend to communicate through electronic media with those whom they also meet at least few times a month.

2.3.2 Modelling of traffic flows

It is of prime importance for network operators to know how traffic flows in their networks. As we saw, the Internet is structured based on the AS concept, where an organization has control over its own domain. Measurement and modelling of traffic within an AS is essential for proper link dimensioning and network upgrade scheduling. For these tasks, IP traffic should not be considered as thousands of individual micro-flows (same source-destination address and port numbers), but as a few aggregate flows. Aggregation can be done in various ways; most commonly those flows are treated together that are routed towards the same edge router inside the actual AS. If the differentiated services concept is used, an aggregated flow can be further divided based on the used diffserv code point (DSCP). For traffic engineering and routing purposes aggregated flows are characterized by the

ingress routers where the flow enters the AS, moreover with some description of the bandwidth requirement (sometimes time dependent). In case of *point-to-point* modelling, one egress is enough to describe the exit point of the flow from the AS. In case of *point-to-multipoint* modelling, more egress links can be assigned for a traffic demand. We elaborate on this in the next sections.

The choice of edge router —or more specifically the exit link from an AS— depends on different configuration settings. BGP configuration options, as well as OSPF settings affect the exit point of a traffic flow from an AS. In practice, policies represent a very important part of BGP: they specify how routes received from a peer will be accepted, selected and redistributed towards other BGP peers. By setting 'local preference' attributes, manipulating the 'AS path length' or configuring 'multi-exit discriminator' values, the path of a traffic flow can be influenced. Similarly when two routers are identically desirable exit points, OSPF settings (e.g., the internal path length from the source router to the BGP routers) affect which edge router will be taken as an exit point from an AS.

Bhattacharyya et. al. [12] measured POP and access-level traffic dynamics at a point of presence (POP) of a large ISP (Sprint). They conducted measurements at a single POP location and determined point-to-point traffic demands destined to any other POPs, which represents one row of a POP-to-POP traffic matrix. They determined the egress POP for every packet by downloading IBGP routing tables from the monitored POP. This study intended to measure both internal flows and outbound flows. With the help of threir measurements the authors were able to deduce the geographic spread of traffic among POPs and the time-of-day behaviour of flows. Among their findings was the discovery that access links of medium size POPs do not distribute traffic uniformly across egress POPs.

Feldman et. al. [13] proposes to model traffic not as point-to-point flows, but as point-to-multipoint flows that can be described by the traffic source and the set of links the flow can use as an exit. This model is needed when one wants to keep the same model when manipulating internal routing settings, such as OSPF weights. They proposed an architecture to derive the traffic demand model that includes the following components: flow level measurements (e.g., with the help of NetFlow [14]), reachability information at egress links, dumping of router configuration files and forwarding tables. Since in an ISP network there are a much fewer number of peering links connecting to other providers than all the ingress links, [13] proposes to measure only at peering links, and to identify ingress links from this basic data set. The consequence of measuring at peering links is that only transit and outbound flows are measured, internal flows are unrecognizable.

When traffic routing inside the backbone network is done with the help of tunnels/virtual circuits (MPLS/ATM), it is possible to measure packet statistics for each tunnel. In another case, when an operator offers Virtual Private Network (VPN) services, the flows are explicitly known, since there is a contract for

providing fixed bandwidth connections between VPN sites. In the VPN case, conformance to the contract can be checked with the help of per tunnel statistics.

2.3.3 Changing traffic patterns

When measuring and modelling traffic demands in a network, time plays an important role. There is a daily, as well as a weekly fluctuation in traffic generated by humans behind the applications. Moreover, the amount of data carried in backbone networks generally is continuously growing which means that a trend should be predicted based on a series of past measurements. Feldman et al. [13] showed time-of-day fluctuation of domestic business, domestic consumer (residential) and international traffic. Since peak hours do not coincide for these three categories, in an international backbone daily traffic engineering is a reasonable strategy to influence the paths of traffic demands in the network. The existence of traffic demands with different peak hours is more probable in ASs residing at upper AS hierarchy levels. However, in the case of a dial-up ISP probably a single peak due to residential users can be measured, which means that the network could be engineered for the peak-hour demand.

Beside regular traffic variations there are unpredictable ones. Although a network can be well planned to carry the expected traffic load of the peak workday, it also has to provide reasonable service in case of a global overload, regional overload or after a link or node failure. Past experiences of course can be used to analyze special situations, e.g., earthquakes, concert-broadcasts on the web etc., however, since these events are sudden, automated mechanisms should be implemented in network equipment to react and carry the increased traffic volume.

2.4 Acquiring TE information

In the previous section the main task was to get an idea of how aggregated traffic flows can be characterised, modelled and measured in a network. In this section a further step is made, namely to identify what kind of information is necessary to decide what path a traffic flow should take. The traditional IP model uses distributed mechanisms and procedures for this purpose. Contemporary networks built to provide interconnection services to businesses implement increasingly centralized procedures, mostly for network operation and management. In this section we discuss distributed procedures for basic network connectivity discovery in Section 2.4.1, then in Section 2.4.2 methods to obtain network state information.

2.4.1 Network connectivity discovery

Auto-configuration support is essential for efficient management of network upgrades, moreover, during failures the methods used for auto-configuration can distribute changed connectivity information in the network for every device.

In small scale networks the simplest method for discovering which destinations are reachable in the network is the *distance-vector* method. The Routing Information Protocol (RIP) [15] is based on this principle. In RIP every router periodicly distributes its used routing tables — including the distance to every known destination — to every neighbouring router. Neighbours distill their own routing tables from those received on all of their links. The distance-vector principle is based on the distributed Bellman-Ford shortest path calculation algorithm (see e.g. [14]). Unfortunately experiences of RIP deployments have shown that this method has bad convergence properties, moreover, it may create routing loops.

Noticing the bad route-convergence time of RIP, work started in 1987 on the OSPF protocol [16] that adopts the *link-state* principle. The basic idea of link-state routing is to provide a private map of network connectivity to every router. This is achieved by distributing basic information elements about every router and link in the network. By maintaining a complete picture of interconnections, routers could compute multiple shortest paths to each destination. This feature is often referred to as Equal Cost MultiPath or shortly ECMP.

The Private Network-Network Interface protocol specification [17] developed for ATM networks around 1996 builds on the experiences with OSPF. In most of the basic principles it is the same as OSPF, though it introduces new concepts e.g., the concept of forming a single adjacency over parallel links between two neighbors. This concept appeared recently on the work-list of the OSPF working group as well [18].

Almost fifteen years have passed since the work started on development of link-state routing, and there are still new proposals to polish parts of the protocols. In the next few sections OSPF studies and proposals will be discussed.

The need for uninterrupted operation requires very fast convergence from OSPF, which can only be solved by distributed procedures. A key component of link-state protocols is the Hello protocol that checks lower layer connectivity. Over such media where carrier loss is detectable, it is possible to trigger link-down events in the protocol state machine by monitoring lower layer (e.g. SDH) alarms [19]. If this is not possible, there are proposals to enable fast link-down detection with the help of Hello messages sent more frequently then 1 second.

Alaettinoglu, Jacobson and Yu in [20] propose changes in IGP specification to reach *milli-second* IGP convergence. One of their proposals is to allow sub-second Hello intervals. They argue that the acceptable minimum interval should be determined by the links' physical constraints. To avoid stability problems, they also propose to treat link-down events (bad news) differently from link-up events

(good news) and to treat routing control packets preferentially over data packets. Basu and Riecke in [21] determined with simulation that if the Hello Interval is decreased from 10 seconds to 500ms (or 250ms), the processor utilization increased by 1% (or 2%) under normal network load. After a node failure, the utilization peak increased from 8-9% to about 15%. They say this worth the much lower convergence time (1 sec instead of 30-40 sec). In the simulation they used a detailed processor model where job processing times (e.g., LSA processing, origination and duplication time) were set based on measurements of a commercially available router. The actual processing times ranged from 10 to 1101 microseconds [21].

The effect of message loss on OSPF performance was also studied in [21]. The authors noticed that when loss increased from 2 to 5%, processor utilization increased on the average from 10 to about 30%. However, when loss further increased to 10%, utilization decreased, since due to lost Hello messages adjacencies were also lost, thus flooding of LSAs were performed to less neighbours. We could also assume that at this time parts of the network became unreachable (because of lost link-state information) which might caused smaller SPF computation times.

Choudhury, Maunder and Sapozhnikova [22] carried out another simulation evaluation of link-state protocols. They focused on the stability of routing protocols under LSA storms (large number of updates received during a short time period). They state that the primary reason for some recent network meltdowns (OSPF, and PNNI) was the consequence of LSA storms, the high delay in Hello packets and lost adjacencies caused by this. By simulating a 300 node, 800 link network they showed that when the LSA storm size was 600, topology databases remained inconsistent in the simulated network for 40 seconds. In case of a 900 LSA storm, the network entered an unstable state and secondary storms were created. To model the situation, they used an analytical approach, and determined that with default OSPF setting and node adjacency of 20 (50), the point where the network enters an unstable state is around storm size of 800 (325). When they used a 2 second Hello interval and minimum SPF calculation interval of 1 second, the unstability threshold decreased significantly. With a node adjacency of 10, 20 and 50, the critical storm size was 310, 160 and 65, respectively. Authors of [22] suggest methods to give prioritized treatment to Hello packets, LSA acknowledgement packets, and LSA packets carrying bad news. They also propose to accept any control packet over a link as an "implicit Hello" in order to keep the Hello final state machine (FSM) in link-up state.

A recent internet draft [23] summarizes a number of link-state protocol enhancements, including procedures for database backup, hitless restart [24] and protocol changes (e.g., no ack for timed out messages).

The convergence time of link-state routing protocols also depend on the optimality of the path computation algorithm implementation. Shaikh and Greenberg performed black-box OSPF measurements in [25] and measured not only the pro-

cessing times of LSA messages, but also the routing table recomputation time for three router implementations. They found a $c_1 n^2 + c_2$ relationship between the SPF calculation time and the number of nodes in the network. Marváez, Siu and Tzeng in [26] propose dynamic algorithms for shortest path tree computation which avois the total recomputation of the entire routing table. By using either these or some other dynamic algorithms, Alaettinoglu, Jacobson and Yu in [20] concluded that current $\tilde{1}00$ millisecond SPF recomputation times can be decreased to the microsecond range, which would result in 10.000 times faster SPF code in routers. This would shorten the convergence time of link-state routing protocols as well.

2.4.2 Network state discovery

It is not always enough to obtain information about the topology of the network. When a routing decision has to be made for aggregate flows having bandwidth requirements it is necessary to take into account resource utilization levels of network links as well. To guarantee the required bandwidth, the bandwidth constrained path of traffic aggregates must be determined one-by-one (per-flow routing). When the routing decision is made in a central place, network state information has to be conveyed to a single route optimization server. When distributed routing decisions are supported, all devices taking part in the computation should obtain network state information.

When network traffic is predictable it is possible to exploit historical information to tune the network by using time dependent pre-programmed routing plans and other TE control mechanisms [27]. However, this method can not guarantee that the resources will be available. Furthermore, *time-dependent routing* (TDR) methods are not adaptive, i.e., they do not perform well under unexpected events.

Real adaptive methods are the state-dependent (SDR) and event-dependent (EDR) methods, where certain procedures ensure that the current network state is used for path computation. This is important in order to handle load perturbations from regular variations, failures and different overloads (global, or regional). In *state-dependent routing methods*, together with basic topological information, resource utilization information is also distributed by link-state procedures. SDR methods use periodic or triggered flooding mechanisms (e.g., by defining significant change thresholds). *Event dependent routing methods*, on the other hand, do not distribute periodic link utilization information, but instead try out different paths in the network for routing a flow. In case a traffic aggregate can not be established on one path, crankback procedures and different learning methods are used (e.g., success to the top [27]) to find the appropriate path in the network.

Apostolopoulos, Guérin and Kamat [28] report experiments with a SDR implementation. They extended OSPF to distribute available bandwidth information about network links. Two methods are mentioned that trigger the origination of new link-state descriptors: a) a relative policy where a new update is triggered by

comparing to the last flooded value; and b) an absolute policy where link capacity is split to preconfigured classes and boundary crossing triggers the new update. To route traffic flows with bandwidth requirements they used the widest-shortest method with pre-computation [29]. The performance evaluation of the proposed methods have shown that LSA traffic associated bandwidth consumption represented only a small fraction of link bandwidth and that the increased processing cost of bandwidth constrained path pre-computation remains well within the capabilities of medium-range modern processors [28].

In all SDR methods triggering mechanisms introduce inaccuracy to the link state information. On the other hand, without them too much network resources would be used. Apostolopoulos, Guérin, Kamat and Tripathi [30] therefore studied new path computation algorithms that take into account 'safety' of link state information. If the triggering mechanism and the last flooded state information are known it is possible to determine the range of values that the actual bandwidth can take. Safety is then an assigned probability that on a link the required bandwidth will indeed be available. For example in case of the absolute triggering policy, if the requested bandwidth is smaller (larger) than the lower (upper) class boundaries of the last flooded link available bandwidth, probability 0 (probability 1) will be assigned to the link.

Shaikh, Rexford and Shin [31] also studied the impact of stale (inaccurate) link state information on routing performance. They show that out-of-date information increases connection setup failures and that it makes pruning of infeasible links less effective.

In SDR methods inaccuracy is introduced because flooding of every link utilization change would consume too much network resources. As the size of a network (especially the number of links) increases, the amount of control traffic grows as well. SDR methods usually use hierarchical network structure to limit the flooding scope and thus the amount of control traffic inside an area. However, this has a drawback: when hierarchies are introduced in the network, routing is less flexible then in a pure flat topology, since flows are constrained to go through a small number of border routers when the source and destination nodes are in different areas/groups. Therefore Ash argues [27] that by using EDR methods much larger networks can be built maintaining the same flow routing success probability.

2.5 Interdomain traffic engineering practices

The previous section concentrated on how routers in the network obtain topology and state information. These two provide a capacitated graph, on which one can route traffic flows according to the methods overviewed in Section 2.3. The problem of mapping flows on network paths is defined by RFC 2702 [32] as the basic task of traffic engineering. Depending on the method used for traffic engineering, the paths

of traffic flows will meet different criteria. For example when using OSPF weights to optimize traffic flows, we cannot influence the routing of the flows independently of each other. In this case at any point in a network, all flows having the same destination should be routed on the same path irrespective of their sources [33]. On the other hand, when per-flow routing is used, it is possible to determine the path of a flow independently of other flows. Before discussing TE proposals for the *source invariant* case (weight optimization) and for the *per-flow* case (explicit routing), in the next few sections we first elaborate on shortest path routing and load balancing in general.

Given a fixed set of demands and an uncapacitated graph, we have the simplest design problem, i.e., to determine the routing of flows that require the smallest amount of overall link capacity in the network. Obviously *shortest path routing* gives the optimal solution to this problem. Similarly, looking at a capacitated network at severe overload, it is not surprising that shortest path routing achieves the highest throughput. However life is not that simple. As we have discussed in Section 2.3.3, traffic is not constant, moreover, capacity can only be upgraded in step-wise manner. Therefore, in certain situations non-shortest-path based load balancing should be used for flow routing.

In th ecase of unknown future flow arrivals *load balancing over non-shortest paths* provides a viable alternative to simple shortest path routing. Within the usual operational region of networks when overload/blocking is low, load balancing can be used to *minimize the load of the most utilized link* without consuming too many extra resources compared to shortest path routing. This improves the performance of the network and ensures that future flow routing can be better supported than it would be with simple shortest path routing.

In the next few sections we give an overview of the traffic engineering possibilities available when OSPF is deployed in a network, and per-flow routing based TE practices achievable with ATM and MPLS. In the former method, flow routing is influenced by changing link weights. In the latter method flows are routed independently of each other on their own explicitly defined path.

2.5.1 Source invariant TE methods

The link-state principle used by e.g., OSPF and ISIS, provides a router-router interconnection graph for every router in a network. Based on this graph, every router uses it's own path computation algorithm to determine shortest paths to all possible destinations. To decide which path is shorter, OSPF uses weights that are configured for each interface (link) separately by the network administrator. In OSPFv2, multiple shortest path can be used for routing a packet towards a specific destination. This is often referred as the *equal-cost multipath* (ECMP) concept. Since basic IP routing is hop-by-hop, even though a router computes the complete path for a destination only the next-hop is saved. By always using the next hop

towards the destination the packet will eventually reach its endpoint, since every router computes its routes based on the same topology graph. In case there are more shortest paths between two routers, a packet may travel through a different shortest path then the one calculated by e.g., the first router.

When ECMP is used, among equal cost paths load can be balanced on per-destination or per-packet basis. The former uses destination address hashing and therefore preserves packet order, but it can not split load equally among the paths. Per-packet load balancing produces an exact load split, but it has the drawback of possible packet re-ordering [34].

With the help of hashing it is possible to achieve non-equal split of destination flows, e.g., a 30%-70% split among two outbound interfaces. There were proposals for doing non-equal traffic split in IP, based on an intelligent automated procedures implemented in router software. The basic idea of *OSPF optimized multipath* (OMP) concept [35] is to move traffic away from overloaded paths by changing split ratios. In order to avoid oscillation, OMP uses hold-down timers and slow changes. Simulation studies showed that OMP can provide higher throughput than traditional IP routing. Regardless of this, OMP has not been implemented in commercial router software, probably because of the high complexity, and limited benefit compared to what per-flow routing offers.

Optimizing OSPF weights [36] is another proposal to do central traffic engineering in OSPF, or in any network using link-state protocols. This method is actually used in a large operational network [37]. It is based on the simple idea of increasing the weight of overloaded links in order to move away traffic from them. The biggest advantage of weight manipulation is its simplicity. The network operator should only change one link's weight value and because of the automated topology connectivity distribution mechanisms of OSPF and the built-in shortest path first calculation of routers, as a result, all devices' routing table will be updated. Fortz and Thorup [36] proposed methods for calculating the weight system for an OSPF network —using equal cost multipath— that optimizes resource utilization. Results showed that on the investigated AT&T network model, the source-invariant routing method performed surprisingly close to the optimal solution obtainable by a per-flow routing method. The authors run simulations for synthetic network models as well. They have found that compared to simple heuristics (e.g., assigns weights inversely proportional to link capacity) by using the optimized weight system 50-110% increase in demands can be supported.

Another paper [38] proved that with optimized weights and load splitting the same network performance can be achieved as with MPLS based per-flow routing. However, in this work authors assume that traffic can be split arbitrarily among two outgoing links. However, this is not the case with current IGP protocols e.g., OSPF. Therefore this result has only theoretical importance. Another paper [33] has shown that achieved performance of source invariant (OSPF weight optimiza-

tion based) methods can be arbitrarily far from that of per-flow routing (MPLS explicit routing). As pointed out by Rexford [39], in their proof the authors of [33] use a special topology, which does not preclude that weight setting works pretty well in real-life network topologies, which seems to be the case.

2.5.2 Per-flow routing based TE methods

When an operator wants to optimize traffic routing in its network, one possibility is to decide the paths of each individual flow independently of other flows. To obtain optimal resource utilization this should be carried out in a central place of the network, where global information is available about physical resources and traffic demands. For resources the available bandwidth (capacity) is the constraint, while for demands it is the required bandwidth. Global path optimization algorithms intend to solve a multicommodity flow problem. Since MCF problems are NP-hard, either the optimum solution is determined for network sizes for which the algorithm can still finish within reasonable time (few hours), or heuristics are used to obtain a close-to-optimal solution. Liljenstolpe reports in [40] that to calculate *an optimal set of paths* (primary as well as secondary), Cable & Wireless uses an in-house developed tool running on a dedicated server. For problem instances with 30.000 paths the algorithm provides the solution in approximately 2-4 hours. Another global optimization algorithm [41] uses heuristics to cope with large problem instances in reasonable time (e.g., 1000 nodes 500.000 paths in 3 hours).

Once the optimal paths are computed, those should be matched against the operational path sets and changed traffic flows should be re-routed by a series of management actions. In the case of per-flow routing the paths of the flows can be changed one by one. Józsa et al. [42] propose algorithms to plan the exact order of re-routing in case the sequence of re-routing is important. In the case of source-invariant routing every time an OSPF weight is changed, all traffic flows are possibly affected. As far as we know, transient path changes have not been studied in the case of source-invariant routing. It is certainly solved, however, as a major operator uses this technique in an operational network [37].

Because of the complexity of the calculations, forecast granularity and large restructuring of paths in the network, to perform global optimization based TE one should consider a larger time-scale, e.g., few days or a week. In an environment where traffic flows are established dynamically it is impossible to optimize the complete path set every time a new path is set up. Therefore some automated method is needed that resides on a router and acts when the path setup is initiated. In this case, individual devices in a network should obtain network connectivity information, as well as network state information in order to enable path computation. Distributed path computation algorithms are usually referred to as constrained shortest path first or CSPF algorithms for short, since constraints associated with links (reservable bandwidth) and with traffic demands (e.g., required bandwidth,

hop-limit) are matched against each other. Such algorithms are generally simple and very fast. Although, local routing decisions may result in a degraded global network performance on the long run, intelligent *preventive* strategies [43], [44], [C2] could be used to improve LSP path selection performance and thus achieve globally near-optimal solutions.

Both CSPF and global optimization algorithms determine the path for a traffic flow independently of other flows which was not possible with standard OSPF routing. Once traffic flows are computed with either global optimization or with the help of an automated procedure residing on an edge router, flows are established on their optimized path with the help of such path setup procedures in which all the explicit hops are signalled. Intermediate devices between the source and the destination node forward the flow setup based on the next explicit hop derived from the setup message itself. In this sense traffic engineered paths are independent of the shortest paths of the underlying physical network. They represent a tunnel, for which reservation information is saved at each hop.

Depending on how the IP layer interacts with traffic engineered tunnels, two approaches are distinguished: the overlay and the integrated approach. In case of the overlay approach, IP routers see a TE-tunnel as a single hop to the next router. Frame Relay, ATM or any other technology in which virtual circuits are implemented can be used to implement an overlay solution. If an IP cloud is run on top of a traffic engineered transport network, the large number of tunnels between IP routers could be hard to operate and scale as the network grows. To overcome the scalability problem, overlay networks are often build in a hierarchical manner. However, as we mentioned already, the use of hierarchies may result in sub-optimal flow routing and resource utilization.

In an integrated approach the IP layer shares control plane information with the transport layer, therefore TE-tunnels are such shortcuts for which the intermediate hops are also distinguishable by the IP layer. This makes it possible to use TE-tunnels for only a subset of traffic flows.

Many service providers, who used to run ATM transport networks are planning or already rolling out MPLS based networks which start from a strict overlay architecture, but it allows the move towards the integrated model. This is far from the end of network evolution, however. In certain networking segments, virtual circuits and per-flow routing will become more important in the future with the increased importance of virtual private networks, optical networks, and the possibility of establishing lambda paths.

Chapter 3

Partial path optimization in MPLS networks

3.1 Introduction

This chapter concentrates on traffic engineering methods based on per-flow (explicit) routing of MPLS label switched paths (LSPs). As we have described in Section 2.5.2 of the previous chapter, in an operational MPLS network the Constrained Shortest Path First (CSPF) computations implemented in edge routers can automate the LSP setup process. However, edge routers have only local information about the network resources. These local routing decisions made in routers may result in a degraded global network performance in the long run. One solution for this is to perform periodic (not too frequent, e.g., daily or weekly) global re-optimization of LSPs with a centralized off-line network optimization tool (e.g., [J6] [45] [46]). Another approach is to make the CSPF computation itself more intelligent by implementing a kind of *preventive* strategy. Concepts used in dynamic routing methods proposed for ATM networks (see e.g., [43], [C2]) could be re-used to improve LSP path selection performance. These methods implement local strategies that result in globally near-optimal solutions. Recently [44] proposed the concept of *minimum interference routing* that is based on this idea.

The newly developed algorithm presented in this chapter is intended to be used when on-demand CSPF based LSP setup in a Label Edge Router fails. Such a failure can happen when there is not enough bandwidth, since previously established LSPs occupy network resources. This kind of situation may arise even if a preventive routing strategy is used, although at higher network utilization levels. In such failure situations, supposing that the operator wants to fit in the new LSP, a quick re-optimization of as few LSPs as possible is preferable to an immediate global re-optimization of all LSPs. A special case of the above strategy is to allow the re-configuration of only a *single LSP*. This requires the smallest amount of

management interactions, while still providing an efficient solution to overcome the problem arising from the CSPF failure. Thus, our proposed algorithm implements such a *failure triggered partial re-optimization strategy* in which we try to re-route only one LSP.

The rest of the chapter is structured as follows: in Section 3.2 we formulate the problem. Section 3.3 describes the algorithm by presenting its three main parts. We use an Integer Linear Programming based solution in the algorithm, for which an efficient solution is presented in Section 3.4. Section 3.5 presents numerical results, and finally Section 3.6 summarizes the work.

3.2 Problem statement

As outlined in the introduction, the task we address in this chapter is to select a single LSP that can be re-routed in order to be able to establish a new LSP. The exact definition of the problem is therefore the following:

- We are given all the information on the network topology and status, including the bandwidth reservations and actual paths of currently established Label Switched Paths (LSPs), (we will refer to these LSPs as *old* LSPs)
- We have a *new* LSP request from node s to node t with bandwidth demand b.
- Assume that there is no $s \longrightarrow t$ path with sufficient unreserved bandwidth for accommodating the incoming request b (otherwise the problem is trivially solved by CSPF). Our task is to release sufficient bandwidth on some 'bottleneck' links (i.e., where the unreserved bandwidth b_f is less than b) by re-routing a single LSP, so that we can set up the new LSP.

The non-trivial problem here is to find an existing LSP that can be re-routed so that after the re-routing, the new LSP can also be established. This problem is NP-hard [47], therefore we propose a heuristic algorithm for its solution.

Since we need path information for perviously established LSPs, the re-optimization can be done primarily in a network operation center rather than in the edge routers themselves. The complexity of the optimization algorithm also calls for this solution. However, theoretically, with limited scope, the algorithm can be implemented in a distributed way in Label Edge Routers, as well. The restriction in this case is that only LSPs starting from the optimizing LER can be re-routed, since the path information of only these LSPs are available.

3.3 The algorithm

The main purpose of this algorithm is to re-route an LSP that is already established, so that the new LSP can be placed into the network. Fig. 3.1 depicts the flow chart of the algorithm.

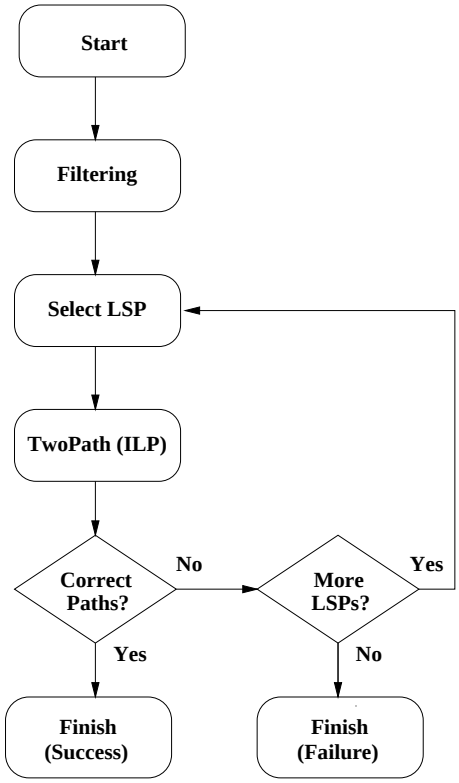


Figure 3.1: Flowchart of the optimization algorithm

The aim of the first function is to *filter* the LSPs. The simplest approach would be to try to re-route all established LSPs one by one. In small networks this is reasonable, but in larger ones it is unacceptable, because of the large number of LSPs. Consequently, it is necessary to decrease the number of examined LSPs. There can be such old LSPs that even when completely released, do not enable the establishment of the new LSP. There is no virtue in trying to re-route such LSPs.

In the second part of the algorithm we take the candidate re-routable LSPs one by one, and execute an ILP based function called *TwoPath*. This function tries to establish the new LSP and re-route the selected old LSP simultaneously. If the TwoPath function finds a solution, the optimization is complete, so the whole algorithm terminates. If TwoPath could not find any re-routable old LSP, the algorithm terminates with failure.

In the rest of this section the main parts of the algorithm are described in more detail.

3.3.1 Filtering

The aim of filtering is to reduce the number of LSPs for which the TwoPath algorithm is executed. We implemented three different filters. The order in which the filters are executed can be chosen freely. The *first filter* examines the so-called bottleneck links. A bottleneck is such a link that does not have enough available bandwidth for the new LSP. First, the filter calculates the minimal number of bottleneck links for the new LSP by Dijkstra's shortest path algorithm with the following weight function:

$$w(e) = \begin{cases} 1 & \text{if edge } e \text{ has insufficient available bandwidth} \\ & \text{for the new demand,} \\ 0 & \text{otherwise.} \end{cases}$$

Dijkstra's algorithm aims to avoid the bottleneck links, therefore the resulting shortest path will contain the minimal number of bottlenecks. Let n_{min} denote this minimum value. If a configured LSP contains less bottlenecks than n_{min}, then rerouting this LSP will surely not help the configuration of the new LSP, because at least on one link the available unreserved bandwidth will be less than the bandwidth requested by the new LSP.

The basic idea behind the *second filter* is the following. Starting in any direction from the source node towards the destination, we will bump against a bottleneck link. Those bottleneck links that are reachable from the source node directly (i.e. without passing another bottleneck link), are called first bottleneck links. The LSP to be rerouted has to use one of the first bottleneck links. This filter uses the same weight function as the one above and searches the first bottleneck links in the following manner:

1. It searches for a shortest path.

2. It searches for the first bottleneck link of this path, deletes it from the graph temporarily, and stores it into a list.

3. It repeats the first two steps until there is no path between the source and destination nodes of the LSP demand.

When all possible first bottleneck links are found, we filter out such old LSPs which do not use any of the bottleneck links of the list.

The *third filter* checks whether the unreserved bandwidth of the bottleneck link — after re-routing the selected old LSP — will be large enough to accommodate the

new demand. This filter precludes the possibility of re-routing those configured LSPs which do not allow the placement of the new demand into the network because of their size.

3.3.2 Select LSP

The aim of the *Select LSP* function is to choose a single old LSP and provide it as an input for the TwoPath function. In our implementation, first those old LSPs are examined that have the same ingress router as the new one. The reason for this is simple: our algorithm identifies an old LSP to be re-routed, and determines the new path for both the old and the new LSP. If the algorithm is implemented in a central place of the network (e.g., in a TE tool), the result of optimization should be communicated back to the network. If the ingress router of both LSPs are the same, the TE tool has to order a single LER to do the change, in this way simplifying the whole re-routing procedure. Of course, when there is no LSP with common ingress all of the remaining LSPs need to be examined.

3.3.3 TwoPath

In this section we formulate an Integer Linear Program that solves the establishment of two LSPs — the new LSP and an old one — simultaneously. Instead of the MPLS terminology, for LSP we use the term *path* that better fits the mathematical context.

We examine the problem of routing two paths. The network is represented by a directed graph $N = (V, E)$ with the set V of vertices and the set E of edges. Two pairs of nodes $s_1, t_1 \in V$ and $s_2, t_2 \in V$ are also given, representing the source and destination nodes of the two paths. The two paths to be routed can have different capacities. Therefore we classify the edges of the graph with the help of three subsets E_1, E_2 and E_b of E. Edges in E_1 and E_2 have enough capacity to accommodate the first path and the second path, respectively. (An edge can be in both sets at the same time). Edges in E_b cannot accommodate the two paths together.

With these notations our task is to find two paths P_1 and P_2 from s_1 to t_1 and from s_2 to t_2 such that $P_1 \subseteq E_1$, $P_2 \subseteq E_2$ and $P_1 \cap P_2 \cap E_b = 0$, that is, the path P_i can use only the edges of E_i and furthermore, the edges which are used by both paths must not be in E_b. By setting $E_b = E$, we can force the algorithm to find disjoint paths.

It can be shown that even this basic problem in itself is NP-hard. Our heuristic solution is based on the following *integer linear programming* formulation.

We seek the *integer* numbers $x_1^e \in \{0, 1\}, e \in E_1$ and $x_2^e \in \{0, 1\}, e \in E_2$

satisfying the conditions:

$$x_1^e \geq 0 \qquad \forall e \in E_1 \qquad (3.1a)$$

$$x_2^e \geq 0 \qquad \forall e \in E_2 \qquad (3.1b)$$

$$\sum_{e \in \rho(v) \cap E_1} x_1^e - \sum_{e \in \delta(v) \cap E_1} x_1^e = \delta_{v,t_1} - \delta_{v,s_1} \qquad \forall v \in V \qquad (3.1c)$$

$$\sum_{e \in \rho(v) \cap E_2} x_2^e - \sum_{e \in \delta(v) \cap E_2} x_2^e = \delta_{v,t_2} - \delta_{v,s_2} \qquad \forall v \in V \qquad (3.1d)$$

$$x_1^e + x_2^e \leq 1 \qquad \forall e \in E_b, \qquad (3.1e)$$

where $\rho(v)$ and $\delta(v)$ are the sets of the incoming and the outgoing edges of the node v and $\delta_{i,j}$ is 1 or 0 according to whether i is equal to j or not. Variable x_i^e indicates whether path i uses edge e or not.

We can extend this problem to an optimization problem by assigning two cost functions $c_1, c_2 : E \longrightarrow \mathbf{R}_+$ to the edges. In this case let the cost of a path P_i be the sum of the corresponding costs of the edges of P_i. We are looking for the paths which satisfy the above constraints and minimize the sum of the cost of the two paths.

Using the above ILP notations our task is to find the integer vectors x_1 and x_2 which satisfy (3.1a)-(3.1e) and approach

$$\min \Big\{ \sum_{e \in E_1} c_1(e) x_1^e + \sum_{e \in E_2} c_2(e) x_2^e \Big\} \qquad (3.2)$$

with respect to the constraints.

3.4 ILP solution

In this section we describe how to solve efficiently the ILP problem formulated in the previous section. Basically, any ILP software package can be used. The ILP solver packages use the so-called *branch&bound* technique [48] to solve an integer program. In our case this means solving several times the problem (3.2),(3.1a)-(3.1e) with some predefined variables but *without* the integrality constraints.

We tested the `lp_solve` package [49] and it gave good results, although the problem is NP-hard. However, we found that for larger graphs `lp_solve` can be slow. This comes from the fact that `lp_solve` uses the *simplex method* to solve the LP problem, which can be inefficient. Without the integrality constraint this problem is a *Multicommodity Flow Problem* [50] with two commodities, so we can improve our algorithm by replacing the simplex method with a more efficient one. We propose to use the so-called *Column Generation Procedure* [48], that we describe with the help of the following model:

It is well known that the optimal flows can be decomposed into paths. (Since we neglected the integrality constraint a flow can be routed on more than one path). We intend to find directly this decomposition. For this, let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the sets of *all* s_1—t_1 and s_2—t_2 paths in E_1 and E_2 respectively. In this formulation we have a variable α_P for all paths $P \in \mathcal{P}_1 \cup \mathcal{P}_2$. This α_P variable indicates what portion of the decomposed flow uses path P. Using these notations our task is to find

$$\min \Big\{ \sum_{P \in \mathcal{P}_1} c_1(P)\alpha_P + \sum_{P \in \mathcal{P}_2} c_2(P)\alpha_P \Big\} \tag{3.3a}$$

subject to

$$\alpha_P \geq 0 \qquad \forall P \in \mathcal{P}_1 \cup \mathcal{P}_2 \tag{3.3b}$$

$$\sum_{P \in \mathcal{P}_1, e \in P} \alpha_P + \sum_{P \in \mathcal{P}_2, e \in P} \alpha_P \leq 1 \qquad \forall e \in E_b \tag{3.3c}$$

$$\sum_{P \in \mathcal{P}_1} \alpha_P = 1 \tag{3.3d}$$

$$\sum_{P \in \mathcal{P}_2} \alpha_P = 1. \tag{3.3e}$$

This LP has only $|E_b| + 2$ constraints but the number of variables is typically enormous. However, it follows from a standard linear programming consideration that there exists an optimal solution having only at most $|E_b|+2$ non-zero variables and we can find it very efficiently without storing the entire huge constraint matrix by using the so-called *column generation procedure*. In order to use this method we transform (3.3c) to equality constraints by adding *slack variables*.

$$z^e \geq 0 \qquad \forall e \in E_b$$

$$\sum_{P \in \mathcal{P}_1, e \in P} \alpha_P + \sum_{P \in \mathcal{P}_2, e \in P} \alpha_P + z^e = 1 \qquad \forall e \in E_b \tag{3.3c'}$$

Because the method needs an initial solution we also modify the constraints (3.3d) and (3.3e)

$$s_1, s_2 \geq 0$$

$$\sum_{P \in \mathcal{P}_1} \alpha_P + s_1 = 1 \tag{3.3d'}$$

$$\sum_{P \in \mathcal{P}_2} \alpha_P + s_2 = 1. \tag{3.3e'}$$

Now we can easily obtain an initial solution to the problem by setting all variables z^e and s_1, s_2 to 1 and the rest of the variables to 0. If the variables s_1 and s_2 have sufficiently large costs then the optimal solution will avoid these variables if possible.

The column generation procedure uses the *primal simplex method* to get the optimal solution from the initial solution by a sequence of "pivot operations". This method stores a so-called *basis structure* which is a set of $|E_b| + 2$ variables such that each non-zero element of the current solution is one of these variables. In each iteration it chooses a new variable which *enters* the basis, modifies the current solution and chooses a *leaving* variable. Fortunately, in our special case we do not need to scan all columns to choose the entering variable but it can be done by using Dijkstra's shortest path algorithm. The reader is referred for example to [50] for a detailed description of this procedure. A good and in-depth survey on the linear programming background can be found in [48].

To get an *integer* solution for the problem (3.3a)-(3.3e) we establish a recursive `branch&bound` procedure which is called with the sets E_1, E_2, E_b and a bounding number B as input. As the output, it either states that there exists no feasible integer solution to the problem (3.3a)-(3.3e) whose *cost is less than* B, or it provides the optimal integer solution with cost less than B.

The `branch&bound` procedure first solves the problem (3.3a)-(3.3e) using the aforementioned column generation procedure. If there are no feasible flows, or the cost of the optimal flows is greater than B, then there will be no sufficient integer solution, so the procedure can state this as an output. It is also a straightforward consideration that if the two flows resulting from the column generation procedure do not use any edge of E_b together, then they must be integer flows, i.e., they are paths. Obviously, in this case this is an optimal solution, so `branch&bound` can return this. Finally, if the flows share an edge $e \in E_b$ then we do two recursive calls of `branch&bound` with the sets $E_1 \setminus \{e\}, E_2, E_b$, and $E_1, E_2 \setminus \{e\}, E_b$, respectively. When calling `branch&bound` with the first sets cost B is used as the bounding parameter. In the case that it finds an integer solution whose cost C is less than B, then C is used as the bounding parameter in the second call of `branch&bound` . When the first branch did not find an integer solution, cost B is used in the second call.

Our task is to have an integer solution to the LP problem, or to know that such a solution does not exist. Therefore we call `branch&bound` with the original sets E_1, E_2, E_b and $+\infty$ as the bounding parameter. If none of the recursive calls provide a feasible solution, then we can state that there exists no integer solution. Otherwise, one of the solutions provided by the recursive call will give us the best integer solution. By using the `branch&bound` method and the column generation procedure we get a more efficient method than the one implemented in an ILP software package.

3.5 Numerical results

We have conducted numerical investigations in order to show the real improvements resulting from the use of the proposed algorithms. Our goals with the numerical evaluation were twofold:

- primarily, we wanted to get some insight to the computational efficiency, and thus the practical applicability of the algorithms. Recall that the basic problem is NP-hard; it is therefore essential to check the limits of the ILP-based solution heuristic.

- secondly, we wanted to quantify the actual performance gain resulting from the use of the proposed algorithm in a practical network. Besides showing the trivial fact that allowing a re-routing of an existing LSP — when simple CSPF-based routing fails — does indeed improve the success probability of a new LSP setup, we wanted to characterize the network-level gain in a real-world system.

Several different approaches are used for the performance evaluation of routing strategies. For example, Plotkin [43] proposes an analytical approach for this purpose. Another method is to use discrete event simulation, assuming a stochastic process for the arrival of demands [C2].

In our case it was hard to find a tractable analytical approach that can be used to obtain practical results. Discrete event simulation was again out of the scope because of the lack of a well-established and justified LSP demand arrival model in the case of MPLS. Therefore, we investigated the performance of the proposed algorithms by creating random snapshots at different network loads.

3.5.1 Computational efficiency

To measure the running time of the algorithm we generated *random network topologies* of 50, 100, 150 and 200 nodes. We have run our column-generation based algorithm several times using Sun UltraSparc 5 computers, and calculated average running times for the different examples. For comparison we solved the same problem instances with the `lp_solve` [49] package as well. Table 3.1 summarizes the results.

Two important conclusions can be drawn from the results. The first — and probably most important — message is that our ILP-based solution scales very well for practical network sizes (150-200 nodes), although the problem itself is NP-hard. Second, the results show that — with respect to running times — our column-generation based procedure performs significantly better than `lp_solve`.

Table 3.1: Average running times

	50 Nodes	100 Nodes	150 Nodes	200 Nodes
Lp_solve	0.14s	1.4s	4.6s	8.2s
Column Generation	0.06s	0.5s	1.8s	5.2s

3.5.2 Performance evaluation

In order to have practically relevant results we decided to base our investigations on the *AT&T IP Backbone network topology*. This real-world data network contains 25 backbone nodes and 88 links, resulting in an average node degree of 3.52. The exact topology and capacity values of this network can be found in Appendix A. The most common link capacity is OC-48, i.e., 2.5 Gbps.

Since it is hard to get traffic statistics from real data networks deploying MPLS, we created randomly generated traffic situations. This means that we have loaded the network to a certain extent with randomly placed LSPs having random bandwidth values uniformly-distributed in the interval [8Mbps, 622Mbps]. Using these random traffic situations we evaluated the probability of being able to add a single new LSP by trying to establish several new ones, one after another. All the new LSPs were tried with random capacity, origin and destination nodes, using the same initial traffic situation. This test has been conducted for several different network load values, resulting in a series of probabilities describing the quality of the routing strategy at different traffic levels.

The main purpose of the first round of our simulations was to determine the difference in successful LSP establishment probability between a constrained shortest path first (CSPF) algorithm and our advanced ILP based algorithm that allows the re-routing of an already established LSP in the case when CSPF fails. As we have already mentioned all CSPF algorithms do a bandwidth based filtering of the graph and remove those links that do not have enough free capacity to accommodate the LSP to be routed. On the remaining graph several approaches are used to select the actual path for the LSP. In our simulations we used such an CSPF algorithm, that restricted path selection to the shortest path. As a further optimization we used a cost function that selects a path on which the total free bandwidth after the LSP setup, summarized for all path links, was the smallest. This strategy ensures that at high link loads the algorithm performs close to the optimum, while at low link loads it can balance the load on different shortest paths. More precisely, the weight of an edge was determined by using the equation:

$$w(e) = 1 - \frac{1}{N}\frac{B_{free} - B_{LSP}}{B_{max}}$$

where B_{max} is the total reservable bandwidth of the link, B_{free} is its free capacity, B_{LSP} is the bandwidth of the LSP for which we calculate the path, and N is the total number of nodes in the graph. Since the path we are looking for is loopless, its maximal length is limited by the number of nodes. Therefore the division by N in the subtracted part ensures that only shortest paths are used. The bandwidth dependent part is similar to the residual bandwidth ratio method [51] that aims at load balancing.

In the simulations first we loaded the network to a certain level by generating LSPs randomly between all nodes. This traffic situation is characterized by the *total throughput*, i.e., the sum of all established LSPs' bandwidth. In order to approximate the success probability we randomly generated several new LSPs, and tried to route them both with the CSPF method and with our optimized method, all of them in the same traffic situation. During the simulation we only noted whether the setup was successful or not, thus the time factor was eliminated from the experiment. In the simulation results below, the 95% confidence intervals were so close to the lines that for the sake of better visibility we decided to omit them from the figures.

As we can see in Fig. 3.2, the success probability of our ILP-based method is always above of CSPF's, as it trivially has to be, considering the basic principle of our method. (Giving another chance for the set-up when CSPF failed will definitely not decrease the success probability.) More importantly, however, Fig. 3.2 shows that the performance gain of our optimized method is quite significant when compared to the original CSPF method.

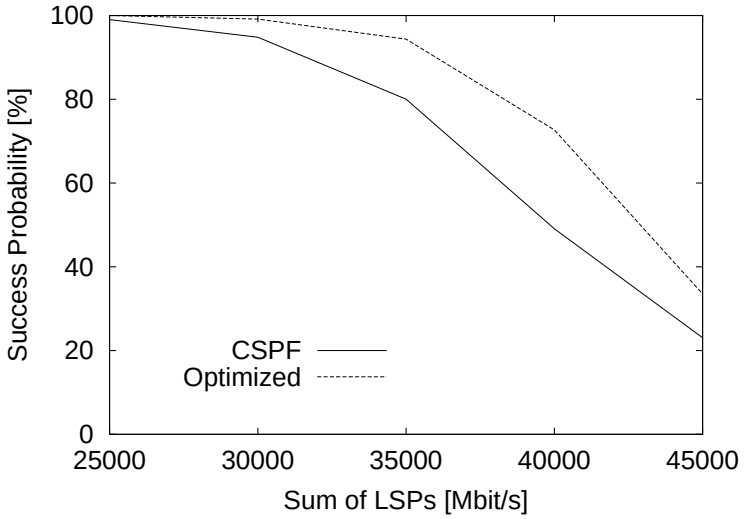


Figure 3.2: Efficiency of optimization, same initial state, centralized method

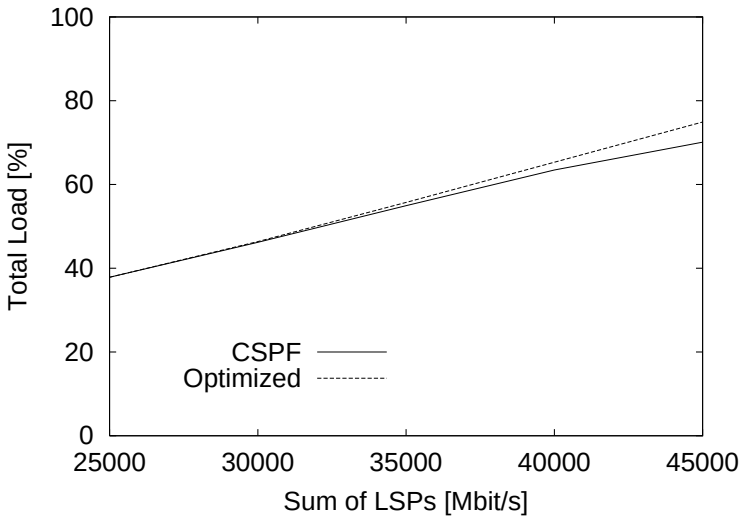


Figure 3.3: Total network load, different initial states, centralized method

Applying our proposed re-routing strategy, however, implies that LSPs originally routed on shortest paths will possibly be directed to longer bypasses in order to accommodate new incoming requests. This means that, when applied several times, our optimization method might result in a higher global network load (link utilization) for a certain throughput than the original CSPF. Therefore, in our second simulation we investigated the case when an operator uses either CSPF or our optimized method for LSP establishment all the time from day one. This means that the initial loading of the networks to a certain throughput level was achieved in different ways for the two curves. In the first case, all LSPs were established by CSPF. In case of the optimized method, from the very beginning LSPs were routed by our ILP-based method whenever CSPF failed. The results presented in Fig. 3.3 seem to confirm our theoretical expectations: use of the optimized method indeed generates higher network load for the same throughput.

We may expect that this phenomenon will adversely effect the success probability of our optimized method. However, as we see in Fig. 3.4, this effect is significant only in the range of higher network load values. The optimized method retains its significant advantage above CSPF until the utilization level of the network reaches 70% (observe that this is a rather high load value, considering the achievable 20% success probability).

As we mentioned in the introduction, our optimization method was originally intended to be part of a central network traffic engineering tool, where there is global access to information on all LSPs. However, it is possible to base the optimization on local information only and implement the algorithm in the routers of the network in a distributed way. In this way we restrict ourselves to re-routing

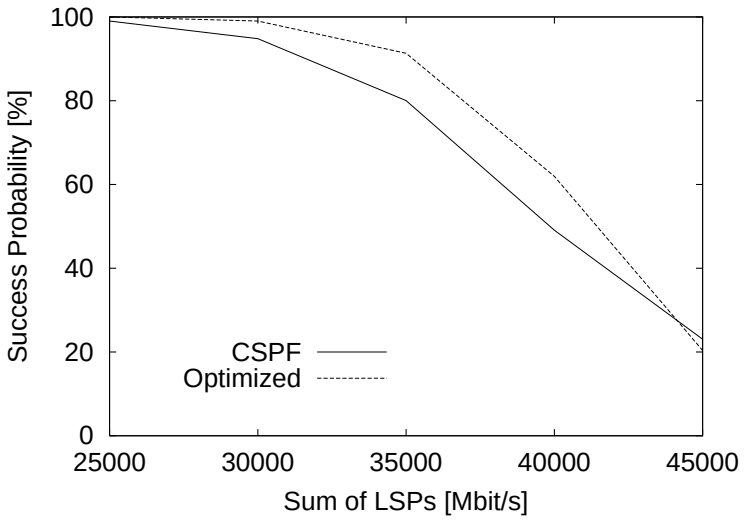


Figure 3.4: Efficiency of optimization, different initial states, centralized method

only those LSPs that originate from a particular router.

In Fig. 3.5 the results of simulating this distributed implementation shows that we can still expect some gain by using the optimized method instead of the simple CSPF method. However, this gain is only marginal, therefore the distributed implementation of the algorithm has less practical relevance.

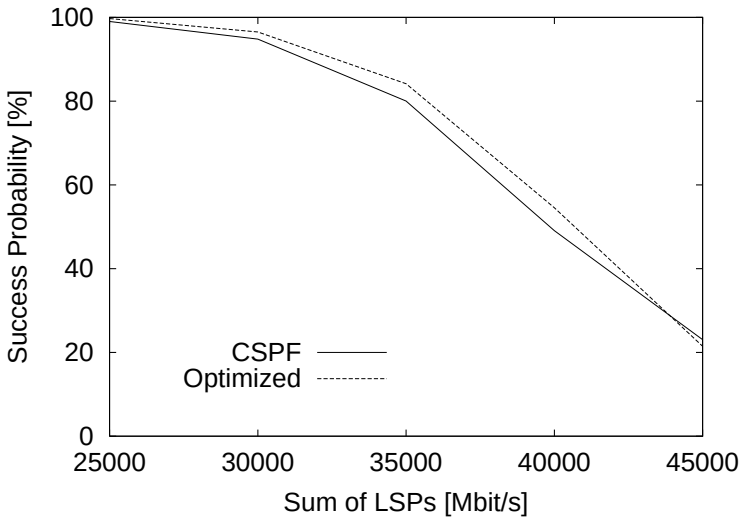


Figure 3.5: Efficiency of optimization, different initial states, decentralized method

3.6 Summary

The algorithm described in this chapter is based on the idea that when there is insufficient capacity in the network to set up a new LSP, we try to free some capacity on bottleneck links by re-routing one of the established LSPs. We have proposed a systematical method for finding such an LSP. We have developed an ILP-based solution that can efficiently solve this problem for practical network sizes. On the simulated real-world network topology empirical results have shown that the proposed algorithm, if implemented in a centralized place, significantly improves the probability of successful LSP establishment. The distributed method provided some benefits compared to the simple CSPF based solution, although with lower efficiency than the centralized method. These performance results derived for a specific real-world network suggest that the use of our algorithm holds the promise of a higher LSP setup probability for other network topologies as well. However, to fully underline this, further numerical evaluations are needed on diverse network and traffic configurations.

Future investigations may include the development of additional heuristics to minimise the running time of the algorithm. One improvement could be to sort LSPs based on a probability measure describing the ease with which new LSP may be routed if a particular old LSP is re-routed. Another direction for future work is to investigate the more complicated problem of allowing the re-routing of not only a single, but more LSPs.

Chapter 4

On the effectiveness of restoration path computation methods

4.1 Introduction

To provide higher service availability for customers willing to pay extra for the improved service, ISPs use protection and restoration mechanisms in their networks. Baroni et al. [52] carried out a detailed overall network cost analysis of different protection and restoration architectures. The results show that mesh-based restoration provides lower overall cost than ring-based methods. While routing design of ring-based methods is quite straightforward, path computation in a meshed network is a challenging task.

Path optimality aspects of restoration methods for meshed networks were studied by Benerjea in [53] where path-based (head-end), link-based (local reroute) and hybrid rerouting methods were compared. Not surprisingly, in simulation studies path-based rerouting turned out to have better resource utilization than a local reroute.

Depending on how much service interruption is allowable, restoration path computation can be carried out in real time (i.e., after the failure), or paths can be precomputed. Furthermore, if the restoration paths are precomputed there is a choice whether they are pre-established or not.

In order to achieve high capacity utilization, certain methods use centrally planned routing computation. These are generally path-based see e.g., [2]. The disadvantage of such methods is that they may not scale well. Alternatively, distributed restoration path computation can be used. Methods in this category span from simplest Penalty or Suurballe's method [54] to such methods where sharing of backup paths are optimized [55].

In this chapter, we introduce a new distributed restoration path computation algorithm called 'Shortest Backup' method. We do not consider sharing of backup

paths. The new method can be used when paths are *precomputed, but not pre-established*. Our basic idea is to select strictly shortest paths for the working (primary) and restoration (backup or secondary) paths as well. We allow common links between the two paths, to achieve higher efficiency than Suurballe's method. In order to prepare for the failure of the common links between the primary path and the first (shortest) restoration path, an additional restoration path is computed for a working path. By minimizing the number of common edges between the working path and the shortest backup path, the method *maximizes the probability that the shortest backup path can be used for restoration*.

By forcing the restoration path to be a shortest path the used network resources are minimized. More importantly, in this case it is not necessary to revert to the original working path after the failed link is restored. Instead, sub-optimal paths can be re-optimized in the network in order to balance traffic load.

The rest of this chapter is organized as follows. We start in Section 4.2 with an overview of MPLS technology. The application of the studied algorithms are not limited to this technology, but the selection of a well defined architecture eases the understanding of the chapter. In Section 4.3, we survey restoration path computation methods from the literature and describe the new Shortest Backup method in Section 4.4. In Section 4.4.1, we demonstrate the differences between various restoration path computation algorithms via an example. In Section 4.5, we present the results of detailed numerical evaluations. Finally Section 4.6 summarises the chapter.

4.2 MPLS background

In this section we give a brief overview of the most important restoration related functionality of MPLS networks. We concentrate on the entire establishment, recovery, reversion and re-routing cycle as discussed in [56].

When configuring a Label Switched Path (LSP) at its head-end (ingress) router, its destination (egress) router, as well as its bandwidth requirement and optionally other constraints (hop-limit, explicit hops) are specified. The complete path of the LSP can be computed in a central traffic engineering tool, or alternatively by the head-end router using a Constrained Shortest Path First (CSPF) algorithm. In both cases the LSP setup is initiated with explicitly defined strict hops.

To provide restoration for an LSP both *path-based* and *link-based* methods are defined in IETF. The MPLS terminology for the former is *backup* or *secondary* LSP, while the latter is usually referred to as *detour* or *fast reroute*. To protect the working path of an LSP one or more backup paths can be configured. If the path of the primary LSP is determined by CSPF it is possible to extend the simple bandwidth constrained path computation algorithm [51] to find two disjoint paths. Algorithms that can be used for this purpose will be discussed in Section 4.3.

In MPLS, link-based protection is only intended to be used only to eliminate service interruption for the short time that is needed to notify the head-end about the failure. A detour provides the immediate upstream router (from the failure) with an alternate path that can route around the failed downstream node, link, or shared risk link group [57], [58]. Detour paths are determined at each hop of the working path by a local path computation algorithm in the label switching routers. A single bypass tunnel (detour) can be established for all interrupted LSPs between two routers [59]. In the rest of this work we concentrate only on path-based restoration, i.e., we do not consider detours in our study.

Once computed, a backup path can be established together with the working path. In this case the backup path is called hot-standby. Alternatively, backup LSPs can be established only after the failure of the primary path. If there are more backup LSPs configured they can be tried one after the other, or if the failed link/node can be identified a backup LSP can be chosen which does not include the failed link. As a third possibility, LSP paths can be calculated on-demand after the failure. However, this can take quite a long time when there are lots of LSPs, since the CSPF algorithm has to calculate a path for them one-by-one. According to [60], the best possible strategy is to precompute the backup path without pre-establishing it.

LSPs are established using the CR-LDP [61] and RSVP-TE [62] signalling protocols to allocate labels and reserve resources along the precomputed path. There are proposals for crankback routing extensions to CR-LDP and RSVP-TE to indicate the location of the failed link or node to the ingress router [63]. This can be viewed as a fast topology update, enabling the ingress router to select a restoration path that does not include the failed link. Crankback extensions are helpful if there are more (not totally disjoint) precomputed paths, or when restoration paths are computed after the failure.

Since restoration paths may not be optimal, after a hold-down time, reoptimization of paths may be triggered. Reoptimization should be performed only in such cases when significant bandwidth utilization gain can be achieved. As Villamizar pointed out [64], the hold-down time after a failure should be long enough to allow feedback of resource consumption changes caused by all LSP rerouting attempts.

When failed resources are put back to service, traffic can be restored and the traffic layout in the network can be balanced. There are different ways of achieving healthy traffic routing in the network. If LSPs are established frequently, one possibility is to let new traffic fill the gap on the restored link. Alternatively, when restoration is configured to be revertive, traffic from backup paths can be moved back to the preferred primary paths. When LSPs are configured to be non-revertive, LSP reoptimization can be used to utilize the free capacity of the restored link. A specific implementation of the non-revertive case is when primary

and secondary paths change roles; once traffic is redirected to the backup path, the former active path becomes the backup of the new active path. In this case, it is important that the two paths are equal quality paths, i.e., both are shortest paths. The non-revertive mode with reoptimization may result in less path change in the network.

4.3 Backup path computation methods

Calculation of working and restoration paths as previously mentioned, can be done in either a centralized or a distributed way. In this section we provide an overview of proposed path computation methods in the literature. All discussed algorithms provide protection for single link or node failures.

Kodialam and Lakshman [55] propose both a central and a distributed method in which *sharing of backup paths* is allowed among certain active paths. The basic idea in both cases is the following: if two active paths do not have a common link, then their backup paths can share bandwidth on the entire path since no link failure can cause the activation of both backup paths. In the central method the authors suppose that complete path information is available and give an integer programming formulation for the problem. Moreover they show that by maintaining aggregated and not per-path information, a distributed method can perform almost as well as the complete information scenario. Since implementation of any backup path sharing algorithm needs modification to currently proposed traffic engineering extensions of interior routing protocols, the authors propose extensions for OSPF and ISIS [58].

There are a number of basic backup path selection algorithms that do not require protocol changes. Without a global view of LSP paths, distributed algorithms prefer the shortest or close to shortest paths in order to provide good resource utilization (close to the global optimum). Since working paths are those that really reserve resources, it is important to have a *shortest working path* (e.g., in terms of hop count). The restriction for the backup path usually is to be *disjoint* from the active path, and to be as short as possible. Unfortunately, the problem of finding a shortest path having a shortest disjoint backup path is NP-hard. In the following paragraphs we survey simple restoration path selection algorithms.

A heuristic solution referred to as the Penalty method in the literature [4], first obtains an arbitrary shortest path between the source and the destination node using Dijkstra's method. In order to find a disjoint backup path from the selected shortest path, Dijkstra's algorithm is executed on a modified graph where *the link weights of the working path are increased*. If the weights are significantly increased, the second shortest path computation will find a shortest disjoint path if there is any. There is a weak point of this heuristic algorithm; depending on the selection of the shortest path, secondary paths that are longer then necessary

might be found. There is no way to optimize how long this secondary path will be.

If we relax the constraint that the primary path has to be shortest, we get the problem of finding two disjoint paths for which the *sum weights are minimal*. An alternative statement of this problem is that of sending two units of flow as cheaply as possible from node s to node t in a unit capacity network. Therefore, this problem is generally referred in the literature as the *two-valued unit capacity minimum cost flow problem*. The book of Ahuja, Magnanti and Orlin [50] provides an excellent overview of general minimum cost flow algorithms. For the special case of this problem when we are interested in finding the two-valued minimum cost flow between a source node and *all other nodes* in a graph, Suurballe and Tarjan provided a fast method [54]. This is widely referred to in the literature as Suurballe's method.

Although the problem of finding a shortest path having a shortest disjoint backup path is NP-hard, an *integer linear programming problem* can be easily formulated and more importantly, it provides very good practical results as e.g., reported by Cinkler et al. [2], [3]. Considering the quality of the found paths (length of secondary path), this algorithm is more efficient than the Penalty method. On the other hand it takes more time because of the complexity of the Integer Linear Programming problem.

All of the methods above compute one restoration path for an active path. If the crankback feature is used and we can store more restoration paths, failure dependent path precomputation methods can be used. One solution is to compute one restoration path for each link in the working path. These paths can be calculated by the Penalty method, by increasing only one link's weight at a time. The disadvantage of this algorithm is that many secondary paths have to be stored.

4.4 Proposed method

In this Section a restoration path computation method is proposed, called 'Shortest Backup' that pre-computes and stores *two backup paths* for a working path. Since the backup paths are not totally disjoint from the working path, we must know which link has failed when selecting the backup path from among the stored two paths.

We use *precomputation* to provide *fast restoration*. To select a stored backup path the method uses a simple lookup operation which is much faster than a Dijkstra-based algorithm used in on-demand schemes. In real time methods, after a link failure CSPF should be performed for every affected LSP, since the LSP's bandwidth is treated as a constraint. If real time path computation is used, this sequential computation can take a significant amount of time depending on the link size and the number of LSPs on it.

We search for a shortest backup path, since this *minimizes the used resources*. Moreover — as described in Section 4.2 — when non-revertive restoration paths are used, working and backup paths may change roles after the failure. This can only be done efficiently when both paths are optimal, i.e., both are shortest. When the failed link is restored, it can be utilized to *reoptimize other sub-optimal paths* in the network.

Let us formulate a path computation problem in which we seek two shortest paths having a minimal number of common edges. Imagine that we have an algorithm that provides for example two four-hop shortest paths having only one common edge. If we configure one as the primary path and the other as a non-revertive secondary path, we achieve a 75% protection against single link failures (three out of four links are protected). To achieve full protection we need one more backup path to protect against the failure of the common edge of the previously found two shortest paths. Since this second backup path may be sub-optimal we may configure it to be revertive. Thus we have identified two sub-tasks. One is to *compute two shortest paths having minimum number of common edges* (sub-task 1), while the other is to *compute a third path that provides full protection for the working path* (sub-task 2). The proposed algorithm for each sub-task is given below.

4.4.1 Sub-task 1

The key step in our algorithm is to determine two shortest paths having a minimal number of common edges. More precisely: given a directed acyclic graph $D = (N, E)$ with a weight function $w : E \rightarrow \mathbf{R}^+$ associated with each edge $e \in E$, a source and a destination node $s, t \in N$, the task is to find two paths P_1, P_2 between s and t, such that:

$$\sum_{e \in P_1} w(e) = d_{min}(s,t)$$
$$\sum_{e \in P_2} w(e) = d_{min}(s,t)$$
$$min \mid P_1 \cap P_2 \mid$$

where, $d_{min}(s,t)$ is the sum of edge weights of the shortest path between s and t in D.

To describe our solution for the above problem, first we construct an auxiliary graph in which every path represents a shortest path in the original graph. We can obtain this auxiliary graph by running a simple shortest path calculation algorithm, determining a feasible potential for the graph nodes and marking edges as 'tight' or 'not tight'. In the next section we describe these steps one after the other with necessary definitions.

Solution

Our solution presented later in this section builds upon some graph theoretical definitions and theorems. Although these concepts are more or less well-known, we summarize the essentials here in order to support the reader in understanding the algorithm. For further details and proofs, see e.g., [50].

Definition 1 *Given a directed graph $D = (N, E)$ and a weight function $w : E \to \mathbf{R}^+$, a function $\pi : N \to \mathbf{R}$, called potential is feasible if $\pi(v) - \pi(u) \leq w(u, v)$ for every edge $uv \in E$.*

It can be proven that the output of Dijkstra's classical algorithm — the length of the shortest paths from s to each other node in the graph — defines a *feasible potential*. That is:

Proposition 1 *If w is non-negative, and $\pi_{min}(v)$ is defined to be the minimum cost of a path from a fixed s to v, then π_{min} is a feasible potential, and $\pi_{min}(v) - \pi_{min}(s) \geq \pi'(v) - \pi'(s)$ holds for every node $n \in N$, and for every π' feasible potential.*

Now let us show why this potential is useful. Remember that our aim is to construct a graph in which each path represents a shortest path in the original graph. Let us call an edge *tight* if its weight equals to the potential difference of its end nodes, i.e. $\pi(v) - \pi(u) = w(u, v)$.

Proposition 2 *A path in the graph is shortest if and only if each of its edges are tight.*

If we know a feasible potential of a graph, the sought auxiliary graph — or as we call the *Graph of Tight Edges*, D_t — can be constructed.

After these prerequisites we can describe the actual algorithm. Recall that we intend to find two paths having minimum number of common edges. If we seek these path in the graph of tight edges, it is guaranteed that both paths will be shortest. In this subsection we show how we can make further modifications to D_t in order to be able to simply use Suurballe's method for our purposes.

Let us duplicate each edge in D_t and assign weight '0' to the old edge and weight '1' to the new edge, respectively. If Suurballe's minimum cost disjoint paths algorithm is executed on this second auxiliary graph, the output will be surely two disjoint paths, since there are two distinct edges between each node. However, it may happen that these paths use both an edge with weight '0' and an edge with weight '1' between two nodes. This means that in the original graph, the two paths have a common edge. Suurballe's algorithm minimizes the sum weight of the two paths, therefore the number of common edges will be minimized, due to the proposed special weighting. Fig. 4.1 depicts the flow chart of the algorithm.

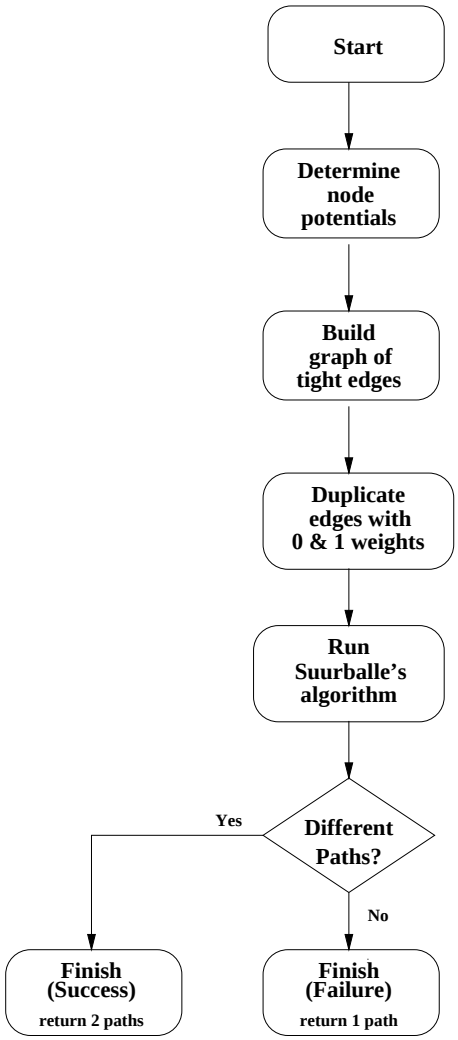


Figure 4.1: Flowchart of the algorithm for sub-task 1

Example

To show the differences between the different backup path computation algorithms — namely Penalty, Suurballe and Shortest Backup methods— we have created a simple artificial example graph in which for each edge $w(e) = 1$. The paths found by the different methods are shown in Fig. 4.2, 4.3 and 4.4. As we will see in Section 4.5, the basic characteristics of the algorithms identified in this section can be spotted in the resulting path statistics as well.

Suurballe's method minimizes the total weight of the two paths. In our case, the working path has $w = 4$ while the backup path has $w = 5$ (Fig. 4.2). It may happen that none of the paths found by Suurballe's method are shortest. Imagine

that the third edge of the lower path in Fig. 4.2 has $w = 2$. In this case Suurballe's algorithm would still find the same paths, but none of them would be shortest. The remaining two shortest paths would not be disjoint from the $w = 5$ long backup path and for these a $w = 7$ long backup path would not result in such a path set that has minimum total weight.

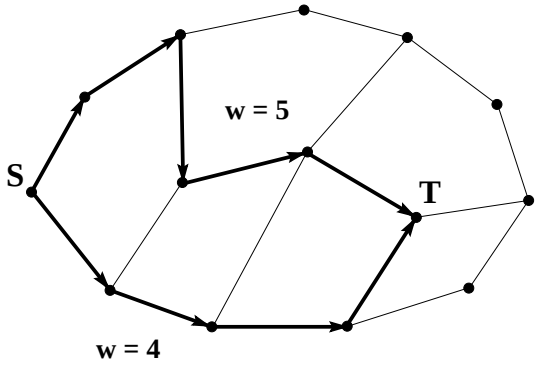


Figure 4.2: Routes found by Suurballe's method.

The Penalty method selects a random shortest path and then selects a disjoint backup path by increasing the edge weights of the shortest path. In Fig. 4.3 a pessimistic case is plotted, in which a shortest path is chosen that has a $w = 7$ long backup path. If the same shortest path were chosen as before, the Penalty method would output the same paths as Suurballe's method. One can note that the probability of selecting a 'bad shortest' path is 66%, since there are three possible shortest paths, among which two has a long backup path and only one has a short backup path.

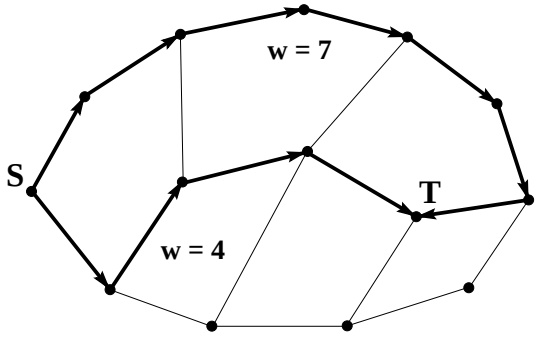


Figure 4.3: Routes found by Penalty method.

The Shortest Backup method finds two shortest paths having minimum number of common edges. In the graph of Fig. 4.4 only tight edges are shown together with the feasible potentials determined by the shortest paths. In order to protect

against the failure of the common edge, a third path must be calculated. The next subsection describes how this is done.

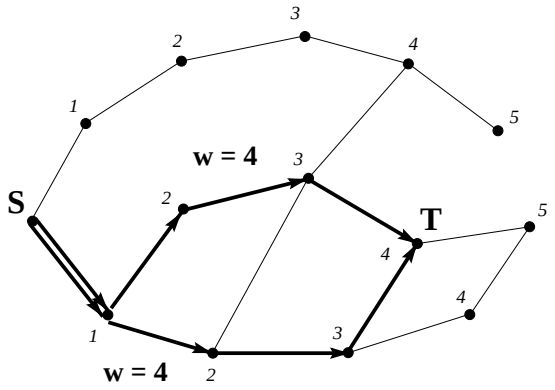


Figure 4.4: Routes found by Shortest Backup method.

4.4.2 Sub-task 2

With the above algorithm we produced two shortest paths, that may have common edges in the original graph. Thus a third path is needed that can be used as a revertive backup path. The only requirement for the third path is to be disjoint from the common edges of the working and the non-revertive backup paths. Further optimizations are possible, but it is believed that the third path has very limited effect on the most important measures, namely the average secondary path length and the probability that the backup path is a shortest one. Therefore, such a simple requirement seems to be enough.

The third path guaranteeing protection for the failure of any link of the working path is computed with the following algorithm (sub-task 2):

1. Start with the original graph;

2. Assign such a weight $w(e)$ for the common edges of the first two shortest paths $\mid P_1 \cap P_2 \mid$ that is larger then the summarized weights of all edges in the graph. This means that all edge weights of the working path will be increased if the two paths were identical in the previous step;

3. Determine a minimum cost s, t path by running Dijkstra's shortest path algorithm;

Fig. 4.5 depicts the flow chart of the algorithm for sub-task 2.

Looking at Fig. 4.4 we may notice that by increasing the weight of the common edge of the two shortest paths, a subsequent shortest path calculation would find the same path as the $w = 5$ backup path of the Suurballe method (see Fig. 4.2). This five hop backup could then be used to protect the first link of the active path.

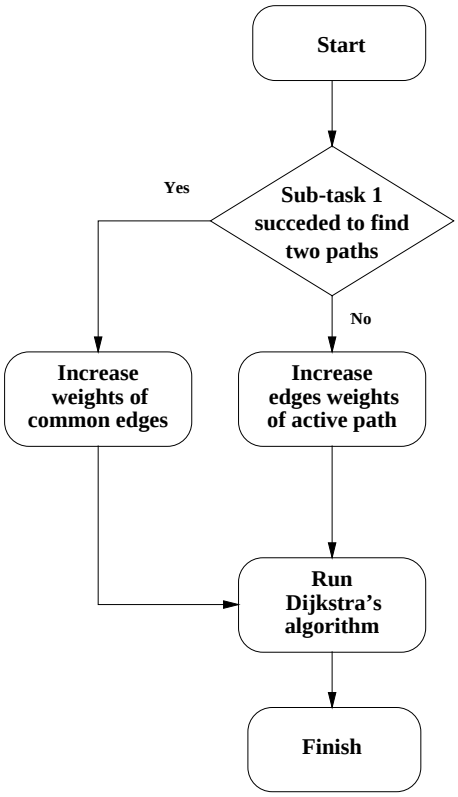


Figure 4.5: Flowchart of the algorithm for sub-task 2

4.4.3 Applicability of the method

In section 4.4.1 we said that two such minimum cost paths are searched that have the least number of common edges. Here we defined $d_{min}(s,t)$ as the sum of edge weights of the shortest path between s and t. The algorithm works theoretically for arbitrary weighting. It can also be used after pruning of non conforming links, based on e.g., a bandwidth check. However, as the probability of equal cost paths in the graph reduces, so does the chance of finding two shortest paths with a minimal number of edges. The reason for this is simple: when the number of edges of the auxiliary graph determined by the potentials of the first Dijkstra algorithm becomes very low, it will be impossible to find two paths by the subsequent Suurballe's algorithm executed after edge duplication. However, if we use such a weight function that is not continuous but discrete, the probability of successful application of the proposed 'Shortest Backup' methods can be increased.

4.5 Numerical Results

We have conducted numerical experiments to investigate the produced path lengths of different CSPF methods. In order to get practically relevant results, the first two simulation scenarios were carried out on publicly available real-world backbone network topologies. In the third experiment we used randomly generated topologies to get more general results for different network sizes. The first experiment aims at measuring path length without any statistical effect. Therefore, path lengths are counted between all ingress egress pairs. The second and third experiment uses random traffic to measure the path length statistics of backup LSPs. These more realistic scenarios will be described in Section 4.5.2 for real-world topologies and in Section 4.5.3 for random topologies.

4.5.1 Simple backup path statistics

In the first round of experiments we generated a full-mesh traffic matrix. At this experiment bandwidth reservations were neglected when doing path calculations. The purpose of this setup was to get simple backup path length statistics for the four path computation methods, namely Suurballe's method, the Penalty method, the ILP method and our Shortest Backup method. Since the Penalty method always produces a shortest path as a working path, we could compare this with the length of the paths produced by Suurballe's method. We say that Suurballe's method produced *long paths* if neither its working, nor its backup path is shortest. On the other hand, we say that Suurballe's method produced *short paths* when both of its paths were shortest paths. We say that the Penalty method produced *long paths*, if the length of its backup path was longer than the length of the backup path generated by Suurballe's method.

We carried out simulations on two network topologies, namely on the Cable & Wireless and on the AT&T network. The exact topology of the networks can be found in Appendix A. Suurballe's method produced 4 long paths out of 870 and 600 cases for the C&W and AT&T networks respectively. This means that although Suurballe's method aims at minimizing the total length of the two paths, it almost always achieves this in such a way that one of the paths is shortest. Moreover the number of short Suurballe paths were 116 for both networks. This means that for around $15 - 20\%$ of the source destination pairs, both paths were shortest paths. The Penalty method produced long paths altogether 22 and 30 times for the C&W and AT&T networks respectively, which is around 5% of the total cases.

In Table 4.1 and 4.2, based on the length of the theoretical shortest path between the source and destination of traffic demands, we present the distribution of the number of paths and the number of cases when the Suurballe method produced optimal paths, moreover when the Penalty method produced sub-optimal paths.

The network diameter for the C&W network is 7, while for the AT&T network it is 5. The ILP method always produced optimal paths, i.e., its first path was shortest, and its second path was never longer than the backup path produced by Suurballe's method. The drawback of the ILP solution is its slow speed. The running times of all other methods does not differ significantly. Both Suurballe's methods and the Penalty methods execute two shortest path calculation and some additional graph modifications, while the Shortest Backup method produces its results with the help of three shortest path calculations.

Table 4.1: Cable & Wireless 30 node network

SP hop count	1	2	3	4	5	6	7
Number of paths	104	204	236	170	104	46	6
Short Suurballe	0	30	32	26	18	4	6
Long Penalty	0	0	4	4	4	10	0

Table 4.2: AT&T 25 node network

SP hop count	1	2	3	4	5
Number of paths	82	184	192	118	24
Short Suurballe	0	34	40	38	4
Long Penalty	0	0	20	8	2

The performance of the Shortest Backup method cannot be measured by the length of its paths, since it produces shortest paths for both of the paths. Therefore we measured the number of common edges between them. In Table 4.3, cases in column 0 are actually those that can also be found by the Suurballe's method (note that the row sum of 'Short Suurballe' in Table 4.1, 4.2 is the same as column 0 in Table 4.3). On the other hand, columns 1-4 in Table 4.3 represent the additional cases when our method managed to find shortest backup paths. Summing up the respective cells in Table 4.3, we can see that this method produced 196 and 136 more shortest paths than the simple Suurballe's method. Moreover, more than half of these paths had only one common edge. Altogether this method found two shortest paths for 36% (312 out of 870) and 42% (252 out of 600) of the total source destination pairs. This is a promising result.

4.5.2 Advanced backup path statistics

In this round of simulations we used the AT&T topology with randomly generated LSPs. The exact topology of the network can be found in Appendix A. The

Table 4.3: Number of shortest paths for Shortest Backup

Number of common edges	0	1	2	3	4	Total
C&W network	116	102	52	30	12	312
AT&T network	116	106	30	0	0	252

required network load was achieved by increasing or decreasing the number of LSPs in the network. The LSP ingress, egress points were selected randomly, while the LSP bandwidth was either DS-1 (1.5Mbps), DS-3 (45Mbps), OC-3 (155Mbps) or OC-12 (622Mbps) modelling a leased line interconnect service. Out of 63 LSPs on average one had OC-12 bandwidth demand, 8 had OC-3, 7 DS-3, finally 47 LSPs had DS-1 bandwidth demand.

In our simulations we used bandwidth constrained path selection, i.e., unfeasible links were deleted from the graph used for path computation of a specific LSP. We loaded the network to a predefined load value and calculated the average length of secondary paths for the Suurballe and the Penalty method with the following equation: $(\sum w_b w_a)/\sum w_a$, where w_b is the length of the backup path and w_a is the length of the active path. The Shortest Backup method uses two backup paths depending on whether the failed link is included in shorter backup path or not. Therefore we used a weighted function to determine the average path length, namely: $(\sum w_{b_1}(w_a - c) + \sum w_{b_2} c)/\sum w_a$, where c is the number of common edges between the b_1 and the a paths. The simulation results plotted with 95% confidence intervals can be seen in Fig. 4.6.

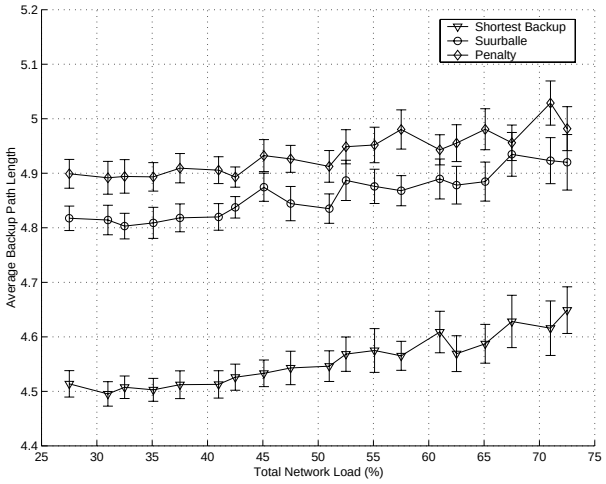


Figure 4.6: Path length statistics

We also measured the probability of having a shortest path as the backup path, with 95% confidence intervals as before (Fig. 4.7). For the Penalty and Suurballe's

methods it was 15% and 17% respectively, while for the Shortest Backup method it was significantly higher. As shown in the figure, 35% of the failed paths need not be reverted back to the original active path, which means increased network stability.

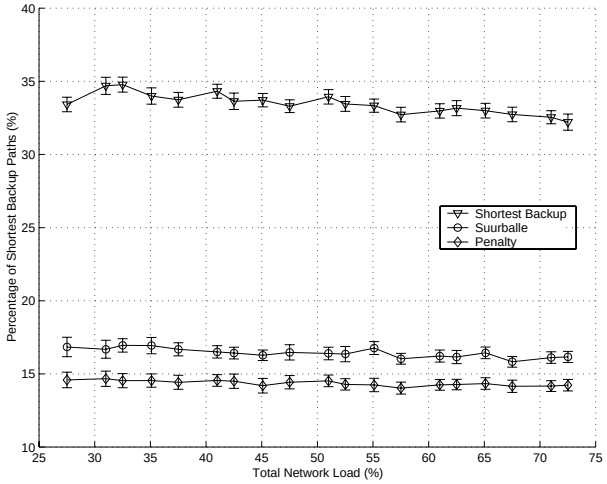


Figure 4.7: Shortest backup path probability

4.5.3 Backup path statistics for random graphs

In order to obtain a more general result we investigated the performance of the backup path computation algorithms on random topologies in a third experiment. For each round of simulation we generated different random networks using the tool of Józsa and Orincsay [65], with 10, 20, 30, 40, 50, 60, 70, 80 nodes and 3.0, 3.25, 3.5, 3.75, 4.0, 4.25, 4.5 link densities. Having seen in the previous experiment that the network load does not fundamentally affect the two measures, we investigated all the networks at 50% average link load.

We found that all three algorithms achieved approximately the same average path lengths (Fig. 4.8). The probability of having a shortest path as the backup path differed however (Fig. 4.9). The 95% confidence intervals in the figures are much larger for this round of simulations, due to the fact that in each round a new random network was generated. As one can see in Fig. 4.9, as the number of nodes in the network increased, so did the performance of the 'Shortest Backup' method. Similarly if the number of nodes are fixed and the edge densities are increased, the difference between the algorithms become more noticeable. We can conclude from these experiments that the proposed new backup path computation method produces the best results when the number of alternative paths in the network is large, in line with the expectations of section 4.4.3.

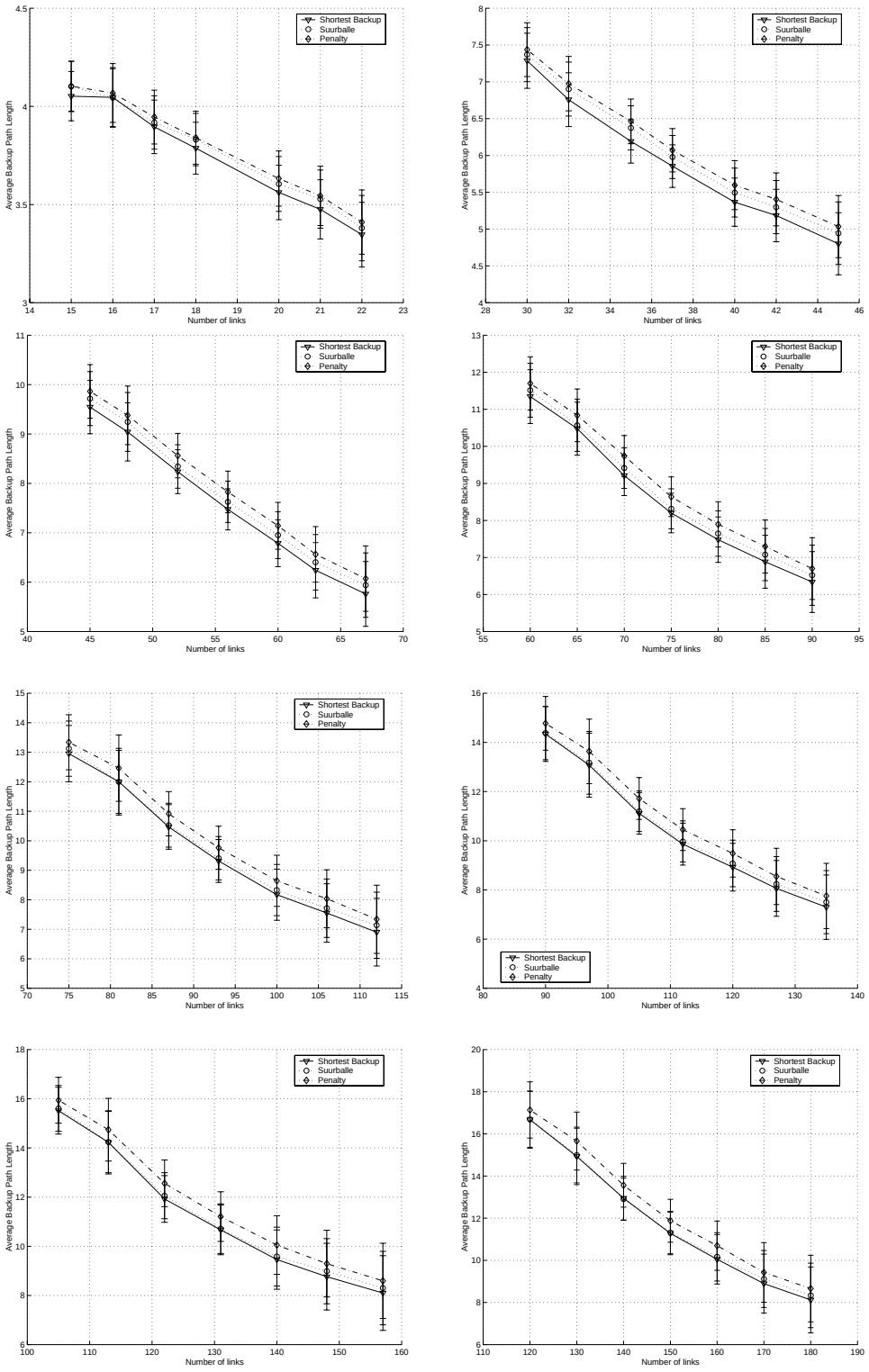


Figure 4.8: Path length statistics for random graphs of [10, 20; 30, 40; 50, 60; 70, 80] nodes

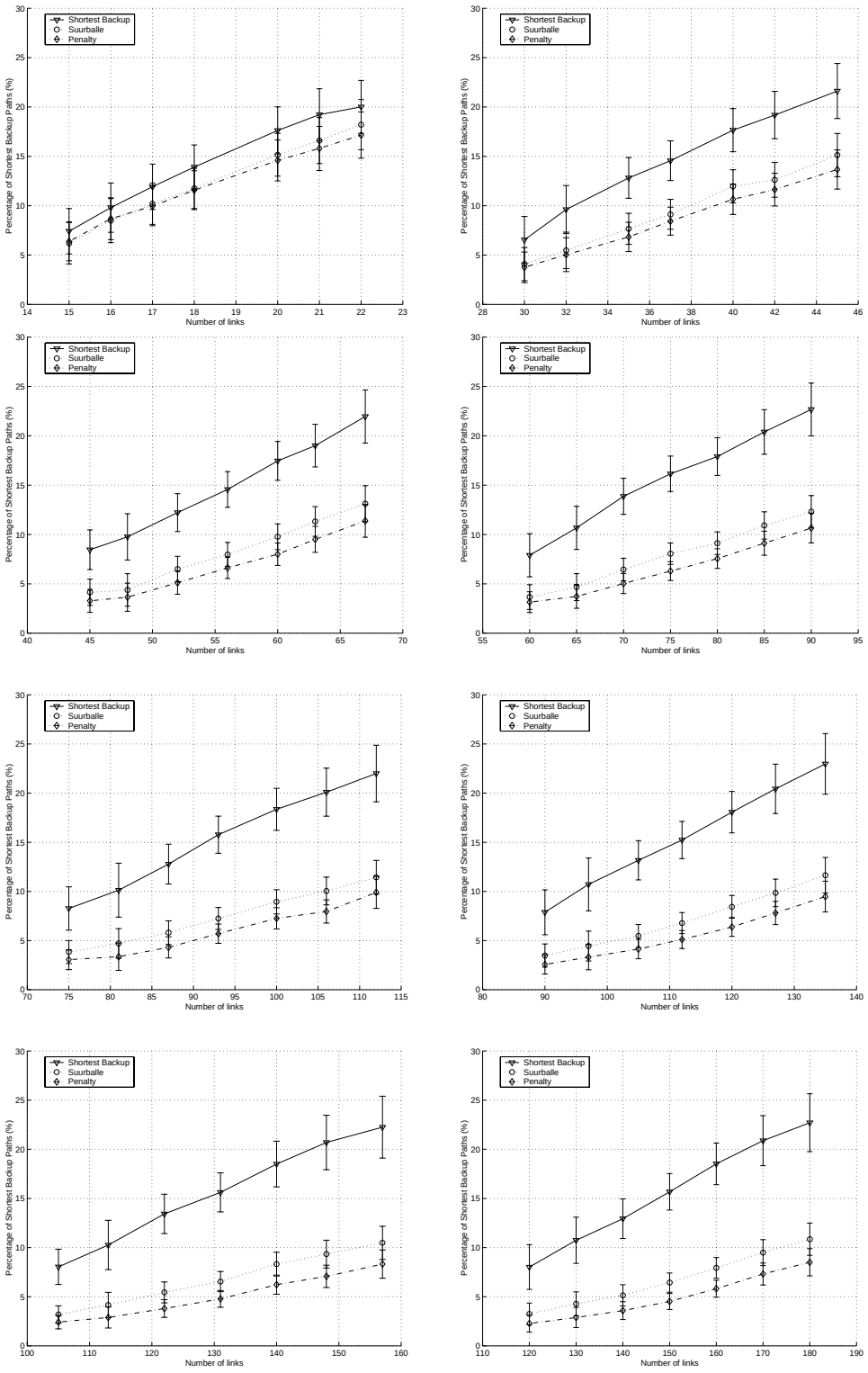


Figure 4.9: Shortest backup path probability for random graphs of [10, 20; 30, 40; 50, 60; 70, 80] nodes

4.6 Summary

This chapter concentrated on mesh-network protection architectures. We addressed the problem of computing a restoration path and a working path at the same time in a distributed way. In order to minimize resource consumption along the backup path and to be able to use non-revertive mode efficiently, we have proposed to compute two shortest paths such that the number of common edges are minimized. To protect for the failure of the common edge(s), a second backup path is computed with the Penalty method. Our method requires crankback extensions to notify the head-end router about which link was failed. The complexity of the proposed algorithm is still in the order of Dijkstra's shortest path algorithm. We have compared the performance of the proposed new algorithm with known methods from the literature. In real-world topologies we have found that the probability that a shortest secondary path can be used at failure is doubled when the Shortest Backup method is used. For the AT&T network topology we have also shown that the average secondary path length is also smaller for our method than for existing methods such as Suurballe's or Penalty method. In randomly generated networks the average secondary path length measure was nearly the same for all algorithms. However, the same positive effect of the 'Shortest Backup' method could be noticed in case of the probability measure, especially for such topologies when the graph of the tight edges has many alternative paths.

Chapter 5

Minimizing re-routing in MPLS networks with preemption-aware constraint-based routing

5.1 Introduction

The Differentiated Services and Multiprotocol Label Switching (MPLS) working groups of IETF are proposing architectural enhancements to the basic best-effort IP infrastructure that makes — among many other things — bandwidth reservation for aggregate flows a reality. In MPLS — as we will discuss in more details in Section 5.2 — LSPs, beside having a bandwidth requirement also have setup and holding priorities. All MPLS protocol components support bandwidth reservation on different priority levels. Between these levels LSP preemption is supported. If at path setup there is not enough free bandwidth available on a link, lower priority LSPs will be preempted [66]. MPLS specifies eight different priority levels, therefore it can happen that by taking another path a preempted LSP again preempts lower priority LSPs. However, there has been little work on the *dynamics of LSP preemption*. A notable exception is [64], in which Villamizar study failure scenarios and the effect of different network mechanisms on overall network performance. He specifically studies the effect of independent path re-computations, delayed re-flooding of new resource reservations, LSP priorities and subsequent re-optimization of suboptimal paths. This work suggests timing and ordering strategies for re-routing and re-optimization of LSPs after failures, but does not deal with path computation methods.

There are global optimization algorithms that consider the issue of preemption. For example, Garay and Gopal in [67] studies preemption in ATM networks. Their objective is to find the minimum number of calls that should be removed from the network in order to have enough capacity to accept a new call. This optimization

method can only be implemented in a central place, since detailed knowledge is needed about the path of each and every call in the network. Similarly, Mitra and Ramakrishnan [68] provide multicommodity flow solutions for the global LSP optimization problem which can be used for multi-priority traffic as well.

As we will discuss in more detail in Section 5.3, all proposed constrained shortest path first algorithms operate in a simple way regarding LSP priorities. When computing the path for an LSP, reservations of lower priority LSPs are not taken into account. Known CSPF methods try to maximize the success probability of future connection establishments, thus preparing for future connection arrivals.

In this chapter, we study the effect of constraint-based path selection methods on the dynamics of LSP preemption in MPLS networks. As the main contribution of this chapter, in Section 5.4 we propose new CSPF algorithms that take into account the present state of the network by considering the available resource reservation information of lower priority LSPs as well. By doing this, our algorithms aim to *minimize preemption of lower priority LSPs* and thus enhance the stability of multi-priority MPLS networks. In Section 5.4 we demonstrate that in order to have some idea of the number of affected lower priority LSPs, useful measures can be extracted from the currently flooded IGP link-state information. Based on these measures we propose two methods to select such a shortest path on which the probability of preempting lower priority traffic is the lowest. One method aims to minimize the affected priority levels, the other to minimize the sum of affected LSPs' bandwidth. Our algorithms are based on the standard Dijkstra algorithm and implement the same concept of multi-level metrics as, e.g., the widest-shortest path method [29]. This means that path selection is restricted first to feasible links (having the required bandwidth), then to shortest paths based on the link weight (or hop count) metric, and then (only if there are more than one feasible paths after these two steps), comes our metric to minimize preemption.

In Section 5.5 we compare the preemption performance of the CSPF algorithms proposed in this chapter to the most important methods discussed in the literature. To aid understanding of LSP preemption and thus the dynamics of multi-priority MPLS networks, we first show results that are general for all CSPF methods. For example, the different CSPF failure ratios and path lengths at each of the eight priority levels provides general network performance measures. Then we study in detail how the performance of CSPF algorithms differ in terms of preemption, e.g., what is the probability of preempting a lower level LSP and the amount of preempted bandwidth during an LSP establishment. Finally in Section 5.6 we summarize the work and propose future research topics.

5.2 Preemption in MPLS networks

In this Section we give an overview of the bandwidth reservation and preemption related attributes and mechanisms of MPLS. We follow the classification of traffic engineering control attributes based on RFC 2702 [32], i.e., we use the terms 'traffic trunk associated' and 'resource associated' attributes. Finally, we show how these are used by constraint-based routing to select paths for traffic trunks, and by other traffic engineering mechanisms (e.g., preemption), to achieve service differentiation.

5.2.1 Traffic trunk attributes

Besides basic traffic engineering attributes that specify 'from' and 'to' where the traffic trunk should be established and the 'bandwidth to be reserved', [32] specifies also 'preemptor' and 'preemptable' binary attributes. In RSVP [62] and LDP [61] implementations a more flexible approach was chosen in the form of 'holding' and 'setup' priorities. Setup priority specifies the importance of an LSP establishment, while holding priority specifies how important it is for an already established LSP to hold on to its reserved resources. Both priorities have a range of 0 (highest priority) to 7 (lowest priority). An LSP with higher (numerically lower) setup priority can preempt an LSP with lower (numerically higher) holding priority. To avoid continuous preemption, holding priority should never be lower (numerically higher) than setup priority. Preemption can be used also to assure that high priority traffic trunks can always be routed through relatively favorable paths e.g., on shortest paths [32].

5.2.2 Resource related attributes

Similarly to the traffic trunks related attributes, resource related attributes defined in the framework document differ from the ones realized in actual protocols. In [32] the maximum allocation multiplier (MAM) attribute specifies the proportion of the resource available for allocation to traffic trunks.

IGP extensions of OSPF [69] and ISIS [70] fulfill this requirement by flooding beside the *maximum bandwidth* also the *maximum reservable bandwidth* of a resource. The latter may differ from the actual maximum bandwidth because administrators may choose to dedicate only part of the link's bandwidth to traffic engineering. Alternatively, in order to exploit statistical multiplexing gain the reservable bandwidth may exceed the actual bandwidth of the link.

The most important IGP extension to enable preemption is the one in which actual resource reservations are distributed. In [69], [70] *unreserved bandwidth* ($\mathbf{B}_u = (B_{u0}, B_{u1}, \cdots, B_{u7})$) is specified as the amount of bandwidth not yet reserved at each of the eight priority levels on a specific link. This is counted in an

accumulative way i.e., if a highest priority LSP is established, all elements of this vector will decrease.

All flooded resource information is stored in a traffic engineering database (TE-DB) of label edge routers. The TE-DB is in turn used by constraint-based routing to select paths.

5.2.3 Constraint-based routing

As mentioned in the introduction the aim of CSPF in MPLS networks is to automate the path selection for LSPs. To achieve this, operators should configure resource and traffic trunk attributes and deploy a simple path selection algorithm that matches these attributes against each other. In MPLS the simplest bandwidth constrained CSPF works as follows:

Considering an LSP with setup priority s and bandwidth requirement B_{LSP},

1. in the router's TE-DB mark all links as 'invalid' where the link's unreserved bandwidth at the priority level of the LSP's setup priority is less than the LSP's bandwidth requirement ($B_{us} < B_{LSP}$),

2. run Dijkstra's shortest path algorithm on the graph composed of the links not marked as 'invalid'. The resulting path (if there is any) will be the LSP's path.

The above algorithm, when looking at the B_{us} level, treats lower level reservations as if they were not present, therefore, when the LSP's PATH message traverses the explicit path, preemption decision and admission control is needed at each hop. The former determines which lower priority LSPs will be preempted. The latter is essential, since when link-state information is inaccurate, there may not be enough unreserved bandwidth at the LSP's priority level.

5.2.4 Admission control and preemption decision

The admission control function at each router checks whether there is enough unreserved bandwidth to support the setup of the new LSP. In the case that there is enough totally unreserved bandwidth at an interface, no preemption is needed and admission is allowed. If this is not the case, the setup priority in the PATH message is taken and a check is made to see if established lower priority LSPs can be preempted. If $B_{us} < B_{LSP}$ there are not enough preemptable LSPs so the PATH message is refused. Otherwise, as many lower holding priority LSPs are preempted as needed.

The selection of the LSPs to be preempted is a local matter. Local preemption strategies for two priorities have been studied by Peyravian and Kshemkalyani in

[71]. In the rest of this chapter we use an extension of their *Min_Conn* algorithm to support the eight priority levels of MPLS:

- always preempt the lowest priority LSPs first (i.e., until there are LSPs on the 7^{th} priority level, LSPs can not be preempted from the 6^{th} level),
- minimize the number of preempted LSPs inside a single priority level (greedy selection of always the biggest LSPs as long as more than one LSP should be preempted),
- perform a bandwidth-wise optimal selection of the last preempted LSP (e.g., if 5Mb/s should be preempted at the last step, preempt the smallest LSP among the ones that are bigger than 5Mb/s).

An LSP may preempt more LSPs when moving downstream. For the preempted LSPs, a notification is sent upstream. Preemption does not differ from any other kind of failure, so the head-end tries to re-route the LSP. Re-establishment of preempted LSPs may cause further preemption, so a preemption chain may start.

5.3 Previous work on bandwidth constrained path computation methods

In this section we give an overview of bandwidth constrained path selection methods proposed in the literature. Since most of the proposed methods can be realised as an extension to Dijkstra's well known shortest path first algorithm, we use the terminology of [51] to describe differences between the algorithms presented here. To make path selection bandwidth constrained, all methods start by pruning non conforming links based on the bandwidth check $B_{us} < B_{LSP}$ (B_{LSP} is the required bandwidth, s is the setup priority of the LSP and B_{us} is the unreserved bandwidth at the s level). In [51] this operation is described with an *acceptor function*.

The Dijkstra algorithm aims to find the best candidate path to a node. At each step two paths are compared with the help of a *comparator function* [51]. At this point, the metrics used have an important role. In the case of single or mixed metrics the comparator function simply evaluates which path's metric is smaller. In the original Dijkstra algorithm this means the simple arithmetic comparison of two real numbers (weights or 'distances'), but for more complicated metrics this comparison can be more advanced.

To provide more path optimization possibilities besides a simple shortest path selection, many algorithms use the concept of multiple metrics [72] which can be easily incorporated into Dijkstra's algorithm. Simply, when comparing the metrics of two links or paths the initial comparison is done according to the first metric (e.g., the configured link cost or hop count as a metric), and in case of equality,

the value of the second-level metric brakes the tie (i.e., the available bandwidth in case of the widest shortest path algorithm).

To make the picture complete, for each metric an *accumulation function* [51] defines how the metric is accumulated along a path. When we know the 'distance' to a node and also we know the cost (metric) of the edges starting from that node to the neighbors, we use this function in the Dijkstra algorithm to determine the candidate distances to the neighboring nodes. In an accumulation function, depending on the type of metric, different operations are done. Hop count or cost is an *additive* metric, so simply the link costs should be summed to get the path cost. Bottleneck bandwidth is the maximum of metrics along a path, so this is a *concave* metric.

A short overview of path computation algorithms follows this introduction. At the end of this section we present a summary table (Table 5.1) and published simulation results comparing some of the algorithms discussed below.

Guerin, Williams, Przygienda, Kamat and Orda [29] propose OSPF extensions to support QoS routing. They propose the *widest-shortest* path algorithm for the bandwidth-constrained routing problem. The main idea of this algorithm is to first prune links without sufficient unreserved bandwidth and then compute a shortest path on the remaining graph. When several shortest paths are available, the preference is for the path whose bottleneck link unreserved bandwidth is maximal. This strategy aims at using minimal amount of network resources and at balancing load.

The *residual bandwidth ratio* method (RB-CSPF) [51] selects the bandwidth constrained shortest path for an LSP such that being the second metric (instead of the bottleneck link unreserved bandwidth), it takes into account the bandwidth that is still unreserved after the LSP setup, normalized by the total reservable bandwidth. $(B_{us} - B_{LSP})/(B_{max})$ where B_{max} is the total reservable bandwidth on the link. Moreover, instead of treating only a single bottleneck link, [51] stores and uses a configurable number of bottleneck links (e.g., four).

The WSPF and the RB-CSPF methods both aim at balancing load. There have been many load-based routing algorithms proposed in the past for different networks [43], [44], [C2]. All such algorithms are common in their basic idea to prepare for future call arrivals by routing traffic on longer paths then the topological shortest one. We discuss here the method of Kodialam and Lakshman [44]. They introduced the concept of *minimum interference routing* for computing bandwidth constrained paths for LSPs. Their proposal is a load based approach which exploits information about where traffic enters and leaves the network (ingress and egress points). This information is used when computing the maximum flows between all possible ingress and egress points to determine so called critical edges. Critical edges have the property that whenever an LSP is routed over them, the maximum flow value of one or more ingress-egress pairs decreases. The weight for

a link is determined by counting, for how many ingress-egress pairs this edge is among the critical edges. Weights are used at the subsequent Dijkstra shortest path calculation in the standard additive way.

In [31] Shaikh, Rexford and Shin proposes a load-based additive link-cost metric as a second metric in the Dijkstra algorithm. They propose this to be used after the original link-cost metric which restricts path selection to shortest paths. Their link-cost metric for link i has the form:

$$c_i = \begin{cases} 1/C, & r_i/R_i > u_{min} \\ \frac{\left\lceil \left(\frac{r_i}{R_i} - u_{min}\right)^{\alpha} (C-1) \right\rceil + 1}{C}, & otherwise \end{cases}$$

where r_i/R_i is the normalized load of the link. Parameter u_{min} is the minimum cost utilization level; any link utilization below this value is considered to have the minimum cost. The exponent α controls how expensive heavily-loaded links are judged relative to lightly-loaded links, and finally the constant C determines how many discrete cost values are used for link-costs. The link-cost are used in the Dijkstra shortest path calculation in the standard additive way. Examples in [31] show the effects of different α, C and u_{min} parameters on the shape of the link-cost function and on the performance of the network.

In [73] Shaikh, Rexford and Shin proposes bandwidth constrained path selection for long-lived IP flows. Their path selection method is a widest shortest path method. If more shortest paths can support the required bandwidth, the 'widest' rule selects the path for which the unreserved bandwidth of the bottleneck link (i.e., the smallest unreserved bandwidth value on any of the links in the path) is the largest. However, pruning of bottleneck links are done after the Dijkstra computation. This can result in a situation where none of the topological shortest paths can support as much bandwidth as requested. For this reason, [73] permits the use of non-minimal paths whose hop-count is one more then the shortest path's hop-count. This is similar to the dynamic-alternative routing (DAR) concept used for PSTN networks [74].

In [72] Wang and Crowcroft studied the complexity of the multi-constraint QoS routing problem. They proposed the *shortest-widest* path algorithm, i.e., their first constraint is to find a path with maximum bottleneck bandwidth, and when there are more than one such widest paths, choose the one which is the shortest. The authors of [72] point out that non-bottleneck links have no effect on widest paths. Consequently, for a given topology there are usually many equal width widest paths.

Ma, Steenkiste and Zhang in [75] provide an algorithm that can dynamically balance the impact of hop count and path load. The so called *shortest-distance* algorithm uses the following distance function for path P:

$$dist(P, n) = \sum_{i \in P} \frac{1}{r_i^n}$$

Table 5.1: Metrics and ordering in different routing methods

	1^{st} metric	2^{nd} metric
widest-shortest path method [29]	shortest	bottleneck bandwidth
residual bandwidth ratio method [51]	shortest	residual bandwidth
minimum interference routing [44]	complex mixed metric	—
discrete link cost method [31]	shortest	load based mixed metric
routing for longlived flows [73]	shortest+1	bottleneck bandwidth
shortest distance method [75]	tunable mixed metric	—
shortest-widest path [72]	bottleneck bandwidth	shortest

where r_i is the max-min fair rate of a link i. Instead of the fair share, however, we may use the unreserved bandwidth of link i in a reservation based routing model. Variable n can be used to tune the algorithm, i.e., by choosing $n = 0$, we get the shortest path, and if $n \to \infty$, we get the widest path. The experiments in [75] show that $n = 1$ provides the best average throughput for the studied max-min fair share networks.

Ma and Steenkiste in another work [76] compare path selection algorithms for traffic with bandwidth guarantees. The 'widest-shortest' [29], the 'shortest-widest' [72], the 'shortest-distance' [75] and a variant of the 'dynamic-alternative routing' [74] methods were simulated. The experiments showed that algorithms limiting the hop count and as a result resource consumption — e.g., the 'widest-shortest' path and the 'dynamic-alternative routing' method — provide good results when the network is becoming overloaded. On the other hand, algorithms aiming at balancing the network load — the 'shortest-widest' and the 'shortest-distance' algorithm

— perform better in light and medium load situations. Recently, authors of [44] showed that 'minimum interference routing' outperforms both simple 'minimum-hop' routing and 'widest-shortest' path routing. The price of better results is, however, the complexity of the proposed algorithm compared to simple Dijkstra based methods.

5.4 Proposed preemption-aware CSPF methods

In this Section we propose preemption-aware CSPF methods that aim at minimizing unnecessary preemption, thus enhancing the stability of MPLS networks. First in Section 5.4.1 we derive new measures to estimate how much preemption will occur on a path at LSP establishment. Then in Section 5.4.2 and 5.4.3 we show that these measures can be used to construct new metrics and CSPF algorithms. Basically, all of our new algorithms restrict path selection to shortest paths that have the required bandwidth. Preemption minimization is only used as a secondary objective.

5.4.1 Preemption measures

To have practically relevant results technological limitations present in MPLS networks must be taken into account. Currently proposed IGP extensions [69], [70] provide only *summarized information* about the reserved resources. This means that they do not provide per-LSP information on distant links, i.e., neither the number of LSPs nor their bandwidth values are available. This summarized information is sufficient for CSPF route computation, i.e., to tell if a link has the required resources to accommodate a new LSP on a certain priority level. However, it is insufficient for determining how many and how big LSPs will be preempted due to a setup of the new connection. Moreover, when traversing more links, we can not tell whether e.g., two times 20 Mb/s will be preempted or only once. The latter case occurs when LSPs preempted on the first link automatically free resources on the second link, thus eliminating the need for further preemption.

Although detailed LSP information is not available for the path computation, it is still possible to develop *heuristic methods* for minimizing the volume of preemption, strictly based on the currently available standard IGP extensions. Our goal in the following is to propose and evaluate such solutions.

As we have already mentioned in Section 5.2, traffic engineering extensions to link-state IGPs [69], [70] distribute the following reservation-related information for each link in the network:

- link capacity (i.e., maximum bandwidth), B_{MAX},
- maximum reservable bandwidth, B_{max},

- unreserved bandwidth vector, (unreserved bandwidth at each priority level), $\mathbf{B}_u = (B_{u0}, B_{u1}, \cdots, B_{u7})$.

Free bandwidth

In order to implement priority-aware CSPF algorithms in a Label Edge Router, we have to identify useful preemption measures. It is easy to see that if we are given a link and an LSP to be established on it (for which $B_{us} < B_{LSP}$ holds). The bandwidth that should be preempted on the link will be larger if we have smaller amount of free bandwidth (unreserved bandwidth on the 7^{th} priority level: $B_{free} = B_{u7}$). By using free bandwidth as a measure we actually achieve the simplest preemption-aware widest path selection. This allows us to take into account lower priority reservations and to minimize the amount of bandwidth preempted by the new LSP.

Per priority preempted bandwidth

Besides the above discussed measure we show that per-priority preempted bandwidth also has an important role. Let's define $\mathbf{B}_p$ as the vector of bandwidth values that should be preempted at each priority level of a link by the new LSP:

$$\mathbf{B}_p = (B_{p0}, B_{p1}, B_{p2}, \cdots, B_{p7}) \tag{5.1}$$

The *bandwidth preemption vector* provides us useful information. With the help of it, we can derive for example which is the *highest affected priority level* on a link, i.e., the smallest such number i for which $B_{pi} \neq 0$. As we have already discussed, this measure is not included in the information flooded by the IGP. However, we can derive a good *estimate* by using the original parameter set. Fig. 5.1 shows a procedure for this estimation (considering the preemption strategy defined in Section 5.2.4). In the next Section we show an example case where this procedure is used.

Example

Let us demonstrate the calculation of the preemption vector with an example. Suppose we have a 100Mb/s link with the following unreserved bandwidth vector: $\mathbf{B}_u = (100, 100, 100, 80, 40, 20, 20, 20)$ and an LSP with $B_{LSP} = 70$ Mb/s bandwidth requirement and $s = 3$ setup priority. Since $B_{u3} = 80$, the LSP can use this link. If this link will be included in the LSP's path, preemption will occur. The per-priority preempted bandwidth values are calculated based on the procedure on Fig. 5.1. Let us follow the *for* loop. For priority 7, $B_{u7} = 20$ is smaller than B_{LSP}, therefore the *minimum* operator is used. The second term of the *minimum* operator reflects the fact that there are no LSPs at this level, so $B_{p7} = 0$. Similarly,

```
procedure CalcBwPreemptionVector(B_u, B_LSP)
    B_p = 0
    B_u(-1) = B_max
    for (i = 7, i ≥ 0, i--)
        if B_LSP ≤ B_ui return B_p
        B_pi = min((B_LSP − B_ui), (B_u(i−1) − B_ui))
    end for
    return B_p
end procedure
```

Figure 5.1: Calculating the bandwidth preemption vector

$B_{p6} = 0$. For $i = 5$ the second term determines that 20Mb/s should be preempted on the 5^{th} priority level. On level 4, there is altogether 40Mb/s ($B_{u3} - B_{u4}$), but since our new LSP has already preempted 20Mb/s at the previous level and 20Mb/s was free on the link, at this level only 30Mb/s should be preempted. This is guaranteed by the first term in the *min* operation. At the next cycle ($i = 3$) the procedure will exit after the check of the *if* operation. So, finally the bandwidth preemption vector will be: $\mathbf{B}_p = (0, 0, 0, 0, 30, 20, 0, 0)$.

Now let us take the three links below and show how the discussed measures — free bandwidth and bandwidth preemption vector — can be used to decide which is the more desirable link to be used in path selection. The first link is the one for which we have calculated the bandwidth preemption vector above. For the other two, we just show the bandwidth preemption vector (the original unreserved bandwidth levels can be calculated from this).

$$\mathbf{B}_p^1 = (0, 0, 0, 0, 30, 20, 0, 0), \qquad B_{u7}^1 = 20Mb/s$$
$$\mathbf{B}_p^2 = (0, 0, 0, 0, 0, 10, 30, 30), \qquad B_{u7}^2 = 0Mb/s$$
$$\mathbf{B}_p^3 = (0, 0, 0, 0, 0, 20, 20, 30), \qquad B_{u7}^3 = 0Mb/s$$

By looking at the affected priority levels, we can see that on the first link there is bandwidth to be preempted on the lowest 4 priority levels, while on the second and the third only the 5^{th} level is affected. It may be important to use such links for path setup on which only lower priority levels are affected. We can also see from the bandwidth preemption vector that the second link has less affected bandwidth on the 5^{th} priority level than the third one. This makes the second link more desirable than the third one.

However, if we consider free bandwidth, we can notice that on the first link there is 20Mb/s free bandwidth, while on the other two links there is no free bandwidth at all. Therefore our *free bandwidth* measure indicates that the first link is more desirable.

5.4.2 Priority-aware CSPF metrics

Based on the above measures ($\mathbf{B}_p$ and B_{free}) we construct preemption minimization metrics that can be used in CSPF algorithms. We define how the above derived preemption measures are used as path metrics in the Dijkstra algorithm with the help of *comparator* and *accumulator functions*.

Maximize free bandwidth

By maximizing *free bandwidth* (B_{sum}) as a link metric, we aim to preempt the fewest possible lower priority LSPs in terms of bandwidth sum, and among those paths on which no lower priority LSPs are preempted we choose the widest one. Since bandwidth is a concave metric we define an accumulator function in which the bottleneck link's free bandwidth determines the path's metric. The comparator function is defined in such a way that it chooses a path with larger free bandwidth as a better candidate path.

Minimize affected priority levels

When using the bandwidth preemption vector ($\mathbf{B}_p$) as a link metric, we aim to minimize the affected priority levels by preferentially preempting low priority LSPs. We define the comparison operation for this metric as follows:

- $\mathbf{B}_p^1 < \mathbf{B}_p^2$ iff for their first (from 0) different coordinate with index i, $B_{pi}^1 < B_{pi}^2$ (which is actually a lexicographic comparison).

With this definition the comparator function actually implements two tie-braking concepts. If two paths have different highest affected priority levels, the choice is for the path on which the highest affected priority level is smaller. But if the affected levels are the same, selection of the candidate path is done by selecting the path which has smaller preempted bandwidth on the highest affected priority level.

In order to minimize the *affected priority levels* ($\mathbf{B}_p$) along the path of the LSP setup, two kinds of accumulator function can be defined. As already mentioned, from the flooded bandwidth reservation information we do not know whether the same set of LSPs are carried on two consecutive links. If we assume that they are, we can say that $\mathbf{B}_p$ is a concave metric. However, if we assume it to be more probable that the LSP sets differ on consecutive links, we have to minimize the sum of the preempted per priority bandwidth on each link of the new LSP's path. Based on these different assumptions we define two variants of the accumulator function:

- Concave case: the larger $\mathbf{B}_p$ vector is taken as the path's metric,

- Additive case: the elements of the $\mathbf{B}_p$ vector are added one by one to find the path's metric.

When no preemption is needed we would like to use the widest path. We achieve this by incorporating the free bandwidth as a last (9^{th}) element in the bandwidth preemption vector. To be able to use the above defined comparator function, we use $-B_{u7}$ as the 9^{th} element in $\mathbf{B}_p$ (preempted bandwidth is to be minimized, while free bandwidth is to be maximized).

5.4.3 Priority-aware CSPF algorithms

When reviewing CSPF algorithms proposed in the literature, we saw that the order of metrics in the comparator function has huge significance, e.g., widest-shortest and shortest-widest algorithms performed differently. We propose such an ordering for our new metrics that minimizes preemption without adversely affecting the CSPF success ratio, and path length of high priority LSPs. To achieve this, in both algorithms we first prune links for which $B_{us} < B_{LSP}$. (s is the setup priority of the LSP for which the path is calculated). On the resulting graph we restrict path selection to shortest paths based on the original OSPF metric. By using the link cost as the first metric in the comparator function of the Dijkstra algorithm, we achieve the result that the LSPs are always routed on shortest paths irrespective of lower priority traffic. We utilize preemption information only after this, i.e., when selecting a candidate path among otherwise shortest feasible ones, as summarized in Table 5.2.

Table 5.2: Proposed algorithms

	1^{st} metric	2^{nd} metric
maximize free bandwidth	shortest	free bandwidth
minimize affected priority levels	shortest	bandwidth preemption vector

The overall flow chart of the algorithm is depicted in Fig. 5.2. Note that the two boxes at the right are joined to the main action flow with round arrows. This means that e.g., when the algorithm initializes the graph it calls the CalcBwPreemptionVector function for each link. Similarly the proposed comparator and accumulator functions are used in the Dijkstra algorithm instead of the original arithmetical comparison and addition operators.

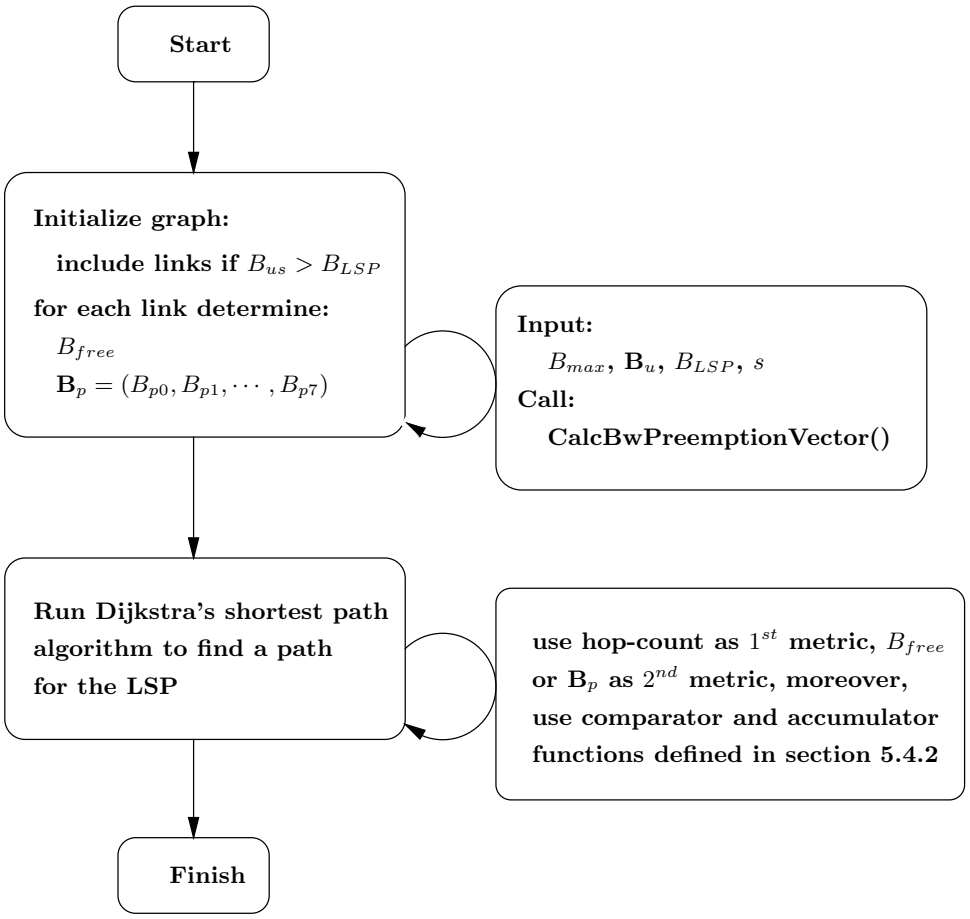


Figure 5.2: The flow chart of the algorithm

5.5 Numerical results

We have conducted numerical investigations in order to show the real improvements resulting from the use of our algorithms. Section 5.5.1 starts with the description of the used performance evaluation methodology, then in Section 5.5.2 we describe the simulation environment and survey the implemented algorithms and measurements. Finally in Section 5.5.3 we introduce our simulation experiments. Our goals with the numerical evaluation were twofold:

- first, we wanted to get some insight to the preemption process, and its performance effects on LSPs having different priority levels.
- second, we wanted to quantify the actual performance gain resulting from the use of our proposed algorithms.

5.5.1 Performance evaluation methodology

Several approaches are used for the performance evaluation of routing strategies. For example, Plotkin [43] proposes an analytical approach for this purpose. Another method is to use discrete event simulation, assuming a stochastic process for the arrival of demands [C2].

In our case, it was hard to find a tractable analytical approach that can be used to obtain practical results. We were also lacking a well-established and justified LSP demand arrival model for MPLS. Therefore we have decided to use flow-level simulation for our performance evaluation purposes and omit the time factor from our simulations.

5.5.2 Simulation model

Network and traffic model

In order to have practically relevant results we carried out our first groups of simulations on the *Cable & Wireless* backbone network topology. This was selected from many ISP topologies [77], since for this network not only the topology, but also the link capacity information was also available. This real life data network contains 31 backbone nodes and 102 links, resulting in an average node degree of 3.29. The link capacities vary between DS-3 and OC-192, with the majority of links having OC-12 capacity. The exact topology and capacity values of this network can be found in Appendix A. The topology was fixed throughout the simulations, i.e., we did not consider link or node failures. Furthermore, we supposed that demands are generated from every node to every other node. Most of the results in this chapter were derived with the help of the Cable & Wireless network topology. However, in section 5.5.3 we also show results of random network simulations.

Since it is hard to get traffic statistics from real MPLS networks, we created randomly generated traffic situations. This means that we loaded the network to a certain extent with randomly placed LSPs having random bandwidth values and ingress egress points. LSP bandwidth is uniformly-distributed within the interval $(0, B_{MaxLSP}]$. In our experiments we focus on such situations when B_{MaxLSP} is around 8-10% of the most common OC-12 link capacity. The priority levels of the LSPs were also set randomly with a uniform-distribution in the range [0-7].

After loading the network to a certain level, we randomly generated several new LSPs, and tried to route them with the selected CSPF method. If load increased above the required level after a successful LSP setup, we randomly released some LSPs from the network, until we returned to the operation point used in the given simulation experiment. This test has been conducted under different traffic situations resulting in a series of probability estimates describing the quality of the routing strategy at different points. Numerous measurements have been taken at each point in order to ensure acceptable confidence intervals. In fact, in all of

these graphs confidence intervals are below resolution and so, for the sake of better visibility are not shown the figures. At the end of this section, however, we show two plots with confidence intervals for reference.

We characterize a traffic situation by the *total throughput*, i.e., the sum of all established LSPs' bandwidth. We use total throughput as a measure instead of average link utilization on the x-axis, since we believe that carried traffic is more important to operators than link load. At a given average link load, all CSPF algorithms most probably have the same CSPF failure ratio. However, ineffective path selection at an early stage results in longer paths for LSPs routed afterwards. This causes higher average link loads at the same amount of carried LSP volume, which in turn results in noticeably higher CSPF failure ratio when evaluated based on the same total throughput.

Since our main area of interest is path selection we did not implement protocol modeling in packet level at our simulations. As an important simplification, we did not model the exact operation of the link-state update generation and flooding protocol. Instead, we assumed that edge routers have accurate bandwidth reservation information about the network.

Compared algorithms

We investigated the performance of the proposed priority-aware CSPF algorithms and compared it to the most promising CSPF algorithms selected from those surveyed in Section 5.3. According to [32], multiple priority levels and preemption among them can be used to assure that high priority traffic trunks are always routed through relatively favorable paths (i.e., shortest paths). This suggested that we should concentrate only on such algorithms that use strictly shortest paths. Therefore, all simulated algorithms are common in two points:

1. links not having enough unreserved bandwidth at the priority level of the LSP are pruned.

2. In the comparator function the first metric is always the OSPF/ISIS weight. Optimization based on other measures are considered only as a second metric.

The simulated algorithms proposed in the literature are:

- The basic *shortest* path first algorithm has been selected for reference. It performs a random selection among equal cost paths, therefore we are able to measure the gain of other CSPF algorithms to a base algorithm.

- The *widest-shortest* algorithm (WSPF) [29] has been selected because by performing load-balancing inside a priority level (irrespective of lower priority levels), it most probably provides the best service to high priority LSPs. We wanted to measure whether our preemption-aware strategies result in performance degradation for higher priority levels.

- The *residual bandwidth ratio* method [51] has been selected, because we were interested in whether it improves the performance of the 'widest-shortest' path method, by keeping track of k bottleneck links. For our simulations we used $k = 4$ for the tunable parameter.

- The *discrete link cost* method [31] has been selected because it was the most promising among the algorithms that include a load dependent metric. In the simulations we used the following settings: $C = 10, \alpha = 4, u_{min} = 0.1$.

Moreover, our two preemption aware algorithms were implemented:

- The *maximize free bandwidth method* is basically a widest-shortest path method, taking into account the unreserved bandwidth on the lowest priority level.

- The *minimize affected priority levels method* was implemented with the help of the bandwidth preemption vector measure. We have incorporated the free bandwidth measure as the last element. Therefore, at light loads when no preemption is needed, a widest path selection is performed based on the free bandwidth.

Performance metrics

We compared the behaviour of the implemented algorithms with the help of the following empirical measures:

- *Success ratio:* measure of CSPF path computation effectiveness. This measure provides information about how many path computation attempts failed due to CSPF not being able to find a feasible path for the LSP with the required bandwidth. In our experiments the graph topology used for CSPF always matches the actual network topology, therefore, once CSPF finds a path, the LSP setup attempt will always be successful.

- *Success ratio per priority:* Path setup success ratio differences are measured for the eight priority levels. In CSPF different priorities see different unreserved bandwidths thus one can suspect that the path computation of higher priority LSPs will be more successful than lower priority LSPs.

- *Preemption ratio:* The probability that at least one lower priority LSP is preempted during an LSP establishment. We measure the amount of preemption for all LSPs that were successfully established, i.e., there was no CSPF failure.

- *Average preempted bandwidth:* The average bandwidth of preempted lower priority LSPs during an LSP establishment. Again, we only include LSPs with successful bandwidth-constrained path computation. We include all LSP's bandwidth in the preemption chain.

- *Distribution of preempted LSPs between the priority levels:* In the nominator we count how many times an LSP with a given priority has been preempted. In the denominator we have the total number of preempted LSPs.

- *Path length per priority:* The average LSP path length for each priority level.

5.5.3 Performance evaluation

The effects of preemption on previously proposed CSPF algorithms (e.g., the widest-shortest path method) are presented in this section followed by the experimental results for the priority-aware CSPF algorithm.

General preemption effects

In our first experiments plots for the 'widest-shortest' and the simple 'shortest' bandwidth-constrained path selection algorithms were generated. With the help of these simple algorithms we demonstrate the general effects of preemption on the performance of LSPs with different priority levels.

In Fig. 5.3 we can see the success ratio seen by the eight priority levels (priorities are represented on the z-axis, with '0' representing the highest priority, and '7' the lowest one). As we have discussed before, success ratio in our case is the inverse

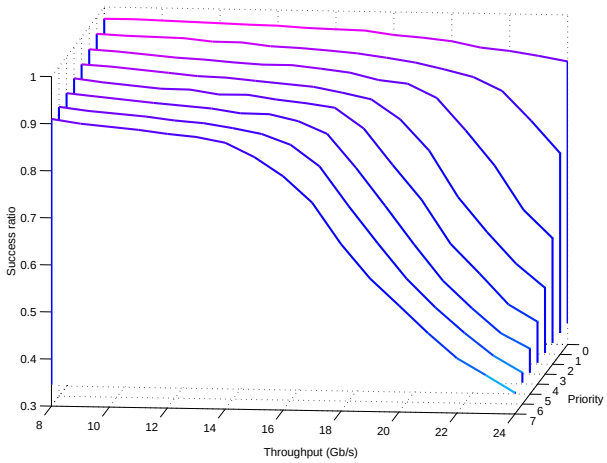


Figure 5.3: Success Ratio per priority (for the widest method)

of CSPF failure ratio. It can be seen that path selection failure increases at much smaller total throughput levels for the low priority LSPs than for higher ones. The reservation differentiation concept with (i) multiple priority levels, (ii) link pruning and (iii) cumulative calculation of unreserved bandwidth values is successful in ensuring that the reserved resources of higher level LSPs cannot be used by lower level LSPs. The effect of this can be seen on the increased CSPF failure ratio of lower priority LSPs.

We have also determined the overall LSP establishment success ratio and compared the performance of two CSPF algorithms. We realised that 'widest-shortest' path selection method improves LSP establishment success probability. CSPF failure is higher for the simple 'shortest' method, since this method may block links at an early stage, which forces LSPs arriving later to use longer paths. The difference between the 'widest-shortest' and the 'shortest' method in terms of success ratio is around 3% (determined with the help of the numerical data).

Per priority success ratio provides information about such LSPs that are to be established. However, lower priority LSPs are not only affected by CSPF failure, but also by being preempted by higher level LSPs. To gain some insight into which priority levels were preempted, both the total number of preempted LSPs and the number of preempted LSPs at each priority level were counted. From these figures we determined the distribution of preempted LSPs between the priority levels, as shown in Fig. 5.4. Highest priority LSPs can never be preempted, explaining why 'priority 0' is not shown in the figure. LSPs with 'priority 1' are on the next level and thus these can be only preempted by level 0 LSPs. However, before these relatively high priority LSPs are preempted on a link, the local preemption

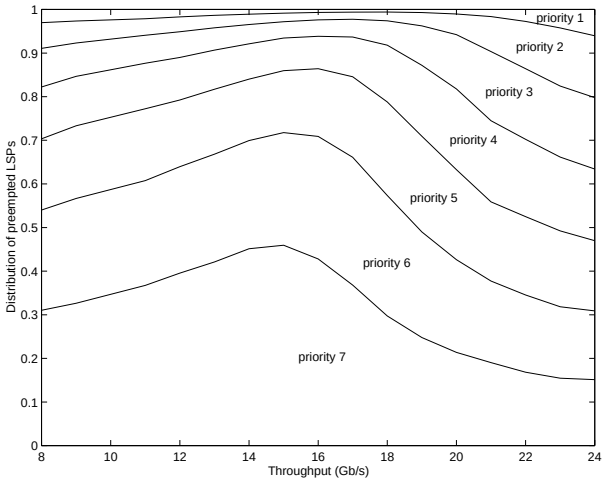


Figure 5.4: Distribution of preempted LSPs between the priority levels (for the widest method)

algorithm discussed in section 5.2.4 always tries to preempt lower levels. The probability that among all preempted LSPs a 'level 1' LSPs is preempted, only increases when practically almost all lower levels are removed from network links, and traffic is dominated by 'level 0' and 'level 1' LSPs. In contrast, at low total throughput values, 35-45% of the preempted LSPs are 'priority 7' LSPs.

It is interesting to examine the effect of preemption on the path length of LSPs shown in Fig. 5.5. In the case of constraint based routing, path lengths are also influenced by the bandwidth constraints, i.e., the link loads. In Fig. 5.5 we can observe that at light link loads, for all priority levels the path length is around 3.2 hops. As link load increases first the path length increases, then it starts to decrease. The basic reason for this is the following: in Fig. 5.3 when for a

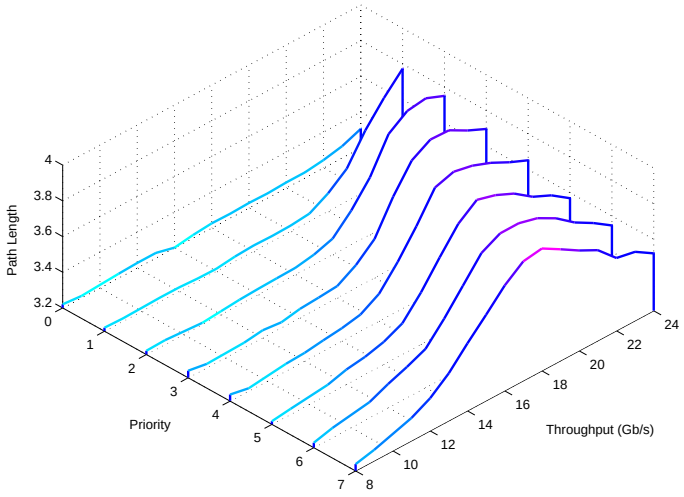


Figure 5.5: Path Lengths of different priority levels (for the widest method)

priority level CSPF failure increases, it means that it is hard to find a feasible path for LSPs. At this stage, most probably the topologically shortest paths do not have enough unreserved bandwidth, so only longer bypass paths can be used. Typically, for all priority levels, when success ratio decreases to 60-80%, the path length increases. However, after a given load level, preemption affects path length since it is more probable that LSPs established on long paths will be preempted. At the same time LSPs having their source and destination nodes closer to each other will have a bigger chance to be established successfully. Therefore, after a critical level, path length of established LSPs decreases.

Impact of preemption minimization

In this section we present the main preemption performance differences between CSPF algorithms. In the plots we name the algorithms based on the used metrics.

Since the first metric is the fixed IGP metric for all algorithms, we omit this in order to have shorter names. With this concept, in the figure keys we use 'random', 'widest', 'residual bw' 'discrete link cost', 'max free bw' and 'min affected levels' (the order is the same as in Section 5.5.2). We use the term 'random' for the simple Dijkstra algorithm because without a second level metric it does a random selection among equal cost paths. As we see in Fig. 5.6, the most important measure, the overall LSP establishment success ratio is roughly the same with all methods. The difference between 'widest' and 'random' methods are much larger than differences between the proposed preemption minimization methods and previously proposed CSPF methods. This means that the probability that the

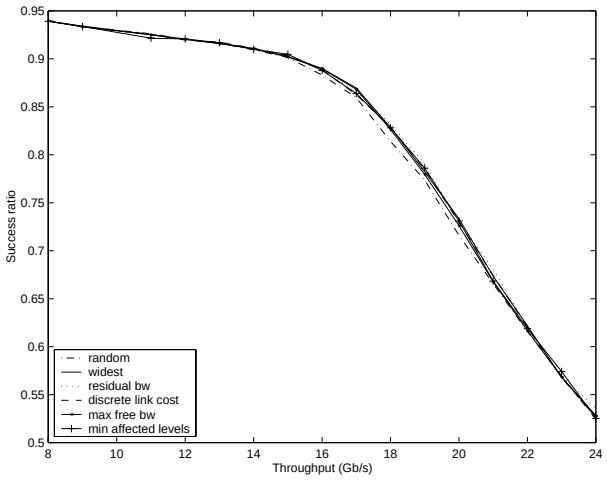


Figure 5.6: Success ratio of LSP establishment

LSP can indeed be established on the computed path is the same with our proposed preemption minimization methods as with e.g., the widest method. From this we could deduce that when we use our preemption measures and try to avoid links on which preemption is probable, we actually balance load in a similar way as the widest path and other load based CSPF algorithms.

To quantify the gain in preemption minimization, we used *preemption ratio* and *average preempted bandwidth* as basic measures. In Fig. 5.7 it is shown that the probability of preemption is significantly lower for our proposed methods. Moreover, it can be seen that our strategy to minimize the affected priority levels is more effective than the simpler one that maximizes the bottleneck free bandwidth of the path. At high loads the former achieves 10%, while the latter 5% improvement compared e.g., to the 'widest' method. At light loads, which better reflect the normal operational range of a typical network, both strategies decrease the preemption ratio by approximately 15% which is a significant improvement.

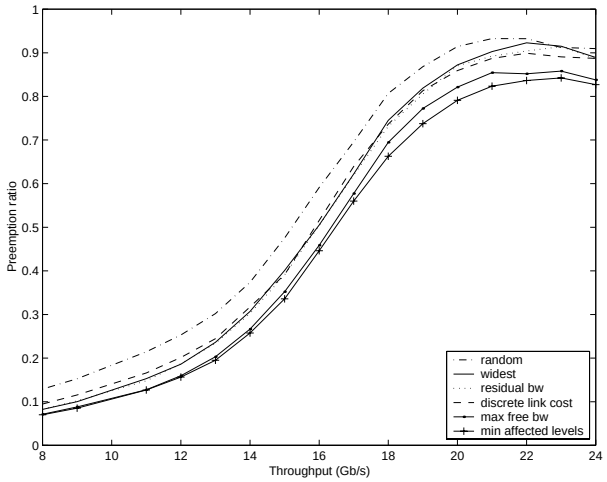


Figure 5.7: Preemption ratio

When preemption occurred we measured the average number of affected LSPs (Fig. 5.8). This includes the directly affected LSPs and also the LSPs preempted by the preempted LSPs that were re-established (chain effect). We found that when preemption occurs the number of LSPs in the preemption chain does not differ significantly for the different methods. Consequently, we can say that the main benefit of using our method can be found in decreasing the probability of preemption, and not in preempting less LSPs when preemption is unavoidable. In Fig. 5.9 the average preempted bandwidth is examined. The shapes of the curves

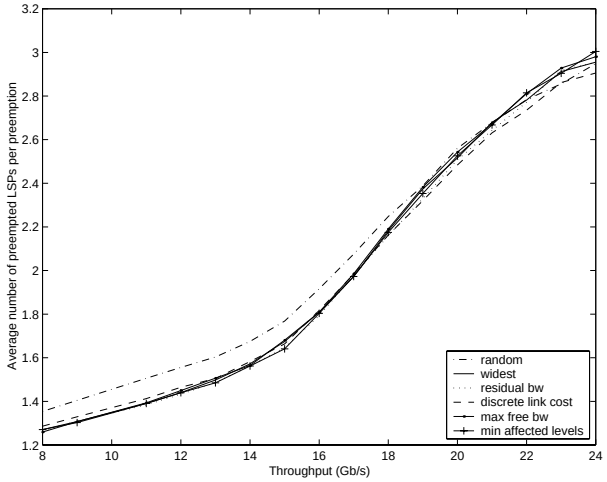


Figure 5.8: Average number of preempted LSPs per preemption

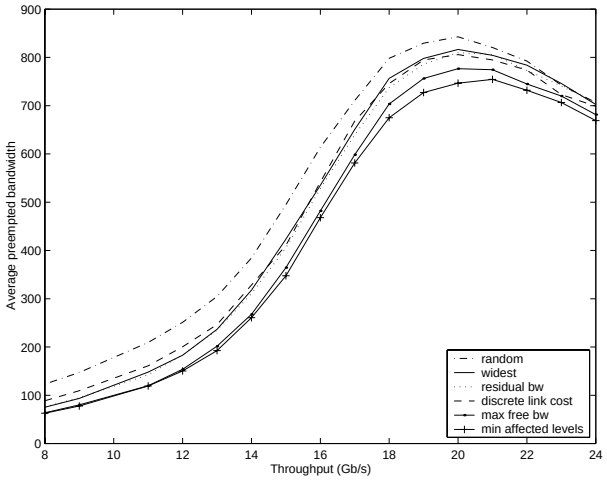


Figure 5.9: Average preempted bandwidth

and the differences between the methods are similar to the results presented in Fig. 5.7.

As discussed in Section 5.4.2 for the bandwidth preemption vector metric, both concave and additive metrics would fit based on different assumptions on how preemption actually happens on consecutive links. We have conducted experiments with both accumulator functions. We have found that a concave metric results in slightly better performance. Therefore, in the previous figures this was used instead of the additive metric.

Accuracy of the measurements

We determined each point in the previous graphs by taking numerous measurements. In Fig. 5.10 and 5.11 we show two already presented graphs by also plotting the 95% confidence intervals. As we can see, at the important low load region, confidence intervals are small enough.

Random network simulations

In order to provide more general results, we carried out simulation experiments on random networks as well. Similarly to our investigations in chapter 4, we varied the number of nodes of the networks (30, 50, 70), and the average node degree (3.5, 4, 4.5). However, we did not intend to change traffic load and in fact the required average link load level was fixed in the simulations. The load level was selected in such a way that for all networks the setup success ratio was high and the preemption probability was in the 10-25% range.

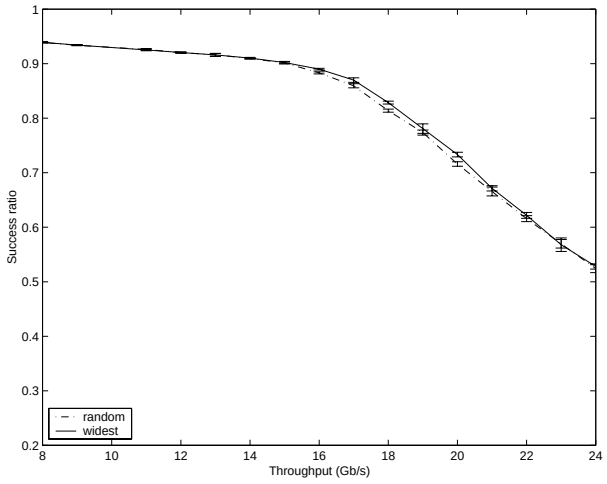


Figure 5.10: Success ratio

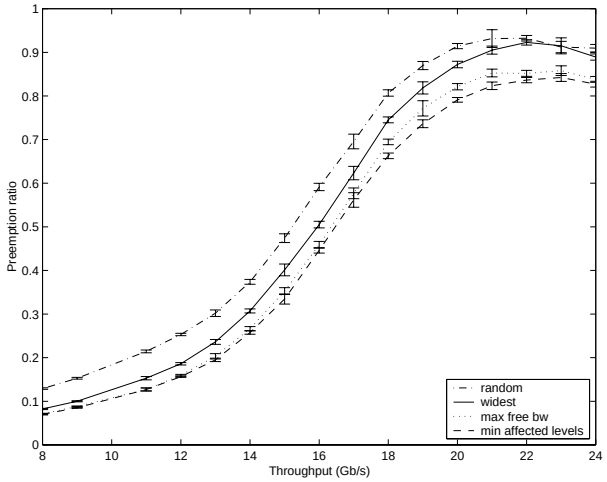


Figure 5.11: Preemption ratio

The graphs depicted in Fig. 5.12 show similar results to our real-world network simulations. The difference between the 'widest' and 'min-aff-level' plots are in the 3-11% range. The exact differences between the methods – determined based on the numerical data– can be seen in Table 5.3. One can see that as the number of nodes and the average node degree increases, so does the improvement that we can achieve by using preemption-aware CSPF techniques.

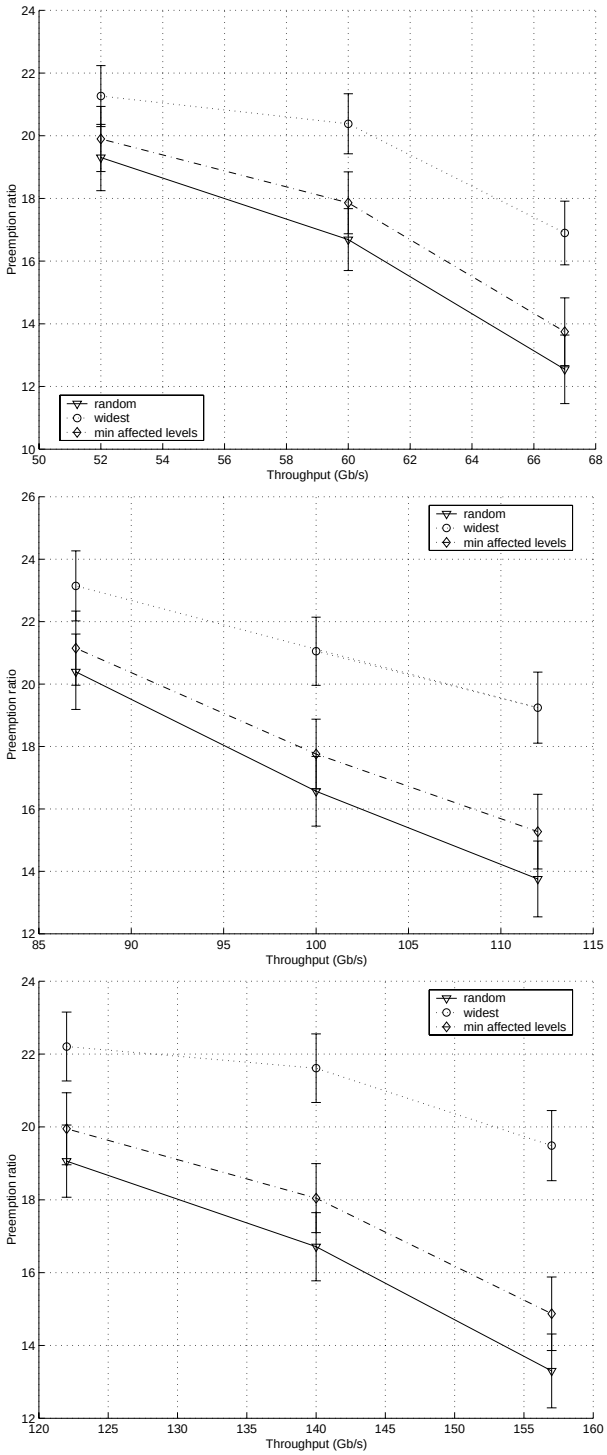


Figure 5.12: Preemption ratio for random graphs of 30, 50 and 70 nodes

Table 5.3: Average improvement with the 'min-aff-levels' method compared to the 'widest' method

Node degree	3.5	4	4.5
30 nodes	3%	7%	9%
50 nodes	4%	7%	10%
70 nodes	6%	7%	11%

5.6 Summary

In networks supporting traffic with diverse QoS requirements, setup of low priority traffic should be done in such a way that subsequently arriving higher priority traffic is not adversely affected. A simple way to solve this problem is to allow preemption between the priority levels. In this chapter we have investigated the effects of bandwidth constrained path calculation on the preemption process. The cornerstone of our method is to perform path selection by taking resource utilization of lower priority traffic into account. In the MPLS environment that was studied we have shown that premium quality can be achieved for high priority LSPs, even if we target preemption minimization.

As a basic step for our preemption minimization algorithms we have specified preemption measures calculated from basic flooded unreserved bandwidth information distributed by IGP traffic engineering extensions. One measure estimates the amount of bandwidth that will be affected, while another one quantifies how many levels will be affected by the new LSP's setup. These measures are then used to construct metrics that are directly applicable in a modified shortest path first algorithm. We build on Dijkstra's well known shortest path first algorithm because of its speed. Its running time is determined by the number of edges in the graph, since it considers each edge only once. Our modifications increase the running time of the original algorithm by a constant factor.

Our simulation experiments demonstrate that the proposed priority-aware path selection algorithms significantly outperform traditional load-balancing CSPF methods in terms of the number of preempted lower priority LSPs, thus resulting in *less re-routing* in the network. In addition, we have found that this is achieved while retaining *equal path establishment success ratio*. We obtained the best results when we aimed at minimizing the affected priority levels with the help of a second metrics.

We have not carried out experiments to determine the performance of the algorithms in the case of *inaccurate* link state information. In order to carry out the necessary and valuable simulations, a deeper understanding is needed of the LSP setup arrival processes and bandwidth distributions, which significantly influence the accuracy of reservation information.

Chapter 6

A novel architecture for testing real path selection algorithm implementations

6.1 Introduction

In previous chapters we have discussed networking issues that required different routing algorithms. We have seen that for a specific task, numerous methods are proposed in the literature. Whenever a new method is developed, its benefits are shown with the help of analytical methods, or by simulation. Given numerous algorithms and an analytical or simulation comparison, system designers of an equipment manufacturer are able to select the most viable method to implement in a specific device (e.g., an MPLS enabled router or an ATM switch). At one point in the development cycle the device must be properly tested in order to guarantee conformance to protocol specifications and correct functioning of the various parts. The former is known as conformance testing and there are well known procedures for dealing with it (test case generation, usage of tools based on the TTCN language [78]). Unfortunately, for the latter task i.e., to test implementation dependent parts, such a standard apparatus does not exist. In particular, routing algorithms implemented in device code cannot be easily tested since there may not be a suitably large test network available, or those built are already used for commercial services. Manufacturers developing MPLS or ATM equipment need empirical information about the behavior and performance of their product in networks of a realistic size. The main problem is the lack of test networks of a suitable-size. A similar problem is the testing of equipment in specific network configurations. In a test network the reproduction of a complex situation that occurs rarely in real life is a complex and sometimes impossible task.

In this chapter we address the above problem and presents a novel architecture

to examine the behavior of a switch operating in a large ATM network running the Private Network-Network Interface (PNNI) protocol [17]. Our aim is to achieve such a configuration where the implementation under test is loaded with signalling messages generated by a large simulated network.

The rest of the chapter is organized as follows: in the first section we introduce some commonly used test methods for routing protocols. In Section 6.3 we present our tool architecture for evaluating the performance of real implementations of the PNNI protocol. In Section 6.4 we survey practical applications of the tool. Finally, in Section 6.5 we summarize the chapter.

6.2 Background

There are a number of approaches for implementing test environments:

- Building test networks
- Running multiple device code instance on workstations, and interfacing them
- Connecting a simulated network to a real switch

To provide an efficient test environment for the path computation component of an ATM switch, a number of requirements should be met. Let us list the most important ones:

- Configuration of the test network should be handled easily, from a single control point
- The environment should allow the creation of large test networks (e.g., up to 100 nodes)
- The Implementation Under Test should not notice that it is functioning in a test environment and not in a real network

Obviously a test network built from real devices can not meet the first requirement, since its configuration is quite cumbersome. The next common practice of equipment manufacturers is to interface more software instances of the actual device code on single or multiple workstations, and carry out tests in such an emulated network. Here, minor development work is needed in order to extend the actual interface functions of device code and allow interprocess communication through e.g., TCP sockets. The drawback of this method is that the configuration of the network from a single point is still not supported.

Gremmelmaier, Winter and Jocher [79] took the above approach, i.e, they ran multiple instances of a commercially available PNNI routing software component

from Trillium [80], but they extended the system with central control code that handled the configuration and measurement tasks of each emulated node. The drawback of such an emulator concept is the fact that each emulated node includes a complete PNNI implementation including PNNI message processing, building of the topology database etc. This limits the number of emulated nodes that a workstation can handle. In [79] authors report that a Sun Ultra Sparc workstation is capable of running around 15 emulated node processes. Of course this does not limit the size of the test network, since [79] supports the interconnection of more workstations as well.

In case a simulator is used to build the test environment, the abstraction level of the simulator strongly determines the capabilities of the test system. Normally a simulator models a node in full detail, by implementing all the functions of a real device. For example PRouST (the PNNI Routing and Simulation Toolkit) models switches in such a way that they are analogous to hardware ATM switches [81]. ProuST switches have a switch fabric, physical ports and a control part. The architecture includes protocol stacks, Q93b call and message handling, multiplexing and a topology database implemented in each simulated node. These details are required when one wants to closely investigate the operation of an ATM switch in the simulator. On the other hand, for our purposes simulating these details would be of no use at all. Therefore in our tool architecture a number of abstractions were introduced. The details of these are discussed in the next section.

6.3 Tool architecture

In this section, we present a simulator architecture and its extension that enables the interconnection with a real ATM switch running the PNNI routing protocol [C3]. Our primary aim is to provide an efficient test system for evaluating the routing control and path computation component of a PNNI capable ATM switch. The first subsection concentrates on the abstractions used in the simulator, then such functions are detailed that are necessary for interconnecting the simulator with a real device.

6.3.1 Simulator component

PlasmaSIM, developed in Ericsson Traffic Laboratory [82], is an event-driven simulator program capable of running on Sun workstations. The PNNI version of PlasmaSIM allows the investigation of the behaviour of the PNNI protocol. This version was developed by a small team, namely Gábor Privitzky, Áron Szentesi and the author of this dissertation. Szentesi in [83] described the abstract processor model of the PNNI component of the simulator, which enables the reflection of the characteristics of any switch in a simulated node. This work is important since

with the help of this, a simulated network provides measurement results close to those obtainable from a network built from real devices.

By using the basic simulator functions of PLASMA, as well as the ATM and PNNI features described in [83], the author of this dissertation concentrated on such parts of the PNNI simulator architecture that enable modelling of the routing protocol in large networks. As a significant work, a simplification regarding the topology database handling in the simulator has been implemented. Among the functional requirements for emulating large networks, it was specified that the tool should:

- Implement the 'Hello' core finite state machine (Down, Attempt, 1-Way and 2-Way Inside states),
- Fully implement the Database Synchronization procedures of the Neighboring-FSM (Database Summary, PTSE packets handling),
- Use all the specified messages queues related to the Flooding procedures, ie., Retransmission lists, PTSE Request lists, Delayed Acknowledge lists,
- Fully implement PTSEs aging, flushing and re-origination,
- Model triggered updates of PTSEs with significant change events controllable with avCR_PM and avCR_MT
- Support DTL stack handling, crankback and alternate routing and basic path selection algorithms (Dijkstra algorithm, Equal Cost MultiPath (ECMP))

The above items describe what is needed in the tool's functional model, but there are important simplifications as well. These are described in the next subsection.

Efficient Topology Database implementation

It was previously discussed that emulator tools built from device code and simulators that model ATM switches generally use their own topology database implemented in each emulated/simulated node. On the contrary, in our model only a single instance of each PNNI Topology State Element is stored in the simulator. These are kept in a Global Topology Database. Each simulated PNNI node holds only a PTSE reference: a pointer to the PTSE in the global database with two time related attributes. The latter are required as nodes must keep track of the remaining lifetime of the PTSEs individually. Moreover, to be able to update the RemLifeTime attribute of PTSEs distributed to the neighbours, the arrival times are also stored (we subtract the time spent in this node's database from the RemLifetime, i.e., the difference of the arrival time and the current simulation time).

Keeping track of the remaining lifetime is also needed for implementing PTSE aging procedures. When a node receives a PTSE its schedules the flushing of this PTSE from its database by using the RemLifeTime attribute. The pointer and these two time related attributes are kept in each node separately with the help of a 'Database Filter' (See Fig. 6.1).

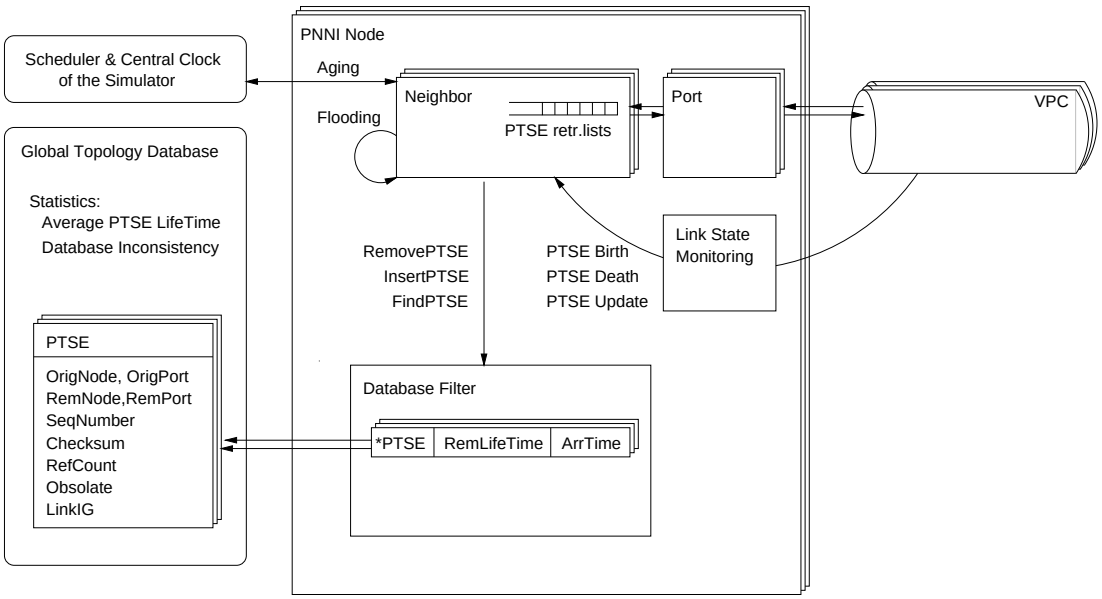


Figure 6.1: Functional diagram of the Topology Database Implementation

With the help of Fig 6.1 let we list the basic operation of the tool. Whenever a link becomes operational between two switches and the 'Hello' finite state machine reaches the two-way inside state, a PTSE is originated by the 'Link-state monitoring' block of the simulator. This is what we call a 'PTSE Birth'. At this time the 'Neighbour FSM' should be triggered to insert the new PTSE to this node's filtered database and also to the global database. Also when the load on a link crosses a significant change threshold because of connection establishments, a PTSE update is triggered. Whenever a node floods a PTSE, the reference count attribute of that PTSE is duplicated in the global database. Actually the constructor, destructor and copy operator of the PTSE reference update the refCount attribute in the global database.

A PTSE is deleted from the global database when this reference count becomes zero. When the originating switch of a PTSE flushes the old PTSE (originates a new instance and sends the old one with remaining lifetime of zero to its neighbours), we mark the old PTSE in the global database as obsolete. Of course, due to the delay in flooding, this PTSE may still be valid in other node's filtered database with non-zero RemLifeTime. By summing up the refCount attribute

value of obsolete PTSEs, we can derive a statistic that reflects the inconsistency of the simulated nodes' topology databases. In the global database we also keep track of the origination time of each PTSE, so that when deleting a PTSE, we can update another global statistic attribute that measures the average PTSE lifetime in the database.

With the above implementation of the PTSE storage the message length of flooded PTSEs are also shortened, thus simplifying the flooding and database synchronization procedures.

6.3.2 Emulator component

Due to the abstractions used in the simulator architecture mentioned in the previous section, it is possible to simulate realistically sized networks at a much faster rate than real time in event-driven mode. However, when we want to use the simulator for test purposes and connect it to a real ATM device its central scheduler should operate in real-time mode. This is achieved by a modified version of the scheduler that was developed specifically for emulation purposes.

The second essential component of the emulator is the so called 'interface node' or 'interface function'. This is actually a PNNI node that does not contain the protocol components such as the 'Hello' and 'Neighbour' final state machines. Instead it converts simulated messages to bit conform PNNI signalling messages, and the device under test on the other side of the interface responds to them. In an interface node, similar to the situation in a simulated PNNI node, access is needed to the global topology database to build a whole PTSE from the PTSE reference. We can easily visualize this with the help of the following two figures. In Fig. 6.2 the functional diagram of a two-node simulated network is shown, while in Fig. 6.3 the same network is depicted, the difference being the use of the emulator concept.

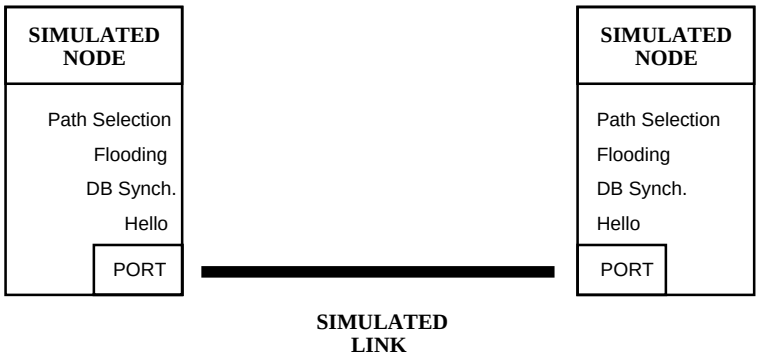


Figure 6.2: Functional diagram of a two-node simulated network

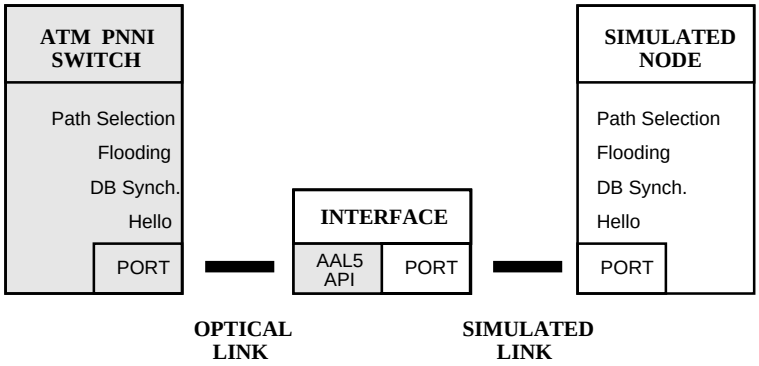


Figure 6.3: Functional diagram of a two node emulated network

While in Fig. 6.2 the nodes are connected to each other by a *simulated* ATM link, on the second figure the logical link is split into two. The first part up to the interface node is the same as a link in the simulator, while the second part is an ordinary ATM link with and ATM VPC being a PNNI routing control channel (RCC). Two allow this, of course the workstation where the emulator runs should be equipped with an ATM card for sending and receiving ATM messages on the RCC, and the interface function should implement ATM AAL5 message construction with the help of the ATM card's application programming interface. The interface function and the API handling part was implemented in such a way that it is possible to connect the tool with more logical ATM links to an IUT. Thus the IUT can be tested as if it was connected to the middle of a test network as well. This was another important part of the tool's development work.

To summarize, the next list includes the most important functions of the tool architecture:

1. Scheduler operating in real-time mode instead of the simple event-driven mode

2. An ATM card, that enables the communication with an ATM switch through the routing control channel

3. An interface function that receives the simulated PNNI messages, translates them into bit-conform ones and sends them to the real switch.

6.4 Practical usage of the tool

Manufacturers can face situations in which such a tool can be helpful in different phases of their products' life-cycle. In the development phase a single switch can

be tested as if it was part of a bigger network. In later phases rarely occurring problematic scenarios can be set up quite easily.

In early stages of product development, when installed networks using the given device do not exist, engineers and manufacturers cannot gain operational experience. Using the tool it is possible to test the equipment being developed before installation and probable protocol errors can be detected. In this development phase corrections are much cheaper than in later stages.

At later stages the method has still its benefits. Let us suppose an operator has a large running PNNI-based ATM network. Since this real network is used for commercial services it can not be used to conduct special tests, e.g., to study the effect of network failures caused by a transmission link cut. This is usually analyzed in a simulated environment where the failure can be re-enacted several times with different simulated network configurations until the best possible restoration configuration is found. However, it is also desirable to analyze thoroughly the real PNNI equipment's behaviour in such crucial situations. This can be done with the help of the tool efficiently in an emulated environment, where we load a real implementation with the huge amount of protocol messages generated by the link cut event.

We show the usage of the tool in more detail via presenting two emulated networks. In the left-hand side screenshot of Fig. 6.4, the real switch is connected to one simulated node. In the right-hand side screenshot of Fig. 6.4 a second example is shown where the switch is in the center of a network.

In the first case there is a simulated network, generating a large number of PNNI protocol messages that are sent to the real switch. The switch builds a topology database that is coherent with the network given in the configuration file of the simulator, so we can verify whether the real equipment is able to build the correct topology database of a large network. The real switch should also build its routing table according to the received information.

In the second case the switch is connected to two simulated nodes. In this case the network configuration bears the same advantages as in the first case. However, this is also suitable for observing the message flow through the real equipment.

With the two emulated networks above we conducted tests in order to study the effect of frequent flooding on the processor load of a real PNNI switch. We were interested in the performance of the switch under heavy signalling load. When using the default setting of the PNNI architectural variables the load on the processor was very low, therefore we changed some protocol settings in order to have higher load figures. In PNNI a variable called *PTSERefreshInterval* determines how often each node originates new link-descriptors (PTSEs). In Fig. 6.5 we plotted the processor load versus this refresh interval. At the highest flooding frequency each of the six simulated nodes re-originated all of its PTSEs in every four seconds. This means that the real switch was flooded by more than five PTSEs in a second. As

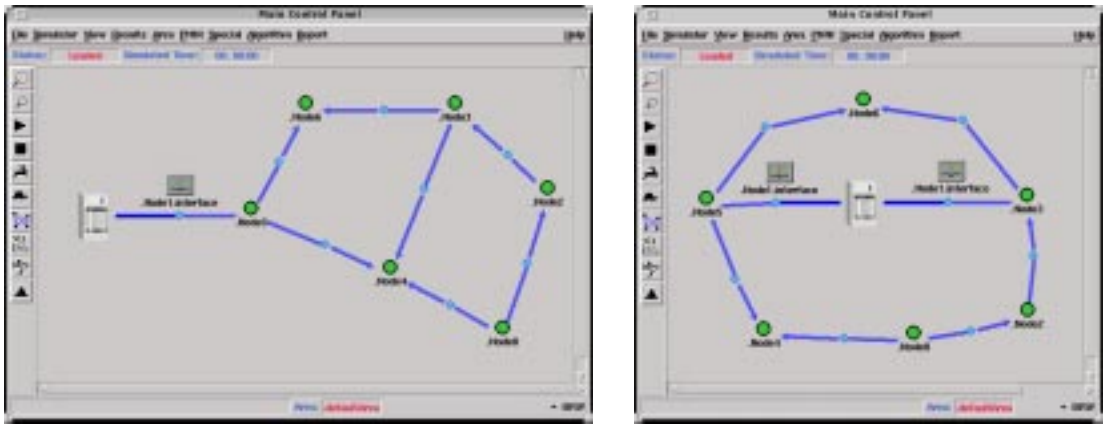
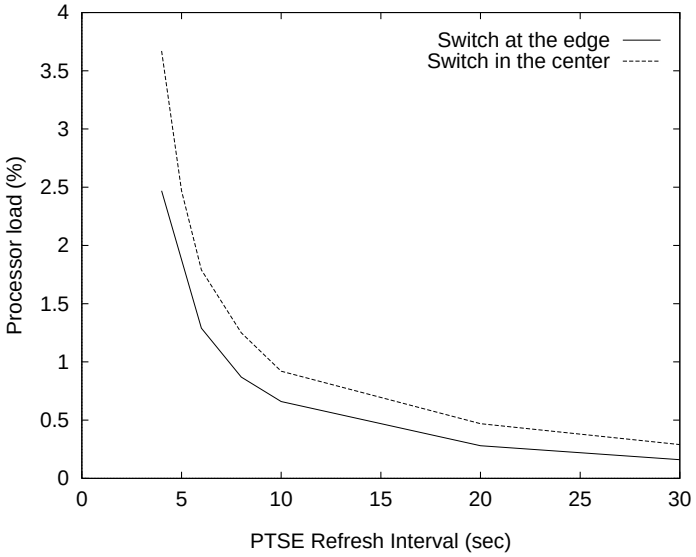


Figure 6.4: Seven node networks with one real node at the edge and in the center of the network

we can see in the figure in this situation the implementation under test worked still well with low processor load. The steepness of the curves follows approximately the function $1/x$ which comes from the fact that processor load is proportional to flooding frequency, which is approximately inversely proportional to the time interval between consecutive flooding events.

In this simple test we have changed the PNNI protocol settings in order to have high signalling load, however, in larger networks similar flooding frequency could occur with the default protocol settings. These kind of situations could be also studied with the emulator tool.



Settings

Real switch and Emulator:
MinPTSEInterval: 0.1s,
PTSELifetimeFactor: 101%,
PeerDelayedAckInterval: 1s.

Real switch:
PTSERefreshInterval: 30s,

Emulator:
PTSERefreshInterval:
decreased from 30s down to 4s.

Figure 6.5: The processor load of a real switch connected to the emulator

6.5 Summary

In this chapter we presented a novel tool architecture for testing PNNI capable ATM equipment. The most important components of the tool were discussed, namely the efficient topology database implementation on the simulator side, and interface functions on the emulator side. The system — consisting of a workstation equipped with an ATM card and the described software — is capable of emulating a large PNNI network, that can be easily set up with the help of a single configuration file.

We have also compared our architecture to other emulator scenarios described in the scientific literature [79], [81]. Although recently commercial PNNI emulator tools appeared among company offerings [84], [85], the implementation details of these tools are not available publicly. Therefore the comparison with these commercial tools could not be carried out in this work. The lack of scientific bibliography of these commercial tools means that it is difficult to determine a clear development or publication date for them. Therefore it is not possible to relate time-wise the tool discussed in this chapter [C3] and the commercial tools [84] and [85].

Chapter 7

Conclusions

As web-based applications in IP networks becomes a central part of the customer support services of huge companies such as banks, airlines and different trading houses, the requirements on the performance of IP based networks changes significantly. Internet Service Providers should guarantee low latency and packet loss as well as short restoration times when network elements fail. Distributed control and standard shortest-path routing of traffic flows is no longer sufficint to achieve these goals. There is a definite need for centralized monitoring facilities in 24-hour operations centers, from where immediate actions can be initiated whenever needed. By studying the dynamics of traffic flows in backbone networks, many service providers intend to automate certain network operation tasks e.g., the optimization of network resource utilizations with the help of traffic engineering methods.

In this dissertation we evaluated the effect of different routing optimization and traffic engineering methods on the performance of aggregate traffic flow routing. We investigated such cases when per-flow routing is available, i.e., paths of aggregate flows can be determined individually, and resources can be reserved on such explicit paths. Industry standard technologies that enable this are the Multiprotocol Label Switching technology and the virtual circuit switching concept of Asynchronous Transfer Mode networks. In the case of ATM the relevant routing and signalling protocol investigated in this work is based on the Private Network-Network Interface specification. In Chapter 2 of this dissertation we provided an overview of the most important components of backbone networks that are related to routing and traffic engineering. In subsequent chapters we proposed novel algorithms for improving the efficiency of backbone networks. The remainder of this chapter discusses the contribution of the dissertation in more detail, and also presents some related problems that are the subject of future research.

7.1 Research contribution

Chapter 3 proposed and evaluated a new approach to establish a label switched path in an MPLS network. As far as can be determined, recent research on path selection algorithms was limited to either CSPF [29, 31, 44, 72, 75, 76] or global path optimization algorithms [43, 44, 45, 86, 87]. When the Constrained Shortest Path First algorithm (CSPF) implemented in Label Edge Routers cannot find an appropriate path for an LSP, either the LSP setup is blocked or a global optimization is triggered. However, a complete optimization of all LSPs—depending on the number of paths—can take hours on a dedicated server. The method proposed in Chapter 3 suggests a third option, namely to trigger a *prompt partial path optimization* in order to route the new LSP. This requires a fast algorithm that affects the established paths of only a few LSPs' in the network. The algorithm and simulation study discussed in Chapter 3 considers the rerouting of only one LSP in order to facilitate the establishment of the new LSP. We described heuristic methods to identify candidate re-routable LSPs and conducted simulation experiments to measure the efficiency and applicability of the algorithms. We witnessed that even with this simple optimization scenario a significant increase in LSP setup success rate could be achieved. The results reported in this chapter are also presented in [C5]. In a later work [J5], we investigated the possibility of rerouting 1, 2 or 3 LSPs.

A number of applications have stringent availability requirements. To meet these, operators must configure not only working paths, but backup paths as well. There are distributed CSPF based algorithms proposed in the literature to compute both the working and the backup path of an LSP. As a main contribution of Chapter 4, we provide a numerical comparison of two most widespread backup path computation algorithms. The main investigation concerns the path lengths of primary and backup paths. Moreover, we introduce a new method for computing two backup paths for a working path. By requiring that one of the paths should be shortest, we aimed to minimize the resources used in the network at restoration. The numerical results showed that the average secondary path length is smaller for our new method than for existing methods. The proposed method can be used to advantage when the signalling protocol provides information about the location of the failure (crankback), and when backup paths can be configured to be non-revertive. The results reported in this chapter are also presented in [C8].

In MPLS, preemption procedures provide automated mechanisms for restructuring network resources. Upon arrival of a high priority e.g., VoIP traffic trunk, already established low priority LSPs carrying e.g., best-effort traffic can be rerouted. There are built-in procedures in routers both to compute the appropriate explicit paths for LSPs and to select which lower priority ones should be preempted (if there are not enough resources otherwise). Recognizing that network stability is important for operators, Chapter 5 contributes with a novel path computation

algorithm that minimizes preemption of lower priority connections. The path preemption studies reported in this chapter are also presented in [J4], [C7].

While Chapter 3, 4 and 5 discussed new routing algorithms, Chapter 6 provided a new method for testing these algorithms when they are implemented in actual devices. The proposed method eliminates the need to build large test networks. The basic idea is to provide an interface between a simulated network and the equipment that is to be tested. As discussed in Chapter 6 and also in [C3] we demonstrated the viability of this concept by testing the PNNI path computation component of a commercially available ATM switch.

7.2 Future research directions

When we carried out the simulation experiments of this dissertation, MPLS deployment had been just started and therefore real network topologies, LSP configuration information and traffic statistics were not available. In emerging multiservice networks there is a chance that on-demand setup of communication paths will become a reality. It will also be possible to change the bandwidth requirement of connections between different locations based on the time-of-day and usage patterns. In this environment distributed control for fast setup times and on-the-fly optimization for efficient resource utilization should work together. As deployment of such dynamic connection setup, or bandwidth modification procedures becomes a reality, the work of this dissertation could be carried on, by investigating the performance of routing algorithms using more realistic traffic patterns.

A dynamic architecture for accessing λ paths is another area where the role of routing algorithms will be important in future. Here the basic concept is to allow the management and control of wavelengths from the edge of the network. A recent work [88] proposes protocol enhancements to BGP to support dynamic optical light-path provisioning between ASs. In both environments the path optimization algorithms proposed in this dissertation could be carried on as part of a future work. The investigation of algorithms for the latter environment is a challenging task, and it opens the door for inter-operator traffic engineering.

Appendix A

Simulated network configurations

The simulation results discussed in this dissertation were obtained by using real-world network topologies and by generating random-networks. When we were looking for IP backbone network provider topology maps we used the site of Russ Haynal [77]. From this site, we selected such topologies that were neither too small nor too sparse, and for which link capacity information was also available. Based on these requirements, we selected two networks, namely the IP backbone network of AT&T and Cable & Wireless. These are discussed in Section A.1 and A.2 respectively. The details involved in generating random test networks are discussed in Section A.3.

A.1 AT&T network configuration

The 25 node, 88 link network configuration that we have used for our simulations in chapter 3 and 4 is depicted in Fig. A.1. Generally the links have OC-48, i.e., 2.5Gbps capacity. The exception is the link connecting node 0 and 1 which consists of an OC-192 (10Gbps) and an OC-48 link. Moreover, there are two node pairs (5-10, and 10-12), that are connected by two OC-48 links. The precise link capacities are summarized in Table. A.1.

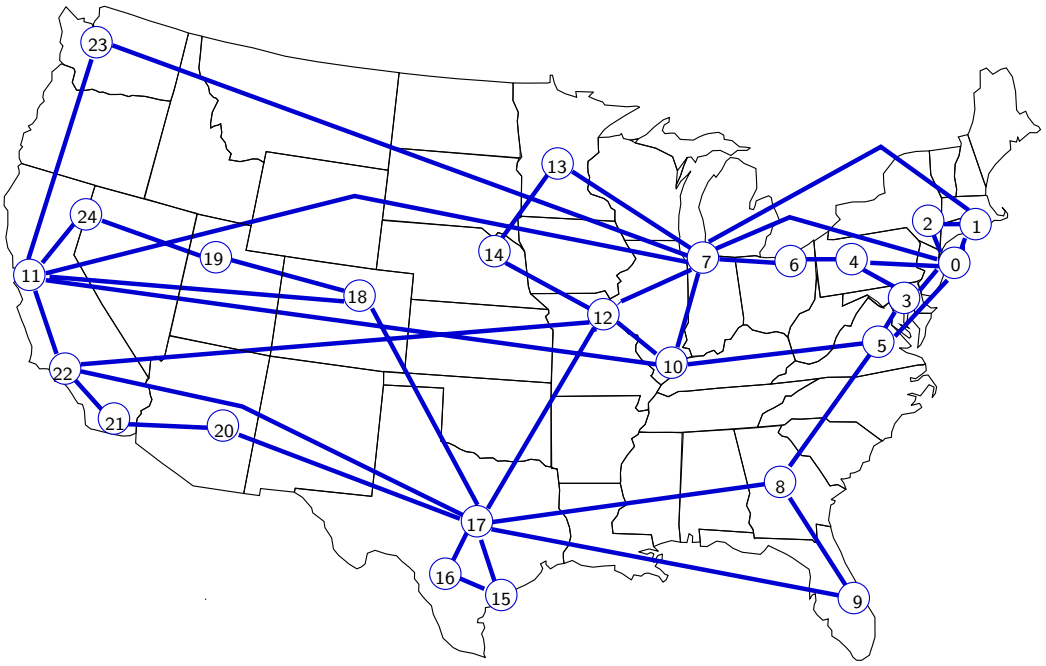


Figure A.1: AT&T USA Backbone network topology

Table A.1: Capacity values for the AT&T network

Capacity	List of links
OC-48	0↔2, 1↔2, 0↔5, 0↔3, 3↔5, 1↔7, 0↔7, 0↔4, 4↔6, 6↔7, 5↔8, 10↔7, 8↔9, 8↔17, 9↔17, 17↔16, 17↔15, 16↔15, 12↔7, 12↔14, 14↔13, 13↔7, 17↔12, 17↔18, 18↔19, 19↔24, 24↔11, 11↔18, 10↔11, 12↔22, 11↔22, 22↔21, 21↔20, 20↔17, 17↔22, 7↔23, 23↔11, 7↔11,
2*OC-48	5↔10, 10↔12
OC-192 + OC-48	0↔1

A.2 Cable & Wireless network configuration

In our simulations we have used two slightly different variants of the C&W network configuration; chapter uses 4 a 30 node network, while chapter 5 uses a 31 node one. The topology of the 30 node network is depicted in Fig. A.2 and the used link capacities are summarized in Table. A.2. The 31 node version is only slightly different, the difference being that on the west-coast the 5-7 and 1-11 links are deleted and on the east-coast a new stub node is added which is only connected

to node 17 with two OC-12 links. Otherwise the same capacity values are valid as before.

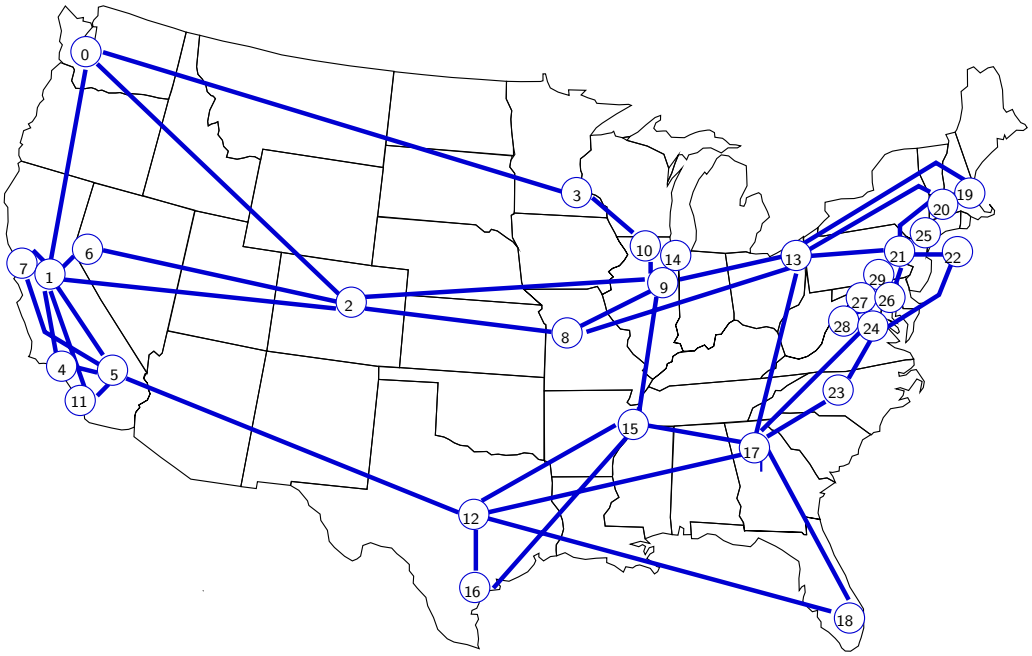


Figure A.2: Cable & Wireless USA Backbone network topology

Table A.2: Capacity values for the C&W network

Capacity	List of links
2*DS-3	1↔4, 4↔5, 12↔16, 22↔21, 15↔16, 24↔22, 12↔18, 17↔18
OC-12	0↔1, 0↔2, 0↔3, 1↔6, 2↔6, 2↔8, 3↔10, 8↔9, 8↔13, 9↔10, 9↔15, 13↔17, 13↔19, 13↔20, 17↔24, 19↔21, 20↔21, 21↔26, 26↔28
2*OC-12	12↔17, 9↔14, 10↔14, 12↔15, 1↔2, 2↔9, 5↔11, 1↔11, 15↔17, 17↔23, 21↔24, 23↔24, 24↔26
3*OC-12	9↔13, 5↔12, 1↔5, 13↔21
OC-48	25↔29, 27↔29, 27↔28, 1↔7, 5↔7, 21↔25
2*OC-48	26↔27
OC-192	25↔27

A.3 Example random networks

To generate random graphs we used a tool developed at Ericsson Research, Traffic Analysis and Network Performance Laboratory by Józsa and Orincsay [65]. Fig. A.3 depicts some example network topologies. Both in chapter 4 and in chapter 5 a new random network was generated for each round of simulation. In one round, different algorithms were tested. Finally the measurement results were averaged and the confidence levels were determined. For random networks due to the differing network topologies the 95% confidence intervals are wider compared to our simulations with fixed real-world network topologies.

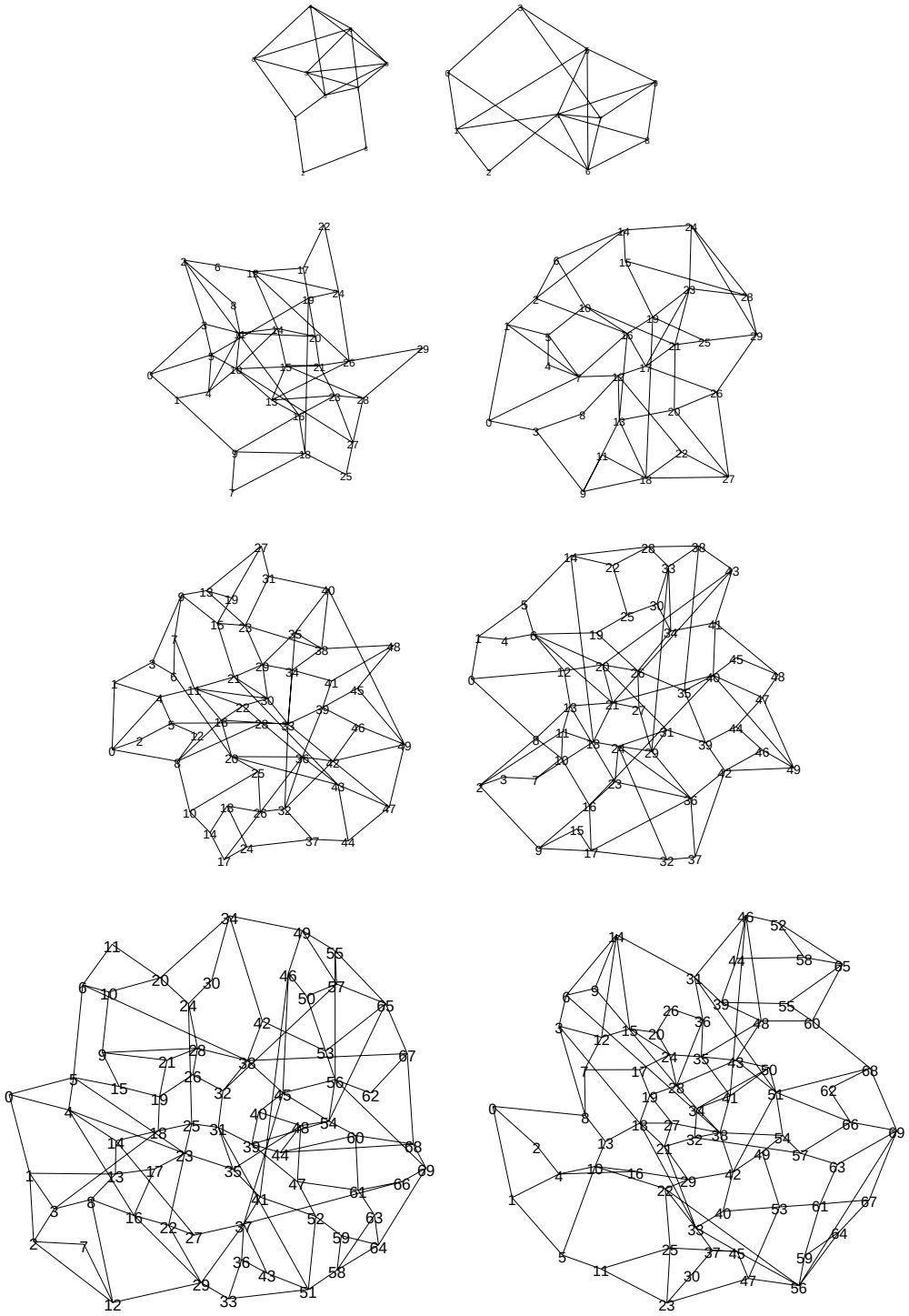


Figure A.3: Random networks with 10, 30, 50 and 70 nodes (node degree of 4)

Bibliography

[1] D. O. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and principles of internet traffic engineering. Internet Draft, Internet Engineering Task Force, October 2001. Work in progress.

[2] T. Cinkler, P. Laborczi, and Á. Horváth. Protection through thrifty configuration. In *16th International Teletraffic Congress*, volume 3b, pages 975–987, June 1999.

[3] P. Laborczi, J. Tapolcai, P.-H. Ho, T. Cinkler, A. Recski, and H.T. Mouftah. Algorithms for asymmetrically weighted pair of disjoint paths in survivable networks. In *Third International Workshop on Design of Reliable Communications Networks, (DRCN)*, October 2001.

[4] A. Iwata, R. Izmailov, and B. Sengupta. Alternative routing methods for PNNI networks with partially disjoint paths. In *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, November 1998.

[5] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the internet hierarchy from multiple vantage points. Technical Report UCB/CSD-1-1151, University of California, Berkeley, August 2001.

[6] J. W. Stewart III. *BGP4: Inter-Domain routing in the Internet*. Addison–Wesley, The Addison–Wesley Networking Basics Series, 1999.

[7] S. Uhlig and O. Bonaventure. IST project ATRIUM - Report 14.2 Analysis of interdomain traffic. Technical Report 2001-12, University of Namur, September 2001.

[8] Home Page. Cable News Network, Inc. http://www.cnn.com/.

[9] Home Page. British Broadcasting Corporation. http://news.bbc.co.uk.

[10] Home Page. Akamai Technologies, Inc. http://www.akamai.com.

[11] Home Page. Digital Island, Inc.
http://www.digitalisland.com.

[12] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. POP-level and access-link-level traffic dynamics in a tier-1 POP. In *SIGCOMM Internet Measurement Workshop*. ACM, November 2001.

[13] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: Methodology and experience. In *SIGCOMM Symposium on Communications Architectures and Protocols*, Stockholm, Sweden, August/September 2000.

[14] Cisco Systems – product information page. Cisco NetFlow.
http://www.cisco.com/warp/public/732/Tech/netflow/.

[15] G. Malkin. RIP version 2. Request for Comments (Proposed Standard) 2453, Internet Engineering Task Force, November 1998.

[16] J. T. Moy. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley, Reading, MA, USA, 1998.

[17] Private network-network interface specification, version 1.0 (PNNI 1.0). Technical Report AF-PNNI-0055.000, ATM Forum PNNI Sub-Working Group, March 1996.

[18] A. Zinin and M. Shand. Flooding optimizations in link-state routing protocols. Internet Draft, Internet Engineering Task Force, August 2001. Work in progress.

[19] K. Tesink. Definitions of managed objects for the SONET/SDH interface type. Request for Comments (Proposed Standard) 2558, Internet Engineering Task Force, March 1999.

[20] C. Alaettinoglu, V. Jacobson, and H. Yu. Towards milli-second IGP convergence. Internet Draft, Internet Engineering Task Force, November 2000. Work in progress.

[21] A. Basu and J.G. Riecke. Stability issues in OSPF routing. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 225–236. ACM, 2001.

[22] G. L. Choudhury, A. S. Maunder, and V. D. Sapozhnikova. Faster link-state IGP convergence and improved network scalability and stability. In *IEEE Conference on Local Computer Networks*, November 2000.

[23] J. Ash, G.L. Choudhury, J. Han, V.D. Sapozhnikova, M. Sherif, M. Noorchashm, A. Maunder, and V. Manral. Proposed mechanisms for congestion control / failure recovery in OSPF & ISIS networks. Internet Draft, Internet Engineering Task Force, October 2001. Work in progress.

[24] J. Moy. Hitless OSPF restart. Internet Draft, Internet Engineering Task Force, August 2001. Work in progress.

[25] A. Shaikh and A. Greenberg. Experience in black-box OSPF measurement. In *SIGCOMM Internet Measurement Workshop*. ACM, November 2001.

[26] P. Narváez, K.-Y. Siu, and H.-Y. Tzeng. New dynamic algorithms for shortest path tree computation. In *IEEE/ACM Transactions on Networking*, volume 8, pages 734–746, December 2000.

[27] G. Ash. Traffic engineering & QoS methods for IP-, ATM-, & TDM-based multiservice networks. Internet Draft, Internet Engineering Task Force, October 2001. Work in progress.

[28] G. Apostolopoulos, R. Guerin, and S. Kamat. Implementation and performance measurements of QoS routing extensions to OSPF. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, volume 1, March 1999.

[29] G. Apostolopoulos, R. Williams, S. Kamat, R. Guerin, A. Orda, and A. Przygienda. QoS routing mechanisms and OSPF extensions. Request for Comments (Proposed Standard) 2676, Internet Engineering Task Force, August 1999.

[30] G. Apostolopoulos, R. Guerin, S. Kamat, and S. K. Tripathi. Improving QoS routing performance under inaccurate link state information. In *16th International Teletraffic Congress*, June 1999.

[31] A. Shaikh, J. Rexford, and K. G. Shin. Evaluating the impact of stale link state on quality-of-service routing. *IEEE/ACM Transactions on Networking*, 9(2):162–176, April 2001.

[32] J. Agogbua, D. Awduche, J. Malcolm, J. McManus, and M. O'Dell. Requirements for traffic engineering over MPLS. Request for Comments (Proposed Standard) 2702, Internet Engineering Task Force, September 1999.

[33] D. H. Lorenz, A. Orda, D. Raz, and Y. Shavitt. How good can IP routing be? Technical Report 2001-17, DIMACS: partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore), May 2001.

[34] Cisco Systems – Application Note. Load balancing with Cisco Express Forwarding. http://www.cisco.com/warp/public/cc/pd/ifaa/pa/much/prodlit/loadb_an.pdf.

[35] C. Villamizar. OSPF optimized multipath (OSPF-OMP). Internet Draft, Internet Engineering Task Force, October 1998. Work in progress.

[36] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, March 2000.

[37] Jennifer Rexford. Traffic engineering for internet service provider networks. Slide presentation given at the conference on stochastic networks (invited talk), AT&T Labs – Research, June 2000. Available at http://www.research.att.com/~jrex/papers/.

[38] Y. Wang, Z. Wang, and L. Zhang. Internet traffic engineering without full mesh overlaying. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, April 2001.

[39] IRTF Routing Working Group Mailing List Archive. Thread – Re: interesting Infocom paper on traffic engineering via routing metrics. http://puck.nether.net/lists/irtf-rr/.

[40] C. Liljenstolpe. Offline traffic engineering, a best current practice from a large ISP. Internet Draft, Internet Engineering Task Force, October 2001. Work in progress.

[41] B. G. Józsa, Z. Király, G. Magyar, and Á. Szentesi. An efficient algorithm for global path optimization in MPLS networks. *Optimization and Engineering*, 2(3):321–347, September 2002.

[42] B. G. Józsa and G. Magyar. Reroute sequence planning for label switched paths in multiprotocol label switching networks. In *The 6th IEEE Symposium on Computers and Communications (ISCC'2001)*, pages 319–325, July 2001.

[43] S. Plotkin. Competitive routing of virtual circuits in ATM networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1128–1136, August 1995.

[44] M. Kodialam and T. V. Lakshman. Minimum interference routing with applications to MPLS traffic engineering. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 884–893, March 2000.

[45] P. Aukia, M. Kodialam, P.V. Koppol, T.V. Lakshman, H. Sarin, and B. Suter. RATES: A server for MPLS traffic engineering. *IEEE Network*, 14(2):34–41, 2000.

[46] D. Wang. Network planning and analysis tools for MPLS networks. Slide presentation given at the IW-MPLS-1998 conference, Wandl, Inc., November 1998. Available at http://www.wandl.com/html/support/papers/npatiwmpls98s.pdf.

[47] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[48] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley&Sons, Wiley-Interscience series in discrete mathematics, 1986.

[49] M. Berkelaar and J. Dirks. lp_solve 2.2. ftp://ftp.es.ele.tue.nl/pub/lp_solve.

[50] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, New Jersey 07458, 1993.

[51] K. Kompella and D. Awduche. Notes on path computation in constraint-based routing. Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[52] S. Baroni, J.O. Eaves, M. Kumar, M.A Qureshi, A. Rodriguez-Moral, and D. Sugerman. Analysis and design of backbone architecture alternatives for IP optical networking. *IEEE Journal on Selected Areas in Communications*, 18(10):1980–1994, October 2000.

[53] A. Banerjee. Fault recovery for guaranteed performance communications connections. *IEEE/ACM Transactions on Networking*, 7(5):653–667, October 1999.

[54] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14:325–336, 1984.

[55] T.V. Lakshman and M. Kodialam. Dynamic routing of bandwidth guaranteed tunnels with restoration. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, March 2000.

[56] V. Sharma, B. Crane, S. Makam, K. Owens, C. Huang, F. Hellstrand, J. Weil, L. Andersson, B. Jamoussi, B. Cain, S. Civanlar, and A. Chiu. Framework for MPLS-based recovery. Internet Draft, Internet Engineering Task Force, July 2001. Work in progress.

[57] D.H. Gan, P. Pan, A. Ayyangar, and K. Kompella. A method for MPLS LSP fast-reroute using RSVP detours. Internet Draft, Internet Engineering Task Force, April 2001. Work in progress.

[58] S. Kini, M. Kodialam, S. Sengupta, and C. Villamizar. Shared backup label switched path restoration. Internet Draft, Internet Engineering Task Force, May 2001. Work in progress.

[59] R. Goguen and G. Swallow. RSVP label allocation for backup tunnels. Internet Draft, Internet Engineering Task Force, November 2000. Work in progress.

[60] R. Doverspike and J. Yates. Challenges for MPLS in optical network restoration. *IEEE Communications Magazine*, 39(2):89–96, February 2001.

[61] B. Jamoussi et al. Constraint-based LSP setup using LDP. Request for Comments (Proposed Standard) 3212, Internet Engineering Task Force, January 2002.

[62] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP tunnels. Request for Comments (Proposed Standard) 3209, Internet Engineering Task Force, December 2001.

[63] A. Iwata, N. Fujita, G. Ash, and A. Farrel. Crankback routing extensions for MPLS signaling. Internet Draft, Internet Engineering Task Force, November 2000. Work in progress.

[64] C. Villamizar. MPLS traffic engineering in a QoS capable network. Slide presentation given at the MPLS-2000 conference, Avici, Inc., Washington DC, October 2000. Available at http://www.ail.gmu.edu/MPLS2000.

[65] B. G. Józsa and D. Orincsay. Random graph generator (for telecommunication networks). Technical report, Ericsson Research, Hungary, Traffic Analysis and Network Performance Laboratory, April 1999.

[66] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. A framework for internet traffic engineering. Internet Draft, Internet Engineering Task Force, April 2001. Work in progress.

[67] J. A. Garay and I. S. Gopal. Call preemption in communication networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, volume 3, pages 1043–1050, March 1992.

[68] D. Mitra and K.G. Ramakrishnan. A case study of multiservice, multipriority traffic engineering design for data networks. In *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, December 1999.

[69] D. Katz, D. Yeung, and K. Kompella. Traffic engineering extensions to OSPF. Internet Draft, Internet Engineering Task Force, March 2001. Work in progress.

[70] T. Li and H. Smit. IS-IS extensions for traffic engineering. Internet Draft, Internet Engineering Task Force, September 2000. Work in progress.

[71] M. Peyravian and A. D. Kshemkalyani. Decentralised network connection preemption algorithms. *Computer Networks and ISDN Systems*, 30(11):1029–1043, June 1998.

[72] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1288–1234, September 1996.

[73] A. Shaikh, J. Rexford, and K. G. Shin. Load-sensitive routing of long-lived IP flows. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 215–226. ACM, 1999.

[74] F. P. Kelly. Bounds on the performance of dynamic routing schemes for highly connected networks. *Mathematics of Operations Research*, 19(1), 1994.

[75] Q. Ma, P. Steenkiste, and H. Zhang. Routing high-bandwidth traffic in max-min fair share networks. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 206–217. ACM, August 1996.

[76] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. In *International Conference on Network Protocols (ICNP)*, pages 191–202, October 1997.

[77] R. Haynal. Russ Haynal's ISP page. http://navigators.com/isp.html.

[78] Methods for testing and specification (MTS); the tree and tabular combined notation version 3; TTCN-3: Core language. Technical Report ETSI DES / MTS-00063-1 v1.0.10, European Telecommunications Standards Institute (ETSI), 2000.

[79] U. Gremmelmaier, M. Winter, and J. Jocher. The PNNI Emulator: A versatile tool for planning and operation support of PNNI networks. In *8th International Telecommunication Network Planning Symposium*, 1998.

[80] Trillium Digital Systems – product information page. Atm software source code – data sheet. http://www.trillium.com/assets/broadband!atm/datasheet/8742023.pdf.

[81] B. Khan, D. Talmage, Mountcastle S, A. Battou, and S. Marsh. Introducing PRouST: The PNNI routing and simulation toolkit. In *2001 IEEE Workshop on High Performance Switching and Routing*, May 2001.

[82] Zs. Haraszti, I. Dahlquist, A. Faragó, and T. Henk. PLASMA - an integrated tool for ATM network operation. In *International Switching Symposium*, 1995.

[83] Áron Szentesi. *Routing and Topology Optimization in ATM Networks*. PhD thesis, Department of Telecommunications and Telematics, Budapest University of Technology and Economics, 2002.

[84] Spirent Communications Adtech Systems – product information page. Telcordia PNNI Network Emulator.
http://www.adtech-inc.com/products/pnni.asp.

[85] NetTest – product information page. PNNI interEmulator – data sheet.
http://www.nettest.com/products/pnni_emulator/pdf/PNNI_IE_NA_20.pdf.

[86] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line machine scheduling with applications to load balancing and virtual circuit routing. In *Proc. 25th Annual ACM Symposium on Theory of Computing*, pages 623–631, May 1993.

[87] R. Guerin, A. Orda, and R. Williams. QoS routing mechanisms and OSPF extensions. In *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, November 1997.

[88] M. Blanchet, F. Parent, and V. B. St-Arnaud. Optical BGP (OBGP): InterAS light-path provisioning. Internet Draft, Internet Engineering Task Force, March 2001. Work in progress.

Publications

Journal papers

[J1] Á. Magi, Á. Szentesi, B. Szviatovszki, A. Faragó: *Dinamikus útvonalválasztás ATM hálózatokban*, Híradástechnika, Vol. 50, No. 11, pages 2–11, November 1999.

[J2] Á. Magi, Á. Szentesi, B. Szviatovszki: *Analysis of link cost functions for PNNI routing*, Computer Networks, Vol. 34, No. 1, pages 181–197, July 2000.

[J3] A. Faragó, Á. Szentesi, B. Szviatovszki: *Inverse Optimization in High Speed Networks*, To appear in the Spec. Issue on Combinatorial and Algorithmic Aspects of Telecom., Discrete Applied Mathematics Journal (submitted: January 1998, reviewed: April 1999.)

[J4] B. Szviatovszki, Á. Szentesi, A. Jüttner: *Minimizing Re-Routing in MPLS Networks with Preemption-Aware Constraint-Based Routing*, Computer Communications Journal, Spec. Issue on Advances in Performance Evaluation of Computer and Telecommunications Networking, Vol. 25, No. 11–12, pages 1076–1084, May 2002.

[J5] D. Orincsay, B. Szviatovszki, G. Böhm: *Prompt Partial Path Optimization in MPLS networks*, Submitted to The International Journal of Computer and Telecommunications Networking, (submitted: November 2002)

[J6] J. Forslow, I. Jarrett, P. Moran and B. Szviatovszki: *Management solutions for IP networks*, Ericsson Review. vol.77, no.1, pages 42–52. 2000.

Conference and workshop papers

[C1] A. Faragó, Á. Szentesi, B. Szviatovszki: *Allocation of Administrative Weights in PNNI*, In proc. of the 8th International Telecommunication Network Planning Symposium, Networks'98, pages 621–626, Sorrento, Italy, October 18–23, 1998.

[C2] Á. Magi, G. Nagy, Á. Szentesi, B. Szviatovszki: *Analysis of link cost functions for PNNI routing*, In proc. of the 6th IFIP Workshop on Performance Modeling and Evaluation of ATM Networks, pages 50/1–50/10, Ilkley, UK, July 20–22, 1998.

[C3] Á. Magi, B. Szviatovszki, G. Rózsa, L. Németh: *Novel Performance Evaluation of PNNI Equipment*, In proc. of the XVII World Telecommunications Congress incorporating ISS'2000, Birmingham, UK, May 7–12, 2000.

[C4] B. G. Józsa, D. Orincsay, Á. Magi, B. Szviatovszki: *On the Use of Trunk Reservation in PNNI Routing*, In proc. of the IEEE Conference on High Performance Switching and Routing, (ATM2000), pages 135–139, Heidelberg, Germany, June 26–29, 2000.

[C5] A. Jüttner, B. Szviatovszki, Á. Szentesi, D. Orincsay, J. Harmatos: *On-demand optimization of label switched paths in MPLS networks*, In proc. of the 9th International Conference on Computer Communications and Networks, IEEE ICCCN 2000, pages 107–113, Las Vegas, USA, October 16–18, 2000.

[C6] A. Jüttner, B. Szviatovszki, I. Mécs, Zs. Rajkó: *Lagrange Relaxation Based Method for the QoS Routing Problem*, In proc. of the IEEE Infocom 2001, pages 100–109, Anchorage, Alaska, USA, April 22–26, 2001.

[C7] B. Szviatovszki, Á. Szentesi, A. Jüttner: *Minimizing Re-Routing in MPLS Networks with Preemption-Aware Constraint-Based Routing*, In proc. of the 2001 International Symposium of Performance Evaluation of Computer and Telecommunication Systems, Spects 2001, pages 249–261, Orlando Florida, July 15–19, 2001.

[C8] B. Szviatovszki, Á. Szentesi, A. Jüttner: *On the Effectiveness of Restoration Path Computation Methods*, In proc. of the IEEE International Conference on Communications, New York, April 28–May 2, 2002.

Patents

[P1] B. Szviatovszki, Á. Szentesi, A. Jüttner: *Priority-Based Path Selection in a Data Network Control*, (US Patent Application No. P14200US, 2001.)