RSVP-TE daemon for DiffServ over MPLS under Linux

1 Abstract

The RSVP-TE daemon for DiffServ over MPLS under Linux project supports the important Internet Engineering Task Force (IETF) standards for the set up of MultiProtocol Label Switching (MPLS, RFC3031) tunnels with DiffServ support (DS, RFC2475) under Linux by using the ReSource reserVation Protocol (RSVP, RFC2205). These tunnels support scalable Quality of Service (QoS) in IP networks. While the project might be very specialized for the typical Linux user, it is used by ISPs, network operators and research institutes all over the world. For some of them this project is a "killer application" making them look at Linux for the first time.

The project reuses the code of a few existing projects (some of them abandoned) and created a useable RSVP MPLS daemon for Linux. The project deliberately uses an open, bazaar model with frequent releases to leverage on the ever-growing user base.

This paper will address the theoretical basis of MPLS, RSVP and DiffServ. It will elaborate on the architecture and used components both in user and kernel space (netfilter, netlink, scheduling and queueing, MPLS).

The paper concludes by investigating the pros and cons of open sourcing a research project like this that is traditionally developed in house or in a closed group. The comparison is based on a use case: the public demonstration of MPLS technology as proof-of-concept. We compare the close model used in the Ithaci project with the open source project code that was used in the Tequila demo. The author was responsible for the MPLS signaling software in both demos.

2 DiffServ over MPLS

This section will describe MPLS and DiffServ over MPLS from a theoretical point of view. Those that are familiar with these techniques or those that are only interested in the Linux specific issues can skip directly to section 3.

2.1 IP and MPLS

The Internet Protocol (IP) is a connection-less networking layer protocol to route IP packets over a network of IP routers. Every router in the IP network examines the IP packet header and independently determines the next hop based on his internal routing table. A routing table contains information about the next hop and the outgoing interface for the destination address of each IP address. MPLS allows the setup of Label Switched Paths (LSPs) between IP routers to avoid the IP forwarding in the intermediate routers. IP packets are tagged with labels. The initial goal of label based switching was to increase the throughput of IP packet forwarding. Label based switching methods allow routers to make forwarding decisions based on the contents of a simple label, rather than by performing a complex route lookup based on destination IP address. This initial justification for MPLS is no longer perceived as the main benefit, since nowadays routers are able to perform route lookups at sufficiently high speeds to support most interface types. However, MPLS brings many other benefits to IP-based networks, they include: (i) traffic engineering, i.e., the optimization of traffic handling in network (ii) Virtual Private Networks (VPNs), networks that offer private communication over the public Internet using secure links and (iii) the elimination of multiple protocol layers.

MPLS paths are constructed by installing label state in the subsequent routers of the path. Labels are fixed length entities that only have a local meaning. Labels are installed with a

label distribution protocol. The MPLS forwarding of the IP packets is based on these labels. The IP and MPLS forwarding principles will be detailed first, followed by a description of the MPLS label distribution process.

2.2 Forwarding in IP and MPLS

In regular non-MPLS IP networks packets are forwarded in a hop by hop manner. This means that the forwarding decision of a packet traversing the network is based on the look-up of the destination in the local routing table (also called Routing Information Base, RIB). Figure 1a illustrates IP for a network consisting of four routers: A, B, C and D. A simplified IP routing table of router B is shown. It consists of entries, which map the destination network addresses of the IP packets to the IP addresses of the next hop and the router interface, which is connected to the next hop. When forwarding a packet, a router inspects the destination address of the packet (found in the IP header), searches through his local router table via a longest prefix match and forwards it to the next hop on the outgoing interface.

The destination addresses in this table are aggregated in order to reduce the number of entries in this table. These entries are aggregated by indicating the length of the significant part of the destination addresses (from 0 to 32 bits). If *n* is the length of address *a* then only the first *n* (most significant) bits of *a* are considered. The resulting partial address is called a prefix and is noted as a/n (e.g. 10.15.16.0/24). This aggregation of addresses has the drawback that searching through the table needs to be done with a *longest prefix* match. A longest prefix match is more complex than an exact match because the result of the search must be the entry with the longest prefix that matches the address (see [ROUTIP]).

An important characteristic of IP forwarding is that packets arriving at a router with the same destination prefix are forwarded equivalently over the network. A class of packets that can be forwarded equivalently is a Forwarding Equivalence Class (FEC). Because of the destination based forwarding of IP FECs are usually associated with IP prefixes. The forwarding in an IP router can be restated as the partitioning of packets in FECs and assigning a next hop to the FECs. It is important to note that the determination of the FEC needs to be done in every hop for every packet.

On the other hand, MPLS forwarding relies on labels, instead of prefixes to route packets through the network (see [RFC3031]). Labels are fixed length entities that have only a local meaning. Because a label only has a local meaning they can be different at every hop and therefore needs to be adapted before forwarding the packet, this process is called label switching. The labels are distributed over the MPLS domain by means of a label distribution protocol. MPLS routers are called Label Switching Routers (LSR) (Figure 1c) because they operate on labels rather than on IP prefixes when forwarding packets. The concatenation of these installed labels in the different LSRs is called a Label Switched Patch (LSP). An LSP is set up between the ingress LSR and the egress LSR, these edge LSRs are also called Label Edge Routers (LER). Packets belonging to a certain FEC are then mapped on an LSP. Determining the FEC of a packet is only necessary in the ingress of the LSP. The segregation of packets in FECs needs only be done once, in the ingress router, and this segregation can also be based on more than the destination prefix of the packet. For example, it is possible to take both the source and the destination into account. Because LSPs are based on FEC-to-label binding makes that an LSP is a unidirectional path.

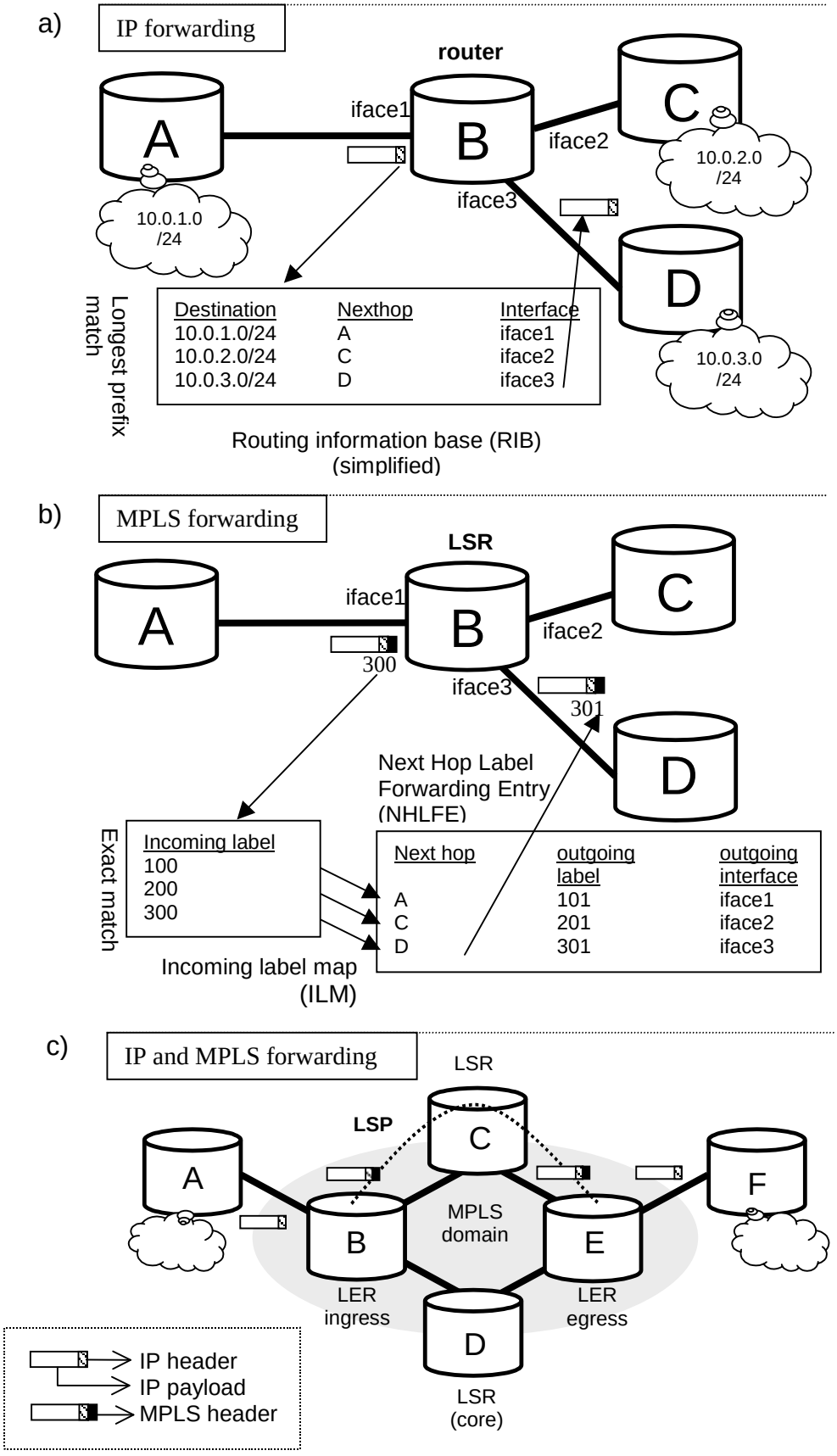

Figure 1: IP and MPLS forwarding

In the case of label switched routers, every router contains two tables: an Incoming Label Map (ILM) table that contains all the incoming labels the router has allocated and a table that contains all the necessary information to forward a packet over an LSP (Figure 1b). The latter table is populated with Next Hop Label Forwarding Entries (NHLFE). There is a mapping between the ILM and an NHLFE mapping the incoming labels to an output label, the outgoing interfaces and the next hop. The router inspects the incoming label and consults the ILM table to find the right NHLFE. This NHLFE contains the outgoing label, the next hop and the outgoing interface. Before the packet is sent to the next hop, the label is switched to the outgoing label value.

Two distinct cases can be distinguished: (i) Hop-by-Hop routed LSP setup and (ii) Explicit Routed LSP setup. These two cases will be described in the subsequent sections.

2.3 Label distribution in MPLS

(i) Shortest path Routed LSP setup

To distribute labels over the network and consequently setup an LSP a *label distribution protocol* is used. Path setup typically consists of two steps: (i) a request is sent to the egress of the LSP and (ii) the response propagates back to the ingress. The first step is denoted by the generic term “Label Request” whereas the second step is denoted by the term “Label Mapping”. Figure 2a illustrates the label distribution process. When LER A wants to setup an LSP to network *netD*, it will send a Label Request to its next hop towards netD (step a, Figure 2). The intermediate nodes from the ingress towards the egress (like LSR B) will install state about the request and will forward the request towards D according to their routing information bases (step b). When the request reaches the destination of the LSP, the egress node will allocate a label for this LSP and will store this information in the Incoming Label Map (ILM). The LSR will then send a Label Mapping back to the previous hop. The Label Mapping message contains the label previously allocated by the LSR (step d). LER B will then receive the Label Mapping from node D. The label contained in the Label Mapping will be used to make a Next-Hop Label Forwarding Entry (NHLFE). B will then, in his turn, allocate a label and store this label in his ILM. The information in the ILM (incoming label) and the NHLFE (outgoing label) is combined, effectively storing the information about the label switch (step e). After allocating the label and storing the relevant information, LSR B will send a Label Mapping to his previous hop (step f). Finally, the initiator of the LSP setup (node A) will receive the Label Mapping from its next hop. LSR A will store this information in a NHLFE. This ingress LER will then map traffic to the newly established LSP by mapping a class of packets (FEC) to the LSP, which implies that traffic that belongs to this traffic class will be forwarded over the LSP. The FEC is thus mapped on the NHLFE (step g). All the FEC to NHLFE mappings are stored in the FEC to NHLFE map (FTN). The FTN is used by the ingress of an LSP to forward the packets belonging to a certain FEC over the LSP (to find the outgoing label).

Because the request is forwarded according to the local RIB of the intermediate routers, the resulting LSP is called a hop-by-hop routed LSP. Another type of LSP is called an explicit routed LSP (ER-LSP).

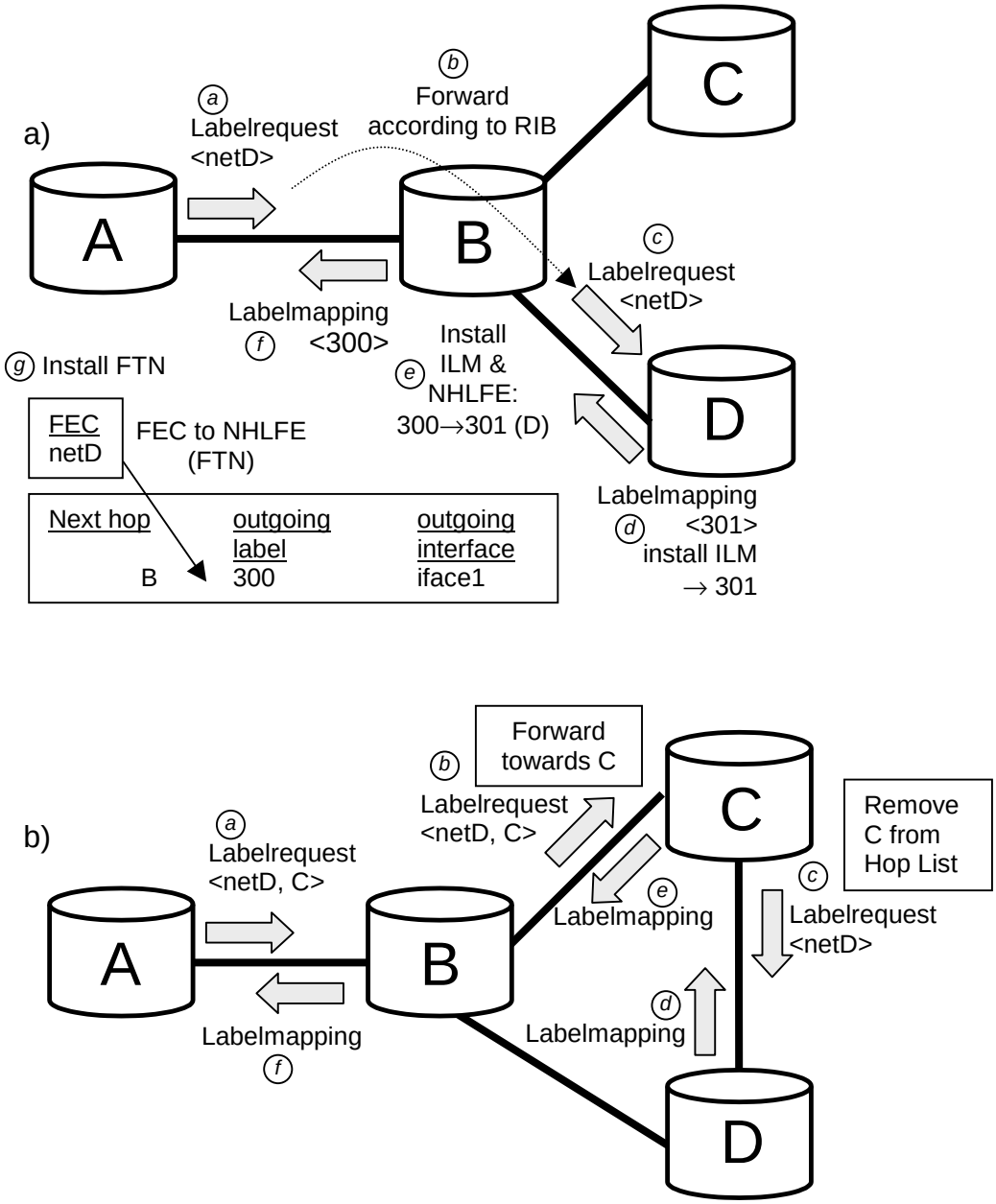

Figure 2: a) example of hop-by-hop routed label distribution and b) example of explicitly routed label distribution.

(ii) Explicit Routed LSP setup

The real power of MPLS lies in the fact that paths can be setup with a great deal of flexibility. An example is an Explicit Routed LSP (ER-LSP). Explicit Routed means that some or all of the hops between the ingress and the egress of the LSP can be specified. Figure 2b illustrates this, in step (a) LSR A sends a Label Request for *netD* and the Label Request explicitly states that the LSP should be routed along node C. Node B will receive this Label Request and will forward it towards C along the shortest path (step b). When LSR C receives this Label Request it will remove itself from the Hop List in the Label Request and forwards the Label Request towards the destination. From then on the LSP setup continues as detailed in Figure 2. It is important to note that every node keeps state about the Label Request so that the Label Mappings are sent to the correct previous hop, i.e. the hop it received the corresponding Label Request from.

2.4 DiffServ over MPLS

There are two major approaches to Quality of Service in IP: Integrated Services (IntServ) and Differentiated services (DiffServ). IntServ associates and allocates resources to a single flow. Obviously requires per-flow state in every router on the path from ingress to egress. This has inherent scalability problems certainly in backbone networks with thousands of flows running through them.

In order to solve the IntServ scalability problem, Differentiated Services (DiffServ) classifies packets into a limited number of *classes* and therefore does not need for per-flow state or per-flow processing. The identified traffic is assigned a value, a DiffServ code point (DSCP). A DiffServ Behavior Aggregate (BA) is a collection of packets with the same DiffServ code-point (DSCP) crossing a link in a particular direction. A Per-Hop-Behavior (PHB), the externally observable forwarding behavior, is applied to a behavior aggregate.

The classification is usually based on multiple fields in the IP header (Multi-Field, MF classification) at the edge and based on the DiffServ CodePoint (Behavior Aggregate, BA classification) in the core of the network (see Figure 3b).

An example PHB is Expedited Forwarding (EF) which offers low loss, low delay and low jitter with an assured bandwidth. This means that the collection of packets marked with the EF code-point traversing a link in a certain direction (BA) will receive low loss, delay, jitter and an assured bandwidth. The Assured Forwarding (AF) PHB group is a group of PHB. A PHB of the AF group is noted as AFxy, where x is the class and y is the drop precedence. Packets belonging to a different AF class are forwarded separately. Usually more resources are allocated to the lower classes. Packets within a class that have a higher drop precedence will be dropped before packets with a lower drop precedence.

An Ordered Aggregate (OA) is the set of Behavior Aggregates that share an ordering constraint. This means that packets that belong to the same OA must not be reordered. When looking at DiffServ over MPLS it immediately becomes apparent that packets that belong to a certain OA must be mapped on the same LSP, otherwise this ordering constraint can not be enforced. This is trivial if only one PHB is applied to the ordered aggregate. However, PHBs can be grouped in a Per-hop-behavior SCheduling group (PSC). A PSC is the set of one or more PHB(s) that are applied to a given OA. For example, AF1y is a PSC comprising the AF11, AF12 and AF13 PHBs. Combining the notion of OA and PSC means that in DiffServ over MPLS OA-PSC pairs will be mapped on LSPs. If the PSC contains more than one PHB this means that it must be possible for an LSR to enforce different PHBs to the packets that belong to the same LSP. This in turn means that the LSR must have some information in the packet header to determine the PHB to be applied, this information must be encapsulated in a

way that is accessible to the LSR and thus must be part of the label or shim header. The next subsection will discuss how to encapsulate the PHB.

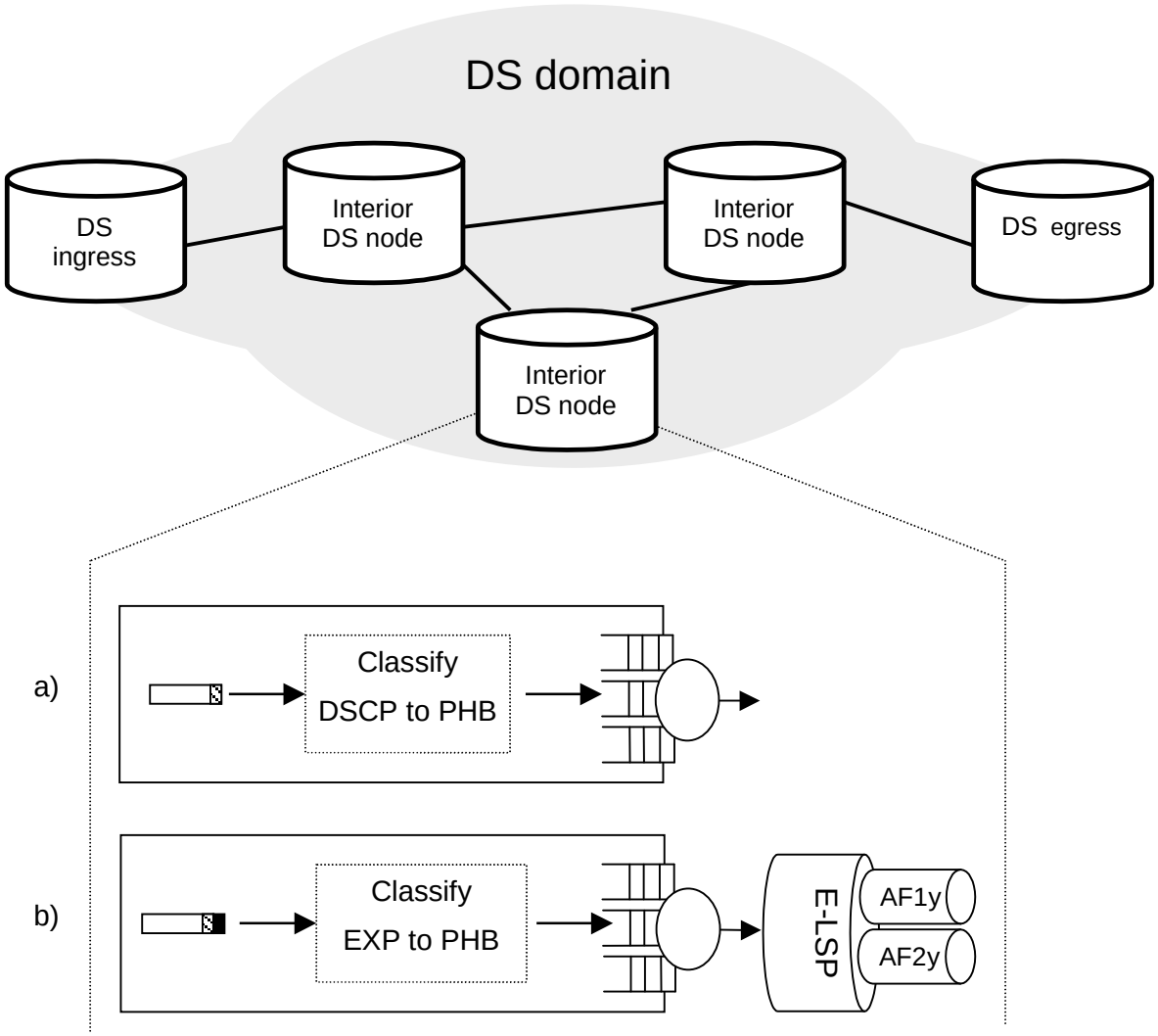

Figure 3: BA classification in DiffServ: a) classification on the DSCP value of the IP header, b) classification on the EXP bits of the MPLS shim header

Encapsulating the PHB

In IPv4 the "Type of Service" (ToS) field or in IPv6 "Traffic Class" field is used to encapsulate the DSCP. With generic MPLS encapsulation there is a mapping from the IP DSCP space to the Experimental (EXP) field of the shim header (see Figure 4). The DSCP field uses the 6 most significant bits of the 8 bits of these IP header fields. Since the DSCP field is 6 bits wide it can comprise 64 different values. However, the EXP field of the shim header is only 3 bits wide so it can only comprise 8 different values. This means that the mapping from DSCP to EXP value can not be a one-to-one mapping. This is quite a problem because currently there are more than eight defined DSCP values (Best Effort, 12 AF values and EF). If the DiffServ domain uses less than 8 different DSCP values then the mapping between DSCP and EXP can be fixed over the domain. If the domain uses more than eight different code points then the mapping must be explicitly defined on a per-LSP basis.

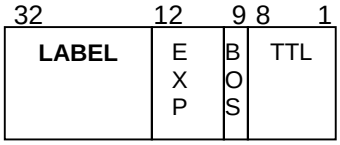

Figure 4: MPLS shim header

When the EXP value is used to indicate the set of PHBs applied to an OA (the PSC) then we call this an EXP-Inferred-PSC LSP or E-LSP for short. This means that the PSC is inferred from the EXP value in the shim header (see Figure 3b). Since the daemon currently only supports E-LSPs will not describe the alternative of Label-Infered-PSC LSPs or L-LSP (see [WILEY] and [RFC3270]).

2.5 RSVP-TE

The "Extensions to RSVP for LSP tunnels (RSVP-TE)" protocol is an extension to the Resource Reservation Protocol (see [RSVP]), a signaling protocol originally developed for IntServ reservations. RSVP-TE extends RSVP with (i) the possibility to set up LSPs and ER-LSPs and (ii) adds traffic engineering functionality.

LSPs are set up by piggy-backing Label Request objects on RSVP's native PATH messages. The labels are then mapped by piggy-backing a Label object on the RESV messages sent in the opposite direction (see Figure 2a,b and replace the Label request and Label Mapping messages with PATH and RESV messages respectively). For more information we refer to RFC3209 [RFC3209].

3 The RSVP-TE for DiffServ over MPLS under Linux project

This section describes the features of the daemon, how the daemon was merged from different projects and the resulting architecture. The section ends with a description of how a packet is forwarded through the forwarding plane.

3.1 Features

First of all the daemon supports the set up of Label Switched Paths (LSPs) in the network according to the IP routing tables or by explicitly specifying the hops to be traversed (shortest path LSPs and Explicitly Routed-LSPs respectively). There's an RSVP API (RAPI), an API to aid developers to build custom applications that interact with the RSVP daemon. There is support for DiffServ allowing to differentiate the forwarding behavior based on the value in

the EXP field of the MPLS header. There is also support for IntServ where resources are explicitly allocated on a per-LSP level. Traffic can be mapped on the LSPs very flexibly based on the destination address, protocol, destination ports and port ranges of the IP packets. There is also the ability to trace an LSP, comparable to IP's traceroute. It checks the route taken by an LSP by probing the routers along the path. Resilience is also supported with LSP rerouting and LSP protection switching.

3.2 Merged components

The daemon is based upon the Nistswich version 2.0 daemon for Free BSD by USC (Figure 5: RSVP-MPLS BSD) and a port of an IntServ RSVP daemon to Linux by Alexey Kuznetsov (Figure 5: RSVP Linux). Both daemons are based on the same code base (ISI RSVP implementation, Figure 5: RSVP BSD) but they forked a while ago. This effort combines the daemons again so that the MPLS support found in the Nistswitch version is now available on Linux. Moreover support for DiffServ over MPLS (DS/MPLS) is also added (Figure 5: DiffServ extension).

The MPLS Linux kernel code is based upon mpls-linux by James R. Leu (Figure 5: MPLS Linux). Our release originally added kernel support for DiffServ over MPLS support, the use of multiple routing tables and LSP byte and packet counters. However the more recent v1.1 branch adds this functionality so we are using unpatched version now.

Figure 5 illustrates the merge process. We started by extracting the MPLS extensions from the Nistswitch version and applying these extensions to Alexey's Linux RSVP daemon. We then added support to daemon for James Lieu's Linux MPLS patches. Finally we added our own support for DiffServ over MPLS and flexible traffic mappings.

Other small patches are required to iptables (DSCP based matching) and tc (support for the MPLS protocol).

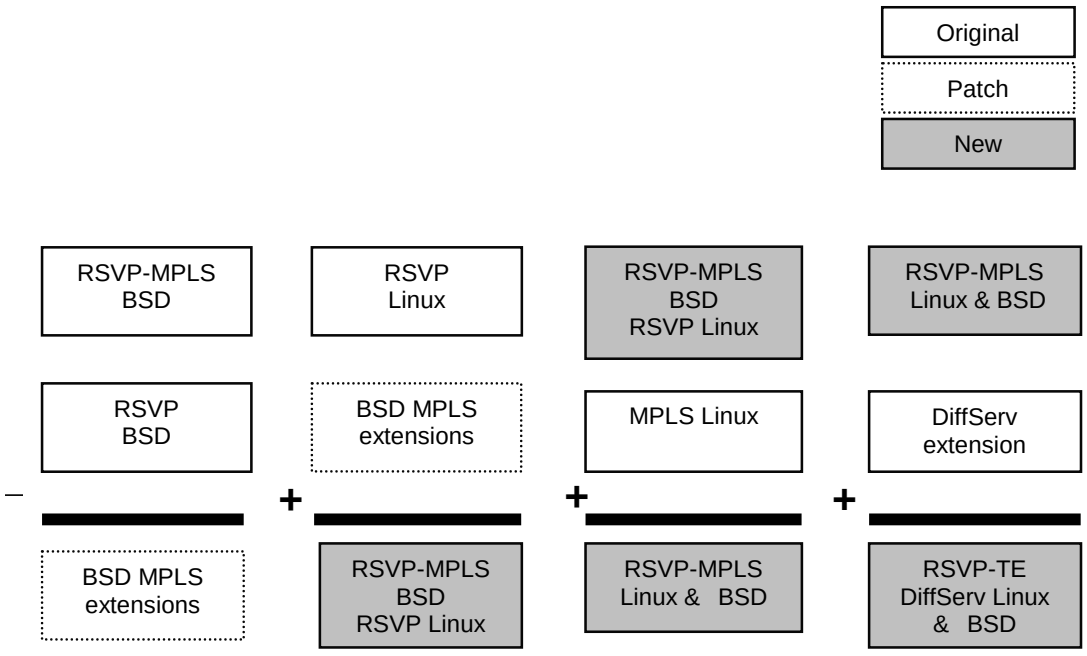

Figure 5: Software Merge Map

3.3 The overall architecture

The overall architecture (Figure 6) consists of a number of components both in user space and kernel space. The important parts of the kernel that are used are netfiler to classify the packets, QoS and faire queuing to support differentiating between flows and of course MPLS support.

The prime user space component is the RSVP daemon that is responsible for the RSVP signaling and the maintenance of the MPLS state. The daemon is responsible for the allocating and installation of the MPLS labels during LSP set up and freeing and removing labels on LSP tear down.

Two components use the RAPI: "rtest" and "rapirecv". rtest is an application that takes LSP requests and issues them to the daemon. rapirecv is an application that receives label requests at the egress and dictates the daemon to send a response back to the ingress. rtest2 and rapirecv_auto (not shown in the figure) are extended version of rtest and rapirecv respectively that support the automatic set up of a (large) number of LSPs.

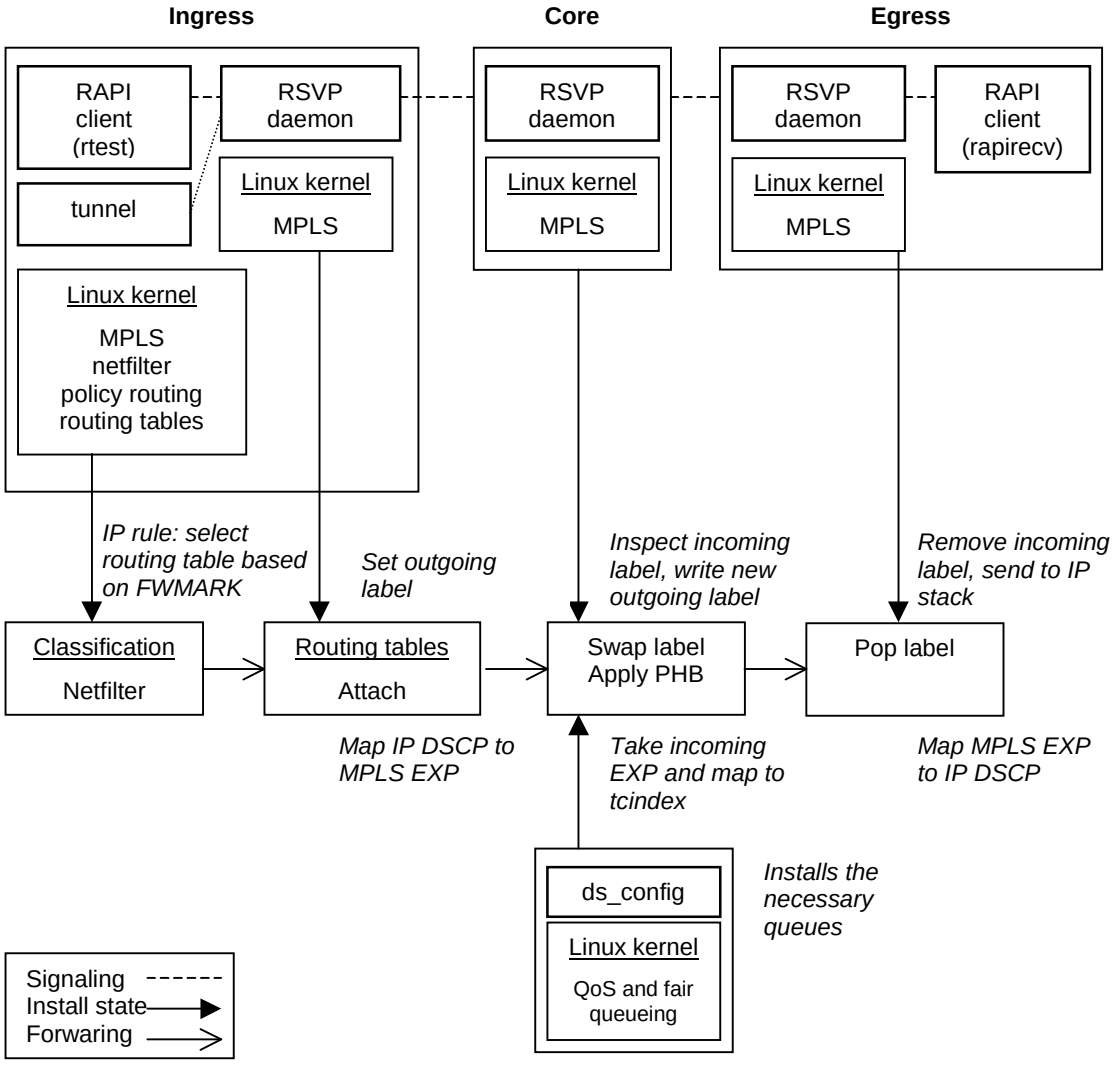

Figure 6: DiffServ over MPLS using RSVP-TE under Linux overall architecture

"Tunnel" is an application that maps traffic on an existing LSP. Mapping can be based on destination address, protocol, destination ports and port ranges (TCP and UDP). So basically anything that netfilter supports. Tunnel sends status packets requests (mstat packets) to the RSVP daemon in order to receive information about the installed state in the daemon (existing sessions, labels, reservations etc.). For example mstat packets are used to trace an LSP or to automatically send a RESV message at the egress.

3.4 The forwarding path explained

In the ingress the packets are classified with netfilter (refer to Figure 6). Packets are filtered on the OUTPUT and PREROUTING chain of the mangle table. The mangle table is needed because the fwmark needs to be set. The OUTPUT and PREROUTING chains are used in order to filter on both locally generated and incoming traffic. Based on the value of the fwmark a routing table is selected (using policy routing). In the resulting routing table there is a MPLS tunnel interface acting as the default gateway. The tunnel interface encapsulates the packets on the LSP by attaching the correct outgoing label. The incoming DSCP is mapped to the EXP field in the MPLS header. A limitation of this architecture is that netfilter can only write (mark) on the mangle table and that only a single mark operation is possible. So while we do can map traffic on the LSP also setting the DSCP at the same node is impossible. The solution is to set the DSCP before the traffic enters the ingress LSP.

In the core node the MPLS stack inspects the incoming label and sets the new outgoing label and next hop. At the DiffServ level the current EXP value of the packet is inspected. There is a mapping from this EXP value to a tcindex. The tcindex in turn determines the correct outgoing queue so the correct forwarding behavior (PHB) can be applied.

Finally in the egress the incoming EXP field is mapped to the DSCP field and then the MPLS header is stripped off and the packet is sent to the IP layer.

4 Open source community issues

This section covers some less technical issues related to the consequences (both positive and negative) of publishing a work as an open source project.

4.1 Motivation

The decision to open-source the project was motivated by a number of factors (given in a random order). A first reason was to prevent others from having to integrate the daemons all over again or to write RSVP-TE code from scratch. At same time while you prevent others from doing that you can also find developers and users willing to debug and develop the system. Starting an open-source project was also seen as nice way to create some publicity for our department "INTEC", the European project "Tequila" and ourselves. Another reason was that dissemination of information should be an important task of research institutes. And finally we wanted to give back to the community and strengthen the networking offering of Linux.

Now let's have a look at how we can reflect on these factors a year after the start of the project. I believe that the project has indeed prevented others from building from scratch or integrating a new daemon. The project has attracted a number of external developers and quite a few users. These users have helped to track down numerous bugs and developers have contributed important code parts and a signification number of bug fixes. The project also led to a number of new opportunities although it is difficult to assert which opportunities were caused by the project and which were not.

4.2 A look at the pros and cons

It will sum up some of the advantages of open-sourcing a project as opposed to working quietly on you own. I will start with the disadvantages. I first of all maintaining a web site and mailing list is a non-trivial task. In my particular case I am more or less forced to maintain a public and a private tree which obviously increases overhead. Also certain types of questions on the mailing list tend to involve quite a bit of work. For instance bug reports often involve looking through debug backtraces, daemon log files and MPLS kernel state. Probably the most annoying things are people asking questions that already have been addressed in a mailing list thread or in the FAQ.

On the positive side, one of most often mentioned benefits is that extra users lead to finding bugs more quickly. This is undoubtedly true. For example some bugs are only triggered by specific configurations. Also when somebody stumbles on an error before you, you can fix the bug when you want to, often before it is on your critical path. Complex software systems also have the property that fixing a certain problem can also fix another, on first sight unrelated, problem. This can lead to the situation where you fix a bug for a user and as a side-effect fix a bug you have been chasing down unsuccessfully till now.

Programming in an open environment tend to influence your coding style too. You rely less on hacks because you know that a hack may cause problems on other configurations and if that happens your users will report that back to you. An open-source project also requires that you keep at least a minimal set of documentation (installation instructions, change log and FAQ).

Finally there are the external developers. If have been lucky enough to have received a good deal of bug fixes but also important functionality enhancements from external developers.

4.3 Case study

We will now investigate the pros and cons of open sourcing a research project like this that is traditionally developed in house or in a closed group. The comparison is based on a use case of the public demonstration of MPLS technology as proof-of-concept. Both of the demonstrations were part of the experimentation activity of European Commission sponsored projects. We compare the closed model used in the ACTS Ithaci project [ITHACI] with the open source project code that was used in the IST Tequila project's demo [Tequila]. The author was responsible for the MPLS signaling software in both demos.

The European projects like Ithaci and Tequila are run by a consortium that typically consists of research institutes, private companies and universities. Within a limited time period they try to tackle a specific problem space. In the case of Ithaci this was IP switching and MPLS technology with a special focus on multicast. In the case of Tequila this is Traffic Engineering and QoS in large scale networks. The projects are typically split up in a number of work packages. The work conducted in these packages can vary but both projects took a similar approach. The first package handled the administration and management tasks, the second package the theoretical research, the third package involves the development of the experimentation platform and the last package the integration and experimentation itself. The European Commission representatives and the auditors usually encourage the projects conduct a public demonstration of the developed system.

The focus of the Ithaci demonstration was a testbed where unicast and multicast MPLS was run simultaneously on the same routers. During the Ithaci project we used a closed source approach even between the partners of the consortium. The MPLS stack was written by one partner and made available to us. The code was a small kernel patch and binary module. We were responsible for the unicast LDP (Label Distribution Protocol) signaling daemon. We tested our daemon in our own network prior to the integration meeting. During the integration

meeting we took the components from every partner and installed and integrated them in to a large network. The integration was the preparation step before the actual demonstration. During the integration of the demo numerous bugs were discovered and we basically needed a whole week to straight things out. This is not surprising and we were well prepared and the demo was very successful. It however does contrast with the approach taken in the Tequila project.

For the Tequila demo we were, among other things, responsible for the MPLS signaling daemon. Because the daemon was open-source, it already received diverse and thorough testing. Not only the daemon itself but also the installation instructions (which are not trivial and involve patching the kernel, tc and iptables). Some partners already got to test the daemon at their premises before the actual integration meeting. This did not create any additional overhead for us because it was sufficient to point our projects' URL. Integrating the RSVP daemon in the final experimentation platform only revealed two bugs. Both of them were related to the embedding of the applications rtest and tunnel (see section 3.3) software in a Genertic Adaptation Layer (GAL). (The GAL is used to make abstraction of the router used, the GAL supports Linux routers, Cisco and an experimental router based on a fast hardware based translator (IFT) and Linux [IFT].)

It is an oversimplification to state that the significant lower number of bugs discovered in the signaling daemon during integration meeting of the Tequila project is solely caused by the public availability of its source code. However distributed testing of code will lead to bugs being found more early and to more bugs being exposed so it undoubtedly played an important role. As a personal note I would like to add that I never felt so confident about my code during a demo then at the Tequila demo.

5 About the authors

Pim Van Heuven graduated in Computer Science at the Ghent University in 1998. In July 1998, he joined the INTEC Broadband Communications Networks (IBCN) Group where he is preparing a Ph.D. His research interests include mainly the area of Quality of Service, Traffic Engineering and Rerouting in IP. He is currently active in the IST Tequila project, before he worked on the ACTS Ithaci project. He has been a Linux enthusiast since 1996. In 2001 he open-sourced the DiffServ over MPLS for Linux project he had been working on. He published several papers on MPLS and rerouting techniques in IP and WDM networks.

Steven Van den Berghe joined IBCN in July 1999. He is focusing on measurement-based Traffic Engineering in a DiffServ/MPLS/MultiPath environment. He is also active in the IST Tequila project. He knows his way around the Linux MPLS kernel stack and published, next to several papers, an Internet Draft on the requirements for measurement architectures for use in Traffic Engineered IP Networks.

Jan Coppens joined the IBCN group in 2001. He specializes in (Linux) Traffic Control mechanisms and is responsible for the Linux part of a generic adaptation layer (GAL), an abstraction layer for different router platforms, in the Tequila project.

Piet Demeester he has been active in the research on broadband communication networks since 1992. He published over 400 papers and has been member of numerous technical program committees. His current interests include optical circuit and packet switching, network resilience, Quality of Service in IP based networks, IP based mobile and wireless networks, peer-to-peer and active networking, grid computing, network and service management.

6 References

[DSMPLS] P. Van Heuven, S. Van den Berghe, J. Coppens, P. Demeester, "RSVP-TE daemon for DiffServ over MPLS under Linux", [Online], http://dsmpls.atlantis.rug.ac.be, web site.

[IPOM] P. Van Heuven, S. De Maesschalck, D. Colle, S. Van den Berghe, M. Pickavet, P. Demeester, Recovery in IP based networks using MPLS, IEEE Workshop on IP-oriented Operations & Management IPOM' 2000, ISBN 83-88309-00-5, 70-78, 2000.

[IFT] C. Duret, F. Rischette, J. Lattmann, V. Laspreses, P. Van Heuven, S. Van den Berghe and P. Demeester; High Router Flexibility and Performance by Combining Dedicated Lookup Hardware (IFT), Off-the-Shelf Switches and Linux; Networking 2002, Pisa, Italy, 19-24 May 2002. [Online]. http://www.ist-tequila.org/publications/routers-networking2002.pdf

[ITHACI] I. Andrikopoulos, G. Pavlou, P. Georgatsos, N. Karatzas, K. Kavidopoulos, J. Rothig, S. Schaller, D. Ooms, P. Van Heuven, Experiments and enhancement for IP and ATM integration: The IthACI project, IEEE Communications Magazine, Vol. 39, Nr. 5, 146-155, 2001.

[RFC3031] E. Rosen, A. Viswanathan, R. Callon, Multiprotocol Label Switching Architecture. (January 2001). IETF RFC. [Online]. http://www.ietf.org/rfc/rfc3031.txt

[RFC3209] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow. (December 2001), RSVP-TE: Extensions to RSVP for LSP Tunnels, IETF RFC. [Online], http://www.ietf.org/rfc/rfc3209.txt.

[RFC3270] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen. (May 2002), Multi-Protocol Label Switching (MPLS) Support of Differentiated Services, [Online], http://www.ietf.org/rfc/rfc3270.txt

[ROUTIP] C. Huitema, Routing in the Internet, New Jersey: Prentice Hall, 1995.

[RSVP] David Durham and Ray Yavatkar, Inside the Internet's Resource reSerVation Protocol, New York: Wiley, 1999

[TEQUILA] Traffic Engineering for Quality of Service in the Internet, at Large Scale (TEQUILA), [Online], http://www.ist-tequila.org/

[WILEY] P. Van Heuven, S. Van den Berghe, F. De Turck, P. Demeester, "Wiley encyclopedia of Technology" (MPLS section), Wiley and Sons, to be published.

7 Acknowledgments

Part of this work has been supported by the European Commission through the IST project TEQUILA and by the Flemish Government through an IWT scholarship. Part of this text (section 2) will also appear in the upcoming Wiley Encyclopedia of Technology as part of the MPLS section. The authors would like to thank all the users of the project who went to the trouble to track down bugs in the daemon and especially to those that have contributed additional functionality and bug fixes.