An Experimental Analysis of Diffserv-MPLS Interoperability

S. Avallone[1], M. Esposito[1,2], A. Pescapè[1,2], S. P. Romano[1], G. Ventre[1]
[1]Università degli Studi "Federico II" di Napoli
DIS, Dipartimento di Informatica e Sistemistica
Via Claudio, 21
80125, Napoli
tel +39 081 7683908
fax. +39 081 7683816
{stefano.avallone, mesposit, pescape, spromano, giorgio}@unina.it
[2] ITEM - Laboratorio Nazionale CINI per l'Informatica e la Telematica Multimediali
Via Diocleziano, 328
80125, Napoli
{marcello.esposito, antonio.pescape}@napoli.consorzio-cini.it

Abstract

In this paper we present some interesting research studies conducted over both a Differentiated Services (Diffserv) and a Multi Protocol Label Switching (MPLS) experimental testbed. Diffserv operation over an MPLS backbone is also investigated. The trials have been performed on a network made of Linux routers and the measures have been collected by exploiting state-of-the-art open source software, either available from other researchers or developed at our own University.

Keywords: Diffserv, MPLS, Diffserv over MPLS.

1. Introduction

This paper presents an experimental approach to the study of the interoperability between the Diffserv and MPLS paradigms: this piece of work represents part of a more comprehensive research aimed at designing and implementing a general framework for the effective negotiation and delivery of services with quality assurance. Nowadays, when discussing about enhanced Quality of Service (QoS) models, the basic assumption is that a QoS-enabled network infrastructure is available at the lower-most level. With regard to this point, Differentiated Services (Diffserv [1]) and Multi Protocol Label Switching (MPLS [2]) currently appear as the best choices. While being far from original with such a statement, it has to be noticed that a crucial factor is definitely missing inside the research community: few implementations currently exist of both such architectures and most of them represent proprietary solutions, full of device-specific contaminations. Our contribution in this field is related both to exploiting to their best state-of-the-art open source implementations of these architectures and to adding new functionality to them whenever deemed necessary. With respect to Diffserv, we became familiar with two different implementations, based, respectively, on the FreeBSD and Linux operating systems. The former is the ALTQ [3] package developed at the Sony Research Laboratories in Japan; the latter is the Traffic Control (TC) [4] [5] module (together with the TC next Generation – TCng – package, which adds lots of useful functions to TC). Such modules basically make available the fundamental traffic control components which are needed in order to realize the Diffserv paradigm: classifiers, schedulers, conditioners, markers, shapers, etc. Coming to MPLS, we utilized a brand new package running under Linux [6], which required some fixes and modifications (also in cooperation with the author) in order to function properly. We already have substantial experience with this platform, since we exploited it in order to make a thorough evaluation of its performance in a number of different situations: comparative analysis of the behavior of different protocols (ICMP, TCP, UDP) on top of MPLS, impact of label stacking in terms of protocol overhead, etc. [7]. In the following we will show some interesting experimental results obtained over a testbed set up at the University of Naples. More precisely, we will herein concentrate on the Linux implementations of both Diffserv and MPLS.
After analyzing Diffserv and MPLS behavior, we will finally expand on issues related to their interoperability: to this purpose, we will present a scenario where Diffserv runs over an MPLS backbone and we will make a comparison of the results obtained with those related to the plain Diffserv case. The paper is organized in four sections. The experimental testbed we setup in order to carry out the trials is presented in section 2. Section 3 shows the results we obtained from our experimentations and draws a comparison among the four different scenarios we analyzed: best effort, Diffserv, MPLS and Diffserv over MPLS. Conclusions and directions of future work are provided in section 4.

2. The Experimental Testbed

In this section we present an experimental testbed developed at the University of Naples with the intent of gaining experience from actual trials and experimentations. Such testbed, shown in Figure 1- The testbed used for the trials, is made of Linux routers and closely mimics (apart

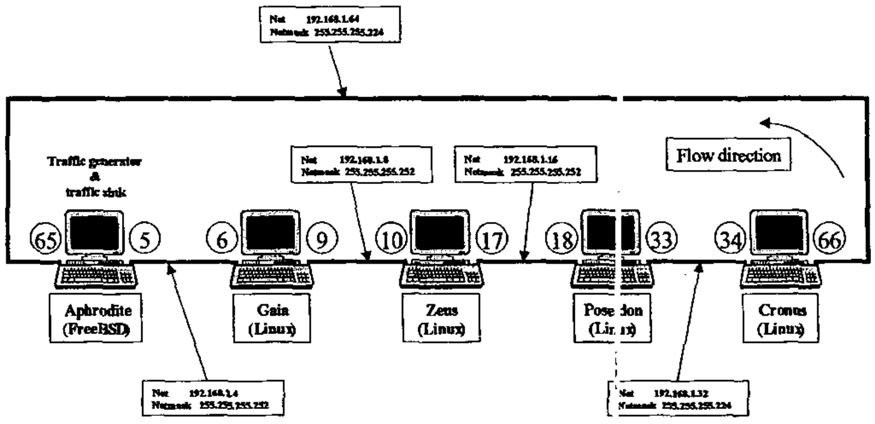


Figure 1- The testbed used for the trials

from the scale factor) an actual internetworking scenario. It is built of a number of interconnected LAN networks, each realized by means of one or more cross-connections between pairs of routers. The setup allows for communication between each network element and all the others, thanks to ad-hoc static configuration of some routing paths. For these tests, Gaia, Zeus and Poseidon represent the QoS-enabled IP infrastructure whose performance we want to evaluate; Aphrodite hosts the traffic generator and also acts as a sink. The direct connection from Cronus to Aphrodite, in fact, was added in order to allow a precise evaluation of transmission time for the packets: Aphrodite just sends data from one of its interfaces (192.168.1.5) to the other (192.168.1.65) and, provided that network routes have been appropriately configured, such data flows first to Cronus (thus crossing the QoS backbone) and then directly back to Aphrodite. Thanks to this mechanism, we are able to measure transmission times by relying on a single clock (the one residing on Aphrodite), thus avoiding any distributed clock synchronization issue.

Before digging into the detailed performance analysis for the four different configurations (Best Effort, Diffserv, MPLS and Diffserv over MPLS) let's spend a few words about the traffic pattern we adopted for all of the experiments. We used a traffic generator we developed at our university [8] [9], which lets you choose packet dimensions, average transmission rates and traffic profiles (either deterministic or Poisson), also giving a chance to set the Diffserv Code Point (DSCP) of the generated packets. An interesting feature of Mtools is the capability to reproduce the same realization of the packet generation random process, by setting the same generating seed for different trials. We fed our network with the following traffic:

- packet size: 1000 bytes;
- traffic pattern: Poisson;
- traffic duration: 30 seconds;

Aphrodite sends four different flows simultaneously:

1. 760 Kbps Expedited Forwarding (EF) traffic;
2. 1.24 Mbps Default (DE) traffic;
3. 846 Kbps Assured Forwarding traffic, with a low drop precedence (AF11);
4. 160 Kbps Assured Forwarding traffic, with a higher drop precedence (AF12);

The overall bit rate of the traffic injected into the network is thus about 3 Mbps. In order to appreciate the impact of scheduling (and in the case of Diffserv, also policing) on the network, we configured all of the routers so to use a Class Based Queuing (CBQ) scheduler, with a maximum available bandwidth of 2.8 Mbps. Since the injected traffic is more than the network can bear, packet losses will be experienced and we expect them to be uniformly distributed among the various classes of service only in the Best Effort and plain MPLS (i.e. MPLS with just one LSP carrying all the traffic) scenarios.

2.1 Best Effort scenario

In the Best Effort case, we have just one CBQ class for all of the traffic flows (which are thus scheduled without taking into consideration the DSCP inside the IP header). 2.8Mbps have been assigned to this class. Here is the TCng script used to configure network nodes in this simple case:

```
// Output interface (towards Zeus)
eth1  cbq(bandwidth  100Mbps,  avpkt  1000B,
allot 1514B, maxburst
8p) {
      class(rate 2.8Mbps, bounded) if 1
    }
```

2.2 Diffserv scenario

In this case, of the available 2.8 Mbps, 0.8 are assigned to the EF class, 1.0 to AF1 and 1.0 to default traffic (DE). Expedited Forwarding is served in a First In First Out (FIFO) fashion, while the Assured Forwarding and Default queues are managed, respectively, with a Generalized Random Early Discard (GRED) and a Random Early Discard (RED) algorithm. Following, you can find an excerpt of the traffic control script used to enforce such a configuration onto router Gaia.

```
#define EF_PROFILE      rate 1.1Mbps, burst
10kB

#define AF11_PROFILE  rate  0.9Mbps,  burst
10kB, peakrate 1.2Mbps

#define AF12_PROFILE rate 0.9Mbps, burst 5kB

#define DE_PROFILE      rate 1.5Mbps, burst
15kB

#define CBQ_QDISC_PARAM bandwidth 100Mbps,
avpkt 1000B, allot 1514B, maxburst 10p

#define FIFO_EF_PARAM limit 10kB

#define  RED_DE_PARAM  bandwidth  1.0Mbps,
limit  120kB,  min  10kB,  max  100kB,  avpkt
1000B, burst 11kB, probability 0.3

#define GRED_AF1_PARAM  bandwidth 1.0Mbps,
limit  120kB,  min  30kB,  max  100kB,  avpkt
1000B, burst 35kB
```

Please notice that in this case Gaia acts as the ingress Diffserv edge router and has thus been configured with a policer, whose function is to either drop or remark out-of-profile packets.

2.3 MPLS scenario

In a pure MPLS network, packets are assigned to the various Label Switched Paths (LSPs) based on information such as sender's and receiver's IP addresses. More specific fields, such as the DSCP (i.e the Type Of Service byte of the IP header), are completely ignored.
In this scenario we will thus rely on a single LSP to carry all of the traffic flows that enter the MPLS cloud. This means that all packets will be marked with the same label and will experience the same treatment. When leaving the MPLS backbone (i.e. upon arrival at Poseidon) the MPLS header is bumped off and the packet is passed to the IP module for further forwarding.

2.4 Diffserv over MPLS scenario

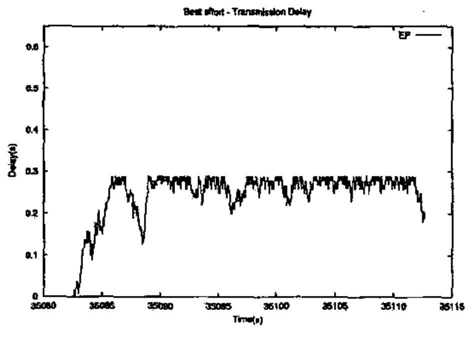
As opposed to the previous case, this scenario provides for MPLS support to Diffserv, that is to say packets are forwarded via MPLS label swapping, but different packet flows (as identified by the DSCP code) are treated in a different fashion. This is achieved by inserting additional information into the MPLS header. Here we will exploit one LSP for EF and DE traffic and a different one for the two AF1 flavors. In the first case, information about the Per Hop Behavior will be encoded in the EXP bits of the MPLS header, thus creating a so-called E-LSP [10]; on the other hand, for the Assured Forwarding flows (which have to be scheduled in the same fashion) the EXP bits carry information related to the drop precedence level (L-LSP). For this trial, again, router Gaia has to perform policing at its ingress interface.

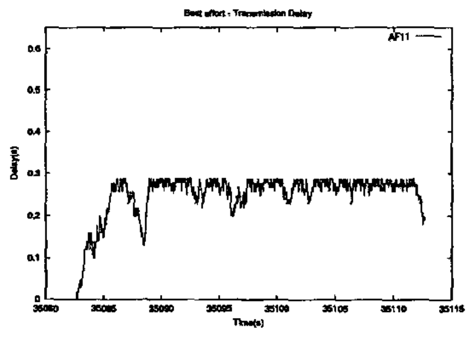
3. Experimental results

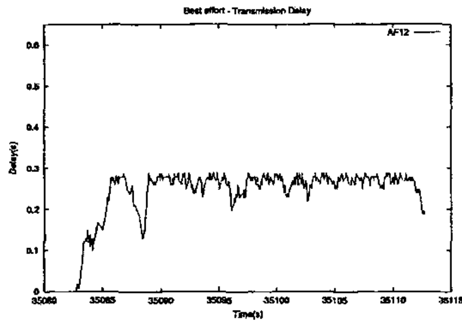
In this section we will show, comment and compare the experimental results we obtained for the four aforementioned scenarios. As a preliminary consideration, please notice that transmission time is a useful indicator when evaluating performance, since it contains information related to two fundamental aspects: the time needed to process packets and the time spent waiting inside queues.

3.1 Best Effort scenario

Let's get started by analyzing the results obtained with the Best Effort network setup. Since no differentiation is provided in this case, we expect that all the flows receive more or less the same treatment. This is what actually happens, as witnessed by Figure 2 - Best Effort scenario: transmission delay which shows the transmission delay for every flow. As you can see from the pictures, all the curves have the same aspect, which means that no special treatment is reserved to any of the flows.







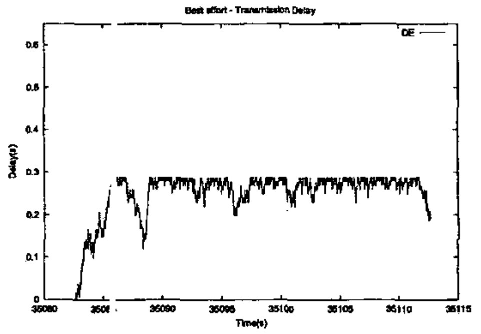


Figure 2 - Best Effort scenario: transmission delay

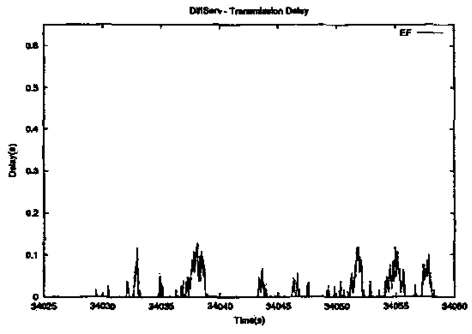
The previous considerations are also confirmed by table Table 1 - Best Effort scenario: statistical indicators, which summarizes some of the most interesting indicators associated to the traffic. Notice that packet losses are directly proportional to the different rates of the flows.

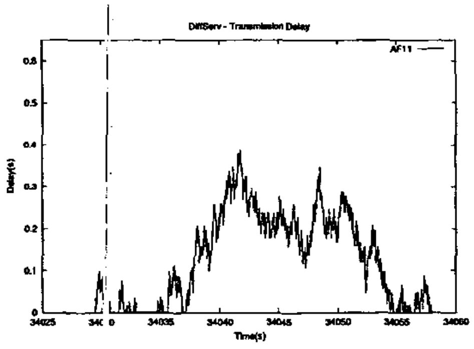
3.2 Diffserv scenario

Let's now switch to Diffserv. What we expect now is that the requirements we specified for the single flows are actually met: EF packets should be forwarded much faster than the others, AF packets should be reliably delivered, while DE packets should be treated in a Best Effort fashion. Figure 3 reports packet transmission times in the Diffserv case. Table 2 - Diffserv scenario: statistical indicators, in turn, gives some statistical indicators concerning the traffic. The first comment that can be done in this case is that packets belonging to different flows are definitely treated in a different manner: transmission delays vary from one class to the other and this was exactly what we envisaged, since Diffserv is nothing but a strategy to differentiate packets based on their specific requirements. EF packets are those that endure the smallest delay; AF packets, in turn, experience a reliable delivery (as witnessed by the zero packets lost for both AF11 and AF12 classes). Finally, the DE class is the one which suffers from the highest delay and the greatest packet losses (see Table 2 - Diffserv scenario: statistical indicators).

TX time(sec)/flow	EF	AF11	AF12	DE
mean	0.253	0.252	0.253	0.251
max	0.298	0.298	0.298	0.298
min	0.001	0.001	0.001	0.001
jitter	0.297	0.297	0.297	0.298
dropped packets	99	185	3	453

Table 1 - Best Effort scenario: statistical indicators





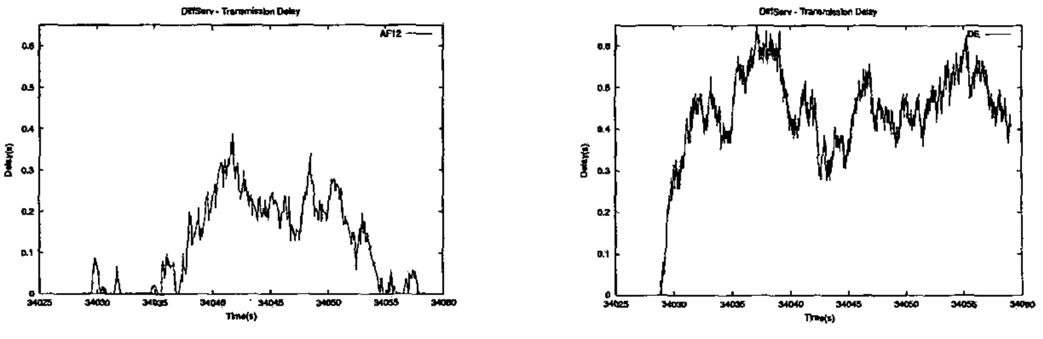


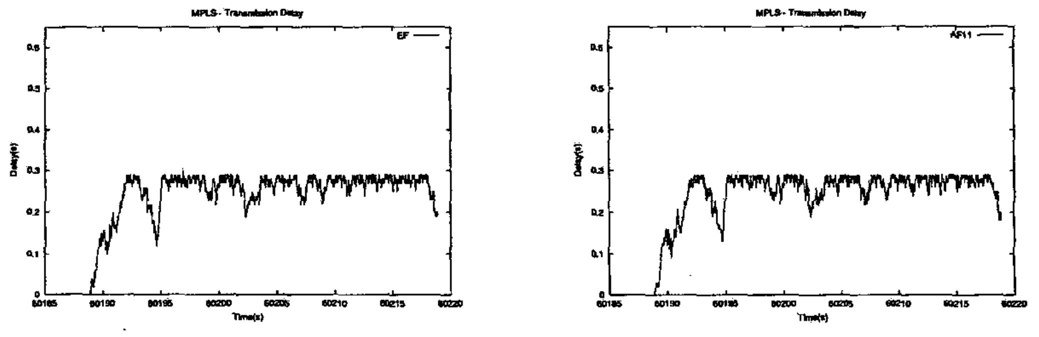
Figure 3 - Transmission delays for the Diffserv scenario

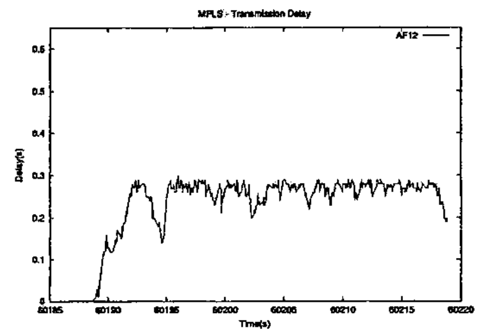
TX time(sec)/flow	EF	AF11	AF12	DE
mean	0.015	0.127	0.127	0.444
max	0.128	0.387	0.388	0.656
min	0.001	0.001	0.001	0.001
jitter	0.127	0.387	0.387	0.655
dropped packets	11	0	0	769

Table 2 - Diffserv scenario: statistical indicators

3.3 MPLS scenario

In the MPLS scenario all the flows should be treated the same way, since no differentiation mechanism has been set up and just one LSP is in place. Figure 4 - Transmission delays in the MPLS scenario shows the transmission delay for the MPLS configuration. As the reader will have noticed, such graphs are surprisingly similar to those we showed for the Best Effort case. The total delay for a packet is approximately given by the processing time inside network elements (to compute the appropriate route), plus the queuing time, plus the actual transmission time over the outgoing link. In the above computation, only the first contribution is dependent on the specific forwarding strategy adopted (in our case, standard IP forwarding versus MPLS switching). With the enforced network setup the actual bottleneck is definitely represented by the output interface's bandwidth, hence the contribution due to packet processing is negligible when compared to the others: that's why we were not able to appreciate the performance improvement associated to the use of MPLS.





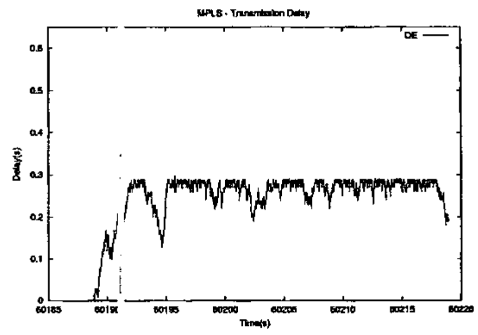


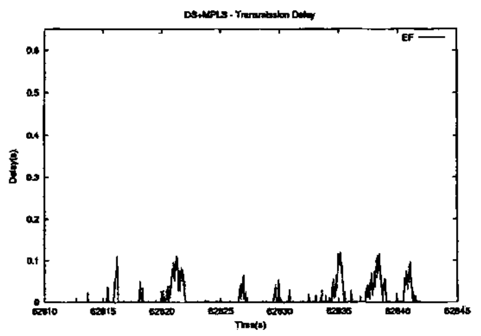
Figure 4 - Transmission delays in the MPLS scenario

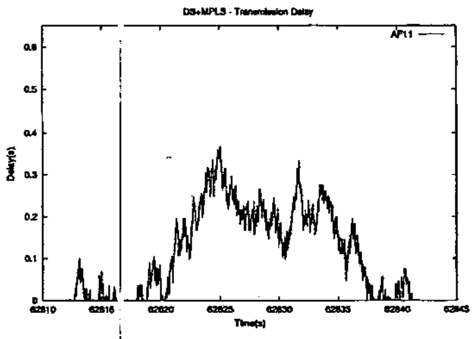
TX Time(sec)/flow	EF	AF11	AF12	DE
mean	0.249	0.249	0.250	0.247
max	0.288	0.279	0.289	0.287
min	0.001	0.001	0.001	0.001
jitter	0.287	0.278	0.288	0.286
dropped packets	141	152	2	445

Table 3 - MPLS scenario: statistical indicators

3.4 Diffserv over MPLS scenario

Let's finally take a look at the results obtained with the last configuration, where Diffserv is running over an MPLS backbone. Figure 5 and Table 4 show how the presence of a Diffserv infrastructure meets the requirements for both EF and AF flows. By comparing the graphs in Figure 3 with those in Figure 5 we notice that the traffic profiles are almost the same in the two scenarios. As we already mentioned in the previous section, with the network setup considered, the MPLS improvement does not come to the fore. First of all, in fact, also in this scenario the bottleneck is represented by the output link bandwidth, which definitely overwhelms the delay contribution due to packet processing time. Moreover, as it is well known, the more the routing tables grow, the easier it becomes to appreciate the performance gap between the longest IP routing match lookup and MPLS label switching. Finally, if you are wondering why MPLS should be used, since it adds little to network performance in the context described (i.e. one in which the routing tables dimensions are pretty small), notice that its major benefit is disclosed as soon as constraint-based routing techniques are taken into account. Stated in different terms, it is our opinion that a comparison based solely on the delay performance figures is not fair, since a thorough analysis cannot avoid considering the fact that MPLS enables traffic engineering, by evading traditional (e.g. shortest path) forwarding, in favor of fully customized routing paradigms.





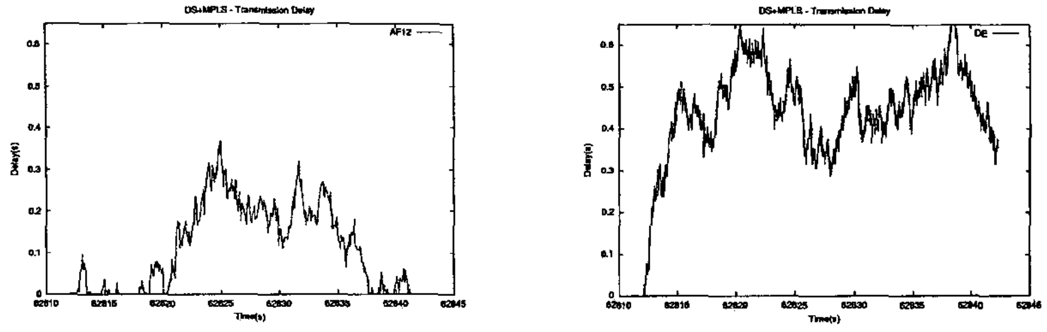


Figure 5 - Transmission delay in the Diffserv over MPLS scenario

TX time(sec)/flow	EF	AF11	AF12	DE
mean	0.013	0.120	0.119	0.444
max	0.119	0.369	0.371	0.699
min	0.001	0.001	0.001	0.001
jitter	0.118	0.369	0.370	0.699
dropped packets	10	0	0	774

Table 4 - Diffserv over MPLS scenario: statistical indicators

4. Conclusions and Future Work

In this paper we summarized the lessons we learned when applying an engineering approach to the study of state-of-the-art QoS-enabled network architectures. We presented a number of experiments aimed at investigating network performance in the presence of either Diffserv, or MPLS, or a hybrid Diffserv/MPLS infrastructure.

As to the future work, we are currently facing the issue of dynamically controlling such advanced networks, by applying to them a policy-based management paradigm. The research direction we are most interested in further exploring is related to the realization of a complex architecture for Service Assurance, capable to take the best out of the two technologies. A TRAffic-engineered Diffserv Environment (TRADE) is the final target of this specific research: the major ingredients behind it are a Diffserv-capable network on top of a traffic-engineered MPLS backbone.

Bibliography

[1] S. Blake et al., An Architecture for Differentiated Services, *IETF RFC2475*, December 1998

[2] E. Rosen, A. Viswanathan and R. Callon, Multiprotocol Label Switching Architecture, *IETF RFC3031*, January 2001

[3] Kenjiro Cho, Alternate Queuing (ALTQ) module, Available at http://www.csl.sony.co.jp/person/kjc/programs.html, January 2001

[4] W. Almesberger, Linux network traffic control — implementation overview, *White paper, EPFL ICA*, February 2001

[5] W. Almesberger, J. Hadi Salim and A. Kuznetsov, Differentiated Services on Linux, draft-almesberger-wajhak-diffserv-linux-01.txt, *1999*

[6] J. R. Leu, MPLS for Linux, available at http://sourceforge.net/projects/mpls-linux

[7] S. Avallone, M. Esposito, A. Pescapè, S. P. Romano and G. Ventre, Measuring MPLS overhead, *Proc. ICCC2002*, August 2002

[8] S. Avallone, M. D'Arienzo, M. Esposito, A. Pescapè, S. P. Romano and G. Ventre, Mtools, *Networking column on IEEE Network*, September 2002 (the tool is available at http://www.grid.unina.it/grid/mtools)

[9] S. Avallone, M. Esposito, A. Pescapè, S. P. Romano and G. Ventre, MTools: a One-way Delay and Round-trip Time Meter, *Proc. of the 6th WSEAS International Conference on Computers*, July 2002

[10] F. Le Faucher et al., Multi-Protocol Label Switching (MPLS) Support of Differentiated Services, *IETF RFC3270*, May 2002