ELSEVIER

Future Generation Computer Systems 19 (2003) 291–302

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/future

Distributed policy-based management of measurement-based traffic engineering: design and implementation

S. Van den Berghe*, P. Van Heuven, J. Coppens, F. De Turck, P. Demeester

Department of Information Technology (INTEC), Ghent University, St. Pietersnieuwstraat 41, B-9000 Ghent, Belgium

Abstract

This article discusses an architecture using monitoring feedback as an assisting factor for delivering QoS on packet-based networks. The handling of this feedback is done in an automated way, through the use of a policy-based management architecture. For this, a formal model for describing data plane and measurement objects was translated into an XML-based configuration language. On top of this, a proof-of-concept management architecture was developed and evaluated, using both a modified network simulator and enhanced Linux prototype routers.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Traffic engineering; Monitoring; Policy-based management; Real-time management

1. Introduction

1.1. Overview

In a networking environment where applications become more demanding in terms of performance requirements, the ability to offer QoS-guaranteed services is considered to be an important added value for Internet Service Providers. On the other hand, because the network performance model of aggregated packet streams is hard to determine, it is difficult to provision the network for every possible QoS requirement (as expressed in a service level agreement). Admission control algorithms, constraint-based routing algorithms, etc. all use this limited model (e.g. assumed residual bandwidth per link), and diagnostic monitoring of the actual network behaviour is needed to check that their results are not too much impaired by the uncertainties in the network performance model. To avoid this, a bottom-up approach was developed based on a generic measurement infrastructure, aimed at fulfilling the requirements as described in [9]. By linking the measurement results with an automated policy-driven (short timescale) management of tunnels in a multipath DiffServ over MPLS [6] environment, simulation results showed an adequate reaction to short-term fluctuation in network performance.

This paper describes the next step in this research: porting the architecture on a Linux-based testbed. For this, the necessary extensions to the Linux kernel were developed in our laboratory [8], to allow for the necessary functionality (DiffServ over MPLS, dividing traffic over multiple paths by mapping a set of classifiers and monitoring).

To allow a flexible configuration, a description of the capabilities of the extended router is translated into a generic model for tunnel management. In addition, a proof-of-concept policy-based management architecture is built, delivering *Common Open Policy Service* (COPS)-like [3] paradigms through a CORBA interface, and configuring the automated decision process through XML-encoded commands. For other

* Corresponding author.

0167-739X/02/$ – see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0167-739X(02)00154-1

information and operations, like topology analysis and LSP-information retrieval, the software interacts with corresponding signalling processes (an OSPF routing daemon and RSVP-TE signalling daemon, i.e. RSVP extended for end-to-end lsp configuration). After the description of the node and management software functionality and algorithms, some tests performed on the testbed are discussed. Here, users are simulated by a set of SmartBits-generated [12] streams from different edges of the network, and an analysis is made as to how the algorithm handles these traffic fluctuations. The resulting charts are analysed in terms of improved network performance, and compared with the same setup in the simulations.

In the third section, a conclusion is drawn from these practical experiences with a short-term automated management system, and future extensions are looked at. These include a generalization to policy-based tunnel management (i.e. application in overlay IP-in-IP networks or secured VPN tunnels) and even in sample service deployment without tunnelling mechanisms.

1.2. Related work

The use of monitoring to optimize network provisioning in near real-time is under study within several major research projects. The concept of a two-level traffic engineering was used in the TEQUILA approach [13]. Here, a major part of the provisioning was done based on specifications (SLSs) provided by the users describing the requested service. In order to handle fluctuations in the network (occurring due to the probabilistic nature of the SLS specification and provisioning algorithms), monitoring information is given as an input to admission control, queue management and tunnel management. The approach taken in this paper simplifies this model (by restricting itself to tunnel management), and on the other hand enlarges the role of this dynamic part of network management. The concept of a unified measurement architecture and its relationship with traffic engineering will also be investigated during a recently started IST project SCAMPI.

The concepts of policy-based management is also being reflected in the work done in the IETF's Resource Allocation Protocol (RAP) working group. Especially, the recent standardization of a feedback framework that allows policy decisions to be taken based on status reports from the network provides a powerful framework for adaptive network management. Publications on the subject of adaptive management and traffic engineering were published at recent PAM and Infocom conferences [1,4].

2. Architecture and algorithms

As mentioned in Section 1, a policy-based management approach was chosen to drive the short-term traffic engineering. This implies that each network element has a *Policy Enforcement Point* (PEP), which controls the functionality and executes operations on the network element. The decision to perform an operation is taken by a more central *Policy Decision Point* (PDP), which will use information reported by the PEPs. As opposed to the classic *Internet Engineering Task Force* (IETF) approach, the PDP is not centralized, but distributed over all the ingress nodes (to take decisions for that ingress locally), and a PEP can communicate with multiple PDPs (more specifically, multiple PDPs can use reports from a single PEP in their decision process), as depicted in Fig. 1.

Creating new configurations, and performing operations on them, is done by instantiating and manipulating objects (e.g. representing an LSP) in the PEPs. Reporting can be done in two ways: solicited and unsolicited. In the first mode, a report is immediately sent as a response to an execution performed in a PEP. This type of report can be sent from PEP to PDP or between PEPs (e.g. a PEP on the ingress of an LSP, may request statistics from the egress PEP). Unsolicited reports are sent asynchronously, and can only be going from PEP to a PDP.

In order to describe the actions, a formal model is made, which describes the different entities on which a PEP can execute operations. In the current architecture these can be divided in two major categories, namely data plane and monitoring:

1. *Data plane*. At the data plane level, packets travelling through the ingress node, encounter three packet-processing steps. As the first step, the classifier will map a packet onto an LSP, and determine the per hop behaviour (PHB) it should receive in the network. After this, the packet is encapsulated in an MPLS packet (in the LSP block) and queued/scheduled on the output according to

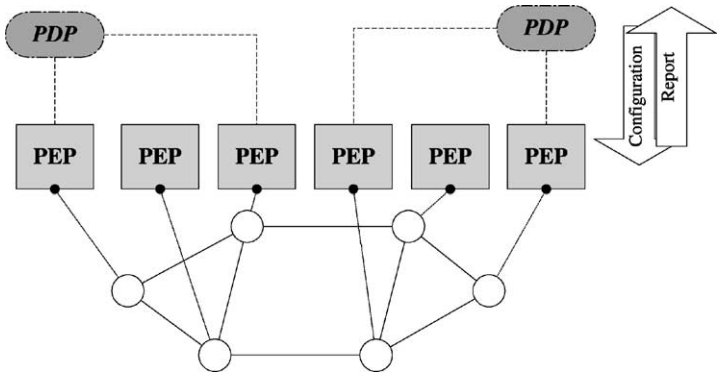


Fig. 1. PDP–PEP–network relationship.

a given PHB. The role of the PDP decision will be to manage the way classifiers are bound to LSPs. In order to do so an additional semantic class, the *LSP group*, is introduced in the model. An LSP group keeps track of both the corresponding set of the classifiers and LSPs for each PHB and for each egress. Each LSP in the LSP group is also tagged with a *weight* variable, which is used in the algorithm to keep track of the performance history of the LSP (a higher weight means less performance problems in the past). A more formal description of this model is given in Fig. 2.

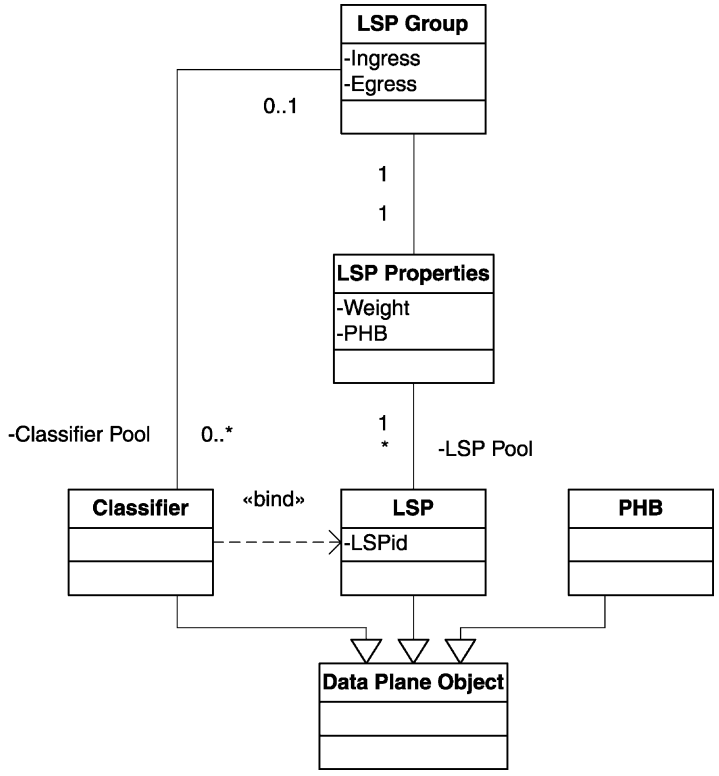


Fig. 2. Model data plane.

It is important to note that an LSP can belong to multiple LSP groups. This ability allows the support of both L-LSPs (where the traffic of an LSP is mapped to a single PHB, defined by the label) and E-LSPs (where the mapping on the PHB is not given by the label, but by another field—the so-called exp-field—in the MPLS header). In the second case, a single LSP can carry traffic of multiple service classes, so it belongs to multiple LSP groups.

2. *Monitoring*. One of the authors worked in the IETF on the specification of architectural requirements for getting useful measurements for performing traffic engineering, resulting in [9]. This document describes both the methodology requirements and the metric involved in measurement for traffic engineering. Two main types of measurements have been identified. The first one is based on keeping statistics inside queues, forwarding engines, etc. Getting the results offered by these counters is called a *passive* measurement. The class offering this is modelled as a *passive reporter*, which is connected to a data plane object for getting the actual counters (in the current implementation, "octets received" counters are offered for classifiers, LSP and PHB objects) (Fig. 3).

The second option is to perform a "ping"-like measurement: inject packets into a network as a test stream from a source to a destination, and analyse the quality with which they were transported. In this work (roughly based on the concepts in [2]), within an active monitor the sender side will be described as a *synthetic source*, and the receiver an *active reporter*. In order to configure the active measurement, the reporter is installed by the destination PEP, electing a new free destination port for the UDP test stream generated by the synthetic source that is installed at the source PEP. The triplet (source address, destination address and destination UDP port) uniquely defines the test session. Observe that in certain circumstances (e.g. to actively monitor an LSP), the injected packets need to be classified, in which case a classifier for this triplet is installed. This configuration approach replaces the control-plane signalling as defined in [11].

Both active and passive reporters will submit their results, at configured *read-out intervals*, to an analysing class, referred to as *evaluators*. Not only does this allow for a flexible analysis, but it also improves scalability of monitoring (by pushing measurement aggregation and analysis as close to the wire as possible).

The evaluator can analyse measurement results in two operational modes ("greater or equal" for upper threshold checking or "lower than" for lower thresholds) and can operate in different ways. In the current implementation, three different types are supported. The first one, a *fixed* classifier will be triggered whenever de metric is bigger (or smaller) then a configured value. The second type, a *linear* classifier will be triggered according to a probability $p(x)$ for measurement result x, upper border u and lower border l, as given for upper threshold checking in Eq. (1). Finally, a pseudo-evaluator is also available which will simply report every value (e.g. for diagnostics).

$$p(x) = \begin{cases} 0, & x \leq l, \\ \dfrac{x-l}{h-l}, & h \leq x < l, \\ 1, & x > h. \end{cases} \tag{1}$$

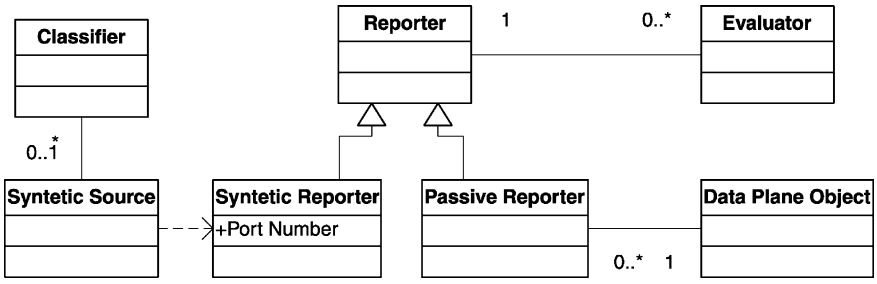


Fig. 3. Model monitoring.

2.1. Operations

Given these building blocks, actions are defined by XML-encoded strings that are transported using a CORBA interface on the PDP and PEP. At the PEP side, this interface offers the functions *install* for adding new actions (and of course also a function to remove them) and *execute* for actually performing them. Once installed, an action is assigned an object identification by the PEP, for use in later reference. The format of this identifier is <object routerId=*r* objectId=*o*>. Other actions are described by linking object identifiers together in action XML strings (observe that router identifiers are the same for both operands, since the operations are performed inside the PEP):

```
<action oper=operation>
  <object routerId=r objectId=o1>
  <object routerId=r objectId=o2>
</action>
```

The most important actions used in the current work are:

- **lspgroup** → **add**(**lsp or classifier**): adds a classifier or lsp to an lsp group (and corresponding remove operations).
- **lspgroup** → **decr**(**lsp** *l*): takes a classifier away from *l*, decreases the weight of *l*, and re-maps the classifier to the lsp with the highest weight (and different from *l*) in the LSP group.
- **lspgroup** → **incr**(**lsp** *l*): increases the weight of *l*.
- ***object*** → **show**: used on any object, to immediately report back configuration information.
- ***object*** → **showstats**: used on any object, to immediately report back statistics information.

The increase/decrease functionality is used to manage the amount of traffic on an LSP, and the two show functions are for triggering solicited feedback. An exception to the action specification format is the *create* operation for actually creating new objects of a certain type (e.g. LSP, Classifier, etc.). The specification describes object specific elements (e.g. path of the LSP). The creation of objects, and operations on them uses the same structure and both yield a new object identifier:

```
<action oper=''create'' object=Object
 Type>
  Specification
</action>
```

The PDP side offers a *report* interface operation, which in this case is only called if an evaluator is triggered to send an unsolicited report. There is also an interface operation to define rules, which declares a binding between two object identifiers at the PDP level: one to identify the rule triggering object (e.g. an evaluator) and one to identify the action to be taken (which can either be an operation on an existing object or the creation of a new one):

```
<action oper=operation>
  <trigger routerId=r objectId=o1>
  <object routerId=r objectId=o2>
</action>
```

The top-level auxiliary file: tsvdbels2.au.

2.2. Algorithms

As an example of the possible applications of the means described above, we first describe the algorithm used in later experiments. Per LSP, two rules and corresponding actions are defined. To each LSP, two evaluators are attached: a fixed one for a lower threshold and a linear one for upper threshold checking, both using input from a passive monitor watching end-to-end tunnel loss (i.e. a passive reporter at the ingress, which requests solicited usage report from the egress, and calculates difference between sent and received).

If for a given LSP the upper threshold evaluator is triggered, its weight is decreased and one classifier is swapped to another LSP, selected by the maximum weight. Secondly, if the monitored loss of an LSP is below the lower threshold, its weight is increased. In this way, the weight represents how willing an LSP is to take extra classifiers from other LSPs that have become overloaded, and do nothing if everything on the network seems all right. To summarise, the decision process is described by the algorithm given below (using the operations defined previously), where the LSPs are all part of the LSP group $G = \{L_1, \ldots, L_n\}$. Every measurement result is checked against evaluators $e_u(i)$ and $e_l(i)$, both sending a report to the ingress

PDP when triggered:

$$\omega_i = 0, \qquad \forall i \in G$$

start evaluator $e_u(i)$ for lsp l_i, $\forall i \in G$
start evaluator $e_l(i)$ for lsp l_i, $\forall i \in G$
loop
  if condition $e_u(i)$ arrived **then**
    $G \rightarrow decr(l_i)$
  else if condition $e_l(i)$ arrived **then**
    $G \rightarrow incr(l_i)$
  end if
end loop

3. The enhanced Linux router

The algorithm below was deployed in both a simulation and a Linux-based router testbed. The Linux kernel provides an advanced traffic control to implement DiffServ PHBs natively, and through an extension, MPLS support. However, the combination of both was not available. Furthermore, we needed a multi-field classification before making a forwarding decision (in order to drive the mapping of traffic on an LSP). Since the classic DiffServ-on-Linux classifiers are in the output part of the network stack, the firewalling code was reused to achieve these goals. Together with the implementation of RSVP-TE signalling, this extension was published to the open source community.

To summarise, the enhanced Linux router as shown in Fig. 4, uses the result of firewalling to put packets on the right LSP, with the correct outgoing label and exp-field (in-segment). This is then used to select the PHB throughout the network. After popping the MPLS label at the egress (out-segment of the LSP), it continues on a normal IP path. Passive monitors are looking at traffic entering and leaving the LSP (using a solicited report from the ingress to the egress).

In the current approach, a choice is made to keep most of the complexity of the architecture at the edges of the network. The core-PEP is limited to the management and reporting of its local traffic control, and is a light-weight implementations of its much more powerful cousins at the edge.

Operations in the core, like LSP establishment are performed through signalling. This implies that PEPs must communicate with the corresponding signalling blocks, which in our case are an OSPF daemon (for topology discovery) and an RSVP-TE signalling daemon (for LSP control).

4. Results

4.1. Environment

To validate this architecture, the simulator *ns-2* [10] was modified and used. The PEP–PDP functionalities were written outside the simulator (in order to be able to reuse the same codebase for the testbed) and communication between management and the simulator was done using CORBA-calls. The topology and configuration used is shown in Fig. 5. This contains five nodes (shown as 0 through 4) that actually constitute the network under test. In this network the capacity of all links, except for one, is big enough to operate loss-free with the traffic generated (all 100 Mb/s links). The link (0,4) is configure to be the bottleneck

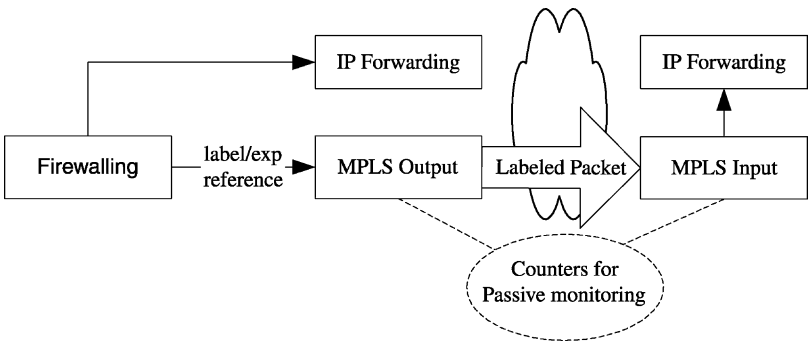


Fig. 4. Enhanced Linux router.

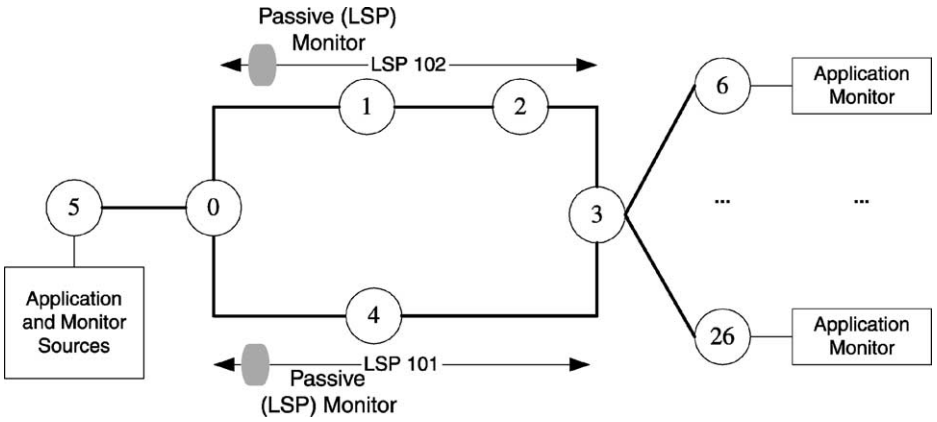


Fig. 5. Topology and setup used in this evaluation.

in the network, capable of handling 10 Mb/s. These links are part of two distinctive LSPs: LSP 101 following the shortest path (0–4–3) and LSP 102 following (0–1–2–3). Two passive monitors are installed, examining the tunnel statistics related to these lsps.

In this configuration, traffic is generated by 20 applications. Corresponding to these applications, classifiers are installed and added to an LSP group together with the two LSPs. Traffic is generated by the applications during 15 s, followed by a 3 s off-period. With each 'traffic burst' the rate per application is increased In consecutive steps of 0.10 Mb/s, generating a total traffic of between 11 and 15 Mb/s in steps of 2 Mb/s (this is repeated at every test run, so for instance traffic mapping decisions taken at 11 Mb/s are *not* reset after the burst). After each burst, the cumulative loss of all the applications for that burst is evaluated. Initially, all traffic is mapped to the shortest path LSP.

A corresponding experiment was performed on a testbed of Linux routers, equipped with the enhanced Linux kernel as described earlier. A Smartbits 2000 system was used for traffic generation, generating the same pattern as the traffic in the simulator (except for the fact that no off-time was predefined, since the Smartbits needs some time by itself to read-out results and configure the hardware for the next burst).

4.2. Results

Not running any re-mapping algorithms, this leads to the result shown in Figs. 6 and 7, using one-way loss as the evaluated metric. This chart also contains the result of the same test on a Linux testbed (the same topology as the one used for the simulator). As a reference, the graph also shows the test without mapping to an LSP on the testbed. This performance shows that introducing MPLS deteroriates the performance by only a small factor, and all three are close to the theoretical value (e.g. sending 15 Mb/s on a 10 Mb/s link, theoretically yields a loss of 33.33, 32.9% in the plain set-up—thanks to some spare room in the queues—and 33.42% by using MPLS).

For each of the LSP, two evaluators were then created with corresponding rules. This resulted in a configuration in which the weight was increased if the loss was smaller than 0.01% and was decreased (including a re-mapping of traffic) according to Eq. (1). with upper and lower bounds of, respectively, 2 and 5% one-way loss. In the experiment, the read-out period of the passive reporters providing the input was twice per second.

A first striking observation is that the simulator shows a much greater performance improvement than with the real testbed experiments. There are several reasons for this:

- When transporting measurement results to the evaluators, during the evaluation, during the decision process, etc. the simulator is actually stopped, while in the real testbed this happens asynchronously. As a result, the decision to swap traffic has a much later impact on the real testbed.
- Re-mapping traffic is disruptive on the testbed. While mapping traffic from one LSP on another is

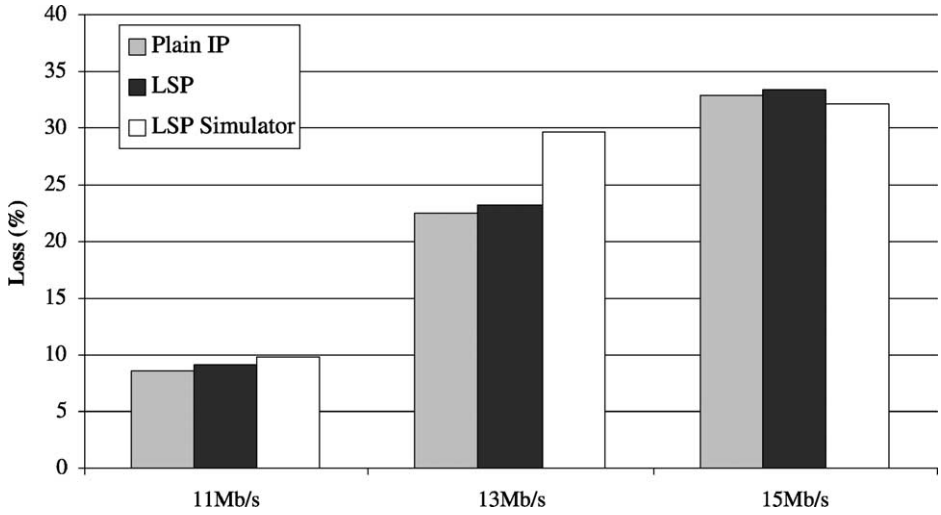


Fig. 6. Reference performance.

done through modification in the simulator, in the testbed this is done through a remove–add combination of firewall rules. In between the removal and addition, traffic is dropped before entering the LSP. Hence, the end-to-end application-level performance deteriorates, but the LSP-loss is not touched.

- Generally speaking, the testbed implementation is currently a proof-of-concept implementation to validate functionality. It makes heavy use of CORBA-calls, XML-parsing, debugging output, flat file input and output, etc. As a result, the effect of the first bullet point is even worse.

Despite these practical obstacles, the testbed results clearly show a performance improvement (although it is not as impressive as the simulation results) by relying on measurement-based, short-term traffic engineering.

A second conclusion can be drawn from the evaluation with different read-out times for each rate (Fig. 8). Here, in the simulation environment, the experiment is done with a read-out period varying between four times per second (so 0.25 s between each poll) and every 2.5 s.

The improved performance with shorter inter-measurement times is quite logic. It means more probes,

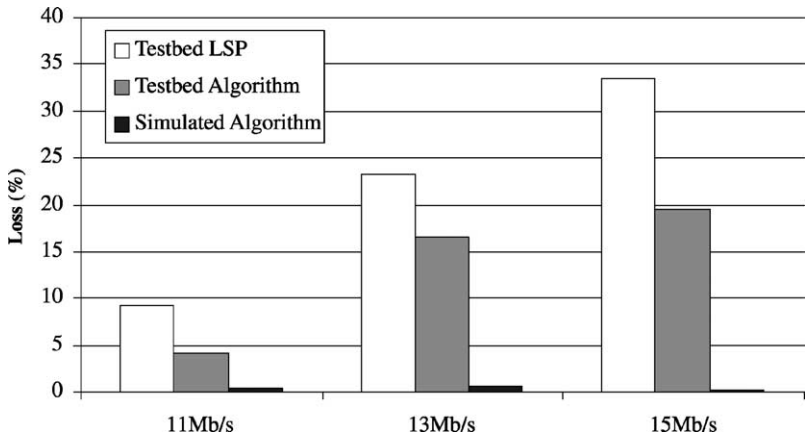


Fig. 7. Simulator and testbed performance improvement.

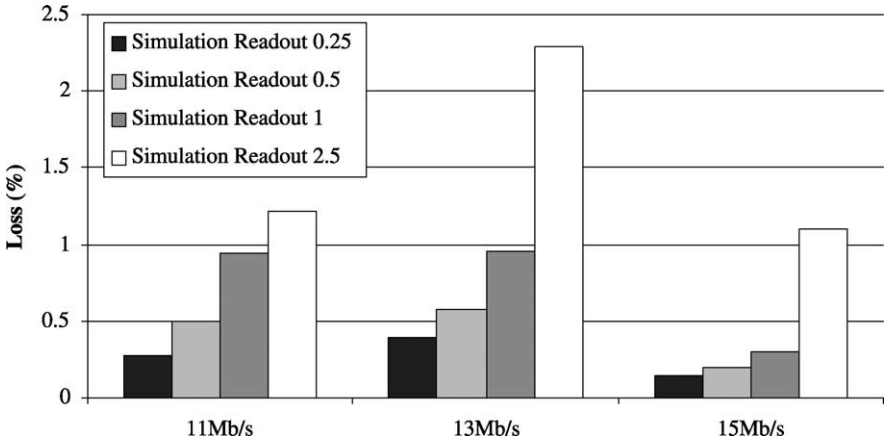


Fig. 8. Simulator and testbed performance improvement.

and as a result more opportunities to react and switch traffic to a less loaded LSP. On the other hand, it of course increases the burden of evaluating the delays.

5. Future applications

5.1. Generic model for tunnel management

While MPLS is currently being used for creating alternative paths in the network, the approach is easily implemented on top of other tunnelling mechanisms, like Generic Routing Encapsulation (GRE [5]) or secured VPN tunnels. Generally speaking, the model shown in Fig. 2 can be reused, with the LSP replaced with a more generic tunnel representation. As an example of this approach, a generic model (Fig. 9) is currently under study, where the tunnel is abstracted through the use of *virtual interfaces*, and where the tunnel encapsulation of traffic is transparent for the ingress classification. Mapping traffic to a tunnel is then transformed to setting a route via an interface. In Linux, virtual interfaces are supported for a number of mechanisms including MPLS, GRE and PPP.

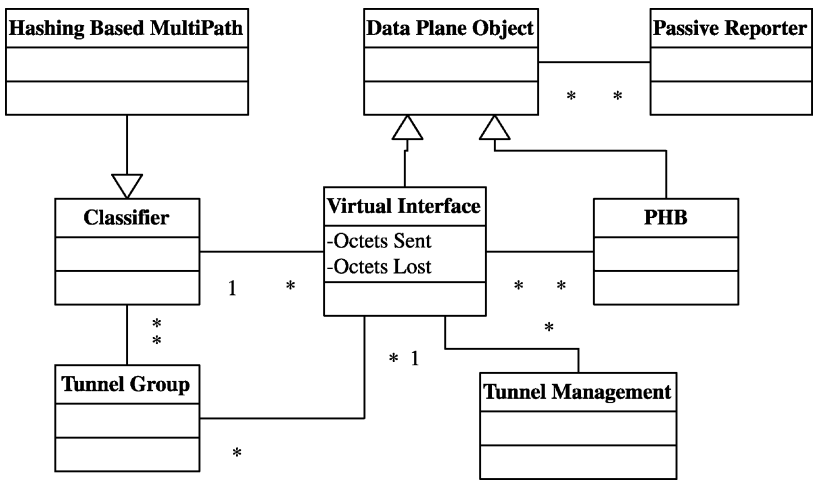


Fig. 9. Generalized model.

Furthermore, several advanced routing mechanisms (like source routing and hashing-based multipath) are available in the 'off-the-shelf' Linux kernels [7], and can be used to map traffic on the interfaces.

For monitoring, the generic nature of this model cannot be maintained so easily, due to the fact that not all tunnels are bi-directional. In the case of MPLS for instance, there is a virtual interface at the ingress, but not at the egress. In this case, the transport of statistics (e.g. the 'octets received') from egress to ingress can be done either by the tunnel management block, or through more low-level functions (e.g. by attaching it as an explicit congestion notification to the regular messages RSVP refresh messages).

5.2. Other algorithm inputs

In all the previous sections, unidirectional end-to-end loss has been used as a metric to drive the traffic mapping and the emphasis was on the management of tunnels. However, other metrics and applications can benefit from the given approach. For instance in the network shown in Fig. 10 in which a network domain is depicted that peers with two other domains. If this peering is (e.g. contractually) limited to a certain bandwidth, the domain can use the residual bandwidth at each of the peering points to direct the mapping of traffic entering the network, destined to a subnet advertised by both peers.

This mapping does not even require a tunnel mechanism to be used, but can be done by selecting different 'gateways' at the domain ingress.

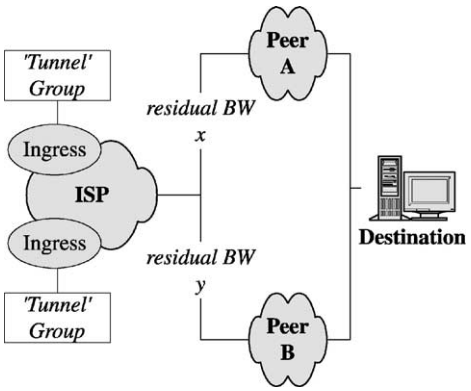


Fig. 10. Using residual bandwidth at peering points.

6. Conclusions

In this paper, a short-term traffic engineering was presented, based on the policy-based management paradigm. A formal model and corresponding configuration language was described. This allowed for a flexible proof-of-concept software architecture to be built, both on top of a simulator and on a Linux-based router testbed. In the latter case, the necessary modification done on the Linux router was described.

From the experiments it was shown that the algorithm, although very simple in nature, was able to handle some short-term fluctuations. On the other hand, by comparing the results retrieved from simulation, to those in real-life, it became clear that more has to be done than just a proof-of-concept implementation in order to achieve a more optimal performance enhancement (although already a significant improvement was made).

In a last section, it was shown that the concepts, although now aimed at MPLS tunnels, could be applied to a wider range of problems dealing with congestion avoidance.

As presented, the current evaluated environment relies only on the reactive use of monitoring feedback in a local (ingress only) node. Since no information is exchanged between these processes, the network-wide stability is not guaranteed. As a result another level of control needs to be introduced, which can perform traffic engineering on a longer timescale, and which uses—amongst other information—feedback from the architecture presented to do so.

Acknowledgements

The authors would like to thank the participants in the TEQUILA project for the interesting discussions concerning the subject of two-level traffic engineering. The authors would also like to thank Wai Sum Lai, Blaine Christian and Richard W. Tibbs for co-developing the basic monitoring ideas within the IETF. Part of this work has been supported by the Flemish Government through IWT scholarships and by the Information Society Technologies (IST) Tequila project, which is partially funded by the European Commission.

References

[1] J.L. Alberi, T. Chen, S. Khurana, A. Mcintosh, M. Pucci, R. Vaidyanathan, Using real-time measurements in support of real-time network management, in: Proceedings of the Passive and Active Monitoring Workshop, 2001.

[2] R. Cole, R. Dietz, C. Kalbfleisch, D. Romascanu, A framework for synthetic sources for performance monitoring, Internet Draft, Internet Engineering Task Force, Work in Progress, 2001.

[3] D. Durham, et al., The COPS (Common Open Policy Service) Protocol, RFC 2748, Internet Engineering Task Force, 2000.

[4] A. Elwalid, C. Jin, S. Low, I. Widjaja, MATE: MPLS adaptive traffic engineering, in: Proceedings of the Infocom, Anchorage, Alaska, April 2001.

[5] D. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina, Generic routing encapsulation (GRE), RFC 2784, Internet Engineering Task Force, March 2000.

[6] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen, MPLS support of differentiated services, Internet Draft, Internet Engineering Task Force, Work in Progress, April 2001.

[7] B. Hubert, et al., Linux advanced routing and traffic control howto. http://www.Linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html.

[8] INTEC Ghent University, DiffServ Extensions for MPLS for Linux. http://dsmpls.atlantis.rug.ac.be.

[9] W. Lai, B. Christian, R. Tibbs, S. Van den Berghe, A framework for internet traffic engineering measurement, Internet Draft, Internet Engineering Task Force, Work in Progress, 2001.

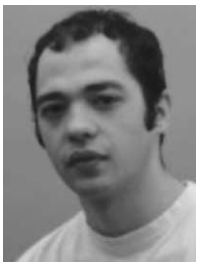
[10] Network Simulator (ns-2). The ns Manual. The VINT Project. http://www.isi.edu/nsnam/ns/doc/index.html.

[11] S. Shalunov, B. Teitelbaum, M. Zekauskas, A one-way delay measurement protocol, Internet Draft, Internet Engineering Task Force, Work in Progress, 2001.

[12] Spirent, Smartbits 2000. http://www.netcomsystems.com.

[13] P. Trimintzios, I. Andrikopoulos, G. Pavlou, P. Flegkas, D. Griffin, P. Georgatsos, D. Goderis, Y. T'Joens, L. Georgiadis, C. Jacquenet, R. Egan, A management and control architecture for providing IP differentiated services in MPLS-based networks, IEEE Commun. Mag. 39 (5) (May 2001).

S. Van den Berghe graduated in Computer Science at the University of Gent in 1999. In July 1999, he joined the Broadband Communications Networks Group and he is preparing a PhD in January 2001, he was granted an IWT scholarship. His research interests are mainly the area of Quality of Service and Traffic Engineering in IP. He is focusing on measurement-based Traffic Engineering in a DiffServ/MPLS/MultiPath environment. He is active in the IST TEQUILA project, development of DiffServ support for MPLS in the Linux community and has published, next to several papers, an Internet Draft on the requirements for measurement architectures for use in Traffic Engineered IP Networks.



P. Van Heuven graduated in Computer Science at the University of Gent in 1998. His graduation thesis ("Computer busses and caches in future processors") examined the benefits of pre-fetching in future processors by means of simulation. In August 1998, he joined the Broadband Communications Networks Group and he is preparing a PhD. In January 1999, he was granted an IWT scholarship. His research interests include mainly the area of Quality of Service, Traffic Engineering and resilience in IP and MPLS. He worked on the ACTS Ist ACI project and is currently working on the Ist TEQUILA project. He is also the maintainer of the open source "RSVP-TE daemon for DiffServ over MPLS under Linux" project.

J. Coppens joined the IBCN research group in 2001 after studying computer science at Ghent University. 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 IST Tequila project. His research interests are Quality of Service, Traffic Engineering, network monitoring and distributed computing. While doing a PhD on monitoring in Content Distribution Networks, he is currently active in the IST Scampi project.

F. De Turck received his MSc degree in Electronic Engineering from the Ghent University, Belgium in June 1997. In May 2002, he obtained the PhD degree in Electronic Engineering from the same university. From October 1997 to September 2001, Filip De Turck was research assistant with the Fund for Scientific Research Flanders, Belgium (FWO-V). At the moment, he is affiliated with the Department of Information Technology of the Ghent University as a post-doctoral researcher of the FWO-V. His research interests include scalable software architectures for telecommunication network and service management, performance evaluation and optimization of routing, admission control and traffic management in telecommunication systems.

P. Demeester received his PhD degree from the University of Gent at the Department of Information Technology (INTEC) in 1988. He became professor at the University of Gent where he is teaching telecommunication networks and where he is responsible for the IBCN research group. He is senior member of IEEE and he was or is member of several technical program committees. He was involved in about 15 European ESPRIT, RACE and ACTS projects. He was co-editor of two special issues of the IEEE Communications Magazine: "Optical Networks Research in Europe, 1997" and "Survivable Communication Networks, 1999". He was chairman of the First International Workshop on the design of Reliable Communication Networks (DRCN98) and he is member of the editorial board of the journals: "OPTICAL NETWORKS MAGAZINE" and "JOURNAL PHOTONIC NETWORK COMMUNICATIONS". He has published over 300 articles and papers in the field of optoelectronics and broadband networks. His current interests are related to broadband communication networks (IP, ATM, SDH, WDM, access) and include network planning, network and service management, telecommunications software, inter-networking, etc.