A Prototype SNMP Management Framework for DiffServ Linux Routers, its Implementation and Performance

Theodore Kotsilieris*, Panagiotis Zikos, Efstathios Vlachos, Stelios Kalogeropoulos, Angelos Michalas, George Karetsos and Vassilis Loumos
National Technical University of Athens

Division of Communications, Electronics and Information Engineering,

Department of Electrical and Computer Engineering

9 Heroon Polytechneiou Str., GR-15773 Athens, Greece

Abstract: In Linux Router management, the Traffic Control tool is still the unique method provided to network managers for monitoring and configuration. Scripts invoked through telnet are used for this purpose. Router monitoring is obtained by parsing the report created from the *tc show* command. This leads in significant load for the management system imposing barriers for real time and dynamic management. The purpose of this paper lies in presenting a prototype implementation of the DiffServ MIB based on the Traffic Control API. The fundamental objective of this implementation is to provide an SNMP management framework for DiffServ Linux Routers. It is shown that, on the basis of the SNMP functionality, efficient and real-time management can be obtained, enabling the introduction of relative differentiated services.

Key words: SNMP, DiffServ-MIB, Differentiated Services, Network Management, Linux Router

* Corresponding author: e-mail tkots@central.ntua.gr

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35620-4_43

D. Gaïti et al. (eds.), *Network Control and Engineering for QoS, Security and Mobility*

1. INTRODUCTION

Emerging multimedia applications (e.g. VoIP, video on demand) need real-time performance guarantees such as bounded delay, jitter and maximum throughput. The current Internet does not support these QoS features and it is a real challenge to embody them in the existing infrastructure. The DiffServ WG [1] is focused on providing the architecture to support various types of applications through class of service differentiation of Internet traffic. The differentiated services framework proposes the use of a well-defined set of components that afford the opportunity for a variety of services to flourish [2]. The goal of the DiffServ WG is to suggest a reference model for boundary routers (i.e. ingress and egress) defining traffic conditioning parameters, configuration and monitoring data. Towards this aim, DiffServ MIBs (Management Information Bases) are proposed in the form of Internet drafts [1].

Differentiated service mechanisms [2] allow network providers to allocate different quality levels to different users of the Internet and investigations are being done on mechanisms such as traffic meters, shapers/droppers, and packet markers to be used at the boundaries of the network. That is the point where Linux comes into play. Due to the fact that Linux is an open, flexible, and easily extensible platform with state of the art functionality for QoS support, it is a widely accepted solution for research purposes. Its flexibility and robustness allows development and validation of experimental systems. Such an area is relative and absolute service differentiation [3,4,5,6], where the basic idea is that the bandwidth assigned to each class of service can be dynamically allocated depending on the traffic load and the contracted SLAs (Service Level Agreements) between ISPs and users.

This paper presents a prototype implementation of the DiffServ MIB for a Linux router and its performance against existing techniques. The motivation was to enable Differentiated Services management with SNMP features. The most recently proposed draft-ietf-diffserv-mib-16 was chosen for implementation.

Though the traffic control technology [7,8] has been introduced on the Linux platform since 1998, there has not been proposed an efficient way to monitor and configure Linux Routers except from the tools offered from the Traffic Control Next Generation project [9] (tcng command-line tool). *tc* is a user-space application used to handle the incoming traffic at the network interface card by defining queues, classes and filters [10].

Through its command set it provides detailed information about the current configuration and performance statistics of a Linux router. The existing monitoring capabilities are limited to parsing the report generated from the tc show command. The configuration of a Linux Router is based on scripts defining the alternative queue management mechanisms, the classes and

filters that provide the desired QoS. Specifically, the tc scripts contain detailed information and define the queue characteristics of a router, the filters that classify the incoming traffic and the classes (schedulers) that enqueue the packets into the appropriate queues and forward them according to specific disciplines.

This paper is organized as follows: Section 2 presents a methodology for enabling SNMP based network management on Linux routers. In particular we first elaborate on the inefficiencies of the Tc approach and then we continue by presenting an architecture for incorporating SNMP by enhancing the Tc API from IBM. All the required enhancements are given in detail. We conclude this section by presenting the application of the DiffServ MIB implementation on monitoring and configuration of Linux routers. Section 3 presents a set of experimental results that took place on a real test-bed which prove the performance enhancement that is achieved by the presented approach. We finish the presentation of our work by providing conclusions as well as directions for further enhancements.

2. THE SNMP APPROACH TO LINUX ROUTER CONFIGURATION

There are several drawbacks in using the TC tool for a Linux router management. An important one is that configuration can be only achieved by executing predefined scripts through a telnet session. The complexity induced by this approach is increased, as the network manager apart from dealing with a not user-friendly environment, has also to face the problem of an indeterminate framework.

Furthermore, monitoring appears to be inefficient due to defective set of operations. The *tc show* command appears to be insufficient especially for real time and of specific parameters monitoring.

On account of the aforementioned shortcomings, an implementation of the DiffServ MIB is suggested in this paper in order to efficiently integrate Linux router management with the dominant SNMP [11] management protocol.

SNMP is an application level protocol that is part of the TCP/IP protocol suite. For a standalone management station, a manager process controls the access to a central MIB at the management station and provides an interface to the network manager. No ongoing connections are maintained between a manager station and remote network devices. Instead, each exchange is a separate transaction between the manager station and the managed node (i.e. an SNMP agent). Due to its bulk retrieval and transfer mechanisms for

minimizing network resources consumption, SNMP is nowadays the de facto standard in IP network management.
A Network Management system based on SNMP provides uniform access to the management information, whether real time alarms and alerts are sent or trend analysis reports are requested, acting as the cornerstone component leading to a homogeneous solution. The objective of this paper is to present an efficient DiffServ MIB implementation in order to allow Linux router management through SNMP. We will prove that SNMP enabled router management provides enhanced functionality compared with the TC tool, due to its simplicity in monitoring and configuration, and significant performance gains as we present later in section 3.
Apart from configuration and performance management other key functional areas are also supported. Fault management is obtained through the trap mechanism that allow unsolicited message reception while SNMP specific security issues partially satisfy access control mechanisms to the network elements.
The suggested SNMP agent, communicates with the Linux kernel collecting or setting all the appropriate information in order to update the objects of the DiffServ MIB tree. The DiffServ MIB is a model for the functional data path elements, allowing the network manager to erect them in any way that meets the target policy. These data path elements include Classifiers, Meters, Actions of various sorts, Queues, and Schedulers. A brief description for each element and its role is given in the following table.

Table 1. The main data path elements description

Data Path Element	Description
Classifier	Classifiers are used to differentiate among types of traffic. Classifiers can be simple or complex depending on whether they apply in core or edge interfaces respectively.
Metering Traffic	A meter integrates the arrival rate of traffic and determines whether the shaper at the far end was correctly applied. A shaper schedules traffic for transmission at specific times.
Actions applied to packets	An Action is what a differentiated services interface PHB may do to a packet in transit.
Queuing and Scheduling of Packets	Through the combined use of Queues and Schedulers, it is possible to build multi-level schedulers, such as those which treat a set of queues as having priority among them.

Through these components, the user operates over a high level interface avoiding direct interaction with the kernel. Furthermore, SNMPv3 secures

the basic principles that rule the transactions between the management station and the agent (Linux router) by protecting against:

- Modification of Information,
- Masquerade,
- Message Stream Modification, and
- Exposure.

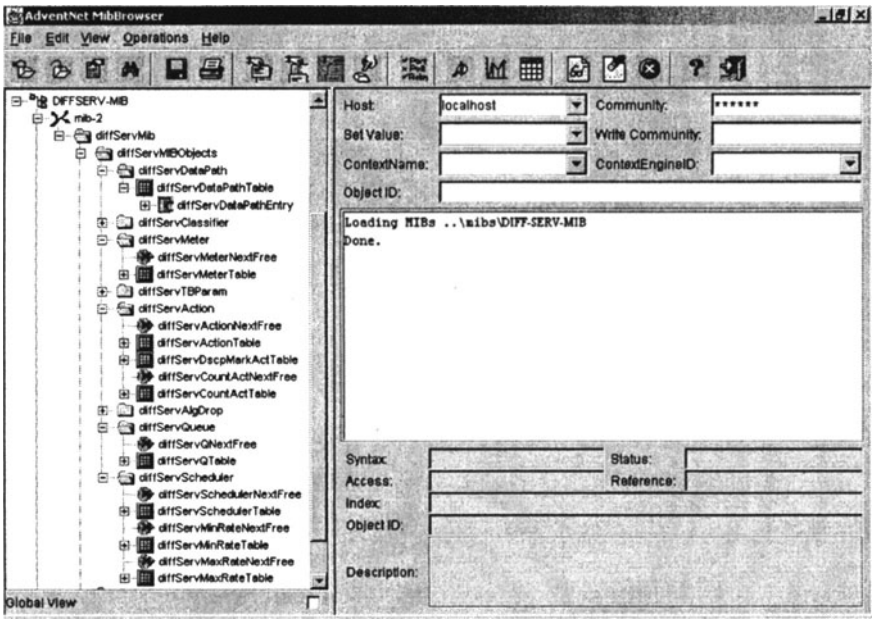

Figure 1. The DiffServ MIB and the SNMP management environment

The partially expanded tree of the DiffServ MIB is presented in figure 1 through the MIB Browser tool of AdventNet [16]. All the previously mentioned components are depicted along with a typical SNMP management environment.

Finally, the applications based on the SNMP are very flexible and take full advantage of the SNMP API as well as of innovative network management techniques [12, 13, 14].

2.1 The DiffServ MIB implementation

The implementation of the DiffServ MIB is based on the TC API [15], which is an interface to the Linux Kernel QoS mechanisms. Its main benefit is that it allows user applications (i.e. SNMP agents) to communicate with

the Linux kernel through the netlink interface. Netlink consists of a standard socket based interface for user processes and an internal kernel API for kernel modules.

We used components of the TC API, in order to fetch the management information contained in the MIB. We also enriched the TC API by creating new functions and data structures enhancing its functionality. The necessity for new functions stemmed from the fact that all the components are treated as entities. Most of the information already existed in the TC API but either it was accessible from the user through an interface or it was not grouped in appropriate structures so as to be easily handled. Although it is not complicated to access the statistics, there were no functions to obtain this. More specifically we developed the functions depicted in table 2 so as to obtain the statistics for the elements indicated in the first column:

Table 2. New functions in the TC API

Objects	Function Name	Description
Classifier	Void*classifierstats(dFilterAnchor *filterAnchor, int counter)	The classifier statistics of incoming traffic.
Meters	Void *u32meterstats(du32 *u32filter,int counter) void *tcindexmeterstats(dTC *tcindex,int counter) void *routemeterstats(dRoute *route,int counter) void *fwmeterstats(dFW *fw,int counter)	The meters statistics that a system may use to police a stream of classified traffic.
Filters	Void *u32_filterstats(du32 *u32filter,int protocol) void *tcindex_filterstats(dTC *tcindex,int protocol) void *route_filterstats(dRoute *routefilter, int protocol)	The filter statistics that a system may use to identify IP traffic
TBMeters	Void *u32TBmeterstats(du32 *u32filter) void *tcindexTBmeterstats(dTC *tcindex) void *routeTBmeterstats(dRoute *route) void *fwTBmeterstats(dFW *fw)	The specific token-bucket meters statistics that a system may use to police a stream of traffic.
Queues	Void *cbq_stats(dQdisc *qdisc) void *dsmark_stats(dQdisc *qdisc) void *TBF_stats(dQdisc *qdisc) void *fifo_stats(dQdisc *qdisc) void *red_stats(dQdisc *qdisc) void *ingress_stats(dQdisc *qdisc)	The individual queues statistics on an interface.
Classes	Void *cbqClass_Stats(dQdisc *qdisc) void *dsmarkClass_Stats(dQdisc *qdisc) void *prioClass_Stats(dQdisc *qdisc)	The individual class statistics on an interface.
AlgDrop	Void *AlgDrop_Stats(dQdisc *qdisc)	The AlgDrop_Stats provides statistics for dropped packets

Objects	Function Name	Description
Count Action	Void *countactstats(NIC *nic)	The *countactstats(NIC *nic) provides counters statistics for all the traffic passing through an action element

Furthermore modifications were done in the core functions of the TC API illustrated in table 3. The necessity for these enhancements arises from the fact that filters have no statistics. Instead of them policers' statistics can be used. They are attached to filters and maintain their own statistics. Usually each filter can have a policer, and this way the filter statistics are these collected from the policer.

Table 3. Modified functions of the TC API

Function Name	Description
int TreeBuilderTFilter(Qsession *qSession,struct nlmsghdr *reply, NIC *nic)	Enhanced functionality in order to obtain filter statistics.
static void TreeBuilderAddTCEntry(dFilterAnchor *filterAnchor, dTC *node) static void TreeBuilderAddTCHashTable(dFilterAnchor *filterAnchor, dTCHT *node)	Both are used in order to add a tcindex filter statistics indicator to the previously created hash table.

On top of the TC API, we used the AdventNet Agent Toolkit [16] that provided a C code framework for implementing the MIB. The C language was chosen for the implementation because of the necessity to have a common interface with the linux kernel and time essential processing. The stub code generated from the Agent Toolkit associates the data accessed through the TC API to the appropriate MIB objects. The outcome of this work was a complete DiffServ SNMP Agent. The high level architecture of our implementation is presented in figure 2.

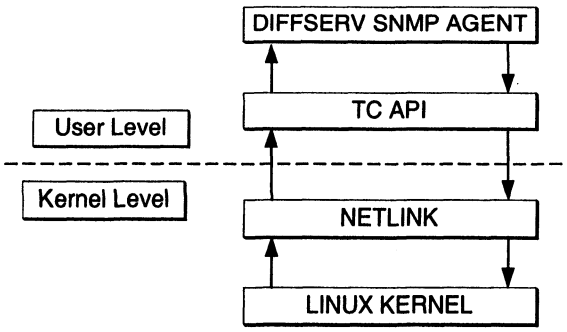

Figure 2. The DiffServ MIB prototype implementation high-level architecture

2.2 Monitoring and Configuration

A leaf node in the MIB tree represents each managed object of the router. Every node is assigned a label consisting of an integer. The identifier of an object is a series of integers that mark the full path from the root of the MIB tree to the specified object. The DiffServ MIB describes every component of the router (queues, filters, classes) and the associated information is organised in MIB Tables; each Table corresponds to a specific component and contains entries for every queue, filter or class.

During the initialization phase the SNMP agent checks the DiffServ configuration of the router through the initAgent() function, each queuing discipline module adds itself to a global list named qdisc_base and fills the associated MIB entries. Once started, the agent runs as a separate Linux process and listens for incoming SNMP requests to a specified port.

The manager is able to make a query through the GET operation of the SNMP, providing as an argument only the OID of the desired component and the ip address of the router. Each time the agent is triggered, either locally or remotely, it communicates with the Linux kernel via the TC API. The TC API functions direct the incoming requests, either for monitoring or for configuration, from the user to the kernel level in order to perform the desired management operations and produces a response with the requested information.

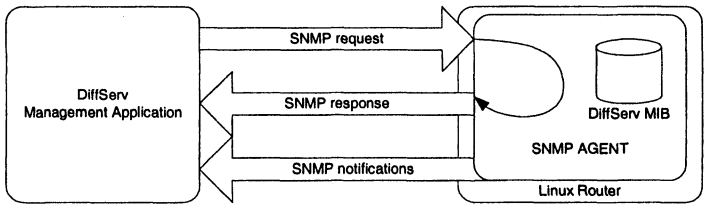

Figure 3. Monitoring and Configuration through SNMP.

Especially for the configuration case, it can be achieved via the SET operation of SNMP. Through the network management application the user selects the desired DiffServ component to be created or modified. A new Queuing tree, which describes the updated structure of the DiffServ router, is constructed each time a configuration is accomplished. Figure 3 illustrates how monitoring and configuration is obtained through SNMP.

3. EXPERIMENTAL RESULTS

In order to verify the optimization introduced by our system we performed a simple experiment, of managing a DiffServ Linux router using the SNMP agent. The same experiment was also conducted using the previously described methodology, based on parsing the TC tool report. The performed tests were focused on the evaluation of execution time needed for both methodologies to extract the diffServCountActTable entries on a varying number of nodes. The test-bed is illustrated in figure 4.

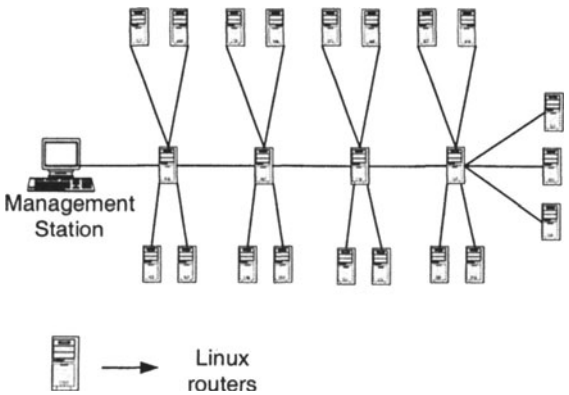

Figure 4. The test-bed topology

In figure 5 the results of the tests demonstrate that the SNMP agent implementing the DiffServ MIB performs better than parsing the report file generated from the tc show command.

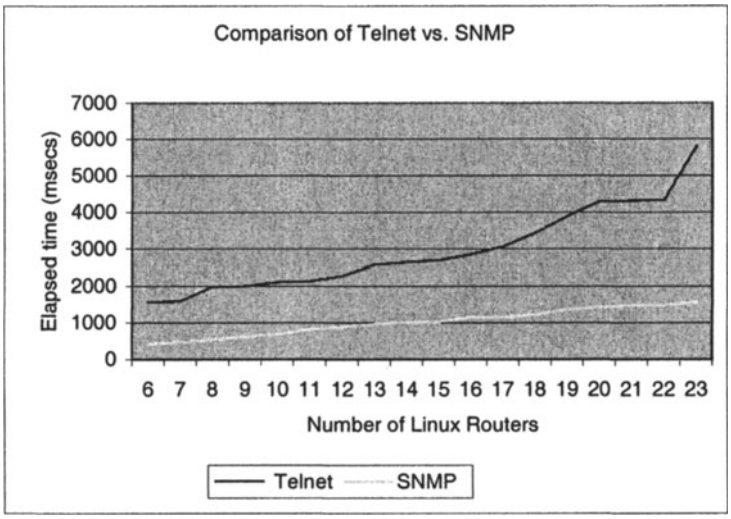

Figure 5. Comparison of Management through Telnet versus SNMP

Table 4. The experimental results

Number of Nodes	Telnet	SNMP
6	1570.9	413
7	1590.2	475.7
8	1981	541.4
9	1999	624.4
10	2114	675.6
11	2134	819.3
12	2279.8	885.3
13	2583.6	971.2
14	2648	1020.2
15	2695	1027.6
16	2852	1150.1
17	3059	1154.1
18	3425	1225.6
19	3882	1363.4
20	4282	1427.1
21	4304	1478

Number of Nodes	Telnet	SNMP
22	4322	1488.7
23	5793	1557

From the results of our experiments we conclude that response time decreases by 20% for a simple GET operation on 6 nodes and 70% for 25 nodes. The execution time for management through Telnet increases rapidly as the number of managed routers increases. On the contrary, the execution time for management through SNMP increases in significantly slower rate due to an overhead of 100 msecs accrued each time a new node is introduced in the test-bed.
The operation performed in this experiment is simple but representative enough to confirm the performance improvement. This is quite a promising outcome, since we haven't considered possible overheads introduced in the case of the Tc approach for the preparation of scripts capable to return every requested variable by parsing the report file. We will focus our next steps on studying in detail these trade-offs, by performing further experiments and measurements.

4. CONCLUSIONS

We have presented the high level architecture and the implementation of a system that proved to deliver better performance in monitoring and configuration of DiffServ Linux routers. Today's evolution in Internet QoS requires flexible network management solutions. Towards this end, the implementation of the DiffServ MIB presents distinct advantages over previously proposed techniques that are currently in operation.
In this work, we also presented results from a real experiment concerning the time needed to accomplish a task. The execution time of a task is critical in network management and especially in areas that real time interaction is necessary. The experimental results revealed that significant gains could be obtained on the performance of a Linux router management system through our implementation.
Our next objective within this framework is to design and implement a distributed management system for Linux routers. Decentralisation of the SNMP protocol, through the usage of CORBA or Mobile Agents, could lead to significant enhancements in terms of both functionality and performance.
Furthermore, another promising research area is the design and implementation of appropriate mechanisms to control resources managed by the kernel [17] (i.e. CPU usage, disk access, memory usage). This

functionality will enhance QoS applications and control algorithms by altering the existing approach which is limited to packet filtering, marking and scheduling.

REFERENCES

[1] http://www.ietf.org/html.charters/diffserv-charter.html
[2] "Performance evaluation of a Linux DiffServ implementation", G. Stattenberger, T. Braun, M. Scheidegger, M. Brunner, H.J. Stuttgen, In Computer Communications Journal, vol.25, issue13, pp 1195-1213
[3] "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling", C.Dovrolis, D.Stiliadis and P. Ramanathan, Accepted for publication at the IEEE/ACM Transactions in Networking.
[4] "Dynamic Class Selection: From Relative Differentiation to Absolute QoS", C.Dovrolis, P.Ramanathan, Proceedings of the 2001 IEEE International Conference on Network Protocols.
[5] "A Case for Relative Differentiated Services and the Proportional Differentiation Model", C. Dovrolis, P. Ramanathan. In IEEE Network, 13(5) p.p. 26-34, September 1999
[6] Relative Jitter packet scheduling for Differentiated Services, T.N.Quynh, H.Karl, A.Wolisz, K.Rebensburg, 9th IFIP Conference on Performance Modelling and evaluation of ATM & IP Networks 2001.
[7] Almesberger, Werner. Linux Traffic Control – Implementation Overview
[8] W. Almesberger, J. Salim, A. Kuznetsov, Differentiated services on Linux, Globecom '99, Rio de Janeiro, December 1999, pp. 831-836
[9] http://tcng.sourceforge.net/
[10] http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html
[11] http://www.ietf.org/html.charters/snmpv3-charter.html
[12] "Integration of Mobile Agents with SNMP: How and Why", Pagurek, B., Wang, Y., White, T., Proceedings of NOMS 2000
[13] "Evaluating Tradeoffs of Mobile Agents in Network Management", Pagurek, B., Wang, Y., White, T., Networking and Information Systems Journal, Hermes Science Publications, vol. 2, no. 2, pp. 237-252, 1999
[14] "Integration of SNMP into a CORBA- and Web-based Management Environment", Gerd Aschemann, Thomas Mohr, Mechthild Ruppert, KiVS '99, March 2-5, 1999, Darmstadt.
[15]http://oss.software.ibm.com/developerworks/projects/tcapi/
[16] www.adventNet.com
[17] "Enhancing the Performance of Mobile Agent based Network Management Applications", A. Michalas, T. Kotsilieris, S. Kalogeropoulos, G. Karetsos , Moshe Sidi and V. Loumos, 6th IEEE Symposium on Computers and Communications, IEEE ISCC'01, 3-5 July 2001 Hammamet, Tunisia.

POSTERS 2