Traffic Control in a Linux, Multiple Service Edge Device

Joana Urbano, António Alves, António Raposo, Edmundo Monteiro

Small and medium size networks with access to the Internet are each day more common in SOHO (Small Office/Home Office) environments, comprising domestic and small to medium size organizations. Inherent to the spread of these networks is the search for small access devices that allow the connection to the Internet in a very user friendly way but with a growing number of offered services, comprising basic IP functionalities (e.g., DNS, DHCP, SMTP and NAT), security and traffic control services. Traditionally, these services are available in dedicated, proprietary, and often-patented devices, with complex integration and high cost. More recently, new products appeared in the market that integrate several services in a single device, as is the case of 6WINDGATE from 6WIND [1], Access Point from Lucent Technologies [2] and ERX Edge Router from Juniper Networks [3]. However, these products still present some complexity at management and still have medium to high cost. To overcome an open space in the market, Critical Software and the Laboratory of Communications and Telematics (LCT) from University of Coimbra are developing a Linux, multi-service edge device, aiming the offering of basic and advanced IP services at low cost and with trivial management requirements. This paper presents one of the key components of this Edge Device, the traffic control system. Particularly, it presents a solution for the DiffServ-based modeling of upstream and downstream traffic in a single Linux router that is also an authentication, NAT (network address translation), firewall, service provider and application gateway machine. Performance tests of the control traffic system and correspondent analyses of the results conclude the paper.

Keywords: Resource management, congestion control, QoS, edge device, Linux

I. INTRODUCTION

The Edge Device project aims the developing of an equipment that will render trivial configuration and operation of Internet access for small and medium sized organizations, based on four key principles:

1. Integration of Internet Services
2. Easy of use
3. Security and availability
4. Low cost

Main author's affiliation: stmaria@dei.uc.pt, LCT, Universidade de Coimbra, Pólo II, Pinhal de Marrocos, 3030-290, Coimbra, Portugal, http://lct.dei.uc.pt.

By supporting common physical interfaces (Serial, Ethernet, WiFi) with intuitive set up wizards, the connection of the entire organization to the Internet would be possible without assistance from a network engineer.

Edge Device management application is based on a layered architecture and was designed for easy integration of new services and applications. Its management offers the functionalities common to all services, like authentication, authorization in an LDAP repository, backup and restoring of configuration, services upgrade, etc. It has a well-defined interface with all the services, and mediates the relationship between the user interface layer and the services layer (Figure 1).

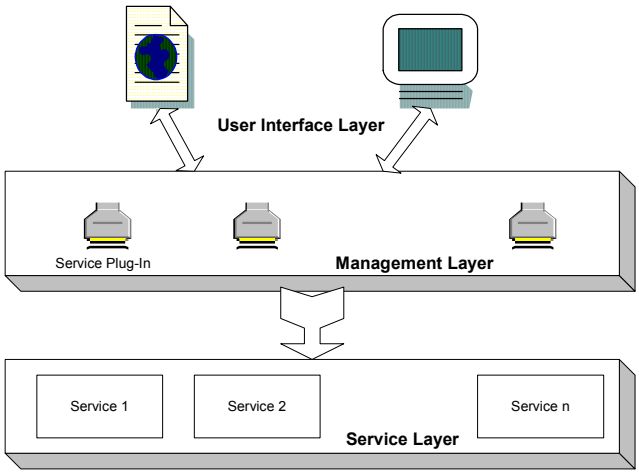

Figure 1 Architecture of the Edge Device

A. *The Edge Device and the Traffic Control Module*

This section mentions the key features of Edge Device that directly interact with the traffic control system and permit to understand how it works. All the system is being developed over Linux 2.4.18, for wired and wireless environments.

As already mentioned, Edge Device is a firewall, NAT (network address translator) router operating at the edge of a LAN (local area network). All users of the LAN, including wireless users, must authenticate before they connect. Edge Device is also a service server and an application gateway, integrating a Squid server for HTTP

traffic, operating in transparent mode. In order to optimize the generally scarce resources at the access border, upstream and downstream traffic is classify by user into different DiffServ [4] traffic classes and prioritized according to its importance to the organization. All system resources are managed according to a policy based management.

The paper is organized as follows. The next section presents the traffic control system of the Edge Device, pointing some problems that arise when integrating multiple IP services onto a single Linux machine, and presenting a possible solution for up and downstream shaping. Section III evaluates the implemented traffic control system, and describes two sets of tests related to the shaping of upstream and downstream interface bandwidth. Section IV concludes the article.

II. Traffic Control System

In an edge router accessing the Internet, it is important to control the bandwidth of traffic that leaves the LAN and enters the ISP link, i.e., the upstream traffic. In the example of a 128 Kbps upstream ADSL link connecting a medium size enterprise, aggregate uploads that exceed this rate will fill the ADSL modem – generally FIFO – queues, either leading to packet drops or, in the presence of big FIFO queues, to values of latency far too high for interactive traffic. A better approach is to conditioning the upstream traffic in the edge device up to somewhat less than the uplink bandwidth, and to prioritize this traffic according to its importance to the organization. Looking at the downstream side, the problem is somewhat different, in that edge routers do not have effective control in what is send do the LAN. Downloads at a medium size enterprise, or even at home, easily exceed the downstream link capacity, leading to the queuing, delaying and possible dropping of packets at the ISP edge router, regardless the relative priority of the traffic flows. One possible approach for overcoming this situation is to take profit of TCP rate control mechanisms by selectively dropping packets or even mangling the advertised receive window, at the edge device side, in the hope that sender side slows down the transmission. Both solutions present evident drawbacks[1] and need to be thoroughly tested in the scope of this project. Next is a description of the solution found to the control of upstream and downstream traffic in the Edge Device.

A. Upstream traffic

Linux traffic control can be done at QoS Ingress, QoS Egress or at the virtual device IMQ (Intermediate Queueing Device [6]), called on PREROUTING and POSTROUTING Netfilter hooks (see Figure 2).

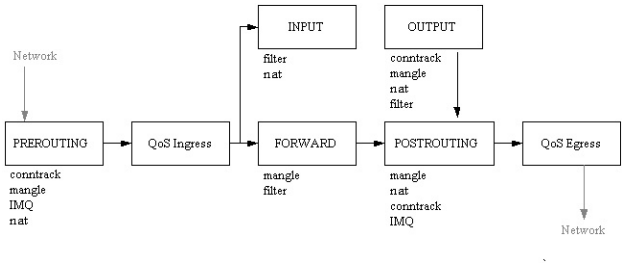

Figure 2 Kernel packet travel diagram (based on [7])

QoS Ingress uses a rather limited classless queuing discipline (*ingress qdisc*) that only filters traffic, using a token bucket algorithm, and would not be used in the Edge Device. On the other hand, IMQ modeling at PREROUTING or POSTROUTING is done in the exact same way as at QoS Egress, exploring the full capabilities of Linux queuing disciplines, classes and filters. As the IMQ module at POSTROUTING immediately precedes QoS Egress, it cannot perform any better than QoS Egress, having the drawback of introducing one more queue to the traffic control system.[2]

The Edge Device traffic control system manages upstream traffic at QoS Egress, using some artifacts to overcome the effect of NAT and broken TCP connections issues. In fact, the transparent redirection of HTTP traffic to Squid at PREROUTING and the realization of NAT at POSTROUTING pose some difficulties for the differentiating of traffic per source IP at the QoS Egress, since the addresses seen by the traffic control would be the Edge Device address and the NAT address, respectively. To overcome Squid drawback, HTTP traffic is DSCP-marked at Squid using the *tcp_outgoing_dscp* functionality as follows:

```
acl tc_service src IP1 IP2 … IPn
http_access allow tc_service
tcp_outgoing_dscp service_dscp tc_service
```

On the other hand, NATed traffic that does not pass through Squid is previously marked in the PREROUTING chain (using iptables) in a per user basis, as follows:

```
iptables -t mangle -A PREROUTING -s
ips_from_tc_service -j MARK --set-mark
tc_service_mark
```

[1] By selectively dropping TCP packets one are indeed dropping "perfectly condition" packets that have already consumed downstream link bandwidth. One the other end, some authors claim that mangling the advertised receive window of TCP ACK packets works well on an individual flow basis, but has a negative impact on the system performance in an congested, mixed traffic link (see, e.g., [5]).

[2] For further information on Linux QoS, IMQ and Netfilter, see [8] and [9].

This way, it is easily seen that HTTP traffic that goes through Squid is differentiated and shaped at QoS Egress based on the DSCP field, and the remaining NATed traffic is differentiated and shaped by iptables mark.

B. *Downstream traffic*

Downstream traffic differentiation can occur at QoS Egress, where NATed and HTTP traffic addresses are already resolved. However, this solution presents the drawback of shaping HTTP traffic that probably is already at Squid cache. Taking this into account, we choose to implement downstream traffic control at the IMQ device using, once again, an artifact, which consists in the registration of IMQ after NAT, at PREROUTING. For this, we used the *imqnat patch* by Patrick McHardy.[3] By changing IMQ and NAT positions at PREROUTING, we can guarantee that traffic arriving at IMQ is already de-NATed and can, this way, be differentiated by source IP.

This does not solve, however, the problem of HTTP traffic that goes through Squid. Looking again at the kernel packet travel diagram (Figure 2), it is easily understandable that HTTP traffic real addresses are known after OUT chain, way after the traffic shaping at the IMQ virtual device. We came across four possible solutions, none of them satisfying us completely:

1. HTTP traffic is not forwarded to IMQ, being shaped at Squid *delay pools*[4] instead;
2. HTTP traffic is differentiated at IMQ and put in a specific traffic class, regardless of its real address;
3. HTTP traffic is not forwarded to IMQ and it is shaped at QoS Egress. As was mentioned before, this solution has the drawback of shaping all HTTP traffic, including the traffic already in cache.
4. HTTP upstream traffic of certain priority users is not forwarded to Squid and, therefore, can be shaped by real IP address at IMQ, on the downstream direction. The cache benefits associated with the HTTP traffic of those priority users are, however, loosed.

As of the writing of this article, it was announced a patch for Squid that uses the *Stream Identifier* option of IP header to mark HIT packets [12]. Packets could then be classified by Linux traffic control using the u32 filter:

```
tc filter (…) match u32 0x8804 ABCD
0xffffffff at 20
```

This and the previously solutions need to be thoroughly tested in order to choose the best downstream scenario for the Edge Device.

[3] This patch can be found at [10].

[4] Delay pools are a mechanism provided by Squid to limit the bandwidth of certain requests, based on a criteria list (see [11]).

III. EVALUATION OF THE TRAFFIC CONTROL SYSTEM

Figure 3 presents the testbed used for testing the traffic control system of Edge Device. WAN Emulator is a Linux QoS-capable router that shapes both interfaces to the bandwidth being emulated. When testing upstream traffic control, WAN Emulator works as the access device (e.g., ADSL modem), shaping both interfaces to the ADSL rate. When testing downstream traffic, WAN Emulator works as the ISP router, shaping the downstream interface according to the downstream ADSL link bandwidth and letting upstream interface unchanged.

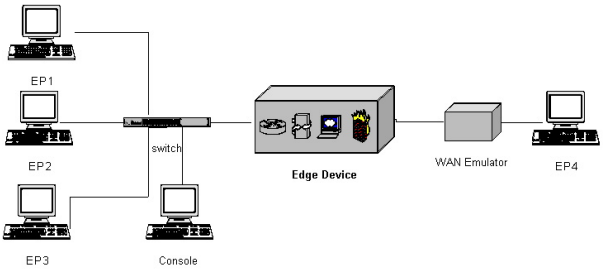

Figure 3 Testbed for Edge Device testing

In all tests, EP1, EP2, EP3 and EP4 are Chariot Endpoints and Console is the Chariot Console [13]. The Edge Device under test was compiled with full netfilter and QoS support, and there were installed iptables-1.2.7a package and IMQ, h323-conntrack-nat[5] and imqnat patches. It was also installed version v.2 of Squid.

Two sets of tests were done with 64 bytes packets. The first set tested the capability of a specific Linux DiffServ configuration to differentiated upstream traffic using five different codepoints, and the performance impact associated with the activation of upstream control traffic at the Edge Device. It also tested how low priority traffic classes access bandwidth in congestion situations and how exceeding bandwidth from one class is distributed among the others. In the entire set of these testes, we used the dsmark qdisc for DSCP retrieval and marking, cbq as the primal queueing discipline, tbf for high priority traffic classes and sfq for low priority traffic classes. Gred qdisc was also used to put medium priority traffic into different drop precedences of a DiffServ AF class [14]. An explanation of the queue disciplines used can be found at [8]. The second set of tests aimed to study the effect of shaping TCP traffic in the downstream direction and to tune the downstream shaping.

A. *Upstream Tests*

The set of upstream tests simulated an asymmetric access to the Internet, with 256kbit/s available at upstream and 1500Mbit/s available for downstream traffic. Three test flows were generated, each with 0,512 Mbit/s.

[5] H323-conntrack-nat patch allows H.323 *streaming* applications (e.g., NetMeeting) to coexist with POSTROUTING SNAT (see [9] for source code).

Table 1 shows the throughput results obtained with those test flows in the absence of traffic control and Figure 4 plots these results.

			Throughput		
Src.	Dst.	Flow Rate	Av	Min	Max
		(Mbit/s)	(Mbit/s)	(Mbit/s)	(Mbit/s)
EP1	EP4	0,512	0,089	0,066	0,125
EP2	EP4	0,512	0,084	0,058	0,154
EP3	EP4	0,512	0,079	0,058	0,135
			0,252		

Table 1 Numerical results for upstream tests without TC

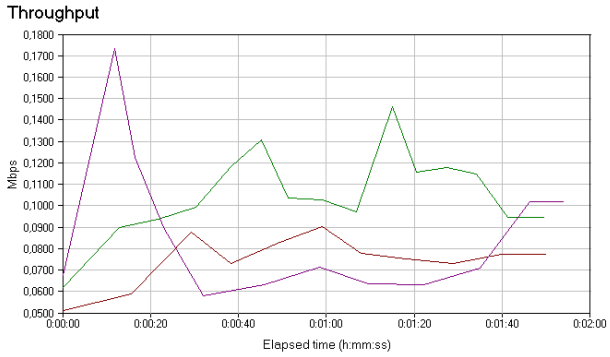

Figure 4 Graphical results obtained without traffic control. Red plot stands for EP1→EP4, green and pink for EP2→EP4 and EP3→EP4, respectively.

We repeated these tests with the traffic control service activated at the Edge Device. We have assigned drop precedence 1 and 2 of a DiffServ AF class to EP1→EP4 and EP2→EP4 test flows, respectively. Those two flows were assigned 80% of the available upstream bandwidth. Flow EP3→EP4 was assigned to the best-effort service class. Numerical and graphical results of the tests are shown at Table 2 and Figure 5, respectively.

			Throughput		
Src.	Dst.	Flow Rate	Av	Min	Max
		(Mbit/s)	(Mbit/s)	(Mbit/s)	(Mbit/s)
EP1	EP4	0,512	0,102	0,082	0,132
EP2	EP4	0,512	0,093	0,072	0,119
EP3	EP4	0,512	0,049	0,045	0,050
			0,242		

Table 2 Analytic results for upstream tests with TC

The results show that, even though average throughput performance is slightly worse in the presence of traffic control (about 96-97% of the average throughput without traffic control), there is an effective protection of the most critical flows when overload ranges from moderate to heavy. It was also seen that limiting upstream bandwidth at the Edge Device to a value sensitively equal to the ADSL upstream value frees WAN Emulator from drops, as expected. Surprising enough, we did not find that shaping at a little less than the upstream bandwidth would significantly improve the overall performance.

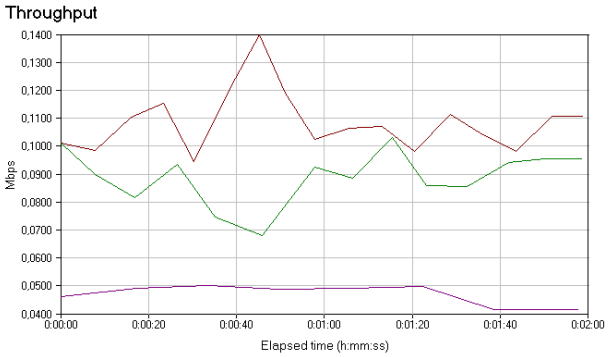

Figure 5 Results obtained with traffic control

B. Downstream Tests

Preliminary results obtained with different load scenarios show that the impact of activating downstream traffic control on global throughput is meaningless, and that best results are obtained shaping at 95-99% of the downstream link capacity. Figure 6 illustrates the results of one of those tests, obtained with two 8,448 Mbps downstream test flows for an ADSL downstream link of 1,500 Mbps, with no traffic control at the Edge Device. Figure 7 depicts the results obtained with two different priority classes defined at the Edge Device. Apart the evident protection of priority traffic, it can be seen a slight smoothness of traffic in the presence of traffic control.

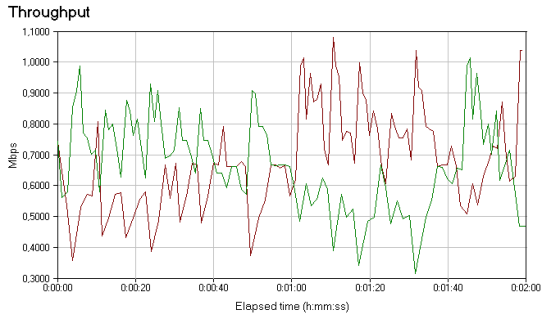

Figure 6 Downstream results without traffic control

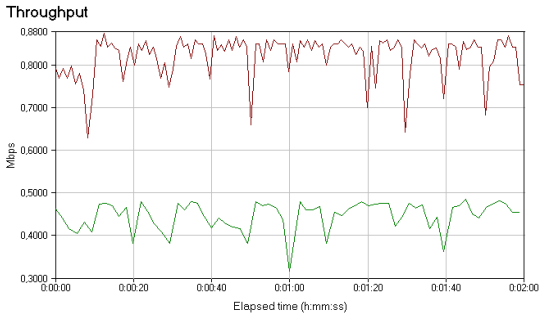

Figure 7 Downstream results with traffic control. Green plot corresponds to the highest priority test flow.

IV. CONCLUSION

The configuration of a Linux traffic control service in a multiple service router is not straightforward. In fact, the inclusion in a single machine of firewall, NAT, authentication, wireless links and content cache proxy poses some difficulties when it comes to differentiate traffic based on IP source. This article focused on some somehow less discussed questions related to the implementation of traffic control in such a multi service machine, and described a possible solution for the shaping of upstream and downstream traffic.

Several tests were done and are still being done in the context of the Edge Device project. The last section of this article presented some tests related to the shaping of upstream and downstream interface bandwidth. Results have shown that limiting upstream to less than 99% of the available bandwidth does not improve upstream performance. Results have also shown that assigning a tbf qdisc to high priority traffic, a gred qdisc to medium priority traffic and a sfq qdisc to low priority traffic is very effective in terms of performance of the overall traffic control system for cable-modem, dsl and alike connections. Finally, tests done for the downstream scenario showed that best results are obtained shaping at 95-99% of the downstream link capacity. Also, preliminary results seem to show that the limitation of downstream rate is a rudimentary rate control mechanism that seems to be very effective in the traffic control module implemented.

V. ACKNOWLEDGMENT

This work was partially funded by Agência de Inovação, in the aim of the EDGEDEVICE project.

VI. REFERENCES

[1] 6WIND Homepage, http://www.6wind.com
[2] Lucent Technologies Homepage, http://www.lucent.com
[3] Juniper Networks Homepage, http://www.lucent.com.
[4] X S. Blake et al., “An Architecture for Differentiated Services Framework”, RFC 2475, http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2475.html
[5] Check Point Homepage, http://www.checkpoint.com
[6] The Intermediate Queueing Device, http://luxik.cdi.cz/~patrick/imq/
[7] http://www.docum.org
[8] Linux Advanced Routing & Traffic Control Homepage, http://www.lartc.org/
[9] Netfilter Homepage, http://www.netfilter.org
[10] The imqnat patch, http://mailman.ds9a.nl/pipermail/lartc/2002q3/004725.html
[11] Squid delay pools, http://www.squid-cache.org/Doc/FAQ/FAQ-19.html
[12] M. Stavrev, Zero Penalty Hit patch for SQUID (ZPH), http://www.it-academy.bg/zph/
[13] NetIQ Chariot, http://www.netiq.com/products/chr/default.asp
[14] J. Heinanen, “Assured Forwarding PHB”, RFC 2597, http://www.faqs.org/rfcs/rfc2597.html