A Prototype Implementation of the Two-Tier Architecture for Differentiated Services

Andreas Terzis, Jun Ogawa, Sonia Tsui, Lan Wang, Lixia Zhang

UCLA Computer Science Department

{terzis, ogawa, sonia, lanw, lixia}@cs.ucla.edu

Abstract

We present here a partial prototype implementation of the Two-Tier architecture for resource allocation in Differentiated Services networks, first presented in [ROTZY98]. Specifically, we have implemented the low level forwarding path support for providing different levels of network services. The allocation of resources is managed by the Bandwidth Broker. Our Bandwidth Broker implementation contains a policy database which stores information about flows requiring increased network service, and the Bandwidth Broker configures routers forwarding parameters in the domain accordingly. This policy database can be accessed and updated through a web interface.

Our preliminary experimental results show that this two-tier architecture can allocate network level resources according to administrative policies and provide high quality of network service to specific flows effectively. Our implementation is part of Qbone effort under Internet2 initiative, a testbed for IP quality of service technologies.

1. Introduction

Providing end-user applications with increased levels of network service has been the holly grail of the QoS network community for the last 10-15 years. Over the last few years, research and development effort has focused on providing Differentiated Services. The architecture of differentiated services specifically focuses on scalability as its fundamental goal and provides a close match to the business needs for the Internet services.

In this paper, we describe a prototype implementation of the Differentiated Services architecture and present our preliminary performance results. We modified the FreeBSD kernel to provide differential packet forwarding services. The allocation of resources is orchestrated by a novel entity called a *Bandwidth Broker* which was first introduced in [NJZ97]. Our prototype implementation of the Bandwidth Broker retrieves users bandwidth usage and policy information from a Flow database and sets appropriate control parameters in the routers along the packet forwarding paths.

The rest of the paper is structured as follows. In the rest of Section 1, we briefly present the Differentiated Services framework and the Two-Tier resource management architecture. Section 2 gives an overview of our implementation architecture. Section 3 presents the enhancements needed in the packet forwarding path to provide different levels of service. Sections 4 and 5 describe the control elements and functions used in resource allocation. Finally, Section 6 presents our preliminary experimental results and Section 8 discusses future work.

1.1. Differentiated Services

The Differentiated Services architecture, [RFC2475] was conceived by the IETF community as a way to provide scalable service differentiation over the Internet. This new framework achieves scalability and flexibility by making a fundamental distinction between the following two components of the architecture:

- **Forwarding Path**. This component includes differential treatment of individual packets at each network node, as implemented by packet service disciplines and/or queue management disciplines. In addition, the forwarding path may require that some monitoring, policing and shaping be done on the network traffic designated for “special” treatment.

- **Management Plane.** This part includes the configuration of network nodes with respect to which packets get special treatment and what kind of rules are to be applied to the use of resources.

In the forwarding path, differentiated services are realized by sorting each packet into specific forwarding treatment by the value contained in the TOS[1] field of the IP packet header at each network node on its path. For example, if the value carried in a packet means "low delay", then a router would put that packet on a high priority queue to forward it promptly. Since routers only have to look at the packet TOS field to decide the proper treatment, neither complex classification nor per-flow state is needed. End-hosts or first-hop routers set the values in the DS field according to the service quality each packet is required and entitled to receive.

Given the mode of operation described above, it is clear that if packets are marked irrespective of the amount of available network resources, the desired service behavior cannot be assured. Thus one must control the amount of traffic injected to the network at every service level. To carry out this task, the Differentiated Services architecture introduces *control elements* whose job is to enforce that traffic conforms to predefined profiles. The family of control elements includes **policers** that cut off excess traffic and **shapers** that provide temporary buffering to make traffic conform to the specified profile. Setting the shapers' and policers' parameters is the task of the Management Plane described in the next section.

1.2. The Two-Tier Architecture

Up to now work in Differentiated Services in IETF has mainly focused on defining the low-level forwarding mechanisms, while the design of the management plane remains a research issue. One proposal for scalable resource allocation to provide Differentiated Services was presented in [ROTZY98] which introduces a *Two-Tier* architecture for resource management. This architecture has two main characteristics:

1. Individual administrative domains manage their internal resources with approaches that best match their own needs. For example, some domains may rely on adequate provisioning, some may use static configuration, while others may deploy dynamic reservation protocols such as RSVP.
2. At domain boundaries, two neighboring domains allocate resources for each other's border-crossing traffic in *bulk* accounting for the total needs of aggregate data flows. The agreement on the amounts of the allocated resources, commonly called service level agreements (SLAs), is strictly *bilateral* between neighboring domains only. In such an agreement the sender side offers the payment and the receiving side pledges to serve the incoming traffic according to the specified treatment classes and up to the specified amount. The amount of resources described in SLAs can be dynamically adjusted, however the adjustment periods should be much longer than the lifetime of individual application flows in order to achieve the desired system scalability and stability.

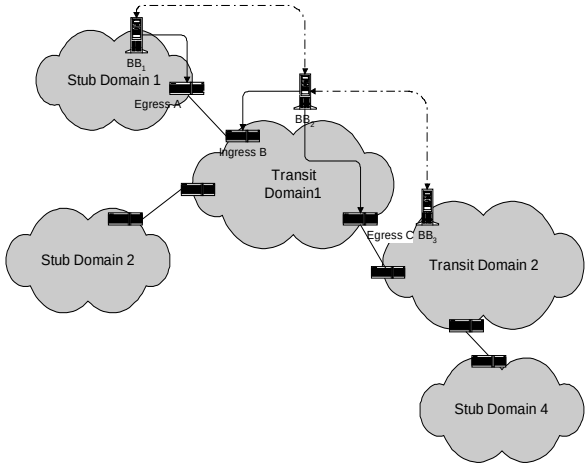

Fig. 1. The Two Tier Hierarchy

Both the allocation of *intra-domain* resources and the arrangement of *inter*-domain agreements are managed by the *Bandwidth Broker (BB)*, a resource manager entity which was first introduced in [NJZ97]. A Bandwidth Broker for each domain can be configured with organizational policies. It keeps tracks of the current resource usage and interprets new requests in light of the policies, current allocation of internal resources, as well as the current SLAs with neighbor domains. To provide its users or clients with end-to-end premier services a Bandwidth Broker negotiates adequate amount of resources with the BB of next hop neighbor domain, which in turn assures adequate allocation further down the road. It is this concatenation of bilateral agreements across domains, together with adequate intra-domain resource allocation, that achieves end-to-end service goals.

An example of this model is shown in Fig. 1. Assume that some new data flow requiring premier network service is traveling from Stub Domain 1 to Stub Domain 4. To satisfy the resource needs for this new flow, BB_1 and BB_2 readjusted resources at the domain boundaries between Stub Domain 1 and Transit Domain 1. BB_1 then instructs Egress Router A to set the appropriate shaping parameters at the outgoing interface, and BB_2 instructs Ingress Router B to adjust the corresponding policing

[1] The IP header contains an 8-bit field called Type of Service (TOS). This field was not widely used up to this date. The Differentiated Services architecture renamed this field to *DS Field*.

parameters at the incoming interface from Stub Domain 1. BB_2 will also need to make sure that the incoming traffic will receive appropriate treatment inside Transit Domain 1 and will make the necessary arrangements to allocate resources for this traffic further downstream. In our example, BB_2 could aggregate the requirements from Stub Domains 1 and 2 and allocate enough resources over the next downstream boundary to Transit Domain 2.

2. Intra-Domain Implementation

After describing the overall picture of the Two-Tier architecture, this section presents our implementation of the *intra-domain* resource allocation. As Fig. 2 reveals, our implementation is roughly divided to two major components:

1. **Bandwidth Broker:** The entity responsible for resource allocation as described in Section 1.2.

2. **Routers:** forward packets according to the specified level of services.

There are two types of routers in our implementation: *interior* routers and *edge* routers. An edge router connects the local domain to neighboring domains. It is the task of edge routers to check and enforce that traffic crossing domain boundaries conforms to the existing SLAs. Another difference is that while resource allocation at edge routers is dynamically adjustable by the domain's Bandwidth Broker, resources at interior routers can be either statically configured, or dynamically allocated by a resource setup protocol, such as RSVP, to reserve resources on the path from the Ingress to the Egress Router.

The Bandwidth Broker (BB) maintains the domain's policy, which contains information regarding flows requesting increased level of service. These flows are either locally generated, or coming as an aggregate from neighboring domains for transit service. Based on the contents of the flow database, the bandwidth broker instructs the domain's edge routers to set their policer and shaper parameters appropriately. In our implementation, the protocol used for this communication is COPS [BCD99] because of its availability, although other protocols can also be used to serve the same purpose. The Bandwidth Broker contains a COPS server that sends configuration commands to a COPS client residing at edge routers. At the edge router side, the COPS client receives these commands and via the Forwarding Path Driver (FPD), translates them to parameters understood by the forwarding path.

3. Forwarding Path

We use the the term *forwarding path* to collectively refer to all the low level mechanisms used to forward packets from one node in the network to the next. Our prototype implementation uses PCs running FreeBSD as both end hosts and network routers. Our choice is based on the fact that FreeBSD offers high performance, is freely available, and is based on the BSD network code

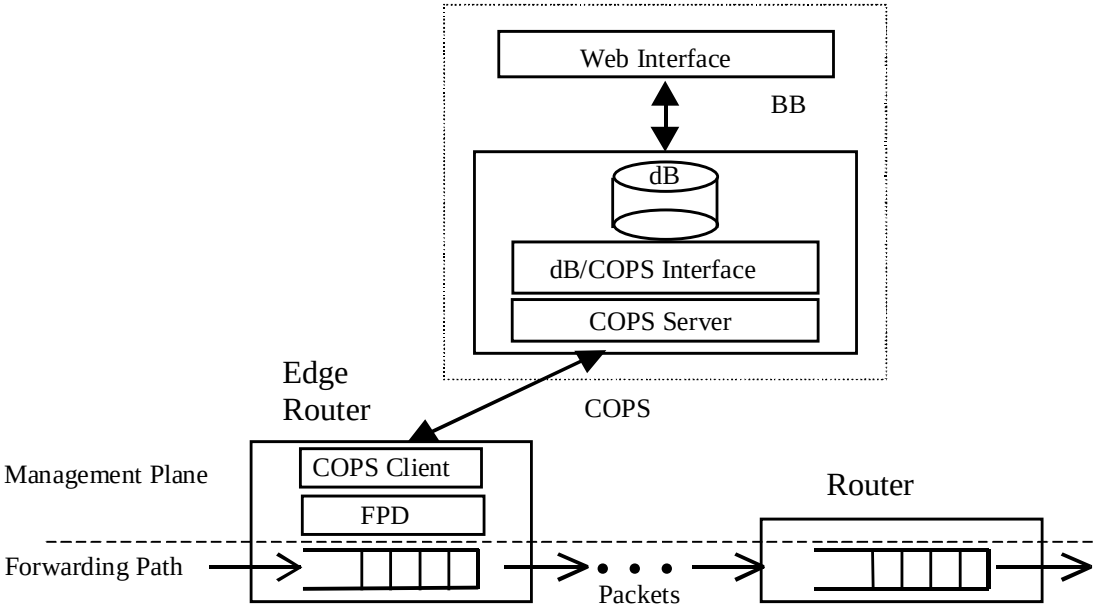

Fig. 2. Intra-Domain resource allocation architecture

which has been highly optimized and well documented. In addition, the ALTQ [Cho98] package implemented on FreeBSD, which is freely available from SONY CSL, offers a wide range of packet queuing disciplines.

As a first example of premier network services we decided to implement the EF PHB described in [JKP99]. The EF PHB can be used to build a low loss, low latency, low jitter, and assured bandwidth end-to-end service through domains supporting differentiated services. Although we have implemented only one service in addition to best effort, our implementation is extensible and we plan to implement other services in the future.

Subdividing the forwarding path functionality furthermore we can distinguish two subsets: the functionality required at the routers and the functionality required at the end-hosts. Routers do shaping and policing while end-hosts set the DS field in the packets. Each of these is described in the following sections.

3.1. Router

Figure 3 shows the processing path that packets go through inside a router. Packets arrive from an incoming interface, are processed inside the kernel, and are eventually forwarded through an outgoing interface. The processing phase consists of several stages. Initially the packet is delivered from the network interface in kernel space. Some sanity checks are then performed to make sure that the packets is in good shape and then the packet is delivered to the forwarding code that decides which is the appropriate outgoing interface based on the destination address in the packet's header. Once this decision has been made, the packet is delivered to the outgoing interface's queue where it awaits its turn to be transmitted.

In our architecture policing is implemented in the incoming interface while shaping is implemented in the outgoing interface. We took this decision to make flow identification and separation easier, since flows from different incoming interfaces that converge to the same outgoing interface can be policed separately before getting multiplexed over the same outgoing interface. Had we decided to do policing at the outgoing interface, we would have to implement more complex classifiers able to discern between packets belonging to flows from each of the incoming interfaces.

Since all of ALTQ's functionality is implemented at the outgoing interface we had to add the needed functionality in the incoming interface. Once packets arrive at the kernel, they go through a classifier that categorizes them according the value of the DS field in the IP header. If the packet is an EF packet, it is passed to the Policer module, otherwise it is immediately given to the forwarding module. The Policer module is implemented using a token bucket mechanism. A token bucket has two parameters: a token rate and a bucket depth. Tokens are generated at a specified rate. No more tokens than the bucket depth can be accumulated. Each time a packet arrives, the Policer checks if there are enough tokens. If there aren't enough tokens the packet is dropped, otherwise the packet moves to the next stage. This next stage is forwarding where a route lookup is made and the appropriate outgoing interface is found.

The specification of the EF behavior dictates that EF packets should be given priority at the outgoing interface over best-effort packets. Moreover at domain boundaries the amount of EF traffic must be shaped according to the existing SLA. For these two tasks, we use the existing CBQ module [FJ95] provided by ALTQ. CBQ is a mechanism that allows a hierarchy of arbitrary defined traffic classes to share the bandwidth on a link in a controlled fashion. In our case, the trivial CBQ hierarchy shown in Fig. 4, is used. There are only two classes: EF and best effort traffic. EF traffic is allocated up to x % of the outgoing interface's bandwidth and is given priority over best effort traffic. Best effort traffic can in the worst case use the remaining (100-x)% of the outgoing interface's bandwidth.

The difference between *interior* and *edge* routers

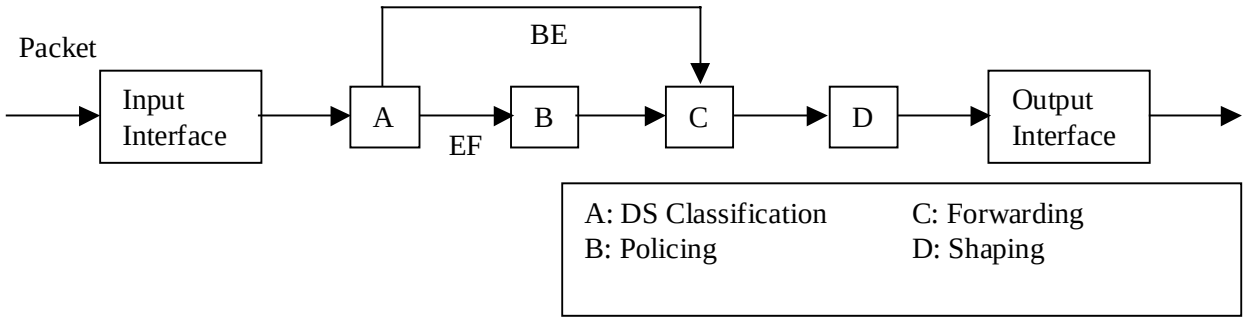

Fig. 3. Forwarding Path Components

inside a domain is that for interior routers the policer and shaper parameters are statically configured while in edge routers these parameters are configured by an agent following the commands of the domain's BB, as we will see in later sections.

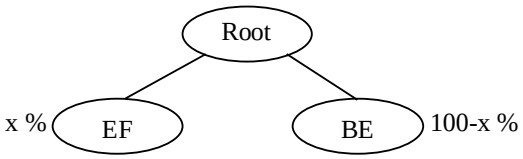

Fig. 4. Allocation of resources at the output interface.

3.2. End Host

At the end host, applications can specify their need for increased level of network service for their packets by requesting from the kernel to mark their packets with the EF codepoint. To do so, we have implemented a new socket option named `SO_EF`. Using this option TCP, UDP and raw sockets can be marked with the EF codepoint. Applications wanting enhanced level of service for their traffic can open a socket using the socket() function and then call `setsockopt(SO_EF)` for that socket. When the kernel receives a packet from that socket it sets the EF value in the DS field of the IP header and forwards the packet.

Based on this feature, we have modified the `tg` traffic generator utility providing the capability of sending EF packets. This utility has been used to test our router implementation and measure the performance of our router code.

4. The Bandwidth Broker (BB)

The Bandwidth Broker, as we have already said in Section 1.2, is the entity in charge of resource management in an administrative domain. There are two fundamental requirements for an entity to be able to perform this task:

1. Enough **information** must be available regarding the flows requiring network service inside and through the local domain.
2. The entity should be able to **control** the behavior of the domain's forwarding elements in order to provide the required service.

To satisfy this dual needs our Bandwidth Broker implementation consists of two parts: **i.)** A flow database containing information about flows requiring differentiated level of service and **ii.)** a COPS server that communicates with a COPS client at the edge routers to set the appropriate parameters at the forwarding path. In the following two sections, we present each of the components in more detail.

4.1. Flow database

The Flow Database stores information about the flows that request increased level of network service from the local domain. These are either local flows whose sender resides in the local domain or flows that transit through the local domain on their way to their final destinations. The database stores the following information about each of these flows:

- **ingress interface**: The interface through which the flow enters this domain[2].
- **egress interface:** The exit point through which the flow leaves this domain.
- **Resources requested:** Expressed as token bucket parameters, that is a pair [*r*,*b*] where *r* is the rate in bytes/second and *b* is the bucket depth in bytes.
- **start time:** Starting from this instant, the flow requires the resources mentioned above.
- **finish time:** After this time, no resources should be allocated for this flow and the ones reserved must be revoked.
- **auxiliary information:** such as contact information etc.

We have implemented the database using the freely available MySQL RDBMS [MySQL]. To provide easier access to the flow database we have also implemented a Web-based front-end to it. Using this front-end, the authenticated network administrator can make queries about existing flows, add new flows or delete old ones. The implementation of the front-end is based on the PHP3 scripting language. PHP3 is a server-side scripting language that integrates nicely with our database back-end. It is relatively easy to use and is supported by the Apache web server. For more information on PHP3, the interested user can find a wealth of information in [PHP3].

4.2. COPS Server

Once the requested resource allocation information is loaded into the flow database, the next step is for the Bandwidth Broker to send the appropriate configuration parameters to the domain's edge routers. But before this is done, an intermediate step takes place. As we have seen in the previous section, the policy database contains

[2] for local flows, the first hop router is consider to be the flow's ingress router

individual flows that have an entry and an exit point through the local domain. Some of these flows will have the same entry or exit points and therefore their resource requirements have to be aggregated before configuration parameters for the total resource usage can be passed to the domain's routers. The BB goes through this process twice, once for incoming traffic and once for outgoing traffic. When this process is complete, it contacts each of the domains edge routers to configure the shaping and policing parameters. The detailed message exchange between the BB and the edge routers is described in detail in the following section.

5. Edge Router

The control plane components of the edge router can be divided to two parts. The first part is the COPS client used to exchange configuration information with the domain's Bandwidth Broker and the second part is the Forwarding Path Driver (FPD) which translates configuration commands received by the Bandwidth Broker to parameters understood by the forwarding path mechanisms.

5.1. Edge router-BB Communication

The COPS (Common Open Policy Service) Protocol [BCD99] is a query and response protocol that is used for exchanging policy information between a policy server and its clients. In this protocol, the server, also called *Policy Decision Point* (PDP), keeps configuration information about a domain's resource management and installs these configurations to clients, also named as *Policy Enforcement Points* (PEPs). Although COPS supports several operations, we mainly use the Configuration operation in which PDP sends configuration commands to the PEPs. In our architecture, BB plays the role of the PDP and edge routers contain the PEPs. A detailed specification of the message exchanges between the BB and the edge routers can be found in [RCD+98]. We will only describe the main mechanism involved briefly here.

Communication between the BB and the edge routers is achieved mainly by three kinds of COPS messages: `Request`, `Decision` and `Report` messages. When an edge router (PEP) makes a request for configuration information during initialization for example, it sends a `Request` message to the BB (PDP). The BB replies by sending a `Decision` message containing the appropriate parameters for the corresponding interface of the edge router. After getting this message, the edge router replies by sending the configuration result (i.e. success or failure) to the BB via a `Report` message. Figure 5 shows the sequence of messages exchanged between the Bandwidth Broker and the edge router.

As time passes, some of the installed parameters may need to be altered. For example a network administrator may want to add or delete a flow from the flow database or some of the existing flows have expired. This will trigger the forwarding of an updated `Decision` message from the BB to the edge router and subsequent forwarding of a Report message from the edge router.

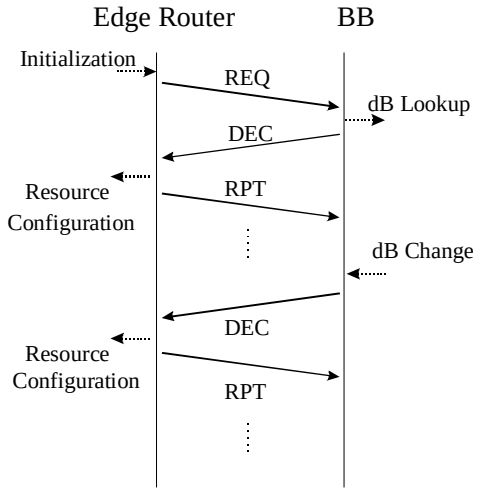

Fig. 5. BB-Edge Router Communication

The configuration parameters supported by our current implementation include enabling and disabling a router's policer and shaper, setting the parameters for the token bucket meter and shaper for the EF packets.

5.2. Forwarding Path Driver (FPD)

The *Forwarding Path Driver* provides an interface between the COPS client and the Forwarding Path. When the PEP receives a configuration request from the BB, it removes the COPS specific headers and sends it to the FPD. The FPD then installs the appropriate parameters to the forwarding path (e.g. the policing parameters) and sends the reply received from the forwarding path back to the PEP. It is this value that will be sent back to the BB inside a `Report` message. One of the significant characteristics of the FPD is its ability to do parameter checking. The FPD before installing any parameters received in a request to the Forwarding Path, it verifies their validity. For example, a request that asks more bandwidth than the interface bandwidth is obviously invalid. If any errors are found, the FPD sends a negative notification to the PEP without installing any parameters. The FPD can also be used to install a static configuration at the forwarding path independently from

the BB. Such a capability provides increased robustness against BB crashes.

6. Experimental Results

To test our implementation and to get some first estimates of its capabilities we have run some initial experiments. The topology we used is shown in Fig. 6. Our topology contains three hosts and one router. Camelot and Gawain are the sending hosts and Lancelot is the destination for both traffic streams. Gawain is sending best effort traffic while Camelot is sending EF traffic. All of Avalon's interfaces are 10Mbps point-to-point Ethernet links. Avalon acts both as ingress and as egress router. So the incoming interfaces I1 and I2 do policing while the outgoing interface towards Lancelot does shaping. Camelot has an exponential source with average transmission rate of 8 Mbps, while Gawain is sending best effort traffic at a constant rate of 8 Mbps.

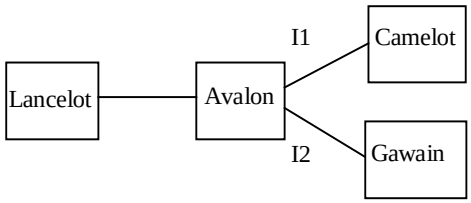

Fig. 6. Experiment Topology

For policing, the token bucket parameters on interface I1 have been set to [4 Mbps, 20000bytes] where 4Mbps is the policing rate and 20Kbytes is the bucket depth. On the outgoing interface we have also allocated 4 Mbps to EF traffic and 6 Mbps to best effort traffic.

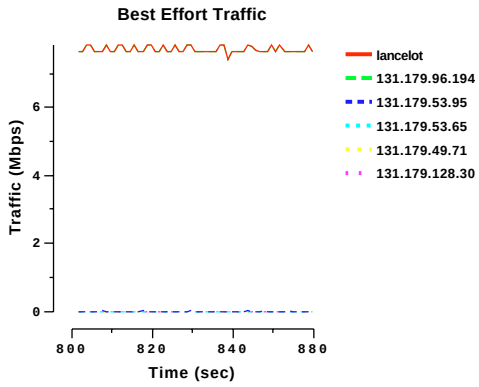

Fig. 7. Incoming best effort traffic

Figures 7 and 8 shows the incoming best effort and EF traffic respectively. Note from the graphs that best effort traffic is smoother than EF as expected. The other lines in Figures 7 and 8 are background traffic.

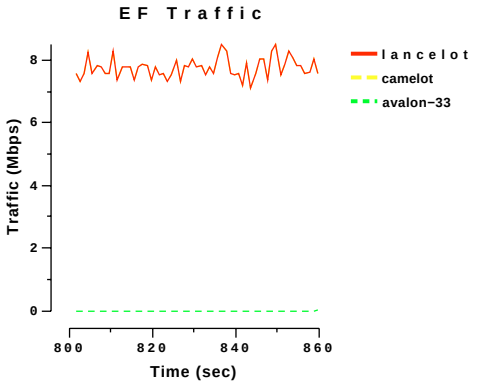

Fig. 8. Incoming EF traffic

Figure 9 shows the traffic at Avalon's output interface. At *t=800* only the EF flow is on. We can see that even though the incoming EF traffic is coming at a rate of 8 Mbps it is successfully policed at the configured rate of 4 Mbps. At time *t=810* we turn on the best effort traffic but we can clearly see that the EF flow is not affected by it. Due to shaping, the best effort flow get 6 Mbps of the output interface's bandwidth. At time *t=850* we turn of the EF flow and we can immediately see how the best effort traffic ramps up and consumes the extra available bandwidth.

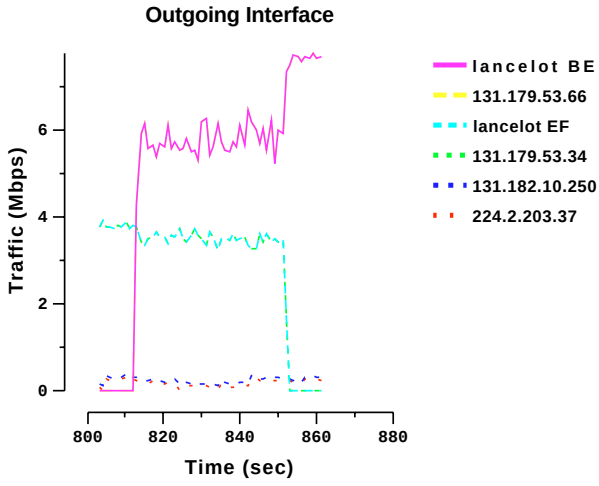

Fig. 9. Outgoing Traffic

7. Summary

We described in this paper a prototype implementation of intra-domain resource control under the two-tier differentiated services architecture. The two-

tier architecture divides the resource allocation task into two major components: the *intra-domain allocation*, by which each administrative domain allocates internal resources independently and the *inter-domain allocation*, where neighboring domains jointly allocate aggregate resources at domain boundaries. In our effort we have implemented the forwarding path mechanisms required to provide the EF behavior. We also implemented a control entity known as the *Bandwidth Broker*, which oversees the allocation of resources inside the local domain and allocates resources at domain boundaries. The Bandwidth Broker contains the domain's flow database holding information about resource needs and communicates through COPS with the domain's edge routers to set policing and shaping parameters according to these needs. Our preliminary results show that the mechanisms implemented can provide improved support to applications requiring increased levels of network service and at the same time protect the domain's resources against resource abuse.

8. Future Work

At this point our prototype implements only one part of the complete architecture presented in Section 1.2, namely the intra-domain resource allocation mechanism. Our plan for the future is to implement the rest of our architecture. Specifically we plan to move along the following two directions:

- *Implement a dynamic intra-domain resource allocation protocol*. The allocation of internal router resources is statically configured at this time, which we plan to replace with a dynamic allocation mechanism along the paths between ingress and egress routers of a domain. One simple scheme may work as follows: The ingress router uses information received by the domain's BB to send RSVP PATH messages to the required egress routers, and the egress routers will respond by sending RESV messages reserving the required resources on the path between the ingress and the egress routers.
- *Implement automated inter-domain resource allocation control.* At this point, information about transit flows is exchanged between neighboring domains via some out-of-band mechanism, such as e-mail or phone calls between network administrators. We plan to develop a BB-to-BB protocol to automate the communication between neighboring BBs allowing for more dynamic inter-domain resource allocation..

References

[BCD99] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, A. Sastry, *"The COPS (Common Open Policy Service) Protocol"*. Work in progress, Internet-Draft, February 1999

[Cho98] K. Cho, "*A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers.*" In Proceedings of USENIX 1998 Annual Technical Conference, New Orleans LA, June 1998.

[FJ95] S. Floyd, V. Jacobson, *"Link Sharing and Resource Management Models in Packet Networks"*. IEEE/ACM Transactions on Networking, August 1995.

[JKP99] V. Jacobson, K. Nichols, K. Poduri, *"An Expedited Forwarding PHB"*. Work in progress, Internet-Draft, February 1999.

[NJZ97] K. Nichols, V. Jacobson, L. Zhang, *"A Two-bit Differentiated Services Architecture for the Internet*". Work in progress, Internet-Draft, Nov 1997.

[RCD+98] F. Reichmeyer, K. Chan, D. Durham, R. Yavatkar, S. Gai, K. McCloghrie, S. Herzog, "*COPS Usage for Differentiated Services*". Work in progress, Internet-Draft, Dec 1998.

[ROTZY98] F. Reichmeyer, L. Ong, A. Terzis, L. Zhang, R. Yavatkar, *"A Two-Tier Resource Management Model for Differentiated Service Networks*". Work in progress, Internet-Draft, Nov 1998.

[RFC2474] K. Nichols, S. Blake, F. Baker, D. Black, *"Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers"*. RFC 2474, December 1998.

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *"An Architecture for Differentiated Services"*. RFC 2475, December 1998.

[MySQL] Available at http://www.tcx.se/

[PHP3] Available at http://www.php3.net/