Building an IP Differentiated Services Testbed

Jaudelice Oliveira, Tricha Anjali, Benjamin King, and Caterina Scoglio

Broadband and Wireless Networking Lab
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332
E-mail: {jau,tricha,king,caterina}@ece.gatech.edu

Abstract— **In this paper, the steps taken towards setup and configuration of the DiffServ testbed in our laboratory are listed. A topology is suggested and some simple experimental results for setup validation are presented. A DiffServ domain is deployed and several experiments are run. We obtained acceptable performance for a DiffServ Expedited Forwarding Per Hop Behavior in our testbed using Committed Access Rate and Class Based Weighted Fair Queueing.**

Keywords— **Quality of Service, DiffServ, Testbed, Internet, IP QoS.**

I. INTRODUCTION

THE current Internet Protocol (IP) provides a single service class - Best Effort. Best effort does not provide any Quality of Service (QoS) guarantees, which are required for most present and emerging applications. The Next Generation Internet [15] and Internet2 [13] consortiums have the same goal: to enhance the current Internet protocol in order to include support for multiple service classes. The IETF is studying several possible solutions to the issue of providing QoS over the Internet, or IP QoS.

One of the most promising IETF solutions for IP QoS support is the Differentiated Services (DiffServ). The approach followed by DiffServ is to classify individual microflows at the edge of the network into several existing service classes and then apply per-class services at the core of the network. Fig. 1 illustrates a DiffServ network.

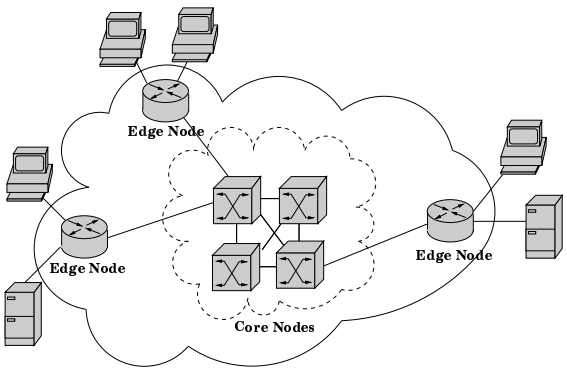

Fig. 1. Edge and core routers in a DiffServ network.

At an ingress (edge) router, packets are classified based on the information contained in one or more fields in the packet header. Each packet is marked as belonging to a certain class (by setting some bits [*codepoints*] in the packet header), and reinserted into the network. A Per Hop Behavior (PHB) defines the service the packet should receive on its way through the network. The core routers provide the PHB based on the codepoint. Typically, queue management and scheduling disciplines are employed to provide the PHB, such as RED [4] and CBWFQ [9].

Packet markers set the DiffServ field in the IP header to a particular codepoint. Shapers and droppers are used to ensure the conformance of the traffic stream to the traffic profile.

Fig. 2 illustrates the edge and core functions in a DiffServ domain. A DiffServ domain (DS domain) is a set of DiffServ nodes which operate under a common policy. Moreover, every node in a DiffServ domain has the same set of PHBs.

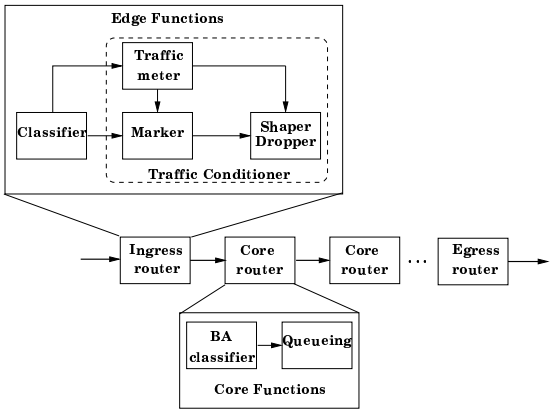

Fig. 2. Example DiffServ domain.

DiffServ may be able to provide a scalable and coarse level of service to enterprise networks. There exist several approaches to DiffServ implementation [1], [5], [6], [7]. In order to test the performance of these different approaches, a physical testbed is a necessary entity. In this paper, we describe how to setup and configure a DiffServ testbed. We also describe some experiments to validate the correctness of the implementation. The rest of the paper is organized as follows. In Section II, we describe the

testbed implementation, covering the software and hardware used. Testbed setup experiments are detailed in Section III. In Section IV, requirements fot setting up a DiffServ domain and experimental results are discussed. Finally, we conclude the paper in Section V.

II. TESTBED DESCRIPTION

The key elements of our testbed are routers capable of performing QoS functions and switches to segregate the network into separate VLANs. A traffic generator capable of overloading the network and a measurement tool for sampling the network statistics are also essential.

In the following, we are going to motivate and describe the hardware/software utilized in our testbed implementation, as well as the chosen testbed topology.

A. Testbed Hardware

Cisco has released new routers that provide alternative queuing mechanisms, therefore supporting DiffServ. Both Cisco 7200 and 7500 router series make use of Cisco Internetwork Operating System (IOS) software, and provide these capabilities.

When deciding on the testbed topology, we chose a Cisco 7500 router to be configured as an edge node, and two Cisco 7200 Series VXRs as edge and core routers. Moreover, a Cisco Catalyst 6506 Layer 3 switch was connected to the routers. The host PCs were connected to a Cisco Catalyst 4000 Layer 2 switch. VLANs were configured in both switches.

In the near future, experiments will be performed over Abilene [10]. For that reason, and also to be able to perform ATM/IP QoS experiments, an ATM switch, Cisco LightStream 1010 was included, and the testbed was connected through a Gigabit link to Abilene . In Fig. 3 a physical view of the testbed is shown.

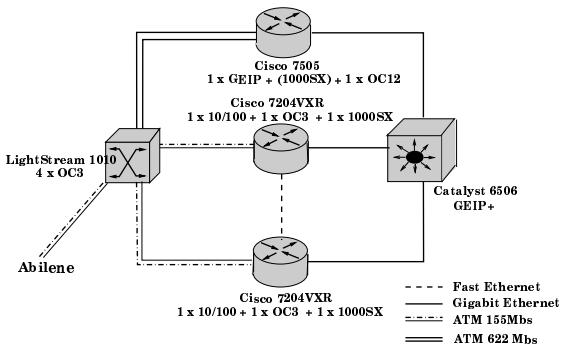

Fig. 3. Testbed configuration: physical view.

Both 7204VXR Cisco routers contain one PA-A3 Enhanced ATM Port Adapter and a PA-GE Gigabit Ethernet Port Adapter. The Catalyst 4000 contains 32 10/100 ports. The Catalyst 6506 contains the GEIP+ (Enhanced Gigabit Ethernet support). The ATM switch, LightStream 1010, contains 4 OC3 ports.

B. Testbed Software

All computers have MS Win 98 and FreeBSD 4.2 [12] installed. The tests were conducted in FreeBSD using Alternative Queue (ALTQ) 2.2 [2], [3], [11]. ALTQ is a package for traffic management in FreeBSD environment. It includes a queuing framework and several advanced queuing disciplines (CBQ, WFQ, PQ etc). It also includes an implementation of the simple FIFO mechanism for researchers to modify for customized queuing. Moreover, ALTQ also supports RSVP and DiffServ.

CBQ is a non-work conserving scheduling mechanism. It consists of a classifier, estimator and a packet scheduler. In our testbed, we are using CBQ to configure the bandwidth assigned to different flows in classes. Queuing is effective at the ingress of a bottleneck link. The configuration file for CBQ is set as shown in the examples below. First, the interface and the bandwidth (in bps) are specified. The "*class*" command creates a class and specifies the CBQ mechanism, interface, class name, parent class name, and the percentage of the interface bandwidth assigned."*borrow*" is set when the class can borrow bandwidth from the parent. "*filter*" command sets a packet filter to a class. CBQ uses <dst_addr, dst_port, src_addr, src_port, protocol> argument. The "*altqstat*" command monitors the ALTQ process.

Iperf [14], used for traffic generation, is a tool for measuring maximum TCP and UDP bandwidth. When running an experiment, *iperf* server is started on one of the computers. Then on another computer *altqd* and *iperf* client are run to generate the traffic. The routers involved in the testbed are running Cisco IOS version 12.1.

III. TESTBED SETUP EXPERIMENTS

In order to validate the testbed setup, we performed two simple experiments and observed the expected results. Following we describe these experiments and how to configure ALTQ and *iperf* to run them.

Experiment 1: We connected three PCs to the testbed. Two of them were connected to the same VLAN and the other one to a different VLAN. Two TCP flows, belonging to the same class (A), were sent from one host through the 7200 router to hosts in another VLAN. Only 10% of the total available

bandwidth was assigned to this class. The assigned bandwidth was equally divided among the two flows.

The configuration files for ALTQ and *iperf* are reported in Fig 4, and the experiment results are shown in Table I.

ALTQ configuration file:
interface ep0 bandwidth 10M cbq
class cbq ep0 root_class NULL pbandwidth 100
class cbq ep0 def_class root_class borrow pbandwidth 95 default
class cbq ep0 A def_class pbandwidth 10 filter ep0 A 0 0 0 0 6
Traffic generator file:
Server: iperf -s -f m -w 64k -p 5000
Client: iperf -c SERVER-ADD -f m -w 64k -p 5000

Fig. 4. ALTQ and iperf configuration for Exp. 1.

TABLE I
RESULTS FOR EXPERIMENT 1.

Flow	Stream	Class	Class Bandwidth	Output
A	TCP	A	10%	0.5 Mb/s
B	TCP	A	10%	0.5 Mb/s

Experiment 2: With the same topology as before, we defined two classes (A, B) in ALTQ. 10% of the total bandwidth was assigned to each class. Two traffic flows were generated, one for each class, and sent from Athens to Sydney and Atlanta. The results were as shown in Table II. The configuration file for ALTQ is in Fig.5. The *iperf* configuration is the same as in Experiment 1.

ALTQ configuration file:
interface ep0 bandwidth 10M cbq
class cbq ep0 root_class NULL pbandwidth 100
class cbq ep0 def_class root_class borrow pbandwidth 95 default
class cbq ep0 A def_class pbandwidth 10
filter ep0 A 192.168.210.11 0 0 0 6
class cbq ep0 B def_class pbandwidth 10
filter ep0 B 192.168.210.12 0 0 0 6

Fig. 5. ALTQ configuration for Exp. 2.

TABLE II
RESULTS FOR EXPERIMENT 2.

Flow	Stream	Class	Class Bandwidth	Output
A	TCP	A	10%	1 Mb/s
B	TCP	B	10%	1 Mb/s

These simple experiments had the purpose of validating the configuration files since the results were very easy to predict. Now, we discuss how to setup a DiffServ domain prototype, making use of CAR and CBWFQ features from Cisco Routers and IOS software.

IV. DIFFSERV EXPERIMENTS

In order to support DiffServ packet coloring and policing are required. Packet coloring can be done either by the application originating the traffic or by a node in the network. Cisco features such as CAR and CBWFQ [8] can be used to perform these tasks in a node.

We constructed a prototype DiffServ domain to provide premium service using the expedited forwarding (EF) per-hop behavior (PHB) and three layer olympic service using assured forwarding (AF) PHB. For metering, policing, and marking of traffic, three rate color markers were used. CBWFQ was the queuing discipline supporting all services (i.e., premium, olympic, and best-effort).

CBWFQ is an alternate queuing discipline released in 12.0.5T version of Cisco IOS. It extends the standard WFQ functionality to provide support for user-defined traffic classes. Coupled with Committed Access Rate (CAR) policing and marking capabilities, CBWFQ provides the congestion management and traffic shaping needed to support DiffServ boundary router functionality.

To configure CBWFQ, first a numbered access list has to be defined as the match criterion for any class. This is done using the "*access-list*" command. Then a class-map is defined which specifies the match condition of a packet for a class by using the "*class-map*" command. Next step is to define a "*policy-map*" which characterizes the policy for the classes. These policies can then be attached to the interfaces of the router. A different subqueue is allocated for each traffic class in CBWFQ. A minimum bandwidth guarantee per class can be directly specified by using the "*bandwidth*" command.

The traffic policing or rate-limit function is provided by CAR. CAR offers two primary functions: packet coloring by setting IP precedence, and rate-limiting. As a traffic policing function, CAR does not buffer or smooth traffic and might drop packets when the allowed bursting capability is exceeded. CAR is implemented as a list of rate-limit statements. You can apply it both to the output and input traffic on an interface.

To configure CAR, the "*rate-limit*" command is used on the appropriate interface (input or output) with mean rate, normal burst size, maximum burst size and conform- and exceed- actions. The mean rate is given in bits per second and burst sizes in bytes. Using the conform and exceed actions we can decide to either set a lower transmit priority or drop a non-conforming packet.

We tested CAR and CBWFQ to determine the

operating parameters that would produce acceptable performance using the testbed configuration shown in Fig. 3. Three series of tests were conducted to measure the effect of Cisco QoS features on combinations of TCP and UDP flows. These flows were sourced from two PCs to a sink PC in another VLAN. The router used CAR to police and mark ingress traffic. The same router applied CBWFQ on egress traffic. The configuration commands for CAR and CBWFQ in a Cisco router are shown in Fig. 6.

For the following experiments we considered the EF PHB implementation. An EF PHB requests every router along the path to always service EF packets at least as fast (if not faster) as the rate at which the EF packets arrive. In order to achieve such behavior, CAR and CBWFQ need to be configured to shape and police the traffic so that the EF traffic rate is not affected by any non-EF traffic.

CAR configuration:
```
interface Hssi0/0/0
    ip address vvv.xxx.yyy.zzz. 255.255.255.255
    rate-limit input 30000000 200000 800000 conform-action trans-
mit exceed-action drop
```
CBWFQ configuration:
```
access-list 1 permit host aaa.bbb.ccc.ddd
class-map premium
   match access-group 1
policy-map premiumservice
    class premium
    bandwidth 1000
interface serial0
    service-policy output premium-service
access-list 10 permit host eee.fff.ggg.hhh
class-map olympic
   match access-group 10
policy-map olympicservice
    class olympic
    bandwidth 200
interface serial0
    service-policy output olympic-service
access-list 100 permit host iii.jjj.kkk.lll
class-map besteffort
   match access-group 100
policy-map besteffortservice
    class besteffort
    bandwidth 10
interface serial0
    service-policy output besteffort-service
```

Fig. 6. Example of CAR and CBWFQ setup commands.

A. Case Study 1:

In the first series of experiments, two TCP flows compete for the available bandwidth, and policing and scheduling mechanisms are applied to enforce a specific share.

Experiment 1: Two TCP flows were sent with no scheduling/policing in effect. The TCP flows shared the available bandwidth equally, as expected.

Experiment 2: CAR is now used to police the traffic of one TCP stream to a maximum rate of 30 Mbps.

Flow	Stream	Rate	CAR	CBWFQ	Output
1	TCP	-	-	-	45 Mb/s
2	TCP	-	-	-	45 Mb/s

Another TCP stream has a minimum rate of 60 Mbps specified by CBWFQ. We observe that the policed flow (CAR) has a rate below the maximum required (30 Mbps) and that the other flow keeps its rate above the minimum imposed by CBWFQ (60 Mbps).

Flow	Stream	Rate	CAR	CBWFQ	Output
1	TCP	-	30 Mb/s	-	25 Mb/s
2	TCP	-	-	60 Mb/s	65 Mb/s

When CAR is used alone to policy a TCP flow, the flow rate will be kept within the maximum specified. Other TCP flows sharing the same link will equally share the rest of the available bandwidth. Due to the TCP congestion mechanism, in the event of congestion TCP flows redude their rate and increase it when there are available resources. Using CAR one can guarantee a certain maximum rate to an specific flow. On the other hand, CBWFQ enables the user to directly specify the required minimum bandwidth per traffic class.

B. Case Study 2:

The second series of experiments shows two UDP flows and the effect of CAR and CBWFQ.

Experiment 1: Two UDP flows are sent, each at 50 Mbps source rate. The flows share the available bandwidth equally. No policy or scheduling technique is used.

Flow	Stream	Rate	CAR	CBWFQ	Output
1	UDP	50 Mb/s	-	-	45 Mb/s
2	UDP	50 Mb/s	-	-	45 Mb/s

Experiment 2: One of the UDP flows now has a maximum of 30 Mbps (CAR) and the other a minimum of 60 Mbps. The results again show that the CBWFQ flow uses more than the minimum specified and about its source transmission rate, and the CAR flow uses the leftover bandwidth, which is below its specified maximum rate.

Flow	Stream	Rate	CAR	CBWFQ	Output
1	UDP	50 Mb/s	30 Mb/s	-	20 Mb/s
2	UDP	70 Mb/s	-	60 Mb/s	70 Mb/s

C. Case Study 3:

The third series of experiments shows two flows, one TCP and one UDP, competing for the available

bandwidth. Again, policing and scheduling mechanisms are applied to enforce a specific share.

Experiment 1: The UDP flow is sent with a transmission rate of 50 Mbps. No policing or scheduling is applied. The result shows that the UDP flow got the required rate and that the TCP flow used the rest of the bandwidth.

Flow	Stream	Rate	CAR	CBWFQ	Output
1	TCP	-	-	-	35 Mb/s
2	UDP	50 Mb/s	-	-	50 Mb/s

Experiment 2: The TCP flow is now policed with CAR (30 Mbps) and the UDP flow has a minimum rate of 60 Mbps specified by CBWFQ. Here we see that UDP gets its fair share, and TCP uses the rest of the available bandwidth.

Flow	Stream	Rate	CAR	CBWFQ	Output
1	TCP	-	30 Mb/s	-	25 Mb/s
2	UDP	70 Mb/s	-	60 Mb/s	65 Mb/s

Experiment 3: UDP flow sent with 70 Mbps source rate is now policed with CAR, with a maximum rate of 30 Mbps. We observe that the UDP flow is indeed below the maximum rate, and that the TCP flow took over the rest of the available bandwidth, as expected.

Flow	Stream	Rate	CAR	CBWFQ	Output
1	TCP	-	-	60 Mb/s	60 Mb/s
2	UDP	70 Mb/s	30 Mb/s	-	25 Mb/s

The above experiment results show that Cisco's CAR coupled with CBWFQ are capable of providing DiffServ boundary router service when properly configured. Tests of TCP and UDP streams in various combinations demonstrated the feasibility of deploying QoS services for IP traffic using the Cisco 7500 series router as a DiffServ boundary router.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we described how to setup and configure a DiffServ testbed. We suggested a topology and explained how to configure the hardware/software involved. We also showed a few experiments to validate the correctness of the implementation. A DiffServ domain was deployed and several experiments were performed in order to configure the expedite forwarding per-hop-behavior. The results show how to configure EF PHB using Cisco's CAR and CBWFQ.

As future work, we will study the effects on real-time traffic, and also the appropriate assignment of Expedited Forwarding (EF) and Assured Fowarding (AF) per-hop behaviors for such traffic. MPLS deployment is also part of the next step, as well as to run experiments over Abilene.

Acknowledgments

Special thanks to Dr. Ian Akyildiz, George Uhl, and Didier Contis for their collaboration and support of this project.

This work was supported in part by NASA Goddard, Raytheon, and by CAPES (The Brazilian Ministry of Education Agency).

References

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

[2] K. Cho, "A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers," in *Proceedings of USENIX'98 Annual Technical Conference*, New Orleans, LA, June 1998.

[3] K. Cho, "Managing Traffic with ALTQ," in *Proceedings of USENIX'99 Annual Technical Conference*, Monterey, CA, June 1999.

[4] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[5] B. Li, m. Hamdi, D. jiang, Xi-Ren Cao, and Y.T. Hou, "QoS-Enabled Voice Support in the Next-Generation Internet: Issues, Existing Approaches and Challenges," *IEEE Communications Magazine*, April 2000.

[6] L. Mathy, C. Edwards, and D. Hutchison, "The Internet: A Global Telecommunications Solution?," *IEEE Network*, July/August 2000.

[7] C. Metz, "IP QoS: Traveling in First Class on the Internet," *IEEE Internet Computing*, pp. 84–88, March-April 2000.

[8] S. Vegesna, "IP Quality of Service," *Cisco Press*, 2001.

[9] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet Switching Networks," *IEEE Proceedings*, vol. 83, no. 10, pp. 1374–1396, October 1995.

[10] Abilene Website, *http://www.internet2.edu/abilene/*.

[11] ALTQ Website, *http://www.csl.sony.co.jp/person/kjc/kjc/kjc/software.html*.

[12] FreeBSD Website, *http://www.freebsd.org/*.

[13] Internet2 Website, *http://www.internet2.edu/*.

[14] Iperf Website, *http://dast.nlanr.net/Projects/Iperf/*.

[15] Next Generation Internet Website, *http://www.ngi.gov*.