A Differentiated Services API for Adaptive QoS Management in IP Networks

Gérard Nguengang[1], Dominique Gaiti[2], and Miguel F. de Castro[3]

[1]LIP6, Université Pierre et Marie Curie, R&D Ginkgo-Networks
g.nguengang@ginkgo-networks.com
[2]ISTIT, Université de Technologie de Troyes
gaiti@utt.fr
[3]DC, Universidade Federal do Ceará
miguel@ufc.br

The introduction of quality of service in IP Networks brings solutions to real time applications requirements over Internet. Because of its simplicity, Differentiated Services architecture proposed by the Internet Engineering Task Force (IETF) is becoming the most employed solution for providing end-to-end QoS (Quality of Service) to network based applications. However, tuning QoS parameters in edge and core routers of a DiffServ domain for efficient resource utilization and stable network behavior is not an easy task. Because of the unpredictable and dynamic behavior of the Internet traffic, network entities need to take fast decisions and perform fast adaptations for the provision of adequate end-to-end QoS. This paper proposes an Application Programming Interface for near real time bandwidth allocation and parameters tuning of DiffServ queues in Cisco routers as a first step to address to the aforementioned problem.

1 Introduction

Due to the emergence of real time voice and video applications over Internet which are delay and jitter sensitive, the classic IP best-effort service is no longer sufficient. These multimedia applications have high bandwidth requirements as well as delay constraints and just adding links capacity is

not a rational solution. That is why, in recent years, there has been considerable research focused on extending the Internet architecture to allow Quality of Service. The Internet Engineering Task Force (IETF) has proposed two architectures for providing end-to-end QoS in the Internet: Integrated Services (IntServ) and Differentiated Services (DiffServ).

IntServ [4] architecture is based on reserving network resources for individual flows by the means of the Resource Reservation Protocol (RSVP) [5]. This raises two important deployment issues. First, all the routers along an end-to-end network path must be RSVP-capable in order to realize IntServ benefits. Second, a router has to manage per-flow state and perform per-flow processing. This is why IntServ is not a scalable solution for providing end-to-end QoS. It ensures absolute guarantees for each flow crossing the network but requires too much memory resource to maintain per-flow state since there may potentially be thousands of flows composing the Internet traffic.

In the DiffServ [3] architecture, traffic flows that are similar in terms of QoS requirements, are aggregated and identified as classes. Packets are classified based on contracted parameters according to their service requirements and are marked to receive particular per-hop behavior. Typically, different traffic streams that belong to the same traffic aggregate receive the same treatment when crossing the network. The Differentiated Services Code Point (DSCP) field in the IP header is used to carry the marking information. Since the number of DiffServ traffic classes is expected to be far fewer than the number of flows in IntServ, DiffServ is seen as the emerging technology to support QoS in IP backbone networks in a scalable fashion. However, tuning QoS parameters for edge and core routers of a DiffServ domain for efficient resource utilization and stable network behavior is not an easy task. The following quote from a router manual highlights the problem: “When first implementing WRED on the router, we recommend that you initially use the default values for min-threshold, max-threshold, the weighting constant, and the probability denominator. If you begin to experience congestion or if congestion continues (especially with TCP traffic), try adjusting WRED by making small changes to the various parameters (one at a time) and observing the effect on congestion” [13]. To overcome it, the network must be able to recognize and adapt itself to the running traffic fluctuations to enhance service performance. The unpredictable and dynamic behavior of network traffic requires a real-time operation to take fast decisions and to perform fast adaptations [15]. There is also a need for cooperation between network devices to define a global QoS policy for network stability and efficient end-to-end QoS in each traffic class. Our approach for solving this problem is to find a way to dynamically act on a router in order to perform effective

QoS parameters tuning. Hence, the aim of this paper is to better understand the provision of Quality of Service in market's routers and provide a Java library for DiffServ queues monitoring and scheduling configuration. The API is Cisco-specific but can be easily extended to other vendors.

This paper proposes a DiffServ based API for the monitoring and the configuration of QoS features in Cisco QoS routers. It is organized as follows. A brief description of the QoS features on a Cisco router is presented in Section 2. Afterwards in Section 3, we make reasoning about how to implement Differentiated Services with Cisco Internetworking Operating System (IOS). The developed API is described in Section 4. In Section 5, we present some experiments performed to check and validate the API's functionalities. Finally, we present our conclusion and outline some future works.

2 QoS features in Cisco routers

In order to provide end-to-end QoS, Cisco Systems has implemented a set of features for packets classification and marking, congestion management / avoidance and traffic policing /shaping.

2.1 Packet classification and marking

Classification comprises using a traffic descriptor to categorize a packet within a specific group, to assign that packet and make it accessible for QoS handling on the network. Once the packets are classified, they are marked to signal to core routers the QoS policy which must be applied to them. Cisco IOS (Internetworking Operating System) provide three main features for packets classification:

2.1.1 Policy Based Routing (PBR)

PBR [7] allows for the traffic classification based on Access Control List (ACL). ACLs establish the match criteria and define how packets have to be classified. ACLs classify packets based on port number, source and destination address (e.g., all traffic between two sites) or Mac address. PBR also provides a mechanism for setting the IP Precedence or DiffServ code point (DSCP) enabling the network to differentiate classes of service. IP Precedence employs the three precedence bits in the IP version 4 header's Type of Service (ToS) field to specify class of service for each packet [1]. Six classes of service may be assigned. The remaining two classes are re-

served for future use. The DSCP [3] replaces the ToS in IP version 6 and can be used to specify 1 of 64 classes for a packet. PBR finally provides a mechanism for routing packets through traffic-engineered paths. The Border Gateway Protocol (BGP) [14] is used to propagate policy information between routers. Policy propagation allows packet classification based on ACL's or router table source or destination address entry use with IP Precedence [7].

2.1.2 Committed Access Rate (CAR)

CAR [7] implements both classification and policing functions. Classification is done by the setting of the IP Precedence for packets entering the network. CAR provides the ability to classify and reclassify packets based on physical port, source or destination IP or MAC address, application port or IP protocol type, as specified in the ACL [1].

2.1.3 Network Based Application Recognition (NBAR)

NBAR [7] is a classification engine that recognizes a wide variety of applications, including web-based and other difficult-to-classify protocols that employ dynamic TCP/UDP port assignments. When an application is recognized and classified by NBAR, a network can invoke services for that specific application. Hence, NBAR adds intelligent network classification to network infrastructure.

2.2 Congestion management

Congestion management allows congestion control by determining the order in which packets are forwarded out of an interface based on priorities assigned to those packets. It entails the creation of queues, assignment of packets to the respective queues based on their classification, and scheduling of the packets in a single queue for transmission. Cisco IOS provides four queuing and scheduling schemes:

2.2.1 First In Fist Out (FIFO)

FIFO is the default queuing mechanism. There is only one queue and all packets are treated equally and served in a first come first serve fashion. FIFO is the fastest of Cisco's queuing and scheduling schemes.

2.2.2 Priority Queuing (PQ)

Cisco PQ [1] provides four queues with assigned priority: high, medium, normal, and low. Packets are classified to queues based on protocol, incoming interface, packet size, or ACL criteria. Scheduling is determined by absolute priority. All packets queued in a higher priority queue are transmitted before a lower priority queue is served. Normal priority is the default if no priority is assigned when packets are classified.

2.2.3 Weighted Fair Queuing (WFQ)

WFQ [8] offers dynamic scheduling algorithm that shares bandwidth among queues based on their weights. There are two main types of WFQ: Flow Based WFQ (FBWFQ) and Class-Based WFQ (CBWFQ). FBWFQ provides priority management grained by individual traffic flows. It breaks up the sequence of packets within a conversation to ensure that bandwidth is fairly shared among individual conversations and that low-volume traffic is transferred in a timely fashion. The traffic is classified into different flows, based on packet header fields, including characteristics such as source and destination network or MAC address, protocol, source and destination ports, session socket numbers of the session, Frame Relay datalink connection identifier (DLCI) value, and ToS value. CBWFQ extends the standard WFQ functionality to provide support for user-defined traffic classes. For CBWFQ, traffic classes are defined based on match criteria including protocols, Access Control Lists (ACLs), and input interfaces. Packets satisfying the match criteria for a class constitute the traffic for that class. A FIFO queue is reserved for each class, and traffic belonging to a class is forwarded to the queue assigned for that class. In order to characterize a class, on can assign bandwidth, weight, and queue limit. The bandwidth assigned to a class is the guaranteed bandwidth delivered to the class during congestion. The queue limit is the maximum number of packets allowed to accumulate in the queue for a class. After a queue has reached its configured queue limit, enqueuing of additional packets to the class enables the queue management (Tail Drop or WRED) depending on how the class policy is configured. CBWFQ finally provides a mechanism called Low Latency Queuing (LLQ). LLQ [8] enables the use of a single, strict priority queue within CBWFQ at the class level, allowing low jitter for delay-sensitive data.

2.2.4 Custom Queuing (CQ)

Cisco CQ [8] discipline reserves a percentage of an interface's available bandwidth for each selected traffic type. If a particular type of traffic is not using its available bandwidth, then another type may consume it. CQ is composed by 17 queues numbered from 0 to 16. Queue 0 is a system queue reserved for the mechanism and therefore cannot be set by the user. For each queue, the byte count parameter has to be fixed. The router forwards packets from a particular queue until the byte count is exceeded. Once the byte count value is exceeded, the packet that is currently being served will be completely sent. Therefore, if the byte count is set to 100 bytes and the packet size of your protocol is 1024 bytes, then every time this queue is served, 1024 bytes will be sent, not 100 bytes. The bandwidth that a custom queue will receive is given by the following formula:

$$BW_i = \frac{BC_i}{\sum BC} \times TBW$$

where BC_i stands for the byte count of for queue i and TBW means the bandwidth capacity for the ouput link minus the bandwidth for prioritary queues.

2.3 Congestion avoidance

Congestion avoidance techniques are used to monitor the network traffic loads in an effort to identify the initial stages of congestion and proactively avoid it. Cisco IOS provides two features for congestion avoidance: Tail Drop and Weighted Random Early Detection (WRED). Tail Drop [9] is Cisco's default congestion avoidance behavior. Tail Drop treats all traffic equally and does not differentiate classes of service. Queues are filled during periods of congestion. When the output queue is completely full, packets are dropped. This process continues until the congestion is eliminated. WRED is IOS' implementation of RED. WRED combines the features of the RED algorithm with IP Precedence to provide for preferential traffic handling for higher priority packets [9]. When the average queue size is above the minimum threshold, WRED starts dropping packets. WRED drops packets selectively based on IP Precedence. Packets with higher IP Precedence are less likely to be dropped than packets with a lower precedence. WRED treats non-IP traffic as precedence 0 making them the most

likely to be dropped. WRED can be configured to behave as RED by explicitly ignoring IP Precedence during WRED configuration [1].

2.4 Traffic Policing and Shaping

Cisco IOS QoS offers two kinds of traffic regulation mechanisms [10]: Policing that typically consists in dropping packets when a traffic contract is violate and shaping which role is to delay excess traffic when the data rate of the source is higher than expected. Policing is realized by employing the rate-limiting feature of Committed Access Rate (CAR). The parameters for CAR configuration are those of a token bucket completed with the action to apply if the incoming traffic is conform or exceeds the bucket filling rate. For shaping functionality, IOS provides two main components: the Generic Traffic Shaping (GTS), which shapes traffic by reducing outbound traffic flow to avoid congestion by means of a token bucket mechanism and Frame Relay Traffic Shaping which is a Frame Relay implementation of GTS.

3 DiffServ implementation in Cisco routers

The purpose of this section is to show how it is possible to combine some of the previous Cisco features for DiffServ implementation. Combining such features within a Cisco router can be easily perform thanks to the Cisco Modular QoS Command Line Interface (CLI) which is a kind of script language for rapid QoS configuration. Using Modular CLI, there are three steps [11] to perform for enabling services differentiation in the router:

3.1 Creating a traffic class

A traffic class specifies a set of match criteria to which an incoming and/or outgoing traffic must match to belong to that class. The classification is done based on the match criteria.

3.2 Creating a traffic policy

The next step to implement a Differentiated Service Domain using Cisco is to create traffic policies which are used:

- To collect Traffic Classes together in one object called Traffic Policy. It is more practical to associate a Traffic Policy with an object because in this way, it can be manipulate with flexibility.
- To apply to each Traffic Class component the PHB (Per-Hop Behavior) corresponding to it (bandwidth, queue limit, policing, shaping, congestion avoidance queue behaviour, etc.)
- To mark or re-mark packets belonging to a selected Traffic Class before they are forwarding to their next hop.

By default, scheduling among the traffic classes is done in a Class-Based WFQ manner. It is possible to enable LLQ for one of the traffic classes.

3.3 Attaching the traffic policy to an interface

Once that a traffic policy is built up, the last step consists in assigning it to one or more interfaces of the Cisco router in such a way that the definitions implemented in the object, i.e., the traffic policy, will be applied to traffic crossing the router through the selected interface(s).

4 The API architecture

In Cisco routers, the DiffServ Quality of Service policy is enabled only when congestion occurs. There is no resource allocation during non-congested periods. The available bandwidth is then used for all the incoming traffic. However, when congestion occurs, the QoS policy in charge of QoS management in the router cannot adapt itself to the unpredictable, bursty nature of network traffic. Moreover, QoS parameters turning for global efficient resources utilization and stable network behavior needs their adaptation in a real-time fashion. Since Cisco routers are closed proprietary systems, the DiffServ API provides an interface to control and configure DiffServ QoS features.

The API has to achieve the following goals:

- The continuous monitoring of the DiffServ queues;
- The adjustment of queuing and scheduling parameters for a possible automatic reconfiguration of the QoS features aiming a better optimization of network resources and effective end-to-end QoS.

4.1 DiffServ queues monitoring

Since Cisco routers are closed systems, it is impossible to add new modules on the IOS. Queues monitoring can only be done using the external methods offers by the router. Simple Network Management Protocol (SNMP) is a powerful protocol to retrieve data from Management Information Base (MIB) on network devices. Cisco provides a private QoS MIB called CISCO-CLASS-BASED-QOS-MIB [12]. This MIB allows read-only access to QoS configuration information and QoS statistics based on the modular QoS CLI. Statistics on the various queues are available in **cbQosQueueingStats** table. In order to achieve queues controlling of to evaluate the congestion risk on them, we focus our attention on the following objects of the table:

- **CbQosQueueingCurrentQDepth,** which represent the current depth of the queue.
- **CbQosQueueingMaxQDepth,** which represent the maximum depth of the queue.

Once the current queue depth is known, the API has to provide features for adjusting QoS parameters.

4.2 Configuration of the queuing and scheduling parameters

SNMP provides the ability to configure routers by means of its “snmp-set” command. But since the CISCO-CLASS-BASED-QOS-MIB is in read access only, the configuration of Cisco QoS features can uniquely be done in the Command Line Interface (CLI) mode. As seen previously, in a DiffServ context, only the following parameters are configurable when a policy is defined: bandwidth, queue limit for Tail Drop queues, max-threshold, min-threshold and probability denominator for WRED [11] queues. Cisco also provides a telnet server for entering CLI mode in a remote fashion.

4.3 The architecture

In the proposed API, the SNMP functions are realized thanks to the AdventNet SNMP API (ASA). The ASA [2] offers a comprehensive development toolkit for SNMP-based network management applications. AdventNet's SNMP stack comprises a set of Java SNMP library to build real-time applications for monitoring and tracking network elements that are reliable, scalable, and OS independent.

In addition to SNMP functionalities, a Telnet TCP module enables a telnet connection to the router and, consequently, to the CLI. The telnet connexion consists of the creation of a Java TCP Socket aiming the port 23 of the router. A parser is defined to translate the router responses. Figure 1 illustrates our DiffServ API architecture.

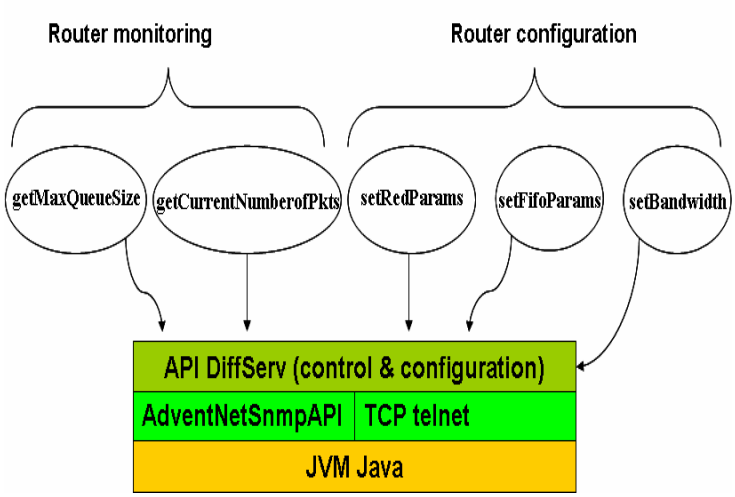

Fig. 1. API Architecture

Five methods are implemented:

- **getMaxQueueSize**: this method retrieves the maximum length of the traffic class specified in argument.
- **getCurrentQueueSize**: this method gets the queue's current depth of the traffic class specified in argument.
- **setRedParams**: it enables the assignment of WRED active queue management parameters.
- **setFifoParams**: this method enables the setting of the tail drop queue given in argument.
- **setBandwitdth**: this method is for the bandwidth setting of a traffic class.

Hence, using the API, it is possible to query the Cisco QoS MIB and get the current and the maximum queue size in terms of number of packets from a given traffic class. It is also possible to modify the allocated bandwidth for a class, to fix the values of a WRED queue parameters and the depth of a Drop Tail queue.

5 Experiments

A test has been carried out to check the proper working of Cisco DiffServ API. Figure 2 shows the main test characteristics and the scenario. The test bed is made up with two routers running in DiffServ mode: a Cisco 1751 router and a Linux based router. Cisco IOS version 12.2(8) T5 is installed on the Cisco router. All the interfaces used on the test-bed have 10Mbps link speed. The IOS allows the commitment of 7.5 Mbps from the 10 Mbps link capacity for user data. The rest of bandwidth is automatically allocated to network signalization and routing protocols exchanges.

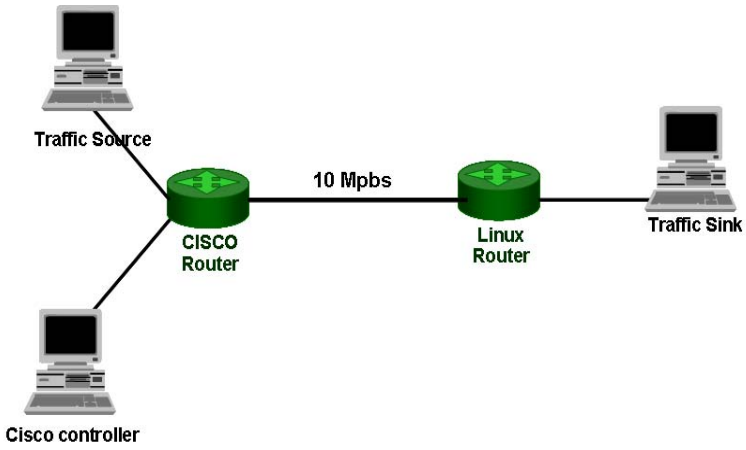

Fig. 2. Scenario Test

A policy named **diffserv** is defined with three traffic classes: a best effort traffic class with a DSCP equal to 0 attached to a Tail Drop queue, an Assured Forwarding (AF) traffic class with a DSCP equal to 10 (AF11) served by a WRED queue and an Expedited Forwarding (EF) traffic class with a DSCP equal to 46 also attached to a Tail Drop queue. The available bandwidth is allocated as follows:

Traffic classes	Bandwidth allocation
Class EF	3000 Kbps
Class AF	3000 Kbps
Class BE	1500 Kbps

JTG (Jugi's Traffic Generator) is used to generate traffic and the Cisco controller runs an application built using the API. The following rule defines the router behavior:

While EF queue_length is more than 10 packets
Do
BE_bandwidth = BE_bandwidth - 1Mbps
EF_bandwidth = EF_bandwidth + 1Mbps
BE_queueSize = BE_queueSize + 10 packets
Until BE_bandwidth = 1 Mbps

In period of congestion, when EF packets start to be delayed, the application decrements the BE traffic class bandwidth and increments the EF traffic class bandwidth. To avoid the dropping of the BE traffic, the buffer size is increased.

A 3 Mbps EF flow and a 4 Mbps AF11 flow are generated simultaneously by the traffic generator. Another TCP BE flow is sent to the Cisco router. Notice that the QoS policy in a Cisco router is applied only when the level 2 transmit queue is full. This queue is called transmit ring and it size is set by the parameter "tx-ring-limit". The transmission queue length is expressed in particles, i.e. in units of 512 bytes. This parameter was fixed to 1 in order to allow fast occurrence of the congestion.

Using the DiffServ API tools during the test, it was possible to read the number of packets in each queue, to modify the length of the queues and to dynamically reallocate the bandwidth for traffic classes. In a first stage of the experiments, we observe that implementing a simple rule on the top of the API allows rapid adjustment of the EF class bandwidth according to his queue length. This propitiates a reduction on the number of packets in the queue and consequently, the avoidance of packets dropping.

6 Conclusion and future works

In this paper, we present an API for the monitoring and configuring Cisco QoS features with the aim to provide the tools for an adaptive QoS management. Now that it is possible to interface external software with a router and modify its QoS parameters "on-the-fly", the next step for this work is to design an intelligent agent on top of this API, whose purpose will be to enhance network performance by allowing automatic adaptation on network devices according to traffic fluctuations. We also intend to extend the experiments by measuring the impacts of continuous router reconfiguration on the network stability and performance.

References

1. Adams K, Nguyen T Router Support for Quality of Service
2. Advent Net. AdventNet SNMP API online site http://snmp.adventnet.com/
3. Blake S, Black D, Carlson M, Davies E, Wang Z, Weiss W (1998) An Architecture for Differentiated Services. RFC 2475
4. Braden R, Clark D, Shenker S (1994) Integrated Services in the Internet architecture: An overview. RFC 1633
5. Braden R, Zhang L, Berson S, Herzog S, Jamin S (1997) Resource reSerVation Protocol RSVP. RFC 2205
6. Cisco Systems (2001) Cisco IOS 12 Documentation, Internetworking Technology Overview
7. Cisco Systems (2001) Cisco IOS 12 Documentation, Quality of Service Solutions Configuration Guide, Classification Overview
8. Cisco Systems (2001) Cisco IOS 12 Documentation, Quality of Service Solutions Configuration Guide, Congestion Management Overview
9. Cisco Systems. Cisco IOS 12 Documentation, Quality of Service Solutions Configuration Guide, Congestion Avoidance Overview, 2001
10. Cisco Systems (2001) Cisco IOS 12 Documentation, Quality of Service Solutions Configuration Guide, Policing and Shaping Overview
11. Cisco Systems (2001) Cisco IOS 12 Documentation, Quality of Service Solutions Configuration Guide, Quality of Service Solutions
12. Cisco Systems. Cisco Class-based MIB online description: ftp://ftp.cisco.com/pub/mibs/v2/CISCO-CLASS-BASED-QOS-MIB.my
13. Enterasys X-Pedition ER16 (2003) User Reference Manual
14. Lougheed K, Rekhter Y (1990) Border Gateway Protocol BGP. RFC 1163
15. Miguel Franklin de Castro (2004) Gestion programmable et adaptative de la qualité de service sur IP. INT Evry