Dynamic QoS Management in the Internet

Rares Serban

UNIVERSITÉ DE NICE SOPHIA-ANTIPOLIS -
UFR SCIENCES

École Doctorale STIC

THÈSE

Présentée pour obtenir le titre de :
Docteur en SCIENCES
de l'Université de Nice Sophia-Antipolis

Spécialité : Informatique

par

Rares SERBAN

Sujet de la thèse

La Gestion Dynamique de la Qualité de Service dans l'Internet

Thèse dirigée par Walid DABBOUS

Soutenue publiquement le 5 Septembre 2003 devant le jury composé de :

Président :	Michel RIVEILL	UNSA
Rapporteurs :	Dominique GAÏTI	Université de Technologie de Troyes
	Nazim AGOULMINE	Université d'Evry
Examinateurs :	Francine KRIEF	Université Paris 13
	Hossam AFIFI	INT Evry
Directeur de thèse :	Walid DABBOUS	INRIA Sophia-Antipolis

(10:00 - INRIA Sophia-Antipolis)

To my parents

Contents

List of Figures

List of Tables

Remerciements

Je voudrais tout d'abord remercier M. Nazim Agoulmine, Professeur à l'Université d'Evry, et Mme. Dominique Gaïti, Professeur à l'Université de Technologie de Troyes, qui m'ont fait l'honneur d'avoir accepté d'être les rapporteurs de ma thèse.

Je tiens tout d'abord à remercier M. Michel Riveill, Professeur à l'ESSI, qui ma fait l'honneur de présider mon jury de thèse.

J'exprime ma profonde gratitude a M. Walid Dabbous, Directeur de Recherche et chef du Projet Planète à l'INRIA Sophia-Antipolis, pour m'avoir proposé ce sujet, pour son assistance lors de la réalisation de ce travail, pour sa disponibilité qui m'a permis de mener à terme cette thèse. La liberté qu'il m'a accordé et les responsabilités qu'il ma confiées ont beaucoup contribué à la formation de ma personnalité et à mon autonomie dans le travail.

Je remercie à mon ami Chadi Barakat qui avec sa patience et sa personalité m'a aidé sur le chemin de la recherche.

Mes remerciements les plus vifs aussi à M. Hossam Afifi, Professeur à l'INT Evry et à Mme. Francine Krief, Maitre de Conférence à Paris 13, qui se sont intéressés à mes travaux et ont bien voulu participer à mon jury de thèse.

Mes remerciements vont à tous les membres du projet Planète pour leur soutien permanent et leur ambiance de travail.

J'adresse également mes chaleureux remerciements à ma famille pour la patience dont ils ont fait preuve à mon égard et pour leur aide et leurs encouragements.

Enfin, j'exprime ma grande reconnaissance à tous mes amis qui maintenant sont dispersés partout dans le monde.

Chapter 1

Introduction

In the previous decade an impressive growth had been produced in the use of communication networks. To initiate and optimize the operations of these networks, good network management facilities must be developed. In literature several definitions of the network management exist [1][2][3][4]. Most of these definitions are produced by standardization organizations, which use specific terminology and aim their definitions at specific fields of application. To summarize all definitions, *network management* is the act of initializing, monitoring and modifying the resources of a telecommunications network in order to meet service-level objectives at all times, at a reasonable cost and with optimum capacity. The network service-level objectives include network performance, support levels, metrics definition, and administrative plans. The importance of research in this area is confirmed by a number of studies that show different issues of network management framework. The network management framework employs billing and accounting issues, security issues, quality of service management issues, policy management issues and organizational management issues.

The *quality of service management* is one of the network management framework issues [5]. In this thesis the term *quality of service* is defined by the ability of a network element (e.g. an application, a host or router) to have some level of assurance that its traffic and service requirements can be satisfied. The cooperation of all layers of the network from top-to-bottom, as well as every network element from end-to-end is required to enable quality of service. The quality of service assurance is given by the management acts (e.g., monitoring, provisioning, etc.). The term *quality of service management* is used in this thesis to denote the act of assuring some levels of traffic or service requirements.

Traditionally, the Internet has offered a single level of service known as *best-effort* service. In a pure best-effort model, the resulting quality of service

depends on the current traffic load over the links involved. The fundamental principle for IP was derived from 'end-to-end' argument [6], which puts 'intelligence' in the ends of the network (the source and destination network hosts) leaving the network "core" dumb. IP routers at intersections throughout the network need do little more than check the destination IP address against a forwarding table to determine the next hop for an IP datagram. If the queue for the next hop is long, the datagram may be delayed. If the queue is full or unavailable, an IP router is allowed to drop datagram. The result is that IP provides a *best-effort* service that is subject to unpredictable delays and data loss. The paradigm *Internet to everyone* increase the variability in delivery delays and packet loss. Internet applications like E-Mail, File Transfer Protocol, Web applications are not affected but other applications with real-time requirements, such as those that deliver multimedia or the most demanding of which are two-way applications like telephony are sensitive of the traffic variations. Many information technologies and commerce applications have already moved on IP, and others are 'en route' (the paradigm: *Everything over IP*). Most applications used by different users are low-priority and low-bandwidth, with high tolerance for delay. Other users with special applications have strict operational requirements and need high-priority and bandwidth. Hierarchical organizations inside the companies request different prioritization among their users. Unfair distribution of the traffic inside the network provides unequal satisfaction among the users or applications. Due the heterogeneity of the devices, users are using different quantity of traffic to and from the network.

Long time ago the IETF engineers have believed in *IP: necessary and sufficient* that can be respected assuming that the network's bandwidth capacity is sufficient to avoid any delays or dropped datagrams. Increasing bandwidth is a necessary first step for accommodating time-critical or real-time applications, but it is still not enough to avoid jitter during the traffic bursts. Even on a relatively unloaded IP network, delays can vary enough to continue to adversely affect real-time applications. To provide adequate network service (some level of quantitative or qualitative determinism) IP services must be supplemented. This requires adding some 'intelligence' to the network to distinguish traffic with strict timing requirements from those that can not tolerate delay, jitter and packet loss. With other words, IP networks need a management to assure some levels of bandwidth, delay, jitter and packet loss. This performance constraint of network depends on the economical model.

Actually, the Internet provides one network service type: *best-effort*. The management of such network service is based on an economical model: zero-priced access [7]. This model has the well-known defect of the problem of

commons. If each user faces a zero price for access, the network resources tend to become congested. So, the role of the management in this case is to avoid the congestion and to guard the reliability of the network. A solution which is not sufficient proposed by many Internet Service Providers (ISP) is to add access levels in the network by offering bandwidth levels (e.g., Gold, Silver Bronze pricing packages). The economical model is using *best-effort* with posted prices. A fixed price schedule for different guaranteed bandwidth at different times. The trouble with posted prices is that they are generally not sufficiently flexible to indicate the actual state of the network at a particular time. The management role in this case is to avoid the congestion, to watch the reliability of the network and to do accounting and billing. The quality of service component of the management framework in both cases has the role to avoid the packet losses (cause by congestion).

In the QoS two aspects influence the management of a network service: the economical viability and the management of data network infrastructures. A number of QoS architectures (e.g., Diffserv[8], Intserv[9], MPLS[10]), QoS signaling protocols (e.g.,RSVP[11], COPS[12], Boomerang[13], BGRP[14], etc.), QoS algorithms (e.g.,admission control[15], resource allocation[16], scheduling[17], etc.) are not competitive or mutually exclusive, but no contrary, they are complementary. For the management of data network infrastructures aspect, they are designed for use on combination to accommodate the varying operational requirements (requested through the economical aspect) in different network context.

The economical model of the network service is implemented by two main components: the service level agreement (SLA)[18] and the quality of service management (QoSM) network framework. The design of the future applications involved a set of service definition into the network service interface. The SLA includes all subscribe services that the network will provide for a given client. Each subscribe network service contains service level specifications (SLS)[19] which must be translated by the QoSM framework to the network. Actually, several services were proposed by a number of research studies [20][21] and standardization organizations [22][23]. The quality of service component of the management framework became sophisticated. The quality of service management must employ appropriately scheduling mechanisms, queuing management mechanisms, business goals and objectives for network policies and interfaces with other management components from the system. The efficiency of QoSM characterizes the way as how and when those components are enforced into the network by the network administrator or by the user through special mechanisms. The efficiency of QoSM is to maximize the gain of the ISP and to minimize the unsatisfaction of the network users.

The important role of the QoSM is to preserve all the features set by Service Level Agreement (SLA). Today, the Internet Service Providers (ISP) are providing the SLA using a Static QoS Management. Static QoS Management (SQM) is characterized by long periods of time when the network resources properties or requirements of network customers remain constant through their activities. Resource allocation operation is performed in large periods of time following the new SLA requests. Tuning operations or provisioning operations, actually, are characterized by extensive manual work, based on a trial-and-error process. When requirements of network customers vary in small periods of time, the network resources cannot follow their behavior and the performances of customer's applications will be poor. When using SQM the network resources usage and flow characteristics should be respected as they were defined in SLA but in reality for different reasons this does not happened. There are different reasons why the QoS parameters are not respected and they may have to be renegotiated. Two examples of such violation of parameters are: new unexpected traffic users and changing users needs. The network service provider should perform manual tuning operations each time when a new customer/user uses his network resources. This operation is time consuming and costly. Speaking in general, the number of users using his network is varying in time so adjusting manually the network parameters in order to preserve the SLAs becomes a difficult and complex task. The SLA is not respected if the network service availability has changed (e.g., due to congestion, due failure devices). Some users consider that their SLA contract specification generates insatisfaction for them. So, they will try to change it, according with their new requirements. The contract negotiation between the customer and the service network provider performs manually operations and the effect of the changes is not immediately. The network service provider needs to tune the network parameters to preserve the others SLAs and to satisfy the new one. Without this operation, the parameters of the network and SLSs (Service Level Service) from the SLAs are violated. As we can observe, in the both cases SQM is not able to cope with slot time scale and with the granularity control of the traffic dynamics.

To provide the appropriate QoS management based on each moment of time on the actual state of the network there is proposed Dynamic Management of the QoS (DQM) for Internet. By definition, Dynamic Quality of Service Management (DQM) provides automated and adaptive mechanisms for maintenance, provisioning, control and trade of network resources (memory buffer, link capacity, CPU).

Dynamic QoS Management is not any one technology, but a combination of different complementary components that comes together to provide for different customers and different applications specific network performance

guarantees that allow then to get their jobs done efficiently and quickly. In this thesis there are considered the following components of Dynamic Quality of Service Management:

- Signaling;
- Monitoring tools;
- Trade mechanisms between two domains;
- Mechanisms for admission control and resource allocation including network policies.

Signaling is used to transport different measured parameters (monitoring) or to spread configurations for resources control during large periods of time (admission control, resource allocation). The design of a signalization system must take care of synchronization, reliability of the network infrastructure, reaction speed [24].

The goal for a monitoring system is to watch the incoming and internal traffic flows for assuring a certain profile of service. The monitoring system follows the state of the network resources used in the admission control operation. Also, it checks the state of the network resources after resource allocation operation. The characteristics of a monitoring system must be the precision and the granularity of the measures.

The trade mechanisms assure for the transit flows the correct resources within the transit domains. In function of different policies of control of each domain there are different types of negotiation [25][26]. Simplicity is the design goal of such a system. This component is out of scope of this thesis.

Mechanisms like resource allocation and admission control should respect network Service Level Specification management. The QoSM design issue is to put in correspondence the management rules with flow characteristics in order to accept or not flows into the network. The goal is to find a good solution, modifying the network parameters in order to provide an efficient usage of the network and to avoid congestion. This correspondence depends on network parameters varying in function of the day hours, in function of the user mobility or in function of the security policy. The resource management is based on admission control and resource allocation mechanisms with respect to the network management policy rules.

To help understand these components and their associated complexity and cost, we consider several important aspects of DQM. There are two key aspects that define a broad design space from which DQM can be built, reflecting various trade-off in QoS network service performance, operations

and management complexity and implementation cost. A key aspect of DQM is the time scale at which a control mechanism operates (fig.1.1). The fastest time scale is the packet level (100us-1ms), which is the smallest unit a network can exert control. QoS control mechanisms like traffic conditioners, packet schedulers, and active queue management are working at this time scale. The next control mechanism following the packet level on the time scale is the flow control (here is included different type of signalization)(1-100ms). Slower than packet time and flow control is the dynamic provisioning and admission control. This is the time scale user sessions. The next time scale is routing and rerouting-based on traffic engineering. Beyond this time scale, a variety of *long-term* QoS control mechanisms operate at time scales ranging from days to weeks or month.

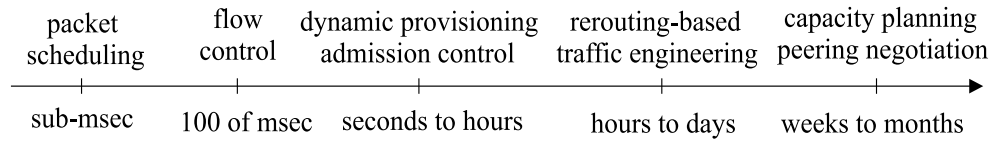


Figure 1.1: Time Scale for DQM Components

For a good performance of the system the main issue is the synchronization between all the management components on the time scale.

Another key aspect of DQM is the space resource dimension. The space resource dimension is composed by the aspects of granularity of control information (i.e., control state), carrier of control state and location of control. The finest granularity of control is per-flow state information and the coarse-grain granularity is per flow aggregation. The two types of granularity are used by the components of DQM like resource allocation, admission control, monitoring, resource trading. The granularity of control varies depending on the level of flow aggregation (e.g., per host, per service class, etc.). The carrier control state defines how the control state informations are transported inside the network (e.g., signaling protocols, packet headers). The control state is stored whether in the routers or in the packet header only. Closely related to the carrier of the control state and control state is another important aspect of DQM - the location of control aspect of the network. In fact is where a control mechanism operates, whether at the end-hosts, the network edges or boundaries between either users and network or network domains, or inside the network core. The control mechanisms can be distributed or centralized.

Both the time and space dimensions of DQM have enormous implication in the design, operations and management of the network resources. Hence to design a scalable and simple DQM architecture, it is imperative to make

judicious design choices along the time and space dimensions and carefully evaluate their trade-off and implications in both network data and control plane.

Thus, a design aspect for a DQM framework represents the scalability. The QoS network management must not be disturbed by the increasing number of "actors". To maintain performance guaranteed by SLA a DQM system should have a degree of QoS adaptation that can be tolerated and scaling actions to be taken in the event the contracted QoS cannot be met. The scalability of a QoSM system can be accepted if a certain level of degradation of QoS is acceptable.

The complexity of a DQM framework depends on the components and networking solutions used in the system. A network service model involves a certain complexity of the QoSM. DQM gains in flexibility by using automated or adaptive mechanisms but it becomes very complex. For example, a DQM architecture that employs per-flow QoS control and stores QoS state at every router requires a signaling and a monitoring protocols that conveys QoS states to every router on per-flow basis. Such architecture involves a sophisticated control plane at every router, fine tuning of signaling protocol, sensitive monitoring operations and thus increasing the complexity and limiting the scalability.

This dissertation presents the motivation of the high level goals, the detailed description of the QoS management, and the Dynamic QoS Management framework. The following section summarizes the key contributions of this thesis.

1.1 Thesis Contribution

Dynamic QoS Management involves various system components from the network and generates several issues. Regarding the issues addressed in this thesis, we were able to make a number of contributions, which we summarize here and discuss in more detail in the following chapters.

- **An Analysis of QoS Signaling Protocols used in Internet:** The performance of each network service depends on the signaling system. Concerning the QoS signaling protocols for Internet, we have studied their behavior over a test-bed network. We proposed a specific criteria classification of QoS signaling protocols. We then developed an analyze based on our criteria classification. We believe that using our analyze, the QoS signaling protocols will be choose in the way to respond to the applicability requirements of a SLA. Tuning parameters of a QoS signaling protocol will enhanced the DQM performance.

- **Efficient Correspondence between the Service Level Parameters and the Network Configurable Parameters:**The SLA parameters agreed with the customers should be map with the network parameters inside of each network element. We have evaluated and identified the network parameters that can be tuned to respect the SLA of the customers using a test-bed network. Efficient policy rules to protect the SLA were developed based on tuning parameters.

- **Static QoS Management Limitations:**Using scenarios we proved that the Static QoS Management has its limitations. We showed that SQM requires many tuning operations and is time expensive. We proposed the necessity to develop automated and adaptive mechanisms for QoS management.

- **Dynamic Resource Allocation Mechanism for Core Routers:** We have designed, developed and validated a simple dynamic resource allocation mechanism for core routers. Our mechanism is easy to implement, and does not require any particular signaling. It ensures that SLAs are respected, and allows at the same time an efficient utilization of network resources.

1.2 Dissertation Outline

The remainder of this thesis is organized as follows. Chapter 2 reviews the work related to this dissertation. We first discuss about the network services designed in Internet. Then, we survey the main QoS frameworks from Dynamic QoS Management perspective. We describe the challenges for Dynamic QoS Management, which are the research problems addressed in this thesis.

In Chapter 3 we investigate the challenge of using QoS signalization protocols for management. We consider that the performances of a QoS signaling protocol depend on the tuning signaling parameters, network element (router/host) architecture, and the characteristics of SLA (type of network services, policy network, etc.). We propose a criteria classification for QoS signalization protocols and we prove the utility of using it.

In Chapter 4 we focus on the impact of tuning network parameters in order to preserve the SLAs. We consider a test-bed network that has implemented packet level mechanisms like schedulers, and queuing management mechanisms. We propose a set of network parameters that have an important impact of service level parameters (delay, bandwidth, jitter, packet loss) for each packet level mechanism. Using the test-bed and the knowledge of

the mapping parameters we prove the necessity of modifying the low-level parameters in order to respect service level specifications.

In Chapter 5 we design a dynamic allocation mechanism for network core routers among Diffserv services. We define a network service and we evaluate the mechanism implemented in our test-bed network. We prove using several scenarios that SLAs are respected using our mechanism. Also, we prove that our mechanism allows an efficient utilization of network resources (reflected by the service level parameters delay and bandwidth). Our proposed mechanism is using one of the schedulers studied in chapter 4 (CBQ). We study the impact of between CBQ and our mechanism.

In Chapter 6, we present a general summary of the work achieved and the conclusions concerning the results obtained during this thesis. Some perspectives and open questions are given for the continuation of this work in the area of QoS management.

Chapter 2

The State of the Art in QoS Management

There has been a wide spectrum of work during the last decade on Quality of Service in packet switched networks. The following QoS available in Internet are complementary or are mixing different technologies and are designed for use in combination for different contexts. This chapter only presents a survey of related trends and issues, which serves as background to our research work.

We provide a brief overview of the main QoS framework models for Internet from the Dynamic QoS Management perspective (DQM defined in the first chapter). In section 2.1 we summarize the Network QoS Management. Section 2.2 presents a brief overview of QoS services employed in the networks. QoS network services are important in the negotiation part of the DQM. We review the main QoS network services. Section 2.3 provides a brief overview of Integrated Services Architecture defined by IETF [9]. In Section 2.4 we overview a potential solution given by IETF using all the QoS features developed, named MPLS (Multiple Protocol Label Switch) [10]. The Differentiated Services Framework [8] overview is presented in Section 2.5. In Section 2.6 we identify the issues related to this area on which we focus in the rest of this thesis. Section 2.7 summarizes this chapter.

2.1 Network QoS Management

The problem of Internet QoS management has been an extremely active area of research for many years. This section presents and survey the quality of service management framework (QoSM). A set of modules utilized in building DQM is described. These modules include: *QoS specification* which captures application level quality of service requirements and *QoS mechanisms* which

realize the desired end-to-end/edge-to-edge QoS behavior.

2.1.1 QoS specification

The end-to-end or the edge-to-edge QoS applications behavior is done by using QoS mechanisms where are following the QoS specifications. The QoS service specification is defined by SLA (Service Level Agreement) and includes SLS (Service Level Specifications) and all business contractual conditions [24]. Service Level Specification elements specify *what* is required rather than *how* is to be achieved by underlying QoS mechanisms. Encompasses but is not limited to the following:

-*flow performance specification*, which characterizes the user's flow performance requirements;

-*service commitment level*, which specifies the degree of the end-to-end resource commitment required;

-*QoS management policy*, which captures the degree of QoS adaptation that the flow can tolerate and the scaling actions to be taken in the event of violations in the contracted QoS;

-*cost of service*, which specifies the price of the user which is willing to incur for the level of service. Cost of service is a very important factor when considering QoS specification.

Flow performance specification describes the requirements of a flow based on the service level parameters [27]. The ability to guarantee traffic throughput rates, delay, jitter and loss packets is important for real-time applications. These requirements vary from one application to another. To be able to commit necessary network resources a QoS architecture should have prior knowledge of the expected traffic characteristics associated with each flow and resource guarantees made. This resource commitment could be deterministic [28], predictive [29], adaptive [30], automated [31] and best effort. According to the topic of this dissertation we are interested in the last two resource commitments. Andrew Campbell et al. [30] specify two adaptations for resource commitments. For discrete adaptation, the resource commitments can be provided by the scalable profiles of the MPEG-2 standard. For the continuous adaptation, the resource commitments are provided using dynamic rate shaping algorithms for compressed digital video flows and using weighted faire share mechanism. R. R-F Liao et al. [31] provide automated specification using a utility generator. The utility generator interacts with a network adaptation mechanism to periodically renegotiate bandwidth to match the bandwidth requested by the generated utility curves.

The flow performance specification is expressed in function of the levels of commitment service specification.

There exist several **service commitment level**. The definition of the service commitment level includes all the service semantics (descriptions how packets should be treated within the network, how the application should inject the traffic into the network as well as how the service should be policed). For *end-to-end communications* the QoS specification is committed on the application level. For example many continuos media applications can tolerate small variations in the QoS delivered by the network without any major disruption to the user's perceived service. In some cases (e.g., device failure, rerouting, severe congestion) fluctuation can be accommodated. The specification of the service will take account the application multi-layer mechanisms (e.g., MPEG codecs, voice codecs). For *edge-to-edge communications* the service-level specification is committed on the network level. For example flow aggregation supports many application commitments that needs to be respected. The aggregate flow performance has his specifications that should be able to support all individual users service-level specifications [32].

QoS scaling policy establish administrative rules that dictates the behavior of the QoS mechanisms under different situations. For example, in the case of an unforeseen scarcity of resources, the application/user may be willing to go through a renegotiation and accept a lower quality of service instead of being denied the service. There are no quantifiable metrics that describe these specifications, but the policies can be classified into different classes of scaling (negotiation) functions.

Cost of service specifies the price the user is willing to inquire for the level of the service; cost service is an important factor when considering QoS specification. Several pricing schemes are proposed in the Internet today [33]: access-rate-dependent charge (AC), volume dependent charge (V) or the combination of the both. Different propositions of adaptable and/or automated pricing schemes were proposed for QoS in Internet. In [34] is presented a dynamic congestion sensitive pricing scheme. The demand behavior in this case of adaptive users is based on a physically reasonable user utility function [35]. Dynamic market-pricing of edge-allocated bandwidth proposed in [36] introduces a game-theoretical model that determines the sustainability of any set of service level agreement (SLA) configurations between users and ISPs and between ISPs.

Technical and commercial aspects of QoS specifications are based on the service level specification elements. The technical aspect is characterized by the flow performance and service commitment level specifications. QoS scaling policy specification involves both aspects. Commercial aspect is provided

by the cost of service specification.

We summarize with an example the QoS specification. A continuous media application flow needs a guaranteed bandwidth, a bounded delay and an amount of packet loss. The commitment level of service is made for end-to-end with no administrative rules for QoS scaling policy. The cost of service is depended of the transmission rate. The network resources should be managed in order to achieve those commitments. The network resources are characterized by the QoS mechanisms.

2.1.2 QoS mechanisms

QoS mechanism are selected and configured according SLSs. QoS mechanism which release the desire end-to-end/edge-to-edge QoS behavior are categorized in: *QoS provision mechanism*, *QoS control mechanism*, *QoS monitoring*, and *QoS trading*. Each component of QoS mechanisms operate on different time scale and in different place in the space dimension of QoS (control) management.

a) QoS provisioning mechanisms include three components:

- **Translation** between representations of QoS at different system levels and the terms of the lower level of specification. The mapping function is doing an automatic translation between representations of QoS flow performance specification terms and low level specification terms.

- **Admission control** which is responsible for comparing the resource requirement arising from the requested QoS against the available resources in the system. The decision whether a new request can be accommodated generally depends on system-wide resource management policies and resource availability. A number of different admission control schemes have been proposed. The simplest is the *rate sum*. It ensures that the sum of existing reservations and the new flow's rate does not exceed a threshold. This scheme does not make any assumptions about the source behavior or aggregate traffic arrival. Guerin et al. introduced the approach of using equivalent capacity of aggregated traffic for admissions control in [37]. Floyd further developed a variant of the approach in [38]. The two schemes differ in their traffic assumptions, estimation method for equivalent capacity, and use of network measurements. Another admission scheme is given by Tse and Grossglauser [39]. They use the concept of acceptance region. The algorithm decides whether to admit a new flow based on the current state of the system and whether the state lies within the 'acceptance' region or the

'rejection' region. A measurement-based scheme with delay and bandwidth constraints is proposed in [40]. If the admission control scheme knows what kind of applications are generating the traffic flows, it can make assumptions about the QoS bounds of the flows and thus optimize its own performances. Holmes and Court present a simulation studying two admission control schemes to improve performance for both HTTP and FTP in [41]. Liao et al. [42] propose automated and adaptive admission scheme. The scheme comprises an application-specific adaptation mechanism which exploiting the video/audio content and user preferences.

- **Resource reservation protocols**, which arrange for the allocation of suitable end-system and network resources according to the SLA. In doing so, the resource reservation protocol interacts with the QoS-based routing to establish a path through the network in the first instance, then, based on QoS translation specification-low level terms, and admission control at each network element traversed end-to-end resources are allocated and committed. The QoS control mechanisms are configured appropriately. Two different approaches have been designed for resource reservation protocol: signaling protocol and packet state. The packet state is where control state information necessary for packet scheduling is carried in packet headers. An attractive approach is the dynamic packet state proposed by Stoica and Zhang [43][44][45], where control state information is carried in data packets and updated at core routers for scheduling purposes. Another approach was developed by Almesberger et al [46]. The packets are carrying in-band messages (flags) to signal resource reservation the intention from sources. It requires routers active participation in admission control process. The signaling approach in general requires a signaling protocol that conveys resource reservation to core routers. RSVP [11] is designed to convey per-flow resource reservation information to core routers. RSVP needs to perform per-flow resource reservation management, thus limiting scalability of the QoS management framework. Other protocols like YESSIR [47], Beagle[48], Boomerang[13] are proposed to address several scalability associated with RSVP. COPS is proposed to transport policy information inside the QoSM framework. In Chapter 3 we present in detail the main QoS signaling protocols in Internet. However, in this section we gave a general overview of the QoS signaling.

b) QoS control mechanism operate on time scales at or close to media transfer speeds. They provide real-time traffic control of flows based on

requested levels of QoS established during the QoS provision phase. The fundamental QoS control mechanisms include the following:

- *flow scheduling*, which manages the forwarding flows in the end-system and network in an integrated manner. Flows are generally scheduled independently in the end-systems, but may be aggregated and scheduled together in the network. This is dependent of the level of service and the scheduling discipline adopted.

- *flow shaping*, which regulates flows based on user-supplied flow performance specifications. Flow shaping can be based on a fixed-rate throughput(i.e., peak rate) or some form of statistical representation(i.e., sustainable rate and burstiness) of the required bandwidth. The benefit of shaping traffic is that it allows the QoS frameworks to commit sufficient end-to-end (edge-to-edge) resources and to configure flow schedulers to regulate traffic through the end-systems and network.

- *flow control*, which includes both open-loop and closed-loop schemes. Open-loop flow control allows the sender to inject data into the network at the agreed levels, given that resources have been allocated in advance. Closed-loop flow control requires the sender to adjust its rate based on feedback from the receiver or network. Applications using closed-loop flow control based protocols must be able to adapt to fluctuations in the available resources. On the other hand, applications which cannot adjust to changes in the delivered QoS are more suited to open-loop schemes, where bandwidth, delay and loss can be deterministically guaranteed for the duration of the session

- *flow policing* which applies administrative rules set by the network administrators. Policing is often only appropriate where administrative and charging boundaries are being crossed, for example at a user-to-network interface.

c) QoS monitoring allows each level of system to track the ongoing QoS levels achieved by the lower layer. QoS monitoring often plays an integral part in a QoSM framework feedback loop that maintains the QoS achieved by resources modules. Monitoring algorithms operate over different time scales. Measured statistics can be used to control packet scheduling and admission control. The expected performance parameters are compared with the monitored QoS parameters. Then extern tuning operations (i.e., fine- or coarse-grain resource adjustments) on resource modules will sustain the delivered QoS. Fine-grain resource adjustment counters QoS degradation

by adjusting local resource modules (e.g., loss via the buffer management, throughput via the flow regulation, and queuing delays and continuous-media palyout calculation via the flow scheduling).

d) QoS trading contains rules and procedures to trade between two or more end-to-end (edge-to-edge) systems. Two issues are identified: QoS signals should preserve the semantics of the originating customer and QoS resource reservations should be honored in the end-to-end network path. Many continuous-media applications exhibit robustness in adapting to fluctuation in end-to-end systems. Based on the user-supplied QoS management policy, QoS trading between two QoS domains can take remedial actions to scale flows appropriately. A working trading system based on market principles and purely local decisions was proposed by G. Fankhauser et al. [49]. Other trading approach based on the aggregate utility function proposed by Liao et al [50]. Both approaches are using adaptive and automated resource negotiation mechanisms.

The QoS model framework can be categorized as either distributed or centralized model of management. Centralized model of management requires a network entity that controls, provision and monitor all the elements of the network. The common framework QoS management needs has been based on a centralized approach with a manager agent architecture. The complexity of managing QoS in the network increases dramatically as the number of services and the number and the complexity of the devices in the network increases. This approach has not proved to scale well or allowed the flexibility in the network with QoS services. Methods of control must be found that work and scale at the same speeds as that of the control plane of the network itself if a major concern of the QoS management system with the dynamic control traffic in a network. Increasingly it is a requirement that customers at the edge of the network be able to have access to management functionality. A centralized management approach may not provide the most convenient architecture to allow this capability. Frameworks based on a distributed model of management have gained momentum in recent year's [51]. A decentralized framework may have advantages with regards to scaling and speed of operation, but information and state management becomes complex in this type of model, resulting in additional complication in developing such systems.

As mentioned in Chapter 1, in the second part of this thesis (from Chapter 4 to Chapter 5), we focus on QoS control mechanisms and QoS provision mechanisms that are included in the topic presented above.

The aim of QoS Management framework is to provide network services satisfying the customer's demands. The issues that address the network services (heterogeneous equipment's, scalability, economical market demands,

deterministic or statistical QoS mechanism settings, etc.) require a short review of the main network services in Internet. In function of the type of the network service adaptable and automated control mechanisms are deployed. The next section presents the network services developed for several QoS architectures.

2.2 Network Services

Internet Service Providers handle with heterogeneous equipments from multiple vendors, each supporting different QoS mechanisms and its own configuration methods. This makes the problem become increasingly complex when a network service other than *best-effort* should be implemented. For that, today, Internet Service Providers try to develop their network services in function of their network configuration and customers applications. We searched to among the ISPs the possibilities to provide other network service than *best-effort* in order to determine whether services are interested in the QoS world. Abilene network (an American ISP) provides a simple network service named *Never Drop* packets (ND) [52]. A ND packet will never be dropped, but may experience queuing delay and jitter. The network service assumption in this case is that Abilene network has all time over-provisioned the bandwidth. Another solution proposed by [52] is a service called *worst than best-effort*. In fact, it is an additional class of the best effort-service. A small amount of network capacity is allocated (in a non-rigid way) for this service when the default best-effort capacity is under utilized. vBNS (Very High Performance Backbone Network Service)[53] delivers advanced IP services using reservation bandwidth on demand. The scope of this service is to assure a bandwidth guarantee edge to edge or end-to-end. There are no delay constraints for this network service. Another example of network service using bandwidth reservation is VideoNet [54]. VideoNet is using video on demand applications which request bandwidth reservation in function of the number of the video service customers. Quantum (QUAlity Network Technology for User-oriented Multimedia)[55] and Garr-it2 provide VLL (Virtual Leased Line) service across a pan-European Network of very high speed. The aim of the VLL service, also known in literature Premium service, is to emulate a point to point connection at the IP layer. The VLL service requirements are minimum delay and jitter for a guaranteed amount of bandwidth for edge to edge communication.

The brief review of the network services actually deployed in the ISP networks allowed us to find four classes of services. We have observed that not all class of services can be supported by some ISPs. The QoS network

services can be categorized in the following class of services:

- **Guaranteed service** provides firm (mathematically provable) bound on end-to-end datagram delay. This service makes it possible to provide a service that guarantees both delay and bandwidth. The datagrams arrive within the guaranteed delivery time and they are not discarded due to queue overflows. That means the flow's traffic stays within its specified traffic parameters (conforming traffic/packets). This service is intended for applications that need a firm guarantee that a datagram will arrive no later than a certain delay after it was transmitted by its source. This service does not attempt to minimize the jitter (the difference between the minimal and maximal datagram delays); it merely controls the maximal queuing delay.

  In Chapter 5 we will use the notion of *conforming packets*(IN profile packets) and *non-conforming packets*(OUT profile packets) for our network service model.

- **Predictive service** provides traffic levels characterized by different advertised delay bounds. Thus, the network seeks to forward the traffic without loss but drops any traffic not forwarded within the advertised delay bounds. This service is designed for playback applications that desire a reserved rate with low packet loss and a maximum bound on end-to-end packet delay, and that are tolerant of occasional dropped or delay packets. The efficiency gain of predictive service comes from allowing more flows into the network than guaranteed service, thus providing more sharing and lower cost.

- **Controlled-load service** provides approximately the same quality of service under heavy loads as good as *best-effort* service under light loads [56]. The controlled-load service may be used by any application that can make use of best-effort service, but is best suited to those applications that can usefully characterize their traffic requirements. Applications based on the transport of continuous media data, such as digitized audio or video, is an important example of this class. This service doesn't provide any quantitative information about the delays that can affect the flow. The percentage of packets not successfully delivered must closely approximately the basic packet error rate of the transmission medium.

- The definition of **Worst than best-effort service** is the following: the unused resources by the best-effort are allocated in a non-rigid way to this service. Applications that are relatively tolerant of greater loss,

delay, and jitter may mark their traffic for *worst than best effort* and receive a level of service that is potentially degrades compared to the default *best-effort* service [57]. Packets receiving *best-effort* treatment preempt packets of the *worst than best effort* down to a certain share. This share must be considerably lower that of the *best-effort* service, but should not be equal to zero in order to retain a low portion of *worst than best effort* traffic even when *best-effort* traffic takes up all available residual bandwidth.

We mentioned *Controlled-load service* and *Worst than best effort* network services for summarize in this thesis the main network services developed by Internet Service Providers.

In the following section, we survey the main QoS management frameworks in Internet using the network QoS management elements presented above. Several QoS management framework approaches are presented. QoS management approach for IntServ requires resource reservations in each node on the transmission path for all end-to-end communications. Thus is limiting the scalability and increase the complexity in each network element. QoS management of IntServ enhances the *best-effort* service to provide a guaranteed support for time constraint applications. Comparing with IntServ, QoS management of MPLS is closed to telephony approach. Circuits with class of service are provided between edge-to-egde network. Thus is increasing the scalability and pushing the complexity to the edge network elements (routers). QoS management for Diffserv is designed to support aggregate scheduling for class of services. This approach is more scalable and less complex than IntServ and MPLS and provides a variety of more resource management and provisioning scheme. In this survey we are interested in the approach of the QoS management for Diffserv because we believe that it presents a challenge for adaptive and automated mechanisms of QoS provisioning and QoS control components (e.g. resource provisioning, admission control and policy decisions).

2.3 QoS management for IntServ

The IntServ (Integrated Services) framework provides the ability for applications to choose among multiple, controlled levels of delivery service for their data packets [9]. Individual network elements (subnets and IP routers) along the path followed by an application's data packets support mechanisms to control the quality of service delivered by to those packets. Each network element is QoS aware and supports interfaces required by the service definition.

The interfaces represent a way to communicate the application's requirements to network elements along the path and to convey QoS management information between network elements and the application (fig 2.1).

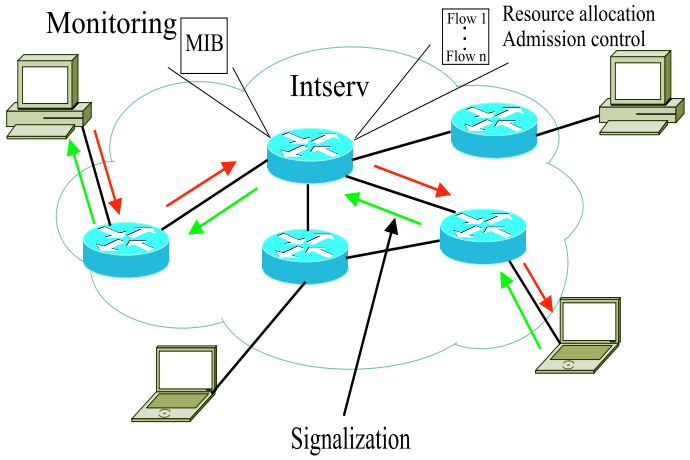


Figure 2.1: QoS Management for IntServ

2.3.1 QoS specification

Intserv IETF Work Group has produced a set of specifications for QoS service-levels based on a general network service specification template [58] and some general QoS parameters [59]. The template allows the definition how QoS mechanisms should be work in the network elements. With the present IPS service enumerated as *best-effort*, two service-level specifications are defined: controlled-load service [56] and guaranteed service [60].

2.3.2 QoS provisioning

QoS provisioning use a signaling protocol, RSVP [11], to allow the use of the two services to be signaled through network [9]. In Chapter 3 we detail the signaling protocol RSVP. RSVP is used by applications to make resource reservation, by asking the network to provide a defined quality of service flow. The reservation request consists of a FlowSpec identifying the traffic characteristics and service-level required. The information about the resource reservation required is hold using soft-state mechanism in each network element on the path transmission. Using RSVP and soft state, the sender adjusts its service-level parameters based on the feedback from the receiver (close-loop flow control).

Controlled-load service admission control will be handled in a different manner from guaranteed service admission control. Where a network element supports both controlled-load and guaranteed services for different flows, careful engineering must ensure that the service commitments (SLA) undertaken by the network element are maintained. Three different approaches for admission control are introduced [61]: measurement-based admission control, endpoint admission control and policy-based admission control.

2.3.3 QoS control

Once service invocation has been accepted, the network must employ mechanisms that ensure that the packets within the flow receive the service that has been requested for that flow. This requires the use of scheduling mechanisms and queue management for the flows within the routers. The details of the scheduling algorithm may be specific to the particular output medium (Ethernet, ATM, etc.), service-level. Multiple models for scheduling have been developed for Intserv framework. However, the realization of these mechanisms is subject to some practical constraints in implementation, due in part to the computational complexity of the algorithms involved. Schedulers that currently receive attention are: Deficit Round Robin (DRR) [62], Hierarchical Fair Service Curve (HFSC) [63] and Weighted Fair Queuing (WFQ) [64].

2.3.4 QoS Monitoring

Intserv IETF Work Group also defines some SNMPv2 extension [65] [66] to allow remote monitoring and management of network elements that support these network services. The role of the QoS monitoring component is to tune the RSVP protocol parameters to avoid the overhead of messages and to follow the changes in traffic and SLAs. The QoS monitoring component is used in billing operations and admission control policies. To set admission control policies the traffic control module uses statistics tables from MIB (Management Information Base) structure. These tables contains the information display by senders or receivers, regarding their potential contribution to session data content or regarding their needs with respect to sessions and senders.

In IntServ, static QoS management is done by SNMP. The soft state created in this way by SNMP does not time out and cannot be deleted by receiving an RSVP teardown message. It can only be deleted by SNMP messages.

Another role of the monitoring component is to secure the QoS request information among the network elements. This issue is out of scope of the thesis.

The monitoring component in IntServ is not a feedback tool. It is not used in the tuning procedure of the list parameters from flow specifications. The monitoring component is not directly involved in the dynamic QoS management system. It has an informal function.

2.3.5 QoS Trading

In Intserv case, QoS trading component defines the ability of the framework to trade the services among IntServ domain and non-Intserv domain [67], among IntServ and Diffserv domains [68]. The applications can negotiate desired QoS without needing to know the details of different specific network architecture.

The resource reservation protocol, RSVP, therefore must provide correct protocol operation even when two RSVP-capable network elements (routers) are joined by an arbitrary cloud of non-RSVP network elements.

To support connection of RSVP networks through non-RSVP networks, RSVP supports tunneling, which occurs automatically through non-RSVP cloud [69]. Tunneling requires RSVP and non-RSVP network elements to forward path messages toward the destination address by using a local routing table. QoS trading task in this case is mapping the flows into the tunnels.

Intserv and Diffserv are viewed as complementary technologies in the pursuit of end-to-end QoS. Intserv enables hosts to request per-flow, quantifiable resources, along end-to-end data paths and to obtain feedback regarding admissibility of these requests. Diffserv enables scalability across large networks. The framework presented in [68] describes two regions: non-Diffserv and Diffserv region. IntServ architecture is used to deliver end-to-end QoS applications. The network includes combination of Intserv capable nodes and Diffserv regions (in which aggregate traffic control is applied). A network example is presented in fig. 2.2

The QoS trading defines in this case the mapping of the network services between these two technologies [68].

In the next section we take another view of using network resources like IntServ. A key distinction of the next QoS framework presentation is that it is geared to a business model of operation based on administrative bounds, with services allocated to users or users groups.

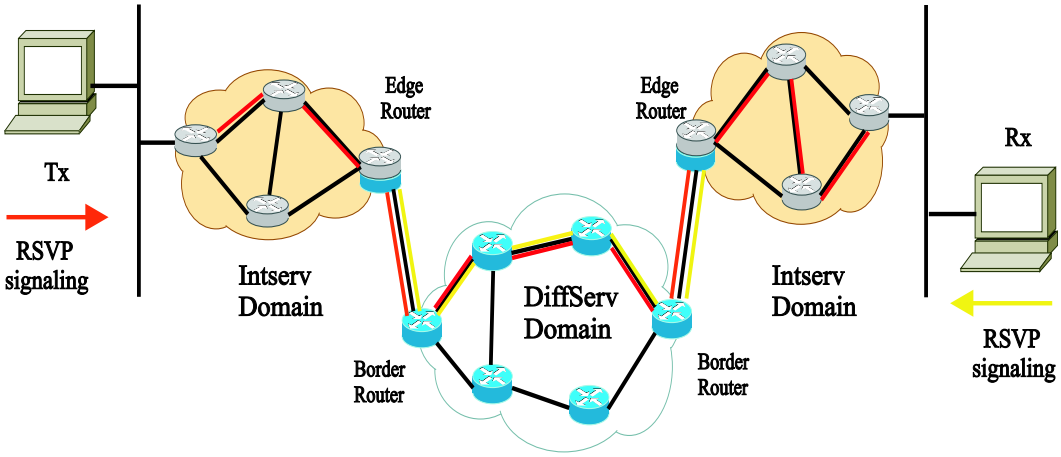


Figure 2.2: Interoperation Inserv/Diffserv

2.4 QoS Management for MPLS

The MPLS (Multiple Protocol Label Switching) working group within IETF is standardizing the technology for using a label-based forwarding paradigm in conjunction with layer 3 routing. The MPLS technology can operate over various link-level technologies, which includes packet over Sonet, frame relay, ATM, Ethernet, and token ring. MPLS combines layer 2 switching technology with layer 3 of the network layer services while reducing the complexity and operational cost [70].

MPLS is similar to Diffserv in some aspects, as it also marks traffic at ingress boundaries in a network, and un-marks at egress points. But unlike Diffserv, which uses the marking to determine priority within a router, MPLS markings (20-bit labels) are primarily designed to determine the next router hop. MPLS is not application controlled (no MPLS APIs exist), nor does it have an end-host protocol component.

MPLS is more of a traffic engineering protocol than a QoS protocol. MPLS routing is used to establish fixed bandwidth pipes analogous to ATM or Frame Relay virtual circuits.

MPLS simplifies the routing process (decreases overhead to increase performance) while it also increases flexibility with a layer of indirection [71]. The route taken by an MPLS-labeled packet is called the Label Switched Path (LSP) [72]. The idea behind MPLS is that by using a label to determine the next hop, routers have less work to do and can act more like simple switches. The label represents the route and by using policy to assign the label, network managers have more control for more precise traffic engineer-

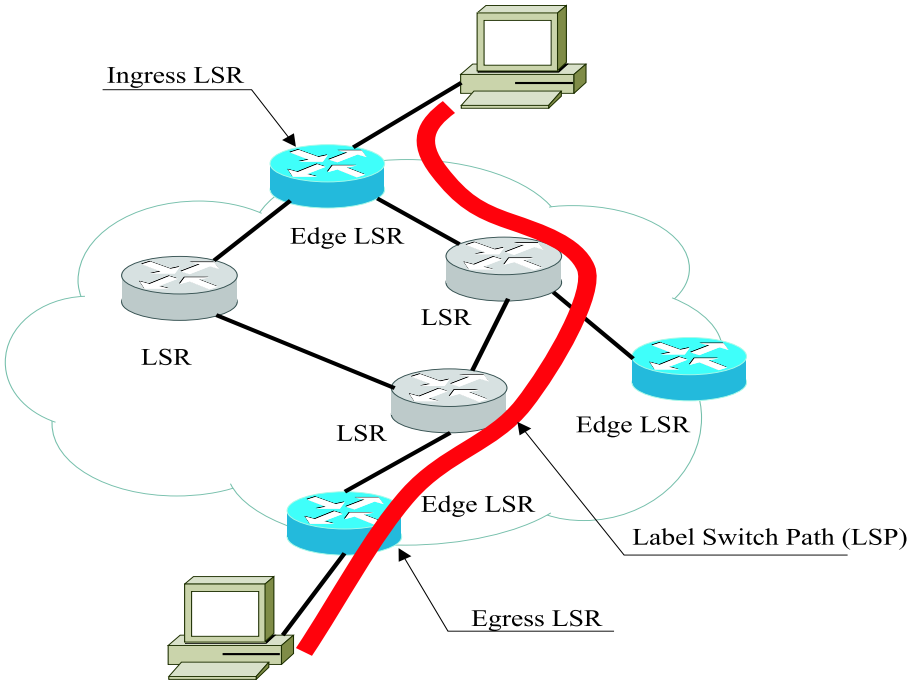


Figure 2.3: MPLS network structure

ing. Nonetheless, label processing is the relatively simple aspect of MPLS. A more complex aspect of MPLS involves the distribution and management of labels among MPLS routers, to ensure they agree on the meaning of various labels.

2.4.1 QoS Specification

The Class of Service (CoS) feature for MPLS enables ISPs to provide differentiated types of service across an MPLS network. The service can be specified in different ways, for example, using the IP precedence bit setting in IP packets.

An ISP can (specify) take two approaches to support MPLS-based class of service forwarding:

- traffic flowing through a particular LSP (Label Switching Path) can be queued for transmission on each LSRs (Label Switching Router) outbound interface based on the setting of precedence bits carried in MPLS header.

- an ISP can provision multiple LSPs between each pair of edge LSRs. Each LSP can be traffic engineered to provide different

performance and bandwidth guarantees. The head end LSR could place high-priority traffic in one LSP, medium-priority traffic in another LSP, best-effort traffic in a third LSP, and less-than-best-effort traffic in a fourth LSP.

2.4.2 QoS Monitoring

One of the MPLS problems is the traffic control for optimization of using resources. Monitoring task in MPLS is part of the traffic control mechanisms. The role of monitoring in MPLS can be used as input to network planning and analysis tools to identify bottlenecks and trunk utilization and to plan for the future network expansion or for the future network resource re-organization.

The MPLS monitoring task introduce more flexibility to develop network applications like virtual private network, traffic engineering, and class of service. VPN tunnel automatically increases or decreases bandwidth reserved based on measured traffic load. Tunnels are resized within a specified range based on actual traffic rates over time. Both time interval and bandwidth range is configurable. The average bandwidth is monitored during this interval. Load balancing mechanism is part of MPLS-TE application [73]. The goal is to load proportional traffic to the configured bandwidth per each LSP. The maximum average value of the path for the LSP is set looking the traffic statistics in the core MPLS routers. Monitoring task is used to automatically adjust the traffic on each LSP. To guarantee point-to-point bandwidth the monitoring task measures and controls failed links/nodes, traffic load per each LSP.

2.4.3 QoS Provisioning

The resource reservation in MPLS is based on the several signaling protocols. Signaling is a means by which routers exchange relevant information across the control plane. In an MPLS network, the type of information exchanged between routers depends on the signaling protocol being used. At a base level, labels must be distributed to all MPLS enabled routers that are expected to forward data for a specific FEC. The Label Distribution Protocol (LDP) [74] deals only with this facet of signaling. However, other dynamic signaling protocols, like RSVP-TE and CR-LDP are capable of signaling and establishing paths through a network to include different types of constraints or reservation of network resources.

It is possible to create MPLS enable networks without the aid of signaling protocols, just as it is possible to create IP forwarding networks without a routing protocol. The ability to statically configure each MPLS node with

the necessary information for packet forwarding, including in and out labels, actions and next hops are available. This approach is roughly equivalent to using static routes in an IP network. As the networks change, get larger and more complex with increased dynamics, the static approach becomes increasing complex and resource intensive to operate, if it can be maintained at all.

Dynamic signaling protocols have been designed to allow single routers to request the establishment and label binding to FEC for an end-to-end path. The instantiating router simply determines the best path through the network conforming to the local constraints and requests the routers in the path to establish a path and distribute the label binding to FEC. Configuring a new LSP, over a core that is MPLS and signaling enabled, does not require anything beyond the configuration in the instantiating router.

Certainly, MPLS is enhanced by a protocol ability to create a path using a dynamic signaling mechanism. Three signaling protocols are available for use in MPLS networks.

Label Distribution Protocol (LDP) defines the protocol specifically designed for the distribution of information required to properly interpreting label binding to FEC. It does not represent an end-to-end path; rather it is a hop by approach.

Resource Reservation Protocol Traffic Extensions (RSVP-TE) [75] is strictly a control plane protocol that is responsible for signaling reservation requests at the micro-flow level between two end stations. The extensions to the original specification result in the MPLS capable signaling protocol, RSVP-TE. This extended protocol introduced new ways to allow RSVP-TE to scale in large and complex, MPLS label dependant networks underpinned by IP. At a high level the micro-flow host-to-host reservation were replaced by the aggregation of packets requiring the same treatment into single FEC, perform label distribution functions and alleviate the soft state concerns of a protocol that does not use a reliable transport.

Constraint-based Routed Label Distribution Protocol (CR-LDP) [76] shares a strikingly similar functional set with RSVP-TE. To incorporate traffic engineering capabilities LDP protocol was developed with extensions. CR-LDP is using TCP protocol for a reliable signalization. Loss of TCP connection causes loss of LSP.

In [75] is proposed an admission control schema based on RSVP. Currently, bandwidth and bandwidth-related information are the only resources tracked and used for traffic engineering. Admission control determines whether a setup request can be honored for an LSP with traffic parameters. Another schema of admission control for MPLS is proposed by Gyu Myoung et al [77]. A flow-based traffic admission control algorithm can achieve reduce the reject

probability of high priority class flow resulting a high resource utilization.

2.4.4 QoS Control

MPLS Class of Service offers packet classification, congestion avoidance and congestion management. Congestion avoidance is implemented in MPLS using WRED that monitors network traffic, trying to anticipate and prevent congestion at common network and internetwork bottlenecks. WRED can selectively discard lower priority traffic when an interface begins to get congested. It can also provide differentiated performance characteristics for different classes of service. The scheduling between different FECs is based on weighted fair queuing (WFQ) or class based queuing (CBQ) mechanisms. The schedulers are using weights (priorities) to determine how much bandwidth each class of traffic is allocated.

2.4.5 QoS Trading

MPLS traffic engineering would provide an explicit forwarding path through multiple network domains. The nature of this trading is based on QoS path management. QoS path management recalculates, adjusts, and maintains paths through domains based on changes in the network topology, traffic QoS requirements, and each network load. The concept of *path pining* is related to QoS trading. QoS-based routing computes the path, and RSVP manages the Path state via its normal method of message processing [78].

In the next section we will survey in the same manner QoS monitoring Diffserv framework. We consider that the resources management of Diffserv is more flexible than other two presented cases.

2.5 QoS Management for Diffserv

Diffserv (Differentiated Services)was proposed to provide service differentiation by creating service classes with different priorities using either the type of service (TOS) field of IPv4 or the priority bits of IPv6 headers [8]. This priority scheme translates into higher throughput for higher priority classes. The priority goal for differentiated services is to define a scalable service discrimination without the need for per-flow state and signaling at every hop like IntServ.

The secondary goal of differentiated services is to allow different classes of service to be provided for traffic streams on a common network infrastructure. Diffserv aggregates multitude of QoS-enabled flows into a small number of

aggregates. The aggregated flows are given differentiated treatment within network. The diffserv approach attempts to push per-flow complexity away from the network core and towards the edge of the network where both the forwarding speeds and the fan-in of flows are smaller. Each diffserv flow is policed and marked according to the service profile (SLA - Service Level Agreement) at the edge of the network, and service only a small number of traffic aggregates in the core (fig. 2.4).

2.5.1 QoS Specification

Diffserv framework is packet oriented. Each packet receives a particular forwarding treatment based on marking in its IP TOS octet called DSCP (Diffserv Code Point). The packet is marked at the domain boundaries. The packet is treated the same way as others marked the same by the individual network elements. Diffserv IETF Work Group produced a set of specifications for DS field [79] and for PHB (Per-Hop-Behavior) identification codes [80].

IETF Diffserv Work Group has defined two service specifications for Expedited Forwarding PHB [81, 82] and Assured Forwarding PHB [83]. Expedited forwarding (EF) emulates traditional leased line service that promises to deliver customer traffic with a low loss probability at a given peak rate, and a strict priority based per-hop behavior for ensuring latency. This network service is appropriate to applications that have strict bandwidth, latency, and jitter requirements. To create a low-loss, low-delay service, nodes must be configured such that the aggregate has a well-defined minimum departure rate, independent of other traffic. The second service model, Assured Forwarding (AF) model emulates a lightly loaded network even in the presence of congestion. This service promises to deliver traffic with a high degree of reliability within the negotiated latency limits. The Assured Forwarding service class provides delivery of IP packets in four independently forwarded classes with three-discard precedence values for each class.

2.5.2 QoS Provisioning

While IntServ is based on the notion of receiver generated control messages for confirming the resource reservation, Diffserv requires that the ISPs for the receiver and sender have a way of allowing the PHB definition to be honored across the network. The meaning of the DS (Differentiated Services) codepoints and the content of SLAs are established at subscription time and although there will be scope for change by agreement between customer and provider, the kind of dynamic and flexible resource reservation that is overviewed in the above section for using RSVP is not envisaged for Diffserv

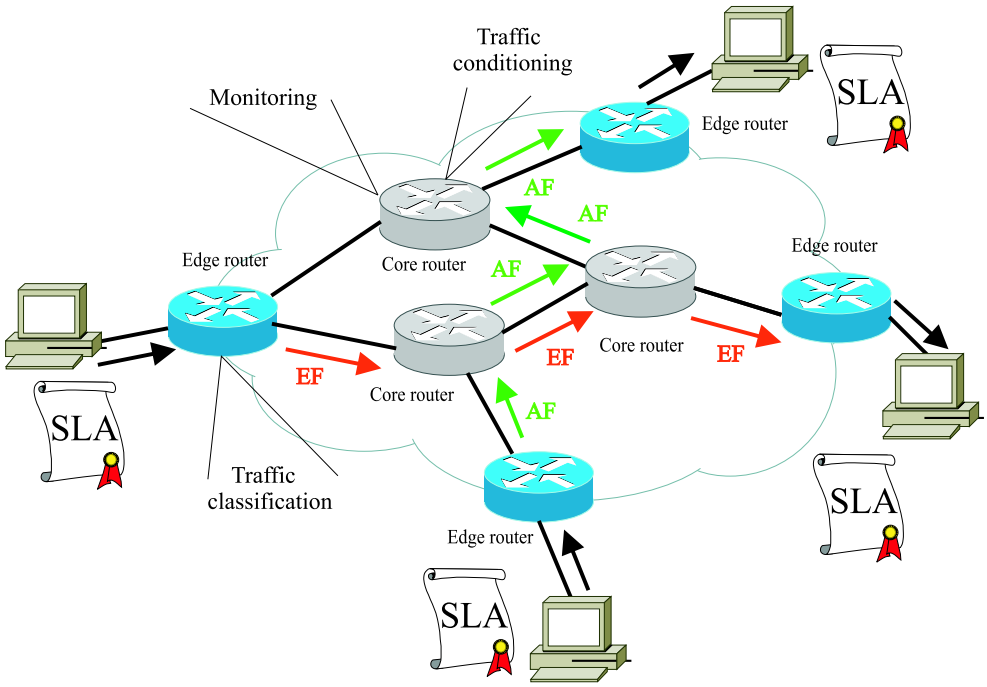


Figure 2.4: QoS Management for Diffserv

framework. However, Diffserv framework currently does not standardize any solution for dynamic resource reservation. Recently, IETF Resource Management in Diffserv Work Group proposes two resource reservation protocols (drafts): PHR (Per Hop Reservation) and PDR (Per Domain Reservation) protocols. The first one is used on a per-hop basis in a Diffserv domain and extends the Diffserv PHB by enabling resource reservation, and the second one tries to represent the resource reservation in the complete Diffserv domain.

As we mentioned in the above sections that the admission control mechanism depends on the type of network service. Diffserv is the framework that supports many services. Recently, several admission control schemes were proposed to provide deterministic or statistical QoS guarantees. Zhang et al. [84] proposed a Bandwidth Broker architecture to manage all the reservation states and store network information. Preemptive deterministic on-line algorithms for line networks were proposed by Garay et al [85]. They study the maximum edge-disjoint path problems. To provide statistical QoS guarantees, L.Breslau et al. propose an end point admission control through probing [86]. Probes are sent out in the requested flow control class and the results gathered by egress router in terms of packet loss, jitter, transmission time, etc decides whether the flow is accepted.

2.5.3 QoS Monitoring

The QoS monitoring component task contribute to implement and to respect guaranteed SLA. The performance of a DiffServ network is strongly dependent on how well edge and core routers work. The monitoring Diffserv network consists in measure the traffic data of each class of service. Each measurement defines a network state. The current state of the network should be compared with the desired stated described by SLAs and decides how to achieve the desired state. The traffic measurements serve as the input to traffic engineering mechanisms.

The role of traffic measurement monitoring component is to enable severe congestion avoidance in the core network and to optimize the network resources usage. When the core links fill beyond configurable thresholds or SLA limits are approached, additional bandwidth is provisioned or resources allocation mechanisms are automatically triggered.

At the edge routers of a Diffserv network the metering process is used to check the SLA between users and other domains, to determine the traffic amount per each user and to determine the traffic amount of inter-domain traffic at the borders.

2.5.4 QoS Control

A differentiated service node is a combination of five component [87]. A packet arrives at the classifier and will be classified dependent on the service. The classifier forwards the packet to its traffic conditioner. The traffic conditioner includes a meter, a marker, a shaper and a dropper. Egress, ingress, interior or first hop router have different demands of functionality.

Traffic meters measure the temporal properties of the stream of packets selected by a classifier against a traffic profile specified in a traffic conditioning agreement. A meter passes state information to other conditioning functions to trigger a particular action for each packet that is either in- or out-of-profile.

Packet markers set the Diffserv field of a packet to a particular code point, adding the market packet to a particular Differentiated Service behavior aggregate. The marker may be configured to mark using all packets which are steered to it to a single codepoint, or may be configured to mark a packet to one of a set of codepoints used to select a per-hop-behavior in a per-hop behavior group, according to the state of a meter.

Shapers delay some or all of packets in a traffic stream in order to bring the stream into compliance with a traffic profile. A shaper usually has a finite-size buffer, and packets may be discarded if there is not sufficient space to hold the delayed packets.

Droppers discard some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. This process is known as "policing" the stream.

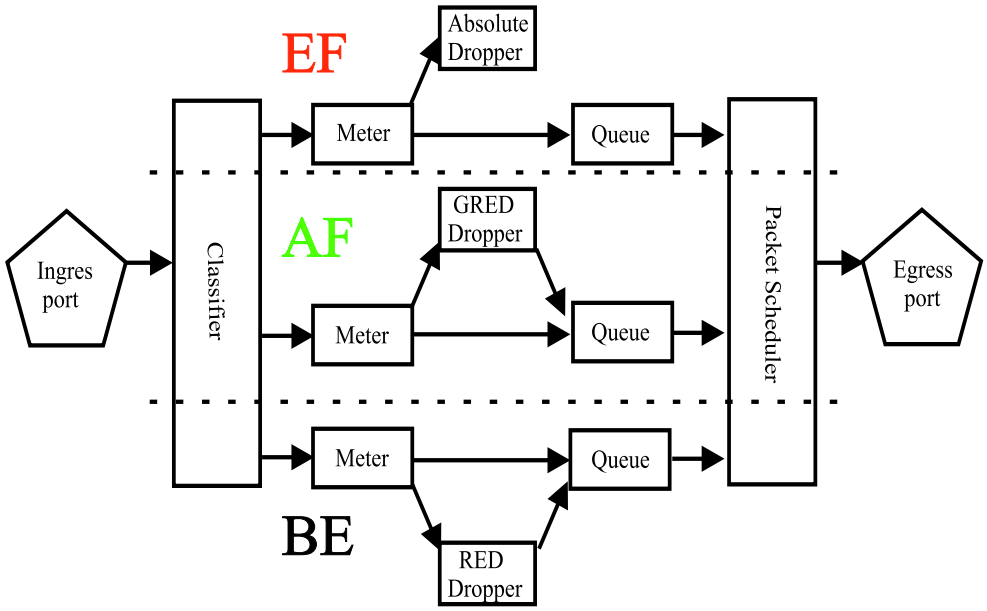


Figure 2.5: Diffserv modules for a core router

Figure 2.5 shows the main modules involved in each class of service design for a core router.

EF class of service is designed to support real time flows with hard delay and jitter constraints. EF packets are queued in a separate buffer and are served before packets of the other classes. The meter module and the absolute dropper are implemented using Token Bucket mechanism [88]. If the traffic burst is greater than bucket size then all the excess packets are dropped. In the case of EF class of service, Token Bucket mechanism shapes the aggregate traffic (especially the burst aggregate traffic).

AF classes of service are designed for flows only asking for bandwidth, mainly TCP flows. The AF has four (sub) classes, each one with three drop precedence [8]. At the onset of congestion, core routers start drop AF packets with the highest drop precedence, then those with the medium drop precedence, and finally if congestion persists, packets with the lowest drop precedence are dropped. Packets in an AF class are served in order. The different AF classes may differ in the applications and transport mechanisms they support. For example, TCP traffic can be protected from non-responsive UDP traffic by separating both types of traffic in two different AF classes. The traffic distribution on the different service classes of Diffserv is defined by

the policies of each Internet Service Provider (ISP). The meter module, the dropper module and the queuing module are implemented using Generalize RED (GRED) or RED-In-Out (RIO) [89].

BE class of service has no guaranteed constraints. Often this service is implemented using a RED mechanism with a minimum threshold to react at congestion before the active queuing mechanisms of AF classes.

Packets of different AF classes are queued in separate buffers and are served using different scheduling algorithms. The main scheduler mechanisms used by the Diffserv implementations are simple round robin, weighted round robin (WRR) [90] and deficit drive round robin (DRR) [62]. Priority scheduling is used for sharing the same output among EF, AF and BE classes of services.

Edge routers are responsible of marking, shaping and policing traffic. Figure 2.5 shows the modules involved in an edge router design.

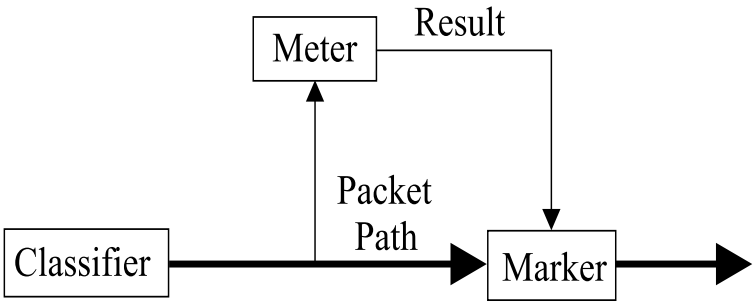


Figure 2.6: Diffserv modules for a edge router

Packets are classified using multiple fields of IP header in different classes. Each class is marked with the same codepoint. The meter is used to measure the traffic that the sources are sending into the network. The meter compares to a SLA and finds the packet in profile or out-of-profile. The behavior of out-of-profile packets is determined by the type of SLA. The out-of-profile packets are marked with different codepoints or dropped. Some routers at the edge may choose to drop out-of-profile packets instead of injecting them into the network. When the congestion occurs in the network the out-of-profile packets are dropped while protecting in-of-profile packets. When all out-of-profile packets are dropped and the congestion persists, in-of-profile of packet start to discard. The mechanism proposed in [91] to support such a preferential dropping is called RIO (RED IN/OUT).

Having this architecture the allocation of the network resources (bandwidth, jitter, delay, drop probability) and the admission control can be done in different ways in Diffserv, depending on the total amount of resources

available in the network and the number of customers asking for a guaranteed service. A possible allocation of the network resources is the notion of effective bandwidth [92]. The effective bandwidth for traffic is the minimum bandwidth to be allocated in the network so that the probabilistic needs of the traffic (e.g., packet loss rate, tail of packet delay) are satisfied. In the effective bandwidth framework, the statistical multiplexing of the different streams of the packets is used to minimize the total amount of resources to be allocated. Another method of resource allocation and admission control is based on the measurement schemes. The traffic rate samples are collected at regular small time intervals during a duration of measurement named window [40],[92]. The average rate of each sample is computed as dividing the sum of packets collected during the sampling period by the length of the sampling period. By using these values traffic rate is estimated. The admission control is then based on this value rather than the worst-case bounds.

2.5.5 QoS Trading

In Diffserv case there is the concept of SLA Trading. SLA Trading is a system based on market principles. It encourages competition among large, well-connected ISPs. Fankhauser et al. proposes a SLA Trading protocol for the Differentiated Services Architecture [49].

2.6 Problem Statement and Scope

It is essential that quality of service should be *configurable*, *maintainable*, *scalable* and *predictive*. Meeting quality of service guarantees in packet switching networks (e.g. Internet) is fundamentally an end-to-end or edge-to-edge issue: from application-to-application. A limitation of most of these presented QoS management frameworks was the static nature of provisioning. In Diffserv framework, the value of a QoS parameter remained constant for duration of a communication (i.e. connection), once negotiated a QoS parameter was never re-negotiated. One implication of this limitation was that the user can not dynamically adjust the communication QoS without the administrator of an ISP helping. For example, the service-provider was committed to provide QoS over the lifetime over the connection. If the provider was unable to maintain its commitment there was no mechanism to inform the user and allow him/her to adapt intelligently to the new level of service. Another example is that the users could not choose to scale back the quality of an existing connection in terms of spatial or temporal performance to allow the possibility of opening a new different connection. Intserv

and MPLS frameworks define QoS management frameworks that make multiple, dynamically selectable qualities of service available to applications in a network.

In the previous sections, we presented a taxonomy for Internet QoS Management and addressed many related work in many areas of network management that are more relevant to this dissertation, in particular:

- Challenges in QoS provisioning
- QoS traffic control mechanisms

Previous research efforts that focused in these problems present a rich spectrum of work in this area, none of these contributions presented a comprehensive solution of dynamic QoS management.

In this dissertation, we focus on the use of Diffserv framework to provide a good network management in order to satisfy each user SLAs. We address in particular on several problems: enhancing resource reservation protocols, efficient correspondence between the SLS parameters and low service-level parameters, automated tuning low service-leve parameters, and improving the local loop resource allocation mechanism.

We propose a criteria classification for QoS signaling protocols that allows to provide suitable QoS management frameworks and service disciplines. We investigate specifies parameters that characterizes the behavior of QoS signaling protocols (e.g. number of sessions, length of signaling packet, CPU, memory, etc.)

Another area that we investigate is the static QoS management. In this case we focus on QoS traffic control mechanisms and QoS provisioning (translation between service level parameters and low level parameters). We consider that the service interfaces are hard configurable and provide only a subset of facilities needed for control and management of continuous media. We propose and implement a Diffserv schema in a local test-bed network. Using test scenarios we study the limitation of static nature of different service provision (e.g.non-efficient utilization of the network resources, no guarantee of service level). We propose for Diffserv a dynamic resource allocation algorithm for core routers. This mechanism ensures that SLAs are respected and allows at the same time an efficient utilization of the network resources (e.g. bandwidth, delay, jitter, packet loss).

2.7 Conclusion

In this chapter we presented the main QoS Internet frameworks looking for DQM components described in chapter 1. A survey of different challenges

that face the large deployment of the network management in the Internet has been given. We mainly focused on explaining the problems that we evoke in this dissertation. In fact, in this thesis we propose a set of contributions in the QoS management in Internet. We consider two research problems: the first one is related to the QoS provisioning, while the second one concerns the QoS control mechanisms.

Chapter 3

Analysis of QoS Signaling Mechanisms

3.1 Introduction

Control plane is an integral part of any QoS provisioning architecture, as support for performance guarantees requires control and management of network resources. In the previous chapter we presented the QoS provisioning components: transfer parameters, resource allocation and admission control.

Resource allocation is composed by the QoS control mechanisms. The QoS control mechanisms are configured appropriately by using signaling protocols.

This chapter describes Internet QoS signaling protocols following the criteria classification proposed in [93]. QoS signaling mechanisms define all algorithms and parameters used to provide QoS between network nodes or applications-network nodes (e.g. end-to-end, edge-to-edge, end-to-edge, etc.). Network elements (e.g. router, host) involved in QoS signaling have the ability to intercept signaling messages and to process them. QoS signaling mechanisms improve the tradeoff between quality of service guarantee and network efficiency. They help to provide differentiated delivery services for individual flows or aggregates, network planing, admission control, network congestion avoidance, etc.

The work described in this chapter is a complementary component of our proposals. Behavior details of QoS signaling protocols are given in this chapter for diagnosing in a QoS management system the weakness of QoS signaling. QoS management system performance depends on the component of QoS signaling protocols. We provide analysis and experimentations for scalability, complexity, flexibility, adaptability and security issues. We are

interested in traffic signaling load, CPU utilization by signaling protocols latency of reservation setup, and granularity of signaling.

We found that the type of implementation, the granularity, the type of network service designing define the behavior of QoS signaling protocols.

The body of this chapter is organized as follows. In Section 3.2 we present the classification criteria proposed in [93] for Internet QoS Signaling protocols. We survey the main QoS signaling protocols in Section 3.4. In Sections 3.5, 3.6 and 3.7 we explore and discuss using proposed classification criteria many issues related to QoS signaling protocols. Section 3.8 summarizes our findings.

3.2 Classification Criteria

We propose a classification criteria of the Internet QoS Signaling protocols. The main goal is to bring the criteria classification on the main existing Internet QoS signaling protocols to point out the differences and open issues to enhance quality of the network service. The *classification* of QoS signaling protocols is divided in *global criteria* (e.g. scalability, complexity, adaptability) and *specific criteria* (e.g. type of bandwidth, reservation initiator, working level, etc.).

Scalability is an important key word for criteria classification. *Scalability* can be defined as the ability to increase the number of reservation states in a network entity, while maintaining the required network performance. When the size of network (e.g. number of nodes, number of applications) is changing, QoS signaling protocols must handle this situation. The QoS signaling scalability is related to the number of messages that need to be processed, average size of packets, CPU utilization, etc. Frequent problems with scalability are achieved in communications with big number of actors.

Another classification criterion is *complexity*. Any QoS signaling protocol has a number of defined messages. A complex protocol requires a big number of messages. The size of messages, the number of buffers, the number of interfaces with applications require a simple design of the signaling protocol to be as simple as is possible. The *complexity* is given by the number of modules implemented, the data structure of resource reservation table, etc.

An important classification criterion is *flexibility*. QoS signaling protocols have the capability to aggregate the information of their messages. The aggregation mechanism will save the amount of traffic signaling in the network and increases the performances of the whole QoS system. QoS signaling protocol should support network-initiated re-negotiation. This is used in cases, where the network is not able to further guarantee resources and want to e.g.

downgrade a reservation. *Flexibility* criterion defines the placement of the signaling initiator. QoS signaling protocol is flexible when there is no importance where it is placed into the network. Network-initiated reservations are available in various scenarios such PSTN gateways, VPNs, and mobility.

The *adaptability* criterion of the signaling mechanism depends on the set-up, maintenance and release a reservation state can be. Two approaches can be characterized as:

- *Hard State* where the resources are released by means of explicit release;

- *Soft State* that needs to be refreshed in certain periods in time, otherwise it will be released. Soft state can also be released by means of explicit release messages.

One classification criterion is the possibility of *IN bandwidth signalization. IN bandwidth* signalization means that on the same flow we have data and signalization messages and *OUT of bandwidth* signalization means that we have two flows one for data and one for signalization.

Initiate reservation classification criterion defines the way in which the reservation is done. In fact it defines who is making the reservation. There are differences between sender and receiver initiated reservation. To initiate reservation, the QoS signals are generated by the senders and some times by the receivers. The main advantage for the sender signals is their small number in the network. Reservation is done using: *bi-directional* way or *uni-directional* way. With *bi-directional* reservations we mean reservations having the same end-points. But the path in the two directions does not need to be the same. The goal of a bi-directional reservation is mainly an optimization in terms of setup delay. There are no requirements on constraints such as the same data path, etc.

A *work domain* criterion differentiates between *inter-domain* and *intra-domain* QoS signaling mechanisms. The inter-domain signaling protocol works between two network domains (e.g. two Autonomous System, Diff-serv domains, MPLS domains) and help to trade the QoS specifications (e.g. service commitment level, flow performance specification, QoS scaling policy, cost of service).

Security criterion provides means for security for each type of QoS signaling protocol. A QoS signaling protocol should request authentication of its signaling messages. Also it should provide means to authorize resource requests. All the message payloads should be protected against modifications. Integrity protection prevents an adversary from modifying parts of the signaling message and from mounting denial of service or theft of service type of attacks against network elements participating in the protocol

execution. Based on the signaling information exchanged between nodes participating in the signaling protocol an adversary may learn both the identities and the content of the signaling messages. To prevent this from happening, confidentiality of the signaling message in a hop-by-hop manner should be provided. Today security criterion is not enough taken in consideration by the network designers. However, we do not discuss the security criterion in this dissertation.

QoS signaling protocols are working on different *network levels*. In function of communication reliability some protocols are designed for routing level. Other protocols are integrated in the application level. For local QoS signalization protocols are defined on the data link level or physical level. We are concentrated on IP level signaling protocols.

We use classification criteria to analyze the QoS signaling protocols surveyed in the section 3.3.

3.3 QoS Signaling Protocols for Resource Management

Before embarking into the analysis and experiments details, let us present a short description of QoS signaling protocols.

This presentation illustrates the features of QoS signaling protocols from various perspectives (end-system, access network, core network) and from various areas (fixed line, mobile, wireless environments). Also, it describes the interaction of the QoS signaling protocols with other protocols, network mechanisms (e.g. QoS mechanisms, security mechanisms), and applications.

3.3.1 Resource Reservation Protocol (RSVP)

The Resource ReSerVation Protocol (RSVP) is a network-control protocol that enables Internet applications to obtain special qualities of service for their data flows [11]. RSVP is not a routing protocol, it works in conjunction with routing protocols. RSVP is the most complex of all the QoS technologies, for applications (hosts) and for network elements (routers and switches). Senders characterize outgoing traffic in terms of the upper and lower bounds of bandwidth, delay and jitter. RSVP sends PATH messages from the sender that contains this traffic specification (Tspec) information to the (unicast or multicast receiver(s)) destination address. Each RSVP-enabled network element (router) along the downstream route establishes a "path-state" that includes the previous source address of the PATH message. To make a resource reservation, receivers send a RESV (reservation request)

message "upstream" to the (local) source of the PATH message. In addition to the Tspec, the RESV message includes the QoS level required (e.g. controlled load service or guaranteed service) in a RSpec and characterizes the packets for which the reservation is being made, called the "filter spec". Together, the Rspec and filter-spec represent "flow-descriptor" that routers use to identify reservations. When an RSVP network element (router or switch) receives an RESV message, it uses the admission control process to authenticate the request and allocate the necessary resources. If the request cannot be satisfied (due to lack of resources or authorization failure), the network element returns an error back to the receiver. If accepted, the network element sends the RESV upstream to the next network element. When the last network element receives the RESV and accepts the request, it sends a confirmation message back to the receiver. There is an explicit teardown process for a reservation when sender or receiver ends an RSVP session.

The scalability and complexity of RSVP are the key words for characterizing the QoS provisioning in IntServ framework. For a big number of flows, RSVP is less scalable and the complexity of signaling modules address performance issues for signaling messages processing. The granularity of RSVP signaling protocol characterizes the adaptability and the flexibility of QoS management system.

3.3.2 Yet another Sender Session Internet Reservation (YESSIR)

YESSIR (Yet another Sender Session Internet Reservations) is a resource reservation protocol that seeks to simplify the process of establishing reserved flows while preserving many unique features introduced in RSVP [47]. This protocol generates reservation requests by *senders* to reduce the processing overhead. It is built as an extension to the Real-time Transport Control Protocol (RTCP), taking advantages of Real-Time Protocol (RTP). It is using soft states to maintain reservations states, supports shared reservation and associated flow merging. YESSIR extends the all-or-nothing reservation model to support partial reservations that improve the duration of the session. This feature improves the advertising network availability.

YESSIR was designed for one-way, sender-initiated end-to-end resource reservation. To support multicast, YESSIR simplifies the reservation styles to individual and shared reservation styles. Individual reservations are made separately for each sender, whereas shared reservations allocate resources that can be used by all senders in a RTP session. Unlike RSVP which supports shared reservation from the receiver's direction, YESSIR handles

the shared reservation style from the sender's direction, thus new receivers can re-use the existing reservation of the previous sender.

YESSIR requires support in applications since it is an integral part of RTCP. Similarly, it requires network routers to inspect RTCP packets to identify reservation requests and refreshes. Routers unaware of YESSIR forward the RTCP packets transparently.

YESSIR is proposed to address several scalability issues associated with RSVP. Using a sender-based model and piggybacks QoS reservation messages on top RTCP [94] to reduce the processing overhead of RSVP, YESSIR becomes flexible and is less complex than RSVP.

3.3.3 Boomerang

Boomerang is a lightweight resource reservation protocol for IP networks [13]. The protocol has only one message type and a single signaling loop for reservation set-up and tear-down, has no requirements on the far end node, but, instead, concentrates the intelligence in the Initiating Node (IN). The Boomerang protocol allows for sender- or receiver-oriented reservations and resource query. Flows are identified with common 5-tuple and QoS can be specified with various means, e.g. service class and bit rate. Boomerang messages are in the initial implementation transported in ICMP ECHO/REPLY messages.

Boomerang can only be used for unicast sessions. There is no support for multicast. The requested QoS can be specified with various methods and both ends of a communication session can make a reservation for their transmitted flow. Boomerang protocol requires an implementation at both communicating nodes and in routers. Boomerang-unaware routers should be able to forward Boomerang messages transparently.

The adaptability of Boomerang is depending on the granularity of signaling because the protocol messages are transported with ICMP(Internet Control Management Protocol). The most complex network element for Boomerang is the Initiating Node because the admission control and resource reservation mechanisms are performed here.

3.3.4 Beagle

Beagle is a resource reservation protocol who provides a way for applications to optimize resource allocation by expressing a wide range of policies to share resources amongst its flows [48]. It allows applications to allocate computational and storage resources in the network.

The Beagle protocol uses an *application mesh* as the basic unit of resource allocation. The application mesh is a logical entity that encapsulates all the computation and communication resources that are allocated to an application at end-points and inside the network. Beagle provides support for allocating computational resources and for delegates - downloadable code fragments that can be used to customize resource management during runtime. Beagle provides support for aggregation of resource allocation at various granularities.

To support multicast, Beagle uses a model where receivers can arbitrarily join and leave multicast groups. Beagle uses hard state for establishing reservation state, but can easily be adapted to use soft-state. Beagle supports both receiver-oriented and sender-oriented resource allocations.

The *application mesh* mechanism increases the adaptability of the protocol because Beagle provides computational resources and customizes resource allocation using code fragments. This issue increases the complexity of signaling modules for each network element and decreases the scalability of signaling protocol.

3.3.5 Other QoS Signaling Protocols

QoS signaling protocols are designed in general to convey per-flow resource reservation information to core routers. Historically speaking before RSVP exist a signaling approach named ST-II. We mention it here to show the improvements made for RSVP.

QoS signaling protocols are designed for interdomain communications to convey resource reservation information to multiple AS's (Autonomous System). QoS scaling policies are enforced using QoS signaling protocols designed to handle policy rules.

ST-II is an experimental resource reservation protocol intended to provide end-to-end real-time guarantees over the Internet [95]. It allows applications to build multi-destination simplex data streams with a desired quality of service. ST-II consists of two protocols: ST for the data transport and SCMP (Stream Control Message Protocol), for all control functions. ST is simple and contains only a single PDU format that is designed for fast and efficient data forwarding in order to achieve low communication delays. SCMP packets are transferred with ST packets.

ST-II has no built-in soft states, thus requires that the network be responsible for correctness. It is sender-initiated, and the overhead for ST-II to handle group membership dynamics is higher than RSVP.

BGRP (Border Gateway Reservation Protocol) is a signaling protocol for interdomain aggregated resource reservation for unicast traffic. BGRP builds

a sink tree for each of the stub domain [96]. Each sink tree aggregates bandwidth reservations from all data sources in the network. BGRP maintains these aggregated reservations using soft state.

BGRP scales in terms of message processing load, state storage and bandwidth. Since the backbone routers only maintain the sink tree information, the total number of reservations at each router scales linearly with the number of Internet domains.

COPS (Common Open Policy Service) [97] is designed specifically for provisioning with all the benefits of a protocol designed for the purpose at hand: event-driven, object-oriented, includes explicit provisions for fail-over, role-combinations, and other provisioning-related issues. COPS has a single connection between client and server, it guarantees only one server updates the policy configuration at any given time (and these are locked, even from console configuration, while COPS is connected to a server).

COPS uses a reliable TCP transport protocol and thus uses a sharing/synchronization mechanism and exchange differential updates only.

In the next section we present the experimental environment for analyze the QoS signaling protocols based on classification criteria proposed above.

3.4 Experimental Environment

The experimental environment includes monitoring tools, traffic generator tools, and traffic control tools with QoS signaling protocols.

Our experimental environment is based on 3 PCs with Linux 7.2, connected to an Ethernet network (figure 3.1).

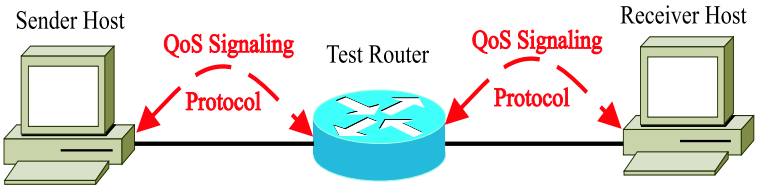


Figure 3.1: Test-bed network

Signaling messages on ingress and egress interfaces of the router were intercepted and logged together with time stamps using *tcpdump* application. We use for hosts 2 Pentium III 450Mhz and for router 1 Pentium III 500Mhz. We install Linux RSVP module, Boomerang module Beagle module and Yessir module. The protocols differ in the implementation of the traffic control. For example, RSVP uses the operating system's built in traffic con-

trol routines, the Boomerang uses own traffic control implementation that is bounded to the protocol.

We use multiple signaling message generators: Boomerang signaling generator [98], RSVP signaling message generator [98] Yessir signaling generator [99]. For Beagle protocol we use application tests developed in the release downloaded from Darwin project [100]. We patched the kernel router with a packet forwarding time measurement tool [101].

During the measurements the interesting issue to study was how the number of concurrent reservation sessions does affect the resource usage. Thus, increasing number of reserved flows was set up and maintained in the router. We are looking for average size of message, CPU loading, message processing time and the latency of reservation setup.

Based on classification criteria proposed in Section 3.2 and on experimental environment presented in Section 3.4 we analyze the QoS signaling protocols presented in Section 3.3.

3.5 Scalability and Complexity

Scalability and complexity are two key words for criteria classification. A complex QoS signaling protocol proposition could be scalable or not. That section reports results of tests conducted to provide those two criteria of several QoS signaling protocols mentioned above.

A QoS signaling protocol is composed by signaling messages and a signal processing module with a resource database. Signaling messages have the capability to transport the information among the signaling entities. The signal processing module intercepts the signaling messages, parsing them and interacts with other modules such routing module, traffic control module, API module. In various situation the processing module sends (repeats, retransmits) messages to other signaling entities.

Using the test-bed network presented in figure 3.1 we installed and tested signaling protocols like RSVP, Boomerang, Beagle and Yessir. For COPS we have used special network configuration that is presented in figure 3.7. For COPS protocol we have used a message simulator provided by Intel COPS release [102].

The complexity of a QoS signal protocol depends on the "number of messages" defined in the protocol framework. An increased number of messages per reservation means that the QoS signaling protocol deals with many management requirements (e.g. synchronization, update databases, etc.). Considering the QoS signaling protocols presented we count the number of messages exchanged between signaling entities. Table 3.1 shows the number

	RSVP	Boomerang	YESSIR	Beagle	BGRP	COPS
Number of messages	5	2	3	6	5	3

Table 3.1: The number of messages for each QoS signaling protocol

of messages per reservation used by the QoS signaling protocol presented in this thesis. The number of messages characterizes the scalability of the QoS signaling protocol.

RSVP defines five "messages" per reservation: PATH, RESV, PATH TEAR, RESV TEAR, and Confirmation or Error messages (figure 3.2).

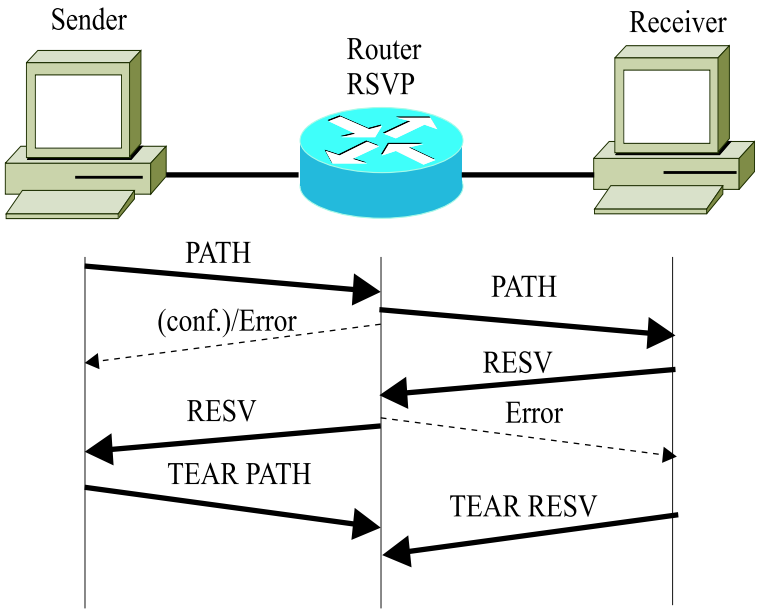


Figure 3.2: RSVP messages

Boomerang protocol messages are wrapped into ICMP Echo/ Echo Replys which are supported in the majority of network nodes and hosts, giving good chance for quick implementation in current Internet. So, Boomerang protocol uses just two messages (figure 3.3).

YESSIR approach takes advantage of the close relationship between RTP (Real-Time Transport Protocol) and RTCP (RTP Control Protocol). YESSIR reservation messages consist of RTCP sender-report messages, enhanced by additional YESSIR-specific data. YESSIR uses three types of messages from RTCP: Sender Reports, Receiver Reports and BYE messages (figure 3.4).

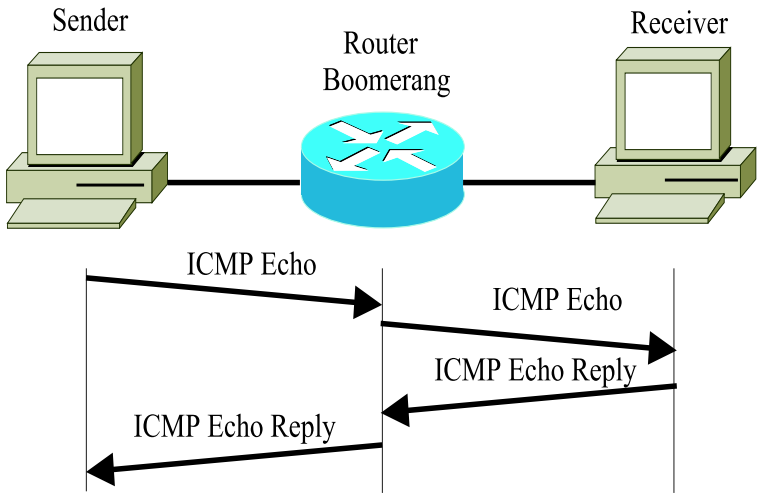


Figure 3.3: Boomerang messages

The YESSIR extension consists of a generic fragment, a flow specification fragment, an optional network monitoring fragment and an optional reservation error fragment. Senders periodically multicast RTCP sender reports (SRs) to all members of the multicast group (or the other party, if unicast). When an RTCP SR is received by a router, the router will attempt to make a resource reservation according to the information specified in the message. As a part of RTCP receiver reports (RR), the receivers will provide failure information to the senders. Based on RRs received, senders can either drop the session, or lower the reservation request and transmitted bandwidth. In addition, an RTCP BYE message, sent when a group of member leaves, releases the YESSIR state record and any resource reservations.

BEAGLE has six messages exchanged between neighboring routers: SETUP_REQ, SETUP_RES, SETUP_CONF, and PROXY_REQ (figure 3.5). Senders are initiates the resources reservation sending SETUP_REQ message. When the destination receives the SETUP_REQ message responses with a SETUP_RES message. At each hop along the path, a node receiving the SETUP_RES messages responds by sending a SETUP_CONF message upstream. For the third-party flow setup Beagle protocol encapsulates the messages in a PROXY_RES message. Flow teardown is accomplished by a two-way handshake using TEAR_REQ and TEAR_RES messages.

BGRP defines several control messages: PROBE, GRAFT, REFRESH ERROR, and TEAR (figure 3.6). Reservation sources always initiate PROBE messages to find out about the network resource availability and the exact reservation path. Reservation sinks, upon receiving PROBE messages, return GRAFT messages to set up the appropriate reservations inside the network.

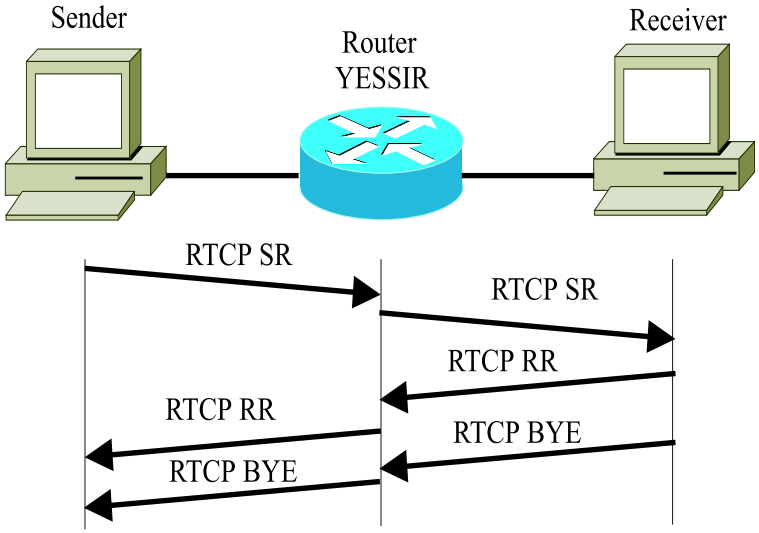


Figure 3.4: YESSIR messages

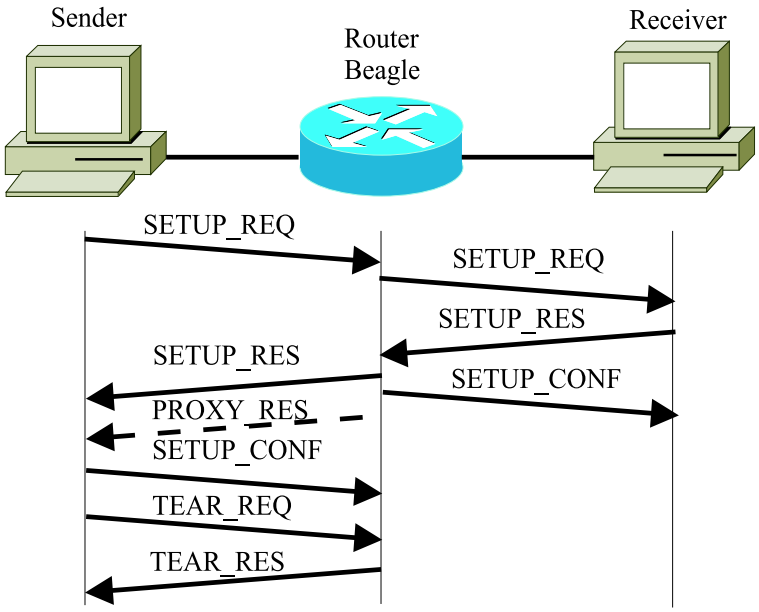


Figure 3.5: Beagle messages

PROBE and GRAFT specify the reservation parameters such as traffic class, priority and bandwidth. Reservation sources and sinks transmit PROBE and GRAFT messages only once during the lifetime of reservation. The border routers periodically exchange REFERSH messages with their peers to make sure reservations are still "alive". In case of reservation failures, the routers send ERROR messages to inform users. The protocol includes optional TEAR messages that routers can send to actively remove reservation when routes change.

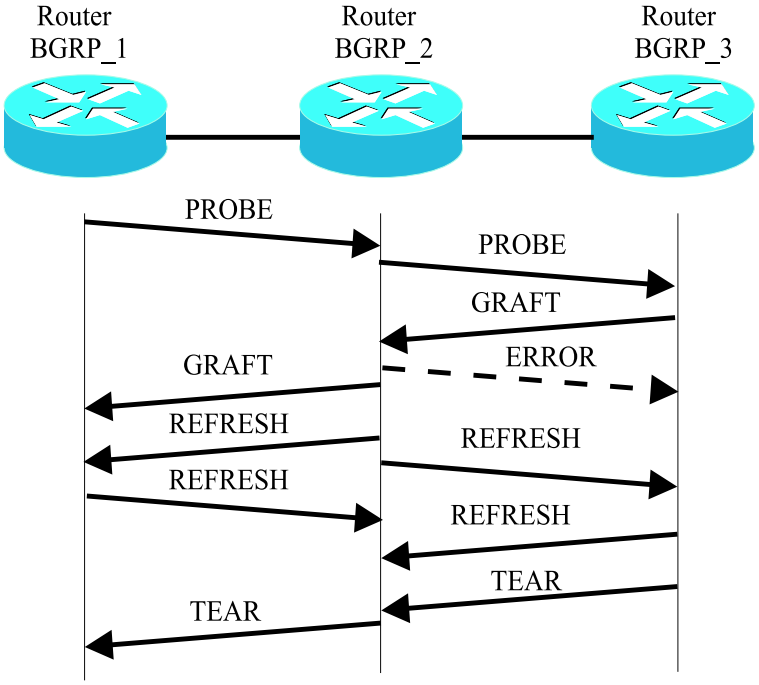


Figure 3.6: BGRP messages

COPS protocol has three basic type of messages: REQ, DEC and RTP (figure 3.7). The policy enforces point request some policy information to policy decision point which response using a decision message. The policy enforces point responses using a report message.

Another feature of the QoS signaling protocols is the average size of the messages. The size of the message depends on the number of critical operations in the signaling handling protocol software. The average size of the packets is computed by looking on protocol specifications and finding the minimum size and the maximum size of the message. Table 3.2 shows the average size of the messages.

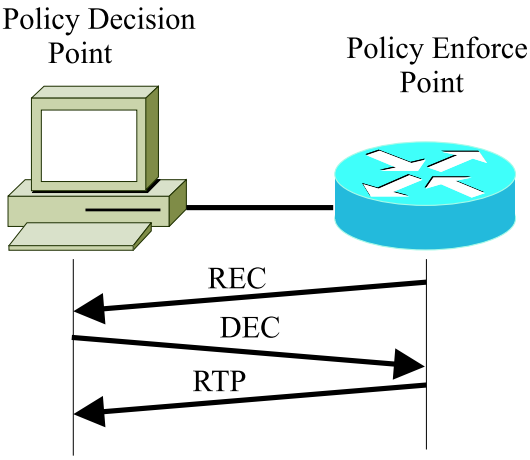


Figure 3.7: COPS messages

	RSVP	Boomerang	YESSIR	Beagle	BGRP	COPS
Average size of message	3.6Kbit	1.2Kbit	0.4Kbit	4.8Kbit	-	-

Table 3.2: The average size of messages for each QoS signaling protocol

As we can observe the average size of YESSIR messages is smallest that others. Features like error reports, reservation reports, types of reservations are not included in the YESSIR messages because there are included in RTCP protocol framework. Beagle protocol has the greatest average size of all protocol messages. Beagle signaling allows reservation requests at different granularities. For example messages can contain objects for a specific flow between endpoints, or reserve resources for a family of applications at once. The signaling messages contain also, downloadable code fragments to customize resource management during runtime.

The average size of message for COPS and BGRP varies with network topology, type of network service offered reservation amount. The characteristic of message design is modularity. Each critical operation of the protocol has one or more signaling object in the messages. We have observed from our experiments that the size of COPS signaling protocol messages', usually does not exceed 10Kbit.

Each QoS signaling protocol has an implementation complexity. This complexity is defined by the number of modules used and the interactions among them. The *signal processing module* interacts with basic modules involved in signaling:

- *traffic control module* which includes: a packet classifier, a packet scheduler and a admission control mechanism;

- *routing module* which informs the processing module the route for signalization;

- *local policy control module* which decides the policy for each packet signalization;

- *management interface module* which controls and provides the behavior of the protocol.

The QoS signaling protocols considered in this dissertation have various type and number of modules.

Beagle allows third-party resource allocation requests through resource broker called Xena [103]. Beagle processing module interacts with a local resource manager module (LRM) to acquire and setup the resources. Beagle supplies a high level interface to an application or Xena and translates requests into lower level control plane signals that the LRM acts upon. Like in figure 3.8 and figure 3.9 the LRM exposes an API to beagled allowing interaction with the packet scheduler and classifier. All the items labeled Beagle, Control API, and Local Resource Manager shown in the control plane comprise a single executable entity.

The RSVP signal processing module interacts through an API with application, with data plane modules (e.g. packet classifier, packet scheduler, admission control module)(figure 3.10).

Yessir reservation setup agent interacts with packet classifier through a flow table, and packet scheduler (figure 3.11). When a Yessir message is received the reservation setup agent will query the local traffic control database for resource availability. If the resource is sufficient at the egress interface, the agent updates the database and the scheduler. The role of flow table is to store during the parsing RTCP messages, RTP data packet information including the IP source and destination addresses, port numbers and protocol type can be learned automatically. After the router successfully sets up the scheduler, it inserts RTCP's IP source and destination address, protocol type, and the corresponding RTP data port numbers into the flow table.

Signal processing module is based on two function classes: session maintenance functions and the message processing functions. The session maintenance functions take care of the soft-state of sessions. Several timers are associated with each block that must be refreshed from time to time. The processing message functions extract the payload of the signaling message sending them to different data structures. Each message type has its own message processing function. The performance of the QoS Signaling protocol

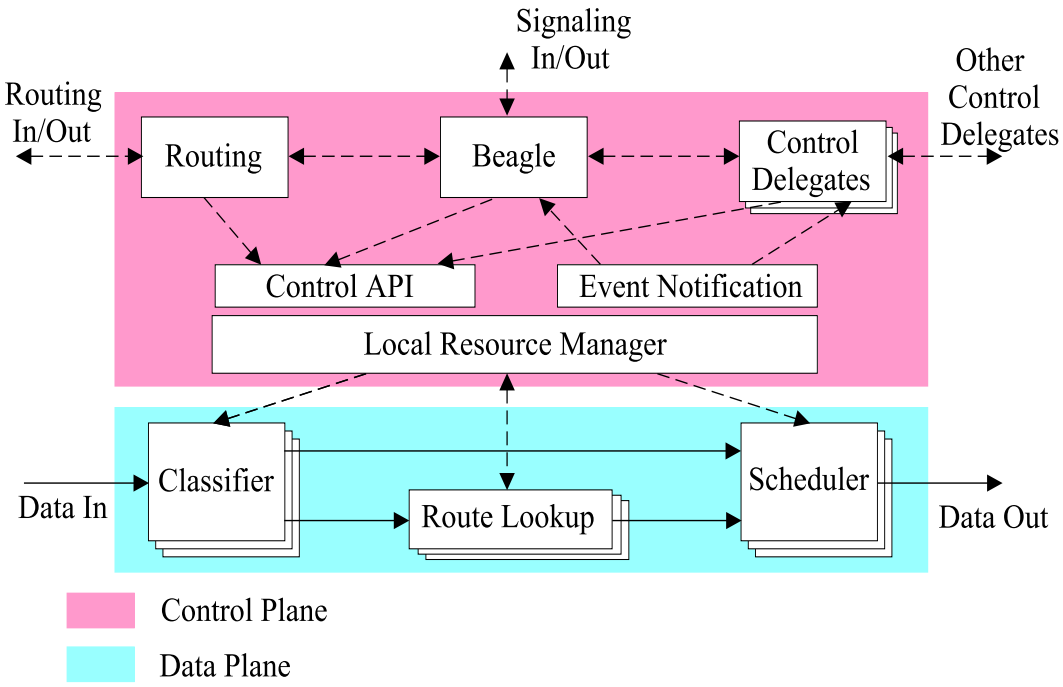


Figure 3.8: Beagle module interaction in a router node architecture

is given by these functions. One of the benchmark is CPU utilization and another is latency of the reservation setup.

CPU utilization characterizes the complexity criterion of a QoS signaling protocols. In fact in figure 3.12 we can see how the session maintenance and the message processing functions impacts the *signal processing module*. We measured the CPU loading with *vmstat* varying the number of flows

Looking at the signaling processor module utilization we can declare in the cases of Beagle and RSVP, the root of the scalability issue is the low performance of the session maintaining routines and the session processing messages routines. In the case of the Boomerang implementation the signaling message processor utilization is significant due to the lacks of signaling message initiation and additional states.

The data structure of reservation table and the reservation lookup scheme is an important factor in scalability. According to the source code *bucket hash* algorithm is used in RSVP implementation, while *modified binomial tree* is implemented for Boomerang. Beagle protocol uses *bucket hash* algorithm for lookup scheme with a complicated structure of resource reservation data base. In case of Yessir, the implementation of lookup scheme is the same that RSVP. Table 3.3 shows the processing time for one signaling message. We

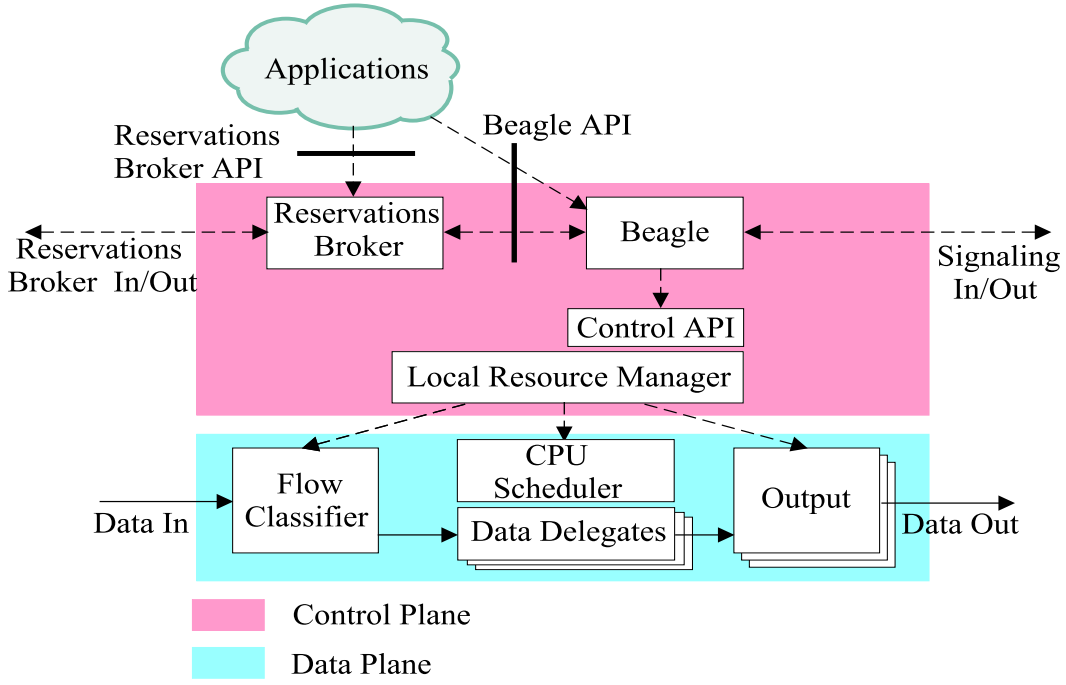


Figure 3.9: Beagle module interaction in end-system/computer server node architecture

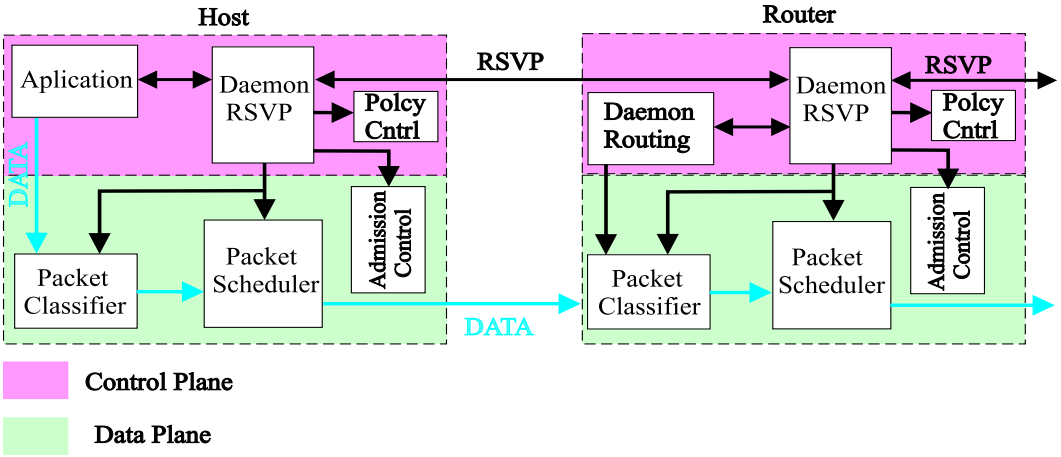


Figure 3.10: RSVP module interaction in a router node architecture

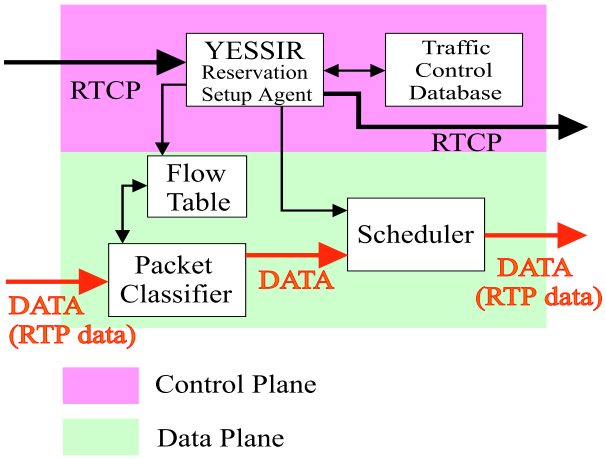


Figure 3.11: YESSIR module interaction in a router node architecture

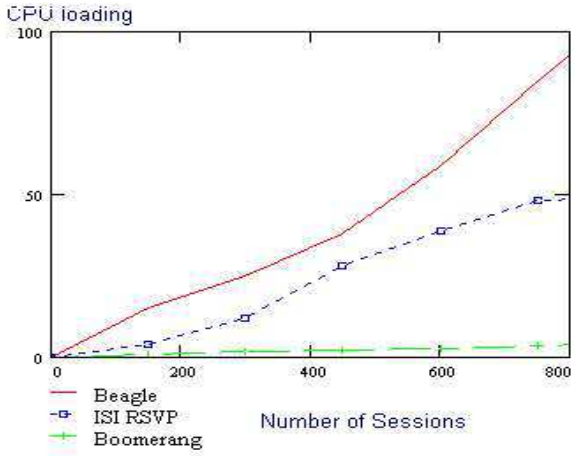


Figure 3.12: CPU loading

	RSVP	Boomerang	YESSIR	Beagle
Message processing time	723us	26us	571us	1997us

Table 3.3: The average time of processing messages in a router

measured using *tcpdump* and logged the packets from both interfaces of the router with the time stamps (figure 3.1). We emulate one flow reservation for each QoS signaling protocol case. The message processing time is obtained from the difference between time among the two interfaces and the forwarding time. The bucket hashing algorithm and large data structures decrease the scalability and increase the complexity of the QoS signaling protocol.

Figure 3.13 shows the slope of the latency of router reservation for different QoS signaling protocols. We measure the delay between two interfaces of the router (figure 3.1) generating new sessions with the signaling generators mentioned in Section 3.3. We can observe that the protocols using complex database reservation, bucket hashing algorithms, and signaling message processing implementation introduce a high latency of reservation setup. Additional experiments linked to figure 3.13 show that signaling intensity -

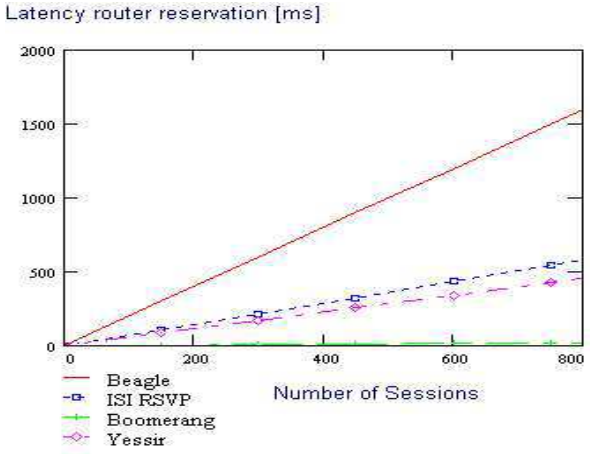


Figure 3.13: Latency of reservation setup

the number of atomic signaling messages received by the router per second - is also an important scalability factor. RSVP protocol loses signaling messages after approx. 42 messages per second. Boomerang could be scaled up to 126 messages per second and Beagle approx. 30 messages per second.

3.6 Flexibility and Adaptability

There are network environments that requires *flexibility* and *adaptability* for the QoS signaling protocols. In networks where resources are very expensive, flexibility and adaptability of the QoS signaling systems cut some management costs. A QoS signaling protocol is flexible if it can be employed in various ways. The adaptability means that the QoS signaling protocol becomes suitable for a new usage or new network conditions.

Internet Service Providers develop new network services. The flexibility of QoS signaling protocols consists in helping to provide network services independently of the network context (e.g. topology, operating systems, etc.).

Flow signaling aggregation increases the flexibility in order to support various reservation types. One reservation option concerns the treatment of reservations for different senders/receivers within the same session: establish a *distinct* reservation for each upstream sender, or else make single reservation that is *shared* among all packets of selected senders. Another reservation option controls the selection of senders. It may be *explicit* list of all selected senders, or a *wildcard* that implicitly selects all the senders to the session. RSVP has three reservation styles: wildcard-filter style (WF), fixed-filter style (FF), and shared explicit style (SE). Shared explicit style implies one type of reservation shared by a list of explicit senders defined by the receiver. Instead of sending for each sender a reservation request, RSVP aggregates the information.

Yessir defines two reservation styles, individual and shared. In individual reservations, every sender in a RTP session has a resource reservation of its own. In a shared reservation, all senders of an RTP session share a single resource reservation in the network. The single resource reservation (in shared reservation case) supports aggregate mechanisms [104] to simplify the reservation procedure across the network.

Beagle protocol is more flexible than RSVP and Yessir because it uses sender-based temporal sharing by using a list of flow combinations and their associated aggregate QoS requirements. In Beagle there are no specified reservation styles because it generalizes the support for resource sharing, uses the application mesh concept as the basic unit for resource allocation, provides mechanisms to allow core routers to process only aggregated resource allocation requests.

Another aspect of flexibility is in the placement of the signaling initiator. Beagle, Yessir, Boomerang and BGRP protocols are supporting both resource setup allocations sender-based and receiver-based. RSVP is less flexible in the placement of the signaling initiator because is receiver oriented.

Both unidirectional as well as bi-directional reservation define a flexi-

ble signaling protocol. Bi-directional reservation is defined as a reservation having the same end-points. But the path in the two directions does not need to be the same. The goal of a bi-directional reservation is mainly an optimization in terms of setup delay.

Tuning the period time between the signaling messages of a same session make the QoS signaling protocol adaptable for the majority of network changes. The higher signaling load is achieved by decreasing the period time between the signaling messages and considerably large the processing power of CPU. Lower signaling load affects the applications behavior since this slows the reaction of them to changes in traffic and SLAs.

The signaling period of messages is tuned looking for the congestion state of the network. It is possible that during short periods of network congestion the signaling protocol is not taking account of them. Thus, the QoS of the applications is decreasing. If the period of the signalization is tuned to follow the changes of the network the QoS of the application will remain constant.

The adaptability of a QoS signaling protocol is characterized also by the interactions between different type of signalizations. For example RSVP is adaptable for a Diffserv domain [68] or BGRP protocol is adapting the soft-state information using RSVP model, etc. The QoS signaling protocols are adaptable on the same operating system level (e.g. majority on network level) or are adaptable on the different levels stack (e.g. Beagle with mesh application, Yessir with RTP protocol, etc.)

The QoS signaling protocol should be adaptable to the network traffic states for reliable reasons. RSVP, Yessir, Beagle, BGRP, Boomerang are using the periodical transmission message mechanism. For the state reservations this mechanism is reliable. COPS is not using periodical information messages. For reliability and adaptability to the network traffic is using TCP (Transmission Control Protocol).

3.7 Security

This section is not related directly with the title of this thesis but the security criterion for a QoS signaling protocol should be taken for the future management issues. Security requirements are related to authentication of signaling requests, resource authorization, integrity protection, confidentiality, and denial-of-service attacks.

Application requests for signalization are authenticated by the signaling processing module. After authentication of signaling requests, the signaling protocol is authorizing the resource requests. This requirement demands a hook to interact with a policy entity to request authorization data. This

allows an authenticated entity to be associated with authorization data and to verify the resource request. Authorization prevents reservation entities, reservations violating policies, and theft of service.

The QoS signaling protocol protects the message payloads against the modifications using integrity coding algorithm [105]. Integrity protection prevents an adversary from modifying parts of the signaling message and from mounting denial of service or theft of service type of attacks against network elements participating in the protocol execution.

The confidentiality of the signaling message enables privacy and avoids profiling of entities by adversary eavesdropping the signaling traffics along the path.

3.8 Conclusion

In this Chapter we proposed a criteria classification of QoS signaling protocols. We presented the main proposals for QoS signaling protocols in Internet. We made clear by means of examples the necessity of having these criteria to design the optimum signaling system and to characterize in function of parameters like scalability, complexity, adaptability, and security.

This criteria classification helps the network designers to understand the importance of choosing the QoS signaling protocol. The performance of the network management depends by the QoS signaling protocols behavior.

We have found that a lightweight QoS signaling protocol cope with large network performance-wise; however, different QoS signaling protocols might not fully meet the criteria demands of today's network applications. For large networks and big number of users or applications, QoS provisioning can not be deployed and managed in "real-time" with QoS signaling protocols. This is the reason that in the next chapters we are looking for provisioning mechanisms without any additional signaling protocols.

Chapter 4

Low Level Parameters of QoS Control Mechanisms

4.1 Introduction

Network QoS Management allows to provide certain characteristics of the network performance. These characteristics are defined by the QoS specification. QoS mechanisms are selected according to user supplied QoS specification, resource availability and resource management policy. The distinguished components of QoS mechanisms are QoS provisioning and QoS control that operate on faster time scale.

In the previous chapter we focused on the QoS signaling protocols (part of the QoS provisioning) which configure appropriately the QoS control mechanisms. In this chapter we are interested in using QoS control mechanisms provisioned by the QoS specification defined by a SLA. As mentioned in the first chapter, DQM is based on several operational path components. Each component performs specific management operations to the network. Resource allocation and admission control operations (QoS provisioning) are performed when a new SLA is done at the network edge routers. First step is admission control operation. It checks if there are enough resources in the core network routers. Admission control ensures that the performance contracts are assured. Network elements must ensure that they do not over commit their resources. After that, resource allocation operation provisions the network domain to "accept" the new flow. Resource allocation operation respects policy rules. Network elements must ensure applications to not claim more resources than they contracted for. The admission control operation is based also on monitoring task of network management. Monitoring operation and resource allocation operations are performed using QoS

signalisation protocols (e.g. RSVP, COPS, etc.)

Admission control and resource allocation algorithms are provisioned in the network using two levels of scaling management: a packet level and a flow level. In practice, a router may decide to delay, reorder or drop packets based on various criteria. The admission control and allocation mechanisms on packet level include classification, policing and marking, shaping, queuing (perflow, per class), buffer acceptance and scheduling. At a second level of scaling management implementation the network can decide to accept or reject new flows based on the available resources inside the network. At this level QoS signaling protocols like RSVP, COPS, etc. establish and release of QoS constraints and network enforcing policies.

Two classes of parameters are defining QoS mechanisms for QoS provisioning and QoS control: service level parameters (e.g. bandwidth, delay, jitter and packet loss) and low level parameters/network parameters (e.g. buffer length, drop probability, link sharing policies, etc.). The customer for a network service meet but do not exceeds the service performance parameters. That means the low level parameters/network parameters are provisioned inside the network in order to respect the service level parameters.

In this chapter we analyze the behavior of low level parameters of QoS control mechanisms in function of service level specification parameters. The goal of this analysis is to check which network parameter influences the behavior of the service level specification parameters in order to provide simple QoS provisioning mechanisms. Our analysis depends on the network environment. Low level parameters differs from one implementation to another of the QoS control mechanisms Tuning the low level parameters we preserve the service level parameters defined in SLA.

Tuning operations or provisioning operations, actually, are characterized by extensive manual work based on trial-and-error process. Modifying the low level parameters/network parameters in large periods of time, the QoS management does not provide an efficient usage of the network and does not avoid congestion's (e.g. short term transitions). Static QoS Management (SQM) is characterized by long periods of time when the low level parameters/network parameters or service level specifications remain constant through network customer activities. When the requirements and the behavior of the network customers vary in small periods of time, QoS provisioning mechanisms cannot follow their necessities and the performances of customer applications will be poor. In this chapter we prove the importance of modifying the low level parameters in two cases:

- when the state of the network is changed for short period of time (e.g. unexpected traffic users);

- when the service level parameters are changed by the customer (e.g. changing user needs).

In each case we perform manual tuning operations that are time consuming and costly.

The remainder of this chapter is organized as follows. We study the behavior of packet level mechanisms namely Priority Queuing (PQ), Class Based Queuing (CBQ), Random Early Detection (RED) and Token Bucket mechanisms in section 4.2. The necessity of tuning low level parameters in order to respect service level specifications will be detailed in section 4.3. We conclude this chapter in Section 4.4.

4.2 Low Level Parameters Provisioning

In this section we evaluate scheduling mechanisms and queue management mechanisms across a range of low level parameter settings and offered loads. Our results will help to study the importance to tune the low level parameters. The remainder of this section is organized as follows. The experimental environment is presented in subsection 4.1.1. Subsection 4.1.2 includes for each QoS control mechanism a short framework description, Linux low level parameters of implementation and several scenarios involving service level parameters and low level parameters.

4.2.1 Experimental Environment

The challenging problems of section 4.2 are experimented using a Diffserv test-bed platform presented in figure 4.1. The experimental environment includes monitoring tools, traffic generator tools, traffic control tools, and link emulator tools.

We use in the test-bed network Linux Redhat 7.2 operating system. All PC's are PIII with different clocks' speed processors and different network cards.

QoS control mechanisms are implemented in the CR1 router. Looking on the figure 4.1 each host at the right-hand side (HG1, HG2 and, HG3) generates TCP or UDP traffic flows to hosts on the left-hand side. The traffic is generated with Iperf tool [106], and is marked at the output of the interfaces of the source hosts. The traffic sources used in our experiments for this section are UDP or TCP with constant rate. We used for test also ON/OFF source behavior. The main reason for using simple source behavior was to have a good understanding of input parameters. The network bandwidth is

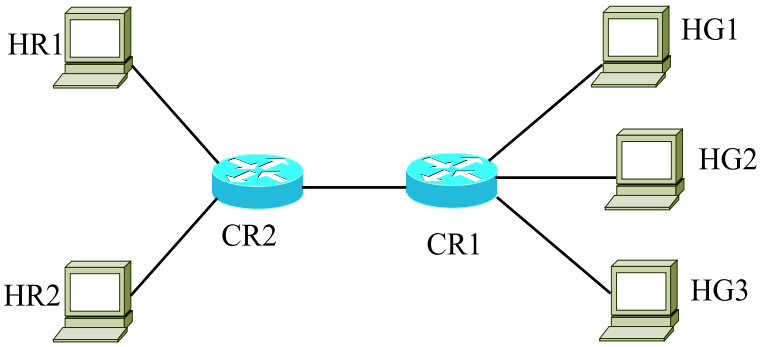


Figure 4.1: Test-bed network

10Mbps. Using NIST Net tool [107] we emulate the link between CR1 and CR2 routers.

The monitoring tool used is TCPSTAT [108] and Netramet [109]. They collect the throughput of different marked flows. For example TCPSTAT works in all interfaces Ethernet of the CR1 router.

We use the tc (traffic control) module implemented in the Linux Diffserv [110][111]. Traffic control can decide if packets are queued or if they are dropped (e.g., if the queue has reached some length limit of if the traffic exceeds some rate limit). It can decide in which order packets are sent (e.g., to give priority to certain flows). It can delay the sending of packets (e.g., to limit the rate of outbound traffic), etc. The traffic control code is implemented in Linux kernel and is configured with a command interface tc. The main components of Linux traffic control are filters, classes and queuing disciplines (e.g., qdisc). Packets are selected by the filters in classes (u32, fw, route, tcindex). Each class uses one of the queuing disciplines (tbf, pfifo, red, gred). Schedulers used by tc are: CBQ (Class Based Queuing), PRIO (Priority queuing), HTB (Hierarchical Token Bucket). There are hierarchies of classes and filters.

In the next section we explore with presented experimental environment, the low level parameters/network parameters of scheduling mechanisms and queue management mechanisms looking on the service level specification parameters.

4.2.2 Scheduling mechanisms

Priority Queuing (PQ)

Priority queuing, which is available in many router products, provides some enhancement to traffic management. A priority queuing mechanism adds

the ability to sort packets based on differences in "priority" and insert them into separate internal queues or shuffle their insertion within a single queue fig. 4.2. The classification of a packet's priority is typically based on packet type, source or destination address, protocol type, port number, or some combination of these parameters. The forwarding algorithm always transmits packets of the highest priority first. If there are no packets of the highest priority level, the next highest priority queue is serviced, and so on.

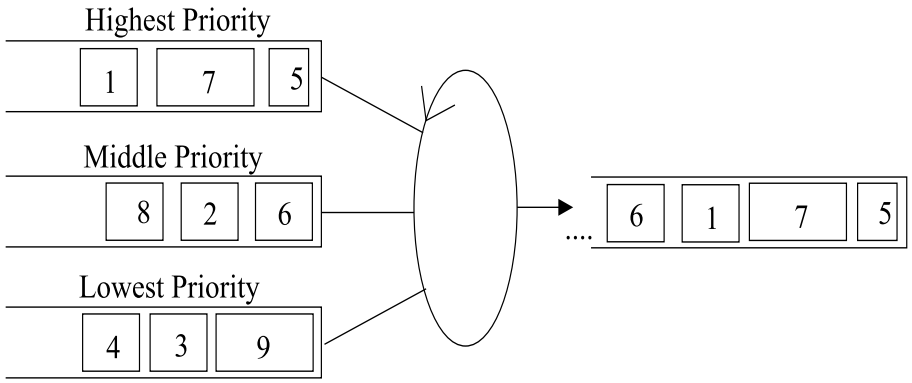


Figure 4.2: Priority Queuing mechanism

There are two modes to operate with PQ :

- absolute priority queuing ;
- rate-controlled priority queuing ;

Absolute PQ ensures that packets in a high-priority queue are always scheduled before packets in lower-priority queues. The priorities in this type of queuing are absolute (i.e. if there is sufficient high priority traffic to saturate a link, all lower priority traffic will be locked out). This can be a considerable problem, since some protocols seek to use the entire available resource. For example, in the absence of competing traffic, a TCP data flow will attempt to maximize its throughput and use all of the available capacity. Thus, a single TCP data flow with a higher priority than all other data flows can lock out all the other data flows for the duration of the TCP connection. In light of the potential for lower priority traffic to be locked out, great care must be taken when assigning the relative priority levels. Finally, FIFO queuing is still essentially used for all packets of a given priority; thus, when multiple traffic flows are aggregated into a single priority level the undesirable behaviors of FIFO queuing still apply.

Rate-controlled PQ allows packets in a high-priority queue to be scheduled before packets in lower-priority queues only if the amount of traffic in

the high-priority queue stays below a user-configured threshold. For example, assume that a high-priority queue has been rate-limited to 20 percent of the output port bandwidth. As long as the high-priority queue consumes less than 20 percent of the output port bandwidth, packets from this queue are scheduled ahead of packets from lower-priority queues. However, if the high-priority queue consume more than 20 percent of the output port bandwidth, packets from lower-priority queues can be scheduled ahead of packets from the high-priority queue.

PQ scheduler mechanism is work conserving (i.e., the scheduler is idle only when there is no packet awaiting service).

Linux Priority Queuing

PRIO scheduler mechanism used in the Linux has just two low level parameters/network parameters: the number of priority "bands" and a map to set priorities for each band [111]. A band can take one of the three priorities: maximum reliability packets (priority = 0), minimum delay (priority = 1) and the rest (priority = 2). The PRIO scheduler operates in absolute priority queuing mode.

Low Level Parameters Behavior

In PRIO case the service level specifications (e.g., delay, bandwidth, jitter, packet loss) are provided by using the low level parameters/network parameters mentioned above.

We assume that among three traffic flows with different traffic marking, in case of congestion, the flow with high priority is not perturbed (the service level specifications are constant). We use three UDP flows generated from HG1, HG2 and HG3. The packets of each flow are marked in each output host interface. Packets with three different TOS fields are filtered and sent in three fifo queues at CR1 router. Each queue represents a "band" class. We use three "band" class. Maximum reliability packets should therefore go to "band 0", minimum delay to "band 1" and the rest to "band 2". We increase with the same value 1Mbps the bandwidth of each flow and we measure on the output interface of CR1 router the throughput of each flow.

Figure 4.3 shows the behavior of bandwidth parameter for three classes, each class has a band priority. If the volume of higher-priority traffic becomes excessive, lower-priority traffic is dropped as the buffer space allocated to low-priority queues starts to overflow.

Figure 4.4 shows that the jitter of the high priority class ("band 0") has a constant behavior. Even the middle class ("band 1") has almost a constant

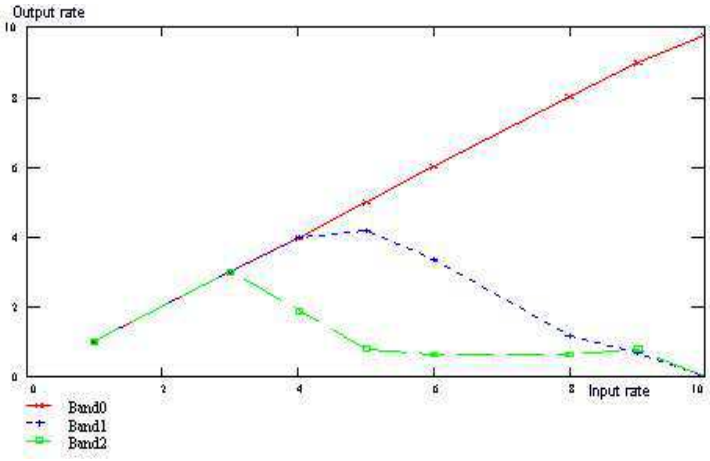


Figure 4.3: Bandwidth Parameter behavior for Priority Queuing

jitter value parameter. We measured the delay service level parameter for high-priority flow and the value is constant.

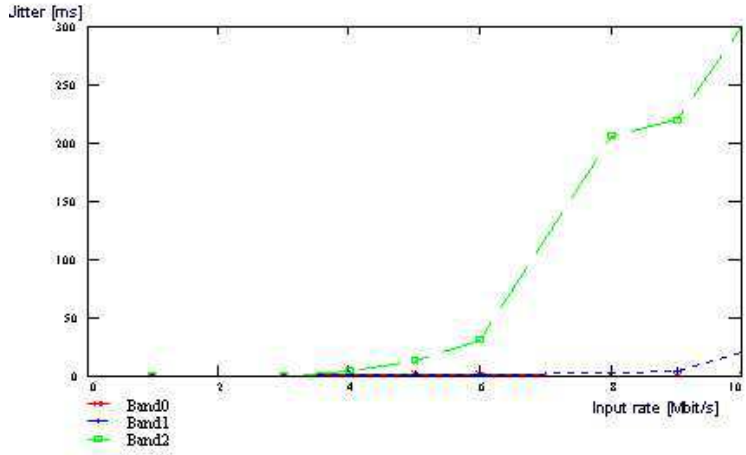
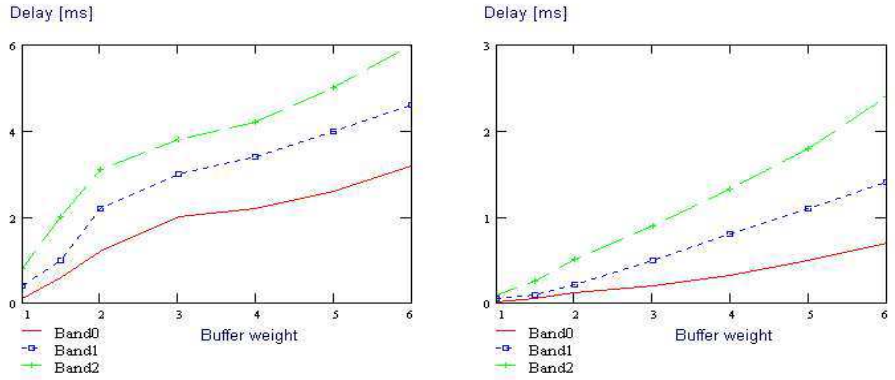


Figure 4.4: Jitter Parameter behavior for Priority Queueing

Priority Queuing can create a network environment where a reduction in the quality of service delivered to the lowest-priority service is delayed until the entire network is devoted to processing only the highest-priority service class.

In the case of PRIO scheduler the service level specification parameters are influenced by the module implementation (e.g., CPU speed and buffer length). Figure 4.5 shows that the packet delay depends on the CPU speed and the buffer size. The buffer size can be configured when the system is installed. We use two PC's with different CPU clock speeds. We compile



(a) Delay caused by the buffer length (CPU 200Mhz)

(b) Delay caused by the buffer length (CPU 450Mhz)

Figure 4.5: Delay Parameter behavior for Priority Queuing

each time the kernel increasing the buffer length of the Linux implementation. The length of the generated packets is 512Kb.

For the PRIO scheduler the low parameters have no major impact for the high-priority flow service level parameters. For the low-priority flow the service level parameters are influenced by the module implementation (e.g. CPU speed, buffer length).

Class Based Queuing (CBQ)

The class based queuing algorithm (CBQ) [112] presents a solution for unified scheduling for link-sharing and real-time purposes. It isolates real-time and best-effort traffic by allowing the separation of traffic in different traffic classes that are then treated according to their requirements, e.g. by giving priority to real-time traffic. Floyd and Jacobson developed a model for hierarchical link sharing and showed how the class based queuing algorithm can be used to set up hierarchical link-sharing structures which allow the controlled distribution of excess bandwidth. Although CBQ has the concept of using a general scheduler, which is used in the absence of congestion, and a link-sharing scheduler for rate-limiting classes, most implementations implement both mechanisms in a single scheduler (e.g., WRR, DRR). Furthermore, an estimator keeps track of the bandwidth a class is receiving. This is done by calculating an exponentially weighted moving average over the discrepancy between the actual inter-departure time of two packets of a

class to inter-departure time corresponding to the specified rate of the class.

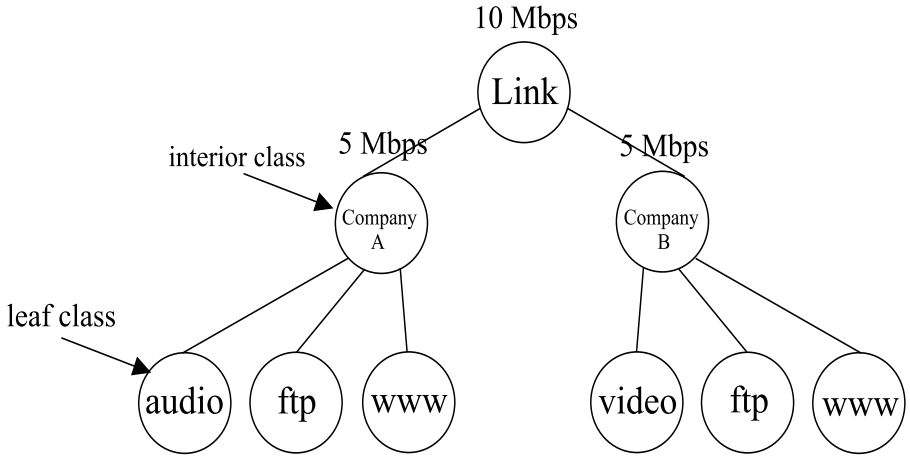


Figure 4.6: Link-sharing between two companies and their traffic classes

An example of link-sharing is shown in figure 4.6, where a 10Mbps link is shared by two companies. In time of congestion, each company should get the allocated 5 Mbps over a relevant time-frame, and within a company share the bandwidth is distributed to the different traffic classes. If a leaf class has no rate assigned to it (e.g. the ftp traffic of Company B), it is not guaranteed any bandwidth at times of congestion. A list of class characteristics is shown in Table 4.1.

CBQ linux implementation [111] acts like priority queuing (PQ) in the sense that classes can have different priorities and that lower priority numbers will be polled before the higher priority ones. Each time a packet is requested by the hardware layer to be sent out to the network, a weighted round robin process (WRR) begins with the higher priority classes. Figure 4.7 illustrates the process of CBQ. The real-time traffic queue is allocated 25 percent of the output port bandwidth, the interactive traffic queue is allocated 25 percent of the output port bandwidth, and the file transfer traffic queue is allocated 50 percent of the output port bandwidth (figure 4.7).

To regulate the amount of the network resources allocated to each service class, a number of low level parameters/network parameters can be tuned to control the desired behavior of each queue:

- The amount of *delay* experienced by packets in a given queue is determined by a combination of the rate that the packets are

Class Characteristic	Definition
regulated	Packets of the class are scheduled by the link-sharing scheduler
unregulated	Packets of the class are scheduled by the general scheduler
overlimit	The class has recently used more than its allocated bandwidth
underlimit	The class has received less than its bandwidth-share
at-limit	The class is neither over- nor underlimit
unsatisfied	A leaf class which is underlimit and has persistent backlog or an interior class which is underlimit and has a child class with a persistent backlog
satisfied	A class which is not unsatisfied
exempt	The class will never be regulated
bounded	The class is never allowed to borrow from ancestors classes
isolated	A class which does not allow non-descendant classes to borrow excess bandwidth and that does not borrow bandwidth from other classes

Table 4.1: Definition of class characteristics in CBQ

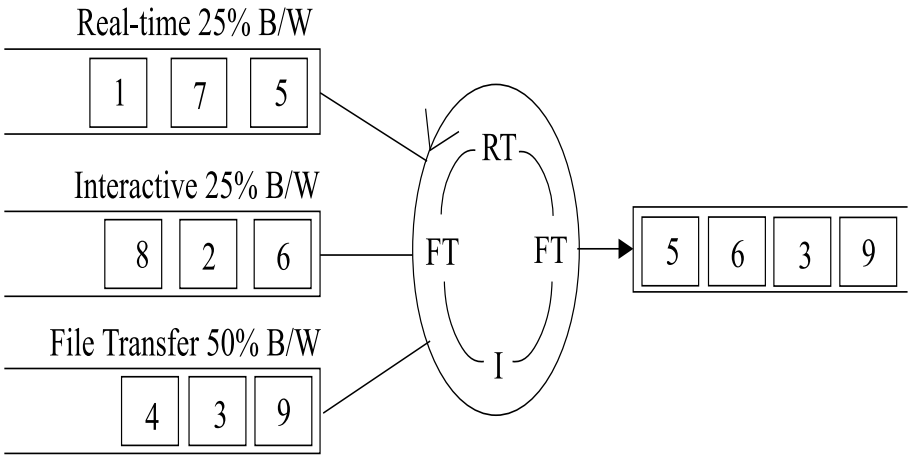


Figure 4.7: Class-based Queuing mechanism

placed into the queue, the depth of the queue, the amount of traffic removed from the queue at each service round, and the number of other service classes (queues) configured on the output port.

- The amount of *jitter* experienced by packets in a given queue is determined by the variability of the delay in the queue, the variability of delay in all other queues, and the variability of the interval between service rounds.

- The amount of *packet loss* experienced by each queue is determined by a combination of the rate packets are placed into the queue, the depth of the queue, the aggressiveness of the queuing management profiles configured for the queue, and the amount of traffic removed from the queue at each service round. The fill rate can be controlled by performing traffic conditioning at some upstream point in the network.

Linux Class Based Queuing

Linux Class Based Queuing implementation has several low level parameters/network parameters. It works by making sure that the link is idle just long enough to bring the real bandwidth to the configure rate. To do so, it calculates the time that should pass between average length of packets. During operations, the effective idle time is measured using an exponential weighted moving average (EWMA) which considers recent packets to be exponentially more important than past one.

Low level parameters/network parameters have impact on the algorithm performance (e.g., granularity) and on the traffic flows (e.g., average packet, rate, etc.). Some low level parameters/network parameters have an implicit value. Other low level parameters are using some thumb rule to be computed. Table 4.2 is showing the low level parameters/network parameters for Linux implementation.

The rule of thumb to provision *weight* parameter is *rate*/10 [111]. The renormalized *weight* is multiplied by *allot* parameter to determine how much data can be sent in one round. As we mentioned in table 4.2, *cell* parameter sets the granularity of the algorithm, and the value recommended in [111] is 8. Another value recommended because we are using Ethernet network is 1514 bytes for *allot* parameter.

Low Level Parameter	Definition
rate	desired rate of traffic leaving each class;
bandwidth	the physical bandwidth of the output device;
avpkt	average size of packet [bytes];
cell	the time transmit of a packet over a device may grow in steps, based on the packet size. This sets the granularity of the algorithm;
maxburst / minburst	the value is used to compute the number of packets stored in one class;
mpu	minimum packet size; needed because even a zero size of the queue a packet is padded to 64 bytes on Ethernet and takes a certain time to transmit;
allot	the base unit of limited amount data for the outer cbq mechanism charge to send out packets on the interface;
prio	classes with high priority are tried first and as long as they have traffic, other classes are not polled for traffic;
weight	allows the class to send more data in one round than the others;
isolated/sharing	a class is configured with "isolated" will not lend out bandwidth to sibling classes;
bounded/borrow	a class will not try to borrow bandwidth from sibling classes.

Table 4.2: Low level parameters in CBQ for Linux

Low Level Parameters Behavior

As we can see from table 4.2 Linux CBQ scheduling mechanism implementation has many low level parameters/network parameters. Tuning low level parameters of CBQ Linux implementation we preserve the service level specifications for each traffic flow or customer. We assume that low level parameters have different impact intensity for each service level parameters. We use the same test-bed network described in Section 4.1. CBQ Linux module is installed in CR1 router.

The amount of delay experienced by IP packets in a CBQ Linux mechanism has two components: the latency in the class queue and the elect time (each time a packet is requested by hardware layer to be sent out to the network). The latency in the class queue depends in general on the depth of queue and the packet size. The depth of queue is designed in function of parameters that characterize the size of the burst: maximum size or minimum size. Higher values of parameter that sets minimum burst lead to more accurate shaping in the long term. In this case a class is allowed to send more packets, followed by a longer suspension of time (when the class is continuously backlogged (steady state)). Higher values of *maxburst* parameter that define the maximum size of the bursts make the CBQ class more tolerant of bursts. When a class is resumed after a suspension, CBQ "forgets" how much this class has sent in the past and the class can now send more than "its limit".

Another parameter that characterizes the latency of the class queue is the average of packets. The maximum idle time (e.g., maxidle) for non-active queues is computed using average packet parameter. We assume that the variation of *avpkt* low level parameter make the class throughput significantly different from the expected value. Using the test-bed network from the figure 4.1 we generate UDP traffic between HG1 and HR1 hosts via CR1 and CR2 routers. The size of packets for the first case is 1500Bytes and for the second case is 128Kbytes. The link between two routers has 6Mb. We vary the *avpkt* to see the impact on the output throughput of two classes. The amount of each class is 1.8Mb and 4.2Mb.

Figure 4.8 shows the throughput of a class (flat hierarchy, borrow disabled) when sending only one UDP flow for each class at a time, repeated with different packet sizes.

When the *avpkt* low parameter is increasing the throughput measured of the output interface of the CR1 router is the expected one around the packet size values chosen 128KBytes and 1500KBytes. In the rest the measured throughput has no exact values. So, to obtain a desired throughput the *avpkt* parameter is chosen in function of the average packet size of the traffic

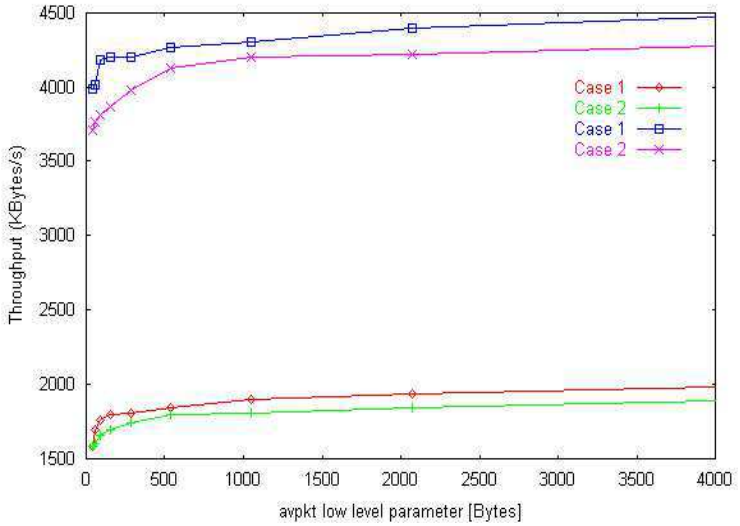


Figure 4.8: Average packet size variation

class.

Average packet size can be hard to tune because of the diversity in packet sizes for flows aggregated in the same class. For instance, even TCP traffic that uses most frequently large packets (e.g. HTTP, RTP) does not have a typical average packet size because of the acknowledgement packets (usually 40 bytes) or interactive applications like telnet.

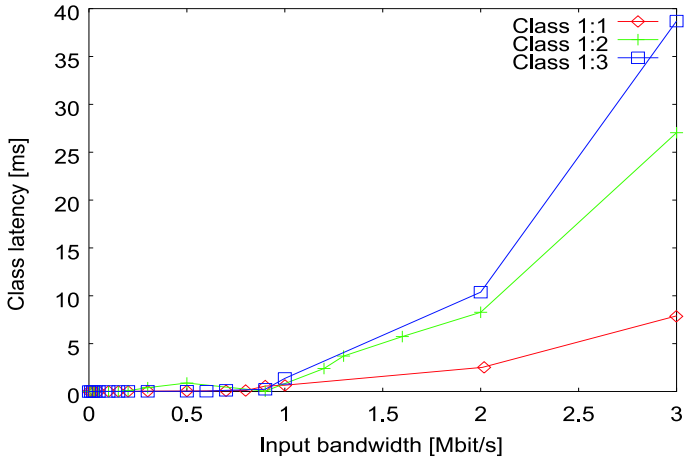


Figure 4.9: Delay of each class caused by prio parameter

Using the class priority schema the high priority class has minimum delay and jitter. Figure 4.9 shows how *prio* parameter affects the delay packets. UDP packets are classified in three classes receiving the same amount of

traffic each one. The class with low priority has the lowest average delay values. The hierarchy of classes used for this experiment is a flat hierarchy. Figure 4.9 has been obtained using the test-bed network from the figure 4.1. We set the same values of low level parameters for each class. The link between the two routers CR1 and CR2 has a bandwidth of 9Mb. Each class has a *rate* of 3Mb. We increase in the same time the throughput of each class using UDP packets. The *prio* low level parameters are: class 1:1 = prio 3, class 1:2 = prio 2 and class 1:3 = prio 1.

The throughput of each class is controlled by the link sharing scheduler. The link sharing scheduler estimates the class' use of the output link bandwidth and marks a class as *underlimit* (if is transmitting at a lower rate than its allocation), *at limit* (if its rate is equal to its allocation), or *overlimit* otherwise. In general a class can borrow bandwidth only if its parent is underlimit or if it has an underlimit ancestor. Three parameters are involved for the distribution of the class throughput in Linux: *bounded* parameter, *isolated* (Table 4.1) and *rate*. Using both parameters CBQ can split the output bandwidth between classes blocking the link sharing scheduler. So, the excess bandwidth is not distributed according to the link sharing structure.

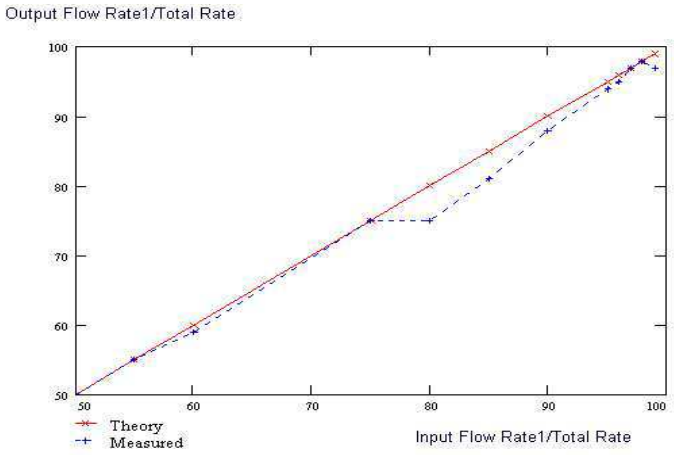


Figure 4.10: CBQ splitting two classes

Figure 4.10 shows how accurate CBQ splits two classes when parameters *bounded* and *isolated* are activated. We observe some imprecision in splitting the bandwidth of both classes, caused by the estimation algorithm implemented in link-sharing scheduler. For a policy rule which changes the amount of the class bandwidth it must to take care of some tolerance (e.g. 1%...3%).

In figure 4.11 a CBQ class was created and a number of parallel flows with the same rate each one have crossed the scheduler. The number of parallel

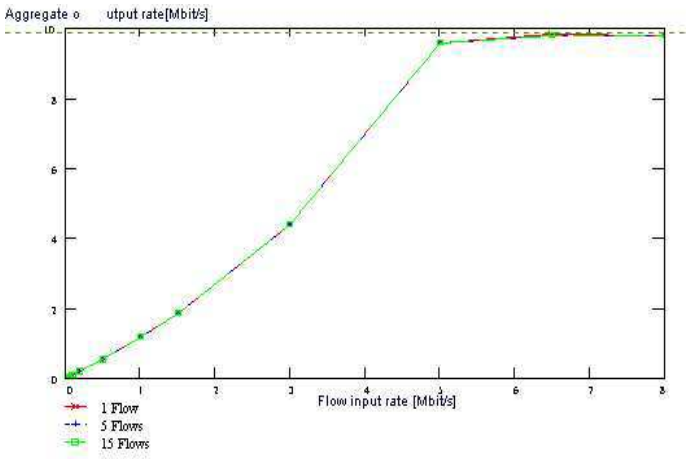


Figure 4.11: CBQ splitting behavior with different number of flows

flows varies with the following values: 1 flow, 5 flows, and 15 flows. As we can see from the figure 4.10 the values are the same for the class throughput in all three cases. The rate parameter of CBQ scheduler varies. Aggregate output rate or the output throughput of the CBQ class has no same values as low level parameter *rate* configured after a certain value. The reason of this behavior is Linux implementation of the Top-level Link Sharing scheduler. We measured the delay for each set of experiments and we validated that it increases in this case with the number of flows. Conclusion is that we can bound each throughput class that we want based on some trial-error process.

The delay of packets is caused also by the number of classes used by CBQ. It increases the elected time. In figure 4.11 we generate four flows, one for each class using a hierarchy of classes. We use parameters *bounded* and *isolated*. The goal of this experiment is to observe the delay and jitter using a hierarchy of classes in CBQ. We use for 1:11, 1:21, and 1:22 UDP packets and for 1:12 TCP packets.

	Flows	Input Bw (Kbps)	Output Bw (Kbps)	Delay	Jitter
Flow1 without CBQ	1:1	1917	1917	35.7ms	7.74 μs
Flow1 with CBQ	1:1	1917	578	104ms	18μs
Flow2 with CBQ	1:2	1917	1917	38ms	8.46μs
Flows 1 and 2 without CBQ	1:1	1000	980	43ms	7.63μs
	1:2	1000	988	46.2ms	9.54μs
Flows 1 and 2 with CBQ	1:1	1000	479	121ms	16.1μs
	1:2	1000	980	89ms	10.32μs
All the flows with CBQ	1:11	250	284	173ms	36.79μs
	1:12	250	241	223ms	40μs
	1:21	750	735	108ms	28.4μs
	1:22	750	709	113ms	33.6μs

Table 4.3: Delay and jitter when using isolated and bounded parameters

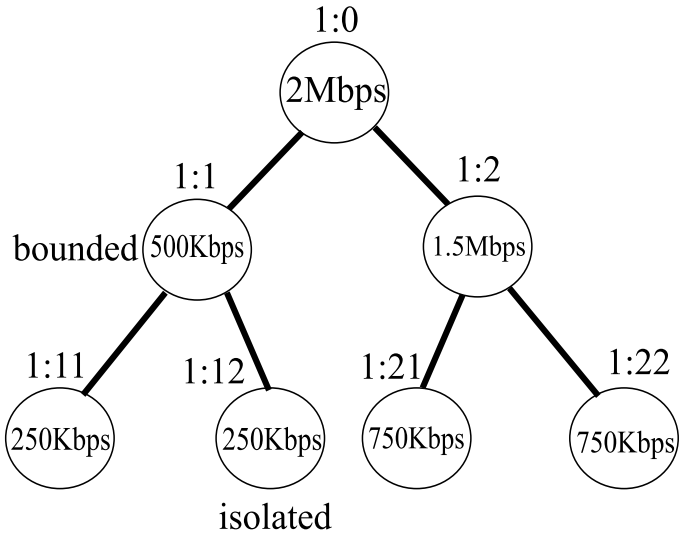


Figure 4.12: CBQ splitting behavior with different number of flows

Isolated parameter of class 1:12 means that the class cannot lend out bandwidth to sibling classes (e.g. class 1:11). *Bounded* low level parameter does not allow to a class to borrow bandwidth from sibling classes (e.g. class 1:2). Table 4.3 shows the impact of delay and jitter when using a hierarchy of classes.

The test with flow1 with CBQ demonstrates that the class 1:1 is bounded to 500Kbps and it does not borrow bandwidth from class 1:2. Class 1:12

is isolated and it does not give to borrow bandwidth for class 1:11. The delay and jitter parameters increase with the number of classes. If class 1:11 sends less than its throughput than classes 1:12, 1:21 and 1:22 use the excess bandwidth. In this case the delay and jitter is decreasing. The QoS conditions for 1:21 and 1:22 classes are improved.

For the Linux CBQ scheduler implementation, several low parameters have major impact for preserving the global service level parameters of the SLA. Low level parameters like *rate*, *maxburst*, *minburst* influence the service level parameters. Using *prio* low level parameter the service level parameters of the class with high priority are not perturbed in case of congestion. The link-sharing hierarchy created using *bounded/borrow*, *isolated/sharing* influences the service level parameters behavior.

The next section presents the low level parameters for queue management mechanisms. The management mechanisms considered are: Random Early Detection and Token Bucket queue management.

4.2.3 Queue Management Mechanisms

Random Early Detection (RED)

One important active queue management is *random early detection*, better known as RED [113]. Under RED, a router probabilistically drops an arriving packet even though the queue for the appropriate outbound interface is not full. The motivation for this is " early "drop comes from the fact that packet loss is the primary indicator of congestion for a TCP connection. By dropping packets before a router's queue fills, the TCP connections sharing the queue will reduce their transmission rates and (ideally) ensure the queue does not overflow. The claim (borne out by significant empirical data) is that dropping packets prior to the overflow of the queue will reduce the overall rate of the packet loss.

Figure 4.13 explains the mechanism of RED. When a packet arrives at the queue, if the weighted average queue length is less than a minimum threshold value, no drop action will be taken and the packet will be simply enqueued. If the average is greater than minimum threshold but less than a maximum threshold, an early drop test will be performed. An average queue length in the range between the thresholds indicates some congestion has begun and flows should be notified via packet drops. The early drop action in the RED algorithm probabilistically drops the incoming packet when the average queue length is between minimum threshold and maximum threshold. If the average is greater than the maximum threshold value, a forced drop operation will occur. An average queue length in this range indicates persistent congestion

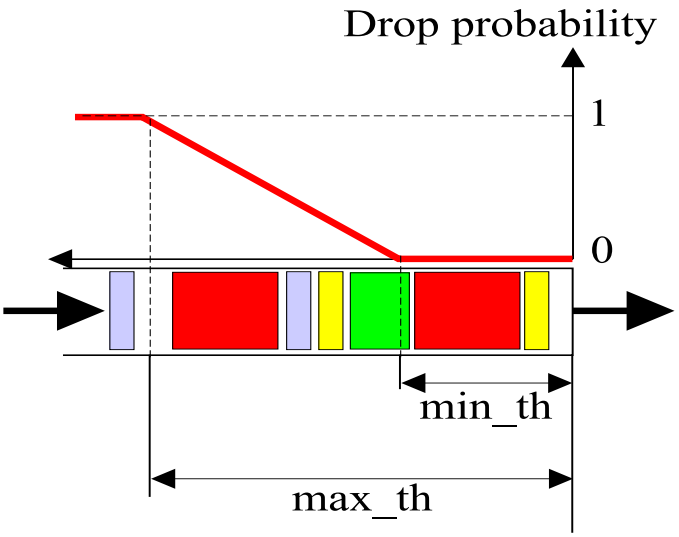


Figure 4.13: Dropping packet rate and minimum threshold using two TCP flows

and packets must be dropped to avoid a persistently full queue. The forced drop is also used when the queue is full but the average queue length is still below the maximum threshold.

Linux Random Early Detection Queuing Management

As we can see in table 4.4 Linux RED queuing management mechanism has several low level parameters/network parameters.

In the next subsection we study the behavior of low level parameters looking on the service level parameters.

Low Level Parameters Behavior

The low level parameters presented in table 4.4 are provisioned and tuned to respect the service level parameters defined in SLAs. We assume that using several approximations there is a simple way to provisioning the algorithm in function of the service level parameters (e.g., delay, bandwidth).

The latency is calculated by setting the minimum threshold and multiplies it by the inverse of the link bandwidth. Setting the minimum threshold too small will degrade throughput and too large will degrade the latency. The RED mechanism works effectively most when the difference between

Low Level Parameter	Definition
limit	hard limit on real queue size [bytes];
min	average queue size at which marking becomes a possibility;
max	maximum threshold where the marking probability is maximal;
probability	the maximal probability for marking;
avpkt	average packet length [bytes];
burst	parameter used for determining how fast the average queue size is influenced by the real queue size;
bandwidth	this rate is used for calculating the average queue size after idle time;

Table 4.4: Low level parameters in RED for Linux

maximum throughput and the minimum throughput is larger than the typical increase in the calculated average queue size in one roundtrip time. A useful rule-of-thumb indicated by Sally Floyd [114] is that the maximum buffer threshold should set at least twice the minimum buffer threshold to prevent synchronization phenomena.

The burst parameter sets the maximum number of packets that can "burst through". This parameter should be set larger than ratio between minimum threshold and average packet. Jamal Hadi and Bert Hubert [115] recommended that this parameter to be set using (min+min+max)/(3*avpkt). This expression approximate the burst parameter used to compute the window queue weight in Exponential Moving Average Algorithm.

As is indicated in [114] the probability parameter should be set between 0.01 and 0.02.

We can observe that after provisioning simplifications, the configuration of RED algorithm depends only on how minimum threshold parameter is chosen. The design of RED is such that during the drop phases of the algorithm, high bandwidth flows will have higher number of packets dropped since their packets arrive at a higher rate than lower bandwidth flows. However, all flows experience the same loss rate under RED. By using probabilistic drops, RED maintains a shorter average queue length, avoiding lockout and repeated penalization of the same flows when a burst of packets arrives. The latency and the packet loss service specifications are depended by the choosing of minimum threshold parameter. The loss rate (or dropping packet rate) shows the behavior of the RED queue.

We assume that increasing the minimum threshold, the dropping rate

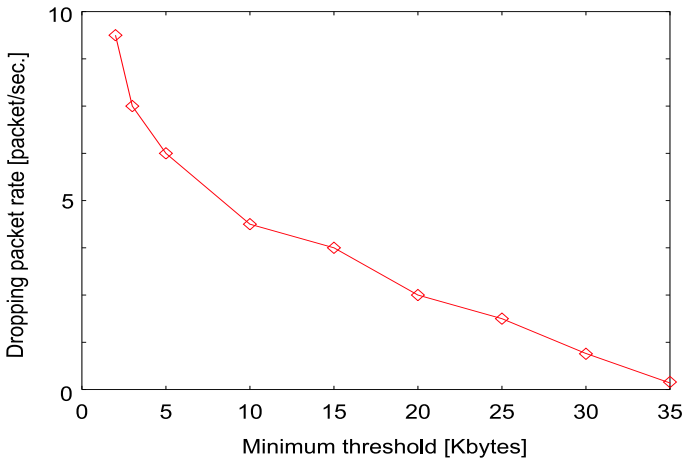


Figure 4.14: Dropping packet rate and minimum threshold using two TCP flows

decreases for the same TCP throughput. We use the same test-bed network presented in figure 4.1. The RED queue is installed in CR1 router. We generate two TCP flows from HG1 to HR1. The window size was fixed at 64Kbit. The measurements are started from min = 3Kbytes. Values under this bound are not working with the Linux implementation. Figure 4.14 shows that the dropping rate decreases when the minimum threshold has high values. The explanation is that the load of the queue became small. In this case the latency increases when minimum threshold parameter has big values. The response of the TCP flows to reduce their traffic in this case is very slow.

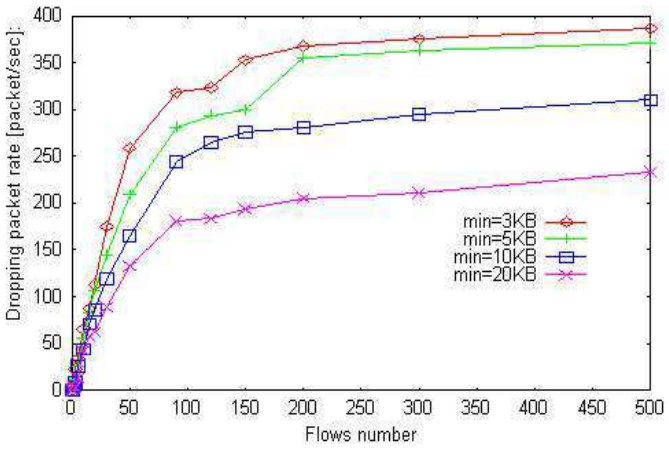


Figure 4.15: Dropping packet rate and flows number

We assume that the dropping rate is influenced by the number of TCP

flows and the minimum threshold values. Using the same test-bed from figure 4.1, TCP flows are sent to the Linux RED mechanism installed in CR1 router. For each minimum threshold value we vary the number of TCP flows. Figure 4.15 shows that for different minimum threshold values exist different dropping packet rate graphics. We observe that for big number of flows the dropping packet rate is almost constant. The explanation is that the queue load is over 100% and the RED mechanism is working with two types of drop actions: early drop action and forced drop action.

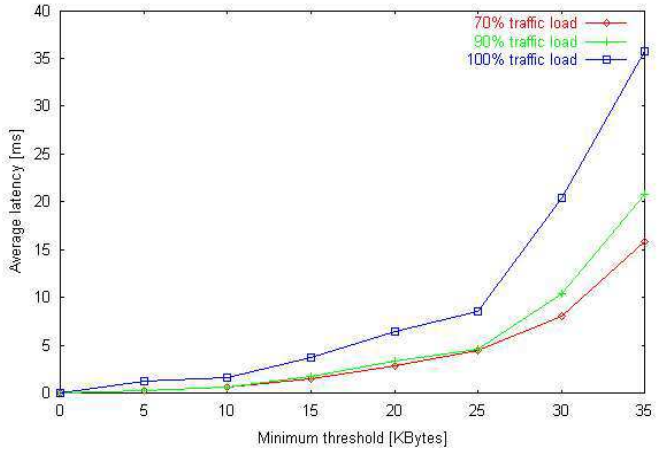


Figure 4.16: Delay and minimum threshold using different traffic loads

The minimum threshold low level parameter influences the delay service level parameter. We assume that for different traffic load values the average latency of the RED is changing. We use one RED Linux queue and we vary the TCP traffic load. The maximum bandwidth between CR2 and CR1 routers is 6Mbit. Figure 4.16 proves that the maximal value of minimum threshold introduce latency. The goal of this experiment is to see how varies the latency when we set the minimum threshold parameter with different traffic loads.

To conclude, the approximated low level parameters *burst*, *maximum threshold* depend on the one low level parameter *minimum threshold*. That simplifies the provisioning operation of mechanism. Service level specifications are influenced by the low level parameter *minimum threshold*.

Token Bucket mechanism

Another of the important queuing mechanism used to implement QoS control mechanisms is Token Bucket. We are using it in chapter 5 for policing the

traffic. In this subsection we focus on analyzing the low level parameters of the mechanism.

Token Bucket is a queuing mechanism that only passes packets arriving at a rate which not exceeding some administratively set rate, but with the possibility to allow short bursts in excess of this rate. In the token bucket metaphor, tokens are generated at a certain rate; the bucket has a specified capacity called size or burst size. If the bucket fills to capacity, newly arriving tokens are discarded. Each token represents permission to send a certain number of bits into the network. To transmit a packet, a number of tokens corresponding to the packet size must be remove from the bucket.

If there are not enough tokens in the bucket, the packets exceed the particular rate limit given by the token bucket. If the bucket is already full, incoming tokens overflow and are not available for future packets. Thus, at any time the largest burst a source can send into the network is roughly proportional to the size of the bucket [90].

A token bucket mechanism is used for shaping traffic or to police the traffic. A token bucket mechanism used for traffic shaping has both a token bucket and a data buffer, or queue; without data buffer it would simply be a policer. Arriving packets that cannot be sent immediately are delayed in the data buffer, thus shaping the traffic. In this subsection, conforming packets are simply sent into the network, non-conforming packets are dropped, thus policing the traffic.

A token bucket permits burstiness within bounds. It guarantees that the burstiness is bounded so that the source will never send much faster than the token bucket's capacity. It also guarantees that the long-term transmission rate will not exceed the established rate at which tokens are placed in the bucket.

Linux Token Bucket Mechanism

For token bucket Linux implementation there are three basic parameters. *rate* is the rate the bucket refills with tokens - which represents the average transmission rate of a traffic flow. The *bucket size* or *burst size* is the number of tokens that the token bucket can store. Linux uses a byte-count token bucket, so every token is equivalent to one byte. The *limit* low level parameter is the sum of the bucket size and the size of queue. If *limit* is equal to the bucket size the queue size is zero and non-conforming packets are dropped, thus the stream is policed. If *limit* is greater than in the *bucket size* some non-conforming packets are queued (the stream is shaped). The token bucket mechanism implemented in Linux is shaper and policer (dropper) all in one. It is based on a token bucket technique to do metering and according to the

value of the *limit* low level parameter it shapes or polices the traffic.

Obviously there is a relation between *limit* low level parameter and latency. In case of policing (limit = bucket size) there is no latency, in case of shaping action, maximum delay is estimated as latency =(limit-burst)/rate. It is possible that the measured value be not the same with the value computed with mentioned formula, caused by the implementation of traffic controller Linux module.

Low Level Parameters Behavior

We assume that the low level parameters of the Token Bucket Linux mechanism have a major impact for the service level specification parameters. Service level specification parameters can not be respected for short period of times cause of the traffic bursts. The role of low level parameters of the Token Bucket is to realize the function of policing or shaping. We use the test-bed presented in figure 4.1. We study the queuing mechanism for traffic policy configuration and for traffic shaping configuration.

We consider that the policy configuration of the Token Bucket mechanism has different behavior in function of traffic type and burst sizes. We generate traffic from HG1 and we policy in the CR1 router.

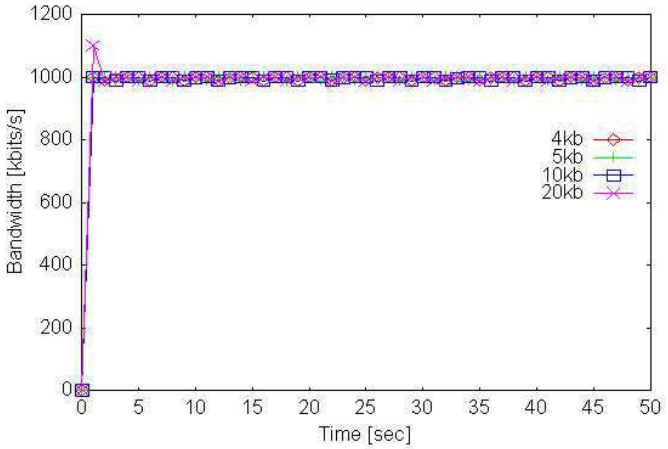


Figure 4.17: Policing - rate 1Mbit/s, different burst sizes using UDP packets

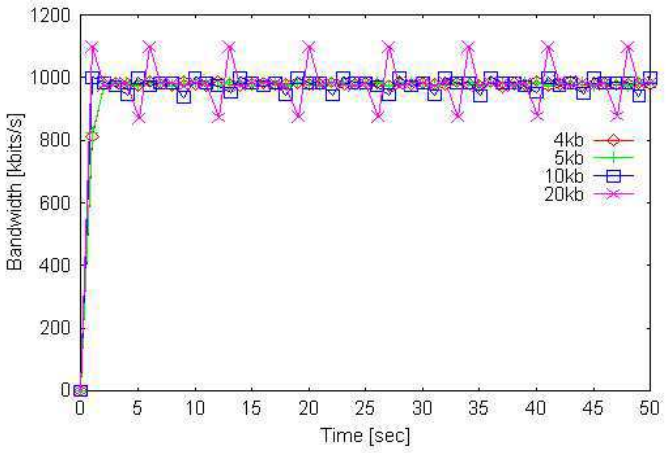


Figure 4.18: Policing - rate 1Mbit/s, different burst sizes using TCP packets

Figure 4.17 and 4.18 shows for various bursts size that the average transmission rate 1 Mbit/s is achieved exactly and the peak at the beginning visualizes the initial burst.

It is clear that UDP packets dropped by the Token Bucket mechanism are really lost, whereas the dropped TCP packets are retransmitted. Choosing smaller burst size parameter causes problems because the theoretical rate of 1Mbit/s can not be reached due to TCP mechanisms that reduce retransmission rate when packet delay or loss occurs. For small values of the burst size parameter it is very hard to police a TCP transmission exactly to a specified rate. Also is very hard for delay and packet loss estimations.

We assume that there is a different behavior of the shaping and policing configuration for the same traffic type. We compare these two configurations.

As we said the *limit* low level parameter for the shaped transmission is greater that the burst size, which means packets that do not obey the Token Bucket meter are queued, so the traffic stream is shaped.

We can observe in figure 4.19, that the shaped TCP traffic flow has a different behavior when there is available rate, whereas the flow policed with exactly the same rate and burst parameter is not able to do so.

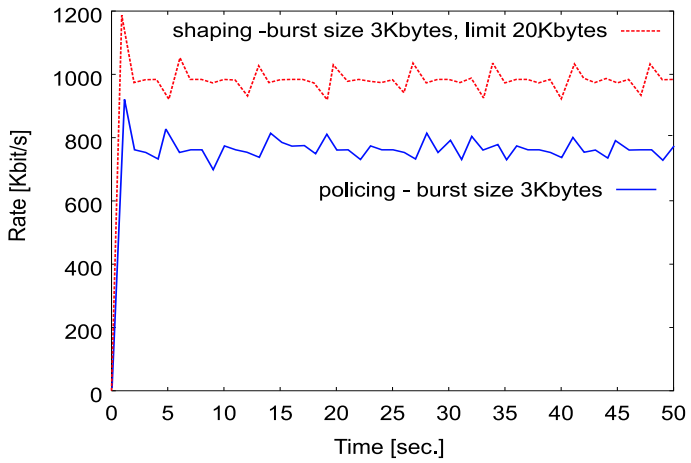


Figure 4.19: Comparison between shaping and policing for TCP - rate 1Mbit/s, burst size 3Kbytes

Looking on figures 4.17 and 4.18 there is visualized the peak burst of the initial burst. This burst can be propagated if it is not possible to guarantee a specific rate. The Linux token bucket implementation can guarantee a specific rate that will not be exceeded (called *peakrate*). The *burst size* low level parameter states how many packets (or how many bytes) can flush through, but it says nothing about the time needed to do so. If we have a source with 1Mbit/s and a token bucket contains tokens equivalent to 20Kbytes, the full burst of 1Mbit/s is propagated for about 146ms and after that the token bucket throttles to the configured rate. In Linux Token Bucket mechanism the traffic stream is really bounded to a specific upper limit rate: setting the *burst size* to a very small value. This means, setting *burst size* to be equal to the maximum transfer unit (MTU). Adopting this solution the bursts are not propagated anymore.

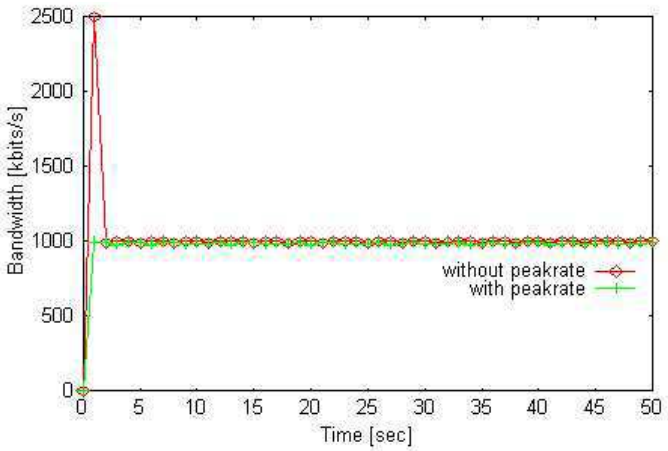


Figure 4.20: Using and not using peakrate parameter

Even there is still only one queue the traffic has to obey to the Token Bucket mechanism. A packet is checked by two conditions: one of them controls the steady state burst size and another uses a MTU-sized token bucket to limit bursts to an upper-bound rate at a smaller time scale. If it is not conform to both conditions it is dropped or queued. So, the formula to calculate the maximum latency due to queuing, has to be modified: latency = (limit -MTU)/rate. Figure 4.20 shows that the case using *peakrate* low level parameter is shaping the bursts to 1Mbit/s.

The both configurations (policing or shaping) of Linux Token Bucket are given by the low level parameters. In the policy configuration service level parameters could be affected by various bursts. Using *peakrate* low level parameter the service level parameters can be protected.

As we have observed, provisioning and tuning the low level parameters in function of various traffic conditions preserve the service level specification set in each SLA. The next section shows two cases, when the state of the network is changed for short period of time and when the service level parameters are changed by the customer.

4.3 Preserving Service Level Specifications

Service Level Specifications (SLS) are included in Service Level Agreements (SLA). It is necessary first to determine the resources that must be provisioned at each network element according to the SLA. Once the resources required for SLA at each network element have been determined, provisioning operation consists of installing or configuring interfaces of the appropriate capacity using the low level parameters/network parameters presented in sec-

tion 4.2. The main operation for an Internet Service Provider is to determine the resources for a new SLA, tuning the network parameters in a way to not damage the current SLAs. The new SLA could correspond to new needs of the customer.

For short periods of time the network resources can not follow the traffic variations. The low level parameters are not adapted to the network state violating the service level specifications of the customers. In this case the network resources are used not efficiently.

We have identified three network states: uncongestion, light congestion and severe congestion. Uncongestion network state means that exists more resources than there are used. In this case low level parameters are provisioned and tuned respecting the service level specifications of the customers. Light congestion network state appears when all the traffic sources filling the resources but the traffic guaranteed is still respected. Short periods of traffic congestion are damaged the service level specifications for this case. When the guaranteed traffic cannot be respected than the network has a severe congestion, and the SLA should be re-negotiated with the customers, or some policy rules should be enforced to decrease the traffic load.

We are using the test-bed environment presented in section 4.1. Using NIST link emulator we set the link between CR1 and CR2 routers to C = 2Mbps. We use a Diffserv configuration for router CR1 that include all the mechanisms presented in section 4.2. We consider a Diffserv router with two AF services (AF1 and AF2), in addition to the classical BE service and EF service. A customer asks the network for bandwidth of service classes (AF1 and AF2) but it may also ask the network for a simple BE service or an expensive EF service. The packets are marked on the output hosts HG1, HG2, and HG3. EF service model minimizes the delay and jitter and guarantees the bandwidth. AF service model guarantees a minimum bandwidth with AF1.1 class. The traffic of AF1.2 and AF1.3 classes is sent when the network congestion's state is diminished. BE service model has no network parameters guaranteed.

We use for traffic sources, RealPlayer and FTP applications. The traffic load is assured by the Iperf generator [106].

The implementation of the three services is done in Linux using two schedulers and two type of queuing management. In Appendix B there are presented the details of this implementation.

First case presents the utility to tune the low level parameters/network parameters of EF queue and of BE queue. In this case we modify *rate* parameter of Token Bucket algorithm of EF class and the *weight* of BE class. We take the case when the state of the network is changed for short period of time. Initially EF service class has three aggregated RealPlayer

Flows	AF class	Class Capacity (kbit/s)	Thr. Offer (kbit/s)	Bandwidth (kbit/s)	Delay (ms)	Jitter (ms)	Loss packets
EF	-	250	256	248	8.89	1.74	0
AF1	1	125	48	47.9	41.8	31.7	0.02%
	2		48	46.1	46.2	31.	0.096%
	3		48	30.4	49	35.5	0.23%
AF2	1	125	48	47.9	40.8	28.7	0.07%
	2		48	46.3	45.2	29.5	0.15%
	3		48	30.9	47.8	34.1	0.43%
BE	-	1200	1600	1187	128	10.5	4.6%

Table 4.5: Initial conditions for first case

flows. Table 4.5 shows the initial conditions of the experiment. The offered traffic is greater than the total class bandwidth.

The aggregate traffic for EF is defined by the RealPlayer flows (UDP packets). The aggregate traffic for AF1 has TCP long connections flows and aggregate traffic for AF2 has various types of connections (short and long). BE offered traffic is based on UDP packets.

If a new flow of RealPlayer (UDP packets) with guarantee bandwidth 20Kbit/s is accepted by the CR1 router, the delay for EF aggregation flow increase at 38.2ms, which is tolerated by all three flows. If the flow has 128Kbit/s the delay increases at 1.2s which is not agreed by the adaptive levels of the application. In this case the service level specifications for all flows are violated. A new provisioning of the EF class must be done to accept the third flow. Because BE class has no guarantee level and the amount of guaranteed levels of EF, AF1 and AF2 classes is under the links capacity, we choose to decrease the bandwidth of BE to accept the new flow. We provision the *rate* and *weight* low level parameters with new values that respect all the service level specifications.

Table 4.6 shows the improvements of the EF class if its bandwidth has increased. The delay and the jitter have values, which are tolerated, by the application level. After tuning we achieved the guaranteed bandwidth level and minimum of delay and jitter.

The second case demonstrates the utility for tuning low level parameters/network parameters when the service level parameters are changed by

Flows	AF class	Class Capacity (kbit/s)	Thr. Offer (kbit/s)	Bandwidth (kbit/s)	Delay (ms)	Jitter (ms)	Loss packets
EF	-	350	384	349	14.46	4.94	0
AF1	1	125	48	47.9	40.2	29.7	0.025%
	2		48	46.1	47.2	32	0.1%
	3		48	30.4	48	36.5	0.37%
AF2	1	125	48	47.9	40.8	28.7	0.08%
	2		48	46.3	45.2	32.5	0.19%
	3		48	30.9	47.8	39.4	0.53%
BE	-	1100	1600	1126	210	18.5	20.6%

Table 4.6: Values used and monitored after tuning process

Flows	AF class	Class Capacity (kbit/s)	Thr. Offer (kbit/s)	Bandwidth (kbit/s)	Delay (ms)	Jitter (ms)	Loss packets
EF	-	250	256	249	9.78	6.84	0
AF1	1	125	100	47.3	93.76	104.7	56.7%
	2		60	49.6	70.6	82.5	9.8%
	3		40	41.3	50.5	42.3	0.68%
AF2	1	125	20	20.1	40.8	28.7	0%
	2		20	21.3	45.2	33.5	0%
	3		20	19.89	49.8	22.6	0%
BE	-	1525	1700	1496	167	40.6	12.6%

Table 4.7: Initial conditions for second case

the customer. The guaranteed level for AF1 and AF2 is based on AF1.1 and AF2.1 throughput (we use three color marker at the edge routers). AF1.2 and AF2.2 classes offers guaranteed of certain bandwidth but with a percentage of packet loss when the network is hard loaded. AF1.3 and AF2.3 classes offer no guaranteed (e.g. Three Colors Marker).

This scenario use the initial configuration described in table 4.7. The *min* low level parameter of each AF1 classes of service is set initially to 48Kbps. The initially service level specification proposed is provision the low level parameters. A FTP flow of 120Kbit/s is aggregate in AF1 class of service. The bandwidth guaranteed for this flow is 60Kbit/s. The load of AF2 class is very low. The guaranteed bandwidth for AF2 class is satisfied. In this case we can observe the amount of guaranteed throughput AF1 is more than the provisioned guaranteed value of the AF1 class. FTP flow

Flows	AF class	Class Capacity (kbit/s)	Thr. Offer (kbit/s)	Bandwidth (kbit/s)	Delay (ms)	Jitter (ms)	Loss packets
EF	-	250	256	249	10.32	6.74	0
AF1	1	200	100	102.3	45.2	29.7	0.025%
	2		60	58.6.	50.2	32	0.1%
	3		40	39.2	53.4	36.5	0.37%
AF2	1	60	20	21.3	47.9	28.7	0.093%
	2		20	20.8	48.3	31.5	0.24%
	3		20	20.4	43.9	36.7	0.58%
BE	-	1490	1700	1453	234	35.5	30.6%

Table 4.8: Values used and monitored after tuning process

losts connection after 10.45s of delay cause of this non-adaptability of the throughput. The amount of guaranteed levels of AF1 and AF2 classes is less that link bandwidth (C=2Mbit/s). The customer can request a new service level specification for AF1 service class.

The service provider is to reduce the bandwidth of AF2 class and give it to AF1 by modifying the *min* low level parameter.

After tuning process FTP flow has 60Kbit/s firm guaranteed bandwidth. In this case the FTP connection is reliable. BE service has decrease his throughput because it has no guarantee bandwidth and the free bandwidth of AF2 class is not enough.

A severe case is when the amount of guaranteed throughputs is more than link capacity. The Internet Service Provider must re-negotiate all the SLAs following certain policies specific of his network.

This tuning of the low level parameters/network parameters based on provisioning and monitoring is time consuming and if the state of the network is varying vary fast than especially the provisioning operation is useless. The tuning has to be done so each class has to ensure that the BE traffic is not penalized by the AF and EF traffic and at the same time, that BE traffic does not consume more than its fair share of the available bandwidth.

4.4 Conclusions

The performance of service level specification parameters depends on the low level parameters/network parameter management. We tested QoS mechanisms implemented in Linux system in order to select the parameters that

have importance for the QoS management level. Our contribution is to analysis interaction between service level parameters (bandwidth, delay, jitter, packet loss) and the low level parameters/network parameters (parameters of QoS mechanisms implemented in Linux).

Priority queuing scheduling mechanism has no network parameters to tuning that can influence the service level parameters. The latency of PQ scheduling mechanism depends on the buffer size implementation and the speed of the CPU.

Class Based Queuing scheduling mechanism is the most complex mechanism of QoS Linux implementation. The goal of implementation is to do shaping and to schedule the output access of each class using WRR (Weighted Round Robin). The delay estimated for each class has two components: queuing delay and elected time. Elected time depends on the number of queues per scheduler. The queuing delay depends of the IP packet size, hierarchy of link-sharing (e.g., bounded, isolated), buffer size (e.g., maxburst, minburst) and the throughput of each class (e.g., rate, weight). Using parameters *bounded/borrow*, *isolated/sharing* the network administrator can create policy rules using a hierarchy of bandwidth sharing.

The provisioning of Random Early Detection queuing mechanism is done by one parameter that is minimum threshold. The packet delay is estimated using minimum threshold parameter.

Token Bucket queuing mechanism has two functions: to policy and to shape the traffic. Using Token Bucket low level parameters we can estimate the latency.

Another contribution is to demonstrate in a test-bed network with real traffic the necessity of changing low level parameters in order to respect the service level specifications. The first case shows that when the state of the network is changed for short periods of time we need to tune the low level parameters. The second case evaluates the reasons to change the AF class network parameter when a customer changes his SLA. In both cases tuning the low level parameters is time expensive and requires many tuning operations in the network. For that it is necessary to develop automated, self-configurable mechanisms for simplify the QoS Management.

Chapter 5

Efficiency of using Dynamic QoS Management

5.1 Introduction

The performance of a DQM system is strongly dependent on how well parameters mentioned and analyzed in chapter 4 work well in the edge and core network routers. Dynamic QoS Management system should avoid the network instability, optimize the goodput and avoid small congestion transitions of the network. This chapter proposes a dynamic, self-tuning mechanism for core routers but which could be adapted also for the edge routers, to fulfill the characteristics of DQM mentioned above.

This dissertation presents the drawbacks of Static QoS Management, and motivates the Dynamic QoS Management. The Dynamic QoS Management is characterized by scalability and complexity. It is ideal to have simple and scalable QoS control mechanisms for DQM. Diffserv framework motivates the usage of automated and adaptive control mechanisms respecting the scalability and the complexity of the management system. This chapter provides a network service model based on Diffserv framework. The network service model is provided in the core network by using automated QoS control mechanisms and simple QoS provisioning methods. We propose a flexible, automated mechanism for bandwidth allocation for core routers. It tunes dynamically routers to allocate efficiently the available resources among Diffserv classes. The mechanism proposition is validated by a campaign of experiments on real network test-bed. We assume that for the hierarchies of classes the link-sharing bandwidth mechanism improves the self-tuning operation of low level parameters of the QoS mechanism proposed in [117].

We study the impact of CBQ mechanism related to bandwidth borrowing

on the performance of our mechanism. Both mechanisms when are used together lead to better overall performance due to the bandwidth borrowing capabilities of scheduler (in our case CBQ). A set of experiments validates the interaction between CBQ and our mechanism [118].

We show that a Diffserv network which include dynamic resource allocation mechanism is stable and has a good behavior with regard to the small congestion transitions of the network.

The remainder of this chapter is organized as follows. In section 5.2 we present the context of our work. The problem description is presented in section 5.3. The solution of our problem is presented in section 5.4. We develop in section 5.5 simple scenarios to compare static and dynamic bandwidth allocation schemes. The solution of our problem is validated in section 5.6 with several experiments. In section 5.7 we study the interaction of our proposition with CBQ (Class Based Queuing). We summarize our work in section 5.8.

5.2 Routers configuration in the core network

Network customers need guaranteed level of service from their Internet Service Providers (ISPs) to achieve their business objectives. Service Level Agreements (SLAs) between providers and customers define service level specifications [18] with traffic conditioning specification [119], monitoring service capability [120], service availability and the fees corresponding to each level. The traffic conditioning specification (TCS) defines service level parameters (bandwidth, packet loss, peak rate, etc.), offered traffic profiles and policies for excess traffic. Given the SLA of a stream of packets, the network has to allocate enough resources so that the *service* required by this stream is *guaranteed*. The allocation can be done in different ways, depending on the total amount of resources available in the network and the number of customers asking for a guaranteed service. One simple allocation is the one that uses the peak rate of traffic. This is the type of allocation supported by PSTNs (Public Switched Telephone Network). Another possible allocation is the one that uses the notion of *effective bandwidth* [92]. The effective bandwidth for traffic is the minimum bandwidth to be allocated in the network so that the probabilistic needs of the traffic (e.g., packet loss rate, tail of packet delay) are satisfied. In the effective bandwidth framework, the statistical multiplexing of the different streams of packets is used to minimize the total amount of resources to be allocated. Note that for any resource allocation scheme, there is always a maximum limit on the number of customers the network can support. When the number of customers ap-

proaches this limit, the resources start to be rare and the Quality of Service (QoS) required by customers cannot by realized. Thus, some kind of Call Admission Control (CAC) [121] has to be implemented by the network, to protect already accepted customers from newly arriving ones.

As we have shown in Chapter 2, Differentiated Services (Diffserv) is an IETF framework in which the network traffic is classified into classes, with different service level for classes. The edge routers in a DiffServ network mark/shape/police flows based on their SLAs, and the core routers offer packets belonging to these flows different treatments using the marks they carry. For our work we consider that a flow is a stream of packets belonging to the same SLA. Core routers handle aggregates of flows instead of individual flows, which is known to considerably reduce the complexity of DiffServ, compared to its counterpart IntServ [9], where core routers allocate resources on a per-flow basis. The treatment a core router gives to packets from one service class is called PHB (Per Hop Behavior). The PHB classes (or service classes) defined in DiffServ are: Best Effort (BE), Assured Forwarding (AF) [83] and Expedited Forwarding (EF) [82]. The EF class is designed to support real time flows with hard delay and jitter constraints. The AF class is designed for flows only asking for bandwidth, mainly TCP flows. The AF has four (sub)classes, each one with three drop precedence [8]. At the onset of congestion, core routers start drop AF packets with the highest drop precedence, then those with the medium drop precedence, and finally if congestion persists, packets with the lowest drop precedence are dropped. Packets within a AF class are served in core routers in order. The different AF classes may differ in the applications and transport mechanisms they support. For example, TCP traffic can be protected from non-responsive UDP traffic by separating both types of traffic in two different AF classes. How to distribute traffic on the different service classes of DiffServ depends on the policy of each ISP.

The performance of a DiffServ network is strongly dependent on how well edge and core routers work. Edge routers are responsible of marking/shaping/policing traffic. Core routers give packets different treatments based on the marks they carry. For example, EF packets are queued in a separate buffer and are served "before" packets of the other classes. Packets of the different AF classes are queued in separate buffers and are served using the CBQ mechanism [113]. We focus in this chapter on the tuning of core routers for AF traffic. This operation reflects the tuning of the weights of the CBQ buffer. The weights of CBQ control the way with which the available bandwidth at the output interface of the core router is distributed among the AF classes and the BE class. The weights do not have any control on the EF class, since EF packets are usually served with a strict priority over

the other types of packets (AF and BE). The tuning has to be done so that each class of traffic realizes its needs, and the resources of the network are efficiently utilized. In particular, the tuning has to ensure that the BE traffic is not penalized by the AF traffic, and at the same time, that the BE traffic does not consume more than its fair share of the available bandwidth.

The tuning of core routers is actually characterized by extensive manual work, based on a trial-and-error process, which is often ineffective, time-consuming and costly to network managers. A static tuning of core routers may result in an inefficient utilization of network resources, and a bias against one or more DiffServ classes. We believe that the tuning of core routers has to be dynamic, so that the available bandwidth is efficiently utilized and fairly shared among the different DiffServ classes. Moreover, the tuning has to be automatic, self-configurable, easy to deploy and to manage, without any additional signaling. In this chapter, we motivate the dynamic tuning of core routers and we present a mechanism that respects the above rules. We implement our mechanism in Linux and we validate its performance by a campaign of experiments on a real network testbed. While presenting our mechanism, we will make the assumption that the network is over-provisioned, i.e., there are enough resources in the core of the network to support the high priority traffic marked at the edge. On the other hand, the resources of the network may not be enough to support the total amount of traffic coming from the edge, i.e., the high priority plus the low priority traffic. The decision on whether to accept a new SLA is done at the edge of the network, and core routers dynamically tune their parameters so as to absorb the marked traffic and to utilize efficiently the network resources. Our mechanism can be easily extended to the under-provisioning case, by making some default assumption on the distribution of the bandwidth among the DiffServ classes. For example, a core router may give the highest priority to EF packets, and set equally the weights of the CBQ buffer for AF classes, with zero or a minimum amount of bandwidth to the BE class. We believe that the under-provisioning case is undesirable for the ISP and the customers and has to be avoided, which can be done by implementing CAC at the edge, or by renegotiating the SLAs of active customers.

We consider the allocation of bandwidth based on the average rate of high priority traffic of each DiffServ class. For simplicity of the analysis, we omit the EF service and we focus on AF and BE. EF is usually handled by a priority queue, without any special tuning to be done by the core router. Without loss of generality, we focus on a AF service with two drop precedence per-class. This latter service has been first introduced in [8]. At the edge, compliant packets of an AF class are marked with high priority (called IN packets), and non-compliant packets are injected into the network with low

priority (called OUT packets). IN and OUT packets are buffered in the same queue in core routers, but are dropped differently at the onset of congestion. The idea is to start dropping OUT packets while protecting IN packets. When all OUT packets are dropped and the congestion persists, IN packets start to be discarded. The mechanism proposed in [115] to support such a preferential dropping is called RIO (RED IN/OUT).

In summary, our mechanism measures the rate of IN packets, and sets the parameters of the CBQ buffer so as to absorb all IN packets and to distribute fairly the rest of the bandwidth (called excess) among OUT and BE packets. As a case study, we consider a max-min fairness for the allocation of the excess bandwidth [122][123].

5.3 Problem Description

The tuning of core routers in a Diffserv network is actually done in a static way by manual work based on a trial-and-error process. A static tuning is time-consuming and costly for the network managers. Moreover, as we can see in Section 5.4, a static tuning may lead to an inefficient utilization of network resources and to an unfairness among the DiffServ classes. A dynamic tuning of core routers, or equivalently a dynamic allocation of resources in the core of the network, is needed to satisfy as much as possible all reservations of customers, and to fairly distribute the excess of bandwidth among them. Consider a DiffServ network proposing to customers two AF services (AF1 and AF2), in addition to the classical BE service. This will be the type of networks we will study throughout the chapter 5. Our results hold in the case when more than two AF service classes are considered. They also hold in the case the network operator proposes the EF service to its customers.

A customer asks the network for some bandwidth of some class (AF1 or AF2)(Figure 5.1). It may also ask the network for a simple BE service, with no bandwidth guarantee. As mentioned in section 5.1, the difference between the two AF classes can be in the transport protocol used (TCP vs. UDP), in the number of customers authorized to join each class, in the policy applied to non-compliant packets, etc. The bandwidth allocated to a user of a AF class represents the maximum rate of IN packets the customer is allowed to inject into the network. All packets non compliant with the contract signed between the customer and the operator are marked as OUT at the edge. Some routers at the edge may choose to drop OUT packets instead of injecting them into the network. This is what we call policing of flows.

Let $R_{AF1.1}$ (resp. $R_{AF2.1}$) denote the rate of data carried by IN packets

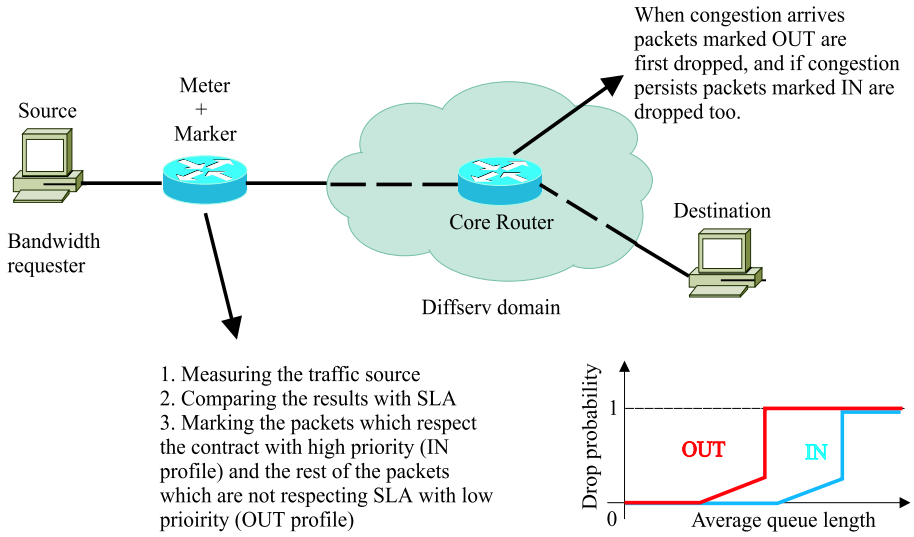


Figure 5.1: Class AF model

of class AF1 (resp. AF2), which arrive at a core router and are destined to the same output interface. Let C be the total bandwidth available at the output interface of the core router. We are working in an over-provisioning case, which means that there are enough resources to absorb the high priority traffic. This can be written as,

$$R_{AF1.1} + R_{AF2.1} \leq C \tag{5.1}$$

Now, denote by R_{AF1} (resp. R_{AF2}) the total rate of data carried by AF1 packets (resp. AF2 data) (IN + OUT). Denote by R_{BE} the data rate of the BE traffic arriving at the core same router and destined to the same output interface as the AF1 and AF2 traffic. Dynamic tuning is only interesting in the case when the core router is congested. Core routers being work-conserving, the tuning of parameters does not impact the QoS perceived by customers in a non-congestion case. Our assumption is then,

$$R_{AF1} + R_{AF2} + R_{BE} \geq C \tag{5.2}$$

We consider that a customer is satisfied if its IN packets get through the network. We also consider that it is in the interest of customers and operator that the excess of bandwidth in the network is fairly distributed among the three classes: AF1, AF2, and BE. The excess of bandwidth is simply equal to C - $R_{AF1.1}$ - $R_{AF2.1}$. Given the variability of the traffic and the change

in SLAs, it is very likely that a static tuning of core routers does not realize the above objectives. The problems that can be caused by a static tuning of weights at the output interface of a core router can be summarized as:

- Unfairness in the distribution of the excess bandwidth;
- Bias against the IN packets of one or more AF classes.

We will study these problems of static tuning in section 5.4 on some real scenarios. We will show the gain that a dynamic tuning of weights can bring. The comparison will be made between static tuning scheme, and our mechanism that dynamically adapts the weights at the output interface of routers, based on the incoming (IN) traffic. Our mechanism is designed with the main objective to realize the above objectives, that is, to accept the high priority IN traffic of the two AF classes, and to distribute the excess bandwidth fairly among the three classes we are considering: AF1, AF2, and BE. We start by explaining this mechanism in the next section.

5.4 Dynamic Resource Allocation Algorithm

Our algorithm has three parts: monitoring, excess bandwidth distribution rule and CBQ scheduler programming.

In the monitoring part, we measure the average data rate carried by the IN packets for each one of the two classes AF1 and AF2. The interval over which the data rate is averaged is an important parameter of our mechanism. It must not be too small, since the system may become unstable, especially when we have transport protocols as TCP that adapt their rates as a function of the reaction of the network. A small averaging interval also results in high computational overhead. The averaging interval must not be very large too, since this slows the reaction of the mechanism to changes in traffic and SLAs. Later, we will come to the impact of this averaging interval on the efficiency of our mechanism.

The excess bandwidth distribution rule part forms the core of the algorithm. With no loss of generality, we use the following rule: the excess bandwidth is split into three parts and equally distributed among the classes. We call this rule *fair division*. Other rules are also possible. For example, one can distribute the excess bandwidth among classes using non-equal weights. This rule is discused in section 5.6. The objective might be to give AF services bandwidth advantages over BE, or the opposite.

After the distribution of excess bandwidth, the algorithm computes the desired class rates for CBQ (the optimal weights for each class) and programs

the CBQ mechanism with the new values. The algorithm has the following description in figure 5.2

while *()* **do**
- Measure the throughput of IN-profile packets for both classes AF1 and AF2, and for each outgoing network interface: $R_{AF1.1}$, $R_{AF2.1}$;
- Compute the amount of excess bandwidth using measured values: $B_w = C - (R_{AF1.1} + R_{AF2.1})$;
- Distribute the excess bandwidth following some rule. Ex: w = $B_w/3$;
- Program the CBQ scheduler using the computed optimal rates: $R_{AF1} = R_{AF1.1} + w, R_{AF2} = R_{AF2.1} + w, R_{BE} = w$;
- Wait x seconds;

end

Figure 5.2: Dynamic Resource Allocation Algorithm (DRAM)

Let us illustrate the operation of our algorithm with the following example, Looking on figure 5.3. We have a scenario where the AF1 service is used by a UDP traffic sending data at a constant rate 4Mbps, and the AF2 and BE services are used by TCP traffic. The TCP traffic is generated by means of long-lived TCP connections. For both service classes AF1 and AF2, the rate of IN-profile data is set to: $R_{AF1.1}$=2.5Mbps and $R_{AF2.1}$=2Mbps. The available bandwidth at the output interface of the core router is equal to C=6Mbps. We start from a case where the bandwidth C is equally divided among the three service classes R_{AF1}= R_{AF2}= R_{BE}=2Mbps. After the first iteration, our mechanism sets: R_{AF1}= 3Mbps, R_{AF2}= 2.5Mbps and R_{BE}= 0.5Mbps. The next iteration measures $R_{AF1.1}$= 2.5Mbps and $R_{AF2.1}$= 2.5Mbps. The algorithm stabilizes then with the following rates for the CBQ buffer: R_{AF1}= 2.8Mbps, R_{AF2}= 2.8Mbps and R_{BE}=0.3Mbps. The convergence is being done in two steps. Clearly, this allocation of bandwidth C satisfies the two objectives of our mechanism. Indeed, $R_{AF1.1}$ and $R_{AF2.1}$ are satisfied, and the excess bandwidth (C - $R_{AF1.1}$ - $R_{AF2.1}$)=1Mbps is fairly divided among the three service classes.

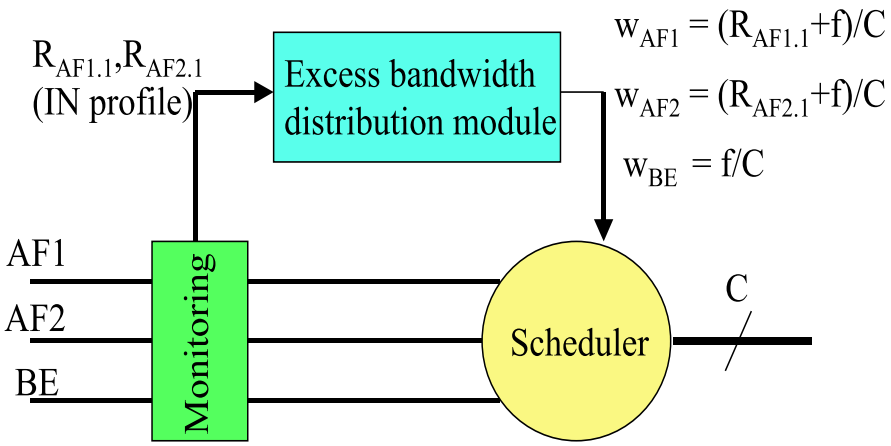


Figure 5.3: Modules of Dynamic Resource Allocation Algorithm

5.5 Comparing static and dynamic bandwidth allocation schemes

The goal of this section is to compare static and dynamic bandwidth allocation schemes using simple scenarios. We show how a static scheme results very likely in an inefficient utilization of network resources, and a bias against one or more service classes. As a dynamic scheme, we use our mechanism that we described in section 5.3.

5.5.1 Unfairness in the distribution of the excess bandwidth

Consider the case when we guarantee 2Mbps for the AF1 service, $R_{AF1.1}$ = 2Mbps, and nothing for the AF2 service, $R_{AF2.1} = 0$. The CBQ scheduler in the core router statically limits each service class to 2Mbps. The total amount of the link capacity is then C=6Mbps. The traffic from one class cannot exceed 2Mbps, even if it is reserving much more bandwidth than the traffic in the other classes. Suppose that the class AF1 wants to send more traffic than is guaranteed (R_{AF1} = 4Mbps), then this class will not obtain any share from the excess bandwidth, since the core router is limiting its maximum rate to the bandwidth it is reserving. The other classes, which do not reserve any bandwidth, monopolize the excess bandwidth, which is completely unfair to class AF1. The weights of the CBQ scheduler are clearly not adapted to this situation.

Our mechanism solves this problem and allows a fair sharing of the ex-

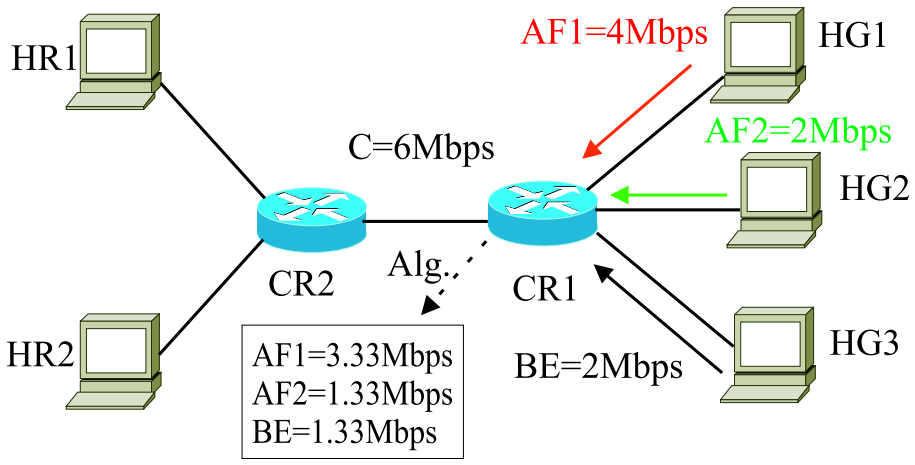


Figure 5.4: Unfairness in the distribution of the excess bandwidth case. HG - host generators and HR - host receivers. CR - core routers

cess bandwidth, which improves the AF1 service in this scenario. The rates allocated by our algorithm are shown in Fig. 5.4. These rates are obtained with a fair division rule, which divides the excess bandwidth equally among the three service classes. Using this rule, we increase the AF1 bandwidth to R_{AF1}=3.33Mbps and we penalize the other services that do not ask for any guaranteed bandwidth, R_{AF2}=R_{BE}=1.33Mbps instead of 2Mbps in the static case.

5.5.2 Bias against the IN packets of one or more AF classes

A customer is satisfied when the IN-profile packets it injects into the network succeed to get through. We recall that we are considering an over-provisioning case, otherwise this condition on satisfaction is not feasible. A static tuning of core routers may violate this condition, by penalizing a class sending only IN-profile packets, while accepting OUT-profile packets from another class or packets from the BE class. We illustrate this misbehavior with the following two scenarios. Our mechanism ensures a complete protection of IN-profile packets in an over-provisioning case. It does this by adapting the rates of CBQ to the rates of IN packets.

a) One Reservation is Not Satisfied and One is Satisfied

Consider the scenario where the AF1 class generates a total data rate of 4Mbps, and the AF2 class generates a total data rate of 0.5Mps, R_{AF1}=4Mbps

and R_{AF2}=0.5Mbps. All packets of the AF2 class are marked as IN, $R_{AF2.1}$ = 0.5Mbps. The rate of data carried by IN packets of class AF1 is equal to $R_{AF1.1}$ = 3Mbps. In the static tuning case, the CBQ buffer is supposed to distribute the bandwidth C=6Mbps equally among the three classes. The BE traffic is set to 3Mbps. In this scenario, class AF2 is satisfied since all its IN packets get through the network. However, class AF1 is not satisfied, since the CBQ buffer limits its rate to 2Mbps, whereas it is sending IN packets at a higher rate of 3Mbps. We are in a scenario where IN packets are dropped, and OUT/BE packets are served instead. The total rate of IN packets generated by the two classes is less than C, therefore it is possible to find another tuning that allows all IN packets to get through, and corrects the bias against class AF1. When running our mechanism, the rate allocated to AF1 increases to more than $R_{AF1.1}$, and the rate allocated to AF2 is kept larger than $R_{AF2.1}$ (Fig. 5.5).

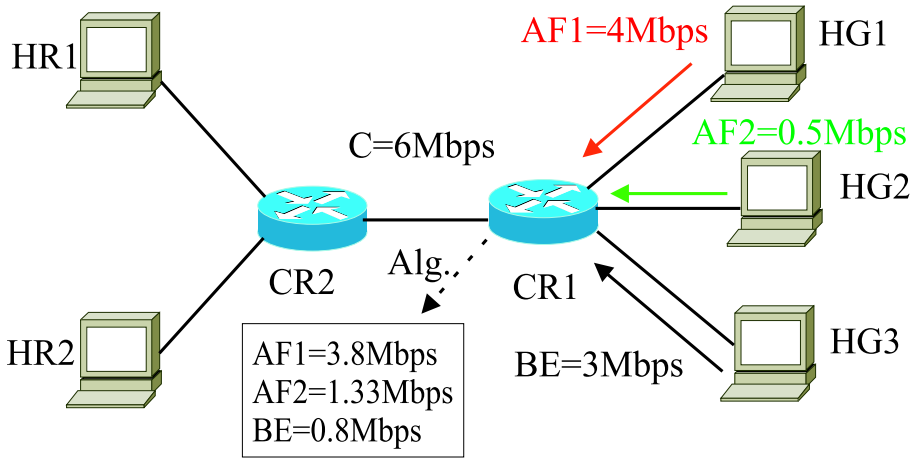


Figure 5.5: One reservation is satisfied from HG2. HG - Host Generator, HR - Host Receiver, CR - Core Router

Concerning the excess bandwidth (equal to 2.5Mbps in this scenario), our mechanism divides it equally among the three classes. So, the new distribution of the CBQ rates is R_{AF1}= 3.8Mbps, R_{AF2}= 1.33Mbps and R_{BE}= 0.8Mbps.

b) All Reservations are Not Satisfied

The last case we consider is when all reservations are not satisfied, due to an appropriate static tuning of CBQ weights. Suppose AF1 has a IN-profile traffic $R_{AF1.1}$=2.5Mbps and AF2 a IN-profile traffic $R_{AF2.1}$=2.5Mbps. If the CBQ limits the maximum rate of each class to 2Mbps, both classes AF1

and AF2 will not be satisfied since their IN packets do not get through the network. On contrary, clients of the BE class are favored, since they obtain more than their fair share of the excess bandwidth (2Mbps instead of 0.33Mbps)(fig. 5.6).

We apply our algorithm and we measure each class throughput at the output interface of the core router. The rule to divide the excess bandwidth is the same as that in the above sections. After a while, the rates allocated in the CBQ buffer to the different classes change: AF1 and AF2 receive R_{AF1}=R_{AF2}=2.8Mbps and BE receives R_{BE}=0.3Mbps. This is exactly the desired allocation, that satisfies customers who ask for bandwidth, and distributes the excess bandwidth fairly among the three classes.

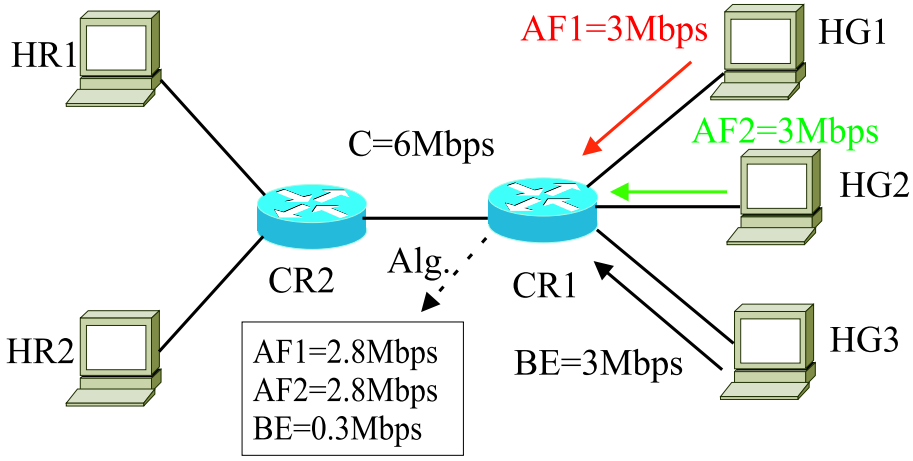


Figure 5.6: All reservations are not satisfied. HG - Host Generator, HR - Host Receiver, CR - Core Router

5.6 Validation of DRAM with Experiments

The evaluation environment includes monitoring tools, traffic generator tools, traffic control tools and link emulator tools. The experimental results are based on cases presented in section 5.4. We use the experimental environment described in section 4.1 of chapter 4. Our mechanism is implemented in the CR1 router. Looking to figure 4.1 each host at the right-hand side generates TCP or UDP traffic flows [106] to hosts at the left-hand side. The traffic is generated with the Iperf tool, and is marked at the output interfaces of the source hosts. The network bandwidth is 10Mbps. Using the NIST Net tool [107], we emulate a link with bandwidth 6Mbps between CR1 and CR2.

	Min.[ms]	Avg.[ms]	Max.[ms]
Without DRAM.	2.108	23.799	68.891
With DRAM	2.023	18.345	56.675

Table 5.1: Delay in AF1.1 service in the distribution of excess bandwidth case

The traffic code is implemented in Linux kernel and is configured with a command interface tc.

5.6.1 Experimental Results for Scenarios presented

Figure 5.7 shows how our algorithm reacts to the unfairness in the distribution of the excess bandwidth (section 5.4.1). We use UDP to generate the AF1 traffic (4Mbps) and multiple long-lived TCP flows to generate the AF2 and BE traffic. The AF2 traffic has only OUT-profile traffic, which is obtained by setting the guaranteed bandwidth of the AF2 class to 0.

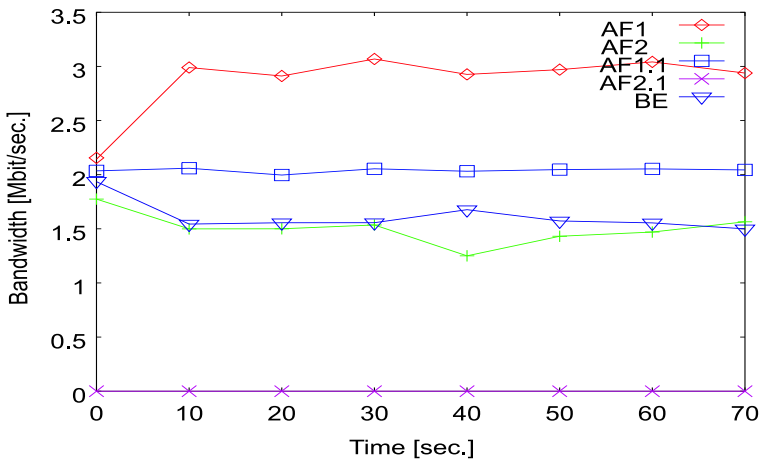


Figure 5.7: Unfairness in the distribution of excess bandwidth

The guaranteed profile for AF1 is $R_{AF1.1}$= 2Mbps and the router receives more than this amount (it receives 4Mbps). After one iteration we observe that AF1 has R_{AF1}=3.33Mbps, AF2 has R_{AF2}=1.33Mbps and BE has R_{BE}= 1.33Mbps. AF1 has realized the guaranteed bandwidth $R_{AF1.1}$= 2Mbps, and is then satisfied. The AF2 class is penalized because it does not ask for any bandwidth in this scenario, so it gets approximately the same throughput as BE.

Table 5.1 shows the impact of our mechanism on the delay perceived by packets of class AF1. Our mechanism improves the quality perceived by this class by giving it a fair share of the excess bandwidth. This additional bandwidth is translated into a smaller end-to-end delay for packets of this class.

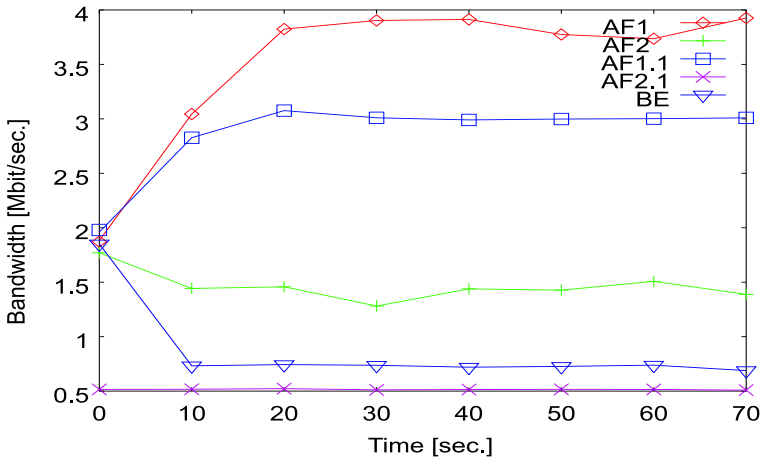


Figure 5.8: One service reservation is not satisfied, one is satisfied

Figure 5.8 shows another case where one of the services (AF2) has low traffic, less than the guaranteed bandwidth and another service (AF1) has more traffic than is guaranteed (section 5.4.2.a). In this case, AF1 has UDP flows and AF2 and BE has TCP flows. The bandwidth guaranteed by the network to AF1 is set to 3Mbps, instead of 2Mbps as before (this follows a change in the contract according to a new agreement between customer and ISP). As expected, after two iterations AF1 gets R_{AF1}= 3.8Mbps, AF2 gets R_{AF2}= 1.33 and BE gets R_{BE}= 0.8Mbps. The priority traffic from each AF service gets through the network. The two AF classes realize their desired rates. The BE service gets the smallest amount of bandwidth because it has zero guarantee.

In this case the delay measured for AF2.1 has the same value with/without our algorithm (min.=1.772ms / avg.=19.708ms / max.=57.876ms).

Figure 5.9 shows the scenario presented in section 5.4.2.b, all service reservations are not satisfied. We used for AF1 class UDP flows and for AF2 and BE classes long-lived TCP flows. AF1 and AF2 classes ask the network for 2.5Mbps each (Section 5.4.2.b). In this case, we observe that our algorithm stabilize the rates of the CBQ buffer after 20s.

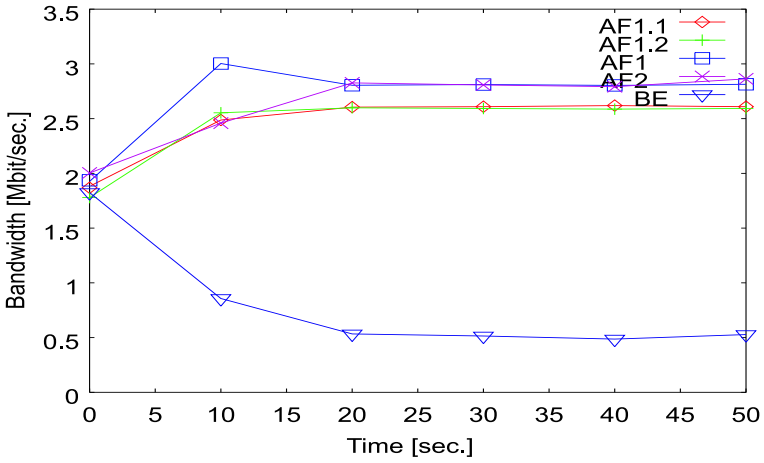


Figure 5.9: All service reservations are not satisfied

5.6.2 Algorithm Performances

The network stability depends on the behavior of the proposed algorithm, which is running in each network core router, and the free bandwidth sharing rule. The reaction speed of traffic changes requires an adaptation period. Figure 5.10 shows how fast our algorithm should run. The speed of the algorithm is determined by the averaging interval. Every averaging interval, our algorithm passes once by the loop while described in Section 5.4. If it runs faster than once each 5 seconds, the system oscillates. We found the optimum value of the averaging interval to be 10 seconds, as shown in fig. 5.11. For this experiment, we use TCP flows for all the services. This instability of the system for small averaging interval does not exist in case of constant rate non-reactive UDP flows.

Consider the scenario where our algorithm is implemented in all core routers. Figure 5.12 shows the topology for testing the algorithm behavior. Suppose that the class AF1 from HG1 wants to send more traffic than is guaranteed $R_{AF1.1}$=1.3 Mbps instead of $R_{AF1.1}$=1Mbps. We are in case when the other classes, which do not reserve any bandwidth, monopolize the excess bandwidth, which is completely unfair to class AF1. The weights of CBQ are not clearly adapted to this situation in router CR1. Our mechanism after a few iterations improves the AF1 service. The rates are obtained with the fair division rule, which divides the excess bandwidth equally among the three service classes. Using this rule, we increase the AF1 bandwidth from 1Mbps to R_{AF1}=1.4Mbps, and we penalize the other services that do not ask for any guaranteed bandwidth, R_{AF2}=1.3Mbps and R_{BE}=0.3Mbps. Figure 5.12 shows the bandwidth repartition between AF1 UDP traffic, AF2 TCP long

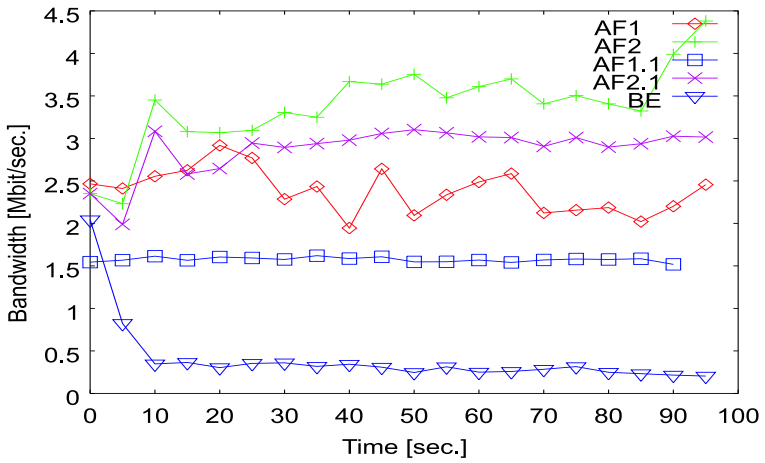


Figure 5.10: The algorithm runs with a period of 5 seconds. In this case the TCP flows from AF services have no time to adapt their windows. So, this creates oscillations.

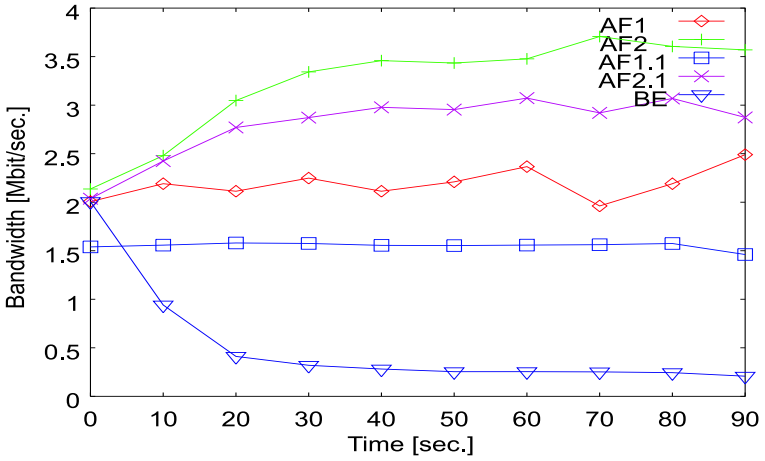


Figure 5.11: The algorithm runs with a period of 10 seconds. In this case the TCP flows from AF services have time to adapt their windows. So, there are no oscillations.

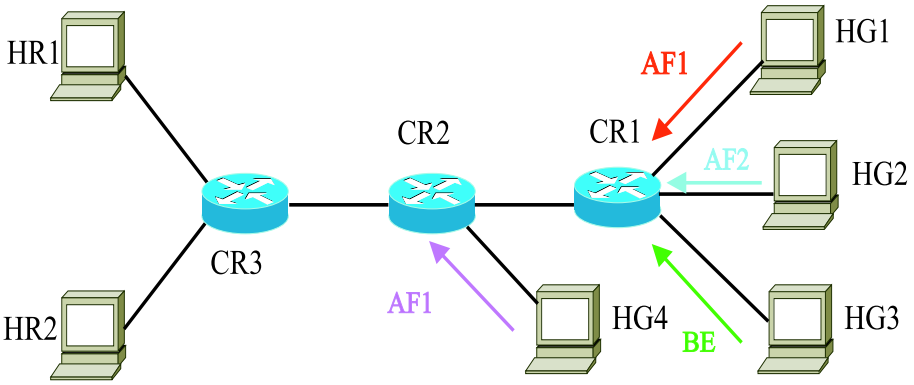


Figure 5.12: Network stability with two core routers. HG - Host Generator, HR - Host Receiver, CR - Core Router

connections traffic and BE TCP long connections traffic.

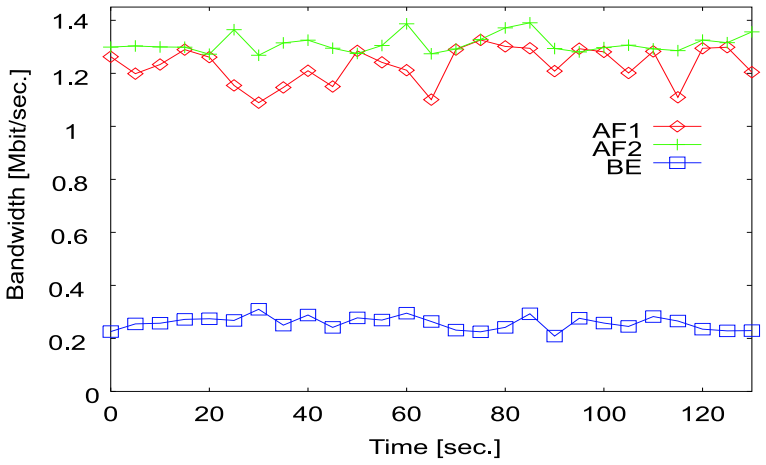


Figure 5.13: Bandwidth redistribution in CR1

The router CR2 is running the same rule with our algorithm. On this moment the algorithm from CR2 has the same behavior like the algorithm from the router CR1. The same traffic configuration is passing through the network.

Let us consider that a new reservation for AF1 arrives on the router CR2 as indicated in figure 5.12. The guaranteed profile for AF1 is increasing. The bandwidth guaranteed for HG4 in AF1 is $R_{AF1.1}$=1Mbps. The aggregate AF1 class is not satisfied, since the CBQ buffer from CR2 limits its rate to 1.4Mbps, whereas it is sending IN packets at a higher rate of 2Mbps. The total rate of IN packets generated by the AF1 and AF2 classes should be less than C between CR2 and CR3. The link bandwidth between CR2 and

CR3 is different than between CR1 and CR2. So, the new distribution of the CBQ rates is: R_{AF1}=2.5Mbps, R_{AF2}=1.4Mbps and R_{BE}=0.1Mbps (Fig. 5.14).

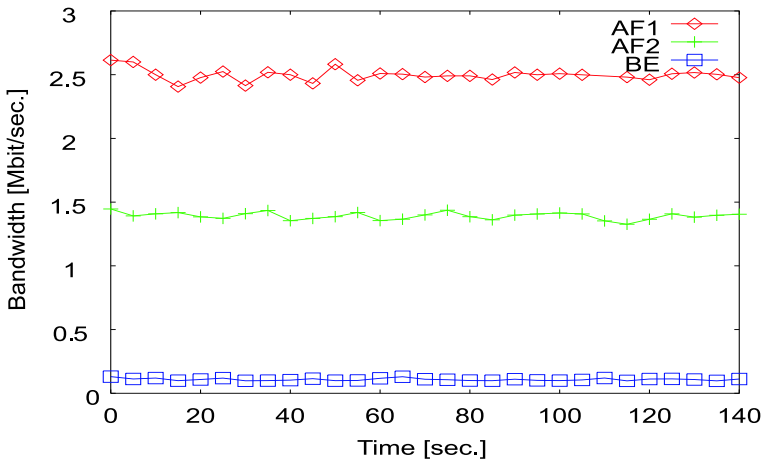


Figure 5.14: Bandwidth redistribution in CR2

From figures 5.13 and 5.14 we can observe two phenomenons:

- the BE class has always the minimum amount of bandwidth;
- the second router adjusts the traffic behavior for AF1 and AF2 because no class can monopolize the excess bandwidth;

The behavior of the core routers are independent. We measured the same values of traffic at CR1 after flow AF1 joint core router CR2.

Consider the scenario when the "fair distribution" rule of free bandwidth is replaced by the weighted distribution rule. The goal of the weighted distribution rule is to give AF service classes bandwidth, more advantages over BE class. The amount of the distribution rule weights should respect the following condition:

$$r_{AF1} + r_{AF2} + r_{BE} = 1 \tag{5.3}$$

For this experiment AF1 has UDP traffic, AF2 has TCP long connections traffic and BE UDP traffic. The values for the weights was chosen: r_{AF1}=0.6, r_{AF2}=0.3 and r_{BE}=0.1. After the distribution of the excess bandwidth the class rate are R_{AF1}=$R_{AF1.1}$+r_{AF1}*B_w, R_{AF2}=$R_{AF2.1}$+r_{AF2}*B_w, and R_{BE}=r_{BE}*B_w.

For see better the behavior of the algorithm with weighted distribution rule, we have chosen the scenario of one reservation is not satisfied and

one is satisfied presented in section 1.4.2., using fair division rule the algorithm distributes the bandwidth as in figure 5.4. R_{AF1}=R_{AF2}=2.8Mbps and R_{BE}=0.3Mbps.

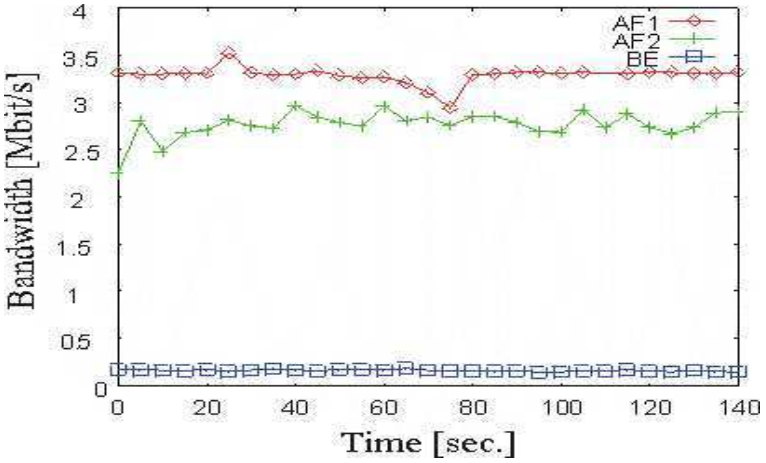


Figure 5.15: Rule with weight division of free bandwidth

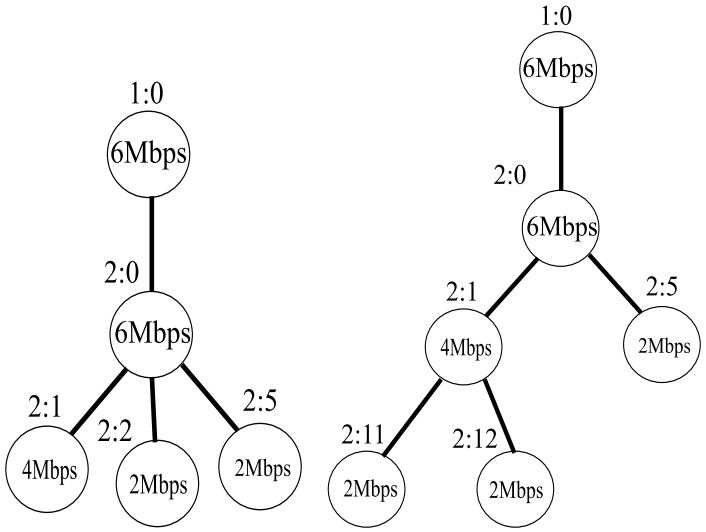
Using weighted division rule the algorithm distributes the bandwidth as in figure 5.15. We observe that the rates allocated in CBQ to the three classes are: R_{AF1}=3.1Mbps, R_{AF2}=2.8Mbps and R_{BE}=100Kbps. With rule from 5.3 BE traffic gets small values compare with fair division case.

5.7 The Interaction of CBQ and DRAM

In this section we study the impact of CBQ mechanisms related to bandwidth borrowing on the performance of DRAM. When both mechanisms used together lead to better overall performance due the bandwidth borrowing capabilities of CBQ. Service Level Agreements (SLA) profiles are then always respected and the network resources are better utilized than when DRAM is used alone. Dynamic Resource Allocation Mechanism does not replace CBQ but is complementary to it. The experiments have the same assumptions like those formulated in section 5.2. One of the issues studied is the influence of hierarchy of classes configuration over DRAM. Two type of configurations are used: *flat configuration* and *hierarchical configuration* (fig. 5.16).

And another issue is the usage of parameters bounded and isolated in the DRAM behavior with class configurations shown in figure 5.16.

The group of scenarios proves that the behavior of DRAM, from the point of view of satisfying user's resevations, is not affected by CBQ. Reservations



(a) Flat configuration

(b) Hierarchical configuration

Figure 5.16: Two configurations of CBQ classes

are satisfied whatever is the hierarchical organization of CBQ classes. The use of CBQ affects the way with which the excess bandwidth (bandwidth not reserved) is allocated to the different classes, and this is in the sole case when one or more classes of traffic transmit at less then the bandwidth attributed by DRAM. When all classes of traffic transmit at more than the bandwidth attributed by DRAM (all classes are congested), the use of CBQ has no impact, and we get the same performance we obtain in [117]. In the opposite case, the hierarchical organization of CBQ classes decides how the unused bandwidth of one class is distributed among the other classes of traffic. For example, we can decide that the bandwidth of one class is distributed among the other classes of traffic. For example, we can decide that the bandwidth left behind the class AF1 is only used by the class AF2 and not by the best-effort class. This is not feasible when DRAM is implemented in a simple WFQ buffer. The use of CBQ improves then the performance of DRAM and enhance it with the capacity to choose how to distribute the bandwidth left behind one or more classes of traffic among the remaining classes.

The scenarios show that the behavior of CBQ is almost the same with and without the bandwidth borrowing feature, and this is for different configurations of traffic class hierarchy.

Two sets of experiments are presented. One test network configuration is described in figure 5.17. The hierarchy of classes defined in the output interface of the core router CR1 are illustrated in figure 5.16.

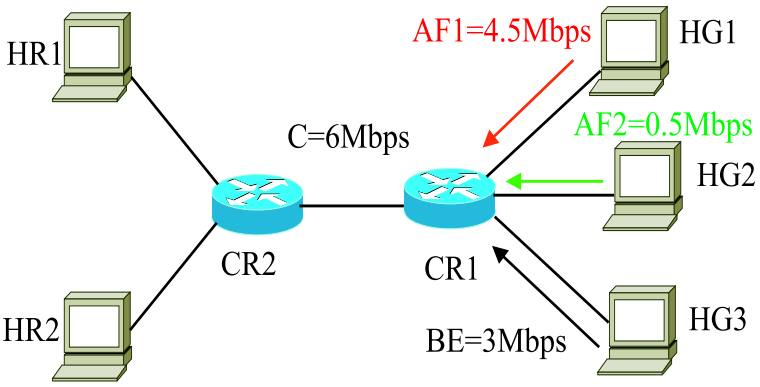


Figure 5.17: Test-bed network for link-sharing abilities if one reservation is not satisfied and one is satisfied. CR - core router; HR - host receiver; HG - host generator

The throughput for each class is provisioned for 2Mbit/s. We use the scenario from section 5.4, one reservation is not satisfied and one is satisfied. AF1 class is not satisfied, since CBQ buffer limits its rate to 2Mbit/s, whereas it is sending IN packets at a higher rate of 2.5Mbit/s. AF2 class is satisfied because it is sending IN packets with rate 0.5Mbit/s. We respect the conditions 5.1 and 5.2. When running our mechanism, the rate allocated to AF1 increases to more than $R_{AF1.1}$ and the rate allocated to AF2 is kept larger than $R_{AF2.1}$. Figure 5.18 shows the case when the DRAM algorithm has a flat configuration of classes. Two cases are presented. The *bounded* low level parameter is not included in the simple configuration hierarchy. The low level parameter *bounded* has presented in the section 4.2.

Because, all the traffic of AF2 is low and marked IN profile the rest of the bandwidth is taken by the AF1 and BE classes. The second case is when the parameter bounded is set to block the sharing algorithm. In the both cases, AF1 class of service has high priority. From figure 5.17 we observe that the two AF classes realize their desired guaranteed rate. The BE service gets the smallest amount of bandwidth, since it has zero guarantee. For presented cases the throughput for AF1 is different. AF2 receives R_{AF2} = 1.33Mbit/s which consumes $R_{AF2.1}$=0.5Mbit/s of bandwidth. If *bounded* low level parameter is set then the rest of the bandwidth from AF2 class will be not used by other classes. If the link-sharing algorithm is activated the rest of excess bandwidth from AF2 is distributed to AF1 and BE using priority criterion.

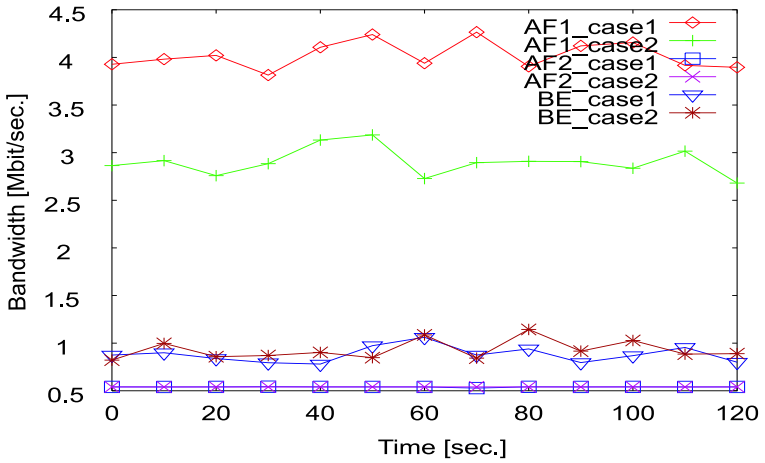


Figure 5.18: The behavior of DRAM without bounding AF1 class (case 1) and bounding AF1 class (case 2) using simple configuration

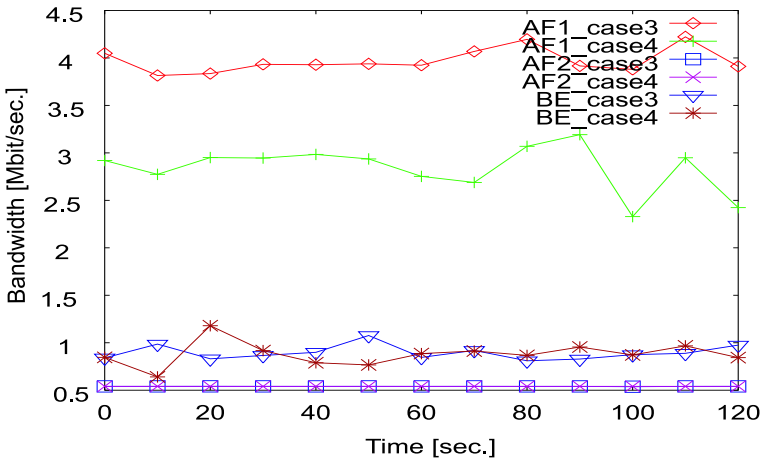


Figure 5.19: The behavior of DRAM without bounding AF1 class (case 3) and bounding AF1 class (case 4) using hierarchy configuration

The same cases are presented in figure 5.19 using hierarchical configuration of CBQ. AF classes of service have 2/3 of the link bandwidth and BE class has 1/3. Inside of the class AF the weight of each sub-classes is 1/2. AF1 class has the high priority and the AF2 the lowest. AF and BE classes have the same priority.

Using hierarchy we observe from figure 5.19 the same behavior like flat configuration.

Figure 5.20 shows another case when we use CBQ link-sharing behavior and DRAM algorithm. The goal of this experiment is to show that DRAM leads to better overall control performance than a simple CBQ. Same assumptions have been made like in the previous experiment.

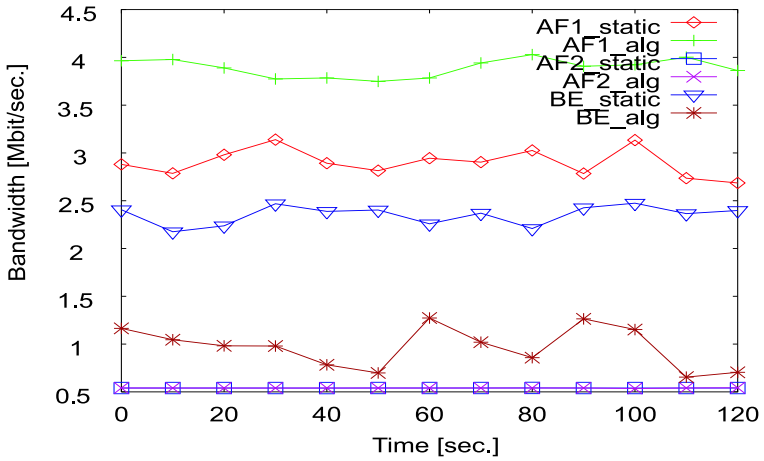


Figure 5.20: CBQ with link-sharing algorithm enabled (static) and DRAM (alg.)

Link-sharing algorithm reacts giving the excess bandwidth from AF2 class to AF1 and BE classes. Having the same priority to all classes, the excess bandwidth is divided in the same proportion. In the case when DRAM is used, BE class gets less. And AF1 class improves the QoS because its throughput has increased.

Another set of experiments test the network configuration used in figure 5.21. We check if the DRAM behavior has changed if we are using two different hierarchies configuration. We have changed the case when all reservations are not satisfied.

We are working using the assumptions of section 5.2. Initial, the CBQ limits the maximum rate of each class to 2Mbit/s. Both classes AF1 and AF2 are not satisfied since their IN traffic is dropped ($R_{AF1.1}$=2.5Mbit/s and $R_{AF2.1}$=2.5Mbit/s). If the rate of each CBQ class is not adjusted according, clients of BE class are favored.

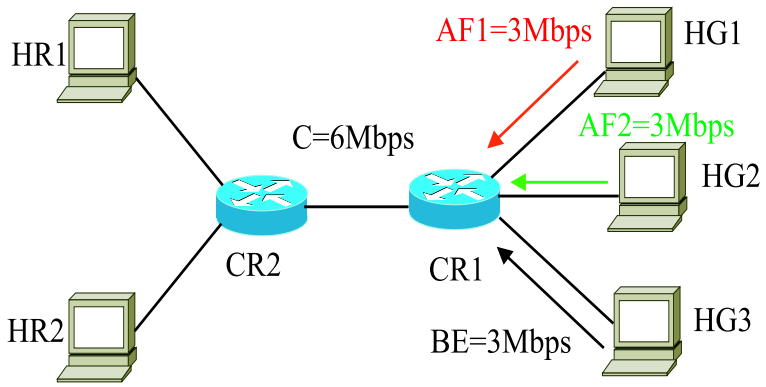


Figure 5.21: Test-bed network for link-sharing abilities if all reservations are not satisfied. CR - core router; HR - host receiver; HG - host generator

We apply our algorithm for both types hierarchy classes' shown in figure 5.16. We measure each class throughput at the output interface of the core router. The rule to divide the excess bandwidth is the same as that in the above sections. Figure 5.22 shows the case when we apply DRAM with flat hierarchy configuration. This proves that our algorithm has changed the class rates in order to satisfy the guarantee bandwidth. DRAM has the same behavior in the case of hierarchy configuration as in the case of flat configuration.

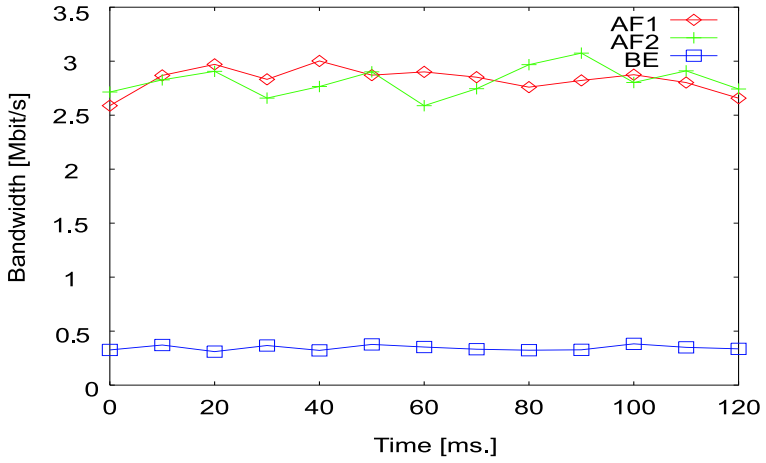


Figure 5.22: Link-sharing test if all reservations are not satisfied using flat configuration hierarchy

Using link-sharing algorithm the behavior of the DRAM is not changed than without it. In some cases the left bandwidth is more efficiently distributed among the classes. The service level specifications are not affected

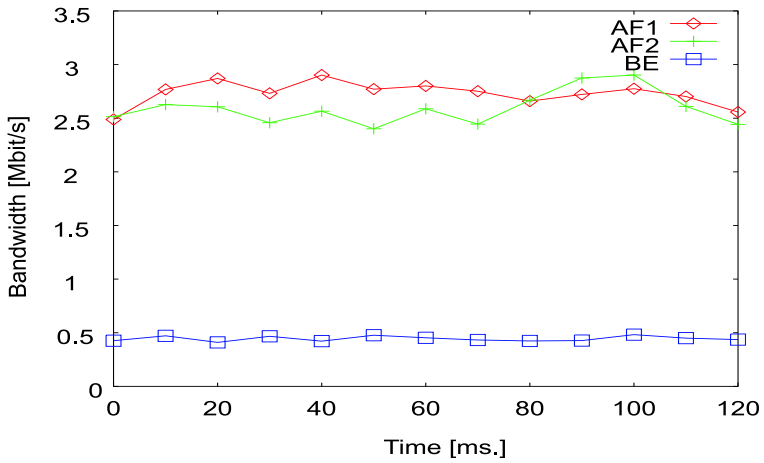


Figure 5.23: Link-sharing test if all reservations are not satisfied using hierarchy configuration

by using one of the two types of hierarchy's' classes.

5.8 Conclusion

In this chapter we proposed a mechanism of Dynamic QoS Management. It ensures that SLAs are respected and allows at the same time an efficient utilization of network resources. It is flexible because it can use other division rules for excess bandwidth. Our mechanism is easy to implement and does not require any particular signaling. Each ISP can define his own rules. We proposed two rules: fair division rule and weighted division rule.

We analysis the behavior of our algorithm for two types of class configurations: flat configuration of classes and hierarchical configuration of classes. With DRAM algorithm the excess sharing bandwidth rule is more controlled. The use of sharing algorithm properties of CBQ improves then the performance of DRAM and enhances it with the capacity to choose how to distribute the bandwidth left behind one or more classes of traffic among the remaining classes.

Chapter 6

Conclusion and Outlook

In this chapter, we first summarize briefly the contributions of this thesis. In section 6.2, we discuss extensions of this work that will provide interesting challenges for future research. We also give a more long-term vision which believe research will be able to make significant contributions in the field of Dynamic QoS Management in the Internet.

6.1 Summary of Contributions

Recent years have witnessed an enormous increase in the use of the Internet for a large variety of applications such as e-commerce, software distribution, news/stock quote broadcasting, multimedia streaming, video conference, and of course, data communication. There are applications that require hard performance guarantee from the network level. Increasing bandwidth by the management level is a necessary first step for accommodating these applications, but it is still not enough to avoid delay and jitter during traffic bursts. Even on a relatively unloaded IP network, delivery delays can vary enough to continue to adversely affect for example real-time applications. The state of the network is changing from various reasons (e.g. traffic load variations, network element failure, social users behavior, etc.). According to respect the need of the customers, is necessary for the management level to optimize all the throughputs in the network. A number of QoS mechanisms have evolved to satisfy the variety of applications needs. The management level should avoid the network instability by coordinating the control and adaptive QoS mechanisms of each network element.

In this context, we consider several aspects of Dynamic QoS Management: QoS provisioning and QoS control.

In the first chapter, we have shown the context of the work described in

this dissertation. We had presented several definitions of network management and explain them how are suitable for the scope of this thesis. This implies that the definition of DQM should not be designed for specific requirements of a particular network management system. Because of that, an important identification of this thesis is the definition of the components to perform the dynamic management of network resources in order to obtain quality of service.

In chapter 2, we have given an overview of the trends and issues in the QoS network management looking for DQM components and we have pointed out the main problems that we have addressed in this thesis.

Chapter 3 has focused on the first component of DQM, signaling. We have analyzed of the main proposals for QoS signaling protocols. A criteria classification was defined and proposed in chapter 3. By using various examples, criteria classification demonstrates the necessity to choose optimum signaling system when designing DQM. The protocols designed after RSVP are more scalable and less complex. We observe that it exists a compromise between the signaling functionality and scalability/complexity. From the point of view of security QoS signaling protocols are still weakness. To respond efficient of the various network states QoS signaling protocols should present flexibility and adaptability.

Another component of our QoS provisioning aspect is parameters' translation that has been presented in Chapter 4. Service level parameters (e.g. bandwidth, delay, jitter, packet loss) are used by admission control and resource allocation mechanisms. Each network element has its configurable parameters (e.g. scheduling mechanisms, queuing disciplines). We performed a comprehensive study how to put in correspondence service level parameters and network configurable parameters. We suggest that the area of mapping parameters require comprehensive solutions that are application specific and depend on network performance. A set of performance results from the point of view of dynamicity (e.g. users number, flows number, etc.) show that service level parameters for each application can closely be achieved by a hard tuning low-level configurable parameters. An adaptive and automated mechanism of translation adapts SLAs to the network state.

We demonstrate on a real network supporting Diffserv, using practical examples, with real traffic, the limitations of a Static QoS Management. Using static QoS management there is no service guarantee for short periods of time and no efficiently bandwidth using. We give two examples to show the tuning low-level parameters/network parameters necessity. Static QoS management requires manual tuning of network configuration parameters, each time when a customer changes his SLA (Service Level Agreement). The parameters tuning is actually characterized by extensive manual work, based

on a trial-and-error process which is often ineffective, time-consuming and costly to network managers. For that it is necessary to develop automated, self-configurable resource reservation mechanism to avoid these limitations.

In Chapter 5 we proposed a mechanism which ensures that SLAs are respected and allows at the same time an efficient utilization of the network resources (e.g. bandwidth, delay, jitter, packet loss) in order to respect the guaranteed constraints for each Diffserv class of service. The mechanism proposed is called Dynamic Resource Allocation Mechanism (DRAM) for Diffserv core routers. DRAM mechanism is easy to implement and does not require any particular signaling. The free bandwidth is divided using rules defined by the ISP (Internet Service Provider). We proposed and evaluated two rules: fair division rule and weighted division rule.

We proposed an analysis of the algorithm behavior using two types of class configurations (e.g. flat configuration of classes and hierarchical configuration of classes). An important conclusion is that with DRAM algorithm the excess sharing bandwidth rule is more controlled. The use of sharing algorithm proprieties of scheduler improves then the performance of proposed algorithm and enhanced it with the capacity to choose how to distribute the bandwidth left behind one or more classes of traffic among the remaining classes.

We believe that the complexity of network QoS management will be the main challenge for networking for coming years. Application and equipment developers face an environment of harsh competition and a great diversity of protocols, vendors, service providers, and applications. The researcher's perspective should take this into account by focusing less on global solutions that assume global control over the infrastructure; rather, the design space should be restricted to solutions that are viable in this dynamic, heterogeneous marketplace.

One very important lesson that we have learned from the impressive success of the Internet is the power of its network service model simplicity. More elaborate network service models had more trouble to become standards defacto because there are hard to deploy in practice.

6.2 Future Work

There remain many specific areas of research in the field addressed by this thesis which should be addressed by the future work. The major areas requiring investigation are enumerated in this section.

Based on the experience gained during the tests made in the networks presented in Appendix A we are thinking to improve our work. The routers are implemented using Linux PCs with multiple network interfaces. The

environment could be changed when other manufacturer router products replace the routers. QoS control mechanisms are different implemented and have different network parameters.

The algorithm proposed in chapter 5 is working for core routers in Diffserv. We assume that our network is working in an over-provisioning case, which means that there are enough resources to absorb the high priority IN traffic. Other interesting case is when the network is under-provisioned. The core routers cannot react to that type of congestion. It is the time of edge routers to react.

The heterogeneity of the network and the range of QoS application requirements make that a variety of QoS network services can be designed. Dynamic QoS Management should satisfy the user requests and maximize the benefits of ISPs (Internet Service Provider). New possibilities of excess bandwidth distribution can be proposed and control admission mechanisms by using microeconomics concepts that describe a user's relative preference. These concepts are defined by utility curves. Typically, a utility curve has an associated pricing function that can serve as a feedback control mechanism from network to applications to enforce application-to-network behavior. The pricing structure can be independent of the utility curve (e.g., monthly flat pricing), or dependent on it (e.g., congestion pricing, auction pricing, etc.)

The utility function concept can be integrated with our DRAM mechanism to have a feedback control mechanism of aggregate flows in core network and also to have a feedback control mechanism from network to applications/users. The utility functions associated with pricing function are used, as we saw in the literature for two main desires: maximize the benefits of ISPs and satisfy some user's relative preferences. The future work will present a network model using utility function model and DRAM.

Another future work topic is linked by the network reality. Today all ISPs have overprovisioned their capacities. To share the excess bandwidth we can use some overprovisioning rules. For example the rate of a real-time flow is measured and because the link bandwidth is overprovisioned, the allocated rate for it will be four times its rate. If the flows are not real-time they will use DRAM mechanism with fair share of the excess rest band rule. The novelty is the following: a flow can choose either between obtaining a fair share of the excess band or some overprovisioning. Our algorithm actually may give a class more than what it needs and sometime it may give it less. So, to guarantee 100% the bandwidth the flow can choose the overprovisioning part of the link bandwidth.

There are several promising and important extensions to the research problems addressed in this thesis. Indeed, a number of issues remain open in the design of efficient schemes for DQM.

A topic to be addressed in the future research is a comprehensive specification of the monitoring schemes for QoS. Two common approaches are defined: passive monitoring and active monitoring. The passive approach uses devices to watch the traffic as it passes by. The active approach relies on the capability to inject test packets into the network or send packets to servers and applications, following them and measuring service obtained from the network. Given the complementary of the two mechanisms, a research aspect is to explore the way to get the best of both worlds. For example, a possibility is for the active measurement probe to schedule passive measurements of appropriate metrics at appropriate points along the path, while the active measurements are being made. When the active measurement is completed then the appropriate passive measurements can be paused thus reducing the gathering of unnecessary data. By comparing and contrasting the active and passive measurements, the co-validity of the different measurements can be verified, and much more detailed information on carefully specified scheduled phenomena are made available.

Data confidentiality, integrity and protection against denial of service attacks are security issues for both *best-effort* and QoS enabled networks, other security issues are specific to QoS networks, such as protection against service theft. Much research is required in the area to provide fully security solutions.

The business and economic aspects of QoS services require special consideration and research. Dynamic QoS Management should be able to cover accounting and billing aspects. There is a fundamental trade-off between service performance and its associated complexity and cost, and research is ongoing for finding the balance the cost and acceptable price. The problem is further complicated by need for inter-operator agreements on dividing the costs and benefits for service spanning multiple network domains.

Appendix A

In this Annex we present two network topologies used for the tests from chapters 3, 4, and 5 of this dissertation. We describe the test-bed network environment: the monitoring tools, traffic generators tools and link emulator tools.

A.1 Test-bed networks

The test-bed network used for our experiments includes monitoring tools, traffic generator tools, traffic control tools and link emulator tools. Traffic control tools are more detailed in Annex B. Our work was validated using the network topology shown in the next two figures (fig.A.1 and fig.A.2):

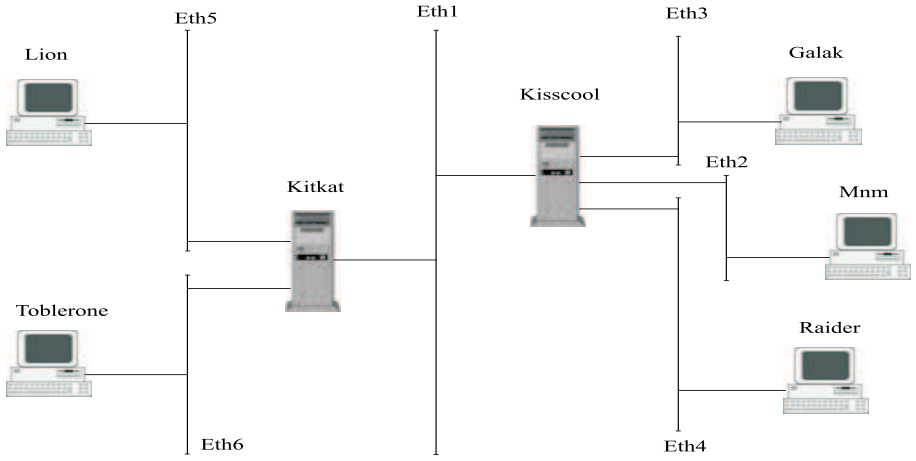


Figure A.1: Test-bed network topology for experiments of Chapters 3,4,5

The networks presented in figure A.1 and figure A.2 are a heterogeneous environment. CPU speed varies between 200Mhz and 600Mhz. We choose the highest CPU speed for the edge routers because, as we have presented in

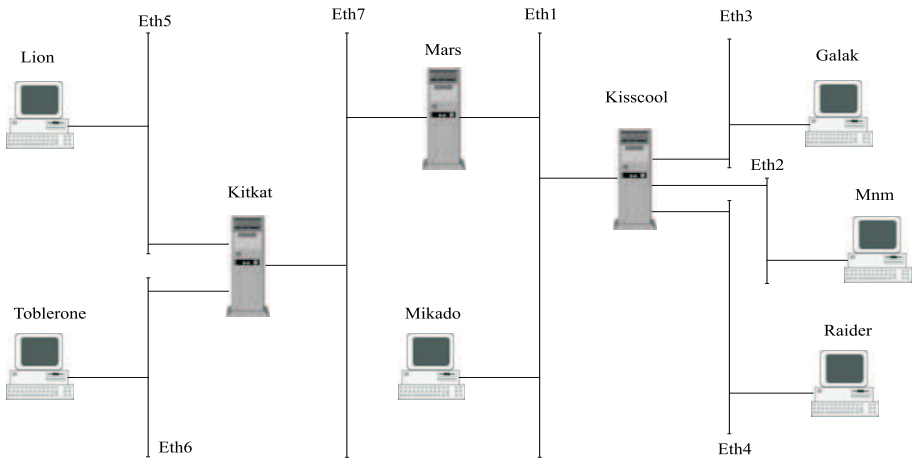


Figure A.2: Test-bed network topology for experiments of section 5.6.2

chapter 2, for Diffserv, the are more "QoS operations" than in core routers. Another reason was to inject traffic and to obtain congestion inside the network cause of the "slow" hardware. Every interface is Ethernet IEEE 802.3, 10Mbit/s. Each network element has one control interface. For accurate measures we isolated the control traffic and the test-bed traffic.

The operating system is Linux RedHat 7.2 with traffic control modules included in kernel. The traffic control modules are described in Annex 2.

A.2 Monitoring tools

The monitoring tools chosen for ours experiments are Netramet, TCPSTAT for bandwidth measurements. For delay we used ping application and for jitter we build an application client-server using a time-stamp in the IP packets.

Netramet is a meter for network traffic flows. It is an implementation of the Realtime Traffic Flow Measurements (RTFM) Working Group's Measurement Architecture. Netramet has three components:

- meters which are small hosts which are attached to a network segment and measure traffic following on that segment;

- meter readers, which retrieve information from meters;

- managers, which instruct meters as to which flows they should measure and meter readers as to which meters they should collect from, at what intervals.

A meter reader can collect flow data from many meters, and each meter may have its data retrieved by several meter readers. Traffic flows of interest are defined by user-specified set of rules.

The protocol used to collect the data from meters to managers and to instruct the meters what should measure, is SNMP (Simple Network Management Protocol). Each flow has an instance of MIB (RTFM Traffic Meter MIB). The manager instructs the meter what to extract from the MIB instances.

TCPSTAT reports certain network statistics. TCPSTAT gets its information by either monitoring a specific interface or by reading previously saved tcpdump data from file. It is very flexible because it can do different statistics (e.g. shows statistics only for http traffic crossing that interface, etc.). TCPSTAT offers two useful modes of operation: accounting mode and bandwidth mode. Accounting mode means that all the traffic for certain flows on interface is recorded. The bandwidth mode measure the traffic estimates based on a small sample. The minimum value of this sample is 0.5 ms. The sizes of the sample used in our experiments are between 1s and 10s.

For jitter measure we have imagined a simple application client-server. The server sends a number of fixed size packets using a periodicity. The time period is set one second. The client started to compute the jitter using the time period between two packets and the fixed value considered at server.

A.3 Traffic generator tools

The simple tool used to generate traffic and to measure end-to-end performances is *ttcp*. Is a server/client application. With *ttcp* is no risk to overload the destination server. For example in an endless loop it keeps for sending data instead of stopping. *ttcp* generates one connection; one data stream. Always tries to generate as much traffic as possibility.

Iperf is a client-server program that generated and measured the bandwidth. It has more options than *ttcp*. *Iperf* can generate parallel connections to the same destination port in one big data stream. For UDP traffic, *Iperf* can specify a limit to transmission. This generator creates a constant bit rate UDP stream. This a very artificial stream, similar to voice communication but not much else. Like *ttcp* always tries to generate as much traffic as possibility. For TCP traffic, *Iperf* can specifies the windows size and the

maximum transmission unit (MTU). *Iperf* uses multicast configuration to generate and measure traffic.

Other sources of traffic used for the experiments are applications: RealPlayer, file transfer protocol, and telnet.

A.4 Link Emulator Tool

NIST Net is a network emulator for general purpose in IP networks. By operating at the IP level, *NIST Net* can emulate the critical end-to-end performance characteristics imposed by various wide area network situations (e.g. congestion, node failure, etc.) or by various underlying subnetwork technologies (e.g. asymmetric bandwidth situations of xDSL and cable modems). NIST Net can emulate numerous complex scenarios, included: tunable packet delay distributions, congestion and background loss, bandwidth limitation, and packet reordering/duplication. NIST Net is implemented as a kernel module extension to the Linux operating system and an X Window system-based user interface application.

NIST Net also provides support for user defined packet handlers to be added to the system. For examples: time stamping/data collection, interception and diversion of selected flows, generation of protocol responses from emulated clients.

A similar package for FreeBSD system is called *dummynet* [124],[125].

Appendix B

The Annex B presents the basic principle of traffic control (tc) tool work. We explain how can be implemented a core and an edge Diffserv router using tc. In Chapters 3, 4, and 5 we provision with traffic control tool the QoS control mechanisms implemented in kernel routers.

B.1 Traffic Control Tools

The Linux kernel distribution offers the possibility to activate during kernel compilation QoS mechanisms. The QoS functionality can be controlled by a user space program called TC (traffic control). The components of QoS functionality are called traffic control modules. Traffic control modules can decide if packets are queued or if they are dropped (e.g. if the queue has reached some length limit, or if the traffic exceeds some rate limit), it can decide in which order packets are sent (e.g. to give priority to certain flows), it can delay the sending of packets (e.g. to limit the rate of outbounding traffic).

The basic principle involved in the implementation of QoS in Linux is shown in figure B.1. This figure shows how the kernel processes incoming packets, and how it generates packets to be sent to the network. The input de-multiplexer examines the incoming packets to determine if the packets are destined for the local node. If so, they are sent to the higher layer for further processing. If not, it sends the packets to the forwarding block. The forwarding block, which may also received locally generated packets from the higher layer, looks up the routing table and determines the next hop of the packet. After this, it queues the packets to be transmitted on the output interface. It is at this point that the linux traffic control module comes into play.

Linux traffic control modules can be used to build a complex combination of queuing disciplines, classes and filters that control the packets that are sent

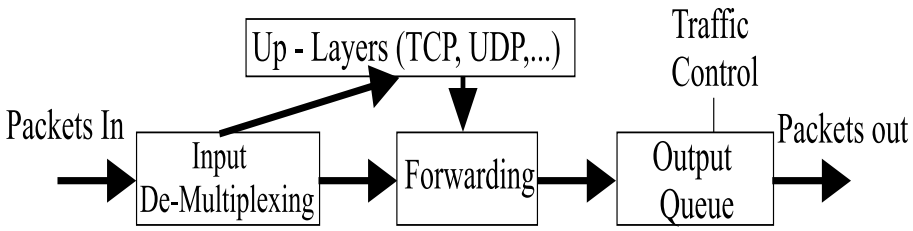


Figure B.1: Kernel Processes for Traffic control in Linux

on the output interface.

The main components of Linux traffic control are filters, classes and queuing disciplines (e.g. qdisc). Packets are selected in classes (u32, fw, route, tcindex) by filters. Each class uses one of queuing disciplines (tbf, pfifo, red, gred). There are hierarchies of classes and filters. Schedulers used by tc are: CBQ (Class Based Queuing), PRIO (Priority Queuing) and HTB (Hierarchical Token Bucket).

Linux traffic control components designed in the kernel can implement IntServ and DiffServ frameworks, respecting IETF RFCs.

There are several different methods of achieving the same functionality for interior and boundary node of Linux Diffserv. The biggest difference between interior and boundary node configurations is that basically boundary nodes divide traffic into different classes according to specific criteria (filters defined by the network administrator) while interior nodes perform the division based on DSCPs.

In the boundary nodes EF packets are matched using u32 filter [110] and out-of-profile packets are dropped. AF packets are matched using u32 or fw filter [110]. The out-of-profile packets are marked with lowest priority than in-of-profile packets. AF packets are using a GRED mechanism. BE packets have a RED queuing discipline and EF packets a FIFO queuing discipline. GRED (Generalized RED) is a RED mechanism with multiple dropping levels [110].

In the interior nodes a priority scheduler has two priorities, a high priority for EF and a low priority for the rest of the traffic. The EF class is policed (or metered) by a TBF to guarantee conformance and then put into the priority scheduler input.

The rest of the classes are served through a CBQ scheduler that shares the bandwidth among them according to adjustable weights and parameters. Each sub-class of the four AF classes is implemented using a GRED queuing discipline with 3 drop precedence. The BE traffic is passed through a RED queuing discipline as a normal router handling.

For the interior nodes the classification is according to the DSCP on each

packet. The filter structure presented in the core router is simpler than the filter structure of the border router. It should be noted that for interior node point of view the packets have already marked by some edge device or end host.

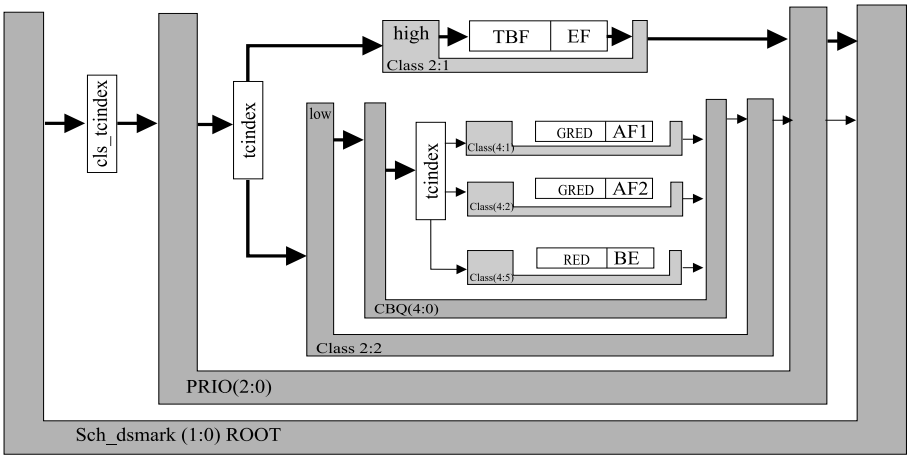


Figure B.2: Internal node in Diffserv Linux network

Figure .4 shown the queuing disciplines and classes used for PHB (Per-Hop Behavior). There are 4 levels of filters of packets. The first level of filters is for differentiated service mark (dsmark (1:0)). The filter obtains DSCP from each packet. The operation done by this filter is to extract the DS field masking TOS field with 0xfc and shift with two bits to the right to get the DSCP. For each handle there is a corresponding classid which is attached to the IP packet (internal reasons). Each classid has three digits (e.g., EF has classid 1:100, AF11 has classid 211, etc.). The second level filter is under the prio scheduler that extracts the leftmost digit from the classid of the packets. In this case the classid value is masked with 0xf00 and shifting with 8 bits to the right. The handle corresponds to the correct class under the priority scheduler (e.g. 1 for EF and 2 for the rest of classes). The third level filter under the CBQ scheduler (4:0) extracts the second digit from the classid by masking it with 0xf0 and shifting 4 bits to the right. The handle corresponds to the correct class under the CBQ scheduler. The fourth level is internal in GRED implementation. Classid value is masked with 0xf to extract the third digit of the classid and put them into their correct virtual queues under GRED.

Figure .5 shows the configuration of traffic control inside the core router used for experiments in chapter 5. The queuing disciplines and classes reflect usual Linux notation of displaying tree-based traffic control framework. We used the same filter procedure like for example in figure B.2. The available

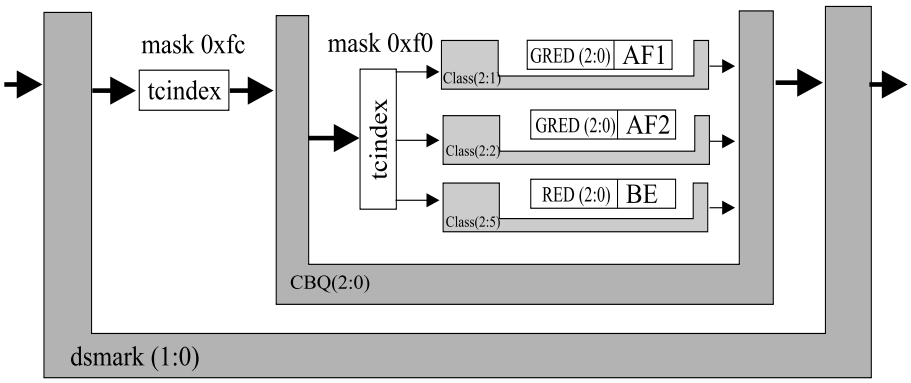


Figure B.3: Interior of core routers using dsmark, tcindex, CBQ, GRED and RED

bandwidth at the output interface of the core router is divided by CBQ, which uses three classes with classid 2:1, 2:2 and 2:5. For the first experiments to avoid borrowing bandwidth from the ancestors and to block the bandwidth sharing of the same parent, we set the isolated and bounded parameters. Buffers associated to AF1 and AF2 in the core router implement the GRED mechanism with two drop precedence class (AF1.1 and AF1.2 for AF1, AF2.1 and AF2.2 for AF2).

Bibliography

[1] CCITT: "Recommendation E.410 - Telephone Network and ISDN - Quality of Service, Network Management and Traffic Engineering - International Network Management - General Information", Geneva 1992;

[2] Graham J.: "The Penguin Dictionary of Telecommunications", Penguin books, 1985;

[3] ISO 7498-4: "Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework", Geneva, 1989;

[4] RACE CFS A150: "General Terminology", Common Functional Specifications, document 11, Issue B, December 1991;

[5] ITU-T: "Standard X.902, Information technology - Open distributed processing - Reference Model", Geneva 1990;

[6] Saltzer J., Reed D., Clark D., "End to End Arguments in System Design", ACM Transactions in Computer Systems, November 1984;

[7] Jeffrey K. Mackie-Mason, Hal R. Varian: "Some Economics of the Internet", 10th Michigan Public Utility Conference, March 1993, Michigan, USA;

[8] S. Blake, D. Black, M. Carlson, E.Davies, Z.Wang, "An Architecture for Differentiated Services", Request for Comments RFC2475, December 1998;

[9] R. Braden, D. Clarck, S.Shenker, "Integrated Service in the Internet Architecture: an Overview", Request for Comments RFC1633, June 1994;

[10] E. Rosen, R. Callon "Multiprotocol Label Switching Architecture", Request for Comments RFC 3031, January 2001;

[11] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSer-Vation Protocol (RSVP) – Version 1 Functional Specification " Request for Comments RFC 2205, September 1997;

[12] D. Durham Ed., J. Joyle, R. Cohen, S. Herzog, "The COPS (Common Open Policy Service) Protocol", Request for comments RFC 2748;

[13] D. Ahlard, J.Bergkvist, I. Cselenyi, "Boomerang - A Simple Protocol for Resource Reservation in IP Networks", IEEE Workshop on QoS Support for Real-Time Internet Applications, June 1, 1999. Vancouver, Canada

[14] P. Pan, E, Hahne, and H. Schulzrinne, "BGRP: A Tree-Based Aggregation Protocol for Inter-domain Reservations", Journal of Communications and Networks, Vol. 2, No. 2, June 2000, pp. 157-167.

[15] Nelson Tang, Sonia Tsui, Lan Wang,"A survey of Admission Control Algorithms", Computer Science Departament UCLA, report for CS 215, Dec. 1998, USA;

[16] Z. Wang, "Internet QoS: Architecture and Mechanisms for Quality of Service", Morgan Kaufmann, 2001;

[17] P. Ferguson, G. Houson, "Quality of Service: Delivering QoS on the Internet and in Corporate Networks", John Wiley & Sons, 1998;

[18] R. J. Gibbens, S. K. Sargood, F. P. Kelly, "An Approach to Service Level Agreements for IP networks with Differentiated Serives", article submitted to Royal Engineering Society, January 2000;

[19] P. Trimintzios, T. Bauge, G. Pavlou, L. Georgiadis, P. Flegkas R. Egan, "Quality of Service Provisioning for Supporting Premium Services in IP Networks" Proceedings of IEEE Globecom 2002, Taipei, Taiwan, November 2002;

[20] H. Zhang, "Service disciplines for guaranteed performance service in packet switching networks", Proceedings of the IEEE, 1996;

[21] L. Georgiadis, R. Guerin, V. Peris, R. Rajan, "Efficient support of delay and rate guarantees in an internet", in Proceedings of Sigcomm'96, August 1996, pp.106-116;

[22] International Telecommunication Union, ITU-T, http://www.itu.int/home/index.html;

[23] The Internet Engineering Task Force, IETF, http://www.ietf.org;

[24] X. Xiao, L. Ni, "Internet QoS: A Big Picture", IEEE Network Magazine, March/April 1999;

[25] Fankhauser. "A Network Architecture Based on Market Principles", PhD thesis, ETH Zurich, March 2000;

[26] N. Semret, R. Liao, A.T.Campbell, A. Lazar, "Market Pricing of Differentiated Internet Services", IEEE-IFIP IWQoS, June 2-4, 1999

[27] C. Partrige, "A Proposed Flow Specification" Request for Comments, RFC1363, September 1990;

[28] D. Ferrari, D. Verma, "A Scheme for Real-Time Channel Establishment in Wide Area Networks", IEEE Journal Selected Areas in Comm., Vol 8 No 3, April 1990;

[29] D. Clarck, D. Lambert, L. Zhang, "NETBLT: A High Throughput Transport Protocol", Computer Communication Review, Vol. 17, No.5, 1987;

[30] A. Campbell, G. Coulson, D. Hutchison, "Supporting Adaptive Flows in a Quality of Service Architecture", Multimedia Systems Journal, November, 1995;

[31] R. R-F Liao, A.T. Campbell, "Dynamic Edge Provisioning for Core IP Networks", 8th International Workshop on Quality of Service, Pittsburgh, USA, June 2000;

[32] Chris Gill, "Design Principle and Architectures for End-toEnd Quality of Service", Course cs6873, Washington University, St. Louis, MO, March 2001;

[33] P. Reichl, S. Leinen, B. Stiller, "A practical review of pricing and cost recovery for Internet services", Proceedings of the 2nd Internet Economics Workshop Berlin (IEW'99), Berlin, Germany, May, 1999;

[34] Xin Wang, Henning Schultzrinne, "Performance Study of Congestion Price based Adaptive Service", Proceedings International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'00), Chapel Hill, NC,June 2000;

[35] Raj Rajkumar, Chen Lee, John Lehoczky, D. Siewiorek, "A Resource Allocation Model for QoS Management", Proceedings of the IEEE Real-Time Systems Symposium, December 1997;

[36] Nemo Semret, R-F.Liao, A.T. Campbell, A.Lazar, "Pricing, Provisioning and Peering: Dynamic Markets for Differentiated Internet Services and Implications for Network Interconnections", IEEE Journal on Selected Areas of Communications;

[37] R. Guerin, H. Ahmadi and M. Nagshineh, "Equivalent Capacity and Its Application to Bandwidth Allocation in High-Speed Networks", IEEE J. Select. Areas Commun., vol. 9, 1991, pp.968-981;

[38] S. Floyd, "Comments on Measurement-based Admissions Control for Controlled-LoadService," submitted to Computer Communication Review, July 1996;

[39] D. Tse, M. Grossglauser, "Measurement-based Call Admission Control: Analysis and simulation" Proc. Of INFOCOM'97, April, 1997;

[40] S. Jamin, P. Danzig, S. Shenker, L. Zhang, "A Measurement-based Admission control Algorithm for Integrated Services Packet Networks (Extended Version)" ACM/IEEE Transactions on Networking, February 1997;

[41] S. Holmes, A. Court, "Admission Control Policies for Internet File Transfer Protocols", Darthmouth College Technical Report PCS-TR-315, May 1997;

[42] Liao R. R-F, P. Bocheck, A.T. Campbell, S-F. Chang, "Utility-based Network Adaptation for MPEG-4 Systems", Proceeding 9th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'99), Basking Ridge, NJ, June 1999

[43] I. Stoica, H. Zhang, "Core-stateless fair queuing: Achieving approximately fair bandwidth allocations in high speed networks", SIGCOMM Symposium on Communications Architectures and Protocols, BC, Canada, August 1998;

[44] I. Soica, H. Zhang, "Providing guaranteed services without per flow management", SIGCOMM Symposium on Communications Architectures and Protocols, Boston, MA, August 1999;

[45] I. Stoica, H. Zhang, S. Shenker, R. Yavatkar, D. Stephens, A. Malis, Y. Bernet, Z. Wang, F. Baker, J. Wroclawski, C. Song, R. Wilder, "Per hop behaviors based on dynamic packet states", Internet Draft 1999, Work in Progress;

[46] W. Almesberger, T. Ferrari, J.-Y. Le Boudec, "SRP: a scalable resource reservation protocol for the Internet", Proceedings of Fifth IFIP International Workshop on Quality of Service (IWQoS'98), Napa, CA, May, 1998;

[47] P. Pan, H. Schulzrinne, "Yessir: A simple reservation mechanism for the Internet", Proceedings of the 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV98), Cambridge, United Kingdom, July 1998;

[48] Prashant Chandra, Allan Fisher, Peter Steenkiste, "Beagle: A Resource Allocation Protocol for an Advanced Services Internet", Technical Report CMU-CS-98-150, August 1998;

[49] Fankhauser, G., D. Schweikert, B. Plattner "Service Level Agreement Trading for the Differentiated Services Architecture" Swiss Federal Institute of Technology, Computer Engineering and Networks Lab, Technical Report No. 59. November 1999;

[50] N. Semret, R. R-F. Liao, A. T. Campbell, "Market Pricing of Differentiated Internet Services", 7th International Workshop on Quality of Service (IEEE/IFIP IWQOS'99), pg. 184-193, London, UK;

[51] The Distributed Management Task Force, http://www.dmtf.org;

[52] Abilene advanced backbone network, http://abilene.internet2.edu/html/about.html;

[53] Very High Performance Backbone Network Service, http://aspin.asu.edu/vbns/Propose/propose.html;

[54] Video High Performance Backbone Network Service, http://www.homechoice.co.uk ;

[55] QUANTUM: Quality network technology for user-oriented multimedia, http://www.aramis-research.ch/d/7203.html;

[56] J. Wroclawski, "Specification of the Controlled-Load Network Element Service", Request for Comments, RFC2211;

[57] S. Shalunov, B. Teitelbaum, "Qbone Scavenger Service (QBSS) Definition for Internet2", Technical Report, March, 2001;

[58] S. Shenker, J. Wroclawski, "Network Element Service Specification Template, Request for Comments RFC2216, September 1997;

[59] S. Shenker, J. Wroclawski, "General Characterization Parameters for Integrated Service Network Elements" Request for Comments RFC2215, September 1997;

[60] S. Shenker,C. Partrige, R. Guerin "Specification of Guaranteed Quality of Service" Request for Comments, RFC2212; ;

[61] Yaw Opoku, JinSoo Park, Todd J.Eshler, Kevin P. Flanagan, "Admission Control for IP Quality of Service", report RPT-98-2000, Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, VA;

[62] M. Shreedhar, George Varghese, "Efficient Fair Queueing using Deficit Round Robin", Proceedings ACM SIGCOMM'95, August 1995;

[63] I Stoica, H. Zhang, T.S. Eugene, "A Hierarchical Fair Service Cureve Algorithm for Link-Sharing, Real-Time and Priority Service", Proceedings of SIGCOMM'97;

[64] A. K. J. Parekh, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks" Ph.D thesis, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, February 1992, No. LIDSTH-2089;

[65] F. Baker, J. Krawczyk, A. Sastry, "Integrated Service Management Information Base using SMIv2", Request for Comments RFC2213, September 1997;

[66] "F. Baker, J. Krawczyk, A. Sastry, Integrated Services Management Information Base Guaranteed Service Extensions using SMIv2", Request for Comments RFC2214, September 1997;

[67] R. Braden, D. Clarck, S. Shenker, "Integrated Services in the Internet Architecture: an Overview", Request for Comments RFC1633;

[68] Y. Bern, F. Baker, L. Zhang, "A Framework for Integrated Services Operation over Diffserv Networks", Request for Comments RFC2998;Jean

[69] A. Terzis, J. Krawczyk, L. Zhang, "RSVP Operation Over IP Tunnels", Request for Comments, RFC2746;

[70] G.Karagiannis, V.Rexhepi, "A Framework for QoS and Mobility in the Internet Next Generation", Report No. 9/036-FCPNB 10288 Uen, Utwente University, 2000;

[71] Bruce S. Davie, Yakov Rekhter, "MPLS: Technology and Applications", Ed. Morgan Kaufmann, 2000;

[72] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, "LDP Specification", Request for Comments, RFC3036;

[73] D.Awduche, J.Malcom, J.Agogbua, M.O'Dell, J.McManus, "Requirements for Traffic Engineering Over MPLS", Request of Comments, RFC2702;

[74] Rick Gallaher, "Introduction to MPLS Label Distribution and Signaling", Converge Network Digest Magazine, November, 2001;

[75] D.Awduche, L.berger, D.Gan, T.Li, V.Srinivasan, "RSVP-TE: Extensions to RSVP for LSP Tunnels", Request for Comments, RFC3209;

[76] J. Ash, Y. Lee, P. Ashwood-Smith, D. Skalecki, "LSP Modification Using CR-LDP", Request for Comments, RFC3124;

[77] Gyu Myoung Lee, Jun Kyun Choi "A Study of Flow-based Traffic Admission Control Algorithm in the ATM-based MPLS Network", The 15th International Conference on Information Networking (ICOIN'01), Beppu City, Oita, Japan;

[78] CISCO VPN Solutions Center: "MPLS Solution Provisioning and Operation Guide", http://www.cisco.com/univercd/cc/td/doc/product/rtmgmt/vpnsc/mpls/

[79] K. Nichols, S. Blake, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", Request for Comments RFC2474, December 1998;

[80] S. Brim, B. Carpenter, "Per Hop Behavior Identification Codes", Request for Comments, RFC2836;

[81] V. Jacobson, K. Nichols, "An Expedited Forwarding PHB", Request for Comments, RFC2598;

[82] B. Davie, A. Charny,J. Bennett, K. Benson, "An Expedited Forwarding PHB (Per-Hop Behavior)", Request for Comments RFC2598, March 2002;

[83] J. Heinanen, F. Baker, W. Weiss, "Assured Forwarding PHB Group", Request for Comments RFC 2597;

[84] Z. -L. Zhang, Z. Duan, L. Gao, Y. T. Hou, "Decoupling QoS control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services", SIGCOMM Symposium on Communications Architectures and Protocols, Stockholm, Sweden, August 2000;

[85] J. Garay, I. Gopal, S. Kutten, Y. Mansour, M. Young, "Efficient on-line call control algorithms", Journal of Algorithms, 29, pg.180-194, 1997;

[86] L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, H.Zhang, "Endpoint admission control: Architectural issues and performance", Proceedings SIGCOMM 2000;

[87] Y. Bernet, S. Blake, D. Grossman, A. Smith, "An Informal Management Model for Diffserv Routers", Request for Comments, RFC3290;

[88] Puqi P. Tang, Tsung-Yuan Charles Tai, "Network Traffic Characterization Using Token Bucket Model", Infocom 1999, p.51-62;

[89] S. Floyd, V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", IEEE ACM Transactions on Networking, Vol.1, No.4, p.397-413, 1993;

[90] M. Grossglauser, Srinivasan Keshav, "On CBR Service", Infocom 1996, p.129-137;

[91] Ikjun Yeom, A.L.Narasimha Reddy, "Modeling TCP Behavior in a Differentiated Service Network", IEEE/ACM Tran. Networking, Vol.9, No 1 p31-46, 2001;

[92] J. Warland, G. Keseilis, "Effective bandwidth for multiclass Markov fluids and other ATM sources", IEEE/ACM Tran. Networking, Vol.1 (1993) 424-428;

[93] Rares S., Slim G., Walid D., "Internet QoS Signaling Protocols", Proceeedings IEEE Communications 2000, December 2000, Bucharest, Romania;

[94] H. Schulzrinne, G. Fokus, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", Request for Comments, RFC 1889;

[95] "Experimental Internet Stream Protocol, Version2 (ST-II)", Request for Comments, RFC1190;

[96] P. Pan, E. Hahne, H. Schulzrinne, "BGRP: A Tree-Based Aggregation Protocol for Inter-domain Reservations", Journal of Communications and Networks, Vol.2, No.2, June 2000, pg.157-167;

[97] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry, "The COPS (Common Open Policy Service) Protocol", Request for Comments, RFC2748;

[98] Boomerang signaling generator / RSVP signaling message generator, http://soha.ttt.bme.hu/boomerang/public/Tools/siggen.tgz;

[99] Yessir generator tool, http://www.cs.columbia.edu/ hgs/rtptools/;

[100] Darwin: Resource Management for Application-Aware Networks, http://pecan.srv.cs.cmu.edu/afs/cs/project/cmcl/www/darwin/ApAwNe.html;

[101] Packet forwarding time measurement tool (kernel patch), http://soha.ttt.bme.hu/boomerang/public/Tools/fwlog-0.1.tar.gz;

[102] Intel COPS Client Sofwtare Development Kit, http://www.intel.com/labs/manage/cops/;

[103] P. Chandra, A. Fisher, P. Steenkiste, "A Signaling Protocol for Structures Ressource Allocation", IEEE Infocom 1999, New York, March 1999;

[104] O. Shelen, S. Pink, "Aggregating resource reservation over multiple routing domains", Proceedings of Fifthe IFIP International Workshop on Quality of Service (IWQoS'99), Cambridge, UK, June 1999;

[105] D.Y. Pei, "Authentication schemes - Tutorial", Institute of Materials Research and Engineering, Singapore, September 2001;

[106] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, "Iperf - a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics", http://dast.nlanr.net/Projects/Iperf/;

[107] National Institute od Standards and Technology: NIST Net emulator tool, http://www.antd.nist.gov/nistnet;

[108] Paul Herman, "tcpstat - Network Interface Statistics", http://www.frenchfries.net/paul/projects.html;

[109] Nevil Brownlee, "NeTraMet tool", http://www2.auckland.ac.nz/net/Accounting/ntm.Release.note

[110] Bert Hubert, Gregory Maxwell, "Linux Advanced Routing & Traffic Control" HOWTO, v0.9.0, January, 10, 2002;

[111] Bert Hubert, Gregory Maxwell, Stef Coene, "Linux Advanced Routing & Traffic Control", http://lartc.org;

[112] S. Floyd, V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking, Vol.3, No.4, August 1995;

[113] S. Floyd, V. Jacobson, "Random Early Detection gatewats for Congestion Avoidance", IEEE ACM Transactions on Networking, Vol.1, No.4, pg.397-413, 1993;

[114] S. Floyd, "RED: Discussions of Setting Parameters", November 1997;

[115] Linux Kernel mailing list Archive, subject: "RED quequeing", 2001; http://www.cs.helsinki.fi/linux/linux-kernel/;

[116] W. Almesberger, "Linux Traffic Control - Implementation Overview", EPFL ICA Swiss, February, 2001;

[117] Rares Serban, Chadi Barakat, Walid Dabbous, "Dynamic Resource Allocation in Core Routers of a Diffserv Network", ASIAN 2002, December 2002, Hanoi, Vietnam;

[118] Rares Serban, Chadi Barakat, Walid Dabbous, "A CBQ-based Dynamic Resource Allocation Mechanism for Diffserv Routers", SETIT 2003, March 2003, Sousse Tunisia;

[119] D. Grossman, "New Terminology and Clarifications for Diffserv", Request for Comments, RFC3260;

[120] A. Asgari, P. Timintzios, "A Monitoring and Measurement Architecture for Traffic Engineered IP Networks", IEEE/IFIP/IEE IST2001, Teheran, Iran, September, 2001;

[121] H. G. Perros, K. M. Elsayed, "Call Admision Control Schemes: A Review", IEEE Communications Mag., Vol.34, No.11, November 1996, pg.82-91;

[122] L. Massouile, J.Roberts, "Bandwidth sharing: objectives and algorithms", IEEE Infocom 1999, New York, March;

[123] Veselin Rakocevic, John M. Griffiths, "Pyhisical Separation of Bandwidth Resources for Differentiated TCP/IP Traffic", IEEE Infocom 1999, New York, March;

[124] Luigi Rizzo, "Dummnet: a simpe approach to the evolution of network protocols", ACM Computer Communication Review, Vol.27, No.1, 1997, pg.31-41;

[125] "Dummynet tool", http://info.iet.unipi.it/ luigi/ip_dummynet/

List of Abbreviations

ADSL	Asymmetric Digital Subscriber Line
AF	Assured Forwarding
AQM	Active Queue Management
ATM	Asynchronous Transfer Mode
BE	Best Effort
BER	Bit Error Rate
BSR	BootStRap mechanism
BGRP	Border Gateway Routing Protocol
CAC	Call Admission Control
CBQ	Class Based Queuing
CBR	Constant Bit Rate
CBT	Core Based Tree
CDN	Content Delivery Network
CoS	Class of Service
COPS	Common Open Policy Protocol
CPU	Central Processor Unit
CRC	Cyclic Redundancy Check
CR-LDP	Constraint-based Routed Label Distribution Protocol
CSFQ	Core-Stateless Fair Queuing
CSC	Common Signaling Channel
DEC	Decision Message
DiffServ	Differentiated Service
DRR	Deficit Round Robin
DS	Differentiated Service

DQM	Dynamic QoS Management
DRQ	Delete Request State
DSCP	DiffServ Code Point
DTCP	Dynamic Tunneling Configuration Protocol
DRAM	Dynamic Resource Allocation Mechanism
EBU	European Broadcast Union
EF	Expedited Forwarding
FEC	Forwarding Equivalence Class
FF	Fixed-Filter Style
FIFO	First In First Out
FRED	Flow RED
FTP	File Transfer Protocol
GRED	Generalized RED
HDLC	High Level Data Link Control
HDSL	High-data-rate Digital Subscriber Line-
HTB	Hierachical Token Bucket
HFSC	Hierarchical Fair Service Curve
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
ICMP	Internet Control Message Protocol)
IN	Initiating Node
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
ITV	Interactive TV
LAN	Local Area Network
LDP	Label Distribution Protocol
LRM	Local Resource Manager Module
LSP	Label Switching Path
LSR	Label Switching Router
MTU	Maximum Transfer Unit

NCC	Network Control Center
NCR	Network Clock Reference
ND	Never Drop
NIT	Network Information Table
MIB	Management Information Base
MPEG	Moving Pictures Experts Group
MPLS	Multiple Protocol Label Switching
OPN	Open Client Message
PATH	Path Message
PES	Packetized Elementary Streams
PEP	Policy Enforcement Point
PDP	Policy Decision Point
PDR	Per Domain Reservation
PHB	Per-Hop Behavior
PHR	Per-Hop Reservation
PID	Packet Identifier
POP	Point Of Presence
POTS	Plain Old Telephone Service
PPP	Point to Point Protocol
PRIO	Linux Priority Queueing
PQ	Priority Queueing
PSI	Program Specific Information
QoS	Quality of Service
OSPF	Open Shortest Path First
RED	Random Early Detection
REQ	Request Message
RIO	RED Input Output
RIP	Routing Information Protocol
RESV	Reservation Message
RFC	Request for Comments
RPT	Reprt State Message

RR Receiver Report
PSTN Public Switched Telephone Network
RSVP Resource Reservation Setup Protocol
RSVP-TE Resource Reservation Setup Protocol Traffic Extensions
RTCP Real time Transmission Control Protocol
RTP Real time Transmission Protocol
RTT Round Trip Time
SCMP Stream Controm Protocol
SE Shared Explicit Style
SLA Service Level Agreement
SLS Service Level Specification
SQM Static QoS Management
SNMP Simple Network Management Protocol
SR Sender Report
SSQ Synchronize State Request Message
TOS Type of Service
TCP Transmission Control Protocol
TCS Traffic Conditioning Specification
TS Transport Stream
TTL Time To Leave
TE Traffic Engineering
UDP User Datagram Protocol
VDSL Very-high-rate Digital Subscriber Line
VOD Video On Demand
VLL Virtual Leased Line
VPN Virtual Private Network
WF Wildcard-Filter Style
WG Working Group
WRED Weighted RED
WRR Weighted Round Robin
WWW World Wide Web
YESSIR Yet Another Sender Session Internet Reservation

Abstract

All applications used in a company are designed for Internet support. Some of these applications (e.g., VoIP, teleworking) request QoS mechanisms in the network to assure their good behavior. Each QoS mechanism is provisioned in function of SLA (Service Level Agreement) established between user/application and ISPs (Internet Service Provider). The QoS management role is to preserve all specifications from SLA during the contracts' period. The QoS management could be static or dynamic. In the static case, the network QoS management is done by the network administrator, using manual tuning work based on trial-and-error process, during a large scale of time. The dynamic QoS management is based on QoS automated and adaptable mechanismes for a flexible and efficient management of the network resources. We consider in this thesis the following components of the dynamic management of the network resources: signaling protocols, algorithms of resource allocation and admission control, network monitoring mechanisms and QoS trade mechanisms among multiple network domains. We propose criteria classifications for Internet QoS signaling protocols and we analyse using this classification the most used QoS signaling protocols. We study the impact of the service level parameters (e.g., delay, jitter, bandwidth, packet loss) and the low level parametres/network parameters of different QoS mechanism implemented for Linux Diffserv network. Using a Linux Diffserv network we demonstrate that the static QoS management in some cases is not efficient. We propose a dynamic resource allocation mechanism (DRAM) for the core routers. We propose with our algorithm different sharing resources rules.

Keywords: Diffserv, Service Level Agreeement, Network QoS Management, Signaling Protocols, Network Parameters

Résumé

L'Internet sert de support de communication a un grand nombre d'applications dans le cadre des réseaux d'entreprise. Cependant, un certain nombre d'applications multimédia (par exemple la téléphonie sur IP) nécessitent le support d'un ensemble de mécanismes de qualité de service (QoS) dans le réseau. Chaque mécanisme de QoS est provisionné en fonction du contrat (SLA - Service Level Agreement) établit entre l'utilisateur/application et le FAI (Fourniseur d'Accès à Internet). Le rôle important de la gestion de la QoS est de conserver toutes les caractéristiques établies par le SLA pendant toute la durée du contrat. Cette gestion peut être statique ou dynamique. Dans le cas statique, la gestion de la QoS dans le réseau est effectuée, d'une manière expérimentale et non-efficace, manuellement par l'administrateur de réseau sur une grande échelle de temps. La gestion dynamique de la QoS est effectuée en utilisant de façon adaptable et automatique des mécanismes de QoS pour une gestion flexible et efficace des ressources du réseau. Dans cette thèse, nous considérons les éléments suivants qui composent l'architecture pour la gestion dynamique des ressources : la signalisation, des algorithmes d'allocation de la bande passante et de contrôle de ressources, des algorithmes de mesure des ressources dans le réseau et des algorithmes de négociation entre plusieurs domaines administratifs. Nous proposons des critères de classification pour les protocoles de signalisation de QoS dans l'Internet et nous effectuons, en utilisant cette classification, une analyse des protocoles de QoS les

plus utilisés. Nous étudions l'impact entre les paramètres de SLA (bande passante, gigue, délai, perte de paquets) et les paramètres de différents mécanismes fournis par défaut par le système d'exploitation Linux. En utilisant Diffserv conjointement à une plateforme Linux (noté par la suite Linux Diffserv), nous montrons dans certains cas que la gestion statique de la QoS n'est pas efficace. Nous proposons un mécanisme d'allocation dynamique de la bande passante dans les routeurs a l'intérieur du réseau (Linux Diffserv). Avec notre algorithme nous proposons plusieurs règles de partage des ressources.

Mots-clés : Diffserv, Service Level Agreement, Gestion de réseau, protocoles de signalisation, paramètres du réseau