REIHE INFORMATIK
10/2000

A Scalable Share Differentiation Architecture for Elastic and Real-Time Traffic

Albert Banchs[1] and Robert Denda
Praktische Informatik IV
Universität Mannheim
L15, 16
D-68131 Mannheim

[1] C&C Research Laboratories, NEC Europe Ltd.

Abstract

Today's Internet lacks the mechanisms necessary for traffic isolation (to divide resources fairly) and differentiation (to allocate resources to users according to their willingness to pay). In this work, we present a new architecture for relative service differentiation, *Scalable Share Differentiation (SSD)*, which allocates network resources on a user basis and properly provides for both elastic and real-time traffic isolation and meaningful differentiation. *SSD* does not require the storage of any per-flow or per-user information at the core nodes and is inherently designed to work without admission control, making it easier to deploy and manage. In addition, we present how to provide more fine-grained flow control to users without changing the concepts of *SSD*, which permits users to choose different priorities for their packets/flows. This allows the integration of a novel concept for improving the service quality given to user flows: *user-based admission control*. With *user-based admission control*, the network is unaware of admission control decisions, and the user himself can perform the tasks of accepting, rejecting and prioritizing flows.

1 Introduction

The current Internet is built on the best-effort model where all packets are treated as independent datagrams and are serviced on a FIFO basis. This model suffers fundamentally from two problems: the potentially unfair distribution of the network resources and the lack of differentiation.

The potentially unfair resource distribution problem results from the fact that the best effort model does not provide any form of traffic isolation inside the network and relies on the application's behaviour to fairly share the network resources among the users. Therefore the cooperation of the end systems (such as provided by TCP congestion control mechanisms) is vital to make the system work. In today's Internet,

however, such dependence on the end systems' cooperation for resource distribution is becoming increasingly unrealistic. Given the current best-effort model with FIFO queueing inside, it is relatively easy for non-adaptive sources to gain greater shares of network bandwidth and thereby starve other, well-behaved, TCP sources. For example, a greedy source may simply continue to send at the same rate while other TCP sources back off. Today, even many applications such as web browsers take advantage of the best-effort model by opening up multiple connections to web servers in an effort to grab as much bandwidth as possible.

The lack of differentiation relates to the incapacity of the best-effort model to provide a better service to those consumers who are willing to pay more for it. In today's Internet there is a growing demand for user differentiation based on their services' needs. For example, there are many companies relying on the Internet for day-to-day management of their global enterprise. These companies are willing to pay a substantially higher price for the best possible service level from the Internet. At the same time, there are millions of users who want to pay as little as possible for more elementary services. Since the best-effort model treats all packets equally (*same-service-to-all* paradigm), it does not allow Internet Service Providers (ISPs) to differentiate among users as needed.

Over the last ten years, considerable effort has been made to provide Quality of Service (QoS) in the Internet, leading to the specification of an Integrated Services architecture for the Internet (IntServ) [1]. However, research and experience have shown a number of difficulties in deploying the IntServ architecture, due to scalability and manageability problems. The scalability problems arise, because IntServ requires routers to maintain control and forwarding state for all flows passing through them. Maintaining and processing per-flow state for gigabit and terabit links, with millions

of simultaneously active flows, is significantly difficult from an implementation point of view. The manageability problems come from the lack of support for accounting, the high administrative overheads and the complexity of inter-ISP settlement.

The above issues have led to a number of proposals for providing *differentiated services* in the Internet. In those proposals, scalability is achieved by pushing most of the complexity and state to the network edges (where both the forwarding speed and the number of flows are smaller). At the edge, incoming packets are classified among several classes, and core routers do not need to store state for each flow, but can instead process packets using different policies for each traffic class. In a similar way, manageability is achieved by focusing on traffic aggregates instead of individual flows, where a traffic aggregate is a large set of flows with similar service requirements.

Most of the proposed differentiated services architectures solve the potentially unfair resource distribution problem of the best-effort model by performing some type of admission control at the edge of the network (see e.g. [2], [3]). Admission control ensures that no user sends more traffic than he or she is permitted. A key point for admission control is to determine how much traffic a user should be allowed to send, such that the network does not become congested and, therefore, can give the service expected. The difficulty lies, however, in estimating at the edge the congestion level to which the acceptance of a certain amount of traffic would lead[2]. One possibility is to use a static over-provisioned configuration. In this case, since the admitted traffic is always much smaller than the network resources, the danger of congestion is minimized. A more dynamic solution is the use of bandwidth brokers (BB), which are agents responsible for allocating network resources among users. In this approach, the knowledge of the

[2] Note that this problem does not arise in the Integrated Services architecture, since in that architecture, the admission control decision is taken individually at each router on the path between sender and receiver(s) based on the local state information.

network usage is centralized at the BB and the admission decisions to be taken are transfered from this BB to the edge. The design and implementation of BB is an ongoing effort [4].

The authors feel that an architecture solving the problems of the best-effort model while avoiding the complexity of the admission control schemes proposed for current DiffServ architectures would be highly desirable. The goal of this paper is to propose an architecture meeting these requirements.

The rest of the paper is structured as follows: In Section 2, we study different possibilities for dividing the network resources among the users. We distinguish among two groups of applications (elastic and real-time) according to the different network resources they require, and propose a differentiation model that satisfies the needs of both traffic types. In Section 3, an architecture implementing the proposed differentiation model is presented: the *SSD architecture*. An extension to this architecture that allows intra-user discrimination is proposed in Section 4, and Section 5 demonstrates how this intra-user discrimination can be used to perform user-based admission control instead of network-based admission control. Section 6 validates the *SSD architecture* through simulations, and, finally, Section 7 gives a summary and concludes the paper.

2 Differentiation Approach

Research on DiffServ is proceeding along two different directions: those proposals that use admission control and those that do not.

In the approach with admission control, it is possible to control the amount of traffic allowed inside the network. In this way, traffic that would decrease the network service to a level lower than a desired limit can be stopped and an admitted user can be assured of his/her requested performance level. This approach, which we refer to as

Absolute Service Differentiation, can be considered as trying to meet the same goals as IntServ, i.e. absolute performance levels, but pushing complexity (admission control and traffic policing) to the edge and to the BB and thus avoiding per-flow state in the core routers.

The second approach, which we refer to as *Relative Service Differentiation*, cannot prevent flooding of the network using admission control, and the only option to provide service differentiation is to forward packets in the network nodes with a quality according to their relative priority. Therefore, absolute performance levels are not guaranteed and only relative ones can be provided. The advantage of *Relative Service Differentiation* is that it is easier to deploy and manage.

In the relative differentiated services model, users are assigned a relative priority, which determines the network resources they receive. This model must be strongly coupled with a pricing scheme that makes higher priorities more costly.

2.1 Resource Allocation Granularity

For relative service differentiation, different granularities of resource allocation are possible. Current relative differentiated services proposals can be classified into class-based allocation, packet-based allocation and user-based allocation.

2.1.1 Class-based Allocation

With class-based allocation, network resources are assigned to service classes. Each network flow belongs to a certain class and, therefore, shares a common set of resources with other flows in that class.

An example of a class-based relative differentiated services proposal is the Paris Metro Pricing scheme [5], where all classes receive identical treatment inside the network and the classes' price itself is the only differentiation parameter. The idea is that

more expensive classes will have less members and therefore provide a better service. Obviously, this does not always have to be true and situations can occur, where a more expensive class is more congested than another less expensive one. In these situations, useful service differentiation is not achieved.

Applied to relative service differentiation schemes, the class-based approach has the drawback that the service quality associated with a class is undefined, since it depends on the arriving load in that class: as the traffic in the Internet is extremely bursty, the load in each class and consequently its service quality would fluctuate.

2.1.2 Packet-based Allocation

With packet-based allocation, network resources are independently assigned to packets according to the class to which their sender belongs. With this approach, the service quality experienced by the packets of a user are thus independent of the user's sending rate.

An example of packet-based allocation is the architecture proposed in [6]. In [6], the services are distinguished by their different packet forwarding behaviour concerning delay and dropping.

The main drawback of the packet-based allocation approach is the lack of isolation it suffers: a well-behaving user may be affected by misbehaving users sending at a higher rate than they should. A misbehaving user increases the global level of congestion of the network, which harms the performance of all users, including the well-behaving ones.

2.1.3 User-based Allocation

With user-based allocation, each user (or organization) is assigned an individual abstract share representing its assigned networking resources. This share is then used

inside the core to give differentiated service quality to each user.

An example of a user-based differentiation scheme for relative service differentiation is USD [7]. In USD, each user is assigned a share, which is stored by the core nodes of the network. The core nodes use this share to give the packets of the corresponding user their fair part of the forwarding capacity[3].

Since class-based allocation has the problem of service quality fluctuations, and packet-based allocation has the problem of lack of isolation, we believe that user-based relative allocation is preferable. In addition, user-based allocation has the advantage that the user is commonly the entity to which pricing schemes apply. Each user's share, for which he/she pays a certain price, directly corresponds to a certain relative share of the available resources. Thus, with user-based allocation, each user could buy a different relative share, which gives him/her much more flexibility than class-based relative differentiation, where users are aggregated in classes of identical shares.

For these reasons, in the differentiation model that we present in Section 2.3 we assume a user-based resource allocation granularity, and the architecture we propose in Section 3 uses a user-based allocation scheme. The scheme proposed is scalable, since it does not need to store per-user state information at core nodes.

2.2 Differentiation Parameters

Even though there is no wide consensus on the most appropriate performance measure for network resource usage, it is generally agreed that a better network service means a higher bandwidth, a lower delay and a lower likelihood of packet losses. The *differentiation parameter* should provide a useful service differentiation and proper isolation. With useful service differentiation we provide additional value to the users that pay

[3] Note that since USD stores information for each user at core nodes, it has the problem of poorly scaling with respect to the number of users, and might result in implementation problems when applied to core routers in large domains.

more. With proper isolation we guarantee that a misbehaving user cannot obtain more resources than he/she has been assigned.

In the following, we evaluate these three *differentiation parameters* (bandwidth, delay and loss rate) with respect to the utility of the differentiation that they provide to the users.

We will base the study of the utility of the differentiation to the users on utility functions. Utility functions map network parameters (delay, throughput, packet drops, etc.) into the performance of an application: they reflect how the performance of an application depends on the network parameters. With this definition, a differentiation parameter will be of utility to a user when an increase in the user share of this differentiation parameter reflects an increase in the utility experienced by the user. In other words, a differentiation parameter will provide useful differentiation if and only if the utility function is strictly increasing with this differentiation parameter.

2.2.1 Bandwidth

In [8], applications are divided into two groups (elastic applications and real-time applications) and qualitative utility functions are given for each group. Examples of elastic applications are file transfer, electronic mail and remote terminal. These applications are tolerant of delays, and experience a diminishing marginal rate of performance enhancement as bandwidth is increased, so the function is strictly concave everywhere. This is illustrated in Figure 1; an elastic application always benefits from a higher bandwidth, independent of the level of congestion in the network:

$$u_i' = f(congestion), \forall t, s_i > s_j \Rightarrow u_i' > u_j' \tag{1}$$

Therefore, bandwidth always provides useful differentiation to elastic applications.

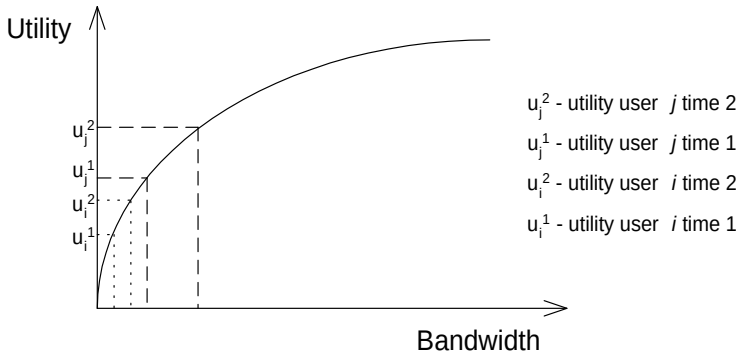

Figure 1: Utility function for elastic applications

Increasing the utility for real-time applications is not as straight-forward. Since real-time applications are delay-sensitive, bandwidth is not enough to provide utility. Further discussion about real-time applications is postponed until the next section.

2.2.2 Delay

The delay has a small impact on elastic applications; therefore the queueing delay differentiation parameter does not provide useful differentiation in this case. On the other hand, real-time applications need their data to arrive within a given delay bound; these applications usually tolerate packets that arrive earlier, but perform badly if packets arrive later than this bound. Examples of such applications are link emulation, audio and video. The qualitative utility function for real-time applications is illustrated in Figure 2.

With this utility function it can be seen that it is not always beneficial to have a smaller delay: a decrease in delay will only be beneficial, if it leads to a delay smaller than the bound.

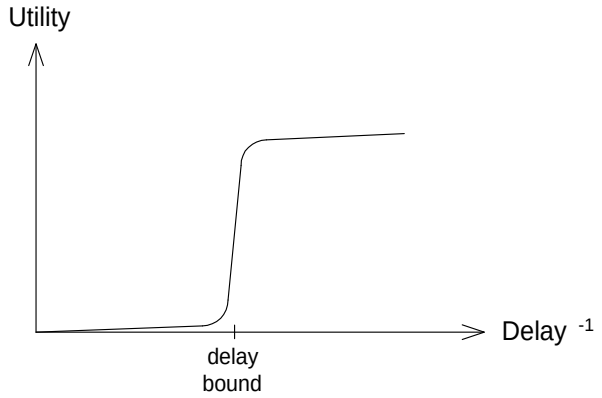

Figure 2: Utility function for real-time applications

2.2.3 Loss-rate

The drawback of this differentiation parameter is that it does not take into account the different behaviours that different users may have regarding losses: some greedy users may not back off when they experience losses and, even if they have a high loss rate, they may experience a high throughput, whereas other users with small loss rate may even back off, resulting in a low throughput.

2.3 Differentiation Model

In this section we present a differentiation model for the two useful differentiation parameters identified in the previous section: bandwidth and delay. As justified in Section 2.1, in our model resources are allocated on a user basis.

Since the two traffic types, i.e., real-time traffic and elastic traffic, have different utility curves, we believe that they must receive differentiated treatment. Real-time traffic is sensitive to delay, whereas for elastic traffic, delay is not as important and bandwidth is the most relevant metric. For that reason, both traffic types need to be treated separately.

The model we have chosen for bandwidth differentiation is the *Proportional Differentiation Model*. This model states that bandwidth should be distributed among

users proportionally to the share they have been assigned. So, if b_i is the bandwidth experienced by user i, and s_i is the share assigned to user i by the network operator, then the proportional differentiation imposes:

$$\frac{b_i}{b_j} = \frac{s_i}{s_j} \forall i, \forall j \tag{2}$$

The basic idea is that, even though the actual bandwidth experienced by each user will depend on the total load of the network, the ratio between bandwidth experienced by the different users will remain fixed and controllable by the network operator, independent of the load.

The choice of this differentiation model for bandwidth is justified by the utility function for elastic traffic (Figure 1): since both the utility function and the differentiation model are strictly increasing, an increase in the share contracted by a user will always reflect an increase in the utility experienced by this user.

Bandwidth differentiation will be applied to both elastic and real-time traffic, since the performance of both traffic types highly depends on the bandwidth received.

For delay differentiation we have chosen a different model: the *Step Differentiation Model*. This model states that delay should be distributed among users in such a way that packets either suffer a very low delay or an infinite delay (i.e. get dropped) depending on the share that has been assigned to their user. So, if d_i is the delay experienced by a packet of user i, and s_i is the share assigned to user i by the network operator, then the step differentiation imposes:

$$d_i = \begin{cases} very\ low, & s_i\ high\ enough \\ \infty, & otherwise \end{cases} \tag{3}$$

The choice of the step differentiation model for delay is justified by the utility

function for real-time traffic (Figure 2): since for real-time packets have to arrive within a given delay bound or otherwise they provide no utility to the user, only the amount of real-time traffic that can be transmitted by the network with a delay lower than the delay bound is accepted, and the rest of the real time traffic is discarded[4]. For elastic traffic, no differentiation in delay is used, since elastic applications are tolerant of delays.

3 Scalable Share Differentiation

In this section, we propose a relative service differentiation architecture that separates elastic and real-time traffic and for both traffic types provides *bandwidth* and *delay differentiation* based on *shares* contracted by users with their ISPs. This architecture does not need to keep per-user state in the core nodes and therefore scales well with the number of users. We have called this architecture SSD (*Scalable Share Differentiation*).

SSD is inspired by the Two-Bit Architecture [3] in that we have one bit in the packet header indicating whether it belongs to a real-time or an elastic application, and for each link we have two queues: a high priority queue for real-time traffic and a low priority queue for elastic traffic.

The SSD architecture has been designed to meet the requirements of isolation and differentiation identified in Section 1 and further developed in Section 2. Isolation is provided by the fact that resources are allocated on a user basis through *user shares*. Proportional differentiation for bandwidth is provided by the algorithm that we present in Section 3.1, and step differentiation for delay in real-time traffic is achieved through the priority scheduling that we propose in Section 3.2.

[4] Note that with the step differentiation model real-time traffic is only allowed to use a limited part of the network capacity; however, the remaining network capacity can still be used by delay insensitive traffic, such as elastic traffic.

3.1 Bandwidth Differentiation Algorithm

In the *SSD* architecture, each user i contracts a *share* S_i with the network operator. This *share* is used to determine the treatment of the user's packets in bottleneck nodes, both for elastic and real-time traffic types.

The share of the user is divided equally among its packets in such a way that each packet gets assigned an *effective share* $W_i = S_i/r_i$, where r_i is the total rate at which the user is transmitting. The share assigned to a user can be understood as some "wealth" that has been given to this user by the network operator. Following this analogy, when the user assigns effective shares he/she is distributing the amount of "wealth" assigned to him/her among his/her packets. Therefore, the effective share of a packet represents its associated "wealth". Obviously, the proposed scheme must assure that the more "wealth" is associated to a packet, the better treatment it should receive from the network.

To assign effective shares to the packets, the transmission rate of each user r_i is measured at the ingress node, and the corresponding value of W_i is inserted into the packet according to the concept of *dynamic packet state (DPS)* [9]. The basic idea of DPS is that each packet header carries some additional state information, in this case the *effective share* W_i of the packets' originator, which is initialized by the ingress node of the diffserv network and processed by the core nodes on the path. Core nodes use this information to update their internal states and the information in the header, before forwarding the packet to the next hop.

Within the *SSD* architecture, the specific DPS mechanism of inserting W_i into the packets is used to provide relative share differentiation: interior nodes use the packets' *effective share* to determine the dropping policy for that user's packets in congestion situations. Whenever there is not enough bandwidth for that traffic type to serve all

incoming packets, the packets with a lower *effective share* should be dropped with higher probability.

That means, when a packet has too low an *effective share*, its user distributed its share among too many packets (i.e., the user sent at a higher rate than allowed according to the user's contracted share). In this case, the router drops some of the packets of that user, thereby reducing his/her packet rate at this link. Then, the node redistributes the *share* of the user among the fewer packets, which leads to a higher value of W_i for the remaining packets of that user.

Therefore, in each node there is a certain value of effective share, below which packets must be dropped with a certain probability to dissolve the congestion. We have called this value the *fair effective share* W_{fair}. The probability of dropping an incoming packet with an effective share lower than W_{fair} will be calculated in the following way: a user's packets with a W_i lower than W_{fair} are dropped with a probability such that the remaining packets of that user get assigned an effective share equal to W_{fair}.

Let us define d_i to be the probability of dropping a packet belonging to user i at some node. Then the redistribution of the user's *share* among the packets that are not dropped leads to:

$$W_i^{new} = \frac{W_i^{old}}{1 - d_i} \tag{4}$$

Taking into account that flows with an *effective share* lower than W_{fair} should adjust themselves to this value (i.e., their new *effective share* should be equal to W_{fair}) and flows with an *effective share* higher than W_{fair} can pass untouched, we have that, given the *fair effective share* W_{fair}, the dropping probability of the packets of user i can be calculated with the formula:

$$d_i = \begin{cases} 0, & W_i > W_{fair} \\ 1 - \frac{W_i}{W_{fair}}, & W_i < W_{fair}, W_i^{new} = W_{fair} \end{cases} \quad (5)$$

With this algorithm, the rates r_i experienced by the users contributing to congestion in a bottleneck link are proportional to their share S_i, as stated by the *proportional differentiation model* described in Section 2.3[5]:

$$\frac{S_i}{r_i} = \frac{S_j}{r_j} = ... = W_{fair} \quad (6)$$

where i, j,... are the users who contribute to the congestion. Note that the users who do not contribute to the congestion (i.e. the users with $W_i > W_{fair}$) are transmitting at a lower rate and thus do not suffer dropping.

The key point of the *SSD* architecture is the estimation of W_{fair}, $\hat{W}_{fair}$, for each congested link. For scalability reasons, $\hat{W}_{fair}$ should be estimated without storing any per-user or per-flow information at the core nodes.

The problem of estimating W_{fair} is similar to the one solved by the CSFQ (*Core Stateless Fair Queueing*) algorithm in [11]. The difference is that while *SSD* focuses on the problem of distributing bandwidth among users, CSFQ focuses on bandwidth distribution among flows.

Both architectures differ not only in the problem they solve, but also in the way they solve it; for the estimation of W_{fair}, *SSD* applies small variations for every incoming packet, while CSFQ applies much bigger changes on a periodic basis. In the following, we present in detail the solution we have adopted within the *SSD* architecture.

In order to determine the *fair effective share* $\hat{W}_{fair}$, we use the fact that, in case of

[5] The bandwidth differentiation algorithm presents a novel fairness distribution based on users, in contrast to traditional fairness criteria dealing with flows. For a thorough analysis see [10].

congestion, the correct value for W_{fair} must lead to a dropping behaviour according to Equation 5 that dissolves the congestion situation, leading to a forwarding rate on the outgoing link equal to the link's capacity. The following equations express this concept.

Let $F(\hat{W}_{fair})$ denote this acceptance rate or offered load to the outgoing link. Then, the *fair effective share* will be the value $\hat{W}_{fair}$ that leads to $F(\hat{W}_{fair}) = C$, where C is the available capacity. According to these considerations, we have

$$F(\hat{W}_{fair}(t)) = \sum_{j \in J_1} r_j + \sum_{j \in J_2} r_j \cdot \frac{W_j}{\hat{W}_{fair}} \tag{7}$$

where J_1 is the set of users i with $W_i \geq \hat{W}_{fair}$, and J_2 is the set of users i with $W_i < \hat{W}_{fair}$.

Given the individual rates r_i we could calculate $\hat{W}_{fair}$ exactly using this formula. Since that would require storing per-user state information at each node, we do not follow this approach, but rather apply the following technique: we continuously measure $F(\hat{W}_{fair})$ for the current value of $\hat{W}_{fair}$ and use this information to adjust $\hat{W}_{fair}$ in such a way that it converges to the actual *fair effective share* W_{fair}.

Let us define a new variable $\alpha = 1/W_{fair}$. Then, the aggregated acceptance rate as a function of $\hat{\alpha}$ is:

$$F(\hat{\alpha}(t)) = \sum_{j \in J_1} r_j + \sum_{j \in J_2} r_j \cdot W_j \cdot \hat{\alpha} \tag{8}$$

Note that F as a function of $\hat{\alpha}$ is continuous and increasing. Thus, an increase in $\hat{\alpha}$ will lead to an increase in $F(\hat{W}_{fair})$, and a decrease in $\hat{\alpha}$ to a decrease in $F(\hat{W}_{fair})$.

In our algorithm, for every incoming packet we increase $\hat{\alpha}$ by a small amount δ in case that $F(\hat{W}_{fair}) < C$, and we decrease $\hat{\alpha}$ by δ otherwise. Let $\hat{\alpha}_i$ be the $\hat{\alpha}$ computed

after the arrival of packet n. Then,

$$\hat{\alpha}_{n+1} = \hat{\alpha}_n + \delta, \delta \begin{cases} < 0, & F(\hat{\alpha}_n) > C \\ > 0, & F(\hat{\alpha}_n) < C \end{cases} \tag{9}$$

A key parameter in the algorithm is δ. If δ is too small, $\hat{\alpha}$ converges too slowly to the desired value, and if it is too big $\hat{\alpha}$ fluctuates too much. There are two considerations that should be taken into account when determining the proper value for δ. The first is: the bigger the distance from the desired point, the larger steps we should take. That is, the bigger $\left|F(\hat{W}_{fair}) - C\right|$, the larger δ should be. Secondly, the queue length can also be used in the calculation of δ. For example, if the queue is almost full, we may want to be more aggressive in decreasing α in order to reduce the queue length.

Therefore, δ can be expressed as a function of the accepted rate (F), link capacity (C), queue length (L) and buffer size (B):

$$\delta = f(F, C) \cdot g(L, B) \tag{10}$$

with f and g satisfying the considerations above. The function f for both elastic and real-time traffic types that we have used in the simulation results given in Section 6 is:

$$f(F, C) = \begin{cases} 0.1 \cdot \frac{C-F}{C}, & \frac{F}{C} < 0.9 \ or \ \frac{F}{C} > 1.1 \\ \\ 0.01 \cdot \frac{C-F}{C} & otherwise \end{cases} \tag{11}$$

Note that with this definition of f, $\hat{\alpha}$ increases linearly with $\left|F(\hat{W}_{fair}) - C\right|$. When F is far from C (10%), $\hat{\alpha}$ increases/decreases 10 times faster.

Since different functions g are used for real-time and elastic traffic, its discussion is

postponed until the next section.

3.2 Traffic Type Separation

The bandwidth differentiation algorithm described in the previous section is used for both traffic types to provide isolation and bandwidth differentiation. In the following, we describe how traffic type separation and delay differentiation are achieved.

The SSD architecture proposed decouples the two traffic types by using two queues for each outgoing link, one low priority queue for elastic traffic and one high priority queue for real-time traffic. This mechanism is depicted in Figure 3.

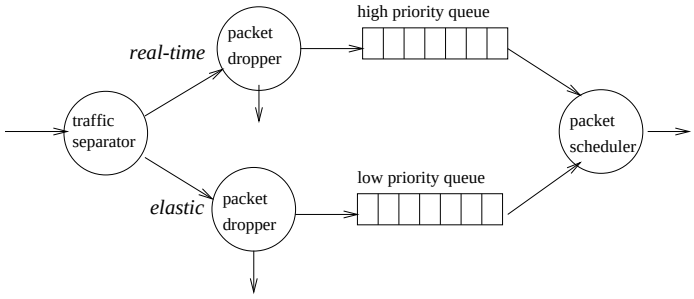

Figure 3: Traffic Type Separation.

The incoming packets are separated according to their type, i.e., elastic or real-time at the traffic separator. The traffic separator then inputs the packet to the corresponding packet dropper, which is based on the bandwidth differentiation algorithm presented in the previous section. For both traffic types, a fair effective share is calculated as described in Section 3.1, i.e. W_{fair}^{el} for elastic traffic and W_{fair}^{rt} for real-time traffic.

Elastic packets will either be allocated to the low priority queue if their effective share W_i is high enough (i.e., $W_i \geq W_{fair}^{el}$). Otherwise they will be dropped with the probability $d_i = 1 - W_i/W_{fair}^{el}$ calculated using Equation 5. The same mechanism applies for real-time packets: they will be allocated to the high priority queue if their effective share W_j is high enough (i.e., $W_j \geq W_{fair}^{rt}$) and they will be dropped with

probability $d_i = 1 - W_j/W^{rt}_{fair}$ otherwise.

With the queueing mechanism presented, elastic and real-time traffic are separated in such a way that the capacity of the link, C, has to be divided among them. A key point in the SSD architecture is to choose the right proportion of the available capacity to be assigned to each type of traffic. If too much capacity is assigned to real-time traffic, we run the risk of filling the real-time queue too much and having real-time packets missing their delay bound due to queueing delay. Consequently, the part of the link's capacity used by real-time traffic needs to be limited to a maximum value, $\frac{1}{k} \cdot C$, to ensure small forwarding delays. $\frac{1}{k}$ is a constant that has to be chosen by the network operator as a function of the amount of real-time traffic to be supported by the network and the desired service quality for this real-time traffic. In order to limit real-time traffic to $\frac{1}{k} \cdot C$, the following function f_{rt} is used in the bandwidth differentiation algorithm described in the previous section:

$$f_{rt} = f(F^{rt}, \frac{1}{k} \cdot C) \tag{12}$$

where f is given in Equation 11 and F^{rt} is the acceptance rate for real-time traffic.

Since keeping the real-time traffic queue as short as possible is important to ensure short queueing delays, the function g_{rt} we have chosen in the bandwidth differentiation algorithm for real-time traffic is:

$$g_{rt}(L, B) = 1 \tag{13}$$

With this choice, the dropping aggressiveness is independent of the queue length. The reason for that is to behave aggressively with respect to the dropping of packets even when the queue is small, in order to avoid queue growth.

Elastic traffic gets assigned the remaining capacity of the link, $C - F^{rt}$. Note that F^{rt} will at maximum equal $\frac{1}{k} \cdot C$; therefore elastic traffic is never starved. Also, elastic traffic will be allowed to use up to the full capacity of the link when there is no higher-priority real-time traffic. The function f_{el} used in the bandwidth differentiation algorithm for elastic traffic will be:

$$f_{el} = f(F^{el}, C - F^{rt}) \tag{14}$$

The dropping of packets for elastic traffic does not need to be as aggressive as for real-time traffic; delay in elastic traffic is not so critical, and therefore having full buffers will not be as harmful. The dropping should be aggressive when the queue is getting full in order to avoid buffer overflow (note that in case of buffer overflow, dropping cannot be controlled any more and we cannot provide differentiation). On the other hand, an aggressive dropping when the queue is not full avoids taking advantage of the buffer capacity in order to support bursts. The dropping policy regarding queue length that we have chosen to use for elastic traffic is a policy that increases its aggressivity as the queue becomes full:

$$g(L, B) = \begin{cases} \frac{L}{B}, & F > C \\ \\ \frac{B-L}{B}, & F < C \end{cases} \tag{15}$$

Delay differentiation in the SSD architecture is provided by means of the two level simple priority queueing shown in Figure 3. Whenever there are packets in the high priority queue, they will be forwarded by the packet scheduler first, and whenever there are no high priority packets to be forwarded, packets are forwarded from the low priority queue for elastic traffic. By assigning real-time packets to the high priority

queue, we achieve a very low delay for those real-time packets that are not dropped, as stated by the *step differentiation model* described in Section 2.3.

Since traffic marked as real-time gets prioritized treatment compared to elastic traffic, the cost to use the high priority queue, which is represented by the effective share W_{fair}^{rt} below which we start having losses, must always be higher for real-time traffic than for elastic traffic. However, W_{fair}^{rt} and W_{fair}^{el} are computed independently as a function of the offered load of real-time and elastic traffic, respectively. Therefore, in the case when there is a much higher density of elastic traffic than real-time traffic in the network, we have a situation where $W_{fair}^{el} > W_{fair}^{rt}$ - i.e. elastic traffic would be more expensive even though it receives a poorer service[6]. In order to correct this undesirable situation, we define a new fair efficient share for real-time traffic,

$$\widetilde{W_{fair}^{rt}} = max(W_{fair}^{rt}, p \cdot W_{fair}^{el}) \tag{16}$$

where p is a constant greater than 1, which is used to express the difference in price between a real-time packet and a packet marked as elastic traffic: thus, for the same price, a user will be able to send p times more traffic marked as elastic than traffic marked as real-time. This is the price that the user has to pay in order to guarantee a low delay.

4 Intra-user Discrimination

In this section we present an intra-user discrimination extension of the SSD architecture. While the differentiation granularity of SSD operates on a user basis, this extension

[6] Note that in this situation real-time traffic would not starve elastic traffic, since real-time traffic is restricted to use a maximum capacity of $\frac{1}{k} \cdot C$, but it would be eating up bandwidth at a lower cost than elastic traffic.

allows more fine grained discrimination between a user's flows using whatever criterion selected by the user.

The extension of *SSD* is, like the basic architecture presented in Section 3, based on *shares*, but with the difference that now flows originating from the same user can have different effective shares. This extension gives the user the possibility to assign arbitrary effective shares to his/her individual flows, thereby giving relative priorities.

To achieve this, the ingress node has to compute the *effective shares* for the packets coming from a user in such a way that the ratios between the *effective shares* given by the user are preserved and the overall effect in the network is the same as if an *effective share* of $W_i = S_i/r_i$ had been assigned to all incoming packets. The first condition is expressed in Equation 17. The second condition is equivalent to saying that the sum of the shares that the different flows of the user originally received should be equal to S_i. This is expressed in Equation 18.

$$\frac{W^{old}_{flow\ 1}}{W^{new}_{flow\ 1}} = \frac{W^{old}_{flow\ 2}}{W^{new}_{flow\ 2}} = \ldots = \frac{W^{old}_{flow\ n}}{W^{new}_{flow\ n}} = \beta \tag{17}$$

$$\begin{aligned} S_i = \quad & W^{new}_{flow\ 1} \cdot r_{flow\ 1} + W^{new}_{flow\ 2} \cdot r_{flow\ 2} \\ & + \ldots + W^{new}_{flow\ n} \cdot r_{flow\ n} \end{aligned} \tag{18}$$

where $flow\ 1, ..., flow\ n$ are the different flows sent by user i, with rates $r_{flow\ 1}, ..., r_{flow\ n}$ respectively. The user marks his/her flows with relative shares $W^{old}_{flow\ 1}, ..., W^{old}_{flow\ n}$, and the network ingress remarks these efficient shares to $W^{new}_{flow\ 1}, ..., W^{new}_{flow\ n}$.

Combining Equations 17 and 18 leads to:

$$S_i = \sum_j r_{flow\ j} \cdot \beta \cdot W^{old}_{flow\ j} \tag{19}$$

where β is the value we need to calculate in order to solve the problem (i.e., to obtain the new *effective shares*).

β could be obtained from Equation 19, but this would require keeping per-flow information at the ingress (specifically, the rates r_j of each flow j).

One way of avoiding per-flow state is calculating the average $\bar{W}$ of all packets of that user; extending Equation 19, we have:

$$\begin{aligned} S_i &= \textstyle\sum_j r_{flow\ j} \cdot \beta \cdot W^{old}_{flow\ j} = \beta \cdot \sum_j r_{flow\ j} \cdot W^{old}_{flow\ j} \\ &= \beta \cdot r_i \cdot \textstyle\sum_j \frac{r_{flow\ j}}{r_i} \cdot W^{old}_{flow\ j} \end{aligned} \tag{20}$$

where the last term of Equation 20 is precisely the average $\bar{W}$ of incoming packets:

$$\bar{W} = \sum_j \frac{r_{flow\ j}}{r_i} \cdot W^{old}_{flow\ j} \tag{21}$$

$\bar{W}$ can be calculated without the need of keeping per-flow state by averaging the values of W_i carried by the incoming packets.

Combining Equation 20 and 21, we obtain a way of calculating β without keeping per-flow state:

$$\beta = \frac{S_i}{\bar{W} \cdot r_i} \tag{22}$$

Therefore, at the ingress, the packets will be marked with the *new effective shares* shown in Equation 23.

$$W^{new}_{packet\ j} = \frac{S_i}{r_i \cdot \bar{W}} \cdot W^{old}_{packet\ j} \tag{23}$$

By this means, the intra-user extension of *SSD* allows the user to discriminate among his or her packets by assigning different weights to them before they are sent to the network operator. The network operator can then change the weights of the

packets according to the user's *share* using Equation 23.

With this approach, a user can discriminate among packets belonging to different flows (*inter-flow discrimination*) or among packets from the same flow (*intra-flow discrimination*)[7]. With *inter-flow discrimination*, it is for example possible to choose to assign more bandwidth to certain people inside an organization (e.g., the CEO)[8]. With intra-flow discrimination we can provide a lower loss probability to certain packets that carry more important information than the other packets (for example, we may want I frames in an MPEG stream to have a lower loss probability than P or B frames; in [12], the benefits of such a discrimination for MPEG are studied). In [10], the use of intra-flow differentiation to properly integrate TCP end-to-end behaviour in the proportional bandwidth differentiation approach described in Section 3.1 is studied.

5 User-based Admission Control

The difficulties of performing network-based admission control in a DiffServ architecture have been outlined in the introductory section of this paper. However, the benefits from using admission control are also high: admission control gives the ability to accept or reject service requests, and thus to meet service commitments for the accepted flows.

In this section we propose a new paradigm for admission control that overcomes the complexity of current admission control proposals for DiffServ. The novelty of the proposed approach is that it is user-based instead of network-based. The fact that the network is unaware of the admission control makes things much simpler. The service commitment provided by our approach is similar to the *Predictive Service* proposed in [13].

[7] Even though the development presented in this section has focused on flows, the result obtained (Equation 23) is independent of flows and can be applied on a packet basis.

[8] Note that in this case, the abstract entity *user* is not an end user, but an organization consisting of several people.

Like the *Predictive Service* of [13], the mechanism we propose relies on measurement-based admission control (see [14]). W_{fair}, which is used at each node in the *SSD* architecture to determine the throughput that each user receives, provides a natural basis for measurement-based admission control: the throughput of a user's flow is determined by the most restrictive (i.e. highest) W_{fair} on the flow's path. Note that W_{fair} takes different values for real-time and elastic traffic (W_{fair}^{rt} and W_{fair}^{el} respectively). The admission control mechanism explained in this section can be applied to both traffic types.

For the measurement-based admission control, we let another piece of information travel with the packets: W_{fair}^{MAX}. W_{fair}^{MAX} starts with a value of 0 at the sender and is updated on each network node according to the following equation:

$$W_{fair_{new}}^{MAX} = max(W_{fair_{old}}^{MAX}, W_{fair_{node}}^{MAX}) \tag{24}$$

where $W_{fair_{old}}^{MAX}$ is the value of W_{fair}^{MAX} carried by the packet before reaching the node, $W_{fair_{node}}^{MAX}$ is the value of W_{fair} at the node for packets going in the opposite direction, and $W_{fair_{new}}^{MAX}$ is the value of W_{fair}^{MAX} after having passed the node.

Therefore W_{fair}^{MAX} gives feedback information to a sender on the return path about the highest W_{fair} on the sender's path.

Assuming that the packets in the forward direction follow the same path, a user can use that feedback information together with the intra-user discrimination technique described in Section 4 to adaptively assign to the flows the necessary effective share $\widetilde{W}$ to reach their destinations without drops:

$$S_{flow} = r_{flow} \cdot W_{fair}^{MAX} \tag{25}$$

where W_{fair}^{MAX} is the information given by the return path value of this flow.

Note that the effective share has to be distributed under the restriction exposed by Equation 26:

$$S_{user} \geq \sum_{i} S_{flow\ i} \tag{26}$$

Using Equation 26, a user can perform admission control in such a way that if a share S_{flow} is requested such that Equation 26 is violated then the request is rejected.

Note that safety margins can be used both in Equation 25 and 26.

The simplicity of the approach presented here results from the fact that it is user-based and therefore does not require complex network mechanisms. To the knowledge of the authors, this proposal is unique in that it can provide a service commitment to flows without the network participating in the signaling when requesting that service[9].

Nevertheless, it is important to note that it has some limitations. Firstly, in this scheme we assume that there exists a return path from the receiver to the sender. It is out of the scope of this work to define how return path information is obtained for usually uni-directional flows. Note also that in this proposal, we assume that the return-path packets follow the same or a similar route than the forward-path packets, which is not guaranteed; nevertheless, we believe that since common routing protocols calculate the routes based on hop-distances and not on load characteristics, the return path usually coincides with the forwarding path. A third restriction is that a user-based admission control scheme cannot control the absolute congestion level of the network. Thus, when the network becomes congested, a user might need to cancel some ongoing

[9] Note that the user-based admission control approach presented here is clearly distinct from end-to-end application layer adaptation based on receiver feedback. The former provides information about the main value on which packet dropping decisions are based, W_{fair}^{MAX}, thereby allowing much more fine-grained control. In addition, the former approach is user-based and permits application-independent admission control; application independency is, for example, important when the user is an organization, and network admission control is to be performed for individual entities inside that organization according to an arbitrary criterion. End-to-end application layer adaptation mechanisms cannot achieve that.

calls in order to give more quality to other calls. This property may be undesirable, but in order to avoid such a situation, the admission control scheme must be network-based and must be coupled with a resource reservation scheme. Performing admission control at the network level has a much higher complexity and, in addition, has the drawback that users may be continously rejected admission to the network, because other users have made an admission reservation that allows them to permanently use all network resources[10].

With the user-based admission control mechanism, it may be beneficial for a user to concentrate his/her share among a few prioritized flows upon congestion in order to allow them to reach their destination without losses, rather than to distribute his/her share among too many flows. With the intra-user extension to SSD, the user has the tool to execute whatever adaptive prioritization strategy is appropriate in a given situation. However, it is up to the user how to use the resources that he/she has been assigned through his/her share in the SSD architecture; the user can choose to use some policy for admission control to provide service commitment to some of his/her flows while rejecting service to others, or can just choose to use the resources in a best-effort way.

6 Simulations

To test the architecture presented in this paper, we have simulated its performance on a network as shown in Figure 4[11]. The flow characteristics used for the simulations are given in Table 1. This network has four 10 Mbps inter-node links, with a propagation delay of 1ms each. There are twelve users, each one sending one single flow. The packet

[10] Note that in this situation, we have temporal unfairness, because the other users who hold their resources might have contracted the same shares as the user who requests a new flow, and only get preferable treatment, because they requested their admission earlier.

[11] All simulations shown in this paper were performed in ns-2 [15].

size used in the simulations for all flows is 1000 bytes. The flows travel along different network paths (user i sends flow i from node s_i to node d_i). Four flows traverse only one inter-node links, four flows traverse three inter-node links, and the remaining four flows traverse all four inter-node links. All elastic traffic flows together offer a load of 10 Mbps, whereas all real-time flows together send at a rate of 1,2 Mbps. The values of k and p used in this simulation are $k = 2$ and $p = 10$, meaning that real-time traffic is twice as expensive as elastic traffic and that a tenth of the bandwidth is assigned to real-time traffic.

Figure 4: Network topology used for the simulations.

flow	type	share	rate	path length
1	real-time	1	0.3	1
2	real-time	1	0.3	3
3	real-time	1	0.3	4
4	real-time	2	0.3	1
5	real-time	2	0.3	3
6	real-time	2	0.3	4
7	elastic	1	2.5	1
8	elastic	1	2.5	3
9	elastic	1	2.5	4
10	elastic	2	2.5	1
11	elastic	2	2.5	3
12	elastic	2	2.5	4

Table 1: Characteristics of the Simulated Flows.

Table 2 displays the throughput (in Mbps), average delay (in ms) and standard deviation (in ms) of the delay experienced by each of the flows[12].

[12] Note that all values shown in the simulation results presented here are rounded to three decimals.

The simulation results demonstrate that the expected behaviour is achieved. For real-time traffic, delays are close to the sum of propagation and transmission delays; queueing delays and, consequently, jitters, are negligible[13]. In contrast, jitter for elastic traffic is much higher (approximately, one order of magnitude higher). The throughput experienced by each user depends on the share; for both traffic types, it can be seen that those users with a higher share suffer no losses, while those with lower shares suffer losses as a consequence of network congestion. Thus the throughputs received are very close to the ones theoretically expected.

flow	theor. throughput	throughput	avg. delay	delay std. deviation
1	0.2	0.211	2.334	0.190
2	0.2	0.211	6.119	0.288
3	0.2	0.173	8.384	0.309
4	0.3	0.295	2.29	0.175
5	0.3	0.288	6.376	0.287
6	0.3	0.282	8.393	0.32
7	2.0	1.985	13.499	5.203
8	2.0	2.012	22.085	4.669
9	2.0	1.974	35.562	6.435
10	2.5	2.509	5.299	1.676
11	2.5	2.496	30.481	5.776
12	2.5	2.474	35.668	6.42

Table 2: Experiment 1: resulting throughput, average delay and standard deviation of the delay.

In a second experiment, we used the same scenario to simulate how well it is guaranteed that packets using the real-time traffic queue must carry a wealth that is at least p times higher than W_{fair}^{el}. For that reason, we increased the shares of the users sending elastic traffic from 1 and 2 to 10 and 20, respectively, thereby increasing the value of W_{fair}^{el} at which the congestion for the elastic traffic queue is dissolved. The other parameters used in the second experiment are identical to the corresponding ones

[13] Note that queueing delay is the only variable component of delay and thus the only one that has an effect on the jitter.

in the first experiment.

Table 3 shows the resulting values from the second experiment. It can be seen that both delay and delay deviation expose a very similar behaviour as in the first experiment. The throughput received by flows 1 through 9 however differs. This is because the bandwidth allocated to real-time traffic is reduced according to the restriction imposed by Equation 16: since the minimum effective share for elastic traffic, W_{fair}^{el}, is higher than in the first experiment due to the high shares of flows 7 to 12, the fair effective share for real-time traffic, $\widetilde{W_{fair}^{rt}}$, is also increased according to Equation 16 to ensure that sending real-time traffic costs at least p times as much than sending elastic traffic. For that reason, all real-time flows perceive higher dropping according to Equation 5 and therefore receive a lower throughput. The bandwidth which is freed by this mechanism is dynamically used up by elastic traffic, as required[14].

flow	theor. throughput	throughput	avg. delay	delay std. deviation
1	0.109	0.116	2.299	0.18
2	0.109	0.106	6.045	0.251
3	0.109	0.104	8.231	0.36
4	0.217	0.180	2.292	0.181
5	0.217	0.184	6.339	0.336
6	0.217	0.228	8.31	0.385
7	2.174	2.163	14.999	3.218
8	2.174	2.193	18.793	3.256
9	2.174	2.065	33.863	4.4414
10	2.5	2.493	4.267	1.317
11	2.5	2.493	29.848	4.128
12	2.5	2.492	33.956	4.453

Table 3: Experiment 2: resulting throughput, average delay and standard deviation of the delay.

In a third experiment, we validated that SSD allows user-based admission control

[14] Note that this process is highly dynamic in that by shifting resources from real-time traffic to elastic traffic, the value of W_{fair}^{el} is decreased, which then again allows more real-time traffic to get through, thereby again decreasing the elastic traffic share of the total bandwidth, and so on. Nevertheless, we validated by simulation that this process is converging to a final distribution (see theoretical throughput values in Table 3).

while maintaining the same network service quality. We simulated the same scenario as in Experiment 2, but with the difference that user 1 applies user-based admission control and only sends as much traffic as indicated by the feedback provided by the network. The other users in this simulation do not apply user-based admission control and send at their full rates. User 1 adjusts its sending rate according to the return-path feedback information provided by SSD. For that reason, we additionally simulated another flow with a rate of 30 kbps traversing the nodes in the opposite direction; with every arriving packet of that flow, user 1 adjusts its sending rate according to the received value of W_{fair}^{MAX} as discussed in Section 5. Table 4 shows the resulting rates and losses of the flows in this scenario. The resulting delay values of this experiment are not presented since they are very similar to the ones exposed in Experiment 2. It can be seen that the fact that flow 1 adapts itself to the available bandwidth has a very small impact on its throughput; the throughput received is very similar to the one in Experiment 2. The losses experienced by flow 1 are very low[15], which validates that *SSD* works with the user-based admission control scheme proposed in Section 5.

7 Summary and Conclusions

As discussed in Section 1, we believe that the main goals for an architecture for differentiated services should be to provide reasonable isolation and differentiation for both elastic and real-time traffic. The presented *Scalable Share Differentiation (SSD)* architecture is an approach for relative service differentiation that fulfils these needs in a scalable and straight-forward way.

In Section 2, we argue that allocating the network resources at a class- or packet-basis is not appropriate, and we identify user-based granularity to be the only one

[15] Note that the losses could be further reduced by applying some safety margin in the estimation of the available bandwidth.

flow	theor. throughput	sending rate	throughput	losses (%)
1	0.109	0.107	0.100	6.542
2	0.109	0.3	0.101	66.333
3	0.109	0.3	0.111	63
4	0.217	0.3	0.180	40
5	0.217	0.3	0.177	41
6	0.217	0.3	0.214	28.667
7	2.174	2.5	2.176	12.96
8	2.174	2.5	2.204	11.84
9	2.174	2.5	2.082	16.72
10	2.5	2.5	2.493	0.28
11	2.5	2.5	2.492	0.32
12	2.5	2.5	2.492	0.32

Table 4: Experiment 3: resulting throughput, average delay and standard deviation of the delay.

providing proper isolation. In addition, we believe that allocating resources on a user-basis has the advantage that it can be directly mapped to pricing schemes, especially since the user is the most natural granularity for pricing.

The network resources assigned to a user are expressed in terms of differentiation parameters. Choosing differentiation parameters that provide utility to the users is essential for the validity of the architecture. We argue that bandwidth and delay are the most appropriate differentiation parameters for elastic and real-time traffic respectively.

The *SSD* architecture, presented in Section 3, is based on user-based effective shares and works without keeping per-flow or per-user state at the core nodes. By separating the two traffic types (elastic and real-time) into two queues with different priorities, we achieve low delays for real-time traffic (which we have called *step differentiation model* for delay). For both real-time and elastic traffic, bandwidth is assigned proportionally to the user's share (which we have called *proportional differentiation model* for bandwidth).

The extension to the *SSD* architecture presented in Section 4 provides the necessary

mechanism for a user to give priority to some of his/her packets compared to others. Therefore, this mechanism allows further differentiation at the user-level, which can be used to discriminate among different flows belonging to one user (*inter-flow differentiation*) or among packets belonging to the same flow (*intra-flow differentiation*).

In Section 5, we demonstrated how the mechanism of intra-user discrimination can be used together with feedback information about the sending path's congestion, to allow for user-based admission control. This mechanism allows the user to assure service commitments to some of his/her flows, while rejecting others because of insufficient resources (represented by the user's share).

The simulation results presented in Section 6, finally, demonstrate that the *SSD* architecture meets the goal of providing a scalable relative service differentiation architecture that properly manages both real-time and elastic traffic without the need of storing any kind of per-user or per-flow information at the network core.

References

[1] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, June 1994.

[2] D. D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Services," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 362–373, August 1998.

[3] K. Nichols, V. Jacobson, and L. Zhang, "A Two-bit Differntiated Services Architecture for the Internet," RFC 2638, July 1999.

[4] "QBone Bandwidth Broker Advisory Council home page," http://www.merit.edu/working-groups/i2-qbone-bb.

[5] A. M. Odlyzko, "Paris Metro Pricing: The Minimalist Differentiated Services Solution," in *Proc. IEEE/IFIP Seventh International Workshop on Quality of Service IWQoS99*, London, UK, June 1999, pp. 159–161.

[6] C. Dovrolis and P. Ramanathan, "A Case for Relative Differentiated Services and the Proportional Differentiation Model," *IEEE Network*, vol. 12, no. 5, pp. 26–34, September 1999.

[7] Z. Wang, "User-Share Differentiation (USD): Scalable Bandwidth Allocation for Differentiated Services," Internet Draft, draft-wang-diff-serv-usd-00.txt, November 1997.

[8] S. Shenker, "Fundamental Design Issues for the Future Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1176–1188, September 1995.

[9] I. Stoica et. al., "Per Hop Behaviors Based on Dynamic Packet States"," Internet Draft, draft-stoica-diffserv-dps.00.txt, Febuary 1999.

[10] Albert Banchs and Robert Denda, "SBSD: A Relative Differentiated Services Architecture based on Bandwidth Sares," Tech. Rep., University of Mannheim, Germany, February 2000.

[11] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," in *Proc. ACM SIGCOMM 98*, Vancouver, Canada, August 1998, pp. 118–130.

[12] A. Albanese, J. Bloemer, J. Edmonds, M. Luby, and M. Sudan, "Priority Encoding Transmission," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1737–1744, November 1996.

[13] D. D. Clark, S. Shenker, and L. Zhang, "Supporting Real-time Applications in an Integrated Services Packet Network: Architecture and Mechanism," in *Proc. ACM SIGCOMM 92*, Baltimore, Maryland, August 1992, pp. 14–26.

[14] S. Jamin, S. Shenker, P. Danzig, and L. Zhang, "A Measurement-based Admission Control Algorithm for Integrated Services Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 56–70, February 1997.

[15] "Network Simulator (ns), version 2," http://www-mash.cs.berkeley.edu/ns.