REIHE INFORMATIK
9/2000

SBSD: A Relative Differentiated Services Architecture based on Bandwidth Shares

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

With this work, we present a scalable relative differentiated services architecture based on bandwidth shares: *Scalable Bandwidth Share Differentiation (SBSD).* SBSD has been designed to provide *isolation* and *differentiation* of user traffic without per-flow or per-user state at the core nodes. In addition, SBSD preserves this isolation and differentiation when being scaled over domain boundaries.

The presented SBSD architecture leads to a weighted maxmin fair bandwidth distribution with respect to each user's sending rate. The novelty of the resulting fairness distribution is that it is user-based, in contrast to traditional fairness criteria dealing with flows.

Even though SBSD operates at a per-user granularity, it allows more fine-grained differentiation, which we use to demonstrate how multicast can be integrated in a straight-forward manner and how SBSD can be optimized for end-to-end TCP congestion control.

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 is increasingly becoming 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 todays 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 allowed to. 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 network usage is centralized at the BB and the admission decisions to be taken are

[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.

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 an admission control module at the edge 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 present a differentiation model that works without admission control. We discuss the effect of using different differentiation parameters with the differentiation model presented and identify an approach that provides isolation and differentiation without admission control. We compare this approach with other proposals that also do not use admission control and point out the weak points. In Section 3, an architecture for our approach that overcomes the weak points identified in the previous sections is proposed: the *SBSD architecture*. Extensions to this architecture for TCP optimization and multicast support are proposed in Sections 4 and 5. Section 6 analyses the differentiation provided by this architecture, and Section 7 validates it through simulations. Finally, Section 8 gives a summary and concludes the paper.

2 Differentiation Approach: Model and Parameters

In this section a differentiation model is proposed. Based on this model, the utility of the different possible differentiation parameters is discussed.

2.1 Differentiation Model

As mentioned in the introduction, 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, the relative priority is determined by shares assigned to the users of the network. This model must be strongly coupled with a pricing scheme that makes higher shares more costly than lower shares; otherwise, everyone would use the highest shares and service differentiation would be ineffective.

The relative differentiated services model we have chosen is the *Proportional Differentiation Model.* The proportional differentiation model states that the network resources should be distributed among users proportionally to the share they have been assigned. So, if q_i are the network resources being used by user i, and s_i is the share assigned to user i by the network operator, then the proportional differentiation imposes:

$$\frac{q_i}{q_j} = \frac{s_i}{s_j} \forall i, \forall j \tag{1}$$

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

The proportional differentiation model presented here is similar to the one presented in [5]. The main difference between the two models is that while the one presented in [5] is class-based (i.e., resources are assigned to a service class according to the share of that class, and then distributed somehow among the users members of this class), ours is user-based (i.e., resources are assigned directly to a user according to his/her share).

The key issue in the proportional differentiation model is to choose the appropriate network resource parameter, which we will call *differentiation parameter* (i.e., q_i in Equation 1). This issue is discussed in more detail in the following section.

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 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 the three *differentiation parameters* that have been proposed in literature (bandwidth, queueing delay and packet drop probability) with respect to the utility of the differentiation and the effectiveness of the isolation they provide to the users.

2.2.1 Bandwidth

When this differentiation parameter is used, each user is allocated a bandwidth proportionally to his or her share:

$$\frac{bw_i}{bw_j} = \frac{s_i}{s_j} \tag{2}$$

Fair bandwidth allocation has the property that it protects well-behaving flows from misbehaving ones. That is, using bandwidth as a *differentiation parameter* provides proper isolation, which is one of the desirable properties identified in the previous section.

The other desirable property is the utility of the differentiation to the users. We will base the study of this property on utility functions. Utility functions map network parameters (delay, throughput, packet drops, etc.) into the performance of an application: it reflects 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.

In [6], 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.

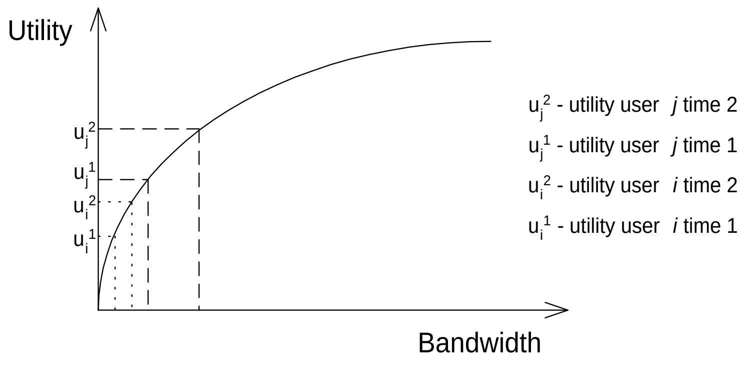

Figure 1: Utility function for elastic applications

This is illustrated in Figure 1; an elastic application always benefits from a higher bandwidth share, independent of the level of congestion in the network:

$$u_i^t = f(congestion), \forall t, s_i > s_j \Rightarrow u_i^t > u_j^t \tag{3}$$

Therefore, bandwidth always provides useful differentiation to 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 Queueing Delay

With this differentiation parameter, the average queueing delays of the packets of a user are inversely proportional to his or her share:

$$\frac{d_i}{d_j} = \frac{s_j}{s_i} \tag{4}$$

where d_i is the queueing delay of the user i's packets at the nodes of the network.

One undesirable behaviour of this *differentiation parameter* is the lack of isolation:

a well-behaving user may be affected by misbehaving users sending at a higher rate than they should. A misbehaving user increases the global queueing delay of the network, which increases the queueing delay of the misbehaving user, but also the queueing delay of the well-behaving ones. The queueing delays of the different users in the network always remain proportional, and therefore some misbehaving users increasing the average delay will also affect the well-behaving ones: $queueing\ delay_{well-behaving\ users} \propto queueing\ delay_{misbehaving\ users} \propto average\ queueing\ delay$

As a consequence, the queueing delay is not well suited as a *differentiation parameter* without admission control, because it does not prevent misbehaving users harming well-behaving ones, not even if the well-behaving ones have a higher share than the misbehaving ones.

Note that this problem did not occur when using bandwidth as the *differentiation parameter*, since in that case the *differentiation parameter* itself plays the role of admission control, letting into the network only the appropriate amount of packets of each flow, and thus precluding the existence of a misbehaving user flooding the network with too many packets and degrading the overall quality.

The other desirable property identified in section 2.2 was the utility of the differentiation to the users. 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; the application does not care if packets arrive earlier, but the application performs very 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

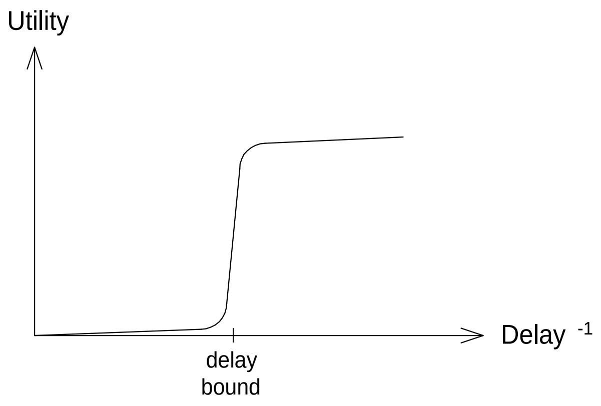

Figure 2: Utility function for real-time applications

higher share (smaller delay): an increase in the share will only be beneficial only if it leads to a delay smaller than the delay limit. As a consequence, using the queueing delay as a differentiation parameter will not always provide useful differentiation for real-time applications. This is because the utility function is not strictly increasing with the delay. This situation is undesirable because we do not guarantee the user a benefit from paying more for a higher share.

As a conclusion, the queueing delay *differentiation parameter* does not provide useful differentiation for elastic applications and does not provide guarantee of useful differentiation for real-time applications; the only property it provides is a higher probability that the utility will be higher for real-time applications.

2.2.3 Packet Drop Probability

With packet drop probability as a differentiation parameter, the fraction of packets of a user dropped are inversely proportional to his or her share:

$$\frac{l_i}{l_j} = \frac{s_j}{s_i} \tag{5}$$

where l_i is the fraction of packets dropped of a user i.

The loss-rate *differentiation parameter* suffers from the same problem of lack of isolation as the delay *differentiation parameter*: a misbehaving user increases the global drop rate, which is also proportional to the drop rate of the well-behaving users.

This parameter 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[3].

2.3 Discussion

Our conclusion in evaluating the possible *differentiation parameters* is that bandwidth is the most sensible *differentiation parameter*. It is the only one that provides proper isolation, and, consequently, the only one that can operate well without admission control. The bandwidth *differentiation parameter* always provides useful differentiation for elastic applications. With real-time applications, bandwidth differentiation cannot guarantee utility, but nor can any of the other *differentiation parameters* alone.

This problem with real-time applications is the price that has to be paid for not having admission control: without admission control, the level of congestion of the network cannot be controlled, and consequently, utility cannot be guaranteed to applications such as real-time applications that need a minimum amount of resources to work. Hence, bandwidth differentiation provides as much useful differentiation as possible without admission control.

[3] Note that from the implementation point of view, differentiation based on bandwidth and differentiation based on packet drop probability are both enforced using packet dropping mechanisms, but while in the former case dropping accords to the ratio between the rate used and the rate assigned to the user, in the latter case, dropping is based only on the share. In case of congestion, with bandwidth differentiation only the users using too many resources will experience losses; in the latter case, all users will experience losses (in different degrees according to their share).

In [7] a relative DiffServ architecture that uses bandwidth as a *differentiation parameter* is proposed. However, this architecture does not scale well with the number of users, since it requires all user shares to be stored in all the routers of the network. Another problem of this architecture is the lack of traffic isolation when crossing ISP boundaries. [7] proposes to aggregate users in a small number of classes in the new ISP, assigning a different share to each class. However, this approach does not provide isolation among users belonging to the same class in the new ISP: one misbehaving user in one class will lead to a bad performance to all the users of this class (also the well-behaving ones).

In [5], two relative DiffServ architectures that use queueing delay and packet drop probability respectively as *differentiation parameters* are proposed. Following the arguments discussed in 2.2, the authors doubt the validity of these *differentiation parameters*. However, if the delay *differentiation parameter* is combined with bandwidth differentiation, the drawbacks discussed in Section 2.2.2 disappear. The question is then whether the further differentiation provided by the delay is worth the additional complexity it involves. In the remaining of this paper we focus only on bandwidth as a *differentiation parameter*.

3 Scalable Bandwidth Share Differentiation

In this section, we propose an architecture that provides *bandwidth differentiation* based on *bandwidth shares* that the users contract with their ISPs. We have called this architecture SBSD (*Scalable Bandwidth Share Differentiation*). It consists of two parts that will be explained next: the intra-domain part and the inter-domain part.

3.1 Intra-domain

In the *SBSD* architecture, each user i contracts a *bandwidth share* S_i within the domain of an ISP. This *bandwidth share* is used to determine the treatment of the users' packets in bottleneck nodes. The goal of *SBSD* is to treat packets in such a way that the throughput experienced by user i increases proportionally with S_i.

The bandwidth share of the user is divided equally among its packets in such a way that each packet gets assigned an *effective bandwidth share* $W_i = S_i/r_i$, where r_i is the total rate at which the user is transmitting. 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)* [8]. The basic idea of DPS is that each packet header carries some additional state information, in this case the *effective bandwidth 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. Interior nodes use this information to possibly update their internal states and to update the information in the header before forwarding the packet to the next hop.

Within the *SBSD* architecture, the specific DPS mechanism of inserting W_i into the packets is used to provide relative bandwidth share differentiation: interior nodes use the packets *effective bandwidth share* to determine the dropping policy for that user's packets in congestion situations. Whenever there is not enough bandwidth at a link to serve all incoming packets, the packets with a lower *effective bandwidth share* should be dropped with higher probability.

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

Thus, by dissolving the congestion situation at that link, we increase the minimum *effective bandwidth shares* of the packets that contributed to the bottleneck up to a certain effective bandwidth share value W_{fair} at which the link capacity is sufficient to forward all remaining packets. We call this value *fair effective bandwidth share* 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 users *bandwidth share* among the packets that are not dropped leads to:

$$W_i^{new} = \frac{W_i^{old}}{1 - d_i} \tag{6}$$

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

$$d_i = \begin{cases} 0 & W_i > W_{fair} \\ 1 - \frac{W_i}{W_{fair}} & W_i < W_{fair}, W_i^{new} = W_{fair} \end{cases} \tag{7}$$

The key point of the *SBSD* 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 Queuing*) algorithm in [9]. The difference is that while *SBSD* focuses on the problem of distributing bandwidth among users, CSFQ focuses on bandwidth distribution among flows. The importance of working on a per-user basis is that it allows aggregation when crossing ISP domains and therefore is a key requirement for a scalable architecture for differentiated services (see Section 3.2).

Both architectures differ not only in the problem they solve, but also in the way they solve it; for the estimation of W_{fair}, *SBSD* 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 *SBSD* architecture.

In order to determine the *fair effective bandwidth share* $\hat{W}_{fair}$, we define it implicitly using the rate with which the algorithm accepts packets as a function of the current estimation of W_{fair}. Let $F(\hat{W}_{fair})$ denote this acceptance rate. Then, the *fair effective bandwidth share* will be the value $\hat{W}_{fair}$ that leads to $F(\hat{W}_{fair}) = C$, where C is the available capacity.

According to the considerations presented above, we have

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

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 bandwidth 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:

$$F(\hat{\alpha}(t)) = \sum_{j \in J_1} r_j + \sum_{j \in J_2} r_j \cdot W_j \cdot \hat{\alpha} \tag{9}$$

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,

$$\hat{\alpha}_{n+1} = \hat{\alpha}_n + \delta, \delta \begin{cases} < 0, & F(\hat{\alpha}_n) > C \\ > 0, & F(\hat{\alpha}_n) < C \end{cases} \tag{10}$$

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 more full the queue is, the more aggressive we should be in decreasing $\hat{\alpha}$. This helps to keep queues small and, therefore, delays short[4]. Therefore, δ can be expressed as a function of the accepted rate (F), link capacity (C), queue length (L) and buffer size (B):

$$\delta = f(F, C) \cdot g(L, B) \tag{11}$$

with f and g satisfying the considerations above. The functions f and g we have used in the simulation results given in this paper are:

$$f(F, C) = \begin{cases} 0.1 \cdot \frac{C-F}{C}, & \frac{F}{C} < 0.9 \ or \ \frac{F}{C} > 1.1 \\ 0.01 \cdot \frac{C-F}{C} & otherwise \end{cases} \tag{12}$$

[4] Note that if we keep all delays short, then bandwidth, which is the parameter we have chosen for *SBSD*, is the right parameter for QoS differentiation. The advantages of keeping queues small have also been identified in [10].

$$g(L,B) = \begin{cases} \frac{L}{B}, & F > C \\ \frac{B-L}{B}, & F < C \end{cases} \tag{13}$$

Note that with these definitions of f and g, $\hat{\alpha}$ increases linearly with $\left|F(\hat{W}_{fair}) - C\right|$. When F is far from C (10%), $\hat{\alpha}$ increases/decreases 10 times faster and when the queue is full, $\hat{\alpha}$ increases slowly and decreases fast.

3.2 Inter-domain

Since the Internet consists of different ISP domains, in order to provide end-to-end QoS, ISPs are required to cooperate. Thus, the QoS behaviour when crossing ISP domains (inter-domain part) is an essential aspect of any DiffServ architecture.

In this section we present an inter-domain extension of the SBSD architecture. The inter-domain extension of *SBSD* is, as the intra-domain part, based on *bandwidth shares*; but in this case it is not a user who contracts a *bandwidth share* to his or her ISP but it is an ISP that contracts a *bandwidth share* with a neighbour ISP. This *bandwidth share* that one ISP contracts with another will be divided among all the users sending from the first ISP to the second. So, for example, if an ISP has 100 users sending to another ISP, and wants them to experience the same quality as one user of the other ISP with a *bandwidth share* of 1 contracted within that ISP, then the first ISP will have to contract a *bandwidth share* of 100 to the second ISP.

When crossing domains, for scalability reasons users have to be somehow aggregated. As it has been explained in the intra-domain part of the architecture, per-user state needs to be maintained at the edges in order to measure each user's rate. If users are not aggregated, boundary routers also need to keep state for every user crossing the router. The number of users crossing edge routers will always be relatively small, but this does not have to hold true for boundary routers between domains. Therefore, if users are not aggregated, boundary routers may need to keep a very large state and will then become bottlenecks concerning scalability. The DiffServ architectures proposed in [11] and [12] suffer from this problem.

One way of aggregating users in our architecture would be to consider all the users sending packets from one domain into another as one user in the second domain with a *bandwidth share* of S_{ISP} (where S_{ISP} is the *bandwidth share* that the ISP of the first domain has contracted to the second). In this case all packets coming from the first domain would get assigned in the second domain an *effective bandwidth share* of $W_{ISP} = S_{ISP}/r_{ISP}$. This solution, however, would not provide proper isolation; if there was a bottleneck in the second domain, all the packets coming from the first domain would be treated in the same way, independently of the *bandwidth share* of its originator, and the bandwidth assigned to each user would not increase proportionally with his or her *bandwidth share*, violating one of the main goals of our architecture. In order to provide proper isolation when crossing domains the packets in the new domain should preserve the ratios between the *effective bandwidth shares* they had in the old domain.

For that reason, the inter-domain architecture has to compute the *effective bandwidth shares* for the packets coming from another domain in such a way that the ratios between the *effective bandwidth shares* of the original domain are preserved and the overall effect in the new domain is the same as if an *effective bandwidth share* of $W_{ISP} = S_{ISP}/r_{ISP}$ had been assigned to all incoming packets. The first condition is expressed in Equation 14. The second condition is equivalent to saying that the addition of the shares that the users from the first domain receive in the second domain should be equal to S_{ISP}. This is expressed in Equation 15.

$$\frac{W_1^{old}}{W_1^{new}} = \frac{W_2^{old}}{W_2^{new}} = \ldots = \frac{W_n^{old}}{W_n^{new}} = \beta \tag{14}$$

$$S_{ISP} = W_1^{new} \cdot r_1 + W_2^{new} \cdot r_2 + \ldots + W_n^{new} \cdot r_n \tag{15}$$

Combining Equations 14 and 15 leads to:

$$S_{ISP} = \sum_j r_j \cdot \beta \cdot W_j^{old} \tag{16}$$

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

β could be obtained from Equation 16, but this would require keeping per-user information (specifically, the rate r_j at which each user j is sending). We already indicated that this solution is undesirable for scalability reasons.

One way of avoiding per-user state is calculating the average $\bar{W}$ of all packets going from the old domain to the new one; extending Equation 16, we have:

$$\begin{aligned} S_{ISP} &= \textstyle\sum_j r_j \cdot \beta \cdot W_j^{old} = \beta \cdot \sum_j r_j \cdot W_j^{old} \\ &= \beta \cdot r_{ISP} \cdot \textstyle\sum_j \frac{r_j}{r_{ISP}} \cdot W_j^{old} \end{aligned} \tag{17}$$

where the last term of Equation 17 is precisely the average $\bar{W}$ of incoming packets. This can be calculated without the need of keeping per-user state:

$$\bar{W} = \sum_j \frac{r_j}{r_{ISP}} \cdot W_j^{old} \tag{18}$$

Combining Equation 17 with $S_{ISP}/r_{ISP} = W_{ISP}$, we obtain a way of calculating β

without keeping per-user state:

$$\beta = \frac{W_{ISP}}{\bar{W}} \tag{19}$$

Therefore, at the *egress* of the old domain the packets will be marked with the *new effective bandwidth shares* shown in Equation 20.

$$W_j^{new} = \frac{S_{ISP}}{r_{ISP} \cdot \bar{W}} \cdot W_j^{old} \tag{20}$$

At the *ingress* of a new domain, it has to be checked that the *bandwidth share* contracted is not violated (i.e., $r_{ISP} \cdot \bar{W} \leq S_{ISP}$), which, like the egress functionality, is easy to compute and does not require per-user state.

This mechanism, therefore, assigns different effective bandwidth shares to the users in such a way that their new effective bandwidth share represents their relative proportion (with respect to their old values) of the ISP's effective bandwidth share in the new domain. Thus, the new average effective bandwidth share is scaled to the effective bandwidth share $\frac{S_{ISP}}{r_{ISP}}$.

Besides providing isolation in a scalable way when crossing domains, the inter-domain extension of the *SBSD* architecture explained in this section has another very important advantage: it allows the user to discriminate among his or her packets by assigning different effective bandwidth shares to them before they are sent to the ISP. The ISP can then change the effective bandwidth shares of the packets according to the user's *bandwidth share* in the same way that it is done at the egress of a domain (Equation 20).

With this approach, a user can discriminate among packets belonging to different applications (*inter-application discrimination*) or among packets from the same application (*intra-application discrimination*). With *inter-application discrimination*, it is for example possible to choose to assign more bandwidth to certain people inside an organization (e.g., the CEO)[5]. With intra-application 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 [13] the benefits of such a discrimination for MPEG are studied).

Another advantage of applying the inter-domain extension of *SBSD* within a user's domain is that the granularity of the discrimination is the user's responsibility; packets can be assigned an effective bandwidth share according to the rate of the flow they belong to, according to the sending rate of the machine where they were originated, etc. The highest grade of isolation is achieved when the effective bandwidth shares are assigned to the packets on a per-flow basis. This form of discrimination at the users site can, for example, be useful to provide proper isolation between TCP and UDP flows, since otherwise, if TCP and UDP streams are marked together, the UDP streams might eat up most of the bandwidth, and the TCP streams will not receive their corresponding share. If the packets are assigned their effective bandwidth shares on a per-flow basis at the first-hop router at the user site, the load should not be high, since a leaf router is usually not crossed by too many flows.

In Section 4 we optimize the behaviour of TCP in the *SBSD* architecture by using intra-application discrimination introduced in this section. In Section 5 we focus on multicast and unicast bandwidth allocation based on the inter-application discrimination of this section. Both cases assume that the effective bandwidth shares are assigned at the users site based on a per flow basis.

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

4 TCP Optimization in SBSD

With *SBSD*, the bandwidth that corresponds to each user is controlled by packet dropping when the user is sending at a higher rate than the rate that corresponds to his or her *bandwidth share*. A potential problem of this way of controlling the bandwidth consumed by a flow is that different transport protocols react in a different way to packet losses.

The most widely used transport protocols in the current Internet are TCP, which reacts to packet losses in order to prevent congestion, and UDP, which is unresponsive to losses (i.e., does not implement any congestion control mechanism at all). In this section we study how to minimize the impact of packet drops on TCP flows in comparison to unresponsive UDP flows.

The problem of TCP flows in the *SBSD* architecture is caused by TCPs congestion mechanisms [14]. This problem is general for DiffServ architectures based on *droppers*. Recent studies have shown the difficulty in guaranteeing to the TCP flows their assigned throughputs in such architectures [15] [16] [17]. When a TCP flow detects packet loss, the flow assumes that the packet loss is due to congestion in the network. A TCP flow tries to resolve this congestion by reducing its transmission rate. This transmission rate may then be less than the rate assigned to the flow according to the user's share, which results in lower throughput.

Figure 3 illustrates this problem when a TCP flow and a UDP flow with the same effective bandwidth shares traverse a common link[6] (both the TCP and the UDP source are configured to send as much as possible). The TCP flow reduces its transmission rate when losses occur, and the bandwidth that becomes available is taken up by the UDP flow, which has no congestion control and never reduces its transmission rate. As

[6] All simulations shown in this paper were performed in *ns-2* [18].

a consequence, we have an unfair distribution of the bandwidth, thereby violating one of the goals stated for our architecture in section 1: *isolation.*

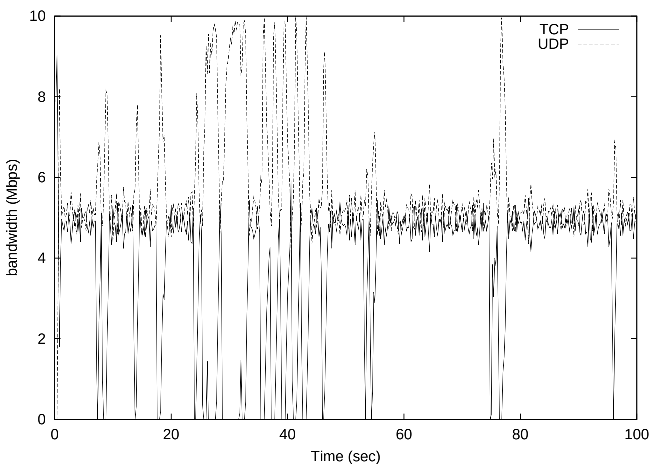

Figure 3: Bandwidth evolution of TCP vs. UDP without intra-application discrimination. *Link capacity: 10Mbps*

The solution to this unfairness problem requires a closer look at TCPs congestion control mechanisms [14], which operate in two modes: the first mode, *fast-recovery*, is triggered by the loss of very few packets, typically one. In this mode, TCP reduces its sending window size, and, with subsequent successful transmissions, increases it again. This mechanism has a relatively small impact on the average transmission rate. The second mode is called *slow-start*, and is typically invoked when a large number of packets are lost. Current implementations of TCP fail to recover from multiple packet losses within a window, and have to rely on the retransmission timer to recover. When multiple packet losses occur, current TCP implementation usually remain silent until the transmission timer goes off, and then enter *slow-start* mode. This behaviour has a much more drastic effect on the TCP performance, since it essentially reduces its transmission rate to zero upon packet loss, thereby reducing the average transmission rate.

Following the explanation above, in order to avoid undesired loss in throughput of a TCP flow, too many closely spaced packet losses should be avoided, since that would force the TCP flow into *slow-start* and (as explained above) cause it to obtain less than its assigned bandwidth. *Intra-application discrimination*, as explained in the previous section, can be used in such a way that, when a loss is detected, a higher weight is assigned to the following packets, making it highly improbable for losses to occur closely spaced.

Equation 21 shows how we have chosen to assign weights:

$$W_n = \begin{cases} 1, & r < n - L \\ 1 + K - \frac{K}{L} \cdot (n - r), & r > n - L \end{cases} \tag{21}$$

where W_n is the weight assigned to packet n and r is the first packet retransmitted after detecting a loss.

After detecting a loss, a weight of $1 + K$ is assigned to the following packet, and then the weight decreases linearly until after L packets the weight has again reached 1[7]. Losses are detected by looking at the sequence number of the TCP header of the packet.

This algorithm has been designed in such a way that it requires little processing and a small state (namely, the sequence number of the last packet transmitted and a counter indicating the last loss detected). In this way, the task of marking the packet for the TCP flow can be easily delegated to the first hop router instead of performing it at the sender's machine [8].

[7] Note that with *intra-application discrimination*, the packets' new effective bandwidth shares are put into proportion to their average value (see Section 3.2). Thus, by assigning higher weights upon loss, we consequently have a time interval during which the packets temporarily get a higher effective bandwidth share than the current average, until the average catches up (slowly, the average value will be reduced again by the algorithm's linear decrease). This effect works in parallel to TCP's rate reduction and subsequent rate increase via *fast recovery*. By enforcing this behaviour, we allow the *fast recovery* mechanism to resolve the loss-indicated congestion situation and avoid *slow starts* when possible.

[8] One of the design guidelines for our architecture has been to avoid modifying end-systems; in this section, for example, we have presented a solution to adapt the *SBSD* architecture to TCP, rather than modifying the TCP itself.

Figure 4 shows for the same scenario the improvements that the marking algorithm proposed in this section provides to TCP: *slow-starts* do not occur any more, and the TCP flow achieves almost the same throughput as the UDP flow. The values of K and L used in this simulation have been $K = 0.33$ and $L = 20$.

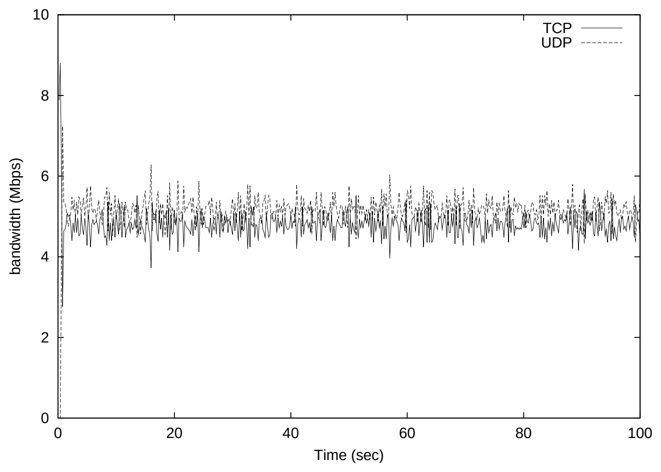

Figure 4: Bandwidth evolution of TCP vs. UDP using SBSD's intra-application discrimination. *Link capacity: 10 Mbps.*

5 Multicast Support in SBSD

In this section we introduce a multicast extension to the *SBSD* architecture explained in Section 3. Multicast is an issue that has not yet been properly addressed in any of the DiffServ architectures so far proposed.

In the *SBSD* architecture, each user gets assigned some "wealth" (the *bandwidth share*), which is divided among the packets originated by this user (the result of this division is the *effective bandwidth share*). Therefore, the *effective bandwidth share* that travels with a packet represents its associated "wealth".

In multicast, packets may be replicated at the network nodes: a packet arriving to a node through the parent link may be forwarded to several child links. When a packet is replicated, new packets with the same source address are introduced into the network. Therefore, according to the philosophy of the *SBSD* architecture explained above, the *bandwidth share* of the user has to be further divided taking into account these new packets. In other words, when a packet from a parent link is replicated in a network node into n_C child packets, the "wealth" carried by the parent packet (*effective bandwidth share* W_{parent}) has to be divided among its children. The easiest way of doing that is assigning to the child packets the following *effective bandwidth share*:

$$W_{children} = \frac{W_{parent}}{n_C} \tag{22}$$

One of the main issues that have been studied in multicast charging is how the efficiency of multicast (with respect to unicast for point-to-multipoint communication) should give additional benefit to the user in terms of charging; the user should be given an incentive to use multicast by charging him or her less than if he or she used unicast for the same purpose. In the *SBSD* architecture, this benefit is clear: a user will receive a better service using multicast than unicast for the same price (i.e., *bandwidth share*). This is because with multicast, in the common links, the *effective bandwidth shares* of all children are accumulated in one packet[9], and thus the probability of this parent packet being dropped is much smaller than if child packets were sent separately (i.e., with unicast). Therefore, in the case that there is a bottleneck in any of those common links, the service with multicast will be better than with unicast.

An issue that arises when a user handles unicast and multicast is how much bandwidth to allocate to multicast with respect to unicast. Note that the user has the capability of deciding how the bandwidth is allocated among his or her flows using to the inter-domain architecture introduced in Section 3.2. The motivation for allocating

[9] Note that with unicast, the sender's rate would be higher and, therefore, the *effective bandwidth share* lower.

different bandwidths to multicast and unicast is that more users benefit from allocating bandwidth to multicast, since a multicast flow has more than one receiver. Therefore, it seems a logical choice to allocate bandwidth to flows as a function of the number of receivers n_R of the flow[10],

$$W = f(n_R) \tag{23}$$

where the bandwidth allocated to a unicast flow is a special case of Equation 23 with $n_R = 1$.

There are different possible choices for $f(n_R)$, between the two extremes $f(n_R) = 1$ and $f(n_R) = n_R$. With $f(n_R) = 1$, bandwidth is allocated independently of the number of receivers (i.e., multicast and unicast flows receive the same bandwidth). This allocation strategy leads to a low *receiver satisfaction*, since it does not take advantage of the fact that by allocating part of the bandwidth of a unicast flow to a multicast one with n_R receivers, we increase the satisfaction of n_R users while only decreasing the satisfaction of one single user.

With $f(n_R) = n_R$, the bandwidth allocated to a multicast flow is proportional to its number of receivers: the bandwidth allocated to the multicast flow is the same bandwidth that would be used if the data were sent to the n_R receivers via separate unicast flows. This allocation strategy leads to a high *unfairness*, because a multicast flow with many receivers will eat up almost all the available bandwidth, letting the unicast flows starve and leaving almost no bandwidth for them.

In [19] it is shown that using $f(n_R) = log(n_R)$ leads to a good tradeoff between *receiver satisfaction* and *fairness*. However, since the per-flow marking is performed at the user site, it is up to the sender to decide which is the allocation strategy that best

[10] It is out of the scope of this paper to define how the marking entity (most likely the first hop router) may get to know the number of receivers of the multicast group, n_R.

fits his or her interests.

6 Theoretical Analysis

With the presented architecture, congestion resolution is performed in such a way that a user i who is sending packets at too high a rate experiences packet dropping that reduces its rate to $r_i = S_i/W_{fair}$, which is the rate that corresponds to the fair bandwidth share W_{fair} for the users share S_i. Let us in the following examine the properties of W_{fair}[11]: when users increase their sending rates r_i, their effective bandwidth shares W_i are decreased. Upon congestion at a link, the *SBSD* algorithm adjusts the effective bandwidth shares in such a way that all remaining packets of users whose flows contribute to the congestion will have a new effective bandwidth share that equals W_{fair}. Since all non-contending users have a higher effective bandwidth share than W_{fair}, W_{fair} is necessarily the minimum of the new set of effective bandwidth shares at that link. In this situation, we have the link fully utilized and, hence, cannot decrease any W_i further without at the same time increasing another effective bandwidth share that equals W_i or that is higher than W_i[12].

We can show that from the above it follows that W_{fair} is the minimum value of the distribution of effective bandwidth shares that minimizes the maximum value of W_i:

Let $W = (W_1, \ldots, W_n)$ be the distribution of effective bandwidth shares that results from the *SBSD* algorithm. Then, given the above, it follows that for any other feasible distribution W' and for each i, it holds that if $W_i > W'_i$ there exists a j such that $W_j < W'_j$ and $W_j \geq W_i$.

[11] In this section we assume that a user does not prioritize its flows for simplicity reasons. The model presented, however, can be extended in a straight-forward manner to the case where users can choose to give priority to some flows with respect to others.

[12] Note that the effective bandwidth shares of non-contending users cannot be decreased, since they already send at their maximum rate

Let $W_m := max(W_1, \ldots, W_n)$. Then, for all distributions $W' = (W'_1, \ldots, W'_n)$ and for all $i \in \{1, \ldots, n\}$ it follows that if $W'_i \geq W_m$ then:

$$max(W'_1, \ldots, W'_n) \geq W'_i \geq W_m = max(W_1, \ldots, W_n) \tag{24}$$

Otherwise (i.e., if $W'_i < W_m$), with the above, there exists a $j \in \{1, \ldots, n\}$ such that $W_j < W'_j$ and $W_j \geq W_m$, thereby yielding

$$max(W'_1, \ldots, W'_n) \geq W'_j \geq W_j = W_m = max(W_1, \ldots, W_n) \tag{25}$$

Thus, in all cases, W minimizes the maximum of all W_i and can therefore be expressed as the distribution that results from the solution to $min(max(W_1, \ldots, W_n))$.

Alternatively, we can express W as the solution to $max(min(W_1^{-1}, \ldots, W_n^{-1}))$, and therefore, using the equality $W_i = r_i/S_i$, W is the distribution provided when solving

$$max(min(\frac{r_1}{S_1}, \ldots, \frac{r_n}{S_n})), \tag{26}$$

which corresponds to a *maxmin fair* distribution concerning the rates r_i, weighted with factors S_i^{-1}. With *SBSD*, the rates r_i are the total rates of all packets sent by user i through the ingress node of the network.

Note that with USD [7], the resulting bandwidth distribution can also be expressed as the solution to $max(min(\frac{r'_1}{S_1}, \ldots, \frac{r'_n}{S_n}))$, but with the difference that the rates r'_i represent the total rate of all flows of user i that traverse the analyzed bottleneck link.

Note also that if all the flows of a same user follow the same path (links) both bandwidth distributions, i.e., [7] and ours coincide. If each user is sending just one flow, they also coincide with the architecture proposed in [9].

Nevertheless, in the other cases, *SBSD* exposes a significant difference: if there is more than one flow per user, with *SBSD* a user is penalized for using many links in the network, while in the [7] approach the user is not penalized (i.e., in our approach, one flow is impacted by the rate of another flow of the same user that is using another link, while this does not happen in [7]).

This difference with [7] seems to be subtle, but is in concept very fundamental: in [7] each user is assigned a share in all links of the ISP, and all nodes need to know all user's shares, while in our approach the user is assigned a global share in the domain, and this share travels with each packet; i.e., the [7] works on a link basis, while ours works on a domain basis, which we believe is a significant advantage, since the domain is normally the granularity on which charging schemes apply. Note, however, that in both cases the user always increases its bandwidth proportionally with its share.

7 Simulations

In this section, we study through simulation the different levels of isolation and differentiation provided by the Relative DiffServ architectures referenced in Section 2.3, i.e., the USD approach [7] and the proportional drop and delay differentiation architecture presented in [5] (hereafter called *DDD*). These results are compared to the performance of the *SBSD* architecture in identical scenarios. The DDD architecture is the result of combining the two architectures proposed in [5], based on queueing delay and packet drop probability differentiation parameters respectively: when a packet arrives, the decision if it should be dropped or not is taken according to its share; if the packet is not dropped, then it is scheduled in such a way that the waiting time in the queue is inversely proportional to its share. The simulation of the *USD* architecture uses class-based queueing (CBQ)[20] as its scheduling mechanism to enforce proportional bandwidth sharing. In all the simulations presented here, there are three senders: user

1 through 3. User 1 and user 2 have each been assigned a share of 2, while user 3 has been assigned a share of 1.

The purpose of these preliminary simulations is, rather than validating the applicability of the *SBSD* architecture in the current Internet, to show the validity of its conceptual approach for isolation and differentiation as compared to the other Relative DiffServ architectures. More complete simulations will be performed and published in the future in order to exhaustively validate the *SBSD* architecture[13].

7.1 Intra-domain

In order to study the intra-domain isolation we have simulated a scenario with a simple network topology and three users (user *i* sends from node *src i* to node *dst i*) using different types of sources (TCP or UDP). This scenario is shown in Figure 5. All the links in the figure have a capacity of 10 Mbps.

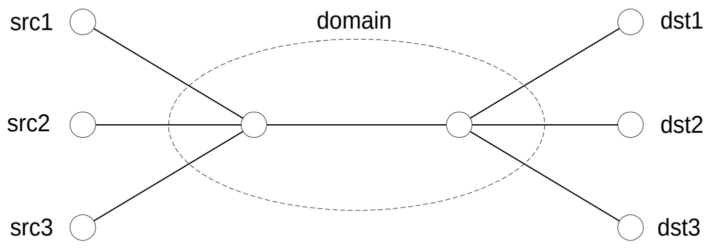

Figure 5: Simulation scenario for intra-domain isolation experiment.

Using this scenario, we performed for each DiffServ approach three tests, the results of which are illustrated in Table 1 (all numbers represent the average throughput in Mbps achieved for each user). In Test 1, all users send TCP flows. In this case, both USD and SBSD[14] provide the required differentiation, while DDD provides a differentiation that is even higher than the one corresponding to the shares assigned to the users. In Test 2, we have one TCP flow of user 1 competing with UDP flows

[13] The simulation source code used in the simulation presented is made publicly available by the authors.
[14] Note that for the presented simulations, the TCP-optimized extension of SBSD is used.

with different data rates of users 2 and 3. The results demonstrate that both USD and SBSD provide the required isolation and ensure the necessary differentiation according to the users' shares. It can be seen, that in this scenario, DDD does neither provide the needed traffic isolation nor the proper differentiation. In Test 3, we simulate three UDP sources, the ones with higher shares send with 4 Mbps, the third user with 8 Mbps. Again, in this case, both USD and SBSD guarantee the proper differentiation, while DDD does not perform as required.

		TEST 1			
user	share	source type	DDD	USD	SBSD
1	2	TCP	4,3	3,9	4,0
2	2	TCP	4,2	3,9	4,0
3	1	TCP	1,4	2,1	2,0

		TEST 2			
user	share	source type	DDD	USD	SBSD
1	2	TCP	0,9	3,9	3,9
2	2	UDP 4Mbps	3,5	3,9	4,0
3	1	UDP 8Mbps	6,4	2,1	2,1

		TEST 3			
user	share	source type	DDD	USD	SBSD
1	2	UDP 4Mbps	3,0	3,9	4,0
2	2	UDP 4Mbps	3,0	3,9	4,0
3	1	UDP 8Mbps	4,0	2,1	2,0

Table 1: Intra-domain simulation results

The results of these three test simulations indicate that drop probability and queueing delay as differentiation parameters cannot provide proper isolation, when misbehaving users send at a higher rate than they should (compare discussion in Section 2.2.2). On the other hand, bandwidth differentiation, as used in USD and SBSD, does provide both proper isolation and differentiation.

7.2 Inter-domain

In order to examine inter-domain isolation, we have simulated a scenario with a two-domain network topology with the bottleneck link in the second domain (see Figure 6; in this scenario, the bottleneck links has a capacity of 10 Mbps, and, as before, user *i* sends from node *src i* to node *dst i*)).

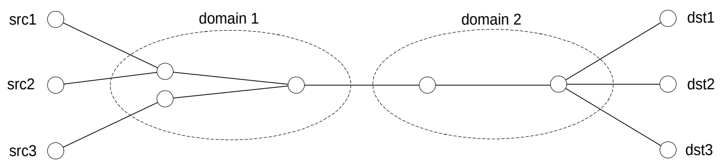

Figure 6: Simulation scenario for inter-domain isolation experiment.

Since DDD does not provide proper isolation in the intra-domain case, we have not included it in the inter-domain simulations and only compare USD and SBSD. The simulated USD implements user aggregation as suggested in [7], i.e., in the second domain, the users from the first domain are aggregated in classes with identical shares. In this case, user 1 and user 2 are aggregated in a class in domain 2 with share 2, while user 3 is the only member of a second class in domain 2 with share 1[15]. The same three tests as in the previous section are performed for the inter-domain case and the resulting figures presented in Table 2. Test 1 shows that USD fails to guarantee proper differentiation due to user aggregation. SBSD preserves the relative shares as discussed in Section 3.2 and, therefore, provides the required differentiation. Test 2 shows, that USD also fails to provide proper isolation, again due to aggregation effects, while SBSD protects the TCP flow against the greedy UDP flows. In test 3, we see again that SBSD ensures both proper isolation and differentiation, while USD suffers from the aggregation problem.

[15] Note that since in USD users are statically assigned to a class, such a situation in which one class is more crowded than another can easily occur.

		TEST 1		
user	share	source type	USD	SBSD
1	2	TCP	3,2	4,0
2	2	TCP	3,0	4,0
3	1	TCP	3,7	1,9

		TEST 2		
user	share	source type	USD	SBSD
1	2	TCP	2,3	3,8
2	2	UDP 4Mbps	4,0	4,0
3	1	UDP 8Mbps	3,7	2,2

		TEST 3		
user	share	source type	USD	SBSD
1	2	UDP 4Mbps	3,1	4,0
2	2	UDP 4Mbps	3,1	4,0
3	1	UDP 8Mbps	3,7	2,0

Table 2: Inter-domain simulation results

8 Summary and Conclusions

As discussed in Section 2, we believe that the main goals for an architecture for differentiated services should be to provide reasonable isolation and differentiation of user traffic both in the intra-domain and in the inter-domain case. In addition we believe that support for multicast, charging schemes and the support of existing end-to-end flow control schemes are necessary requirements for an architecture in order for differentiated services to become widely accepted.

The presented *Scalable Bandwidth Share Differentiation (SBSD)* architecture fulfils all of these needs in a scalable and straight-forward way using *bandwidth* as differentiation parameter.

In Section 2, we demonstrated why we believe that *bandwidth* is the only suitable differentiation paramter for a relative differentiated services architecture without admission control: it is the only parameter providing proper isolation while allowing applications to reasonably profit from relative differentiation.

The presented *SBSD* architecture is based on user-based effective bandwidth shares and works without keeping per-flow or per-user state at the core nodes. As shown in Section 6, *SBSD* leads to a weighted maxmin fair bandwidth distribution and therefore provides proper traffic isolation and differentiation.

In *SBSD*, the shares of the users represent a wealth that the user contracts within a domain and that can be distributed among its packets thereby yielding the effective bandwidth shares that travel with the packets. We believe that since this wealth is directly related to a user in that domain, it provides an ideal basis for charging schemes. In addition, since the wealth is distributed among all packets of a user, independently of what paths these packets follow, it is well-suited for charging the network resources utilized by individual users in a domain. To our knowledge, no other scheme for differentiated services provides such a natural base for charging schemes.

The extension to the *SBSD* architecture presented in Section 3.2 provides the necessary mechanism to aggregate packets with different effective bandwidth shares in such a way that the ratios of their effective bandwidth shares are conserved while the packets' average effective bandwidth share is scaled to a new value. While the *SBSD* architecture without extension applies to flows with homogeneous effective bandwidth shares, the presented extension permits management of flows with varying effective bandwidth shares. As a result, with the extension to the *SBSD* architecture, the inter-ISP aggregation problem is solved when this mechanism is applied at domain boundaries.

Additionally, the same mechanism can be used to provide further differentiation at the user-level: *inter-application differentiation* and *intra-application differentiation. Inter-application differentiation* can be useful, when the user is an organization and wants to provide local differentiation with respect to an arbitrary criterion (such as the originator's ranking in the company, the type of application used, etc.). As shown in

Section 5, inter-application differentiation can be used to provide a natural mechanism allowing the *SBSD* architecture to handle multicast, which is a feature missing in other architectures for differentiated services. *Intra-application differentiation* can be used to prioritize some packets of a stream originated by an application relative to others (such as for layer encoded streams) or for intelligent packet marking in order to allow end-to-end flow control schemes originally designed for the best-effort Internet to properly work with *SBSD*. In Section 4 we demonstrated how intra-application differentiation can be used to properly integrate TCP end-to-end behaviour.

The simulation results presented in Section 7, finally, give a first demonstration of the validity of the *SBSD* approach and present some of its advantages over other relative differentiated services architectures.

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] 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.

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

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

[9] 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.

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

[11] I. Stoica and H. Zhang, “Providing Guaranteed Services Without Per Flow Management,” in *Proc. ACM SIGCOMM 99*, Boston, MA, September 1999, pp. 81–94.

[12] X. Xiao and L. M. Ni, “Internet QoS: A Big Picture,” *IEEE Network*, vol. 13, no. 2, pp. 8–18, March 1999.

[13] 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.

[14] V. Jacobson, "Congestion Avoidance and Control," *Computer Communication Review*, vol. 18, no. 4, pp. 314–329, August 1988.

[15] J. Ibanez and K. Nichols, "Preliminary Simulation Evaluation of an Assured Service," Internet Draft, draft-ibanez-diffserv-assured-eval.00.txt, August 1998.

[16] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Adaptive Packet Marking for Providing Differentiated Services in the Internet," in *Proc. of Int. Conf. on Network Protocols*, October 1998.

[17] I. Yeom and A. L. Narasimha Reddy, "Realizing Throughput Guarantees in a differentiated services network," in *Proc. of ICMCS*, June 1999.

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

[19] A. Legout, J. Nonnenmacher, and E. Biersack, "Bandwidth Allocation Policies for Unicast and Multicast Flows," in *Proc. of IEEE INFOCOM99*, New York, NY, March 1999.

[20] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, August 1995.