The Olympic Service Model: Issues and Architecture

Albert Banchs[1,2], Olga Leon[1,2], Sebastia Sallent[2]

[1] C&C Research Laboratories, NEC Europe Ltd., Heidelberg, Germany
[2] Departament de Telematica, Universitat Politecnica de Catalunya, Spain

Abstract. The Olympic Service Model is a proposal for providing service differentiation in the Internet. With this model, those users who pay more receive a higher amount of network resources, based on a three class granularity (bronze, silver, gold). However, the amount of resources received by a user is not specified, and depends on the level of congestion at a given time. In this paper we analyze the validity and limitations of the Olympic Service Model. We then propose an architecture, the SSD architecture, which provides service differentiation according to this model, both for the intra-domain and the inter-domain cases. Finally, we compare via simulation our approach with other existing architectures of the Olympic Service Model.

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 behavior 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 increasingly becoming unrealistic. Given the current Best Effort model with FIFO queuing 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.

IETF's Differentiated Services (DiffServ) architecture [2–4] solves the potentially unfair resource distribution problem of the Best Effort model by performing some type of admission control at the edge of the network. Admission control ensures that no user sends more traffic than he/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[1]. 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 trans-

[1] 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.

ferred from this BB to the edge. The design and implementation of BB is an ongoing effort [5].

The Olympic Service Model has been designed to solve the problems of the Best Effort model while avoiding the complexity of the admission control schemes proposed for IETF's DiffServ architecture. In this paper we propose an architecture based on the Olympic Service Model and we compare it with existing architectures based on the same model.

The rest of the paper is structured as follows: In Section 2, we present the Olympic Service Model and discuss its validity and limitations. In Section 3, an architecture based on this model is proposed: the Scalable Share Differentiation (SSD) architecture. Section 4 validates the SSD architecture through simulations, comparing it with the other existing approaches for the Olympic Service Model. Finally, Section 5 gives a summary and concludes the paper.

2 The Olympic Service Model

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.

One of the models proposed for the *Relative Service Differentiation* is the Olympic Service Model [3]. This model consists of three service classes: in order of increasing priority, *bronze*, *silver*, and *gold*. Packets are assigned to these classes according to the service contracted by their sender. Then, packets assigned to the gold class experience a better treatment than packets assigned to the silver class, and the same type of relationship exists between the silver and bronze classes. The Olympic Service Model must be strongly coupled with a pricing scheme that makes the gold class more costly than the silver class and the silver class more costly than the bronze class.

2.1 Olympic Service Model for Elastic and Real-time Traffic

In the following, we justify the validity of the Olympic Service Model for Elastic Traffic. We will base the study on utility functions. Utility functions map network parameters into the performance of an application: they reflect how the performance of an application depends on the network parameters. With this definition, the Olympic Service model will be of utility to a user when an increase in the priority of the class contracted by a user reflects an increase in the utility experienced by this user.

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 experience a diminishing marginal rate of performance enhancement as network resources are increased, so the function is strictly concave everywhere. This is illustrated in Figure 1. We can observe from this figure that even though elastic applications benefit from an increasing amount of resources, they can still work with a low amount of network resources.

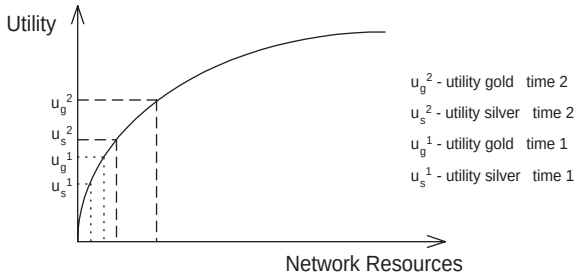

Fig. 1. Utility function for elastic applications

In the Olympic Service Model the amount of network resources received by each user is not defined but depends on the level of congestion in the network. This uncertainty about the amount of network resources associated to a class is not particularly harmful in the case of elastic applications since, as said above, this kind of applications can still work with a low amount of network resources. Also, with the Olympic Service Model, the higher the priority of the class contracted by a user, the more network resources this user will receive. As a consequence, a higher priority class will always lead to a higher utility for elastic traffic, independent of the level of congestion in the network:

$$u_i = f(congestion), \forall t, p_i > p_j \Rightarrow u_i > u_j \quad (1)$$

where p_i is the class priority (i.e. bronze, silver or gold) and u_i is the utility experienced.

Since with the Olympic Service Model, elastic applications always experience a positive performance, and this performance increases with the class priority, we conclude that this model provides a valid service for elastic traffic. Note that, with this approach, admission control can be avoided while still providing a good service for this type of traffic.

Real-time applications, in contrast to the elastic ones, need a minimum amount of network resources to perform well, and perform badly with an amount of resources lower than this threshold. Examples of such applications are link emulation, audio and video. The qualitative utility function for real-time applications is illustrated in Figure 2.

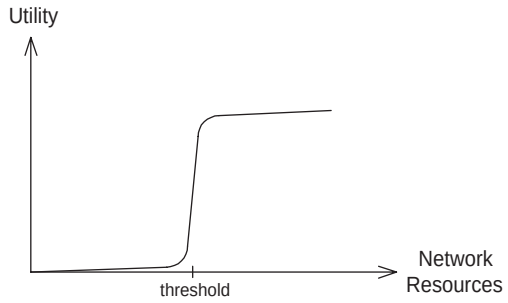

Fig. 2. Utility function for real-time applications

The uncertainty about the amount of network resources associated to a class in the Olympic Service Model may lead to a high priority class receiving a lower amount of resources than the minimum required and, consequently, experiencing a null performance. Also, with this model, an increase in the class priority will not always be beneficial: it will only be beneficial if it leads to an amount of network resources higher than the minimum required.

We conclude that the Olympic Service Model is not appropriate for real-time applications. This kind of applications would be better handled with admission control: without admission control, the level of congestion of the network cannot be controlled, and, as a consequence, utility cannot be guaranteed to applications that need a certain amount of resources to work properly. The Expedited Forwarding of IETF's DiffServ architecture [4] handles real-time traffic in this way. In [7] we provide a real-time traffic extension for the SSD architecture presented here. This extension relies on a user-based admission control scheme to guarantee that a real-time application receives the required amount of resources.

2.2 Sender-based approach

The fact that the Olympic Service Model is sender-based imposes some limitations to its functionality. With a sender-based approach, a user can influence the

treatment experienced by the packets he/she is sending, but not the treatment experienced by the packets he/she is receiving.

The services that fit best the nature of a sender-based approach such as the Olympic Service Model are the *one-to-one* and *one-to-any services for sending*. An example of the one-to-one case could be a VPN service that connects two networks of a company; if the gold service class is contracted for both networks, the VPN service will experience a good quality. An example of the one-to-any case could be a company that does its business on the web and is willing to pay an additional price to provide its users with a fast feel of its web site. If this company contracts the gold service class, its users will experience a good service quality.

While the *one-to-one service for sending* can be relatively easily provided by IETF's DiffServ architecture, the *one-to-any service for sending* is much more complex and it is still an ongoing effort [8].

A *one-to-one service for receiving* does not match the nature of the Olympic Service Model, which is sender-based, but it still could be indirectly provided. An example of such a service could be a user that frequently accesses a specific video server to download movies [9]. With the Olympic Service Model, this user could contract a high quality service with the video server, which would in turn contract the gold service class with the network for the delivery of movies to this specific user. This would result in a good service quality experienced by the user when using this service. Note that in this case the money transaction consists of two steps: a first step from the user to the video server and a second step from the video server to the network operator.

Finally, the *any-to-one service for receiving* cannot be supported by the Olympic Service Model. An example of such a service could be a user willing to pay an extra price for high speed Internet access. Due to the usefulness of this service, the lack of support for it is an important limitation. The problem with any-to-one services is that they necessarily require some kind of signaling between the user and the ingress point of the packets received by him/her. Since the lack of signaling strongly contributes to the simplicity of an architecture, we conclude that the Olympic Service Model trades off functionality with simplicity. Note that within the IETF's DiffServ architecture, the support of any-to-one services has not been addressed yet.

3 Scalable Share Differentiation

In this section, we propose an architecture that implements the Olympic Service Model. In our architecture, each class of the Olympic Service Model (bronze, silver and gold) is mapped to a share, in such a way that the bandwidth experienced by a user is proportional to the share that he/she has contracted. The share associated with each class is chosen by the network operator and depends on the desired level of differentiation between classes; however, the gold class should be mapped to a higher share than the silver, and the silver to a higher than the bronze. This is expressed by the following equation:

$$\frac{r_g}{S_g} = \frac{r_s}{S_s} = \frac{r_b}{S_b} \quad (2)$$

where r_g is the bandwidth that a user would experience if he/she contracted the gold service class, r_s the bandwidth that he/she would experience with the silver class and r_b with the bronze, and S_g, S_s and S_b are the shares associated with each class ($S_g > S_s > S_b$).

The architecture proposed 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*). The SSD architecture consists of two parts: intra-domain and inter-domain.

3.1 Intra-domain

In the SSD architecture, each user i contracts a service class with the network operator, and this service class is mapped to a share S_i (where $S_i = S_g$, S_s or S_b). S_i is used to determine the treatment of the users' packets in bottleneck nodes.

The share of a user is divided equally among his/her packets in such a way that each packet gets assigned a weight $W_i = S_i/r_i$, 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 weights he/she is distributing the amount of "wealth" assigned to him/her among his/her packets. Therefore, the weight of a packet represents its associated "wealth". Obviously, the proposed scheme has to assure that the more "wealth" is associated to a packet, the better treatment it should receive from the network.

To assign weights 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)* [10]. The basic idea of DPS is that each packet header carries some additional state information, in this case the weight 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 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' weight 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 weight should be dropped with higher probability.

That means, when a packet has too low a weight, 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 bandwidth 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, there is a certain value of weight, below which packets must be dropped with a certain probability to dissolve the congestion. We have called this value the *fair weight* W_{fair}. The probability of dropping an incoming packet with a weight lower than W_{fair} will be calculated in such a manner that the packets that are not dropped get assigned a weight that equals 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:

$$W_i^{new} = \frac{W_i^{old}}{1 - d_i} \tag{3}$$

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

$$d_i = \begin{cases} 0 & W_i > W_{fair} \\ 1 - \frac{W_i}{W_{fair}} & W_i < W_{fair}, W_i^{new} = W_{fair} \end{cases} \tag{4}$$

The algorithm used to forward packets in a SSD node, resulting from the equations above, is illustrated in Figure 3.

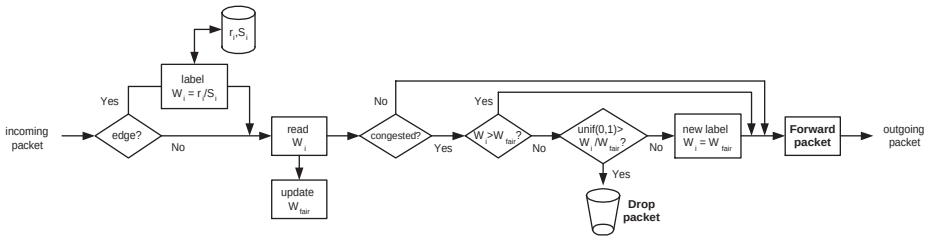

Fig. 3. Forwarding algorithm in SSD

With this algorithm, the rate experienced by flow j of user i is equal to:

$$r_{ij}^{out} = r_{ij}^{in}(1 - d_{ij}) = r_{ij}^{in}\frac{W_i}{W_{fair}^j} = \frac{r_{ij}^{in}}{r_i^{in} \cdot W_{fair}^j} S_i \tag{5}$$

where r_{ij}^{out} is the rate experience by flow j of user i, r_{ij}^{in} is the flow's sending rate, r_i^{in} is the total sending rate of user i and W_{fair}^j is the value of W_{fair} in the bottleneck link of flow j.

From Equation 5, the total rate experienced by user i, r_i^{out}, can be derived:

$$r_i^{out} = \sum_j r_{ij}^{out} = \left(\sum_j \frac{r_{ij}^{in}}{r_i^{in} \cdot W_{fair}^j} \right) S_i \tag{6}$$

From the above equation, it can be seen that the rate received by a user i will depend on the level of congestion of the links traversed by his/her flows, expressed by W_{fair}^j, but this rate will increase linearly with S_i. We conclude that the bandwidth experienced by a user is proportional to the share of the class contracted by this user (S_g, S_s or S_b), which is the objective that we stated for the SSD architecture in Equation 2.

A key point of the SSD architecture is the estimation of W_{fair} for each congested link. For scalability reasons, 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 [11]. In contrast to the SSD architecture, which 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. The advantage of the approach taken in SSD is that it adapts more quickly to network changing conditions (simulation results in [12] show that CSFQ tends to produce large errors under sudden changes from uncongested to congested). The solution adopted within the SSD architecture for the estimation of W_{fair} is presented in detail in [7].

3.2 Inter-domain

Since the Internet consists of different domains, in order to provide end-to-end QoS, domains are required to cooperate. Thus, the QoS behavior when crossing domains (inter-domain part) is an essential aspect of any DiffServ architecture. In the SSD architecture, the inter-domain part is, as the intra-domain, based on shares; but in this case it is not a user who contracts a share to his or her domain (where the share contracted by a user is a function of his/her service class, i.e. bronze, silver or gold), but it is a domain that contracts a share with a neighboring domain. This share that one domain contracts with another will be divided among all the users sending from the first domain to the second. So, for example, if a domain has 100 users sending to another domain, and wants them to experience the same quality as one user of the other domain with a share of 1 contracted within that domain, then the first domain will have to contract a share of 100 to the second domain.

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.

One possible way of aggregating users in our architecture would be to see all the users sending packets from one domain to another in the second domain as one user with a share of S_{d1} (where S_{d1} is the share that 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 a weight $W_{d1} = S_{d1}/r_{d1}$, where r_{d1} is the total sending rate from the first domain to the second. 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 share of their originator, and, as a consequence, the bandwidth assigned to each user would be independent of the service class contracted by him/her, violating thus one of the main goals of the Olympic Service Model. In order to provide proper isolation, when crossing domains the packets in the new domain should preserve the ratios between the weights they had in the old domain.

For that reason, the inter-domain architecture has to compute the weights for the packets coming from another domain in such a way that the ratios between the weights of the original domain are preserved and the overall effect in the new domain is the same as if a weight of $W_{d1} = S_{d1}/r_{d1}$ had been assigned to all incoming packets. The first condition is expressed in Equation 7. 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_{d1}. This is expressed in Equation 8.

$$\frac{W_1^{new}}{W_1^{old}} = \frac{W_2^{new}}{W_2^{old}} = \ldots = \frac{W_n^{new}}{W_n^{old}} = \beta \tag{7}$$

$$S_{d1} = W_1^{new} \cdot r_1 + W_2^{new} \cdot r_2 + \ldots + W_n^{new} \cdot r_n \tag{8}$$

where W_i is the weight of the packets of user i and r_i the rate at which he/she is sending.

Combining Equations 7 and 8 leads to:

$$S_{d1} = \sum_i r_i \cdot \beta \cdot W_i^{old} \tag{9}$$

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

β could be obtained from Equation 9, but this would require keeping per-user information (specifically, the rate r_i at which each user i is sending). We already indicated that this solution is undesirable for scalability reasons.

One way of avoiding per-user state is calculating the average weight $\bar{W}$ of all packets going from the old domain to the new one. Extending Equation 9, we have:

$$\begin{aligned} S_{d1} &= \textstyle\sum_i r_i \cdot \beta \cdot W_i^{old} = \beta \cdot \sum_i r_i \cdot W_i^{old} \\ &= \beta \cdot r_{d1} \cdot \textstyle\sum_i \frac{r_i}{r_{d1}} \cdot W_i^{old} = \beta \cdot r_{d1} \cdot \bar{W} \end{aligned} \tag{10}$$

where the last term of Equation 10, the average weight $\bar{W}$ of incoming packets, can be calculated without the need of keeping per-user state.

Combining Equation 10 with $S_{d1}/r_{d1} = W_{d1}$, we obtain a way of calculating β without keeping per-user state:

$$\beta = \frac{W_{d1}}{\bar{W}} \tag{11}$$

Therefore, when crossing domains, the packets will be marked with the following *new weights*:

$$W_i^{new} = \frac{S_{d1}}{r_{d1} \cdot \bar{W}} \cdot W_i^{old} \tag{12}$$

With this mechanism, the level of differentiation in the second domain is preserved without need of storing per-user state in the boundary routers, and the users coming from the first domain receive in total an amount of network resources equal to S_{d1}.

4 Comparison with existing approaches

In this section, we compare via simulation our architecture with the existing approaches also based on the Olympic Service Model. The purpose of these simulations is, rather than validating the applicability of the SSD architecture in the current Internet, to show the validity of its conceptual approach for isolation and differentiation as compared to the other Olympic Service Model architectures.

For this purpose, we simulated a simple scenario with three users sending through one bottleneck link of 10 Mbps: user 1 with a share of 3 (gold class), user 2 with a share of 2 (silver class) and user 3 with a share of 1 (bronze class). This scenario reflects the level of isolation and differentiation achieved by users belonging to different service classes.

We also simulated another scenario with a variable number of users for each service class: three users for the gold class, two for the silver class and one for the bronze class. This second scenario reflects the level of isolation between the different users of the same class, and the impact of the load in a class.

Finally, we simulated a scenario with a two-domain network topology with a bottleneck link of 10 Mbps in the second domain, in order to study the level of isolation and differentiation provided by the different architectures when crossing domains.

For the above scenarios we used both UDP CBR sources sending at 5 Mbps and endless TCP sources, hereafter referred as UDP and TCP respectively.

4.1 SSD

Tables 1 and 2 show the level of isolation and differentiation provided by the SSD architecture for both the one-user-per-class and several-users-per-class cases. Simulation results show that, when all flows are UDP, SSD provides the desired level of differentiation independent of the number of users per class.

However, when the packet drops of the SSD architecture interact with the congestion control of TCP, results deviate from the desired ones, specially for the case when TCP and UDP flows compete with each other for bandwidth (tests 2 and 5). We conclude from these results that SSD does not provide perfect isolation between UDP and TCP. However, it still provides a fairly good level of isolation, taking into account the different level of aggressiveness of TCP and UDP sources.

The FBA-TCP architecture [13] proposes a modification of the TCP window computation based on the most limiting W_{fair} in the forward path. This architecture, originally designed to be used with CSFQ, could also be used in SSD to improve the performance of TCP.

Table 1. SSD: One user per class

		TEST 1		TEST 2		TEST 3	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	4625	UDP	4979	UDP	4920
2	2	TCP	3744	TCP	3083	UDP	3385
3	1	TCP	1542	UDP	1825	UDP	1687

Table 2. SSD: Several users per class

		TEST 4		TEST 5		TEST 6	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	2052	UDP	2540	UDP	2131
2	3	TCP	2098	UDP	2544	UDP	2120
3	3	TCP	1995	TCP	1276	UDP	2087
4	2	TCP	1502	UDP	1670	UDP	1435
5	2	TCP	1493	TCP	767	UDP	1423
6	1	TCP	758	UDP	843	UDP	699

To test the inter-domain part of the SSD architecture we simulated a scenario in which the users of Table 2 are sending their flows to a second domain with which the first domain has contracted a share of 5, and in this second domain they are competing with a user that has contracted a share of 1 with the second domain. In the simulation results of Table 3 we can see that the differentiation

between the users of the first domain is preserved in the second domain, and in total these users get approximately the share that their domain has contracted with the second domain. These were the objectives of the inter-domain architecture explained in Section 3.2.

Table 3. SSD: Inter-domain

		TEST 7		TEST 8		TEST 9	
user	share	source	Kbps	source	Kbps	source	Kbps
1-d1	3	TCP	1536	UDP	2122	UDP	1726
2-d1	3	TCP	1632	UDP	2086	UDP	1738
3-d1	3	TCP	1635	TCP	831	UDP	1747
4-d1	2	TCP	1050	UDP	1429	UDP	1162
5-d1	2	TCP	1106	TCP	576	UDP	1154
6-d1	1	TCP	546	UDP	677	UDP	594
d1	5	TCP	7507	TCP-UDP	7721	UDP	8121
7-d2	1	TCP	2253	UDP	1860	UDP	1660

4.2 User Share Differentiation (USD)

The User Share Differentiation (USD) architecture [14] also maps each service class into a share. The share corresponding to each user 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. 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.

The results for the intra-domain simulations (Tables 4 and 5) show that USD provides the required isolation and ensures the necessary differentiation according to the service classes. This perfect isolation can be achieved because in USD, in contrast to SSD, per-user state is stored at core routers.

Table 4. USD: One user per class

		TEST 1		TEST 2		TEST 3	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	4999	UDP	4994	UDP	4996
2	2	TCP	3333	TCP	3336	UDP	3336
3	1	TCP	1666	UDP	1668	UDP	1667

For the inter-domain simulations of the USD architecture, we implemented user aggregation as suggested in [14], i.e., in the second domain, the users from

Table 5. USD: Several users per class

		TEST 4		TEST 5		TEST 6	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	2142	UDP	2142	UDP	2142
2	3	TCP	2142	UDP	2142	UDP	2142
3	3	TCP	2142	TCP	2142	UDP	2142
4	2	TCP	1428	UDP	1428	UDP	1428
5	2	TCP	1428	TCP	1428	UDP	1428
6	1	TCP	714	UDP	714	UDP	714

the first domain are aggregated in classes with identical shares. In this case, users 1, 2 and 3 (gold class) are aggregated in a class in domain 2 with share 3, users 4 and 5 (silver class) in a class with share 2 and user 6 (bronze class) in a class with share 1. 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. The simulation results of Table 6 show that USD fails to guarantee proper differentiation due to user aggregation when crossing domains: all users receive the same treatment, independent of their service class. In addition, the inter-domain part of USD also fails to provide proper isolation, again due to aggregation effects: in test 8 we can see that TCP flows starve when sharing a class with UDP flows in the second domain.

Table 6. USD: Inter-domain

		TEST 7		TEST 8		TEST 9	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	1672	UDP	2485	UDP	1667
2	3	TCP	1672	UDP	2504	UDP	1669
3	3	TCP	1655	TCP	3	UDP	1662
4	2	TCP	1665	UDP	3248	UDP	1666
5	2	TCP	1667	TCP	48	UDP	1666
6	1	TCP	1666	UDP	1666	UDP	1666

4.3 SIMA

The Simple Integrated Media Access (SIMA) architecture [15, 16] defines different levels of service based on the Nominal Bit Rate contracted by the user. SIMA provides two types of services, one for real-time traffic and the other for non-real-time. Since the focus of this paper is on elastic traffic, we will only consider the latter.

In SIMA, the sending rate of user i, r_i, is estimated at the ingress of the domain, and, based on this estimation, packets are labeled according to the following formula[2]:

$$L_i = \begin{cases} 6 & if x \geq 6 \\ int(x) & if 0 < x < 6 \\ 0 & if x \leq 0 \end{cases} \quad (13)$$

where x is

$$x = 4.5 - \frac{ln(r_i/NBR)}{ln(2)} \quad (14)$$

and $int(x)$ is the integer part of x.

Then, in core nodes, packets are dropped depending on their label L_i according to the Weighted Random Early Detection (WRED) active queue management algorithm with six separate thresholds, one for each label value.

In the simulations of SIMA, we assigned a NBR of 3 Mbps to the users of the gold service class, 2 Mbps to the users of the silver service class and 1 Mbps to the bronze. Simulation results for the intra-domain case are shown in Tables 7 and 8. These results show that SIMA provides a good level of differentiation in the case of responsive TCP sources (tests 1 and 4), even though the throughputs obtained are not proportional to the NBRs. When dealing with non-responsive UDP sources, however, this feature is lost. In tests 2, 3, 5 and 6, we can see that the bronze service class starves when competing with the gold and silver service classes. This is due to the fact that packet drops in SIMA are not probabilistic but based on thresholds: a user sending too much, thus, will see all his/her packets dropped. Also in tests 2, 3, 5, and 6, the UDP silver sources receive the same treatment as the gold ones. This is due to the coarse granularity of SIMA, which comes from the small number of label values used: a user sending at 5 Mbps receives the same label both for a NBR of 2 and 3 Mbps, since

$$int\left(4.5 - \frac{ln(5/2)}{ln(2)}\right) = int\left(4.5 - \frac{ln(5/3)}{ln(2)}\right) = 3 \quad (15)$$

Finally, in test 5 it can be observed that the level of isolation provided by SIMA between UDP and TCP is not perfect but reasonable.

For the inter-domain simulations, we simulated a scenario in which the users of Table 8 are sending their flows to a second domain with which the first domain has contracted a NBR of 5 Mbps, and in this second domain they are competing with a user that has a NBR of 1 Mbps. In the simulation results of Table 9 we can see that SIMA does not provide proper differentiation when crossing domains: in tests 7 and 9 all users receive the same treatment, independent of their NBR in the first domain. In test 8 it can be seen that the inter-domain part of SIMA does not provide proper isolation either, since the TCP flows starve.

[2] Note that since L_i can only take 6 different values, three bits are enough to store its value.

Table 7. SIMA: One user per class

		TEST 1		TEST 2		TEST 3	
user	NBR	source	Kbps	source	Kbps	source	Kbps
1	3 Mbps	TCP	5325	UDP	4977	UDP	4944
2	2 Mbps	TCP	3063	TCP	4721	UDP	4922
3	1 Mbps	TCP	1572	UDP	287	UDP	132

Table 8. SIMA: Several users per class

		TEST 4		TEST 5		TEST 6	
user	NBR	source	Kbps	source	Kbps	source	Kbps
1	3 Mbps	TCP	2330	UDP	2579	UDP	1991
2	3 Mbps	TCP	2331	UDP	2561	UDP	1978
3	3 Mbps	TCP	2357	TCP	1206	UDP	1985
4	2 Mbps	TCP	1140	UDP	2567	UDP	2015
5	2 Mbps	TCP	1128	TCP	1076	UDP	1995
6	1 Mbps	TCP	693	UDP	0	UDP	34

This inter-domain behavior of SIMA is caused by the fact that the users from the first domain are treated like an aggregate in the second domain.

Table 9. SIMA: Inter-domain

		TEST 7		TEST 8		TEST 9	
user	NBR	source	Kbps	source	Kbps	source	Kbps
1-d1	3 Mbps	TCP	1420	UDP	1991	UDP	821
2-d1	3 Mbps	TCP	1596	UDP	2014	UDP	848
3-d1	3 Mbps	TCP	2023	TCP	1	UDP	819
4-d1	2 Mbps	TCP	818	UDP	1995	UDP	837
5-d1	2 Mbps	TCP	1622	TCP	0	UDP	826
6-d1	1 Mbps	TCP	1238	UDP	2006	UDP	834
d1	5 Mbps	TCP	8717	TCP-UDP	8007	UDP	4985
7-d2	1 Mbps	TCP	1210	UDP	1991	UDP	5011

4.4 Delay Differentiation (DD)

In [17–19] different schedulers for proportional differentiation in delay are proposed. These schedulers basically schedule packets in such a way that the waiting time in the queue is inversely proportional to the share assigned to the corresponding service class.

In order to better understand the performance of this type of schedulers we ran the simulations corresponding to Tables 10 and 11 using the scheduler

implementation of [20]. We can observe from the simulation results that delay differentiation provides a good level of differentiation for TCP traffic alone (tests 1 and 4), since the throughput obtained by a TCP flow is inversely proportional to its round-trip delay. For UDP traffic alone (tests 3 and 6), no differentiation in throughput is obtained; however, for this kind of traffic the delay alone may suffice to provide meaningful differentiation. The main drawback of the delay differentiation approach, however, is expressed by the results of tests 2 and 5. We can observe in these tests that TCP sources almost starve when competing with UDP, which shows that the queuing delay as differentiation parameter cannot provide proper isolation.

We conclude that, even though delay can be an important differentiation parameter for delay sensitive applications, it needs to be combined with bandwidth differentiation in order to provide proper isolation. This is the approach we have taken in the real-time extension of the SSD architecture that we have proposed in [7].

Table 10. DD: One user per class

		TEST 1		TEST 2		TEST 3	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	4884	UDP	4784	UDP	3266
2	2	TCP	3370	TCP	398	UDP	3360
3	1	TCP	1745	UDP	4842	UDP	3376

Table 11. DD: Several users per class

		TEST 4		TEST 5		TEST 6	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	2125	UDP	2481	UDP	1708
2	3	TCP	2132	UDP	2489	UDP	1643
3	3	TCP	2130	TCP	84	UDP	1676
4	2	TCP	1444	UDP	2455	UDP	1679
5	2	TCP	1444	TCP	20	UDP	1652
6	1	TCP	733	UDP	2469	UDP	1640

Inter-domain simulation results for this approach have not been provided since they do not differ from the intra-domain ones.

4.5 Class-Based Allocation (CBA)

A Class-based allocation approach, such as the one in [3], assigns a specific capacity to each service class. Each network flow that belongs to a certain class,

therefore, shares a common set of resources with other flows in that class. This 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 [21], the load in each class and consequently its service quality fluctuates.

This behavior is shown in the simulation results of Tables 12 and 13. This simulations have been performed with the Weighted Round Robin implementation of [22], with a different queue for each class with a weight equal to the share of the class (3 for the gold class, 2 for the silver and 3 for the bronze). Table 12 shows that when the load in each class is uniform, the level of differentiation obtained is the desired. However, when the load in the higher priority classes is larger, the differentiation feature is lost: in the example of Table 13 we can see that all users get the same throughput, independent of the service class they have contracted (bronze, silver or gold). In addition, isolation is not provided inside each class, so the most aggressive sources eat up all the available bandwidth in a class (see test 5 in Table 13).

Table 12. CBA: One user per class

		TEST 1		TEST 2		TEST 3	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	4999	UDP	4994	UDP	4996
2	2	TCP	3333	TCP	3336	UDP	3336
3	1	TCP	1666	UDP	1668	UDP	1667

Table 13. CBA: Several users per class

		TEST 4		TEST 5		TEST 6	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	1666	UDP	2496	UDP	1666
2	3	TCP	1665	UDP	2498	UDP	1670
3	3	TCP	1664	TCP	2	UDP	1663
4	2	TCP	1665	UDP	3284	UDP	1670
5	2	TCP	1667	TCP	48	UDP	1662
6	1	TCP	1666	UDP	1666	UDP	1666

Inter-domain simulation results have not been provided for this approach either, since also in this case they do not differ from the intra-domain ones.

5 Conclusions

The Olympic Service Model is a pricing scheme that provides more network resources to those users who pay more. With this model users are not given absolute guarantees but relative ones: with the gold service class the quality experienced is better than with the silver service class, but this quality is left undefined and depends on the network conditions. Same kind of relationship exists between silver and bronze service classes.

The main advantage of the Olympic Service Model is its simplicity. The fact that the model is sender-based avoids the need of signaling, and its relative nature eliminates the need of admission control. However, this simplicity comes together with some limitations. With a sender-based approach, there are some services like Internet access that cannot be provided. The relative guarantees provided by the Olympic Service Model are well suited for elastic traffic, but not for real-time traffic, which requires of a more complex scheme with some kind of admission control providing absolute guarantees.

We conclude that the Olympic Service Model trades off functionality by simplicity, but still solves some major of the current Best Effort model, such as traffic isolation and service differentiation. Given the current difficulties in the deployment of IETF's DiffServ and IntServ models, mainly due to their complexity, a pricing scheme like the Olympic Service Model could find its application as the next step after the Best Effort model in the evolution of the Internet.

The SSD architecture implements the Olympic Service Model by allocating a fixed number of times more bandwidth to a user of the gold service class than to a user of the silver service class, and the same for the silver and bronze service classes. This level of differentiation between service classes is preserved when crossing domains.

The isolation and differentiation features of the SSD architecture have been validated via simulation. These features have been compared, also via simulation, with other architectures that also implement the Olympic Service Model, namely, User Share Differentiation (USD), Simple Integrated Media Access (SIMA), Delay Differentiation (DD) and Class-Based Allocation (CBA). Simulation results show that the SSD architecture is the only one that provides the isolation and differentiation features for both the intra-domain and the inter-domain cases.

References

1. R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, June 1994.
2. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, December 1998.
3. J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group," RFC 2597, June 1999.
4. V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB," RFC 2598, June 1999.

5. "QBone Bandwidth Broker Advisory Council home page," http://www.merit.edu/working-groups/i2-qbone-bb.
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. A. Banchs and R. Denda, "A Scalable Share Differentiation Architecture for Elastic and Real-time Traffic," in *Proceedings of 8th International Workshop on Quality of Service IWQoS'2000*, Pittsburg, PA, June 2000.
8. M. Brunner, A. Banchs, S. Tartarelli, and H. Pan, "A one-to-any Probabilistic Assured Rate Per-Domain Behavior for Differentiated Services," Internet draft, February 2001, Available at http://www.icsi.berkeley.edu/~banchs/papers/one2any-AR-PDB.pdf.
9. C. Kalmanek, D. Shur, S. Sibal, C. Sreenan, and J. Merwe, "NED: A Network-Enabled Digital Video Recorder," IEEE LANMAN 2001, March 2001.
10. I. Stoica et. al., "Per Hop Behaviors Based on Dynamic Packet States," Internet Draft, draft-stoica-diffserv-dps.00.txt, Febuary 1999.
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. Z. Cao, Z. Wand, and E. Zegura, "Rainbow Fair Queueing: Fair Banwdith Sharing Without Per-Flow State," in *Proceedings of IEEE INFOCOM'2000*, Tel-Aviv, Israel, March 2000.
13. R. Kapoor, C. Cassetti, and M. Gerla, "Core-Stateless Fair Bandwidth Allocation for TCP flows," in *Proceedings of IEEE ICC 2001*, Helsinki, Finland, June 2001.
14. Z. Wang, "User-Share Differentiation (USD): Scalable Bandwidth Allocation for Differentiated Services," in *HPN'98*, Vienna, 1998.
15. K. Kilkki, "Simple Integrated Media Access," draft-kalevi-simple-media-access-01.txt, Internet draft, June 1997.
16. M. Loukola, J. Ruutu, and K. Kilkki, "Dynamic RT/NRT PHB Group," draft-loukola-dynamic-00.txt, Internet draft, November 1998.
17. C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," in *Proceedings of ACM SIGCOMM'99*, Cambridge, MA, September 1999, pp. 109–120.
18. T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "Relative Delay Differentiation and Delay Class Adaptation in Core-Stateless Networks," in *Proceedings of IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.
19. Y. Moret and S. Fdida, "A Proportional Queue Control Mechanism to Provide Differentiated Services," in *International Symposium on Computer System*, Belek, Turkey, October 1998.
20. "WTP packet scheduler for ns2," http://www.cis.udel.edu/~dovrolis/ns-WTP.tar.
21. A. Feldmann, A. C. Gilbert, and W. Willinger, "Data networks as cascades: Investigating the multifractal nature of the Internet WAN traffic," in *Proceedings of ACM SIGCOMM'98*, Vancouver, Canada, August 1998, pp. 25–38.
22. "Diffserv Model for the ns2 simulator," http://www7.nortel.com:8080/CTL/.