End-to-End QoS with Scalable Reservations

Rui Prior Susana Sargento
Sérgio Crisóstomo Pedro Brandão

Technical Report Series: DCC-2003-01

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

End-to-End QoS with Scalable Reservations

Rui Prior, Susana Sargento, Sérgio Crisóstomo, Pedro Brandão
Department of Computer Science & LIACC
Faculty of Sciences, University of Porto
Rua do Campo Alegre, 823, 4150-180 Porto, Portugal
{rprior, ssargento, slc, pbrandao}@ncc.up.pt

1st April 2003

Abstract

This paper proposes a new QoS architecture that provides end-to-end QoS support. The architecture is based on scalable per-flow signaling and resource reservation for aggregates of flows at both core and access networks. The underlying architecture is based on DiffServ, where the edge nodes perform policing of the incoming aggregates in order to ensure conformance to the aggregate reservation. Although in this model the signaling is per-flow based, several techniques and algorithms are developed aiming at the minimization of the computational complexity and, therefore, the improvement of the signaling scalability. More specifically, a label switching mechanism is developed with the goal of reducing the signaling message processing time at each router. Moreover, the architecture includes soft reservations where the expiration timers are scalably implemented with a complexity that is low and independent of the number of reservations. This architecture is then able to scalably support both IntServ service models in high speed networks.

1 Introduction

In the actual Internet the network resources are equally distributed by all users and applications. Therefore, only best effort service is supported. However, new services and applications are becoming more demanding and its characteristics and transmission requirements are much diversified. Moreover, since the multimedia services are composed by several traffic flows with different requirements, each flow needs to receive an adequate treatment independent of the other traffic. Finally, different users have also different requirements and this must be reflected in the quality given to its services. Therefore, there is an increasing need for the service providers to implement mechanisms that ensure that each service or set of services receives the required Quality of Service (QoS).

Two main architectures were proposed by the IETF in order to introduce QoS support in the Internet: Integrated Services (IntServ) and Differentiated Services (DiffServ). The IntServ architecture [9] was the first one to emerge. In the IntServ resources are reserved for individual flows, i.e., on a per-flow basis, using the Resource ReSerVation Protocol (RSVP) [10]. Based on per-flow reservations subject to admission control, it provides both a guaranteed service, with strict guarantees regarding both bandwidth and delay, and a controlled load service, akin to a lightly loaded best-effort network, meant for loss and delay tolerant services which still require better than best-effort treatment. Intserv's strengths lie in both its capability of providing strict QoS guarantees and adequately efficient resource usage (particularly in the case of multicast flows), but it is fundamentally flawed regarding scalability. In fact, the need to implement complex packet scheduling algorithms and to classify every packet based on both layer 3 and layer 4 header information is unbearable to high-performance routers in core networks.

The DiffServ architecture [7] emerged as a scalable alternative to IntServ. In this architecture there are no resource reservations and flows are aggregated in classes according to specific characteristics. Here services have a different treatment according to their class, but there is no admission control mechanism to limit the number of flows in the network. Resource allocation is configured by management procedures, in conformance to the Service Level Agreements (SLA) established between adjacent domains. DiffServ's main strength lies in its low complexity and inherent scalability. Due to its essentially static nature, though, resource usage is usually sub-optimal, there is no admission control for flows, which implies that all flows belonging to a class may be degraded, and there are issues with cross interaction between traffic flows with different ingress/egress node pairs yet to be solved.

With the objective of benefiting from the virtues of both IntServ and DiffServ and solving (minimizing) its problems, several architectures have been proposed in the literature. However, none of these architectures ensures simultaneously the strict and differentiated QoS support and the maximization of the usage of network resources without scalability concerns.

In this paper we propose a new QoS architecture that provides end-to-end QoS support. This architecture is based on scalable per-flow signaling and resource reservation for aggregates of flows at both transit (core) and access networks. The underlying architecture is based on DiffServ. Individual flows are aggregated by service class. The service classes are mapped to per-hop behaviors (PHB), and aggregate classification is performed based on the DS field of the packet header. Policing of the incoming aggregates is performed at the edge nodes in order to ensure conformance to the aggregate reservation. Core nodes do not have policing functionalities. In this model the signaling is performed on a per-flow basis in order to improve the network resource utilization. The reservations are unidirectional, sender initiated and soft state. The signaling protocol is implemented as an extension to the RSVP protocol, providing the functionality needed for our model. Although the signaling is per-flow based, several techniques and algorithms have been developed aiming at the minimization of the computational complexity and, therefore, the improvement of the signaling scalability. More specifically, a label switching mechanism was developed with the goal of reducing the signaling message processing time at each router. This mechanism allows each router to have direct access to the flow reservation structure (with O(1) complexity) through the introduction of a label that is directly associated to the memory address where the reservation state for the flow is stored. Beyond this technique our model also includes soft reservations where the expiration timers are scalably implemented with a complexity that is low and independent from the number of reservations. Resuming, the developed model provides both strict and soft end-to-end QoS assurances with increased scalability.

This paper is organized as follows. Section 2 addresses other QoS architectures and compares them against the one developed in this paper. Section 3 presents an overview of the system architecture. The service differentiation and traffic control techniques are described in section 4 and the label switching mechanism is described in section 5. The signaling protocol and its messages are presented in section 6 and the considerations about the overall system scalability are presented in section 7. Finally, section 8 presents the most important conclusions and describes the current work being developed and the extensions to be applied in the architecture.

2 Related Work

A number of QoS architectures have been discussed in the literature. For example, in an effort to combine the virtues of both IntServ and DiffServ, [4] proposes the operation of IntServ over DiffServ networks with the addition of flow admission control capabilities to the edge nodes of a statically provisioned DiffServ network region. This architecture does not solve the sub-optimal network resource usage nor the cross interaction problems. Another approach at providing IntServ over DiffServ networks consists on the use of Bandwidth Brokers [19] that have sufficient knowledge of resource availability and network topology to make admission control decisions and resource reservation. In the last years, the concept of Bandwidth Brokers has gained preponderance, due to the requirement of end-to-end QoS through heterogeneous networks. However, although in an architecture with Bandwidth Brokers the core routers will be freed from performing admission control decisions, the Bandwidth Broker needs to manage the overall network domain and to store information about all elements, flows and paths in the network domain, which can introduce scalability concerns.

The architecture defined and developed in this paper is able to provide the same QoS guarantees as the architecture managed by Bandwidth Brokers and has the advantage of distributing the functions assigned to the Bandwidth Broker between all nodes in the network domain in a scalable way.

Another architecture, named SCORE [17] (Stateless CORE), has been proposed, which makes use of a technique called Dynamic Packet State (DPS). This technique consists on carrying the needed state information in the header of every packet, updating it at every router along the path, thus keeping the stateless character of the network. This solution has several problems: (i) all routers need to implement the same scheduling mechanisms; (ii) the processing in the core routers is still high; and (iii) it is impossible to use header compression and security headers. Our proposed model does not imposes a scheduling mechanism and improves scalability by performing classification and scheduling on a class basis. Although the nodes need to maintain state information for every reservation, the available routers nowadays have sufficient memory to store information for several hundreds of thousands of flows.

Another attempt at providing per-flow admission control while avoiding the need to maintain per-flow information at the core routers is the solution termed "Egress Admission Control" [12]. By monitoring and controlling egress routers' class-based arrival and service envelopes, egress routers perform resource management and admission control, without any coordination among backbone nodes or per-flow management. Being based on a probabilistic model, Egress Admission Control has the obvious drawback of not being able to assure that the strict QoS guarantees needed by hard real-time services are provided under every circumstance. Moreover, this solution suffers from a resource stealing problem when there are no active flows with the same ingress/egress pair as the flow requesting admission. In contrast, our model supports both soft and strict QoS guarantees and does not suffer from resource stealing problems.

Several schemes have recently been developed in which end hosts probe the network, assess the performance properties of the probes, and admit or reject the flow accordingly [5, 11, 13, 15]. Such schemes have the advantage that no network control is required since all QoS functionality is performed by hosts. However, the flow setup times can be high (which is unbearable for most of the services), they introduce themselves congestion in the network, the measures are imprecise (since they are based on a collection of samples), and, in multiclass networks, probe flows can be admitted stealing network resources from previously admitted flows (which increases the network congestion). None of these problems affects our proposed model, since the flow setup time is in the order of the round trip time and there is no flow intrusion (probes) in the network.

Finally, the reservation of resources for aggregates of flows (instead of individual flows) has been proposed in the context of DiffServ/MPLS (MultiProtocol Label Switching) architecture as a means of reducing significantly the signaling load and the state information stored at routers, while still providing the same QoS for real time flows. An extension to RSVP that allows RSVP signaling messages to be hidden inside an aggregate was recently defined to support aggregation [2]. In the simplest case, all edge routers reserve bandwidth end-to-end, i.e., between ingress and egress routers of the network domain; this reservation can be updated in bulks much larger than the individual flow's bandwidth. With aggregation, signaling messages are only exchanged when the aggregate's bandwidth needs to be updated. Due to cross interaction between aggregates with different ingress/egress node pairs, aggregation implies a tradeoff: with high aggregation, more flows are rejected and the utilization decreases; with small aggregation the decrease in utilization is neglected but the number of signaling messages remains high. This problem is increased in large network domains: the need to set-up a large number of end-to-end aggregates can lead to poor resource utilization. To alleviate this problem the partition of the domain in areas and the configuration of end-to-end aggregates between the area border routers was proposed in [14]. The proposed model in this paper achieves the same QoS guarantees as the aggregation one, with the added benefit of its performance in terms of resource utilization and blocking probability being significantly increased. As a drawback, the signaling overhead will be larger. However, the introduction of algorithms and mechanisms that improve the scalability of the signaling protocol decreases the signaling processing in the routers.

3 General System Architecture

Our architecture (figure 1) combines the strict end-to-end QoS guarantees of a signaling based approach with per-flow reservations subject to admission control (both in terms of bounded delay and minimal loss) with the efficiency and scalability provided by flow aggregation and by several mechanisms and algorithms developed (described in sections 5 and 6).

The underlying architecture for our model is strongly based on the DiffServ architecture. In fact, not only is the underlying infrastructure very DiffServ-like, but the two models may peacefully coexist in the same domain. The resource allocation for the two models can be administratively configured. On top of the DiffServ-like infrastructure, we added signaling based reservations subject to admission control.

The network is partitioned in domains, consisting of core and edge nodes. Individual flows are aggregated according to the service class. The service classes are mapped to DiffServ compatible per-hop behaviors (PHB), and aggregate classification is performed based on the DS field of the packet header. Edge nodes perform policing of the incoming aggregates in order to ensure conformance to the aggregate reservation (the sum of the individual reservations for all flows in the aggregate). Since the traffic is policed in all edge nodes, the core nodes do not have policing functionalities.

The reservations are unidirectional, sender initiated and soft state. There are several advantages to this approach: (1) it is more directly mapped to existing business models, in which a provider sells a service (e.g., streaming) at a price which includes not only the content but also the delivery with the appropriate quality; (2) the sender has more information on the flow characteristics, and therefore is able to parameterize the reservation in a way that assures appropriate quality without waste; (3) unidirectional reservations inherently support path asymmetry. Reservations are setup by means of a signaling protocol. This protocol works in a hop by hop basis, and is meant to be as simple and efficient as possible, as it must run in every node along the path, including core nodes.

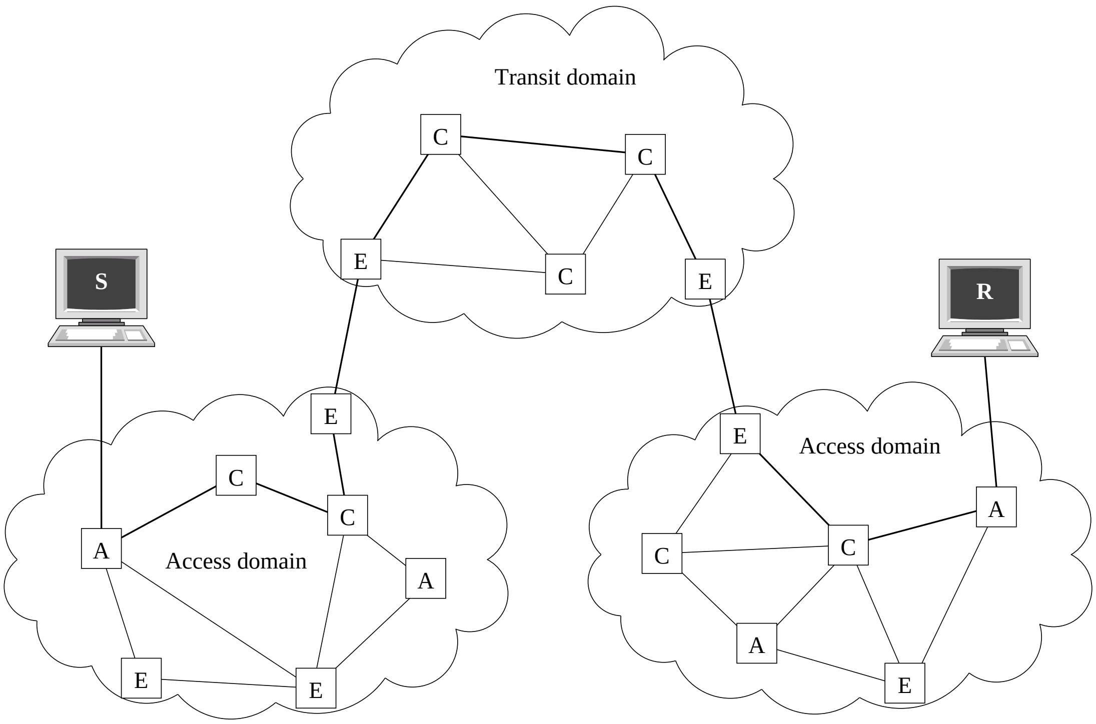


Figure 1: Architecture overview

Every node along the end-to-end path must perform admission control for every flow and must, therefore, run the signaling protocol. In order to achieve scalability, we had not only to make the signaling protocol very simple, but also to ensure that every task performed by it has a low order of computational complexity. To this end, we developed the label switching mechanism described in section 5. The labels are directly mapped to the memory addresses where the structures containing flow reservation information are stored. This allows routers to directly access these memory structures instead of performing an expensive lookup in a flow reservation table. This label switching mechanism is also very useful when per-flow policing is performed in access networks, since the flow's information is directly accessed. The processing load of the routers in the end-to-end path is also decreased with the development of an algorithm that provides an efficient implementation of expiration timers used in the soft reservations. This algorithm has a small and fixed complexity, compared to currently available algorithms, and provides enough functionality. In the following sections we describe in more detail each architecture component.

4 Service Differentiation and Traffic Control

This section presents the service classes available in our model, its mapping within the DiffServ architecture, and the resource reservation and control techniques applied to the traffic flows, such as the admission control, traffic shaping and policing.

4.1 Service Classes and Mapping Strategies

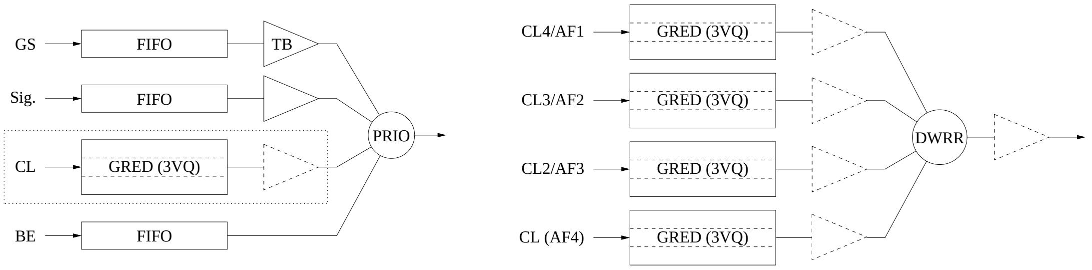
Besides best effort, our model provides two additional service classes: the Guaranteed Service (GS) class that is characterized by hard QoS assurance in terms of both delivery guarantee and maximum delay; and Controlled Load (CL) classes that emulate the behavior of a lightly loaded best effort network. Reservations for traffic flows using the GS class are characterized by a token bucket. Reservations for traffic flows using CL classes are characterized by three average rate watermarks: packets exceeding the first two watermarks will receive a degraded service in terms of drop probability; packets exceeding the third watermark will be dropped. The admission control algorithms for the GS and CL classes are different, as will be shown in sub-section 4.2.

As we have already mentioned, our model is based on DiffServ. The service class is conveyed in the DSCP (DiffServ Code Point) field of the packet header. Packet classification at the routers is performed based on both the DSCP field and the input interface at which the packet arrived. Our guaranteed service class is based on the same principles as the EF (Expedited Forwading) PHB in DiffServ: low delay and very high delivery assurance, provided by a service rate which must be greater than or equal to the maximum arrival rate. Similarly, our controlled load class is based on the AF (Assured Forwarding) PHB, providing a better than best effort service with three different drop precedence levels. Packets from a flow using the controlled load service are marked for one of the drop precedence levels according to rate levels established by the signaling protocol. The recommended DSCP for the guaranteed service class is binary 101100. This value is different from the one used in the EF class, 101110, enabling coexistence of our model and the DiffServ one by protecting controlled load flows subject to admission control, policing and shaping from the EF traffic. The recommended DSCPs for the controlled load service class are binary 100010, 100100 and 100110 which are, in fact, the DSCPs of the DiffServ class AF4[1]. Our model may also contain different controlled load service classes using DSCPs from other AF classes, provided these are not used by DiffServ.

The simplest queuing model for the routers is depicted in figure 2-A. The main scheduler is priority based, with at least four different queues. The highest priority one is used for the highly delay sensitive traffic of the guaranteed service (GS) class. The service discipline inside this class is classical FIFO (First In First Out). With a priority immediately below that of the guaranteed service there is a queue for other critical traffic with lower delay sensitivity, such as signaling and routing protocols (Sig.), internally served in a FIFO fashion as well. Obviously, this class is not subject to admission control. With lower priority, there is a queue for the controlled load (CL) traffic class, which contains three virtual queues (VQ), one for each drop precedence level. The internal service discipline may be

[1]When both our model and the DiffServ model work together, the latter must not use the AF4 PHB.

any one of the GRED (Generalized Random Early Detection) algorithms (e.g. RIO[2]). In the case of multiple, differentiated, controlled load classes, the CL queuing block is replaced by the one shown in figure 2-B. A Deficit Weighted Round Robin (DWRR) discipline[3] is used to dequeue from individual CL / DiffServ AF classes. At the lowest priority there is a queue for best effort traffic with a service discipline that may be FIFO, RED or any other appropriate.



A. Single CL class

B. Multiple CL classes

Figure 2: Queuing model

4.2 Resource Reservation and Control

Admission control is performed at every node along the flow path. As we have already mentioned, the admission control algorithms are different for the guaranteed service and the controlled load classes: while for the GS class parameter based admission control must be performed in order to strictly guarantee that enough bandwidth and buffer space are available at every router even in the worst case, the QoS requirements of the CL class are much looser, allowing the use of measurement based admission control in order to improve the utilization of network resources[4].

Guaranteed service flows are characterized by a token bucket. In order to perform admission control for a GS flow, the router computes the summed token bucket for the new flow and all existing flows sharing the same service class and output interface, using the formulas $r_{sum} = \sum_i r_i$ and $b_{sum} = \sum_i b_i$, where r_i and b_i are, respectively, the token bucket rate and depth of the individual flows. In order for the flow to be accepted, the bandwidth available for the GS service class must be greater or equal to r_{sum} under every circumstance. Similarly, the buffer space available to the GS queue (figure 3) must satisfy the condition $B_{min} \geq b_{sum} + \frac{r_{sum}}{R_o} \times MTU$, where R_o is the output rate of the link. In fact, buffer space must be somewhat larger than B_{min} in order to avoid packet dropping even in presence of some clock drift between routers.

A more effective algorithm (in terms of network resource utilization) for the aggregation of guaranteed service flows characterized by token buckets has been proposed in [16]. However, the cascaded token bucket proposed is more complex to compute than the summed one, and must be recalculated every time a new flow is admitted or terminated. In contrast, the summed token bucket is extremely simple to compute and may be differentially determined by simply adding (or removing, in the case of a flow termination) the flow parameters to the aggregate values. Since scalability is vital in our model, it makes use of the summed formulas.

Controlled load flows are characterized by three average rate watermarks, two for degrading service and a third one for dropping. The three watermarks that characterize the aggregate are obtained by summing the corresponding watermarks from the individual flows. Since some amount of packet loss is tolerated, admission control is not as critical as in the case of GS. This allows us to somewhat improve network resource utilization, either by using Measurement Based Admission Control (MBAC) algorithms or by using Parameter Based Admission Control (PBAC) with some

[2]Random Early Detection (RED) with In/Out bit

[3]Note that any algorithm performing the same function, such as a variant of the Weighted Fair Queuing discipline, can be used.

[4]Though parameter based algorithms may also be used.

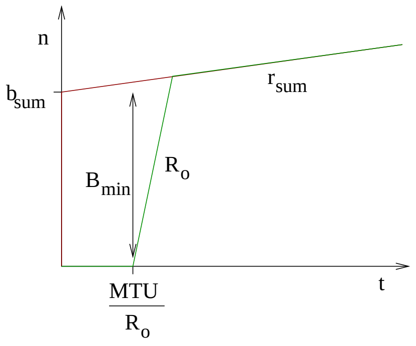


Figure 3: Buffer space for the GS class

overbooking. Maximum values for the summed reservations may be administratively specified in order to keep some remaining bandwidth available for best effort traffic.

4.2.1 Traffic Shaping

As can be seen in figure 2, the highest priority queues, corresponding to the GS traffic class and signaling/routing traffic must be subject to a traffic shaper. The traffic shaper for the GS class is of token bucket type, and is parameterized using the summed values r_{sum} and b_{sum}, which is absolutely needed in order to avoid packet discarding by the policer at the input interface of the following node[5]. Since the traffic is shaped to the summed token bucket for the class, which is an upper bound to the actual traffic in those flows, this shaper does not degrade the quality of service received by GS flows, as is demonstrated in [8].

The signaling/routing traffic, though not subject to admission control, must also be shaped in order to keep some amount of bandwidth available to the controlled load class. Since this traffic will not be subject to policing at the input interface of the following node, the shaper may be simplified to something akin to a work-conserving rate limiter (i.e., rate is only limited when lower priority classes have queued packets waiting for transmission).

The controlled load class may also be shaped, but this is only needed if the network administrator wants to make sure that some amount of bandwidth remains for best effort traffic. Since traffic from the CL class tolerates some amount of packet dropping, the shaper may also be a simple work-conserving rate limiter.

4.2.2 Traffic Policing

There are four different types of nodes in our model: end nodes, access routers (A), edge routers (E) and core routers (C). All of them perform signaling and support the queuing model described in section 4.1. The access nodes perform per-flow policing, ensuring that traffic from each individual flow does not exceed its reservation parameters. Edge nodes perform aggregate policing only, with an aggregate consisting on a DSCP value (or a set of DSCP values) per input interface. There is no need for edge routers to perform per-flow policing, since this has already been done at the access routers. Edge nodes may also perform remarking to a higher drop probability within the CL class. Since the remarking is to be performed at aggregate level, it must be *color aware* (meaning that no packet will be remarked to a lower drop probability) in order to avoid unfair service degradation in "well behaved" flows. Core routers perform no policing, as they assume that all traffic has already been policed in their domain either by an edge router (for traffic coming from different domains) or by an access router (for eventual traffic generated inside their own domain) and is, therefore, conformant to the reservations.

[5]More precisely, at the following edge node, since no policing is performed at the core.

Contrary to traffic shaping (performed at the output interface) which is always performed on traffic class aggregates, policing (at the input interface) is performed in a per-flow basis at the access routers. This is needed in order to ensure that flows are conformant to the reservations, and also to perform drop precedence (re)marking in the case of controlled load flows. Forcing every flow to conform to its reservation ensures that no service degradation is inflicted by a greedy flow on the remaining flows sharing the same service class and output interface.

5 Label Switching

Usually, the most complex tasks in the maintenance of per-flow reservations are not the signaling itself and the maintenance of the flow information. The signaling traffic only represents a small fraction of the data traffic. The maintenance of per-flow information only requires routers with sufficient memory available. For example, consider a core router handling an average of a hundred thousand flows. If each flow's information occupies 100 bytes of memory space, the router requires a total of 10 Mbytes, which is not larger than the memory available nowadays. Therefore, the most complex task for the core routers is the lookup of the flow information stored in the routers, based on the 5-tuple parameters that specify the flow, when the number of flows is very high (in the order of hundreds of thousands).

Processing signaling messages involves accessing a structure in memory where the reservation parameters are held. Usually, this is implemented using hash tables, but this may be quite inefficient in core routers where the number of simultaneous flows may exceed several hundred thousands. In order to efficiently access the reservation structures we developed a label switching mechanism which allows direct access to these structures without any need for hash lookups. These labels are 32 bit values whose meaning is externally opaque, but internally may be an index to a table of reservation structures or the memory address[6] of the reservation structure.

Once a signaling message is received, its LABEL object will be used to directly access the reservation flow state. This procedure significantly reduces the message router processing time and the signaling complexity. As will be shown, only the initial messages sent to perform resource reservation are excluded from this efficient access, but since the reservation does not yet exist, they would not need it anyway. Allocating a new structure is trivial, consisting in taking the first element from a linked list. The labels installation in the flow reservation structure will be detailed in section 6.

The label switching mechanism has also strong advantages in all per-flow processing. For example, the per-flow policing performed at the access routers needs to access the flow state each time it receives a packet. Although the number of flows at the access network is much smaller than the one at the core networks, the use of a mechanism that provides direct access to the flow information significantly reduces its computational complexity. Finally, the route change detection could also make use of the labels. More specifically, the router could compare the next hop value included in the packet header with the one stored in the reservation structure of the flow. A difference between these values indicates that the route has changed.

However, in order to profit from these advantages on per-flow processing, all packets would need to carry the label information.

In IPv4 we could make use of the *Identification* and *Fragment Offset* fields in the packet header, since these fields are only used when fragmentation occurs. Since it is possible to avoid fragmentation (although it was referred in [17] that only 0.13% of the packets are fragmented), these fields can be used without affecting the system functioning. Since the sum of the available bits is only 29 (less than the 32 bits required for the Label object), we may impose that the three last bits of the labels are zero, which implies that all memory structure addresses are aligned to 8 byte (i.e., 64 bit) boundaries. As an alternative, the label could be conveyed by means of an IPv4 header option.

In IPv6 we could only make use of the Flow Label header field to carry the labels. However, the length of this field is just 20 bits, which is not enough to carry the address. In this case, a better option would be conveying the label in an IPv6 extension header.

[6]Most routers will use 32-bit processors. For routers with more than 32 bits of addressing space, the label may be an offset.

Although the introduction of the labels in the packet header increases the scalability of the solution, in IPv6 an extension header becomes necessary, which increases the packet's overhead. The trade-off between these different approaches, introducing the label information only in signaling messages (and do not profit from its advantages in per-flow policing and route change detection) or carrying it in the packet header, will be experimentally investigated.

6 Signaling Protocol

As we have already mentioned, the signaling works on a hop by hop basis, providing unidirectional, soft state, and sender initiated reservations. Instead of creating a whole new protocol from the ground up, we have choosen to implement it as an extension to the RSVP protocol for several reasons: RSVP is a widely known and supported protocol; it also works on a hop by hop basis; it also uses routes established by an underlying routing protocol; and RSVP already implements some of the required functionality for our model. Although our signaling protocol is an extension of RSVP, it is much more scalable, since (1) the access to the flow information is simple with O(1) complexity, (2) it considers soft reservations with expiration timers implemented in a very effective way and with small complexity, and (3) it considers simple reservation identification in order to decrease the length of the refresh and explicit tear down messages. These topics will be described in more detail in the following sub-sections.

6.1 Signaling Messages

Since RSVP is meant to perform receiver initiated reservations, we had to extend it by adding three new message types: SResv (Sender Reservation), SResvStat (Sender Reservation Status) and SResvTear (Sender Reservation Tear Down). The first one is used to establish new sender initiated reservations and refresh existing ones. The second one is used to report status (success or error conditions). The last one is used to explicitly terminate an existing reservation.

SResv messages are usually originated at the sender node, but may also be originated in any other node along the path for local repair purposes in error conditions, particularly in the case of route changes. Full SResv messages include a flow identification, consisting of both a SESSION and a FILTER_SPEC object, an identifier of the service class, a reservation timeout value and a FLOWSPEC class object parameterizing the reservation. The service class is specified by the PHB for that class, encoded as specified in section 2 of [6]. The service class implies a particular type FLOWSPEC class object[7]. Two different FLOWSPEC types have been newly defined for this purpose. GS class reservations are parameterized by a token bucket, whereas CL class reservations are parameterized by three rate thresholds, two for drop precedence degradation and the third one for dropping. Though the rate parameters are specified in IEEE-754 floating point format, the nodes must perform all the calculations for aggregates using fixed point math, in order to avoid the accumulation of floating point rounding errors over time (eventually storing these fixed point values in the memory structure holding the reservation parameters). Optionally, the SResv message may contain an ADSPEC object containing the accumulated service degradation along the path, updated at every node.

```
<SResv Message (initial)> ::= <Common Header> [ <INTEGRITY> ] <MESSAGE_ID>
                              <SESSION> <FILTER_SPEC> [ <LABEL> ]
                              <LABEL_SETUP> <RSVP_HOP> <SRESV_PARMS>
                              <FLOWSPEC> [ <ADSPEC> ] [ <POLICY_DATA> ]
```

The LABEL_SETUP and LABEL objects include a 32-bit value, the label, which is directly mapped to the memory address containing the flow reservation information. These objects will be used, respectively, to install the label at reservation setup time and to access the memory address when further messages are processed. The process of label installation is detailed in sub-section 6.2.

[7] RSVP object C-Type

The SRESV_PARMS object includes a 3-bit value that represents the reservation expiration time. This value is in a 2-logarithmic scale and will be used with the developed algorithm to efficiently implement expiration timers. This algorithm will be detailed in sub-section 6.3.

All SResv messages include a MESSAGE_ID object (as defined in [3]) which allows for an association between the SResv message and an eventual SResvStat message triggered by its reception.

SResv messages used for refreshing reservations without changing parameters (with the possible exception of reservation expiration timeout) are simpler than the full SResv messages used for the initial reservation setup. The SESSION and FILTER_SPEC objects are absent, as is the FLOWSPEC class object. The LABEL_SETUP object, that will be discussed in sub-section 6.2, is also absent from refresh-only SResv messages.

```
<SResv Message (refresh)> ::= <Common Header> [ <INTEGRITY> ] <MESSAGE_ID>
                              <LABEL> <RSVP_HOP> <SRESV_PARMS>
                              [ <POLICY_DATA> ]
```

SResvStat messages are used for both reservation confirmation and error reporting. The first kind is generated at the receiver node, triggered by the reception of a SResv message, and is propagated up to the sender. The second kind may be generated by any other node for reporting an error condition to the previous node which, depending on the error type, may be propagated up to the sender or trigger a local repair procedure. SResvStat messages sent in response to a SResv message include either a MESSAGE_ID_ACK or a MESSAGE_ID_NACK object, as appropriate, corresponding to the MESSAGE_ID of the SResv message. MESSAGE_ID_ACK objects are distinguished from MESSAGE_ID_NACK objects based on the object class type field, whose value is 1 for the former and 2 for the latter. SResvStat messages with a MESSAGE_ID_NACK object may also contain an ERROR_SPEC object further describing the reason for the reservation failure. Non-reactive SResvStat messages, sent in order to report an error condition, always contain an ERROR_SPEC object.

```
<SResvStat> ::= <Common Header> [ <INTEGRITY> ] <LABEL> [ <LABEL_SETUP> ]
                <RSVP_HOP> [ <MESSAGE_ID_ACK> | <MESSAGE_ID_NACK> ]
                [ <ADSPEC> ] [ <ERROR_SPEC> ]
```

SResvTear messages are generated either by the sender node to explicitly terminate a reservation or by any other node along the path which detects a route change in order to remove the stale reservation from the old path. Among other objects, they include the LABEL object, so the nodes can directly access to the flow information in order to remove the reservation.

```
<SResvTear Message> ::= <Common Header> [ <INTEGRITY> ] <LABEL> <RSVP_HOP>
```

If there is no explicit teardown and no refresh messages, the timer associated to the specific flow expires. The flow reservation is then terminated, and the corresponding information is removed from the reservation table.

Basic security in signaling is achieved by means of INTEGRITY objects, as in the standard RSVP protocol.

Since in this model signaling must be performed at every node (including core routers), much care has been taken in reducing the processing load imposed by the signaling protocol by making use of computationally efficient algorithms and techniques only. The next sub-sections will address the details of the signaling protocol that increase its scalability.

6.2 Signaling Dynamics

As we have already mentioned, although the base signaling protocol is RSVP, the proposed protocol is sender initiated. This characteristic introduces a large difference between the proposed protocol and standard RSVP. Being sender initiated means that, upon receiving the first message (SResv), the nodes along the path run the admission control algorithm (based on the token bucket parameters or on network measurements), and perform resource reservation for the new flow. Notice that in this model, performing resource reservation for a new flow is equivalent to increasing the bandwidth allocation for the class to which the new flow belongs, since our model is DiffServ based.

When a router receives an initial SResv message requesting a new reservation and the request is accepted, the SResv message is forwarded to the following router and the updated traffic specification for the class is committed to the admission control module. This traffic specification, though, is not committed to the policing and queuing modules until the corresponding SResvStat, confirming a successful reservation up to the receiver, arrives. This ensures that whenever a router updates its resource allocation for a class, all routers towards the receiver terminal already have performed their updates. Otherwise, packet dropping would be inflicted on the class by the policer module of the following node in the time lapse between resource allocation updates of the two routers.

The SResvStat message will not only inform the sender about the success or failure of the admission request[8], but also to trigger the commitment of the resource reservation to both the policing and the queuing modules at the routers if the reservation succeeded, or remove it from the admission control module if it failed.

Each reservation structure stores, among other parameters, the flow specification and three label fields: B, T and F. These label fields are, respectively, the label to be used in signaling messages (or data packets) sent backwards (towards the sender), the label for the router itself[9], and the label to be used in messages sent forwards (towards the receiver). The obvious exceptions are the end nodes: the sender has no B label and the receiver has no F label.

The advantage of the utilization of these 3 labels can be explained with the following simple example. Upon receiving a refresh SResv message, a node checks the label field in the message, directly accesses the state information of the flow, updates its expiration timer, and copies the F label field stored in the reservation structure to the label field of the SResv message to be forwarded. This label will be used by the next node to directly access the reservation state of the flow.

The labels are installed at reservation setup time, as shown in figure 4, using LABEL_SETUP objects in the SResv and SResvStat messages.

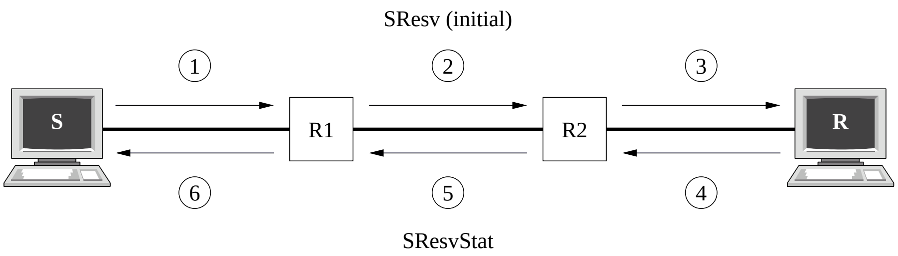


Figure 4: Label installation

The proposed signaling protocol works as follows (figure 5). The initial SResv message, originated at the sender (1) contains, among other objects, the flow reservation specification and the label T_S. This label, conveyed by the LABEL_SETUP object, will be used in messages sent by the R1 router to the sender. R1 performs admission control, based on the flow reservation parameters. If the flow can be accepted, the router updates the resource reservation of the flow's class[10], creates an entry in the reservation structure for the flow, stores the label at the B field for this reservation, and forwards the SResv message to R2 (2) after changing the LABEL_SETUP to T_1. If the flow cannot be accepted, a SResvStat message with a MESSAGE_ID_NACK object is sent towards the sender. Each router along the path that receives this message removes resource reservation information for this flow and releases the reserved bandwidth. If the flow was accepted and the router forwarded the SResv message to the next hop, the procedure is repeated at every router along the forward path until

[8]This message will also be used to install the labels for use in the signalling messages towards the receiver.
[9]The T label may be implicit.
[10]As has been previously stated, this reservation is only committed to the admission control module.

the SResv message finally arrives at the receiver (3) with T_2 in LABEL_SETUP. At this point, all routers along the path have reserved resources for the new flow and all labels required for backward message processing are installed in the reservation state.

Now, the receiver will acknowledge the successful reservation by sending a SResvStat message with a MESSAGE_ID_ACK object towards the sender. Since the labels required for backward message processing are already installed, the SResvStat message will make use of the labels. Note that even a SResvStat message reporting a failure reservation originated by a node in the middle of the path can make use of the labels already installed in that path. The SResvStat message reporting successful reservation (4) contains a LABEL object with the label T_2 and a LABEL_SETUP object with the label T_R. R2 uses the T_2 label in the LABEL object to access the memory structure for this reservation and commits the updates to the policer and queuing modules. The T_R label is stored at the F field in R2. It switches the LABEL to the one installed at the B field, T_1, and forwards the message to R1, containing T_2 in LABEL_SETUP (5). When the SResvStat message reaches the sender (6) with T_1 in LABEL_SETUP, all labels are installed, the reservation is acknowledged, and no further LABEL_SETUP and flow reservation objects need to be sent, except in the case of route changes. Notice that, the path of the SResvStat message is the symmetric path of the SResv message. Since the signaling protocol is hop by hop, the SResvStat message has information on the previous hop of the SResv message. However, these two messages are used in a one way reservation. Therefore, the reservation in the opposite direction can have a completely different path.

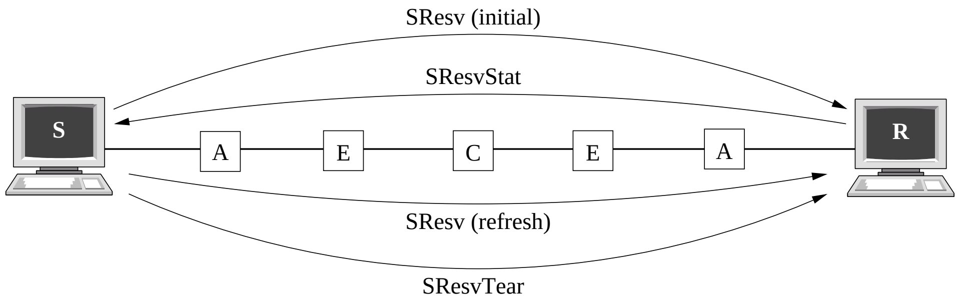


Figure 5: Signaling messages

Each established reservation has an associated timer, which is proportional to the expected flow duration. The algorithm devised to provide efficient timers' implementation will be detailed in the next sub-section. A reservation is explicit released upon the reception of a SResvTear message or implicit released (soft reservation) upon the expiration of the timer for that reservation. In order to refresh the reservation and update the expiration timer (so the reservation is not released), the sender originates a simplified (refresh) SResv message (figure 5) towards the receiver. This message will make use of the labels to provide direct access to the flow reservation information, and each router along the path updates the expiration timer. Refresh only SResv messages do not require a confirmation from the receiver. Notice that, if the label could be inserted in the packet header, the reservation refresh could be performed with data packets, precluding the use of signaling messages.

When a sender wants to explicitly terminate a flow session, it sends a SResvTear message towards the receiver, also making use of the labels. Upon reception of this message, each router along the path removes the reservation information for that flow and releases its bandwidth for future flows.

The SResv messages are also required when a route changes (figure 6). In this case, the router that detects a route change towards the receiver, originates a full SResv message along the new path in order to reserve resources and install the labels required for scalable signaling processing. Each router along the new path processes this message and forwards it to the next hop. The receiver, upon receiving the message, sends back a SResvStat message acknowledging the new reservation. The expiration time for this SResv message must be greater or equal to the remaining time for reservation expiration at the router detecting the route change. A SResvTear message should also be sent, when possible, to the old next hop in order to remove the stale reservation from the old path.

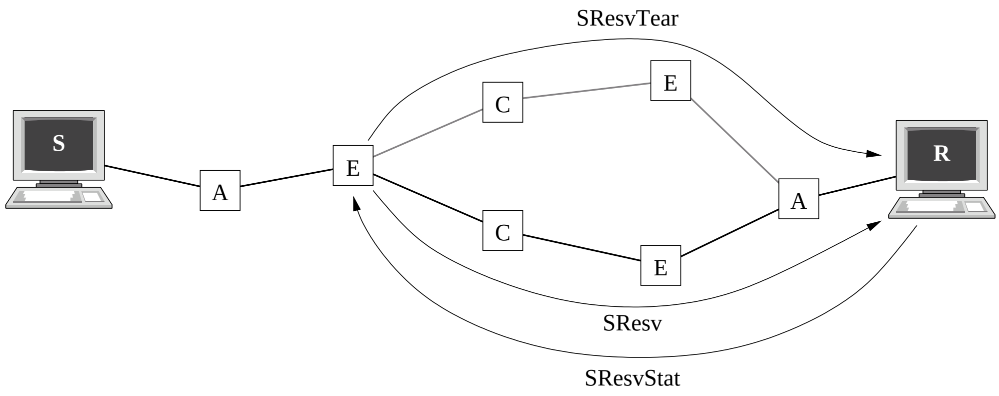


Figure 6: Route change

6.3 Soft Reservations and Efficient Timer Implementation

Soft state reservations have the obvious advantage of providing adaptability to changing network conditions, but they imply the implementation of expiration timers. The basic implementation concept for timers is a sorted event queue: the processor waits until the first timer value in the list expires, dequeues it, performs the appropriate processing, then goes on waiting for the next timer value to expire. While dequeuing an event is trivial, inserting an event with a random expiration time is a very expensive operation, highly dependent on the total number of events queued. Some algorithms have been proposed [18, 1] for rather efficient implementation of timers, but while general enough for implementing any kind of timer, these algorithms are still too complex to be used for reservation expiration timers at the core routers, where the number of reservations (and, thus, the number of expiration timers) is possibly huge.

Contrasting to the complexity of generic timers, fixed delay timers are very simple and efficient to implement: in this case, the event queue is treated in FIFO fashion, providing both trivial event queuing and dequeuing. Fixed delay termination timers, however, are undesirable in the case of reservations which may have very dissimilar life spans.

Trying to achieve some sort of balance between the simplicity of fixed delay timers and the flexibility of generic timers, we have created an algorithm which has trivial timer queuing and a low and constant cost timer dequeuing, providing eight possible timer delays in a base 2 logarithmic scale. These values map to a timer delay range of 1:128, which ought to be enough. The implementation is based on eight different queues, each of which has an associated fixed delay. Internally, therefore, these queues are served using a FIFO discipline. Flow timeouts are chosen from one of the eight possible values using a three bit field in a SRESV_PARMS object included in SResv messages. Queuing an event is a simple matter of adding it to the tail of the corresponding queue, which is trivial. Dequeuing an event means choosing one of the eight possible queues (the one whose timers expires first) and taking the first event from that queue.

Anyway, timers should not expire very frequently, as most of the time reservations will either be refreshed or explicitly terminated by means of a SResvTear message.

7 Scalability Considerations

Although this model uses per-flow signaling, the processing load of the signaling messages and the computational complexity at the network nodes are highly reduced compared to the IntServ architecture.

First, the packet classification and the scheduling mechanisms are performed on a per-aggregate (class) basis instead of a per-flow basis. This makes packet classification and scheduling complexity independent from the number of flows; it depends only on the number of classes, which is fixed and low.

Second, the admission control decision is based on the summed token bucket instead of considering a cascaded token bucket, reducing the computational complexity of the aggregate parameters recalculation whenever a flow is admitted or rejected. The effect of considering the easier approach is a very small reduction in the utilization of the GS class, which has no expected impact in the system performance.

Third, an efficient algorithm was developed to implement the expiration timers. We have chosen to trade off the flexibility of generic timers, which we do not need, for the efficiency of a timer implementation using a small number of discrete values, therefore achieving a low and constant processing cost. The logarithmic scale provides a broad range of expiration timer values.

Finally, the developed label switching mechanism decreases the signaling messages processing in the routers, significantly increasing its scalability. Without the label switching mechanism, the processing time for merely accessing the reservation information is dependent on the number of flows[11], which can be larger than several hundred thousands at the core routers. By contrast, our model allows direct access to the memory structures by means of the labels.

The heaviest task in our model is the per-flow policing at the access routers. Although this is not a huge problem, since the number of flows at the access routers is usually small, its complexity may also be reduced to O(1) complexity if the labels are introduced in the data packet headers.

In light of these arguments, we may state that our model achieves end-to-end QoS guarantees with per-flow signaling without the scalabilty concerns of IntServ.

8 Conclusions and Future Work

In this paper a new QoS architecture was proposed which scalably supports end-to-end QoS with strict and soft guarantees. The proposed architecture includes per-flow signaling with enhanced scalability and resource reservation for aggregates of flows at both core and access networks, with an underlying DiffServ architecture. In order to improve the signaling scalability, several algorithms and techniques were proposed that reduce the signaling messages' processing in core and edge nodes. It was proposed a label switching mechanism that allows direct access to the resource reservation structure, and it was developed an algorithm to efficiently implement expiration timers in soft reservations. Moreover, all mechanisms related to packet classification and scheduling were performed on a per-aggregate basis, making its complexity independent from the number of flows, and the admission control decisions were based on aggregate parameters trivially computed.

Currently, our model is being implemented in the ns-2 network simulator. Several simulations will be performed, not only to further validate the model, but especially to compare its performance to that of other models which provide similar services while also aiming at high scalability. We are particularly interested in models providing reservation aggregation over a DiffServ infrastructure. The expected outcome from these tests will be an improved network resource utilization, while assuring the same quality of service. The drawback is increased signaling traffic, which we expect will be compensated by our efficient signaling processing (and may be confirmed later by a prototype implementation).

In a business-oriented Internet, no QoS model is expected to be widely deployed unless there is a way for the service providers to account and charge for an increased service quality. It is, therefore, desirable to integrate these with our model, which must be done in an efficient way. One possible solution is to have the ingress routers talk to an AAAC server in their domain in order to obtain the applicable policy for the requested reservation. This policy information would be appended to the reservation request (SResv) messages propagated inside the domain, and would be conveyed by means of a POLICY_DATA object. Routers would then perform admission control decisions based not only in available network resources, but also on the acceptance policy. The POLICY_DATA object would be removed from the messages when leaving the domain.

Another topic for further research is security, which is an ever more important issue in networking. From a security point of view, we will look not only at the signaling protocol and possible ways in

[11] Perfect hashing for the 5-tuple would also have O(1) complexity, but is absolutely unfeasible. Other alternatives would have a complexity dependent on the number of flows.

which it could be taken advantage of by malicious users, but also at possible ways in which service theft could be done, using resources reserved by other parties. Compatibility with privacy models is also a topic for further work.

Other topics for further research are the integration with mobility and wireless scenarios, the possibility of interaction with QoS routing protocols and multicast.

References

[1] Mohit Aron and Peter Druschel. Soft timers: Efficient microsecond software timer support for network processing. *ACM Transactions on Computer Systems*, 18(3):197–228, August 2000.

[2] F. Baker, C. Iturralde, F. Le Faucheur, and B. Davie. Aggregation of RSVP for ipv4 and ipv6 reservations. RFC 3175, Internet Engineering Task Force, September 2001.

[3] L. Berger, D. Gan, G. Swallow, P. Pan, F. Tommasi, and S. Molendini. RSVP refresh overhead reduction extensions. RFC 2961, Internet Engineering Task Force, April 2001.

[4] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine. A framework for integrated services operation over diffserv networks. RFC 2998, Internet Engineering Task Force, November 2000.

[5] G. Bianchi, A. Capone, and C. Petrioli. Throughput analysis of end-to-end measurement-based admission control in ip. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

[6] D. Black, S. Brim, B. Carpenter, and F. Le Faucheur. Per hop behavior identification codes. RFC 3140, Internet Engineering Task Force, June 2001.

[7] S. Blake, D. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *Internet Engineering Task ForceInterent*, RFC(2475), December 1998.

[8] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus*. Springer-Verlag, 2001.

[9] R. Braden, D. Clarck, and S. Shenker. Integrated services in the internet architecture: an overview. *Internet Engineering Task ForceInterent*, RFC(1633), June 1994.

[10] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) - version 1 functional specification. *Internet Engineering Task ForceInterent*, RFC(2205), September 1997.

[11] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang. Endpoint admission control: Architectural issues and performance. In *Proceedings of ACM SIGCOMM 2000*, August 2000.

[12] C. Cetinkaya and E. Knightly. Egress admission control. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

[13] V. Elek, G. Karlsson, and R. Ronngren. Admission control based on end-to-end measurements. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

[14] S. Sargento and R. Valadas. Performance of hierarchical aggregation in differentiated services networks. In *Proceedings of 10th International Conference on Telecommunication Systems, Modeling and Analysis*, October 2002.

[15] S. Sargento, R. Valadas, and E. Knightly. Resource stealing in endpoint controlled multi-class networks. In *Proceedings of IWDC 2001*, September 2001. Invited paper.

[16] J. Schmitt, M. Karsten, L. Wolf, and R. Steinmetz. Aggregation of guaranteed service flows. In *Seventh International Workshop on Quality of Service*, pages 147–155, May 1999.

[17] Ion Stoica. *Stateless Core: A Scalable Approach for Quality of Service in the Internet*. PhD thesis, Carnegie Mellon University, December 2000.

[18] George Varghese and Anthony Lauck. Hashed and hierarchical timing wheels: efficient data structures for implementing a timer facility. *IEEE/ACM Transactions on Networking*, 5(6):824–834, December 1997.

[19] Zhi-Li Zhang, Zhenhai Duan, Lixin Gao, and Yiwei Thomas Hou. Decoupling qos control from core routers: a novel bandwidth broker architecture for scalable support of guaranteed services. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 71–83. ACM Press, 2000.